TEXAS INSTRUMENTS

Improving Man's Effectiveness Through Electronics

Model 990 Computer
Terminal Executive Development System (TXDS)

Programmer's Guide

MANUAL NO. 946258-9701
ORIGINAL ISSUE 1 APRIL 1977
INCLUDES
CHANGE 1 . . . . . 1 JULY 1977
CHANGE 2 . . . . . 15 OCTOBER 1977
CHANGE 3 . . . . . 15 DECEMBER 1977

Digital Systems Division

LIST OF EFFECTIVE PAGES

INSERT LATEST CHANGED PAGES DESTROY SUPERSEDED PAGES

Note: The portion of the text affected by the changes is indicated by a vertical bar in the outer margins of the page.

Model 990 Computer Terminal Development System (TXDS)
Programmer's Guide (946258-9701)

Original Issue . . . . . 15 April 1977
Change 1 . . . . . 1 July 1977 (ECN 419567)
Change 2 . . . . . 15 October 1977 (ECN 419599)
Change 3 . . . . . 15 December 1977 (ECN 415098)

Total number of pages in this publication is 220 consisting of the following:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
Cover	3	4-19 - 4-22	0	9-25 - 9-34	0
Effective Pages	3	4-23	2	9-35 - 9-36B	1
iii - ix	3	4-24	0	9-37 - 9-46	0
x	1	5-1 - 5-6	3	10-1	2
xi - xii	3	6-1 - 6-3	0	10-2 - 10-3	1
1-1 - 1-2	3	6-4	3	10-4 - 10-6	3
1-3	0	7-1 - 7-2	3	10-7	2
1-4	3	7-4	1	10-8	3
2-1 - 2-12	3	7-4	1	10-9 - 10-10	2
3-1 - 3-3	3	7-5 - 7-6	0	10-11	1
3-4	0	7-7	3	10-12 - 10-13	2
3-5	3	7-8	1	10-14	1
3-6	0	8-1	3	10-15	2
4-1 - 4-2	3	8-2	0	10-16 - 10-18	1
4-3 - 4-4	1	8-3 - 8-4	3	10-19	2
4-5 - 4-6	0	9-1	2	10-20	1
4-7	1	9-2 - 9-2B	3	10-21	1
4-8	2	9-3	0	10-22	2
4-9 - 4-10	1	9-4	2	10-23 - 10-26	1
4-11 - 4-12	2	9-5	0	11-1 - 11-2	1
4-13	0	9-6	2	11-3	3
4-14	2	9-8 - 9-8B	1	11-4	1
4-15	0	9-9	2	11-5 - 11-6B	3
4-16	3	9-10 - 9-23	0	11-7	3
4-17 - 4-18	1	9-24	2	11-8	1

LIST OF EFFECTIVE PAGES

INSERT LATEST CHANGED PAGES DESTROY SUPERSEDED PAGES

Note: The portion of the text affected by the changes is indicated by a vertical bar in the outer margins of the page.

Model 990 Computer Terminal Development System (TXDS)
Programmer's Guide (946258-9701) (Continued)

Total number of pages in this publication is consisting of the following:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
11-9	3	Appendix C Div	0		
11-10 - 11-14	1	C-1 - C-2	0		
12-1	3	Appendix D Div	0		
12-2	0	D-1	3		
12-3	3	D-2	0		
12-6	0	Alphabetical Index Div	0		
13-1 - 13-2	3	Index-1 - Index-6	3		
Appendix A Div	0	User's Response	3		
A-1 - A-4	0	Business Reply	0		
Appendix B Div	0	Cover Blank	0		
B-1 - B-2	2	Cover	0		

PREFACE

This manual enables the user to employ the Terminal Executive Development System (TXDS) in conjunction with the TX990 Operating System and the Model 990/4 and 990/10 Computer System hardware configuration to develop, improve, change, or maintain (1) the user's customized Operating System and the user's applications programs or (2) any other type of user-produced programs (e.g., the user's own supervisor call processors or the user's own utility programs). It is assumed the reader is familiar with the Model 990 Computer System assembly language and the concepts of the TX990 Operating System.

The sections and appendixes of this manual are organized as follows:

I Introduction – Provides a general description of the TXDS utility programs and their capabilities. Also includes a description of the control functions of the TXDS Control Program.

II Loading and Executing a Program – Provides a step-by-step procedure for loading and executing (1) each of the TXDS and TX990 Operating System utility programs and (2) a user program. Also describes the TXDS Control Program and how to correctly respond to its prompts.

III Verification of Operation – Provides several short step-by-step procedures to checkout proper operation of the TXDS software.

IV Creating and Editing Program Source Code – Describes the capabilities of the TXEDIT utility program and how the user can employ those capabilities to edit or generate the text of source programs and object programs.

V Assembling Source Programs – Describes how the user can employ the TXMIRA utility program to assemble source files (i.e., source code programs).

VI TX990 Cross Reference (TXXREF) Utility Program – Describes how the user can employ the TXXREF utility program to produce a listing of each user-defined symbol in a 990 assembly source program along with the line numbers on which the symbol is defined and all of the line numbers on which the symbol is referenced.

VII Linking Object Modules – Describes how the user can employ the TXDS Linker utility program to form a single object module from a set of independently assembled object modules (in the form of object code or compressed object code).

VIII TXDS Copy Concatenate (TXCCAT) Utility Program – Describes how the user can employ the TXCCAT utility program to copy one to three files to a single output file.

IX TXDS Standalone Debug Monitor (TXDBUG) Utility Program – Describes how the user can employ the TXDBUG utility program to debug programs which have been designed to operate in a "standalone" situation without support of an operating system.

X TXDS PROM (TXPROM) Programmer Utility Program – Describes how the user can employ the TXPROM programming utility program to control the Programming Module (PROM) hardware to make customized ROMs containing user-created data or programs.

XI TXDS BNPF/High Low (BNPFHL) Dump Utility Program – Describes how the user can employ the BNPFHL utility program to produce a BNPF or high/low file format.

XII TXDS IBM Diskette Conversion Utility (IBMUTL) Program – Describes how the user can employ the IBMUTL utility program to transfer standard IBM-formatted diskette datasets to TX990 Operating System files and to transfer TX990 Operating System files to standard IBM-formatted diskette datasets.

XIII TXDS Assign and Release LUNO Utility Program – Describes how the operator can assign and release LUNOs in systems which do not include OCP.

A Glossary – Clarifies selected words used in this TX990 Operating System Programmer's Guide.

B Compressed Object Code Format – Describes the compressed object code format.

C Task State Codes – Lists and describes the task state codes.

D I/O Error Codes – List and describes the I/O error codes available to the user, when coding a program, for printout or display on a terminal device.

The following documents contain additional information related to the TX990 Operating System and are referenced herein this manual:

Title	Part Number
Model 990 Computer TX990 Operating System Programmer's Guide	946259-9701
Model 990 Computer TMS9900 Microprocessor Assembly Language Programmer's Guide	943441-9701
Model 990 Computer Model FD800 Floppy Disc System Installation and Operation	945253-9701
Model 990 Computer Model 913 CRT Display Terminal Installation and Operation	943457-9701
Model 990 Computer Model 911 Video Display Terminal Installation and Operation	943423-9701
Model 990 Computer Model 733 ASR/KSR Data Terminal Installation and Operation	945259-9701
Model 990 Computer Model 804 Card Reader Installation and Operation	945262-9701
Model 990 Computer Models 306 and 588 Line Printers Installation and Operation	945261-9701
Model 990 Computer PROM Programming Module Installation and Operation	945258-9701
990 Computer Family Systems Handbook	945250-9701
Model 990 Computer Communications System Installation and Operation	945409-9701

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

APPENDIXES

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I

INTRODUCTION

1.1 GENERAL

The Terminal Executive Development System (TXDS) provides an extensive software capability to assist in developing, improving, changing, or maintaining (1) the user's customized Operating System and the user's applications programs or (2) any other type of user-produced programs (e.g., the user's own supervisor call processors or the user's own utility programs). Essentially, TXDS delivers this capability by means of the following nine utility programs:

- TXDS Text Editor (TXEDIT) Utility Program
- TXDS Assembler (TXMIRA) Utility Program
- TXDS Cross Reference (TXXREF) Utility Program
- TXDS Linker (TXLINK) Utility Program
- TXDS Copy Concatenate (TXCCAT) Utility Program
- TXDS Standalone Debug Monitor (TXDBUG) Utility Program
- TXDS PROM (TXPROM) Programmer Utility Program
- TXDS BNPF/High Low (BNPFHL) Dump Utility Program
- TXDS IBM Diskette Conversion Utility (IBMUTL) Program
- TXDS LUNO (TXLUNO) Utility Program

Another important feature of TXDS is its capability to function as a control center by means of the TXDS Control Program. The TXDS Control Program simplifies operator interaction with the computer by (1) informing the operator, for example, when a program has been successfully loaded or executed or (2) by requesting the operator for an entry of data/information into the computer via the keyboard of the system console (i.e., the 911 or 913 Video Display Terminal, the 733 ASR/KSR Data Terminal, or the 743 KSR Data Terminal). Basically, the TXDS Control Program functions to prompt (i.e., request) the user for the name of the utility program to load, and the input, output, and options parameters required by the utility program. After the parameters have been entered via the system console keyboard, by the user, in response to the prompts, the specified utility program is loaded into memory and executed. When the utility program has completed execution, the TXDS Control Program again prompts the operator for the name of another program to load, and for the input, output, and options parameters required by the program.

TXDS can also be used to extend and upgrade the capabilities of the TX990 Operating System. By making appropriate use of the TXDS utility programs, users are able to more easily develop, improve, change, or maintain their software. TXDS is an ideal supplement to the TX990 Operating System software package.

The TXDS utility programs are briefly described in the following paragraphs; detailed descriptions (including step-by-step loading procedures, descriptions of available commands, and coding examples explaining typical employment of each utility program) are provided in the other sections in this manual. Figure 1-1 presents the data flow and control paths among the elements of the Terminal Executive Development System software; figure 1-2 presents a typical hardware configuration supporting TXDS capabilities.

1.2 TXDS TEXT EDITOR (TXEDIT) UTILITY PROGRAM

TXEDIT operates interactively with the operator's system console and provides a method of modifying existing source code on diskette files or cassettes and of creating new source files. Its features include the ability to make multiple single directional editing passes on the source file to add, remove, move, or change lines of source.

1.3 TXDS ASSEMBLER (TXMIRA) UTILITY PROGRAM

TXMIRA is a two-pass assembler that produces object code for any member of the Model 990 Computer family, including the TMS9900 Microprocessor. The assembler accepts an assembly language source program and produces a source listing and an object file. For more detailed information, refer to the *Model 990 Computer TMS9900 Microprocessor Assembly Language Programmer's Guide.*

1.4 TXDS CROSS REFERENCE (TXXREF) UTILITY PROGRAM

TXXREF produces a listing of each user-defined label in a 990 assembly source program along with the line number on which each label is defined and all of the numbers of the lines from which the label was referenced. The program may be invoked by either user directive, via the TXDS Control Program, or by chaining to it from the assembler.

1.5 TXDS LINKER (TXLINK) UTILITY PROGRAM

TXLINK links object modules produced by the assembler to form a single object module. The linker allows the specification of up to three input files each of which may contain multiple object modules. TXLINK can also perform partial links which may later be linked with additional modules to complete the linking process.

1.6 TXDS COPY CONCATENATE (TXCCAT) UTILITY PROGRAM

TXXCAT facilitates the transfer of data from file or device to file or device and allows for the specification of up to three source or object files to be copied to one output file or device.

1.7 TXDS STANDALONE DEBUG MONITOR (TXDBUG) UTILITY PROGRAM

TXDBUG is a memory-resident, standalone, system executive that provides extensive program debug features and responds interactively to user input from a 733 ASR Data Terminal.

1.8 TXDS PROM (TXPROM) PROGRAMMER UTILITY PROGRAM

TXPROM provides flexible user control of the PROM programming process as well as standardized programming options.

1.9 TXDS BNPF/HIGH LOW (BNPFHL) DUMP UTILITY PROGRAM

This utility allows a user to produce a BNPF-formatted file, output the file to an appropriate media (paper tape, cassette, etc.) and to compare the media contents to the BNPF-formatted file. It also allows a user to produce a TI 256 by 4 high/low-formatted file, output the file to an appropriate media, and compare the media contents to the input file contents.

1.10 TXDS IBM DISKETTE CONVERSION UTILITY (IBMUTL) PROGRAM

This utility provides a means of transferring standard IBM-formatted diskette data sets to TX990 files and transferring TX990 files to standard IBM-formatted diskette data sets.

1.11 TXDS LUNO (TXLUNO) PROGRAM

TXLUNO allows the user to assign and release LUNOs without using OCP commands.

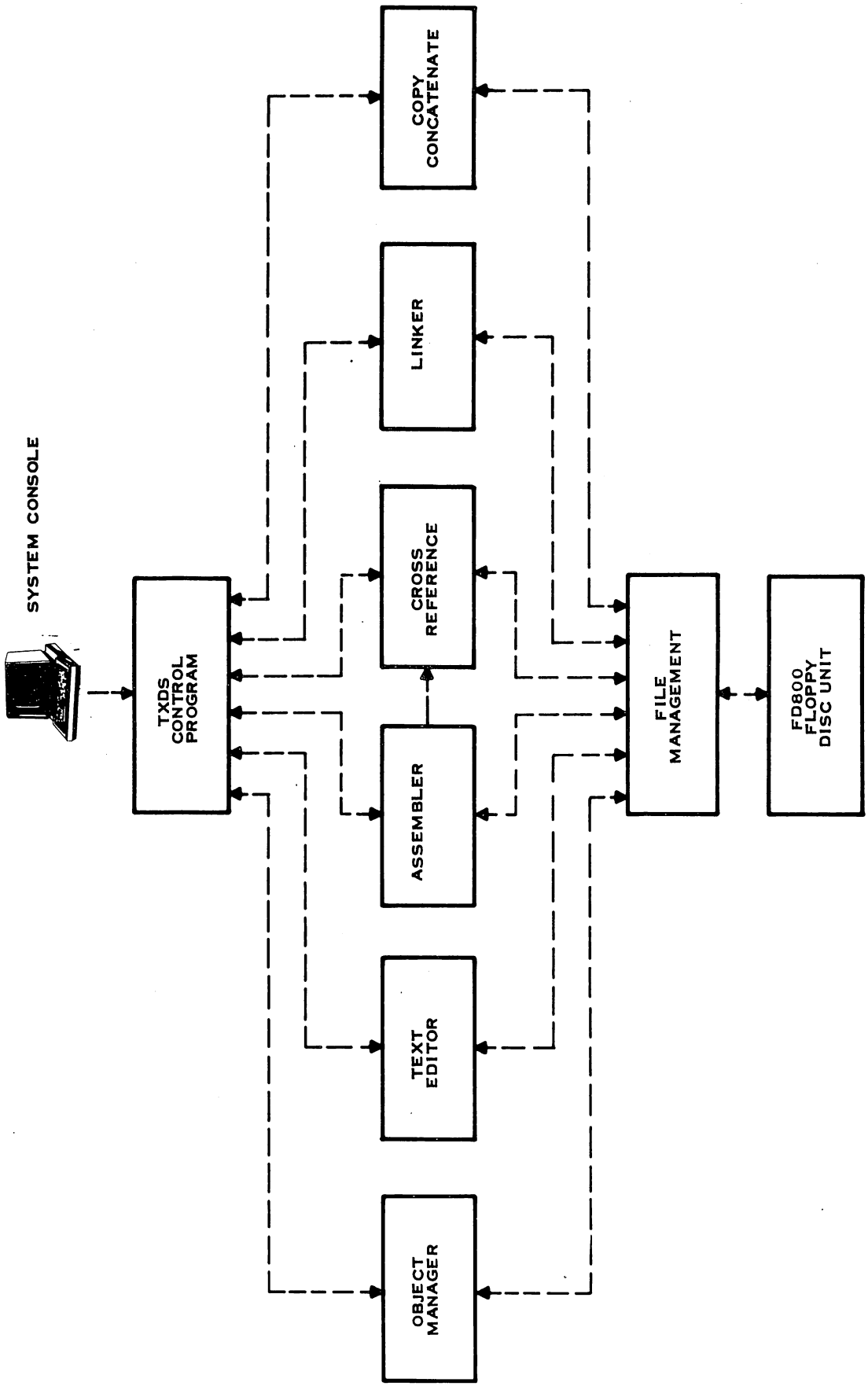


Figure 1-1. Terminal Executive Software Development System, Data Flow and Control Paths

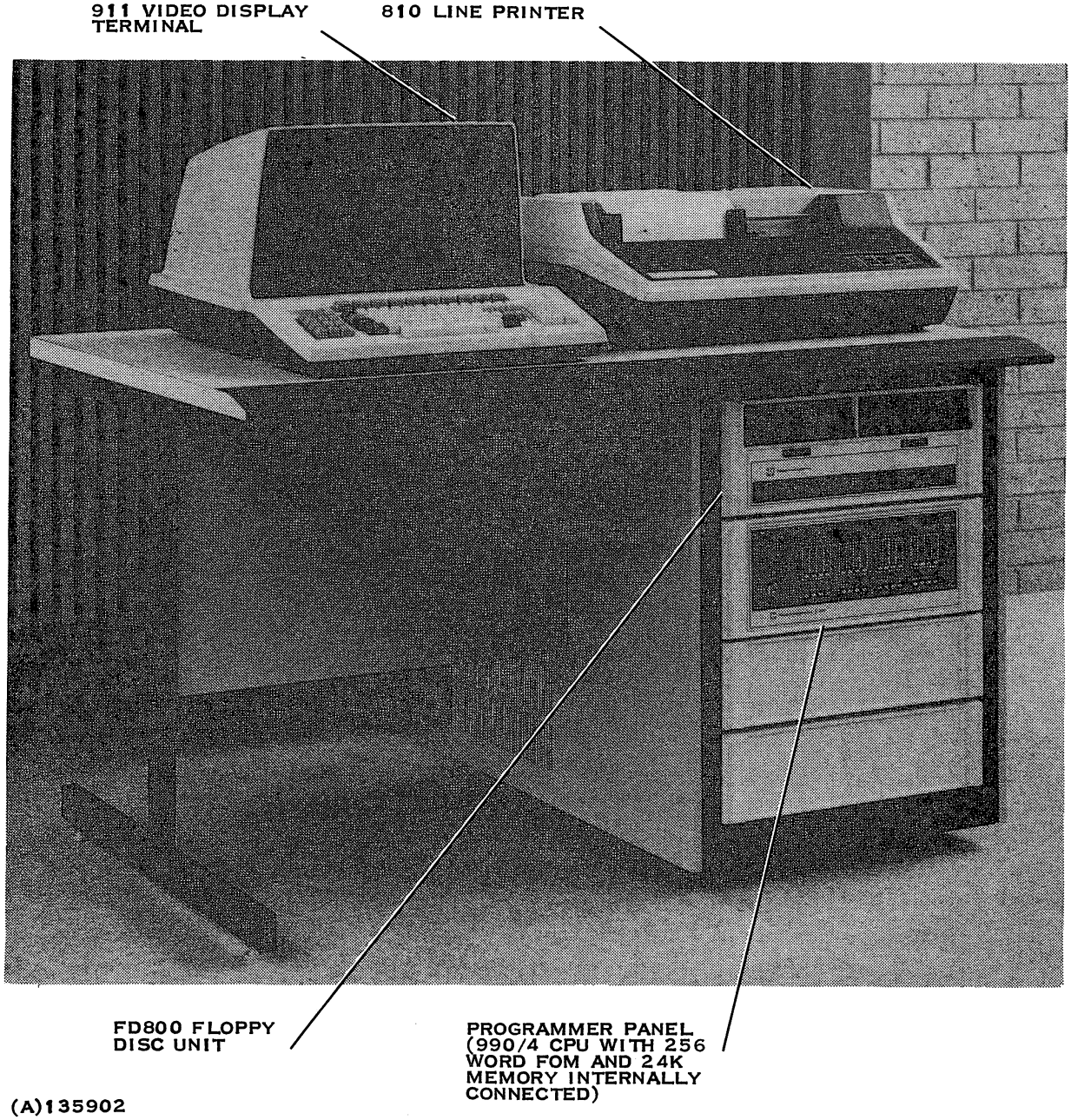


Figure 1-2. Model FS990/4 Floppy Based Software Development System. Minimum Hardware Configuration for TXDS

SECTION II

LOADING AND EXECUTING A PROGRAM

2.1 INTRODUCTION

This section provides the user with a simple procedure for executing: (1) each of the TXDS and TX990 Operating System utility programs; and (2) a user program. The TXDS and TX990 Operating System utility programs are listed as follows:

TXDS Utility Programs	TX990 Operating System Utility Programs
Text Editor (TXEDIT)	System Generation (GENTX)
Assembler (TXMIRA)	Object Manager (OBJMGR)
Cross Reference (TXXREF)	Diskette Backup (BACKUP)
Linker (TXLINK)	Diskette OCP System Utility (SYSUTL)
Copy Concatenate (TXXCAT)	List 80-80 (LIST80)
Standalone Debug (TXDBUG)	Diskette Dump (DSKDMP)
PROM Programmer (TXPROM)	
BNPF/HIGH LOW Dump (BNPFHL)	
IBM Diskette Conversion (IBMUTL)	
LUNO Assignment (TXLUNO)	

The program loading and executing procedure is greatly simplified by the interactive, memory-resident TXDS Control Program, which enables loading and executing of any one of the above utility programs or a user program. The TXDS Control Program (only one of which is included with each Terminal Executive Development System) assists in program loading and execution by printing out or displaying prompts (i.e., requests) on the system console, sequentially, as follows:

```
PROGRAM:
  INPUT:
 OUTPUT:
OPTIONS:
```

The TXDS Control Program also prints out or displays information which indicates to the operator that a program has been successfully loaded or is in the process of being executed. For example, after the TXDS Control Program is executed, the following printout or display is presented at the system console:

```
TXDS    936215   **   152/77   1:05

PROGRAM:
```

The above display tells the operator that the TXDS Control Program is in execution and that the operator may respond to the PROGRAM: prompt by specifying the program to be loaded. The display heading indicates the name of the monitor (TXDS), the part number of the software, the revision status (** = no revision, *A = 1st revision, *B = 2nd revision, etc.), and the date and time of day that the program was loaded (152/77 = 152nd day of 1977).

The following paragraphs in this section present a procedure for loading and executing a program with supplementary supporting information describing: (1) how to correctly respond to the prompts; (2) how to use correct syntax; (3) how to use the special keyboard control keys; and (4) how to code the COMMON memory block. Also included in this section is a procedure for backing up a TI-supplied TXDS diskette and a description of the TXDS Control Program error messages.

2.2 LOADING AND EXECUTING A PROGRAM

Proceed as follows:

1. Load the Operating System (which has been customized to the user's software/hardware configuration) by performing the steps in Section II, entitled "Loading The Operating System", of the *TX990 Operating System Programmer's Guide.*

2. Press the exclamation point (!) key on the system console keyboard.

3. If OCP is included in the system, it responds with a period (.) prompt:

   ```
   !
   .
   ```

   If OCP is not included, proceed to step 5.

4. Execute the TXDS Control Program by responding to the period (.) prompt as follows:

   ```
   !
   .EX,16.TE.
   ```

5. Observe the following printout or display presented on the system console:

   ```
   TXDS   **   010/77   2:05

   PROGRAM:
   ```

   NOTE

   To correctly respond to the PROGRAM:, INPUT:, OUTPUT:, and OPTIONS: prompts, the operator is required to understand the information presented under paragraph 2.3.

6. Respond to the PROGRAM: prompt in accordance with the parameters defined in paragraph 2.3 below by entering the device-name identifier of the input device on which the program to be loaded and executed is stored and/or the file-name identifier of the program to be loaded and executed.

7. After responding to the PROGRAM: prompt, the user can enter a carriage return and respond to the INPUT: prompt; then enter another carriage return and respond to the OUTPUT: prompt; and then enter another carriage return and respond to the OPTIONS: prompt. The user has an alternative and shortened procedure, using the asterisk (*) as described in the paragraph entitled "Special Keyboard Control Keys".

NOTE

1. If a syntax error was made, the prompt for the parameter line in error will be displayed and the operator must reenter that parameter and all of the parameters for the prompt line following the one in error.

2. If a utility program bid by the operator was illegal, the printout or display readout presented in the paragraph entitled "TXDS Control Program Error Messages" will be displayed.

8. After responding to the OPTIONS: prompt, the operator depresses the carriage return key and causes the program to be loaded into memory and then executed. When the program is loaded into memory, a title identifying the utility will be displayed. Observe the following printout/display from the system console if, for example, the TXLINK utility program was loaded:

   TXLINK 937537 **

   (where 937537 is the part number of the TXLINK utility program)

9. After the loaded program has completed execution, observe the following printout or display readout from the system console:

   TXDS 936215 ** 359/77 1:05

   PROGRAM:

   NOTE

   When the user desires to execute a task that already resides in memory without loading the task, a hexadecimal sign is entered, followed by the task ID (10). For example, after the TXEDIT utility program has been loaded into memory, it can be reexecuted as follows:

   TXDS 936215 ** 010/77 2:05

   PROGRAM: >10
   INPUT: DSC:TASK2/SRC
   OUTPUT: DSC:SCRATCH/SRC
   OPTIONS: (carriage return)

A description of the prompts and associated response-entries is provided in the following subparagraphs.

2.3 RESPONDING TO TXDS CONTROL PROGRAM PROMPTS

The operator's response to the PROGRAM:, INPUT:, or OUTPUT: prompt is used to specify (1) the device-name identifier of the input device on which the program to be loaded and executed is stored and/or (2) the file-name identifier of the program to be loaded and executed. When the file is on a diskette input device, the full response to any of the prompts requires inclusion of the diskette-name identifier (e.g. DSC, DSC2, DSC3, DSC4) and the file-name identifier (e.g. :TXLINK or :TXEDIT) and the extension. An example of a full response to a PROGRAM: prompt is:

PROGRAM: DSC:TXLINK/SYS

When the file is on a non-diskette device such as a cassette unit, card reader, line printer or other I/O device, the full response to any of the prompts requires inclusion of solely the device-name identifier (e.g., CS1 or CR). An example of a correct full response to a PROGRAM: prompt is:

PROGRAM: CR

Both device names and file names are called pathnames.

2.3.1 **PATHNAME SYNTAX.** When a pathname is used to indicate a device, it consists of the one- to four-character device name assigned to that device during system generation (see the *TX990 Operating System Programmer's Guide).*

A pathname which designates a file has basically three fields:

- A device or volume name to designate the diskette which contains the file. A device name is the one- to four-character name assigned to the diskette drive during system generation. A volume name may only be used on customized TX990 operating systems which include volume name support (see the *TX990 Operating System Programmer's Guide* section on system generation). Volume names are also one to four characters.

- A file name which is one to seven characters and separated from the device or volume name by a colon (:). The file name is specified when the file is created. The first character must be alphabetic (A-Z); the rest may be alphanumeric.

- An extension to the file name which is one to three characters and separated from the file name by a slash (/). The extension may also be specified when the file is created. The first character must be alphabetic; the rest must be alphanumeric. Extensions are commonly used to describe how a file is used, such as LST for listing files, SRC for source files, and OBJ for object files.

No imbedded blanks are allowed in any of the three fields.

When specifying pathnames in response to prompts made by any of the utility programs, some of the fields may be omitted, and the utility uses a default value for that field. Table 2-1 shows the possible pathname variations.

NOTE

The default-substitutes mentioned in table 2-1 are determined by the utility program being executed. Consequently, in some utility programs, a default-substitute may not exist. Further, the utility program being executed also determines whether or not a default-substitute results in an error.

The TXDS Control Program checks the syntax of all of the pathnames entered for utility programs before they are executed. If the pathname syntax is not legal, then the prompt associated with that entry is again printed out or displayed (to reprompt the operator).

Table 2-1. Pathname Syntax Variations

Pathname	Explanation
DEV:FILE/EXT VOL:FILE/EXT	This is the full pathname response for a diskette file. An example is: DSC:TXLINK/SRC. No default-substitute is employed when a full response is made.
:FILE/EXT	The missing DEV causes the default diskette name, defined during system generation, to be used in the device field.
DEV:FILE VOL:FILE	This causes a blank to be provided for the extension.
:FILE	The default diskette name, defined during system generation, is used for the device field and a blank is used as the extension.
:FILE/	This causes the default diskette name, defined during system generation, to be used for the device field and the extension to be defaulted as specified in the utility program being executed.
DEV/EXT VOL/EXT	This causes a default-substitute to be provided for the file as specified in the utility program being executed.
/EXT	This causes the default diskette name, defined during system generation, to be used for the device field and the file default to be defaulted as specified in the utility program being executed.
/	This causes the default diskette name, defined during system generation, to be used for the device field and the file and extension to default as specified in the utility program being executed.
DEV: VOL:	This causes a default-substitute to be provided for the file and extension as specified in the utility program being executed.
DEV	This is a full device name. No default-substitutes apply.

2.3.2 **PROMPT-RESPONSES**. The TXDS Control Program prompts the user to enter the program pathname, input pathname, output pathname, and option-selections. The TXDS Control Program then checks the pathnames for syntax. If the syntax is not correct, it will prompt the user again. After all of the responses to the prompts are entered, the TXDS Control Program loads and executes the specified program as task 10_{16}.

2.3.2.1 **PROGRAM: Prompt**. The operator's response to the PROGRAM: prompt must specify either the pathname of the program to be loaded and executed, or the task ID of a program already in memory.

Only one pathname can be entered in response to the PROGRAM: prompt. When the program is to be loaded as a privileged task (enabling the task to execute certain supervisor calls), the user must enter the pathname followed by a ",P". A task, when not linked with the TX990 Operating System, can only be made privileged when it is loaded. All tasks linked with the TX990 Operating System are privileged.

When the user enters only a slash (/) for the extension field in the PROGRAM: prompt pathname, the extension will default to SYS and SYS will be substituted into the pathname before any drives are searched.

When a PROGRAM: pathname, for a diskette configuration, does not specify the diskette transport drive, the TXDS Control Program starts a device-file search beginning with the diskette transport drive that is the default-substitute defined during system generation. For a standard TI-supplied TXDS system, the default-substitute is DSC. If the file is not on the diskette of the first default diskette transport drive, the TXDS Control Program will concatenate a 2 to DSC and the file search would then proceed to DSC2. In the same manner, the search continues to DSC3 and to DSC4. The search is only effective when the diskette default-substitute is the main diskette transport drive and when its device-name identifier is comprised of three characters, (i.e., DSC or any other three characters). It should also be noted that whenever the user specifies the device-name identifier in response to the PROGRAM: prompt, only the specified device (e.g., the specified diskette transport drive) is searched.

The file-name identifier for each utility program is listed in table 2-2. If a task ID is entered, it must be preceded by a "greater than" sign (>).

2.3.2.2 **INPUT: Prompt.** The operator's response to the INPUT: prompt is used to specify the pathname of the input information needed by the program during its execution. For example: the TXMIRA utility program uses the response to the INPUT: prompt to specify the pathname of the source file; the TXLINK utility program uses the response to the INPUT: prompt to specify the pathname of the individual object modules to be linked; and the TXCCAT utility program uses the response to the INPUT: prompt to specify the pathname of the individual files to be copied together. The other utility programs each use the response to the INPUT; prompt in the manner described under each of the utility program sections herein this manual. The operator can enter zero to three input pathnames separated by commas. The TXDS Control Program will check each parameter for syntax. If the syntax is wrong, the TXDS Control Program will prompt the user again. The user must enter the entire line again.

The INPUT: pathname default-substitutes for each utility program are listed and described in each utility program section of this *TXDS Programmer's Guide* and in each utility program section of the *TX990 Operating System Programmer's Guide.*

2.3.2.3 **OUTPUT: Prompt.** The response to the OUTPUT: prompt is the pathname for storage of the output information. For example: the TXMIRA utility program uses the response to the OUTPUT: prompt to specify the pathname where object is stored and assembly source file listings are to be presented; the TXLINK utility program uses the response to the OUTPUT: prompt to specify the pathname where the linked object is to be stored and where load map listings are to be presented; and the TXCCAT utility program uses the response to the OUTPUT: prompt to specify the pathname where the copied files are to be stored. The other utility programs each use the response to the OUTPUT: prompt in the manner described under each of the utility program sections herein this manual. Up to three pathnames (separated by commas) can be entered in response to the OUTPUT: prompt.

The OUTPUT: pathname default-substitutes for each utility program are listed and described in each utility program section of this *TXDS Programmer's Guide* and in each utility program section of the *TX990 Operating System Programmer's Guide.*

2.3.2.4 **OPTIONS: Prompt.** The operator's response to the OPTIONS: prompt is used to specify the option(s) selected from the total alternative options available for the program which is to be loaded and executed. These options are described in the applicable utility program section in this *TXDS Programmer's Guide* or in the *TX990 Operating System Programmer's Guide.*

Table 2-2. Utility Program File-Name Identifiers

File Name Identifier	Utility Program
:GENTX/SYS	System Generation [1,3]
:OBJMGR/SYS	Object Manager [1,3]
:BACKUP/SYS	Diskette Backup [1,3]
:SYSUTL/SYS	System Utility [1,3]
:LIST80/SYS	List 80-80 [1,4]
:DSKDMP/SYS	Diskette Dump [1,3]
:TXMIRA/SYS	Assembler [2]
:TXXREF/SYS	Cross Reference [2]
:TXLINK/SYS	Linker [2]
:TXCCAT/SYS	Copy Concatenate [2]
:TXEDIT/SYS	Text Editor [2]
:IBMUTL/SYS	IBM Diskette Conversion [2,3]
:TXDBUG/SYS	Standalone Debug[2]
:TXLUNO/SYS	LUNO Assignment[2,3]

Notes:

[1] - TX990 Operating System utility program.

[2] - TXDS Terminal Executive Development System utility program.

[3] - Capable of being loaded and executed using OCP commands or the TXDS Control Program.

[4] - This utility can only be excuted using OCP.

2.3.3 **SPECIAL KEYBOARD CONTROL KEYS.** The special keyboard control keys are described as follows:

1.	RUB OUT/DELETE LINE	Allows the operator to reenter a parameter. Pressing the RUB OUT key causes a line feed followed by a carriage return. The operator may then enter the line again.
2.	CONTROL H/Back Arrow	Allows the operator to backspace by character and correct a typing error.
3.	Carriage Return/NEW LINE	Causes TXDS Control Program to terminate if the carriage return or NEW LINE was the only entry in response to the PROGRAM: prompt, otherwise terminates a prompt line entry.

4.	ESCAPE/RESET	If an ESCAPE or RESET is entered during a print out, the TXDS Control Program terminates.
5.	,	Causes a default to be activated when entered as the response to the INPUT: or OUTPUT: prompts.
6.	&	In any prompt line, pressing the & key as the first character in the response causes the TXDS Control Program to restart with the PROGRAM: prompt.
7.	*	When entered after a prompt line entry, in place of a carriage return, permits the next prompt line to be entered without being prompted by the TXDS Control Program. When a prompt line is terminated with an asterisk (*) followed by a carriage return, no more prompts are given and default-substitutes are made by the utility program for those pathnames not entered. The experienced user can enter all or several of the parameters on one prompt line.

The following examples utilize the asterisk (*) feature in lieu of the INPUT:, OUTPUT:, and OPTIONS: prompts:

Example 1:

To load the TXEDIT utility program after the TXDS Control Program has been loaded, the asterisk (*) is used as presented in the following example:

```
TXDS   936215   **   010/77   2:05

PROGRAM:   DSC:TXEDIT/SYS*DSC:TASK2/SRC*DSC:SCRATCH/SRC*
```

(where DSC:TASK2/SRC is the INPUT: pathname; DSC:SCRATCH/SRC is the OUTPUT: pathname; and the OPTIONS: entry is provided by the default-substitution specified in the TXEDIT utility program.)

The above can also be entered as follows:

```
TXDS   936215   **   010/77   2:05

  PROGRAM:    DSC:TXEDIT/SYS
    INPUT:    DSC:TASK2/SRC*DSC:SCRATCH/SRC*
```

Example 2:

To load the SYSUTL utility program after the TXDS Control Program has been loaded, the asterisk (*) is used as follows:

```
TXDS   936215   **   101/77   2:05

PROGRAM:   :SYSUTL/SYS***CF,:TEMP/OBJ
```

(where the INPUT: and OUTPUT: parameters are null and the OPTIONS: parameter is CF,:TEMP/OBJ.

NOTE

1. In the above examples, it is necessary to press the carriage return key at the end of the parameter line to cause the program to be loaded and executed.

2. If a parameter line ends with an asterisk (*) and a pathname is not entered for each prompt, then default substitutes are made by the utility program for those pathnames not entered.

Example 3:

The following example utilizes the comma (,) to cause a default-substitution to be made in the OUTPUT: pathname below.

```
TXDS   9326215   **   010/77   2:05

  PROGRAM:    :TXMIRA/SYS
    INPUT:    :TASK1
   OUTPUT:    ,CRT
  OPTIONS:    M800,X,L
```

(where the OUTPUT: pathname defaults to a substitute specified in the TXMIRA Assembler utility program.)

The following example utilizes both the asterisk (*) and the comma (,) special keyboard controls:

To load the TXMIRA Assembler utility program after the TXDS Control Program has been loaded, the asterisk (*) is used as follows:

```
TXDS:   936215   **   010/77   2:05

PROGRAM:   :TXMIRA/SYS*:TASK1*,CRT*M800,X,L
```

(where TASK1 is the INPUT: pathname and where the OUTPUT: pathname is the default-substitute provided in the TXMIRA Assembler utility program.)

2.4 BACKING UP TI-SUPPLIED DISKETTES

Before the TI-supplied TXDS diskettes are used, they should be backed up onto scratch diskette(s): (1) to ensure that a backup diskette is available if the diskette(s) is destroyed; and (2) to ensure that either of the two diskettes (i.e., the original diskette and the backup diskette) will always be available to do future system generations with the use of a minimum 16K-memory configuration.

Backup the system diskette by performing the following step-by-step procedure:

1. Remove the system diskette from diskette transport drive 1 and insert in its place the TX990 parts diskette.

2. Take a scratch diskette and place it into diskette transport drive 2.

3. Start the backup procedure by bidding the BACKUP utility as follows:

   PROGRAM: :BACKUP/SYS*(C/R)

   NOTE

   (C/R) signifies a NEW LINE entry on a 913 VDT, a RETURN on a 911 VDT, and a carriage return on an ASR.

   The items underlined below are the user's responses to the prompts.

   The Diskette Backup and Initialize Utility is loaded:

   BACKUP & INITIALIZE UTILITY 936212*A

   OUTPUT DISC OR VOLUME NAME? DSC2 (C/R)

4. If the diskette is not initialized the following message is displayed:

   THE OUTPUT DISC MUST BE INITIALIZED

5. If the diskette is initialized, the following message is displayed:

   DELETE ALL FILES ON DSC2? (Y/N) N(C/R)

6. The rest of the procedure is:

   INITIALIZE DSC2? (Y/N) Y(C/R)

   OUTPUT DISC ID SCRATCH DISR (C/R)

   OUTPUT VOLUME NAME BKUP (C/R)

   After the diskette is initialized, the following will be printed:

   COPY FILES? (Y/N)

At this point remove the diskettes from drive 1 and load the system diskette to be backed up. Then respond to the prompt as follows:

COPY FILES? (Y/N) Y(C/R)

VERIFY FILES? (Y/N) Y(C/R)

INPUT PATHNAME? DSC (C/R)

ERROR LOG:

For a discussion of any error messages which are output at this point, see the Diskette Backup Utility section of the *TX990 Operating System Programmer's Guide.*

THE DISC IS NOW BEING VERIFIED
ERROR LOG:

For a discussion of any error messages which are output at this point, see the Diskette Backup Utility section of the *TX990 Operating System Programmer's Guide.*

FINISHED!
COPY FILES? (Y/N) N(C/R)
VERIFY FILES? (Y/N) N(C/R)
SYSTEM FILE PATHNAME: DSC2

If the backed up diskette has a system file on it, respond to the prompt with the name of that file. For example, if the system file was :SYSASR/CMP, the response should be:

SYSTEM FILE PATHNAME? DSC2:SYSASR/CMP(C/R)

Termination of BACKUP is accomplished by entering an asterisk to the next prompt:

OUTPUT DISC OR VOLUME NAME: *(C/R)
BACKUP & INITIALIZE UTILITY ENDED
TXDS 936215 **

2.5 TXDS CONTROL PROGRAM ERROR MESSAGES

Refer to table 2-3 for a list of error messages, the reason for each error, and the recovery method.

Table 2-3. TXDS Control Program Error Messages

Error	Reason	Recovery
nn – BAD PGM LOAD	nn represents error code listed in Error Appendix D.	Reenter parameter
– BAD PGM LOAD	Cannot find object file.	Reenter parameter
nn – CAN'T BID TASK	nn represents the task state code of task 10_{16} listed in State Code Appendix C.	Reenter parameter.
CAN'T GET COMMON– ABORTED	System was configured without COMMON.	Configure a system with 168 bytes of common.

SECTION III

VERIFICATION OF OPERATION

3.1 INTRODUCTION

This section provides several short procedures to verify that the software is operating properly. These procedures are listed below and described in the following procedural steps:

- Load and Initialize TX990 Operating System
- Load and Initialize TXDS Control Program
- TXCCAT Verification Procedure
- TXEDIT Verification Procedure
- TXMIRA Verification Procedure
- OBJMGR Verification Procedure
- TXLINK Verification Procedure

The Operating System Diskette mentioned in the procedure refers to either TXDS System Diskette 2 (for VDT systems) or TXDS System Diskette 3 (for ASR systems).

3.2 REQUIREMENTS

This procedure requires the following items in addition to the required hardware for a TXDS system:

- *TX990 Operating System Programmer's Guide*
- TXDS System Diskette
- TX990 Parts Diskette

3.3 OPERATION

The following steps of the verification procedure demonstrate that the system is operational.

1. Initialize TX990 by loading TXBOOT from TXDS system diskette.

   a. Insert TXDS System Diskette in diskette drive #1.

   b. On the front panel, press the following pushbutton switches to load the system.

      HALT
      RESET
      LOAD

   c. Bid the TXDS control program by entering ! at the system console.

2. Place scratch diskette in drive #2. If diskette is not initialized; place TX990 Parts Diskette in diskette transport drive #1 and execute the Disc Backup and Initialize (BACKUP) utility program as described in Section X of *Model 990 Computer TX990 Operating System Programmer's Guide.* Replace the System Diskette in transport drive 1.

3. Copy :TXTST1/SRC from System Diskette to scratch diskette using TXCCAT. With System Diskette in transport drive 1 and the scratch diskette in transport drive 2, enter the following commands to copy :TXTST1/SRC to the scratch diskette:

```
PROGRAM:  DSC:TXCCAT/SYS
  INPUT:  DSC:TXTST1/SRC
 OUTPUT:  DSC2:TXTST1/SRC
OPTIONS:  (carriage return)
```

4. After completion of the copy, TXDS will come up and should be given the following parameters in order to execute TXEDIT:

```
PROGRAM:  DSC:TXEDIT/SYS
  INPUT:  DSC2:TXTST1/SRC
 OUTPUT:  (carriage return)
OPTIONS:  (carriage return)
```

5. Position TXEDIT to line 19 of TXTST1/SRC using the DOWN (D) command and print the line to assure that the pointer is positioned correctly.

```
?D19
?P
```

The following line should be printed

```
19          DATA OLD,CNT1
```

6. Edit line 19 to enable printing the new message by using the Change (C) command. Replace line 19 with the following line, spacing the DATA over seven spaces and the NEW,CNT one more space.

```
?C19-19

        DATA NEW,CNT
```

Enter an extra carriage return to terminate the Change command.

7. Print line 18 and the modified line to ensure the change has been made correctly.

```
?T
?D17
?P2

18          BLWP @WRITE          PRINT MESSAGE
            DATA NEW,CNT
```

8. Terminate the editing session by executing the Quit (Q) command.

```
?Q
```

Upon executing the Quit command, TXEDIT issues a prompt to ensure that it is time to terminate. After responding with a 'T', the current input file and buffer are copied to the output scratch file. Respond with a 'Y' when asked if the scratch file is to be copied to the input file.

```
TERMINATE/CONTINUE?T
    TEXT IN SCRATCH FILE
TRANSFER TO INPUT? Y
END EDIT
```

TXEDIT is then terminated and the TXDS control program is rebid.

9. Now assemble TXTST1/SRC using TXMIRA. The assembled object is directed to the file TXTST1/OBJ on the scratch diskette. The object is designated as compressed object on the options line. The assembly output listing is listed to the default system printer as specified in the OPTIONS: parameter. Also, the cross reference listing output from TXXREF is listed to the default system printer. The sample output listings are illustrated in figure 3-1.

Enter the following parameters.

```
PROGRAM:  DSC:TXMIRA/SYS
   INPUT:  DSC2:TXTST1/SRC
  OUTPUT:  DSC2:TXTST1/OBJ
 OPTIONS:  C,L,X
```

The output listing should reflect the changes made in step 6. Verify that no errors are detected by TXMIRA.

10. Place TX990 Parts Diskette in drive #1.

11. Object Manager is then loaded and executed to combine the three required object modules into one module for linking into the TX990 System. Execute the object manager as follows:

```
PROGRAM:  DSC:OBJMGR/SYS*

990 OBJECT MANAGER 939870 **
```

After printing the above message, object manager requests specification of a file in which the combined object is to be placed. At this point the TX990 Parts Diskette should be removed and the System Diskette installed in drive #1. After specifying the output file, enter the pathname of the three required object files and designate that each input file is to be copied and rewound.

```
TXTST1          TXMIRA  936227 **     17:12:02     088/77      PAGE 0001
TXDS TEST PROGRAM  937808-9901**

  0002                        IDT  'TXTST1'
  0003              *
  0004                        REF  CNT, NEW, WRITE
  0005              *
  0006 0000 0006'             DATA TSTWSP, START, 0
       0002 0045'
       0004 0000
  0007 0006         TSTWSP    BSS  32
  0008 0026 1600    ENDPRG    DATA >1600              END OF PROGRAM OP CODE
  0009              *
  0010                        DXOP SVC, 15            *** DEFINE XOP
  0011              *
  0012 0028   0D    OLD       BYTE >0D, >0A, >0A
       0029   0A
       002A   0A
  0013 002B   20              TEXT ' OLD MESSAGE -- WRONG !!'
  0014 0043   0D              BYTE >0D, >0A
       0044   0A
  0015       001D   CNT1      EQU  $-OLD


TXTST1          TXMIRA  936227 **     17:12:02     088/77      PAGE 0002
TXDS TEST PROGRAM  937808-9901**

  0017              START
  0018 0046 0420              BLWP @WRITE             PRINT MESSAGE
       0048 0000
  0019 004A 0028'             DATA OLD, CNT1
       004C 001D
  0020 004E 2FE0              SVC  @ENDPRG            END OF PROGRAM
       0050 0026'
  0021                        END

 0000 ERRORS


TXXREF 937542 **    17:12:33     088/77     PAGE 0001

CNT                 0004
CNT1      0015      0019
ENDPRG    0008      0020
NEW                 0004
OLD       0012      0015  0019
START     0017      0006
SVC                 0010
TSTWSP    0007      0006
WRITE               0004  0018

THERE ARE 0009 SYMBOLS
```

Figure 3-1. TXMIRA Sample Output Listing

```
                    OUTPUT FILE: DSC2:TXTST/OBJ
     INPUT FILE: DSC2:TXTST1/OBJ
     REWIND INPUT FILE? Y
TXTST1      ?      C
END-OF-FILE

     INPUT FILE: DSC:TXTST2/OBJ
     REWIND INPUT FILE?      Y
TXTST2      ?      C
END-OF-FILE

     INPUT FILE: DSC:TXTST3/OBJ
     REWIND INPUT FILE?      Y
TXTST3    ?          C
END-OF-FILE

     INPUT FILE:   *
END OBJECT MANAGER
```

Object manager terminates upon entering an asterisk on the input line.

12. Place the System Diskette in diskette drive 1. Execute TXLINK to link the object manager output as follows:

```
  PROGRAM:   DSC:TXLINK/SYS
    INPUT:   DSC2:TXTST/OBJ
   OUTPUT:   DSC2:TXTEST/OBJ,LOG
  OPTIONS:   C,ITXTEST,L
TXLINK  937537  **
```

The output line contains the name of the file TXTEST/OBJ in which the linked object is to be placed, as well as the device to which the load map is directed. 'C' specifies that the object is to be compressed, 'ITXTEST' designates that the IDT for the linked object is to be TXTEST, and 'L' specifies that the output of TXLINK is to be listed. The default memory size of 12K is available for the link.

Following is the link output.

```
TXLINK      937537 *A    10:51:01      032/77      PAGE 0001
TXTEST                LENGTH 00A0

    MODULE     LENGTH     ORIGIN     DATE        TIME

    TXTST1     0052       0000       .           :
    TXTST2     001E       0052       01/25/77    15:11
    TXTST3     0030       0070       01/25/77    15:14

                                  DEFINITIONS

A CNT       001D    NEW     0052    WRITE   0070
```

13. The TXDS Test Program is now ready for execution. Execute by entering the compressed object module name on the program line followed by *. Upon executing, the following message is issued prior to termination and TXDS is rebid.

PROGRAM: DSC2:TXTEST/OBJ*

HAVE A GOOD DAY !!

SECTION IV

CREATING AND EDITING PROGRAM SOURCE CODE

4.1 INTRODUCTION

The TXDS Text Editor (TXEDIT) Utility Program provides the user/programmer with the capability of editing the text of source programs and object programs and, in addition, the capability of creating source programs. Basically, 21 TXEDIT commands are available to fulfill the programmer's needs. The commands are grouped as follows:

- Setup commands:
  - Start Line Numbers (SL) command
  - Stop Line Numbers (SN) command
  - Set Print (SP) column margin number command
  - Set Margin (SM) for Find command
  - Set Tabs (ST) command
- Pointer-Movement commands
  - Down (D) command
  - Up (U) command
  - Top (T) command
  - Bottom (B) command
- Edit commands
  - Change (C) command
  - Insert (I) command
  - Move (M) command
  - Remove (R) command
  - Find (F) string command
- Print commands
  - Limits (L) command
  - Print (P) command

- Output commands

  Keep (K) command

  Quit (Q) command

  End (E) command

- Terminate-Sequence commands

  Terminate (T) command

  Continue (C) command

All of the TXEDIT commands are capable of being entered via the keyboard of the system console. To edit a program or record, the user must first have the program or record recorded on a TI diskette or cassette. The text is then edited by feeding it from the TI diskette or cassette (hereinafter referred to as the input file) into a memory buffer where the editing is performed and then out to the scratch file (until an EOF character is read or a Quit command is entered). If further editing is required, the text data is reversed to flow back from the scratch file into the memory buffer and back to the input file (until the EOF is read again or a Quit command is entered again). This transfer between the two files (with multiple editing activities being automatically performed during each pass) continues until the user is finished. At that time, the TXEDIT program provides a printout or display on the system console which states whether the input file or the scratch file contains the final edited text. The user then has the option of using a command to transfer the resultant final edited text back to the input file in substitution of the preedited source program or the preedited object program or record.

CAUTION

The user should ensure that the input file is not destroyed by copying it onto a temporary file diskette or cassette.

The TXEDIT program may be executed in a Model 990/4 microcomputer or a 990/10 minicomputer configured to support a TX990 Operating System. The minimum configuration includes a computer with 16K of memory and an interactive operator system console, the LOG.

The following paragraphs describe various TXEDIT program functions and procedures. A TXEDIT loading procedure is presented. Specific editing procedures using the TXEDIT commands are presented for: changing, adding, moving, or removing source or object records in the buffer and to locate and modify a character string in a group of records; using editor commands to move the text editor's buffer line pointer; moving lines/text into and out of the buffer; and using special terminal keyboard characters. Procedures for coding source or object files and writing a new source program are also explained. A description of possible error and warning messages is provided. Concluding this section is an example of how to enter and edit a source program and a discussion of how to edit an object program.

4.2 LUNOs

LUNOs 7 and 8 are assigned by the text editor. LUNO 7 is assigned to the input file, and LUNO 8 is assigned to the output file. The text editor uses the system console as the interactive device. When the text editor terminates, all files are closed and all LUNOs are released.

4.3 LOADING TXEDIT

The user can load the TXEDIT utility program only by use of the TXDS Control Program. (The TXEDIT utility program cannot be loaded via OCP.) After the TXDS Control Program is executed using the procedure in Section II, responses to the TXDS Control Program's PROGRAM:, INPUT:, OUTPUT: and OPTIONS: prompts are then entered by the user via the keyboard of the system console. The user responds to the PROGRAM: prompt as follows:

```
PROGRAM:  :TXEDIT/SYS
```

The response to the INPUT: prompt requires the pathname of the source program file location on diskette or cassette. If the diskette file is specified by the pathname and none exists, it will be created. This is the correct procedure for generating a new source file. An insert (I) command may then be used to generate lines of source code. The response to the OUTPUT: prompt requires the pathname of the scratch file location on diskette or cassette. If the diskette file is specified by the pathname and none exists, it will be created. The OUTPUT: file pathname must not be the same as the INPUT: file pathname. In response to the OPTIONS: prompt, the user may specify the size of the memory buffer. Under the TXEDIT utility program, this is the only option available to be specified. The size of the memory buffer is specified by the user entering an M followed by a decimal number (which may vary from one to five characters in length). The decimal number specifies the number of bytes to be used for the memory buffer. The memory size is determined using the following procedural example: if the user wishes to edit 75 lines of text, each character on each line is used to specify one byte; further, each line is preceded by a six-byte header and followed by a one-byte carriage return. Consequently, if each line of text has an average length of 40 bytes plus 6 bytes for the header and 1 byte for the carriage return, then 75 lines of text would require 3525 bytes.

An example of loading TXEDIT from *diskette* follows:

```
TXDS  936215**0017 2:10

PROGRAM:  :TXEDIT/SYS
   INPUT:  DSC2:UPDATE/SRC
  OUTPUT:
 OPTIONS:  M4000
```

The above response-entries to the prompts cause the TXEDIT utility program to be loaded from diskette into memory and then to be executed. The OUTPUT: pathname is provided by the TXEDIT utility program with :SCRATCH/SRC as the default-substitute.

An example of loading a file from *cassette* follows:

```
TXDS  936215**0017 2:10

PROGRAM:  :TXEDIT/SYS
   INPUT:  CS1
  OUTPUT:
 OPTIONS:  M4000
```

The above response-entries to the prompts cause the TXEDIT utility program to be loaded from diskette into memory and then to be executed.

Table 4-1. TXEDIT Default-Substitutes

Entry	Pathname	Default-Substitute
INPUT:	DEVICE	Default disc drive
	FILE	No default-substitute
	EXTENSION	SRC
OUTPUT:	DEVICE	Default disc drive
	FILE	SCRATCH
	EXTENSION	SRC
OPTIONS:	M (memory)	3000 bytes

4.4 COMMANDS

4.4.1 GENERAL. The TXEDIT utility program supplies 21 edit commands to fulfill the user's needs. Further, eight special keys/characters are also provided to meet general utility needs (e.g., RUB OUT, ESC, et. al.). The commands are entered at the keyboard of the system console in response to the printing of a question mark (?); and after the command is entered, it is executed by entering a carriage return. The syntax of the command is free form in that one or more spaces may be inserted between characters and operands of the commands. A list of the commands and a brief description of each command is provided in table 4-2. The detailed descriptive information pertaining to each command is provided in the following paragraphs.

4.4.2 COMMAND OPERANDS. Command operands are used to specify a number of lines, line numbers, or displacements from the pointer. The edit commands and one of the print commands may specify a group of lines by first and last line number of by a number of lines relative to the pointer.

4.4.3 SYMBOL DEFINITION. The symbols used in conjunction with TXEDIT commands are defined as follows:

- Angle brackets < > enclose items required to be supplied by the user.
- Brackets [ ] enclose optional items.
- Braces { } enclose items between which a choice must be made; one, and only one, of the items must be included.
- Items in capital letters must be entered as shown.

NOTE

The syntax definitions and examples presented in this section do not have spaces between the characters of the two-character commands, between the command and operands, or between operands. Spaces may be entered at these points if desired, and all operands are decimal numbers.

Table 4-2. List of Commands and Special Keys/Characters

Command Syntax	Description
	SETUP COMMANDS
SL	Start Line numbers (SL) command causes line numbers to be printed with each line of text.
SN	Stop line Numbers (SN) command causes line numbers not to be printed.
SP	Set Print margin (SP) command sets the right boundary for print display.
SM	Set Margin (SM) for Find command sets the left and right boundaries for the Find command.
ST	Set Tabs (ST) command sets up to five tab stops.
	POINTER-MOVEMENT COMMANDS
D	Down (D) command moves the pointer down toward the bottom of the buffer.
U	Up (U) command moves the pointer up towards the first line in the buffer.
T	Top (T) command moves the pointer to the first line in the buffer.
B	Bottom (B) command moves the pointer to the last line in the buffer.
	EDIT COMMANDS
C	Change (C) command removes lines from the buffer and inserts new ones in their place. The new lines are input from the terminal.
I	Insert (I) command takes input from the terminal and places the new lines into the buffer.
M	Move (M) command moves lines from one place in the buffer to another.
R	Remove (R) command deletes lines from the buffer.
F	Find string (F) command searches for the first occurrence of a character string in a line and replaces it with another string of characters.
	PRINT COMMANDS
L	Limits (L) command causes the first line and the last line to be displayed.
P	Print (P) command displays lines of text.

Table 4-2. List of Commands and Special Keys/Characters (Continued)

Command Syntax	Description
	OUTPUT COMMANDS
K	Keep (K) command takes lines of text out of the buffer and puts them in the output file.
Q	Quit (Q) command takes lines of text out of the buffer or the input files and puts them in the output file.
E	An (E) command terminates without writing an EOF to the output file.
	TERMINATE-SEQUENCE COMMANDS
T or C	Allows the user to make multiple single directional editing passes on a source or object program.
	SPECIAL KEYS/CHARACTERS
CTRL-H	Pressing the control key and the H key simultaneously on the hard copy terminal causes the terminal to backspace a character to enable rewriting over an entered character-error.
RUB OUT	The RUB OUT key causes the line just entered to be deleted so that a new line can replace it.
CTRL-I	Pressing the control (CTRL) key and the I key simultaneously on a hard-copy terminal causes a tab stop to be entered in the input string, although only one space will be echoed on the terminal.
ESC/RESET	Pressing the ESCape or RESET key on the system console causes a display to be aborted.
position keys	When using a VDT, only the left position key (←) and the right (→) position key are recognized. The up and down position keys cause garbage to be entered into the input string. The left position key causes characters to be deleted from the character string; a right position key causes whatever was under the cursor to be entered.
DELETE LINE	DELETE LINE on a VDT acts the same as a RUB OUT on a hardcopy terminal.
TAB	A SPACE character is echoed. The TAB is interpreted by the text editor and spaces are inserted to fill the text line to the next TAB setting. Refer to paragraph 4.4.5.5 which follows.

4.4.4 **SPECIAL KEYS/CHARACTERS.** The following special characters are recognized by the text editor when the terminal is an ASR or KSR. A backspace character (CTRL H) backspaces one character position. A RUB OUT character deletes the line that has just been entered from the keyboard. On an ASR or KSR, a tab (CTRL I) echoes as one space upon character input, but moves to the nearest tab stop when the line is printed. (Tab stops are initially defined at character positions 8, 13, 31, and 33.) An escape (ESC) entered from the keyboard during print output causes the current I/O operation and the command to be aborted; a question mark (?) prompt is then printed out or displayed, to which another TXEDIT command-response-entry can be made.

The following special characters are recognized by the text editor when the terminal is a VDT. The position keys will move the pointer for backspace, or forward space. The DELETE LINE key will delete the line that has just been entered from the keyboard. The RESET key, when entered during a printout, causes the current I/O operation and the command to be aborted. If the space bar is entered during a printout, the printout will halt until the space bar is entered again. This allows the user to scan the printout before it rolls off the top of the screen without aborting the I/O operation.

4.4.5 **SETUP COMMANDS.** Setup commands may be entered immediately following loading of TXEDIT to: set limits for the Find command; set the right margin for printing; enable or inhibit printing of line numbers; set tabs. If no Setup command is entered, line numbers are printed. The right margin for lines or print corresponds to column 72. Columns 1 through 72 are scanned by the Find command, and tabs are preset at 8, 13, 31, and 33, which are the standard columns for source code instructions, operands, and comments.

Setup commands may be entered anytime during an editing session. It is often desirable to change the Find command limits before entering a Find command, so that only certain columns are searched. The user may want to inhibit the printing of line numbers to enable more source codes to be printed on a line. If the user is generating code to be assembled by TXMIRA, he may want to set the right margin to column 60, since TXMIRA does not scan characters past column 60.

4.4.5.1 **Start Line Numbers (SL).** The Start Line Numbers (SL) command causes TXEDIT to print line numbers to the left of each statement or record. Syntax for the SL command is as follows:

SL

The SL command is used to restore printing of line numbers after line number printing has been inhibited by execution of an SN command.

4.4.5.2 **Stop Line Numbers (SN).** The Stop Line Numbers (SN) command causes TXEDIT to omit printing of line numbers except in the message resulting from the Limits (L) command. The syntax for the SN command is as follows:

SN

The SN command may be entered initially or at any time during the edit operation. Omitting the line numbers when editing object code may be desirable to permit printing the entire record.

4.4.5.3 **Print Margin (SP).** The Print Margin command specifies the column number of the right margin where printing is to end, except for the message resulting from the Limits (L) print command, described in this section. The syntax for the SP command is as follows:

SPs

The s represents the column number of the right margin where printing is to end (i.e., one of the columns between 10 and 80, inclusive). If line numbers are being printed, the line numbers are included in the margin column. Line numbers use six columns, so that if the right margin is column 60, only 54 characters plus 6 line number character digits and blanks for spacing are printed. The following example shows an SP command that specifies column 60 as the right margin for printing:

```
?SP60
```

4.4.5.4 Set Margin (SM). The Set Margin (SM) command specifies left and right limits for the Find command. Syntax for the SM command is as follows:

```
SMs,t
```

There must be a comma between s and t. The Find command scans from column s through column t and may be limited to the desired field by the SM command. The default value for the scan limits is from column 1 to column 72 (or the end of the line if less than 72). The following example shows an SM command that limits the scan of subsequent Find commands to columns 8 through 25:

```
?SM8,25
```

4.4.5.5 Set Tabs (ST). The Set Tabs (ST) command allows up to five tabs to be set between column 1 and 72. Syntax for the ST command is as follows:

```
STn,n,n,n,n
```

NOTE

There must be a comma between every column number. The column number is indicated by n. Tabs must be set in ascending order, and if they are not, a blank will be inserted for the descending tab. If more than five tabs are entered, an INVALID OPERAND error message is issued; however, the first five tabs are set and ready for use. If no column numbers are entered, all tabs are cleared.

4.4.6 POINTER-MOVEMENT COMMANDS. Pointer-movement commands may be used to move the pointer to any line in the buffer of TXEDIT. Initially, the pointer is at line 1. Moving the pointer with the Down (D) command past the last line in the buffer causes TXEDIT to read source lines or object records from the input file to fill the empty lines. Other commands move the pointer upward a specified number of lines, or to the top of the buffer, or downward to the bottom of the buffer. The pointer-movement commands permit the user to move the pointer as desired for effective use of commands that identify lines by specifying a displacement from the pointer. The pointer commands are described in the following subparagraphs.

4.4.6.1 Down (D). The Down (D) command causes TXEDIT to move the pointer down a specified number of lines. When the specified move is to a line number greater than the contents of the buffer, TXEDIT adds lines to the buffer and reads records from the input file to fill these lines. The syntax for the D command is as follows:

```
Dn
```

The pointer is moved down n lines. The range of n is 1 to 9999, and the default value when n is omitted is 1. The D command may be entered to read in lines from the input file or to move the pointer to a line farther down in the buffer. Initially, or when the pointer is at the bottom of the buffer, TXEDIT reads n lines from the input file. When the pointer is m lines above the bottom of the buffer and n is greater than m, TXEDIT reads n - m lines from the input file. In each of these cases, the pointer is at the bottom of the buffer after execution of the D command. However, when the pointer is m lines above the bottom of the buffer and m is greater than or equal to n, no lines from the input file are read. The pointer is m - n lines above the bottom of the buffer after execution of the command. The following example shows a D command to move the pointer down 30 lines.

```
?D30
```

4.4.6.2 Up (U). The Up (U) command moves the pointer up a specified number of lines. Syntax for the U command is as follows:

```
Un
```

The pointer is moved up n lines. The range of n is 1 to 9999, and the default value when n is omitted is 1. The U command may be entered to move the pointer up to a specific line in the buffer. The following example shows a U command to move the pointer up 6 lines:

```
?U6
```

4.4.6.3 Top (T). The Top (T) command moves the pointer to the first line in the buffer. The syntax for the T command is as follows:

```
T
```

4.4.6.4 Bottom (B). The Bottom (B) command moves the pointer to the bottom (i.e., last) line in the buffer. The syntax for the B command is as follows:

```
B
```

4.4.7 EDIT COMMANDS. The edit commands add, change, remove, rearrange, or scan lines of source or object code, and act upon a set of lines in the buffer specified by line number or by a displacement from the pointer. The edit commands are described in the following paragraphs.

4.4.7.1 Change (C). The Change command deletes a specified set of lines and permits input of one or more lines to replace the deleted lines. The syntax for the command is as follows:

```
C { <s> - <t>   }
  { [+] [<n>]   }
  { - <n>       }
```

Line s through line t are deleted, or n lines with respect to the pointer are deleted. The values of s and t can be equal. As many replacement lines as required are entered. Each line is followed with a carriage return; the last line is followed with two carriage returns. When n is preceded by a minus sign, n lines preceding the pointer line are deleted, but the pointer line is not deleted. The new lines are inserted in their place. When n is unsigned or is preceded by a plus sign, n lines beginning with the pointer line are deleted, and the new lines are inserted. When no operand is entered, the pointer line is deleted. and replaced by the new lines. When the pointer line is deleted, the pointer is moved to the next line of the buffer following the newly inserted lines. If the line that was changed was the last line in the buffer, the pointer will be at the first line in the buffer. The following example shows a C command to change lines 5 through 7, replacing them with four lines.

```
?C5-7
LOD     MOV     1,4
        AI      4,1
        CI      4,WA+60
        JLT     SUM
```

The following example shows a C command to change the pointer line and the two lines that follow the pointer, replacing them with two lines:

```
?C3
LOD     MOV     1,4
        CI      4,WA+60
```

4.4.7.2 **Insert (I).** The Insert (I) command permits input of one or more lines following the pointer or a specified line. The syntax for the I command is as follows:

I [<s>]

As many lines as required are entered. Each line is followed with a carriage return; the last line is followed with two carriage returns. When s is in the range of 1 to 9999, lines are inserted following line s. When s is 0, lines are inserted ahead of the top line in the buffer. When s is omitted, lines are inserted following the pointer line. The following example shows the use of the I command to insert two lines following line 10:

```
?I10
        CKON
        DEC     7
```

4.4.7.3 **Move (M).** The Move (M) command moves a specified block of lines to a specified location and deletes the block of lines at the previous location. The block is specified by first and last line numbers or by a number of lines preceding or following the pointer. The location to which the block will be moved is specified as a line number or as the pointer. The syntax for the M command is as follows:

M { <s> - <t>, [<r>] | [+] <n>, <r> | - <n>, <r> }

Line s through line t are moved, or n lines with respect to the pointer are moved. When n is preceded by a minus sign, n lines preceding the pointer line, but not the pointer line, are moved. When n is unsigned or preceded by a plus sign, n lines beginning with the pointer line are moved.

The specified lines are placed following line r when r is greater than zero. When r is zero, the specified lines are placed ahead of the top line in the buffer. When r is omitted, the lines are placed following the pointer line, but r can only be omitted when specifying lines s through line t. Numbered lines moved by the Move command retain their original line numbers, if any. When the pointer line is moved, the pointer moves with it. When s and t are specified, r must be less than s or greater than t. The following example shows an M command to move lines 6 through 8 to follow line 25:

```
?M6-8,25
```

The command in the following example moves four lines beginning with the pointer line to follow line 30:

```
?M4,30
```

4.4.7.4 Remove (R). The Remove (R) command removes a block of lines. The block is specified by first and last line numbers, or by a number of lines preceding or following the pointer. The syntax for the R command is as follows:

$$R \left\{ \begin{array}{l} \langle s \rangle - \langle t \rangle \\ [+] \ [\langle n \rangle] \\ - \langle n \rangle \end{array} \right\}$$

Lines s through t are removed, or n lines with respect to the pointer are removed. When n is preceded by a minus sign, n lines preceding the pointer line, but not the pointer line, are removed. When no operand is entered, the pointer line is removed. When the pointer line is removed, the pointer is moved to the next line of the buffer. If the last line in the buffer is removed the pointer will point to the first line in the buffer. The following example shows an R command to remove line 12:

```
?R12-12
```

The command in the following example removes the three lines preceding the pointer line:

```
?R-3
```

4.4.7.5 Find (F). The Find (F) command scans a block of lines for the first occurrence in each line of the specified character string. Optionally, the command may replace the string with or without printing the resulting line, or may print the line and permit the user to specify whether or not to substitute the string. In all cases, the command prints the count of matching lines found. The block is specified by first and last line numbers, or by a number of lines preceding or following the pointer. The syntax for the F command is as follows:

$$F \left\{ \begin{array}{l} \langle s \rangle - \langle t \rangle \\ [+] \ \langle n \rangle \\ -\langle n \rangle \end{array} \right\} \left\{ \begin{array}{l} L \\ F \end{array} \right\} \langle dl \rangle \langle string1 \rangle \langle dl \rangle \left\{ \begin{array}{c} [P] \\ \langle d2 \rangle [\langle string2 \rangle] \langle d2 \rangle [V] \ [P] \end{array} \right\}$$

Line s through line t are scanned, or n lines are scanned. When n is preceded by a minus sign, n lines preceding the pointer line, but not the pointer line, are scanned. When n is unsigned or preceded by a plus sign, n lines beginning with the pointer line are scanned.

When an F is entered following the lines to be scanned, the columns specified in an SM command are scanned for the first occurrence in each line. Columns 1 through 72 are the default scan columns unless the line is shorter than 72 columns, in which case it will scan to the end of the line. When an L is entered, the command performs a label scan, beginning at the left limit and extending to the first space.

The character string used in the scan is designated string1, and is enclosed by identical characters, each represented by dl. The character represented by dl may be any character that does not appear in string1.

When no other parameter is entered, the command scans the specified lines and prints the number of lines in which a match of string1 was found. When P is entered following dl, the command prints each line in which a match of string1 was found, and also prints the number of lines in which the string occurred following the last line scanned.

Character string, string2, enclosed by identical characters, each represented by d2, is the replacing string. String2 may be omitted, or may be longer or shorter than string1. When the replacement is

made, the characters of string2, if any, replace the characters of string1 and the length of the resulting line is adjusted as necessary. When there are no characters entered for string2, the characters of string1 are deleted. Character d2 may be any character that does not appear in string2, V, or P.

When no other parameter is entered following string2, the specified lines are scanned and string2 replaces the first appearance on each line of string1, or label string1, each time a match is found. The command prints the number of lines in which the replacement was made after scanning the last line.

Either V or P, or both, may be entered following string2. The verify operation, specified by V, prints the line in which the match is found, and prints the question Y/N? on the next line. The user must enter Y or N followed by a carriage return to continue the operation. When the user enters Y the replacement is made. When the user enters N the replacement is not made. The scan continues in either case.

The print operation is specified by P. After the replacement is made, the resulting statement is printed and the scan continues.

When the specified lines have been scanned, TXEDIT prints the number of lines in which a match was found. The pointer is left unchanged throughout the operation.

The general rule of TXEDIT which allows spaces between characters or operands does not apply to string1 and string2. Any spaces between the characters represented by d1 are considered part of string1, and any sapces between the characters represented by d2 are considered part of string 2. Lines brought into memory have trailing balnks suppressed and therefore comparisons should not be made past the last non-blank character of a line.

The following example shows an F command to replace the first appearance in each line of the string EUEN with the string EVEN in lines 34 through 48 and print the resulting lines:

```
?F34-48F*EUEN*$EVEN$P
```

The command in the following example verifies the replacement of label P1 with string PUN1 in each of nine lines beginning with the pointer line:

```
?F9L'P1''PUN1'V
```

4.4.8 PRINT COMMANDS. The print commands cause TXEDIT to print the first and last lines in the buffer, or to print one or more specified lines. The print commands are described in the following paragraphs.

4.4.8.1 Limits (L). The Limits (L) command causes TXEDIT to print the first and last lines in the buffer, including the line number, if any, with the right margin at column 72. The SN and SP commands do not affect the operation of the L command. The syntax for the L command is as follows:

```
L
```

The L command is used to identify the top and bottom lines of the buffer.

4.4.8.2 Print (P). The Print command causes TXEDIT to print a block of lines. The block of lines is specified by first and last line numbers, or by a number of lines preceding or following the pointer. The SL and SN commands, when entered, control printing of line numbers, and the SP command, when entered, sets the right margin of the print lines. When these commands are not entered, line numbers are printed and the right margin is column 72. The syntax of the P command is as follows.

```
        <s> - <t>
P       [+] [<n>]
        - <n>
```

Line s through line t are printed, or n lines are printed. When n is preceded by a minus sign, n lines preceding the pointer line, but not the pointer line, are printed. When n is unsigned or preceded by a plus sign, n lines beginning with the pointer line are printed. When no operand is entered, the pointer line is printed. The following example shows a P command to print lines 8 through 10:

```
?P8-10
```

The command in the following example prints the pointer line and the next three lines:

```
?P4
```

The user may terminate the Print command at any time by entering an ESC character at the keyboard. TXEDIT then prints a question mark and awaits input of another command.

4.4.9 OUTPUT COMMANDS. TXEDIT provides two commands to write source or object code and one command to end execution of TXEDIT. The Keep (K) command writes the entire buffer or specified lines from the buffer. The Quit (Q) command writes specified lines from the buffer, the entire buffer, or the buffer contents and the remainder of the input file, and writes an end-of-file record on the output file. The output commands are described in the following paragraphs.

4.4.9.1 Keep (K). The Keep (K) command writes a specified number of lines from the buffer to the output device. The syntax for the K command is as follows:

```
K[<n>]
```

The first n lines of the buffer, or all lines in the buffer when n is omitted, are written on the output file. When the pointer line is written, the pointer is moved to the top line remaining in the buffer. The K command is entered to write lines no longer required in the buffer in order to have space in the buffer for additional lines. The following example shows a K command to write the top 15 lines of the buffer:

```
?K15
```

4.4.9.2 Quit (Q). The Quit (Q) command writes lines from the buffer and input file followed by an end-of-file record. The syntax of the Q command is as follows:

```
Q[<s>]
```

The lines of the input file up to and including line number s are written. When line number s is in the buffer, lines are written from the buffer only. When line number s is not in the buffer, TXEDIT writes the lines in the buffer, reads the additional lines from the input file, and writes these lines. If line number s is never found, the rest of the file will be copied. When s is zero the edit is finished; no more data is written from the buffer or from the file, and an EOF character is inserted in the output file. The Q command is used to truncate data. When s is omitted, the lines in the buffer and the remainder of the input file are written. The Q command is entered to write the output file, or the remainder of the output file, including the end-of-file record. After the lines have been copied to the output file, the terminate sequence is entered.

4.4.9.3 **END (E).** Another command available to initiate the terminate sequence is the End command. This command ends the edit function without writing any data to the output file and does not cause the EOF character to be written. This provides an escape route from TXEDIT in the event a nonrecoverable error has been detected and there is no requirement to write an EOF on the output file. The system will respond to the E command with the TERMINATE/CONTINUE? prompt. The user must then enter a T to exit from the TXEDIT program and restart without affecting the current status of the input or scratch files.

4.4.10 **TERMINATE-SEQUENCE COMMANDS.** Two commands can be used to initiate a terminate sequence, depending on the particular situation. The normal method of terminating is with the Quit (Q) command, as explained above. The Q command always writes an EOF on the output file, and the system responds with the following message:

TERMINATE/CONTINUE?

In response to this message, the user enters a T or C. When it is desired to reverse the flow of the data and continue the editing, the Continue (C) command is entered. The system responds with the question mark (?) prompt and editing continues, starting at line one, again. When editing is completed and the T response is entered for terminate, the system responds with one of two messages as follows:

- TEXT IN INPUT FILE.

  END EDIT.

or

- TEXT IN SCRATCH FILE.

  TRANSFER TO INPUT?

The first message, "TEXT IN INPUT FILE", ends the TXEDIT and returns control to the TXDS Control Program. If the second message, "TEXT IN SCRATCH FILE", is printed and the user enters a Y for yes, the text is transferred from the scratch file back to the input file and control is returned to the TXDS Control Program. If an N for no is entered, the system prints "END EDIT" and without any additional action, returns control to the TXDS Control Program.

4.5 ERROR MESSAGES

The TXEDIT error messages capable of being presented on the system console by TXEDIT, their reason for occurring, and the procedure for recovery from each error is presented in table 4-3.

4.6 EXAMPLE: ENTERING A SOURCE PROGRAM ON A CASSETTE OR DISKETTE

The following paragraphs describe the use of TXEDIT to enter a new source program on a cassette or diskette. The Insert (I) command is used to input new source statements. Any of the commands may be used to correct any errors made in entering the statements. Because statements entered with the Insert command have no line numbers, the pointer-relative specification is the only available means of specifying a line in a command.

Table 4-3. TXEDIT Error Messages

Message	Reason	Recovery
INVALID OPERATOR	The operator portion of a command entry is incorrect.	Enter a valid, correct command.
INVALID OPERAND	The operand is not entered correctly or is beyond the range of values for that operand.	Enter a valid, correct command or enter another command.
BUFFER EMPTY	A command that operates on data in the buffer is entered before data has been placed in the buffer from the input file or from the keyboard (either initially or after writing the entire buffer contents).	Enter a D or I command and data.
BUFFER FULL	A D, I, or C command has attempted to put more data into the buffer than the buffer can contain.	Enter a K command or write data from the buffer before entering or reading more data.
END OF FILE	End-of-file has been encountered on input or a D command has attempted to read more records than is contained in the input file.	Enter a Q command. Another editing session may be entered by entering a C to the TERMINATE/CONTINUE? question. TXEDIT will make no further attempt to read the input file until the program restarts.
OFF THE TOP	A negative displacement caused the pointer to be moved past the beginning of buffer. (That is, the negative displacement from the pointer line in a C, M, R, F, or P command is greater than the number of lines in the buffer before the pointer line.)	After printing the message, TXEDIT is positioned at the top (i.e., first) line of the buffer.
LAST LINE	A positive displacement caused the pointer to be moved beyond the last line at the end of the memory buffer.	TXEDIT prints a question mark and waits for another command.
LINE NOT FOUND	A line, or line number, was referenced but was not in the buffer. The first line in a C, M, R, F, or P command, or the line number in an I command, or the destination line number in an M command is not in the buffer.	The command is not executed by TXEDIT. Enter another command in response to the question mark (?) prompt.
CAN'T GET MEMORY	Memory option was greater than available memory.	Enter a smaller memory option.
CAN'T GET COMMON	COMMON was not included at system generation time.	Execute the system generation utility and include COMMON.

Table 4-3. TXEDIT Error Messages (Continued)

Message	Reason	Recovery
nn – I/O ERROR (where nn refers to the error code listed in Appendix I, entitled "I/O Error Codes").	There was an I/O error.	Correct error and restart.
NAMES CANNOT BE THE SAME	The input file name entry and the output file name entry were the same.	Reenter INPUT: and OUTPUT: parameter.
ILLEGAL PARAMETER	The file name was not included in the INPUT: parameter.	Reenter the INPUT: parameter.
TRANSFER I/O ERROR	There was an I/O error transferring the output file to the input file.	Correct and use the TXCCAT utility to copy the output file to the input file.
I/O ERROR, RETRY? (Y, N, or CR to abort I/O)	If "Y" is entered, it will backspace one record and try to read that record again. If "N" is entered, it will try to read the next record. If a carriage return (CR) is entered, the text editor will terminate.	

The following text describes an example of writing a source program using TXEDIT.

The initial message and the first command, with associated entries, are as follows:

```
TXEDIT          936220**
 ?I0
W1      BSS     32
START   RSET
        LWPI    W1
        CLE     R0
```

The I command with an operand of zero causes TXEDIT to place the lines that follow at the top of the buffer. The buffer pointer is not moved as lines are entered and remains ahead of the first line entered. In the above example, an error was made in the operation field of the fourth line, so the user entered an additional carriage return to terminate the command, permitting entry of another command to correct the error.

The next part of the example program is:

```
?K3
?P1
        CLE     R0
```

The K command causes TXEDIT to write the first three lines on the output medium. The P1 command causes TXEDIT to print the pointer line to verify that the pointer is at the line that contains the error. An alternative to using the Keep command to write the correct portion of the program is to use a Down (D) command to position the pointer for correction of the error, leaving the first three lines in the buffer.

The next command and the associated entries are as follows:

```
?C
        CLR     R0
I1      INC     R0
        JNO     J1
D1      DEC     R0
        JNE     D1
        JMP     I1
        END     START
```

The C command deletes the error line and accepts seven lines of source code. The example source program is now complete, with three lines written on the output medium, and seven lines in the buffer.

The next command and the resultant printout or display follows:

```
?F10F'J1''I1'
LAST LINE
0001 FOUND
```

The F command scans the contents of the buffer, replacing the first appearance in each line of string J1 with string I1. The command attempts to scan 10 lines,and prints the message "LAST LINE" because there are only seven lines in the buffer. The V and P options (described above under the Find (F) command paragraph) could have been used. This is an alternate method of correcting an error in a source program entered from the keyboard using TXEDIT.

The next command and the resultant printout or display follows:

```
?P10
        CLR     R0
I1      INC     R0
        JN0     I1
D1      DEC     R0
        JNE     D1
        JMP     I1
        END     START
LAST LINE
```

The P command causes TXEDIT to print the contents of the buffer and the last line message. Entering the Quit command causes the terminate sequence to be entered.

```
?Q
TERMINATE/CONTINUE?
```

The Q command causes TXEDIT to write the buffer contents on the output medium following the records previously written by the Keep (K) command. An end-of-file record is written following the last record. The user then enters a T to terminate the text editor, and a Y to transfer the scratch file to the original input file.

```
TERMINATE/CONTINUE?T
TEXT IN SCRATCH FILE.
TRANSFER TO INPUT?Y
END EDIT
```

4.7 EXAMPLE OF HOW TO EDIT A SOURCE PROGRAM

The capabilities of TXEDIT to edit source programs include adding, moving, and removing statements, and replacing a character string in statements. The edited program may include portions of a number of source programs. The purpose of editing is to correct or modify a source program. The following paragraphs describe an example of editing a source program and considerations for editing source programs. For this example, a typical source program is used for which no Setup command is required because default values for print margin and F command limits are used and line numbers are printed.

The initialization messages and the first command are as follows:

```
TXEDIT  936220  **

?D117
```

The D command moves the pointer down 117 lines, and TXEDIT reads in the source file to fill the buffer as defined by the D command. A smaller value could have been used to read part of the file, followed by a subsequent D command to read the remainder. Had a larger value been entered, TXEDIT would have read the 117 records of the file and printed the end-of-file message. TXEDIT prints the prompt character (?) and awaits another command.

The next command and printout result are as follows:

```
 ?L
0001    TITL 'EDITING EXAMPLE'
0117    END
```

The L command verifies the buffer contents by printing the first and last lines in the buffer. Had the SN and SP commands been entered, they would not have affected the printing of the limits resulting from the L command.

The next command is as follows:

```
?T
```

The T command moves the pointer to the top of the buffer (line 1) from line 117 where the first command had placed the pointer. Moving the pointer to the top of the buffer permits using pointer-relative commands for the area at the top of the buffer.

The following commands move line 46 to a position after line 116 and remove line 117.

```
?M46-46,116
?R117-117
```

The following command is entered.

```
?M81-87,115
```

This M command moves lines 81-87 to a position following line 115 to cause the line numbers in the buffer to be out of sequence.

The following commands prepare the move operation for verifying.

```
?B
?P1
0046    END     START
```

The B command places the pointer on the last line of the buffer, and the P command prints the pointer line to verify that it is on the proper line.

The next command and the resultant printout or display on the system console are presented as follows:

```
 ?P-13
0111    UP2     MOV     *R10,*R10
0112            JNE     UP1
0113            BL      @ATTOP
0114            MOV     *DUMNXT,TMLOC
0115            JMP     UP3
0081    *ROUTINE COMMON TO UP AND DOWN
0082    UDCOM1 MOV      RTN,R5
0083            BL      @SCANOP
0084            INC     UDCNT
0085            MOV     UDCNT,UDCNT
0086            JEQ     EXIT
0087            B       *R5
0116    *
```

The P command prints the 13 lines preceding the pointer line, and the result shows that lines 81-87 have been placed after line 115. This result also shows the effect of the previous move and remove commands.

The next command and associated entries are as follows:

```
 ?I77
*TITLE  =   MSGOUT      MESSAGE OUTPUT
MSGOUT      MOV         *R11,*R10
            MOV         @MCOUNT(R10),R10
            BLWP        @PRINT
            B           *R11
```

The I command inserts five lines following line 77. The number of lines inserted is the number of lines entered with the command and may be one or more lines. After the carriage return that terminates the last line, an additional carriage return is entered to terminate the command.

The next command and resultant printout are as follows:

```
 ?P77-78
0077                JMP     EXIT
        *TITLE=     MSGOUT  MESSOUT OUTPUT
        MSGOUT      MOV     *R11,*R10
                    MOV     @MCOUNT(R10),R10
                    BLWP    @PRINT
                    B       *R11
0078    EOFEXT      BL      @MSGOUT
```

The P command prints lines 77 through 78, which includes the five unnumbered lines inserted by the previous command. The result shows that the lines have been inserted correctly.

The next command and the resultant interaction are as follows:

```
 ?F1-46F'EXIT''EXTDWN'VP
0071                JMP     EXIT
Y/N?  Y
0071                JMP     EXTDWN
0077                JMP     EXIT
Y/N?  Y
0077                JMP     EXTDWN
0080      EXIT      RTWP
Y/N?  Y
0080      EXTDWN    RTWP
0086                JEQ     EXIT
Y/N?  N
0004  FOUND
```

The F command finds the first appearance in a line of the string EXIT in lines 1 through 46. (Remember that line 46 is now the last line, i.e., after line 116.) The entire buffer is scanned because the top line in the buffer is line 1 and the bottom line is line 46. Line numbers greater than 46 between lines 1 and 46 are also scanned. The replacing string is used only when the user enters a Y following the printing of the line found. In the example shown, the replacement was not made in line 86 because the user entered an N following the printing of this line. Lines 71, 77 and 80 were replaced because the user entered a Y following the printing of these lines. The count of lines found is printed after all lines have been scanned. The F command may be used to scan only a portion of the buffer, from one line up to the entire buffer, and replace from one character to the entire statement.

The next three commands are as follows:

```
?R15-15
?R17-17
?R19-19
```

Each R command removes the specified line from the buffer. Three commands that remove one line each are necessary because the lines to be removed are not consecutive. A single R command may remove one or more consecutive lines.

The next command and the resultant printout are as follows:

```
 ?P14-20
0014    DUMNXT    EQU    0
0016    LINAD     EQU    2
0018    LINPTR    EQU    4
0020    CLLOC     EQU    6
```

The P command prints lines 14 through 20. The result shows that the lines specified in the Remove (R) command were removed.

The next command is as follows:

```
?U2
```

The U command positions the pointer to the second line preceding the pointer *line*. The pointer could have been moved any number of lines up to the top of the buffer.

The next two commands, the resultant printout of the first command, and the entry associated with the second are as follows:

```
?P68-68

0068                A       @MAXLIN,UDCNT
?C68-68
          A         @MINLIN,UDCNT
```

The P command prints line number 68 to verify that line 68 is the desired line. The C command changes line 68 to the line entered with the command. One or more consecutive lines may be deleted by a C command, and any number of lines, including zero lines, may be added. The number of lines added does not have to be equal to the number of lines deleted and the added lines have no line numbers.

If there were no more data in the input file, or if the remaining data were to be discarded, the next command would be ?Q117. This would place the data in the buffer in the SCRATCH file and place an EOF at the end of the data. If, however, there were more data in the input file but no editing was required, the next command would be just ?Q. The Q command would write the entire buffer, plus whatever was remaining in the input file would be placed into the scratch file and EOF would be placed at the end.

The next response from the system would be as follows:

```
TERMINATE/CONTINUE?
```

If no further editing is desired, the operator enters a T for terminate. The system would respond as follows:

```
TEXT IN SCRATCH FILE.
TRANSFER TO INPUT?
```

At this point the operator decides where the text is to reside. If a Y is entered for Yes, the scratch file text is transferred directly to the input file, or if N is entered, the text remains in the OUTPUT file. The system ends the TXEDIT by printing END EDIT and the TXDS control program is reactivated.

4.8 EXAMPLE OF HOW TO EDIT AN OBJECT PROGRAM

The capabilities of TXEDIT to edit object programs include adding, moving, and removing records, and replacing a character string in records. These capabilities allow the user to combine object code, correct object code, and add object code at the machine instruction level. In editing object code, it is necessary to thoroughly understand the object code format and the significance of tag characters (described in the Model 990 Computer/TMS9900 Microprocessor Assembly Language Programmer's Guide). Records may be inserted into an object program at any point except that the records that contain tag character 3 or 4, tag character 5 or 6, and tag character 1 or 2 must follow all other records in the object file. Further, the record that contains tag character D, if any, must precede the record that contains the first tag character 0. Each record must end with tag character F. When the contents of a record are altered, tag character 7 and associated field must be removed.

When the length of relocatable code is increased, the contents of the hexadecimal field associated with the final 0 tag character must be changed. The following paragraph describes an example of editing an object program.

NOTE

Compressed object code cannot be edited.

In the example, the purpose of the edit is to add a record to specify a load point, to change instructions that use workspace register 1 to use workspace register 7 instead, to change an instruction, and to add an instruction.

The initialization message and the first command are as follows:

```
?SN
```

The SN command is a setup command that inhibits printing of line numbers. When line numbers are printed, printing of an object record may be truncated because of the length of the print line.

The next command and the associated entry are as follows:

```
?I0
D1000F
```

The I command with an operand of 0 inserts the associated line at the top of the buffer. The line will be the first record in the edited object file, and contains load point of 1000_{16}, specified with a D tag character.

The next command and the resultant printout are as follows:

```
?D10

END OF FILE
```

The D command causes TXEDIT to read in the object file to be edited. The file contains six records, so the operand used causes TXEDIT to attempt to read past the end-of-file record. This inhibits further reading of any input file in this run of TXEDIT.

The next command and the resultant printout are as follows:

```
?L

        D1000F
0006    200CE10010C     7FCABF
```

The L command causes TXEDIT to print the limits. The top line in the buffer is the line entered with the I command, and has no line number. The bottom line is the last line of the object file, line 6.

The next command and the resulting interaction are as follows:

```
?F1-6F'B0002''B000E'VP

00000SAMPROG 90040C0000A0020BC06DB000290042C0020A0024BC81BC002A7F219F
Y/N?Y

00000SAMPROG 90040C0000A0020BC06DB000E90042C0020A0024BC81BC002A7F219F
A0028B0241B0000BCB41B0002B0380A00CAC0052C00A2B02E0C0032B0200B0F0F7F1DEF
Y/N?Y

A0028B0241B0000BCB41B000EB0380A00CAC0052C00A2B02E0C0032B0200B0F0F7F1DEF
0002 FOUND
```

The F command scans for the character string B0002 with the verify and print options. The replacement string, B000E, changes the memory address of workspace register 1 to that of workspace register 7 in two instructions. Verification and printing provides control and documentation of the changes.

The next command and the resulting interaction are as follows:

```
?F1-6F'7F151'''VP

A00D6BC0A0C00CAB04C3BC160C00CCBC1A0C00D0BC1F2B0287B3A00A00ECB02217F151F
Y/N?Y

A00D6DC0A0C00CAB04C3BC160C00CCBC1A0C00D0DC1F2B0287B3A00A00EB0221F
0001 FOUND
```

The F command scans for the character string 7F151, which is a checksum tag character and associated field. The replacement character string is a null string, and the result is to remove the checksum from a record which was changed by an edit command not shown here.

The next command and the associated entry are as follows:

```
 ?I
A00ECB0227A00F0B06C7A010AB04CTF
A010CB10FFF
```

The I command inserts the associated two lines following the line on which the pointer had been positioned by an edit command not shown. The first line will cause the loader to overlay three words of the original file, which is another way of changing object code. The second line is an added instruction which will increase the size of the program module.

The next three commands, the resultant printout of the second, and the associated entry of the third are as follows:

```
?D3

?P1

200CE0010C      7FCABF
 ?C
200CE0010E      F
```

The D command moves the pointer line down three lines, and the P command causes TXEDIT to print the pointer line to verify the pointer position. The C command changes the pointer line to modify the number of words of relocatable code in the program. If this is not done, and another module is loaded following this module without specifying a load address for the subsequent module, the subsequent module will overlay the instruction that was added. The pointer line is also changed to delete the checksum.

The last command and the final messages are as follows:

```
 ?Q
TERMINATE/CONTINUE ?T
```

The Q command causes TXEDIT to write the contents of the buffer and any data remaining in the input file, followed by an end-of-file record, on the output medium. The T command causes TXEDIT to terminate and issue the message:

```
TEXT IN SCRATCH FILE.
TRANSFER TO INPUT?
```

If all editing is completed, the user responds with a Y (yes) and TXEDIT transfers the scratch file to the input file and prints "END EDIT".

SECTION V

ASSEMBLING SOURCE PROGRAMS

5.1 INTRODUCTION

The TXMIRA Utility Program is a member of a family of assemblers that may be used with the Model 990 Computer family. It functions to substitute absolute operation codes and addresses (i.e. Model 990 machine language) for symbolic codes and addresses (i.e. assembly language source code programs). TXMIRA provides for the allocation of storage to the minimum extent of assigning storage locations to successive instructions and for the computation of relocatable addresses from symbolic addresses. The TXMIRA program has the following features:

- Assembles all 72 instructions for both the Model 990/4 and the Model 990/10 Computers
- Supports many assembler directives
- Supports program, data, and common segmentation
- Supports both pseudo instructions (NOP,RT)
- Supports a sorted symbol list option
- Provides error messages in text form
- Supports compressed object code
- Prints or truncates 'TEXT' string option
- Assembles FORTRAN compiler source output

As a two-pass assembler program, TXMIRA reads the program source statements two times, providing maximum programming flexibility in the process of producing object code. On the first pass, the assembler maintains the location counter and builds a symbol table similar to those in a one-pass assembler. During this pass some errors may be detected and printed on the listing device. For the second pass, the source statements are read in again by rewinding the input file. During the second pass, the assembler generates the object code using the symbol table developed during the first pass. The two pass feature reduces the restrictions on forward referencing. TXMIRA produces a listing of the source code and the object code (i.e., machine language). Optionally, the assembler prints out the symbol table. Further, the resultant output produced by the TXMIRA utility program may be linked to other output modules or be loaded separately for execution.

For more details on the Model 990 assembly language, refer to *Model 990 TMS9900 Microprocessor Assembly Language Programmer's Guide,* part number 943441-9701.

5.2 LUNOs AND THEIR USES

LUNOs 5, 6, and 7 are used by TXMIRA program for source input, object output, and listing, respectively. All LUNOs are assigned by the TXMIRA program. Upon termination of the TXMIRA program, all LUNOs are released.

5.3 OPERATION INTERACTION

The TXMIRA program can only run under the control of the TXDS Control Program. The INPUT: parameter must have the pathname of the source file to be assembled. The first OUTPUT: parameter must have the pathname of the file or device to which the object code will be written. The second OUTPUT: parameter must have the pathname to which the listing will be written. The object pathname and the listing pathname must be separated by a comma. If the output file does not exist, it will be created as a sequential file with the name given. If the listing pathname is null, the system default printer will be used. If only part of the listing pathname is used, the defaults in table 5-1 will be used. The following is an example of loading and executing TXMIRA using the TXDS Control Program.

```
PROGRAM:   :TXMIRA/SYS
   INPUT:  :SOURCE
  OUTPUT:  :OBJECT,LP
 OPTIONS:  SLM4000
```

5.4 TXMIRA OPTIONS

The TXMIRA assembler options are specified by a single alphabetic character followed in one case, M, by a numeric field. Input format is free-form in that delimiters (i.e. separators) may be commas, blanks or no delimiters. The options recognized by TXMIRA are listed and described in table 5-2. These options are described further in the following subparagraphs.

Table 5-1. Pathname Defaults

Field	Source	Object	Listing
DEV	DEFAULT DISC NAME	DEFAULT DISC NAME	DEFAULT DISC NAME
FILE	NONE	SOURCE FILE	SOURCE FILE
EXT	SRC	OBJ	LST

Table 5-2. TXMIRA Options

Option	Description
Mnnnnn	Overrides memory size default; default is 2400 bytes
X	Produce cross-reference
L	Produce assembly listing
T	Expand TEXT code on listing
S	Produce sorted symbol list
C	Produce compressed object output
R	Predefine registers

where n is a decimal digit

5.4.1 **MEMORY OPTION (M).** The memory option is used to override the default memory size. The size is expressed in bytes. The syntax of the option is as follows:

Mn (where n is a decimal number up to five decimal digits)

Some examples follow:

M4096
M20000
M01000

5.4.2 **CROSS-REFERENCE OPTION (X).** This option is used when a cross-reference is desired. Upon termination of TXMIRA, the TXDS Control Program will chain to the TXXREF Utility Program (described in Section VI) to perform the cross-reference operations. To enable the cross-reference option to work properly, the TXXREF object code must be in a file with the following pathname:

:TXXREF/SYS

5.4.3 **LISTING OPTION (L).** This option is used when a listing is desired by the user. It may be overriden by the LIST and UNL assembler directives. Errors are always printed.

5.4.4 **PRINT TEXT OPTION (T).** This option is used when expansion of TEXT statements is desired by the user. Default results in no expansion of TEXT statements.

5.4.5 **SYMBOL TABLE LISTING OPTIONS(S).** This option is used when a sorted symbol list output is desired. The list presents four symbols to a line; and each symbol presents the following information in sequence: (1) attribute tag; (2) symbol; and (3) value. Table 5-3 defines the symbols used in the listing.

5.4.6 **COMPRESSED OBJECT OPTION (C).** This option is used when compressed object code is desired, and it may only be written to a diskette file. Compressed object takes up less diskette space. See Appendix B for a description of compressed object.

5.4.7 **PREDEFINE REGISTERS OPTION (R).** This option is used to predefine the register symbols used in a source program, equating the symbol R0 with workspace register 0, R1 with register 1, R2 with register 2, . . . R15 with register 15.

Table 5-3. Symbol Attributes

Character	Meaning
"	Data Relocatable
+	Common Relocatable
'	Program Relocatable
E	External Reference (REF)
D	External Definition (DEF)
X	Extended Operation (XOP)
U	Undefined
S	Secondary Reference (SREF)

5.5 ERRORS

5.5.1 TXMIRA ERROR MESSAGES. The TXMIRA assembler processes fatal errors (table 5-4) and nonfatal errors (table 5-5). The fatal errors cause the run to abort with the appropriate error message printed.

The nonfatal errors do not cause the run to abort. An error message is printed following the statement containing the error. The format of the printout is as follows:

***** SYNTAX ERROR – RCD nnnn

where nnnn is the source record number.

When there are undefined symbols in an assembly, the undefined symbols are listed at the end of the assembly listing under the following heading:

THE FOLLOWING SYMBOLS ARE UNDEFINED:

Table 5-4. TXMIRA Fata Errors

Error	Description	Recovery
CANT GET COMMON	COMMON not in system	Re-Gen system with COMMON
CANT GET MEMORY	MEMORY size requested too large	Decrease request
SYMBOL TABLE OVERFLOW	MEMORY size too small	Increase request
NO END CARD FOUND	END directive missing	Add directive and reassemble
nn-ILLEGAL PATHNAME	PATHNAME syntax is incorrect	Correct pathname and retry
nn-I/O ERROR-A	I/O ERROR on A, where A can be: S=SOURCE O=OBJECT L=LISTING	Correct and retry
nn-ASSIGN ERROR	Error in Assign or Open	Correct pathname and retry
nn-CLOSE ERROR	Error in Close	Correct pathname and retry

Note: nn is a system returned error code. See Appendix D for explanation.

Table 5-5. TXMIRA Nonfatal Errors

***** SYNTAX ERROR – RCD nnnn
***** ILLEGAL EXTERNAL REF. – RCD nnnn
***** VALUE TRUNCATION – RCD nnnn
***** MULTIPLY DEFINED SYM – RCD nnnn
***** INVALID OPERATOR – RCD nnnn
***** ILLEGAL FORWARD REF. RCD nnnn
***** ILLEGAL TERM – RCD nnnn
***** ILLEGAL REGISTER – RCD nnnn
***** SYMBOL TRUNCATION – RCD nnnn
***** UNDEFINED SYMBOL – RCD nnnn
***** COMMON TABLE OVERFLOW – RCD nnnn
***** PEND ASSUMED – RCD nnnn
***** DEND ASSUMED – RCD nnnn
***** CEND ASSUMED – RCD nnnn

where nnnn is the record number in which the error occurred

5.6 TXMIRA EXAMPLE

Following is an example of loading and executing TXMIRA. The diskette file :TXTST1/SRC is entered in the INPUT: source file parameter line. DSC2: is entered in the first OUTPUT: parameter causing TXMIRA defaults for the file name and extension. Therefore, the object (machine) code is written to the diskette file DSC2:TXTST1/OBJ. LOG is entered in the second output parameter producing the source listing output on the system console. Two options are entered in the OPTIONS: parameter line. The L option produces a source listing, and the S option produces a symbol table.

```
TXDS  936215 ◆M      1/ 0    00:01

PROGRAM: :TXMIRA/SYS
  INPUT: :TXTST1/SRC
 OUTPUT: DSC2:,LOG
OPTIONS: L,S
TXMIRA  936227 ◆◆
```

Following is an example of the source listing caused by entering an L in the OPTIONS: parameter line.

```
TXTST1         TXMIRA  936227 **                                  PAGE 0001
TXDS TEST PROGRAM  937808-9901**

  0002                      IDT  'TXTST1'
  0003              *
  0004                      REF  CNT, NEW, WRITE
  0005              *
  0006 0000 0006'           DATA TSTWSP, START, 0
       0002 0045'
       0004 0000
  0007 0006         TSTWSP  BSS  32
  0008 0026 1600    ENDPRG  DATA >1600              END OF PROGRAM OP CODE
  0009              *
  0010                      DXOP SVC, 15            *** DEFINE XOP
  0011              *
  0012 0028   0D    OLD     BYTE >0D, >0A, >0A
       0029   0A
       002A   0A
  0013 002B   20            TEXT ' OLD MESSAGE -- WRONG !!'
  0014 0043   0D            BYTE >0D, >0A
       0044   0A
  0015       001D  CNT1     EQU  $-OLD

TXTST1         TXMIRA  936227 **                                  PAGE 0002
TXDS TEST PROGRAM  937808-9901**

  0017              START
  0018 0046 0420            BLWP @WRITE             PRINT MESSAGE
       0048 0000
  0019 004A 0028'           DATA OLD, CNT1
       004C 001D
  0020 004E 2FE0            SVC  @ENDPRG            END OF PROGRAM
       0050 0026'
  0021                      END
```

Following is an example of the sorted symbol table caused by entering an S in the OPTIONS: parameter line.

```
TXTST1         TXMIRA  936227 *A                                  PAGE 0001
TXDS TEST PROGRAM  937808-9901**

  E CNT      0000       CNT1     001D     ' ENDPRG  0026     E NEW      0000
  ' OLD      0028     ' START    0045     X SVC      000F     ' TSTWSP   0006
  E WRITE    0048

 0000 ERRORS
```

SECTION VI

TXDS CROSS-REFERENCE (TXXREF) UTILITY PROGRAM

6.1 INTRODUCTION

The TX990 Cross Reference (TXXREF) Program is a single pass cross-reference program. The program gives a listing of each user-defined symbol in a 990 assembly source program along with the line numbers on which the symbol is defined and all of the line numbers on which the symbol is referenced. The line numbers of the references to a symbol are in ascending order, and the symbols are in alphabetical order. If the symbol was never defined, only the line numbers of the references to the label will be listed.

6.2 LUNOs

LUNOs 5 and 6 are used by TXXREF. They are assigned when execution begins and released upon termination by the program. LUNO 5 is the source input LUNO, and LUNO 6 is the listing LUNO.

6.3 OPERATING PROCEDURE

TXXREF can only run under the control of TXDS. The object program may be loaded from a device or from the file, :TXXREF/SYS.

The INPUT: parameter must contain the pathname of a source program. The OUTPUT: parameter must contain the pathname of a listing device to which the cross-reference listing will be directed. If there is no response to the OUTPUT: prompt, the default print device will be used.

The pathname defaults are given in table 6-1. The input file must preexist, and if the output file does not exist, it is created with the name given. Lastly, an option may be entered to override the symbol table size.

6.4 LISTING FORMAT

An example of a listing is shown in figure 6-1. The heading gives the name and version of TXXREF and the time and date of the run, if the time and date are initialized. Each line of the cross-reference begins with the symbol, listed alphabetically, followed by the line number on which it was defined (appearance in the label field), if any, and the list of line numbers, in ascending order, on which the symbol was referenced, if any. The last line gives the number of symbols in the cross-reference.

NOTE

If TXXREF runs out of table space, it prints the references found at that point, and attempts to continue. If insufficient space was freed up by that process, then TXXREF terminates.

Table 6-1. Pathname Defaults

Field	Input	Output
DEV	SYSTEM DISC	SYSTEM DISC
FILE	NONE	INPUT FILE
EXT	SRC	LST

```
TXXREF 937542 **     10:32:36     088/77     PAGE 0001

ASREXT               0175  0176
ASRHAN               0167  0168
BAD       0283
BADINT               0243  0285
BUF1      0089       0149
BUF2      0105       0150
BUF3      0124       0151
BUFADH    0143       0064
BUFADL    0066       0064
BUFE      0152       0146
BUFF      0148       0146
BUFH      0147       0146
COMSIZ    0239       0238
CRTHAN               0229  0230
DFLDSC    0060       0059
DFLPTR    0061       0059
DNT       0039       0038
DNT2      0040       0038
DNTEND    0057       0038
FMPBUF    0155       0154
FPYDCD               0274  0275
FPYDSR               0197  0198  0213  0214
FPYINT               0200  0201  0216  0217
FPYSPR               0193  0194  0209  0210
FREQX2    0004       0003  0005
GO                   0170  0171  0266  0267
GO913                0232  0233
IDL913               0225  0226
ILLSVC               0021
KBIDLE               0163  0164
KBTAB     0027       0026
KSB1      0033       0029  0224  0251
KYBIN                0250  0251
KYBUT     0028       0026  0031  0032
LDTSTR    0065       0064
LEV3      0246       0290
LEV4      0254       0291
LEV6      0262       0292
LEV7      0270       0293
LOC3      0249       0247
LOC4      0257       0255
LOC6      0265       0263
LOC7      0273       0271
LPHAN                0184  0185
LPINT                0187  0188  0258  0259
LPSPUR               0180  0181
LVLPTR    0013       0012
```

Figure 6-1. Sample Cross Reference Listing (Abbreviated)
Sheet 1 of 2

```
MAXST2  0032        0026
MAXSTA  0031        0026
PDT0    0160        0041   0043   0045   0078   0083   0267
PDT3    0177        0047   0160   0259
PDT4    0190        0049   0068   0177   0276
PDT5    0206        0051   0073   0190   0277
PDT6    0222        0053   0206
PDTSTR  0159        0158
```

```
TXXREF 937542 **     10:32:36     088/77     PAGE 0002

PWRFLG  0010        0009
RET     0284        0248   0256   0264   0272
SLICE   0006        0003
STINTR  0005        0003
TASKCM  0240        0238
TRABAD  0285        0283
TRAPRT              0243   0284
TRPINT  0289        0288
TXSTR3              0007   0008
UMXSVC  0023        0020
USCTAB  0022        0020   0023
WSP3    0244        0290
WSP4    0252        0291
WSP6    0260        0292
WSP7    0268        0293
XWP4    0276        0192   0208   0275
XWS                 0243   0285

THERE ARE 0072 SYMBOLS
```

Figure 6-1. Sample Cross Reference Listing (Abbreviated)
Sheet 2 of 2

6.5 OPTIONS

The only option that is recognized by TXXREF is the memory-size-override option. The size is given in bytes. The option is as follows:

Mnnnnn where n is a decimal digit.

The following are examples:

M4096

M00200

M2000

The default memory size is 4800 bytes. The memory block is used to build a symbol table. Therefore, the size must be at least 12 times the number of symbols in the source program, plus 4 times the number of references.

6.6 ERROR MESSAGES

The errors, descriptions and recovery for TXXREF are listed in table 6-2.

Table 6-2. Error Messages

Error	Description	Recovery
CANT GET COMMON	System COMMON not in system.	Regenerate system with COMMON
CANT GET MEMORY	Memory request too large	Decrease size
nn - ILLEGAL PATHNAME	Pathname doesn't exist or open error	Correct name and retry
nn - I/O ERROR	Error on read or write	Retry
INSUFFICIENT MEMORY - ABORT	Symbol table exceeded memory	Increase memory request
nn - ASSIGN ERROR	Error on Assign or Open	Correct pathname and retry
nn - CLOSE ERROR	Error on Close	Correct pathname and retry

SECTION VII

LINKING OBJECT MODULES

7.1 INTRODUCTION

In order to link separate object modules together to form a complete program, the user must execute either the Link Editor or the TXDS Linking Utility Program (TXLINK). For linking FORTRAN programs or tasks and procedures, the Link Editor must be used; TXLINK does not support FORTRAN or procedures.

The output of both the Link Editor and TXLINK is a file with a linked object module. The Link Editor can also create program files (see Section I of the *TX990 Operating System Programmer's Guide*) as output.

The Link Editor is discussed in the *Model 990 Computer Link Editor Reference Manual,* part number 949617-9701. TXLINK is discussed in the following paragraphs.

TXLINK accepts standard Model 990 object code modules (described in the *Model 990 Computer TMS 9900 Microprocessor Assembly Language Programmer's Guide,* part number 943441-9701, and compressed object code, available as an option (illustrated in table 6-3) and links the modules according to command information supplied by the user and the linking information in the modules. The linked output module is written on the output file.

Linking allows a set of independently assembled object modules to be linked to form a single object module. The major linking function is the resolution of external references and definitions in the individual unlinked modules.

TXLINK also supports partial linking of modules. A partially linked module may be used as input to another run of TXLINK with additional modules that satisfy the unresolved references.

The following restrictions apply:

- Linking of modules having absolute original addresses (AORG directive) is not supported.
- There must be enough memory for all symbols, (12 bytes/symbol), IDTs (24 bytes/IDT) and twice the length of the longest module to be linked. Memory size is defined by the "M" option.
- TXLINK only recognizes object tags "0" through "F".
- TXLINK does not link FORTRAN programs or tasks with procedures.

7.2 TXLINK FILE STRUCTURES AND LUNO ASSIGNMENTS

Figure 7-1 shows the relationship of the files accessed by TXLINK. Control information and file access names are passed by the Terminal Executive Development System (TXDS) via system COMMON.

TXLINK supports up to three object input files and two output files. Each input file can contain any number of concatenated object modules. Input LUNOs used are 10_{16}, 11_{16} and 12_{16}. The output of TXLINK consists of a linked object file and load map listing. Output LUNOs used are 7 for the object file and 6 for the load map listing.

7.3 TXLINK EXECUTION

TXLINK can only be executed under TXDS Control Program. The INPUT: parameter contains the pathnames of one to three input files. Each pathname must be separated by a comma. An input file may contain several object modules concatenated together. There may only be one end-of-file on each file, and the file must be a sequential file or device. All input files are rewound by TXLINK before they are used.

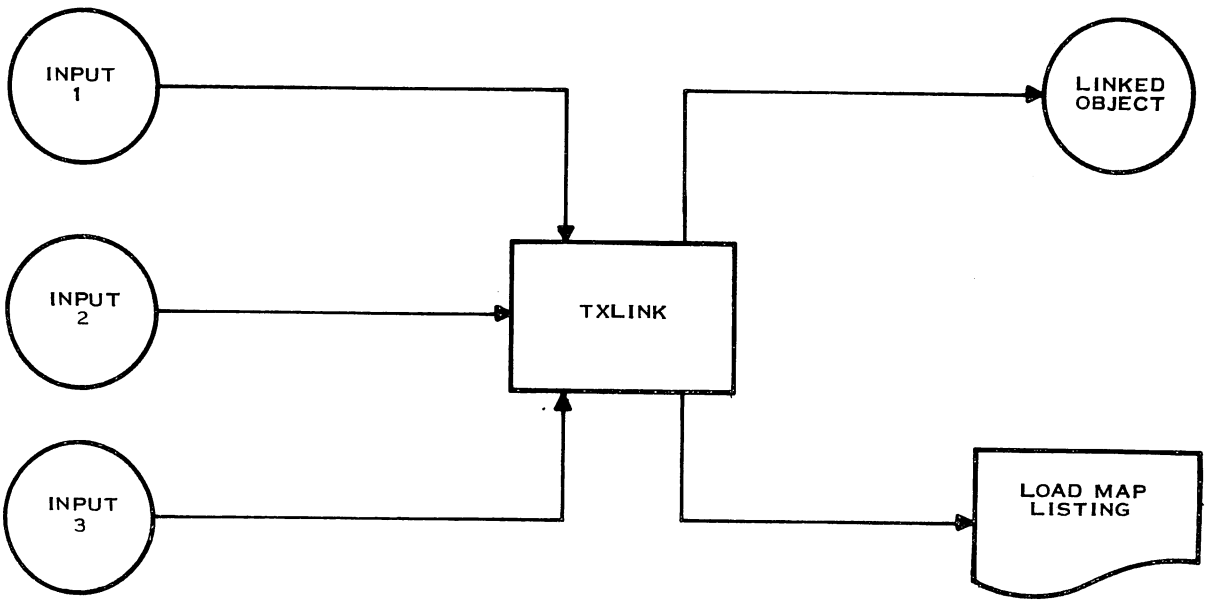


Figure 7-1. Files Accessed by TXLINK

The OUTPUT: parameter may contain two pathnames separated by a comma. The first pathname indicates the file to which the linked object code is written. The second pathname indicates the file to which the load map is listed. If either of the output files does not exist, TXLINK creates a sequential file with the pathnames entered. If the second output pathname is null, the system default printer is used. If only part of the pathname is defaulted, table 7-1 applies.

The defaults for the input and output pathnames are listed in table 7-1. Finally, the options are entered.

Table 7-1. Pathname Defaults

Field	Input	Output	Listing
DEV	DEFAULT DISC NAME	DEFAULT DISC NAME	DEFAULT DISC NAME
FILE	NONE	NONE	OUTPUT FILE
EXT	OBJ	OBJ	LST

7.4 TXLINK CONTROL OPTIONS

The following options control linking operations. All options are specified by a single alphabetic character followed in some cases by an override field. Input is free-form in that delimiters for options may be commas, blanks or no delimiters. The options are listed in table 7-2 and described in the following paragraphs.

7.4.1 **MEMORY OVERRIDE (M).** This option allows a larger block of memory for tables to be allocated to the Linker Utility. The default memory size is 11,800 bytes. The syntax for the option is:

Mnnnnn

Where n is a decimal number of the number of bytes required. There may be up to five decimal digits. The scan terminates when a nonnumeric character is encountered.

The following are examples of the memory option:

M4096
M00300
M24000

7.4.2 **COMPRESSED OBJECT (C).** The use of this object enables TXLINK to write compressed object code to the linked object file. TXLINK writes standard object code unless the C option is used. The syntax of the option is:

C

Compressed object format takes up less room on the diskette than standard 990 object code. The diskette is the only Floppy System device capable of supporting compressed object. The reader should be familiar with the Model 990 Computer object code format. If not, read the *Model 990 Computer TMS 9900 Microprocessor Assembly Language Programmer's Guide,* or refer to Appendix B.

Table 7-2. TXLINK Options

Option	Description
Mnnnnn	Override default memory size, default is 11800 bytes.
C	Compressed object output.
Iaaaaaaaa	IDT for linked object.
P	Partial link desired.
L	Print load map and symbol list.

Note: n is a decimal digit and a is an alphanumeric character.

7.4.3 **PROGRAM IDENTIFIER, IDT, OPTION (I).** This option enables the user to specify an object identifier for the linked object. Otherwise, the IDT of the first module input will be used. (An IDT is generated during assembly. It is the identification name of the module, and it is invoked by the "IDT" assembler directive.) The syntax of the option is:

Iaaaaaaaa

Where a is an alphanumeric character.

The scan terminates on a delimiter, blank or comma, or after eight characters.

Following are examples of the IDT option:

IMYLINK
IBADLINK
IWOW.

7.4.4 **PARTIAL OPTION (P).** The use of this option specifies that the module is to be partially linked. The partially linked module includes information for linking all unresolved references remaining in the module after the link. The partially linked object may be used in a subsequent linking operation to finish resolving the references. The syntax of the option is:

P

7.4.5 **LOAD MAP OPTION (L).** This option specifies that a load map listing is to be produced as shown in the example in figure 7-2. The two-line header of the load map listing identifies the version of TXLINK and shows the time and date of the run, provided the time and date have been initialized. The second line consists of the program name and its length.

```
TXLINK 937537 **    12:07:47    013/77     PAGE 0001
TXLINK           LENGTH 1308

        MODULE     LENGTH  ORIGIN     DATE       TIME

        TXLNKD      03C2    0000    01/12/77    18:03
        TXLNK1      041E    03C2    01/12/77    18:06
        TXLNK2      03AC    07E0    01/12/77    18:09
        TXLNK3      0466    0B8C    01/12/77    18:12
        TXLNK4      0316    0FF2    01/13/77    13:10

                        D E F I N I T I O N S

   A0       032E      ASGFLG   02EC      ASGIN    03BE      ASGLUN   02DF
   ASGPRB   02DC      ASGPTR   02F2      ASSERR   1262    A ATOCRT   0200
   BADTAG   0225      BINDEC   0D52      BINHEX   0D28      BINONE   032B
   BLANK    00DB      CBDA     0312      CBDA4    0316      CBDA5    0317
   CBDA6    0318      CBHA     031A      CBHA2    031C      CDALIM   0B98
   CDDLIM   0B9A      CHAB     030C      CHAB1    030D      CHAB2    030E
   CLOSE    02AC      CLROBJ   0C9E      CLSLUN   02AF      CLSOPC   02AE
   CMPDLA   0320      CMPDLD   0322      COMMA    032F      COMMON   038E
   COMPRS   03B4      CRLF     00FC      DATE     0026      DAY      00E8
 A DEV      0000      DEVSET   1154      DTLIST   034C      EF       032D
   EMSG1    0228      EMSG2    024E      EMSG3    027A      ENTRY    03C2
   EOL      00FD      EPOINT   03A8      EPTAG    03AA    A EXT      000D
 A FILE     0005      FLAGS    07E6      GETCM1   0303      GETCOM   0302
   GETDAT   0308      GETHEX   0790      GETMEM   0304      GETSVL   080E
   GETTAG   0764      HEAD1    00C6      HEAD2    00FE      HEDLEN   0115
   HOUR     00DC      IDTEND   0398      IDTLEN   0328      IDTSAV   00FE
   IDTTBL   0396      ILLCMN   0186      ILLHEX   01AA      ILLMEM   0198
   ILLPTH   01DC      ILLPUN   0202      ILLRED   01F2      ILLSUM   01CE
   ILLTAG   0212      ILLWRT   0202      INCMPR   03C0      INDFT    0356
   INLUN    0330      INLUN1   0331      INPNM    0026      INPNM1   0027
   INPTRS   0388      IOBF82   0076      IOBUF    0026      IOREG    0FF2
   LASTIN   0332    A LIBFLG   158D      LIMCHK   07D0      LNCNT    0392
   LODADD   03B2      LODPNT   0398      LOGERR   0E2E      LOGID    00C8
   LSTLUN   033C      MEMDFT   034A      MLEN     0394      MODCNT   03BC
   MODLEN   03A0      MULMOD   0B80    A NAMESZ   0010      NEWSYM   03AC
   OBJEOF   02D4      OBJIN    02D0      OBJLUN   02D3      OBJPTR   0B94
   OBJR55   00AD      OBJR60   00B2      OBJR80   00C6      OBJRC1   0077
   OBJRCD   0076    N ONE      032A      OPEN     02A0      OPNLUN   02A3
 A OPTCNT   0006    A OPTION   0070      OPTTBL   0344      OUTLUN   0334
 N OUTNM    003A      OUTNM1   003B    N OUTOBJ   0BB0      PAGCNT   0390
   PAGENO   00F8      PARTAL   03BA      PASS     039E      PASS2    04BC
   PGHDR    1012      PNTLOG   104C      PONTRS   093A      PRGLEN   03A2
 N PRGMEM   0CCE      PRINT    1042      PROOPT   108E      PUNCH    02B8
   PUTCOD   0D64      PUTMEM   0306      READ     0DA2      RECORD   0026
   REFDEF   0874      RELABS   03A6      RELES    0333      RELMEM   03A4
   REWIND   02C4      REWLUN   02C7    A SDFTPR   00A4      SEQBIN   03B6
   SEQDEC   00C2      SHEAD1   011C      SHEAD2   0150      SRCSYM   0832
   SVAL     07EC      SVN      032C      SYMBOL   07E2      SYMDMP   09D6
   SYMEND   039C      SYMLST   03B8      SYMOVR   01BC      SYMTAB   039A
   TBLEN    0926      TERM     0300    N TXLINK   0000    A TYPE     0004
   UCPDLA   0324      UCPDLD   0326      VALPTR   07E8      WP       0006
   WP1      0007      WRTOP    02BA
```

Figure 7-2. Load Map Listing

The heading is followed by a list of the modules, with the name (IDT) of each object module in the linked module, the length of each module, the origin of the module within the linked module and the date and time that each object module was generated.

The next section of the listing lists in alphabetical order the definitions in the modules. The symbols and corresponding hexadecimal values are printed four per line. The value is the definition within the linked module. When the listing shows an "N" preceding definition, the symbol was not referenced from another module.

When an "A" precedes a symbol, the symbol is self-defining (absolute). When the value is followed by a "U", the symbol is unresolved. When the value is followed by an "M", the symbol has been multiply defined, the first definition is the one that is used.

After all definitions have been listed and if there are multiply-defined modules, IDTs, or symbols or if there are unresolved symbols, a corresponding message is printed as presented in the examples in the next paragraph.

The load map is a useful tool during debug of an object module. After a task is loaded into memory, it has an absolute task origin, which is the absolute memory address of the task. If the user has the Operator Communication (OCP) Software Module in his system, he can calculate the task origin by using the (STATUS) command. By adding the absolute task origin to any of the values in the load map, the user can get the absolute memory address of that symbol.

The absolute memory address of a symbol that was not defined using a DEF directive within a module can be computed. First, add the relative address in the assembly listing (see Section IV, TX990 Assembler (TXMIRA)) to the relative origin value of the module's IDT found in the symbol map. Then add this sum to the absolute task origin in memory. The final sum is the absolute memory address of the symbol.

7.5 LINKED OBJECT MODULE

The linked object module produced by TXLINK consists of object code similar to that produced by the assemblers. Object code is described in the *Model 990 Computer TMS9900 Microprocessor Assembly Language Programmer's Guide.* As shown below, each module is terminated by a record beginning with a colon and containing the module name, date, time of linking and TXLINK identifier.

```
: MODULE  004/77    13:29:39   TXLINK
```

A fully-linked object module is ready to be loaded and executed by the operating system.

7.6 ERROR MESSAGES

Table 7-3 lists fatal error messages that are printed on the LOG when TXLINK encounters an error.

Table 7-3. Error Messages

Message	Description	Recovery
CANT GET COMMON	System COMMON not in system	Regen system with COMMON
CANT GET MEMORY	Memory requested is too large	Decrease size and retry
ILLEGAL HEX DATA	Nonhex digit found in input object	Reassemble module
MEMORY OVERFLOW	Tables exceeded available area	Increase size and retry
BAD CHECKSUM	Bad input object code	Reassemble module
nn–ILLEGAL PATHNAME	Pathname syntax error	Correct and retry
nn–READ ERROR	Error in reading object input	Check file and retry
nn–WRITE ERROR	Error in writing linked object or load map	Retry
ILLEGAL OBJECT TAG-A	Module contains absolute load or entry address or garbage. "A" is tag that was found.	Reassemble module
nn–ASSIGN ERROR	Error in assigning or opening file	Correct pathname and retry
nn–CLOSE ERROR	Error in closing a file	Correct pathname and retry

Note: The error code nn is returned from the system. See Error Appendix D for meaning.

7.7 TXLINK EXAMPLE

The following three object modules will be linked to form one object module by using TXLINK:

1. This is the contents of the object file entitled :TXTST1/OBJ

```
00052TXTST1   A0000C0006C0045B0000A0026B1600B0D0AB0A20B4F4CB44207F215F   0001
A0030B4D45B5353B4147B4520B2D2DB2057B524FB4E47B2021B210DB0A007F2B7F      0002
A0046B0420B0000C0028B001DB2FE0C00267F856F                               0003
40000CNT  40000NEW  30048WRITE  7F8A8F                                  0004
:  TXTST1    054/77    08:49:34        TXMIRA                           0005
```

2. This is the contents of the object file entitled :TXTST2/OBJ

```
0001DTXTST2   A0000B0D0AB0A48B4156B4520B4120B474FB4F44B2044B41597F1D0F  0001
B2021B2120B2020B2020B200DB0A007F984F                                    0002
6001DCNT      50000NEW   7FB3AF                                         0003
:          TXTST2          054/77   11:18:34   TXMIRA                  0004
```

3. This is the contents of the object file entitled :TXTST3/OBJ

```
00030TXTST3   A0000C0006C0026B0000B0000B0B00B0000B0000B0000B00007F287F  0001
A0012A0026BC0FEBC17EB2FE0C0006B03807F805F                               0002
50000WRITE 7FD29F                                                       0003
:        TXTST3      054/77     11:11:36     TXMIRA                    0004
```

The following example loads and executes TXLINK using the above three files as input parameters. The resultant file, :TXTST/OBJ, holds the linked object module.

```
PROGRAM:    :TXLINK/SYS
  INPUT:    :TXTST1/OBJ,:TXTST2/OBJ,:TXTST3/OBJ
 OUTPUT:    :TXTST/OBJ
OPTIONS:    ITXTST,M400,L
  TXLINK    937537**
```

Following is the link map, which is generated when an "L" is entered in the OPTION: parameter line:

```
TXLINK   937537  **   08:57:21   054/77      PAGE 0001
TXTST                 LENGTH 00A0

     MODULE      LENGTH  ORIGIN   DATE     TIME

     TXTST1       0052     0000   054/77   08:49
     TXTST2       001E     0052   054/77   11:18
     TXTST3       0030     0070   054/77   11:11

                    DEFINITIONS

N   CNT    001D   N NEW    0052      WRITE      0070
```

Following is the contents of the object file entitled :TXTST/OBJ. This file may be loaded into memory and executed.

```
000A0TXTST   A0000C0006C0045B0000A0026B1600B0D0AB0A20B4F4CB44207F21CF      0001
A0030B4D45B5353D4147B4520B2D2DB2057B524FB4E47B2021B210DB0A007F2B7F      0002
A0046B0420C0070C0028B001DB2FE0C0026B0D0AB0A48B4156B4520B41207F2E0F      0003
A005CB474FB4F44B2044B4159B2021B2120B2020B2020B200DB0A00C00767F2EFF      0004
A0072C0096B0000B0000B0B00B0000B0000B0000B0000A0096BC0FEBC17E7F30BF      0005
A009AB2FE0C0076B03807FB62F                                              0006
:        TXTST      054/77     08:57:21     TXLINK                      0007
```

SECTION VIII

TXDS COPY/CONCATENATE (TXCCAT) UTILITY PROGRAM

8.1 INTRODUCTION

The TXCCAT program copies one to three files to a single output file. Although simple record modifications are supported upon output, the program is basically a file copy by which sequential and relative record files may be duplicated or concatenated together into one file with no embedded end-of-files. TXCCAT copies information from cassettes, files or card reader input to cassettes, files, or printing devices.

NOTE

TXCCAT does not insert carriage control characters into its output file; therefore, it will not correctly copy nonlisting type files to a listing device, such as a printer.

TXCCAT is a software module which runs under the Terminal Executive Development System (TXDS).

8.2 TXCCAT LUNOs

The TXCCAT program uses LUNOs 7, 10, 11 and 12. LUNOs 10, 11 and 12 are assigned to the input files; LUNO 7 is assigned to the output file. All LUNOs are released upon termination of TXCCAT.

8.3 OPERATOR INTERACTION

TXCCAT is executed via the TXDS control program. Input and output pathnames are passed via COMMON. Table 8-1 provides the pathname defaults.

The INPUT: parameters may have one to three pathnames separated by commas. The files will be used to generate an output file. The OUTPUT: parameter contains one pathname to be used for the output file. Input files must preexist and can be either sequential or relative record. Output files are assumed to have the same characteristics as input files, although input records may be modified for output. If input is from a device, the output file is sequential and, if the output file does not exist, it is created. If no output file is specified, the default-system print device is used. The default-system print device is defined during system generation.

Table 8-1. Pathname Defaults

Field	Input	Output
DEV	System Disc	System Disc
FILE	None	First Input File Name
EXT	SRC	First Input Extension

8.4 OPTIONS

All TXCCAT options are specified by two alphabetic characters followed, in some cases, by a decimal numeric field. Input format is free-form. Delimiters for options may be commas, blanks or no *delimiter*. Each time a new pair of characters is read it is considered a new parameter. An illegal parameter results in an error message and program termination. Table 8-2 lists the options recognized by TXCCAT.

The numeric scan terminates after the maximum size is exhausted or a nonnumeric character is encountered.

8.4.1 TRUNCATE OPTION (TR). The truncate option truncates records to the size specified. The syntax is as follows:

TRnnnn

where n is a decimal number and four digits is the maximum field size. The following are examples:

TR76	Truncate to 76 characters.
TR0076	Truncate to 76 characters.

8.4.2 FIX RECORDS (FL). This option forces input records to a specified size by either padding with blanks or by truncation. The syntax is as follows:

FLnnnn

where n is a decimal number and four digits is the maximum field size. The following are examples:

FL76	Fix to 76 characters.
FL0076	Fix to 76 characters.

Table 8-2. TXCCAT Options

Option	Description
TRnnnn	Truncate record to length nnnn.
FLnnnn	Fix records to size nnnn by padding with blanks or by truncation.
SKnnnn	Skip nnnn input records, prior to output.
LFnn	List file, page length = nn, default = 55.
SLnn	Space lines on listing, nn = space count, default = 0.
NL	Number lines on listing.
RI	Do not rewind input on open.
RO	Do not rewind output on open.

Note: n is a decimal digit and the maximum field size is given by the number of n's.

8.4.3 **SKIP RECORDS (SK).** This option skips the specified number of input records prior to output. The syntax is as follows:

SKnnnn

where n is a decimal number and four digits is the maximum field size. The following are examples:

SK200
SK0020
SK9999

8.4.4 **LIST FILE (LF).** This option lists files and allows the use of the NL and SL option. The numeric field gives the printer page length. If the page length is not specified, 55 lines per page is the default. The syntax is as follows:

LFnn

where n is a decimal number and 2 digits is the maximum field size. The following are examples:

LF
LF55
LF06
LF99

8.4.5 **SPACE LISTING (SL).** This option is only in effect with the list option LF. The numeric field gives the number of blank lines to print for each input line. The syntax is as follows:

SLnn

where n is one or two decimal digits. The following are examples:

SL	No Spacing
SL1	Single Spacing
SL2	Double Spacing

8.4.6 **NUMBER LINES (NL).** This Option is only in effect with the list option LF and causes the printing of the line numbers associated with the input lines. The syntax is as follows:

NL

8.4.7 **NO INPUT REWIND (RI).** When this option is selected, the input is not rewound when opened. The syntax is as follows:

RI

8.4.8 **NO OUTPUT REWIND (RO).** When this option is selected, the output is not rewound when opened. The syntax is as follows:

RO

8.5 EXAMPLES

The following three examples show how to use TXCCAT to: copy a file to a cassette; copy three files to another file; and list a file on a printer. Each example assumes that the TXDS Control Program *is active* and that a diskette containing TXCCAT is loaded.

Example 1:

```
PROGRAM:    :TXCCAT/SYS
  INPUT:    DSC2:TXTST1/OBJ
 OUTPUT:    CS1
OPTIONS:    FL80
```

Example 2:

```
PROGRAM:    :TXCCAT/SYS
  INPUT:    DSC:TXTST1/SRC,DSC2:TXTST1/SRC,CS1
 OUTPUT:    DSC2:TXTST3/SRC*
```

Example 3:

This example assumes that a line printer is included in the system.

```
PROGRAM:    :TXCCAT/SYS
  INPUT:    DSC2:TXTST3/SRC
 OUTPUT:    LP
OPTIONS:    LF66,SL1,NL
```

8.6 ERRORS

The errors generated by TXCCAT are listed in table 8-3 together with possible corrective action.

Table 8-3. TXCCAT Errors

Error	Description	Action
CANT GET COMMON	System COMMON not in system	Regenerate system with COMMON
ILLEGAL OPTION – aa	Option aa not found	Reenter correct option
nn – ILLEGAL PATHNAME	Input files does not exist or open error	Correct and Retry
nn – READ ERROR	Error in Reading File	Retry
nn – WRITE ERROR	Error in Writing File	Retry
nn – OPEN ERROR	Error in opening or assigning files	Correct pathname and retry
nn – CLOSE ERROR	Error in closing a file	Correct pathname and retry

Note: nn is the system I/O status error given in Appendix D.

SECTION IX

TXDS STANDALONE DEBUG MONITOR (TXDBUG) UTILITY PROGRAM

9.1 INTRODUCTION

This section discusses the capabilities and operation of the TXDS standalone debug monitor, TXDBUG, explains how to debug under monitor control, gives detailed descriptions of the commands available to the user, and supplies debugging techniques. The following topics are covered:

- TXBUG installation procedures for 733ASR, 913 VDT or 911 VDT system consoles.
- A general description of TXDBUG, including functions, features and capabilities.
- A detailed description of the operating procedures necessary to load TXDBUG and the program to be debugged.
- A description of two modes of debugging: one in which the program being debugged executes with minimal TXDBUG intervention, and one in which TXDBUG exercises tight control of the program being debugged.
- A description of TXDBUG command structures, and the operator interface to TXDBUG.
- Detailed descriptions of each of the debug commands.
- A discussion of debugging techniques including general techniques and techniques specific to TX990.
- A discussion of methods used to patch programs (i.e., to correct them in memory rather than at the source code level.)
- A summary of errors which may occur during a debugging session.

The TXDBUG provides for debugging programs which have been designed to operate in a "stand-alone" environment with no operating system support. The debug monitor attempts to "hide" itself from the program being debugged, using as few machine resources as possible in the performance of debug tasks.

The following minimum hardware system configuration is required to run the standalone debug monitor:

- 990/4 CPU (including 6-slot chassis and Programmer Panel), or 990/10 CPU
- When a 990/4 CPU is used, 4096 words dynamic RAM Memory Expansion including Memory Write Protect and Memory Parity are needed.
- 733 ASR Data Terminal, 913 VDT or 911 VDT
- FD Floppy Disc.

9.2 GENERAL DESCRIPTION

The TXDBUG is memory-resident and communicates interactively with the operator through the 733 ASR Data Terminal keyboard and printer. It provides the following capabilities:

- Inspection and modification of memory, registers, and CRU space
- Controlled execution of user programs with optional trace of instructions and/or data
- Multiple breakpoints with optional automatic display of registers and specified memory
- Miscellaneous aids such as hexadecimal arithmetic and search-under-mask.

9.3 INSTALLATION OF TXDBUG

To install TXDBUG, a file name :SADBUG/SYS must be created on the TXDS Utilities Diskette. Then, one of the three stand-alone debug monitors listed below must be copied into it. The three debug monitors are named:

:SADPRT/SYS	733 ASR, 743 KSR or 33 ASR (TTY)
:SAD911/SYS	911 VDT
:SAD913/SYS	913 VDT

Example:

If the device to be used as the debug console is the 911 VDT, perform the following:

1. Place the TXDS System Diskette in DSC and TXDS Utilities Diskette in DSC2.
2. Respond to the following prompts by entering:

PROGRAM:	:TXCCAT/SYS
INPUT:	DSC2:SAD911/SYS
OUTPUT:	DSC2:SADBUG/SYS
OPTIONS:	

This will be create the file :SADBUG/SYS on DSC2 and copy :SAD911/SYS from DSC2 to the :SADBUG/SYS file.

9.3A LOADING TXDBUG

The TXDBUG program is stored as a file on the TXDS Utilities Diskette. To load TXDBUG and begin the debug session, invoke the program load facility of TXDS and specify the following parameters:

PROGRAM:	:TXDBUG/SYS
INPUT:	<file name of program to be debugged>
OUTPUT:	
OPTIONS:	<hexadecimal integer specifying the load point of the program to be debugged>

The file name supplied for the input parameter must include all extensions. The loader will search all available drives if the device name is not specified.

If the load point of the program to be debugged is not supplied, TXDBUG assumes the default value >A0.

After TXDBUG has been successfully loaded, the TXDBUG load point, entry point, and length are printed:

```
TXDBUG    937567 *A
TXDBUG LOAD POINT =      ENTRY POINT  =      LENGTH =
```

NOTE

For certain debug operations the TXDBUG entry point is required. Make note of the entry point at this time.

When the loading process is completed, the TXDBUG will prompt the operator with a period (“.”). At this time the memory configuration (for a 16K system) will appear as illustrated in figure 9-1, with the user's program located as specified by the load point in the option parameter. TXDBUG may be used to debug any program for which the instruction and data space does not overlap TXDBUG.

NOTE

Since the user's program overlays the TX990 executive, the TX990 executive must be rebooted when the debug session is finished.

Once the user program is entirely debugged, it may become the executing program when a disc boot is performed by using the “SF” operation described in the *TX990 Operating System Programmer's Guide.*

In the following description of TXDBUG usuage, the ESCAPE key denotes the ESC key on a 911 VDT or 733 ASR and the RESET key on a 913 VDT.

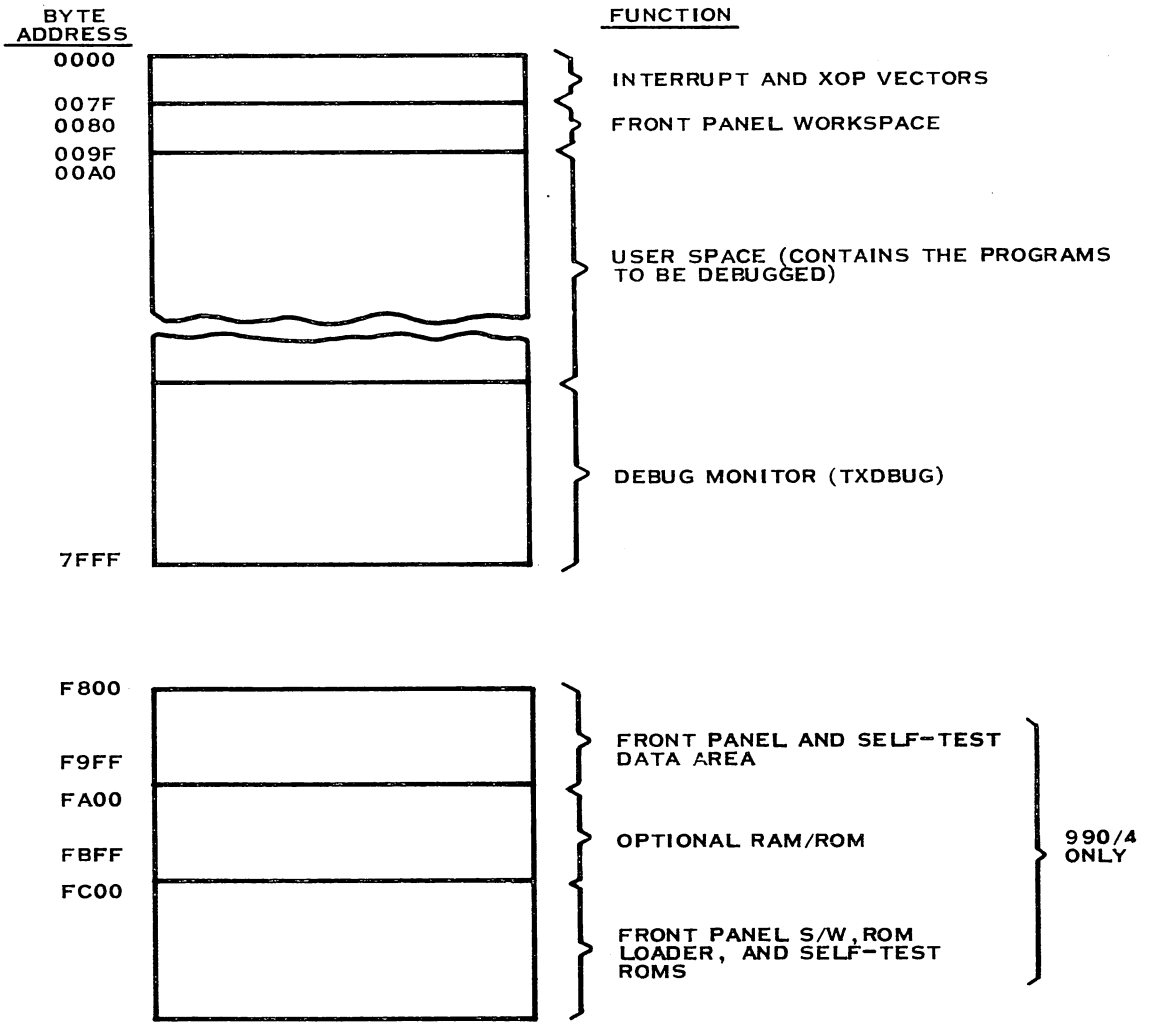


Figure 9-1. Debug Monitor Memory Configuration

9.4 DEBUG MODES

The user may specify that the debug monitor execute the program being debugged in either of two different modes: Execute free, or Run controlled. When executing free, the monitor relinquishes control to the test program which is then executed at full processor speed. This mode is only recommended when a program is expected to be error free or when timing considerations are being examined. The only way to interface with the monitor in this mode is to bracket instruction sequences in the test program with LREX instructions or with unconditional branches to the debug monitor entry point. An LREX has the same effect as pressing the HALT switch on the front panel. An unconditional branch to the monitor entry point restarts the monitor. If this is attempted and the monitor does not respond with a period prompt ("."), the probable cause is that the executing program has destroyed the monitor.

The normal method of execution during program debug is to initialize the PC, WP, ST by using the Modify AU Registers (MR) command and use the RUN (RU) command. In the RUN mode, the monitor uses the Single Instruction Execution (SIE) facility to execute the user's program one instruction at a time. Execution continues until: the number of instructions specified have been executed; a breakpoint occurs; or the operator presses the ESCAPE key on the 733 ASR Data Terminal keyboard. The Execute (EX) command can be used in place of the RUN command. Using this command, the program is executed without using the SIE or trace features.

The highest level of control is exercised when a test program is being executed via the RUN command and the instruction address is within a Trace region. In this case, each instruction is interpretively executed by the monitor. Source and destination operands are examined and optionally printed before and after each instruction. The amount of information printed as each instruction is executed is determined by user-defined Trace regions (SR command).

NOTE

Trace regions are ignored when the EX command (Execute free) is used.

9.5 DEBUG MONITOR COMMAND STRUCTURES

To interact with TXDBUG, the user enters commands at the 733 ASR Data Terminal.

The available debug commands may be classified into the following groups.

- *Set commands.* These commands allow the user to define up to four of each of the following aids: program-counter breakpoints, formatted snapshots, trace regions, and trace formats.
- *Clear commands.* These commands allow the user to remove the effect of a previously set command.
- *Inspect commands.* These commands allow the user to display the contents of AU registers, workspace registers, memory regions, and CRU lines. These commands are also used to force snapshots.
- *Modify commands.* These commands allow the user to examine and optionally modify: memory; workspace registers; AU registers; and CRU lines (by inspecting the input and modifying the output).
- *Miscellaneous commands.* These commands include functions such as word and byte memory searches, and hexadecimal arithmetic with automatic decimal conversion.

When debugging a program, the user may specify that TXDBUG:

- Print data on the terminal for examination,
- Modify data,
- Specify program elements (parameters whose values are determined by the user) for interpreting the progress of his program,
- Set and clear program elements,

- Search for specific bit patterns in bytes and words,
- Perform arithmetic calculations with hexadecimal numbers.

These actions may be performed on memory, registers, and CRU input and output lines. They may also be performed on specifiable debug elements: breakpoints, snapshots, and trace regions. The debug elements are defined as follows:

- *Breakpoint* – A point during the execution of a program at which control is returned to TXDBUG to allow the user to examine the progress of his program or enter any of the debug commands.
- *Snapshot* – A printed display of the contents of contiguous workspace registers plus the contents of an area in memory as defined by the operator. A snapshot may be printed automatically at a breakpoint.
- *Trace region* – An area of the program about which information concerning the execution of an instruction is output on the printer. This information may be printed following the execution of each instruction, each branch, or each change in the contents of a data word.

9.5.1 DEBUG COMMAND CODES. All debug commands are comprised of a two-letter mnemonic, the first of which denotes the operation to be performed (inspect, modify, etc.). The second identifies the debug or machine element upon which the operation is to be performed (memory, CRU, etc.). The four general-purpose operations are as follows:

I – Inspect

M – Modify

S – Set

C – Clear.

The elements on which these operations may be performed are:

M – Memory

W – Workspace registers (R0 –R15)

R – AU Registers (WP, PC, ST) when used with I, or M.

R – Trace Region when used with S or C

C – CRU

B – Breakpoint

S – Snapshot

T – Trace Type

P – Protect region (invalid for computer without write-protect option).

Some combinations of operations and elements are illegal. Table 9-1 identifies the valid combinations. Table 9-2 lists the available two-letter mnemonics associated with the valid combinations.

9.5.2 **MISCELLANEOUS COMMANDS.** The following are classified as miscellaneous commands:

EX – Execute a user program.

RU – Run a user program.

HA – Hexadecimal arithmetic.

FB – Search under mask for a particular 8-bit pattern (Find Byte).

FW – Search under mask for a particular 16-bit pattern (Find Word).

9.5.3 **COMMAND ENTRY.** Readiness of the monitor to accept a command is indicated when the monitor "prompts" the operator by printing a period (".") as the first character of a new line. For all activities except when a user program is being executed free (EX command) the operator may force a return to the command mode by pressing the ESCAPE key on the 911 VDT or 733 ASR terminal, or RESET on the 913 VDT.

From zero to eight parameters may be entered with each two-letter command. The command is separated from its parameter list by a comma (",") or by one or more blanks. Each parameter in the list is terminated by a comma or by one or more blanks, with the parameter list being terminated by a carriage return. As each parameter is entered, its syntax is validated by the monitor. The parameter may either be a hexadecimal number, a binary number, or a character string. The backspace character (CTRL-H on the 911 VDT or 733 ASR terminal, or ← on the 913 VDT) may be used to change the entered characters, or the entire parameter may be reentered by pressing the delete key (RUB OUT on the 733 ASR terminal). The entire command entry may be aborted by pressing the ESCAPE key on 911 VDT or 733 ASR terminal (RESET on 913 VDT).

If an error is detected by the monitor during command entry, one of the following error codes will be printed:

Code	Meaning
MP00	Invalid parameter or hexadecimal number entered, or maximum parameter list exceeded.
MS01	Invalid command. The first two characters do not match any known command.

A complete list of error codes appears in paragraph 9.8.

Table 9-1. Valid Debug Command Combinations

	Element							
Operation	**M**	**W**	**R**	**C**	**B**	**S**	**T**	**P**
I	X	X	X	X		X		
M	X	X	X	X				
S			X		X	X	X	X
C			X		X	X		X

("X" indicates acceptable combination.)

Table 9-2. TXDBUG Keyboard Commands

DEBUG Commands	
IC	Inspect Control Register Unit (CRU)
IM	Inspect Memory
IR	Inspect AU Register (WP, PC, ST)
IS	Inspect Snapshot
IW	Inspect Workspace Registers
MC	Modify Control Register Unit (CRU)
MM	Modify Memory
MR	Modify Registers
MW	Modify Workspace Registers
SB	Set Breakpoint
SP	Set H/W Write Protect Option
SR	Set Trace Region
SS	Set Snapshot
ST	Set Trace
CB	Clear Breakpoint
CP	Clear H/W Write Protect Option
CR	Clear Trace Region
CS	Clear Snapshot

9.5.4 **NOTATIONAL CONVENTIONS.** The notational conventions used in the syntax definitions of the keyboard commands are as follows:

< > Item to be supplied by the user. The term shown within angle brackets is a generic term.

[ ] Optional item – may be included or left out, at the user's discretion. Items not enclosed in brackets are required.

{ } Choice to be made from two or more items, one of which must be included.

Items in capital letters in the syntax definition are entered into the command statement exactly as shown.

The fields in the command (the command mnemonic and the parameters) are separated by either commas or strings of one or more blanks. This choice is shown sumbolically as:

$$\left\{ \begin{matrix} , \\ b \ldots \end{matrix} \right\}$$

When one or more parameters are omitted, two or more field separators may occur in sequence. The user must be sure that he includes the correct number of separators in a sequence; he should be aware of how they are interpreted by the computer. Two strings of blanks run together will be read as a single long string of blanks. A comma preceded or followed by a blank will be read as two separators in sequence. It is suggested, therefore, that commas (without preceding or following blanks) be used to set off omitted parameters.

In the examples of command statements, user-supplied data is underlined to distinguish it from data printed by the monitor. The carriage returns that terminate command statements are not shown in the examples.

9.6 COMMAND DESCRIPTIONS

Each command supported by the debug monitor and a brief functional description is presented in table 9-2. Detailed descriptions of the "miscellaneous" commands are presented in paragraphs 9.6.1 to 9.6.5. The remaining paragraphs provide detailed descriptions of the "debug" commands.

9.6.1 EXECUTE USER PROGRAM (EX). The Execute User Program command is used to execute a user program at speed with neither interference from nor control by the monitor. The program is executed at full processor speed. Initialize the AU registers (ST, PC, WP) using the MR command before using the EX command.

Syntax definition:

.EX

Description: The program is executed directly by the 990 computer without using the SIE or trace features. Execution is started with the PC, WP and ST that would be displayed if an Inspect Registers (IR) command were executed.

Application notes: In order to regain control from an executing user program, the user must transfer control to the monitor's starting memory location. This may be done by inserting a branch in the test program or by using the programmer panel.

Upon regaining control in the monitor, the WP, PC, and ST registers will have the same values as before the EX command unless execution of the user program destroyed the monitor data space.

Example:

Assume the user has written an assembler which assembles source from cassette or terminates depending on user input.

```
.IR
PC=046C WP=0000 ST=0000
.EX
ASM/TERM? A

ASM/TERM? T

.IR
PC=046C WP=0000 ST=0000
.
```

The EX command begins execution with the PC, WP, and ST registers equal to the values obtained when the Inspect Registers (IR) command is invoked. A program run under EX does not change the contents of these registers. The second IR command shows that the contents remain the same.

9.6.2 EXECUTE USER PROGRAM UNDER SIE OR TRACE (RU). The Execute User Program under SIE or Trace command provides controlled execution of the user's program. Initialize the AU registers (ST, PC, WP) using the MR command before using the RU command.

Syntax definition:

RU [{, / b...} <instruction count>]

Parameter:

instruction count	Maximum number of instructions to be executed before returning to command mode. A value of 0 indicates an infinite instruction limit applies.

Parameter default value: The value of the instruction count at the last entry into command mode is used as the default value. If the previous RU command has exhausted the instruction count, the default is 0, implying no instruction limit. The system is initially loaded with a default value of 0.

Description: Instructions in the user's program are executed one at a time using either the hardware SIE feature or the software trace interpreter. The user may specify one of these two modes of operation with the Set Trace Region (SR) command (paragraph 9.6.12). The user is referred to *The 990 Computer Family System's Handbook*, part number 945250-9701, for a detailed explanation of SIE.

Before the monitor executes a user instruction, it checks whether the instruction is within a defined trace region. If the instruction is within a trace region, the trace interpreter is called and the instruction traced. If the instruction is not within a trace region, the instruction is executed using Single Instruction Execution. In both cases, the user's WP, PC, and ST registers are updated after each instruction executed. The monitor checks whether a breakpoint has been reached and if so, prints out the user's registers and snapshot, if defined. If a snapshot is assigned to a breakpoint, the monitor continues execution after the breakpoint has been reached, without operation intervention. If no snapshot was specified, the monitor returns control to the command processor. (Refer to the descriptions of the SB and SS commands in paragraph 9.6.6.1 and 9.6.11.1.) If the run count, number of instructions to be executed, is depleted, the monitor returns control to the command processor. Otherwise the monitor continues execution of the user program.

9.6.3 HEXADECIMAL ARITHMETIC (HA). The Hexadecimal Arithmetic command calculates the sum and difference of two hexadecimal numbers. The 2's complement hexadecimal value and the signed decimal value are printed.

Syntax definition:

HA [{,|b...} [<value>] [{,|b...} <value>]]

Parameters:

value	Hexadecimal number value (0-4 digits).

Parameter default values:

If the value parameter is not specified, a default value of 0 is used.

Application note: No overflow checks are made; therefore, two positive numbers may have a negative sum. All results are represented in 16 bits.

Examples:

```
.HA 103A BA2
SUM=1BDC   07132  DIFF=0498  +01176
.

.HA 89 89
SUM=0112   00274  DIFF=0000  +00000
.

.HA 8030 EF00
SUM=6F30   28464  DIFF=9130  -28368
.

.HA EF00
SUM=EF00  -04352 DIFF=EF00 -04352
.
```

The calculated difference between the specified number values is the first value minus the second value.

9.6.4 **FIND BYTE (FB).** The Find Byte command is used to scan an area of memory for a particular byte value.

Syntax definition:

FB $\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$ [<start mem addr>] $\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$ [<ending mem addr>] $\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$

<desired value> [$\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$ <mask>]

The command is terminated by a carriage return.

Parameters:

start mem addr	Memory address at which search begins. (1-4 character hexadecimal number.)
ending memory addr	Memory address at which search is terminated. (1-4 character hexadecimal number.)
desired value	Hexadecimal value for which the search is made. This value is required.
mask	Hexadecimal value to be ANDed with each byte before comparing it with the desired value.

Parameter default values:

If the starting memory address is not specified, a value of 0 is used.

If the ending memory address is not specified, a value of $FFFF_{16}$ is used.

If the mask parameter is not specified, a value of FF_{16} is used.

Description: Each byte in the memory search range is ANDed with the mask and compared to the desired value. The memory location and contents are printed out whenever a match is found. After each match, the user must enter a space on the terminal keyboard to continue the search. If he enters a carriage return, the command terminates.

Error messages:

DP13 The ending address is less than the starting address. Reenter the command.

MS05 The <desired value> parameter is missing. Reenter the command.

MX06 The beginning address is an invalid memory address. Reenter the command.

Application notes: No check is made to ensure that the mask does not exclude a bit required by the desired value, thereby making a match impossible. If the monitor data area is being searched, results may not appear to be correct since the monitor is changing during the search process.

Examples:

```
.FB 0,2000,0,0F
0000=0000
0000=0000
0002=0000
0002=0000
0004=0000
0004=0000
0006=0000
0006=0000
0008=0000
.

.FB 0,2000,06,0F
0300=0456
0644=0556
.
```

In the first example, the high order four bits of each byte are masked so that any byte with a 0 in the low order four bits will be located. The address of the leftmost byte of each word is printed so that if both bytes of a word are printed, an address location will be printed twice. For example, if bytes 0004 and 0005 are printed, the address 0004 will appear twice in the listing.

In the second example, the high order four bits of each byte are masked so that any byte with a 6 in the low order four bits will be located.

9.6.5 **FIND WORD (FW).** The Find Word command is used to scan an area of memory for a particular word value.

Syntax definition:

FW {,/b...} [<start mem addr>] {,/b...} [<ending mem addr>] {,/b...}
<desired value> [{,/b...} <mask>]

The command is terminated by a carriage return.

Parameters:

start mem addr	Memory address at which search begins. (1-4 hexadecimal characters.) Must be even byte (word) address.
ending memory addr	Memory address at which search is terminated. (1-4 hexadecimal characters.)
desired value	Hexadecimal value for which the search is made. The value is required.
mask	Hexadecimal value to be ANDed with each word before comparing it with desired value.

Parameter default values:

If the starting memory address is not specified, a value of 0 is used.

If the ending memory address is not specified, a value of $FFFF_{16}$ is used.

If the mask parameter is not specified, a value of $FFFF_{16}$ is used.

Description: **Each word in the memory search range is ANDed with the mask and compared to the desired value. The memory location and contents are printed out whenever a match is found. After each match, the user must enter a space on the terminal keyboard to continue the search. If he enters a carriage return, the command terminates.**

Error messages:

DP13	**The ending address is less than the starting address. Reenter the command.**
MP00	**The beginning address is an invalid memory address. Reenter the command.**
MS05	The <desired value> parameter is missing. Reenter the command.

Application notes: No check is made to ensure that the mask does not exclude a bit required by the desired value, thereby making a match impossible. If the monitor is being searched, results may not appear to be correct since the monitor is changing during the search process.

Examples:

```
.FW 0,2999,456,
0300=0456
.FW 0,2000,56,00FF
0300=0456
0644=0556
.
```

In the second example, the monitor searches for words with a 56 in the low order byte. By pressing the space bar on the terminal keyboard, the user can cause the monitor to continue searching for another occurrence of the data word.

9.6.6 BREAKPOINT COMMANDS (SB, CB). These two commands control breakpoint as indicated in the following paragraphs.

9.6.6.1 Set Breakpoint (SB). The Set Breakpoint command is used to define a breakpoint which causes the processor to stop or interrupt execution of a user program prior to executing the instruction at the specified memory address.

Syntax definition:

SB $\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$ <bkpt no.> $\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$ <memory addr> $\left[\left\{\begin{matrix}, \\ b...\end{matrix}\right\}\left[\text{<ref cnt>}\right]\right.$
$\left.\left[\left\{\begin{matrix}, \\ b...\end{matrix}\right\}\text{<snapshot no.>}\right]\right]$

Parameters:

bkpt no.	Breakpoint index number. The number may be 0, 1, 2 or 3. Required parameter which services as a unique identifier for individual breakpoints.
memory addr	Address of an instruction on which the breakpoint is to be set. Required parameter. (1-4 hexadecimal characters.)
ref cnt	The pass number (hexadecimal) on which a breakpoint is to be taken. For example, a reference count of 3 means to break on the third reference to the memory address for an instruction fetch. Default value is 1.
snapshot no.	Index number of a previously defined snapshot which is to be displayed when the breakpoint is taken (see SS command). Default value is no snapshot 0, 1, 2, 3.

Parameter default values:

If the reference count (pass number) is not specified, a value of 1 is used. If the user enters a value of 0, it is equivalent to a reference count of $FFFF_{16}$.

If the snapshot number is not specified, a snapshot is not printed.

Use of breakpoints: The breakpoint is one of the key elements in program debugging because it enables the user to specify conditions under which he wants to receive control. Breakpoints are particularly useful when the user wants to intercept control after an unexpected control transfer occurs from a conditional branch. By setting a breakpoint on the unexpected or error path out of a conditional branch, the program may be allowed to execute without interruption unless some error condition occurs.

When a breakpoint is encountered, the contents of the processor registers are displayed. (The contents are the values that would be displayed if an IR command were to be invoked.) The breakpoint index number is aiso displayed to aid in determining which breakpoint was encountered.

Error message:

DP20	Breakpoint specification error. Required index number may be valid or missing, or the PC value (memory address) may have been omitted.

Application notes: The PC value for a breakpoint must point to the first word of a multiword instruction.

A breakpoint occurs *before* the execution of the instruction to which it points.

If a snapshot is associated with a breakpoint, execution of the user program resumes after the snapshot is printed. If no snapshot is associated with the breakpoint, execution terminates and the debugger accepts another command.

If more than one breakpoint is associated with a specific loation, only the first (lowest numbered) will be found.

When execution is under the control of the Execute User Program under SIE or Trace (RU) command with an instruction count: (1) a breakpoint occurs; and (2) a new count is not specified on the next RU command. Then, when execution is resumed, counting is continued as if no breakpoint was encountered.

Breakpoints are not active when the user code is executed with the EX command.

An error is not reported when a Set Breakpoint (SB) command redefines an already defined breakpoint. The specified breakpoint is modified to take on a new definition. This feature may be used to modify the snapshot index associated with a breakpoint.

Examples:

```
.SB 0,1000,1,2

.SB 1,1000,1,0

.SB 2,1004
```

The first two examples set a breakpoint at address 1000 on the first reference to that address for an instruction fetch. The first example sets breakpoint index number 0 with snapshot index number 2 to be displayed, and the second example sets breakpoint index number 1 with snapshot index number 0 to be displayed. The third example specifies breakpoint index number 2 to be taken at memory location 1004_{16}. No snapshot is printed, and execution of the user program terminates after the breakpoint is encountered.

9.6.6.2 **CLEAR BREAKPOINT (CB).** The Clear Breakpoint command is used to disable previously specified breakpoints.

Syntax definition:

CB [{,/b...}[<starting breakpoint number>][{,/b...} <ending breakpoint number>]]

The command is terminated by a carriage return.

Parameters:

starting breakpoint number	The first breakpoint to be cleared. A number from 0 to 3.
ending breakpoint number	The last breakpoint to be cleared. A number from 0 to 3.

Parameter default values:

If no parameters are specified, all breakpoints are cleared.

If only the first parameter is given, only the specified breakpoint will be cleared.

If only the second parameter is given, breakpoints 0 through the specified ending breakpoint will be cleared.

Description: If an attempt is made to clear a breakpoint that has not been set, the command is ignored.

Error message:

DP13	A breakpoint index greater than the maximum possible index number (3) was specified, or the ending breakpoint index was less than the starting breakpoint index number.

Examples:

.CB 1,3

.CB

The first example clears all breakpoints except number 0. The second example clears all breakpoints.

9.6.7 **COMMUNICATIONS REGISTER UNIT COMMANDS (IC, MC).** Commands to control the 990 I/O port (the Communications Register Unit) are explained in the following paragraphs.

9.6.7.1 **Inspect CRU Input Lines (IC).** The Inspect CRU Input Lines command is used to display in hexadecimal format the contents of one or more consecutive CRU locations.

Syntax definition:

IC [{,/b...} [<CRU address lower limit>] [[{,/b...} <CRU address upper limit>]]

The command is terminated by a carriage return.

Parameters:

CRU lower limit	CRU address that begins the display. The address must be in the range of 0 to $1FFF_{16}$. (1-4 hexadecimal characters.)
CRU upper limit	CRU address that ends the display. The address must be in the range 0 to $1FFF_{16}$. (1-4 hexadecimal characters.)

Parameter default values:

If the CRU lower limit is not specified, a value of 0 is used.

If the CRU upper limit is not specified and the CRU lower limit is specified, the default value is the CRU lower limit. Sixteen bits are displayed.

If neither parameter is specified, the entire CRU is displayed.

Description: Data is displayed in groups of four words, two groups per line. The address of the first word on the line is printed on the left. The display may be terminated at any time by pressing the ESC key on the terminal keyboard.

The address displayed is the actual CRU bit address times two.

Error message:

DP13	The highest CRU address specified is less than the lowest CRU address specified, or the highest CRU address specified is greater than the highest CRU address permitted ($1FFF_{16}$).

Examples:

```
.IC 1000 1060
1000=FFFF FFFF FFFF FFFF
.

.IC 100
0100=608D
.
```

In the first example, the CRU bits at addresses 1000_{16} through 1060_{16}. in 20_{16} bit increments, are displayed. Since the CRU addresses are twice the actual bit addresses, the address of the next 10_{16} CRU bits would be a 20_{16} address increment. In the second example, the 16 CRU bits at location 100_{16} are displayed.

Example:

```
.IC
0000=600D   FFFF   FFFF   40DF   >0000   8001   0D00   409B
0100=FFFF   FFFF   FFFF   FFFF   >FFFF   FFFF   FFFF   FFFF
0200=FFFF   FFFF   FFFF   FFFF   >FFFF   FFFF   FFFF   FFFF
```

9.6.7.2 MODIFY CRU REGISTER (MC). The Modify CRU Register command reads and displays the data on CRU input lines, and sets data on CRU output lines.

Syntax definition:

MC [{,/b...} [<CRU address>] [{,/b...} <CRU width>]]

The command is terminated by a carriage return.

Parameters:

CRU address	The CRU word address. A value from 0 to $1FFF_{16}$. (1-4 hexadecimal characters.)
CRU width	The number of bits to be changed in each CRU word (hexadecimal). A value from 1 to 10_{16}. A value of 0 is interpreted as 10_{16}. (1-2 hexadecimal characters.)

Parameter default values:

If the CRU word address is not specified, a value of 0 is used.

If the CRU width is not specified, a value of 10_{16} is used.

Description: When the CRU bit width is less than 16 bits, the data value is displayed right justified in a four-digit hexadecimal value. The user's data may be input as a four-digit value; the rightmost bits, where the bit width is given by the CRU width parameter, are used to modify the CRU value. Enter a new value to change the value, a space to continue on to the next value, and a carriage return to terminate data modification.

The addresses are displayed as they would be used in workspace register 12 (the CRU base address), which is the actual CRU bit address times 2. Also, data is displayed and entered directly as the STCR/LDCR instruction receives/sends it.

If the CRU word address is greater than $1FFF_{16}$, the command is ignored.

Error message:

DP12 CRU bit width parameter too small (negative) or too large (greater than 10_{16}). Invalid bit string width.

Application note: The Modify CRU Register command may be used to change the data being sent to an external device during the debugging of a new interface.

Examples:

```
.MC 1000 8
1000=00FF 0080
1010=00FF 0040
.

.MC 1000
1000=FFFF 1000
```

9.6.8 **MEMORY COMMANDS (IM, MM).** The commands explained in the following paragraphs allow user knowledge and control of memory contents.

9.6.8.1 **Inspect Memory (IM).** The Inspect Memory command is used to display in hexadecimal format the contents of one or more consecutive memory locations.

Syntax definition:

IM [{,b...} <starting mem addr> [{,b...} <ending mem addr>]]

The command is terminated by a carriage return.

Parameters:

starting mem addr	Hexadecimal value representing the memory address of the first memory word displayed. (1-4 hexadecimal characters.)
ending mem addr	Hexadecimal value representing the memory address of the last memory word displayed. (1-4 hexadecimal characters.)

Parameter default values:

If neither parameter is specified, all memory is dumped.

If the ending address is not specified, only one word is displayed.

An odd address is changed to the preceding word address before the addressed byte is displayed.

Description: Memory is displayed in groups of four words, two groups per line. The address of the first word on the line is printed at the left. The display may be terminated at any time by pressing the ESC key on the terminal keyboard.

Error message;

DP13 The ending address specified is less than the starting address specified.

Examples:

```
.IM 1000,1004
1000=1002 C0E0 023E
.

.IM 1006
1006=1004
.
```

9.6.8.2 **MODIFY MEMORY (MM).** The Modify Memory command displays the address and contents of a memory word and accepts a new hexadecimal data value from the user.

Syntax definition:

MM [{,/b...} <memory address>]

The command is terminated by a carriage return.

Parameter:

memory address Address of memory to be modified.

Parameter default value: If the memory address is not specified, a value of 0 is used.

Description: If the user inputs a new value, the memory location is modified to match the input value. If the user terminates his input with a blank (space), the next location value is printed and the process repeated. If the user terminates his input with a carriage return or comma, the command processing terminates.

Error message:

DP00 An invalid hexadecimal value was input.

Application note: The MM command is useful for setting up desired conditions in order to check out a routine. It is also convenient for creating patches and for examining memory one word at a time.

Example:

```
.MM 1000
1000=FFFF 1
1002=FFFF 3
1004=FFFF
1006=FFFF 8
.
```

These command statements place the value 1 in location 1000, 3 in location 1002, and 8 in location 1006. The user may enter a space (blank) if he does not want to modify a location but wants to go on to the next location. A carriage return terminates the command at any time.

9.6.9 PROCESSOR REGISTER COMMANDS (IR, MR). The following commands allow control of the 990 computer program control registers: the program counter, workspace pointer, and status registers.

9.6.9.1 Inspect Registers (IR). The Inspect Registers command displays the contents of the user's registers: the program counter (PC), workspace pointer (WP), and status (ST) registers for the current user program. These values are displayed in groups of four hexadecimal characters.

Syntax definition:

IR

The command is terminated by a carriage return.

Application note: The displayed register values are those values which are loaded into the processor in response to an EX or RU command.

Example:

```
.IR
PC=0246 WP=0000 ST=0000
.
```

9.6.9.2 Modify Registers (MR). The Modify Registers command displays the contents of the user's internal registers – workspace pointer (WP), program counter (PC), and status (ST) registers – and allows the user to modify them.

Syntax definition:

MR

The command is terminated by a carriage return.

Description: The register name and current contents are printed in hexadecimal and a hexadecimal input is accepted from the user. If the user inputs a valid hexadecimal number, the contents of the registers are changed. If the user enters a space, the processor prints the name and contents of the next register. If the user enters a carriage return, the command terminates.

Error message:

DP00 An invalid hexadecimal number was input, or the number input was greater than $FFFF_{16}$.

Application notes: Modification of the Workspace Pointer (WP) register causes the registers that would be displayed by the Inspect Workspace Registers (IW) command to change. The Modify Registers command is used to establish the initial environment for a program executed with the Execute User Program Directly (EX) or the Execute User Program under SIE or Trace (RU) command.

Examples:

```
.MR

PC=2000 244
WP=0000 A6
ST=0000
.

.MR

PC=0244
WP=00A6 A2
ST=0000 2
.

.MR

PC=0244 246
.
```

The first example changes the value in the PC register to 244_{16} and the value in the WP register to $A6_{16}$. The second example changes the WP register value to $A2_{16}$ and the ST register value to 2_{16}. The third example changes the PC register value to 246_{16}.

As in the second example, the user may press the space bar on the terminal keyboard if he does not wish to modify a particular register. As in the third example, he may press the RETURN key on the terminal keyboard after entering a new PC register value to terminate the command.

9.6.10 **WORKSPACE REGISTER COMMANDS (IW, MW).** The following commands allow precise control of the memory area selected to be the workspace registers.

9.6.10.1 **Inspect Workspace Registers (IW).** The Inspect Workspace Registers command is used to display the contents of a sequence of the user's workspace registers.

Syntax definition:

IW [{, / b}[<starting reg number>] [{, / b} <ending reg number>]]

The command is terminated by a carriage return.

Parameters:

starting reg number	The number of the first workspace register to be displayed. Single hexadecimal number.
ending reg number	The number of the last workspace register to be displayed. Single hexadecimal number.

Parameter default values:

If the starting workspace register is not specified, a value of 0, signaling register 0, is used.

If the ending workspace register is not specified, the value used is the starting workspace register.

If neither parameter is specified, all 16 registers are displayed.

Description: The set of workspace registers displayed are those pointed to by the WP that would be displayed if an IR command were executed. Workspace registers are displayed with the register number preceding the register contents.

Error message:

DP13	Either the starting workspace register number is greater than the ending workspace register number, or a workspace register number greater than F_{16} was requested.

Examples:

```
.IW
R0=0000 R1=0000 R2=0026 R3=0000 R4=0000 R5=2032 R6=0000 R7=0000
R8=0000 R9=0000 RA=0000 RB=0000 RC=0000 RD=3798 RE=2008 RF=0002
```

If no workspace register or range is specified, all 16 registers are printed.

```
.IW 2,8
R2=0000 R3=0000 R4=0000 R5=0000 R6=0000 R7=0000 R8=0000

.IW 2
R2=0000
.
```

9.6.10.2 Modify Workspace Registers (MW). The Modify Workspace Registers command is used to display and change the contents of one or more of the user's workspace registers.

Syntax definition:

MW [{ , / b... } < starting reg number >]

The command is terminated by a carriage return.

Parameter:

starting workspace reg	The number of the first workspace register to be displayed. (Hexadecimal value.)

Parameter default value:

If the starting workspace register is not specified, register zero is assumed and a value of 0 is used.

Description: The mnemonic and current contents of the workspace registers are displayed. The command processor accepts the user's input, which may be a new hexadecimal value for the register contents and a terminator. If this input is a new value, the current contents of the specified register are changed. If the terminator is a blank, the next register is printed for modification. If the terminator is a carriage return or comma, the command processing terminates. The command processing terminates automatically after processing workspace register 15 (F_{16}).

Application note: The user is cautioned to be sure that the workspace pointer actually points to the intended workspace. The Modify Workspace Registers command displays the registers within the current workspace (the workspace defined by displaying the WP in an IR command).

Example:

```
.MW 4
R4=0000 7
R5=0000 89
R6=0000
R7=0000 1000
.
```

This example changes the contents of workspace registers R4, R5 and R7 to 7_{16}, 89_{16} and 1000_{16}, respectively. A carriage return was entered after changing the contents of R7.

9.6.11 **SNAPSHOT COMMANDS (SS, IS, CS).** The following commands provide a convenient way to specify debugging information to be displayed.

9.6.11.1 **Set Snapshot (SS).** The Set Snapshot command is used to define a set of registers and memory locations to be displayed as a single unit.

Syntax definition:

SS {,/b...} [<snapshot no.>] [{,/b...} [<starting reg no.>] [{,/b...} [<ending reg no.>]
[{,/b...} [<starting memory addr>] [{,/b...} <ending memory addr>]]]]

The command is terminated by a carriage return.

Parameters:

snapshot no.	Index number of snapshot to be defined. The index is a number in the range 0-3.
starting reg no.	First workspace register to be displayed.
ending reg no.	Last workspace register to be displayed.
starting memory addr	First memory word address to be displayed.
ending memory addr	Last memory word address to be displayed.

Parameter default values:

If the snapshot number is not specified, a value of 0 is used.

If the starting workspace register number is not specified, a value of 0 is used.

If the ending workspace register number is not specified, the value used is the starting register number if the starting register number is specified. Otherwise, the value is 0_{16}.

If the starting memory address is not specified, a value of 0 is used.

If the ending memory address is not specified, the value used is the starting memory address if the starting memory address is specified. Otherwise, it is 0_{16}.

Description: Snapshots may be invoked with the Inspect Snapshot (IS) command or when a breakpoint which references the snapshot index is encountered.

Error messages:

DP03 **A parameter is greater than the required maximum value. Reenter the command.**

DP04 **Snapshot is already defined. Reenter the command.**

DP13 **The ending parameter (register or memory address) is less than the beginning parameter.**

Application notes: Snapshots are convenient for defining a frequently used display during a debug session. If certain registers or memory data areas are frequently modified, they are likely choices for snapshots.

Since a snapshot may be attached to a PC breakpoint to dump some data and continue execution, a trace can be constructed which will be activated only when some specified event occurs. A dump may be produced and execution will continue without operator intervention.

Snapshots are useful for extended traces when the user wants to leave the computer running with breakpoints established. This would allow the computer to do an automatic dump when an exceptional condition is encountered and then continue execution.

Examples:

```
.SS 1,2,5,1000,1002

.SS 0,0,F
```

In the first example, the snapshot associated with index 1 displays workspace registers 2 through 5 and memory locations 1000_{16} through 1002_{16}. In the second example, the snapshot associated with index 0 displays workspace registers 0 through F_{16} and memory address 0 (the default). Refer to the IS command examples in paragraph 9.6.11.2 for the corresponding commands.

9.6.11.2 **Inspect Snapshot (IS).** The Inspect Snapshot command is used to display sequences of workspace registers and memory addresses.

Syntax definition:

IS [{,/b...}] [<starting snapshot no.>] [[{,/b...}] <ending snapshot no.>]]

The command is terminated by a carriage return.

Parameters:

starting snapshot no.	Index number (number of the snapshot in sequence) of the first snapshot to be displayed. A number from 0 to 3.
ending snapshot no.	Index number of the last snapshot to be displayed. A number from 0 to 3.

Parameter default values:

If neither the starting snapshot number nor the ending snapshot number is specified, all snapshots are displayed.

If the starting snapshot number but not the ending snapshot number is specified, the named snapshot is displayed.

If the ending snapshot number but not the starting snapshot number is specified, the snapshots from 0 through the specified snapshot are displayed.

Description: Snapshots are defined with the Set Snapshot command. Attempts to display undefined snapshots are ignored.

Error message:

DP13	Either the ending snapshot number is greater than the starting snapshot number, or a snapshot number greater than the permitted maximum was input. Re-enter the command with the correct snapshot numbers.

Examples:

```
.IS
SNAP0
R0=0000 R1=0000 R2=0000 R3=0000 R4=0007 R5=0089 R6=0000 R7=0000
R8=0000 R9=0000 RA=0000 RB=0000 RC=0000 RD=0000 RE=0000 RF=0000
0000=0000
SNAP1
R2=0000 R3=0000 R4=0007 R5=0089
1000=0001 0003
.
```

```
.IS 1,3
SNAP1
R2=0000 R3=0000 R4=0007 R5=0089
1000=0001 0003
.

.IS 3
.
```

The snapshots in these examples were set in the examples of the Set Snapshot (SS) command (paragraph 9.6.11.1). In the last example, if a snapshot is not set, the monitor will return control without printing anything.

9.6.11.3 Clear Snapshot (CS). The Clear Snapshot command is used to disable previously specified snapshots.

Syntax definition:

CS [{, / b}[<starting snapshot number>] [{, / b} <ending snapsnot number>]]

The command is terminated by a carriage return.

Parameters:

starting snapshot number	The first snapshot to be cleared. A number from 0 to 3.
ending snapshot number	The last snapshot to be cleared. A number from 0 to 3.

Parameter default values:

If no parameters are specified, all snapshots are cleared.

If only the first parameter is given, only the specified snapshot will be cleared.

If only the second parameter is given, snapshot 0 through the specified ending snapshot will be cleared.

Description: If an attempt is made to clear a snapshot that has not been set, the command is ignored.

Error message:

DP13	A snapshot index greater than the maximum possible index number (3) was specified, or the ending snapshot index was less than the starting snapshot index number.

Examples:

```
.CS 0,2

.CS 2
```

In the first example, all snapshots except index number 3 are cleared. In the second example, only snapshot 2 is cleared.

9.6.12 TRACE COMMANDS (ST, SR, CR)

The following commands allow precise control of regions to be examined in detail during a debug session, including specification of the information to be displayed.

9.6.12.1 Set Trace Definition (ST). The Set Trace Definition command defines parameters that determine what information about instruction trace regions will be printed. There are up to four different trace formats that may be defined, any one of which may be associated with one or more "trace regions". The format determines what is to be displayed for each instruction traced in the associated region.

Syntax definition:

ST $\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$ <format index> $\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$ <char string>

The command is terminated by a carriage return.

Parameters:

format index	Trace format index number; a number from 0 to 3.
char string	Character string describing the options to be printed. The string contains from 1 to 27 characters.

Parameter default values: **There are no default values. Both parameters are required.**

Character string symbols: **The character string symbol definitions and the associated trace printouts are as follows:**

Character	Trace Output	Description
P	XXXX	Program counter. The program counter is printed for every instruction executed. The program counter value is printed if anything else is printed even if "P" was not specified (example 1).
I	F-IIII	Instruction and format. (Instruction formats are described in the *Model 990 Computer TMS9900 Microprocessor Assembly Language Programmer's Guide*, Manual No. 943441-9701.) The instruction and its format are printed for each instruction executed (example 2).
M	ST=XXXX	Status mask. The contents of the status mask which is placed in the user status register is printed after each instruction executed (example 2).
W	WP=XXXX	Workspace pointer changes. When the user's workspace changes, the new workspace is printed.
T	BT=XXXX	Targets for branch or jump instruction. Whenever a branch or jump occurs, the target address of the branch/jump is printed.

Character	Trace Output	Description
C	C=XXXX	CRU address. When one of the instructions that references the CRU (LDCR, STCR, TB, SBO, SBZ) is executed, the address of the first bit referenced is printed. For example, for TB 2, the address is base (=R12) + 2.
N	(null)	Null trace. No printout occurs. If any other characters occur in the string, the null trace is overridden.
X	X-XXXX	XOP level. When an XOP instruction is executed, the XOP level is printed.
S		Source. Refers to the source register. It is followed by an E, B, A or R.
E	SE=XXXX	Source effective address. This address is the memory location that the source field addresses. It is printed for every instruction (example 2) that has a source operand.
B	SB=XXXX	Contents of source effective address before execution. The contents of the source effective address before execution are printed for every instruction (example 2) with a source operand.
A	SA=XXXX	Contents of source effective address after execution. The contents of the source effective address are printed after each instruction with a source operand is executed (example 2).
R	SR=XXXX	Contents of source workspace register after execution for T_s = 3 (indirect addressing with autoincrement). (T_s is the source addressing mode field in an assembly language machine instruction.) The contents of the source register is printed if an autoincrement is specified.
D		Destination. Refers to the destination. It is followed by an E, B, A or R.
E	DE=XXXX	Destination effective address. This address is the memory address that the destination field addresses. The destination effective address is only printed for Format 1, 3, and 9 assembly language machine instructions. All other instruction format types do not have a destination field (example 2).
B	DB=XXXX	Contents of destination effective address before statement executed. This is printed whenever a destination field exists (example 2).
A	DA=XXXX	Contents of destination effective address after execution. This is printed whenever a destination field exists (example 2).
R	DR=XXXX	Contents of destination workspace register after execution for T_d = 3 (indirect addressing with autoincrement). (T_d is the destination addressing mode field in an assembly language machine instruction.) The contents of the destination register is printed if an autoincrement is specified.

Description: The character string is scanned for proper syntax. If the string conforms to the syntax, a trace print control template is built and placed in the trace format table.

The character string in the ST command allows the user to select only those portions of the trace output that he needs. For tutorial purposes, an extensive trace output could be requested, while minimal traces such as a PC or variable trace are also easily selected. Each character in the character string represents a desired portion of the trace.

If any trace option other than PC is printed, PC is also printed.

A trace on a variable (see ST command) is implemented by specifying the desired variable.

The character string is scanned from left to right. The characters E, B, A and R are modified by the most recent occurrence of S or D. If E, B, A or R is encountered before an occurrence of S or D, or if an invalid character is encountered, the scan is aborted and an invalid syntax message is issued. A character string consisting entirely of S or D is also an invalid syntax.

All four trace format table elements have initial values as follows when the debug monitor overlay containing the ST command is loaded:

Index Number	Equivalent Character String
0	P
1	PIWSEADEA
2	T
3	PIMWTCXSEBARDEBAR (all trace output options)

Error messages:

DP23 Syntax error in trace format character string. Reenter the command.

DP26 Invalid trace format index number. Reenter the command.

Examples of typical character strings: Some examples of typical character strings are presented here. To invoke a PC trace, the character string is

P

If a branch trace is desired, the character string is

T

The character string for a trace that includes PC, instruction and format, workspace pointer changes, and source and destination effective addresses is

PIWSEDE

To specify all options, the character string is the same as the string equivalent to default trace format index number 3 (above).

Example 1: Trace format 1 in the following example is defined as a program counter trace. The program counter is the only option printed.

```
.ST 1,P
.SR 1,0,2000,1,N
.MR

PC=198C  46C
.RU
046C
0470
0474
1A92
1A96
198C
198E
1992
1994
1996
.
```

Example 2: This example shows the trace format index number 1 set to a full trace.

```
.ST 1,PIMWTCXSEBARDEBAR
.SR 1,24C,260,1,S
.MR

PC=0250  24C
.RU
024C  8-02E0  ST=0000  SE=00A6  SB=024C  SA=024C
0250  6-04E0  ST=0000  SE=01FC  SB=0054  SA=0000
0254  6-04E0  ST=0000  SE=01B4  SB=C259  SA=0000
0258  6-04E0  ST=0000  SE=01B8  SB=C060  SA=0000
025C  6-0720  ST=0000  SE=01BA  SB=01E6  SA=FFFF
0260  1-C820  ST=C000  SE=021E  SB=109A  SA=109A  DE=00D2
      DB=1850  DA=109A
```

9.6.12.2 **Set Trace Region (SR).** The Set Trace Region command defined a trace region.

Syntax definition:

SR $\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$ <region index> $\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$ <lower mem addr> $\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$ <upper mem addr>
$\left\{\begin{matrix}, \\ b...\end{matrix}\right\}$ <format index> $\left[\left\{\begin{matrix}, \\ b...\end{matrix}\right\}\right.$ $\left[\right.$<step region>$\left.\right]$ $\left[\left\{\begin{matrix}, \\ b...\end{matrix}\right\}\right.$ <v1> $\left[\left\{\begin{matrix}, \\ b...\end{matrix}\right\}\right.$ <v2>
$\left[\left\{\begin{matrix}, \\ b...\end{matrix}\right\}\right.$ <v3>$\left.\right]\left.\right]\left.\right]\left.\right]$

The command is terminated by a carriage return.

Parameters:

region index	Trace region index number; a number from 0 to 3.
lower mem addr	First memory address in the trace region; a hexadecimal number in the range 0 to FFFE.
upper mem addr	Last memory address in the trace region; a hexadecimal number in the range 0 to FFFE.
format index	Trace format index number; a number from 0 to 3.
step region	If this field contains S, an instruction step region is specified. If it contains N, the field specifies no instruction step. Any other character specifies no instruction step.
v1, v2, v3	Addresses of variables to be traced while in the designated region. Up to three variables may be specified. The range of values for each variable is 0 to $FFFE_{16}$. In the printed trace data, only changes are shown.

Parameter default values:

The first four parameters in the syntax definitions are required.

If the step region parameter is not specified, a value of N is used.

If none of the parameters v1, v2, and v3 are specified, no variables will be traced in the designated region.

Description: The specified regions of memory are designated as the program area to be executed under control of the interpretive trace.

The trace region index number determines which trace type will be executed as defined by the Set Trace Definition (ST) command. If two overlapping regions have been defined, the region with the lowest index has precedence and the trace type defined in that region is executed. (See example 1.)

The trace format index number indicates the trace type vector assigned to the trace region. When the trace overlay is loaded, each of the four trace type vectors, indices 0 through 3, is assigned an initial value. These vectors may be modified by the Set Trace Definition (ST) command. Trace types may vary from a null trace to a full trace.

The function of the instruction step region is to control the execution of the user program. If the instruction step region is set by entering an S parameter on the terminal keyboard, only one instruction at a time will be executed and traced. To execute another instruction, the user must press the space bar.

If variables have been specified to be traced, only changes will be printed. The format of the output is:

AAAA = DDDD

Where AAAA is the address of the variable and DDDD is the new value of the variable. These are hexadecimal values.

Error messages:

DP13 The specified last memory address was less than the first memory address. Reenter the command.

DP10 Invalid trace region index number. Reenter the command.

DP26 Invalid trace format index number. Reenter the command.

Example 1: This example shows the setting of two different trace regions, one a PC trace and the other a full trace. The region with the lower index is executed when the two regions overlap. In this manner, the user can get a general trace until he reaches a critical section of the program where he wants everything traced.

```
.ST 1,PIMWTCXSEBARDEBAR
.ST 2,P
.SR 2,0,2000,2,N
.SR 1,24C,260,1,S
.MR

PC=0250  246
.RU
0246
024A
024C  8-02E0   ST=0000   SE=00A6   SB=024C   SA=024C
0250  6-04E0   ST=0000   SE=01FC   SB=0054   SA=0000
0254  6-04E0   ST=0000   SE=01B4   SB=C259   SA=0000
0258  6-04E0   ST=0000   SE=01B8   SB=C060   SA=0000
025C  6-0720   ST=0000   SE=01BA   SB=01E6   SA=FFFF
0260  1-C820   ST=C000   SE=021E   SB=109A   SA=109A   DE=00D2
      DB=1850   DA=109A
0266
026A
0270
0274
0278
027A
027E
.
```

Outside the critical region, a continuous run is desired. Inside the critical region, there is a single instruction step. The operator must press the carriage return or space bar on the terminal keyboard after each statement executed.

Example 2: The trace region is set from 0 to 2000_{16}, with the trace format index number equal to 3. (Trace type 3 defaults to a full trace.) The snapshot prints workspace registers 1 through 4 and memory locations 1000_{16} to 1004_{16}. A breakpoint is set at 0474_{16} with snapshot 1 associated. A Modify Registers (MR) command sets the program counter to $046C_{16}$, and execution is begun by issuing an Execute User Program under SIE or Trace (RU) command.

```
.SR 1,0,2000,3,N

.SS 1,1,4,1000,1004
.SB 1,474,,1
.MR

PC=198C  46C
.RU
046C  8-02E0   ST=2000   WP=044C   SE=1968   SB=0900   SA=0900
0470  1-C2A0   ST=C000   SE=00A6   SB=1A92   SA=1A92   DE=0460
      DB=0000  DA=1A92
BKPT#1
PC=0474   WP=044C   ST=C000
SNAP1
R1=11C0   R2=0000   R3=0000   R4=0000
1000=10D8   C145   1305
0474  6-045A   ST=C000   BT=1A92   SE=1A92   SB=C2A0   SA=C2A0
1A92  1-C2A0   ST=2000   SE=00A8   SB=0000   SA=0000   DE=0460
      DB=1A92  DA=0000
1A96  6-0420   ST=2000   WP=1968   BT=198C   SE=1988   SB=1968
      SA=1968
198C  6-04C3   ST=2000   SE=196E   SB=FFFF   SA=0000
198E  1
.
```

Following is a listing of the portion of the program executed in this example with all references resolved:

Memory Location	Object Code		Source	
046C	02E0		LWPI	MAINW
046E	044C			
0470	C240		MOV	@ENTRY,R10
0472	00A6			
0474	045A		B	*R10
	.		.	
	.		.	
	.		.	
1A92	C2A0	INIT	MOV	@KBLUNO,R10
1A94	00A8			
1A96	0420		BLWP	@OPEN
1A98	1988			
	.		.	
	.		.	
	.		.	
1988	1968	OPEN	DATA	IOWKS
198A	198C		DATA	OPEN1
198C	04C3	OPEN1	CLR	R3

This is a typical example using snapshots, breakpoints and an instruction trace. Since a snapshot is associated with the breakpoint, the snapshot is printed and execution continued. An exit from the RU command is made by pressing the ESC key on the terminal keyboard.

9.6.12.3 Clear Trace Region (CR). The Clear Trace Region instruction is used to disable previously specified trace regions.

Syntax definition:

CR [{,/b...} [<starting trace region>] [{,/b...} <ending trace region>]]

The command is terminated by a carriage return.

Parameters:

starting trace region	The first trace region to be cleared. A number from 0 to 3.
ending trace region	The last trace region to be cleared. A number from 0 to 3.

Parameter default values:

If no parameters are specified, all trace regions are cleared.

If only the first parameter is given, only the specified trace region will be cleared.

If only the second parameter is given, trace regions 0 through the specified ending trace region will be cleared.

Error message:

DP13	**A trace region index greater than the maximum possible index number (3) was specified, or the ending region index was less than the starting region index number.**

Examples:

```
.CR 1,3

.CR
```

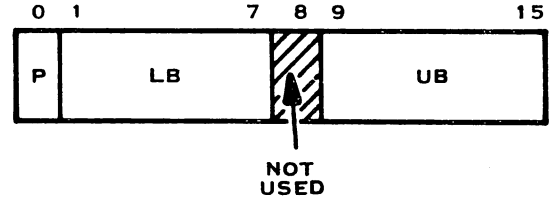
In the first example, all but region 0 are cleared. In the second example, all regions are cleared.

9.6.13 WRITE PROTECT OPTION COMMANDS (SP, CP)

These commands allow control of the optional hardware memory write protect feature on 990/4 computers.

9.6.13.1 Set Write Protect Region (SP). The Set Write Protect Region command sets the write protect region to the address specified in the command. This command is only valid if the user has a 990/4 computer with the write protect option. A protection violation generates a general interrupt signal which may be wired to any available interrupt level. Refer to the *Model 990/4 Computer Computer System Hardware Reference Manual* for the procedure for wiring a memory board to a desired interrupt level.

To set a write protect region, the lower and upper bounds must be output to CRU base address $1FA0_{16}$. The most significant bit (bit 0) is the Protect/Permit bit. Bit 0, when set to 1, indicates write permit, and, when set to 0, indicates write protect. To specify the protect region, memory is divided into 256-word blocks. The lower and upper bounds are each seven bits long and serve as an index into the memory addresses to specify which contiguous 256-word block of memory is to be protected. For example, the lower bound of the protect region equal to 2000_{16} would be represented in the Protect register as 10_{16}. The memory block beginning at location 2000_{16} is the sixteenth 256-word (512-byte) memory block. A bound is calculated by dividing the starting address of the memory block by 200_{16} (512_{10}). In this example, 2000_{16} divided by 200_{16} is equal to 10_{16}. The upper bound is not included in the protect region. When outputting to the CRU Protect register to specify the protect bounds, a Load CRU (LDCR) instruction with a count of 16 must be used to set all 16 bits because the Protect register works like a shift register. To protect the memory range 2000_{16} to 4000_{16}, the lower bound is set equal to 10_{16}, the upper bound is set to 20_{16}, and the Protect bit is set to 0. Therefore, the Protect register is set to 1020_{16} by outputting these fields to the CRU in the format specified in figure 9-1A.



BIT FIELDS

P	PROTECT/PERMIT BIT 0=PROTECT 1=PERMIT
LB	LOWER BOUND
UB	UPPER BOUND

NOTES

THE CRU OUTPUT DATA FORMAT IS THE SAME AS THE FORMAT OF DATA IN MEMORY BEFORE IN LDCR INSTRUCTION IS EXECUTED.

BITS 1 AND 9 ARE THE MOST SIGNIFICANT BITS, AND BITS 7 AND 15 ARE THE LEAST SIGNIFICANT BITS OF THE LB AND UB FIELDS.

(A)133373

Figure 9-1A. CRU Output Data Format

When an attempt is made to write into a memory location within the protected region, the Protect Violation flag is set to $FFFF_{16}$. This flag, which is normally 0, can be sensed by reading any of the 16 CRU bits at base $1FA0_{16}$. If this protected region is within the TMS9900 on-board RAM, the write is not inhibited. If this protect region is on the expansion memory card, the write is inhibited.

The Protect Violation flag may be cleared in two different ways:

1. I/O RESET (RSET) – This machine instruction clears the violation flag and sets bit 0 of the Protect register to 1 (not protected).

2. Output a 1 to any or all of the 16 bits of the Protect register.

If the user has wired his system such that a write protection violation causes an interrupt at a certain level, he must initialize the trap vector for that level and process the interrupt. The level 2 trap vector is initialized automatically by the Debug Monitor. The user may take advantage of this fact and wire his memory board interrupt to level 2. The system then prints:

MX06

when a protection violation occurs. When this happens, a RSET instruction is executed and the user must reestablish the protect bounds before starting execution again.

Syntax definition:

SP $\left\{\begin{matrix}\text{b}\\ \text{'}\\ \ldots\end{matrix}\right\}$ <lower mem addr> $\left\{\begin{matrix}\text{b}\\ \text{'}\\ \ldots\end{matrix}\right\}$ <upper mem addr>

The command is terminated by a carriage return.

When the user issues an SP 0600,0800 and then an EX command, his program begins execution. Should the user program then attempt to write into memory location 0700, hardware write protect sets the protection violation flag in the CRU and interrupts the CPU if the user has wired that interrupt.

Parameters:

lower mem addr	Lower boundary memory address of the protected region. Required parameter. Hexadecimal number.
upper mem addr	Upper boundary memory address of the protected region. Required parameter. Hexadecimal number.

Description: This command sets the write protect region from the lower to the upper memory bound addresses. If the memory addresses entered are not on 256-word boundaries, the bounds will be set at the next lower 256-word boundary. The lower bound is included within the protect region but the upper bound is not.

The SP command overrides any previously defined protect region.

When the upper and lower bounds are sent to the CRU, the Protect Violation flag is cleared if it has been set.

Error message:

MS05 Parameter specification error. Either a required parameter is missing, or the lower bound is greater than or equal to the upper bound.

Application note: This command is ignored if the write protect option is not implemented in the system hardware.

Examples:

```
.SP 1000,2000
```

This command protects a region in memory from 1000_{16} to $1FFF_{16}$.

```
.SP 1000,1F00
```

This command protects a region from 1000_{16} to $1DFF_{16}$. The address $1F00_{16}$ is not a 256-word boundary; therefore, the upper bound is set at the next lower 256-word boundary, 1E00.

9.6.13.2 Clear Write Protect Region (CP). The Clear Write Protect Region command clears the protect register and removes protection from the write-protected region.

Syntax definition:

```
CP
```

The command is terminated by a carriage return.

Description: The CP command clears the Protect register and sets the Protect/Permit bit to Permit. The Protect Violation flag is cleared if it has been set.

Application note: This command is ignored if the write protect option is not implemented in the system hardware.

Example:

```
.CP
```

9.7 DEBUGGING TECHNIQUES

Debugging techniques may be divided into three basic categories:

1. *Preventive techniques* – those which may be used to decrease the number of errors. Most of these techniques emphasize simplicity. Code should be simple and straightforward enough to make it obvious that the program works.

2. *Exposure techniques* – those which may be used to make the operation of a program easier to follow during the debugging process.

3. *Remedial techniques* – those used when a bug occurs in the user's program. Typically, most programmers' efforts are expended on these techniques.

Programming effort devoted to avoiding errors or making them apparent is important. Debugging and maintenance represent the majority of the cost in software development and support. The following paragraphs briefly discuss debugging in general and the specifics of debugging under TXDBUG.

9.7.1 GENERAL DEBUGGING TECHNIQUES. Several debug techniques will be helpful to the programmer in any debugging situation. These paragraphs offer some suggestions about debugging a program under development.

9.7.1.1 Debug Code in the Source Program. Include debug code in the source program. The user should keep the testing process in mind from the moment he starts to create a program. When referencing or changing data, the programmer should consider how to tell if the change is correct when reconstructing the results of a run. This often involves being aware of what intermediate results of a computation are lost.

For example, if the value of a variable D is calculated by the statement

```
D = A + B
```

and the program later encounters the statement

```
D = C + D
```

the second statement will cause a new value D to replace the previously calculated value. The calculated sum A + B will therefore be lost. If, on the other hand, the program contains the statement

E = A + B

and, later in the program, the statement

D = C + E

the value of E will be preserved when D is calculated by the second statement. The programmer can examine the memory location containing the value of E to determine the calculated sum A + B.

After a computation is completed, reconstruction of the results of a program run involves distinguishing which decision paths have been taken through the program's code and determining what variables are relevant in calculating the results of a computation.

When the source code is written, it is often simple to store intermediate results in extra memory to record those results, branch paths, or the number of passes through loops. Such statements can be flagged with a character string (e.g., **DEBUG**) in the comment field. When the source code is ready for production, TXEDIT can be used to locate and remove the code that stores intermediate results.

9.7.1.2 **Checking the Program.** Once a program has been successfully assembled, a thorough check of the program can often turn up errors which are hard to detect when the program is executing. In addition to making sure that the program is a correct implementation of the algorithm, it is often worthwhile to read through the program looking for specific errors:

- *Register errors.* Using the wrong register; referencing a register not in the current workspace; using a register as an immediate value (e.g., AI R1,R2 instead of A R1,R2 or AI R1,2); using byte-level operations or data where the data is in the wrong half of the register; or using byte-level data with the other half of the register containing incorrect data which affects the computation.

- *Variable names.* Misspelling of variable names such as T0 and TO; or using a single variable to contain different quantities.

- *Initialization errors.* Referencing values which may not have been properly initialized. This often occurs when a program is re-executed.

- *Buffer initialization.* Omitting an instruction to clear an input buffer between input operations when variable length records are read into a common fixed-length buffer.

- *Branch conditions and loop terminations.* Using the wrong branch instruction (especially JH, JL, JGT, JLE, JLT, JHE, or JOC with subtracts); or executing a loop one time too many or one time too few.

- *Inconsistent techniques.* Using conventions or debug elements which are inconsistent with the coding practice for the module.

- *Module interfaces.* Using variables or parameters which were not correctly set up for an interface; using registers or variables within a subroutine which have values that are not to be changed within the calling routine.

- *Boundary conditions.* Checking that the full range of the possible input data to a computation is correctly processed by the algorithm.

9.7.1.3 **Execution Tree.** In debugging or testing a program, it is often convenient to visualize the possible paths through the program as a tree with each node of the tree representing a conditional branch. Exhaustive testing of a program would then require testing each possible path through the program under all inputs which follow that path. While it is impossible to test all paths of a typical program, examination of the various paths (or small sets of paths) may reveal errors in the original logic.

9.7.2 **SPECIFIC DEBUGGING TECHNIQUES.** The following paragraphs describe techniques directed specifically to debugging under the debug monitor.

9.7.2.1 **Planning the Debugging Session.** Know the status of the debugging effort at all times. As the user interacts with the program through the console, he should be careful to record any changes made to the program and to be aware of the state of the program when examining it. In a debugging session, the user should have a clear idea of what he wants to accomplish and how he intends to accomplish it. Decisions made in the process of debugging should be carefully thought out.

9.7.2.2 **Use of Breakpoints.** There are three ways of stopping or interrupting the execution of a user's program which is being debugged at a specific location in the program:

1. Set an instruction count on the RUN command.

2. Execute with the single step option under instruction trace.

3. Set appropriate breakpoints.

Breakpoints stop execution at specific points in the user program rather than at arbitrary points controlled by the instruction count. The user may easily determine in advance and check the results of a computation without concerning himself about the state of the program.

When using breakpoints, be sure that the program will actually reach the desired breakpoint. This may involve putting additional breakpoints on the other paths from conditional branches.

Breakpoints are particularly useful when forcing some condition within a program which is not easily created from its parameters, for example, a CRU input. As an illustration of such a condition, an input value is to be read from a pressure transducer in an on-line process control environment. However, if the program is being debugged, a physically connected transducer is usually impractical and the values must be entered by the programmer. Breakpoints may be set prior to the start of a code sequence. When the breakpoint is taken, the user may set or modify the existing conditions in order to cause specific paths to be taken (as if a specific input had been received from the transducer).

The breakpoint reference count can be used to see that a loop is repeated the correct number of times. By setting the reference count equal to the number of iterations through the loop and setting another breakpoint outside the loop, the user may check that the loop is exhausted on the correct iteration. Breakpoints with attached snapshots with dump debug data or key variables yield a good trace aimed at checking the specific progress of a computation.

9.7.2.3 **Excluding Loops from Instruction Traces.** When tracing a program with printout, it is sometimes desirable to exclude printing of small loops which are very frequently executed or which run for many iterations. (See figure 9-2.) These may be excluded by carefully choosing trace regions, which are areas where an instruction trace is to be run within a program. In determining which trace region is applicable (and thus what trace type to use), the system will find the first (lowest numbered) region containing the user's PC. By selecting a high numbered trace (3) for the main trace control and then setting regions within that large region with lower numbered traces which do not print, the user may prevent a large quantity of output where it is not wanted.

An alternate mechanism is to allow the small loops to be executed by SIE and the remaining program traced. (See figure 9-3.) This can be done by setting trace regions to cover all of the program except the small loops or frequently executed parts. Such a mechanism works well unless the user is using XOPs (other than XOP 15 for debug monitor I/O) or interrupts which are processed differently by SIE and instruction trace.

If the user is performing I/O by means of supervisor calls (XOP 15), this XOP is executed directly (without SIE or instruction trace). If XOP 15 is not used for program I/O, it is executed directly under SIE.

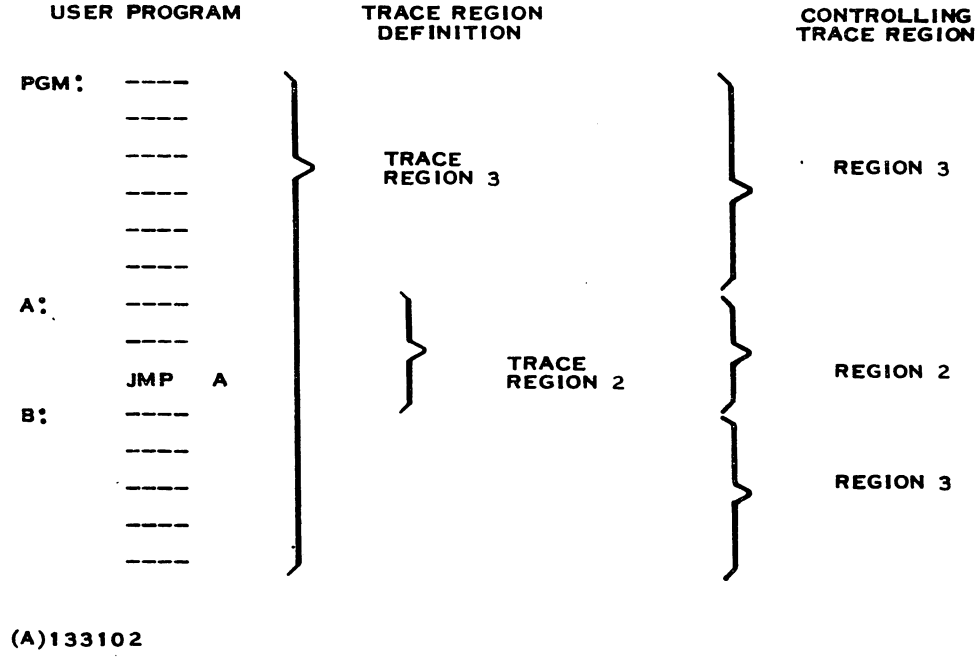


Figure 9-2. Trace Region Precedence of Lower Region Number

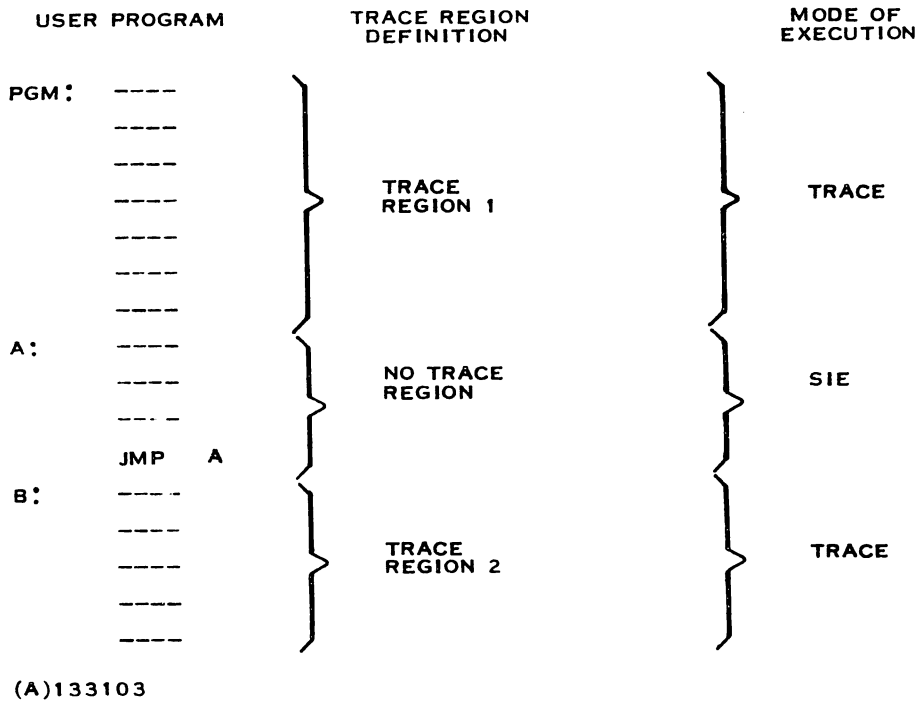


Figure 9-3. Using Both Trace and SIE

9.7.2.4 **Simulating an Interrupt.** A BLWP instruction may be used to control an interrupt routine which is being checked out. This can be handled with the following code sequence. The quantity "i" is the value to which "INTLVL" has been equated.

Instruction	Operand	Generated Code
LIMI	INTLVL	0300 i
BLWP	@INTLVL*4	0420 4*i
JMP	$	10FF

The LIMI sets the interrupt status to the correct level. The BLWP transfers control through the interrupt vector.

9.7.3 **PATCHING.** Patching (attaching portions of code to existing program code) should be avoided if possible.

During a debug session, it is generally necessary to make patches to object code; however, it is advisable never to leave patches in a completed program (or create ROM firmware from a program with patches). An object program for which there is no corresponding source program is inconvenient and troublesome.

The following paragraphs cover patching techniques. The examples show how to patch a two-address instruction; this instruction is used:

```
MOV *R1,*R2+
```

Because of the number of items to be considered, patching a two-address instruction is one of the more difficult operations. There are two ways to approach it: building a bit image and the additive method.

9.7.3.1 **Patching by Building a Bit Image.** In building a bit image, the user merely fills in each field in the 16-bit word on a bit-by-bit basis. When all fields are complete, the value is converted to hexadecimal for the patch contents.

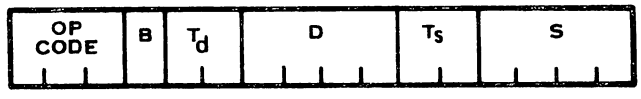
Example:

Patch the following assembly language instruction:

```
MOV *R1,*R2+
```

by building a bit image.

The MOV instruction has this format:



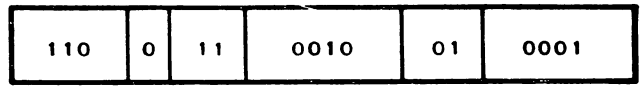
Determine the bits that occupy each field. Starting with the op code field, the hexadecimal op code for a MOV instruction is C000. The first three bits of this op code are 110_2; transfer these bits into the op code field.

The Byte Indicator (B) field specifies whether or not the instruction is a byte instruction. The MOV instruction is a word instruction; therefore, this field is set to 0. (The B field is always 0 for a MOV instruction.) Another way of specifying the same information would be to use the MOV or MOVB instruction (as appropriate) and a four-bit op code.

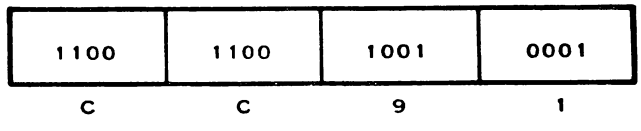
The D field specifies the destination workspace register. The destination address is *R2+, which indicates workspace register 2 and the workspace register indirect autoincrement addressing mode. The addressing mode for the destination, 11_2, is placed in the T_d field. Transfer the binary value of the register number, 0010_2, into the D field.

Use a similar procedure for the source address, which is *R1. In this case, workspace register 1 is specified and the addressing mode is workspace register indirect. Therefore, transfer 01_2 into the T_s field and 0001_2 into the S field.

The instruction field contents will now be:



Now read these 16 bits as a four-digit hexadecimal number.



The resulting hexadecimal number is the desired value. The patch value is CC91.

9.7.3.2 **Patching by the Additive Method.** The second approach to the patching problem is the additive method. With a little practice, the patch described in the first approach can be created a little faster by treating each of the fields as a hexadecimal number and adding the results to produce the patch.

Example:

Patch the same assembly language instruction as in the bit image example:

```
MOV *R1,*R2+
```

by using the additive method. This method involves adding hexadecimal values corresponding to each field to the instruction's op code to get the patch value.

The programmer can think of a bit field value as being placed into the instruction word, right justified, and shifted left the number of bits necessary to move it to the appropriate field. This shift is equivalent to binary multiplication, so the bit field value times an appropriate multiplier will give a value to be added to similarly obtained values for other bit fields to yield a sum representing the contents of the instruction word.

Recall that the values for the addressing modes and workspace registers in the previous examples were:

Destination mode (T_d)	3
Destination register (D)	2
Source mode (T_s)	1
Source register (S)	1

In calculating the patch value by the additive method, these values are used.

The first number in the calculation is the hexadecimal op code for the MOV instruction, C000. The B field is always 0 in the MOV instruction; it can be considered part of the instruction op code and ignored in the calculation.

The second number to be added is the value of the destination mode. The code for the address mode is shifted left ten bits, equivalent to multiplication by 400_{16}. The code is 3_{16}; therefore, the value to be added is

$$3_{16} * 400_{16} = 0C00_{16}$$

The third number is the destination register value. To create the value to be added, the register number, 2_{16}, is shifted left six bits, equivalent to multiplication by 40_{16}. The value is

$$2_{16} * 40_{16} = 0080_{16}$$

Calculation of the fourth value involves a code of 1_{16} for the source mode and a four-bit shift (multiplication by 10_{16}). The value is

$$1_{16} * 10_{16} = 0010_{16}$$

Finally, the source register number, 1_{16}, is unshifted. The value to be added is 0001_{16}.

To calculate the required sum, the values are added:

Op code of MOV instruction	C000
Destination mode	0C00
Destination register	0080
Source mode	0010
Source register	0001
Patch value	CC91

The sum, $CC91_{16}$, is the object code to be patched. The patch value is the same as the value obtained in the previous example.

When the same instruction format is used repeatedly, the multiplication constants – 400_{16}, 40_{16} and 10_{16} – do not change and become simple to handle with practice.

9.7.3.3 **Symbolic Versus Indexed Addressing.** The address mode for both symbolic (actual memory address) and register indexed addressing is the same (mode 10_2). The type of addressing is determined by the register field. A register field of zero is symbolic; therefore, no R0 indexing exists. In constructing a patch with a specific address, process it exactly as if it were a register indexed with a register of zero. Refer to the *Model 990 Computer TMS9900 Microprocessor Assembly Language Programmer's Guide*, Manual No. 943441-9701, for further information about symbolic and indexed memory addressing.

9.7.3.4 **Branch Distance Calculations for Jump Instructions.** The signed displacement in an Unconditional Jump (JMP) instruction is a two's complement eight-bit number which represents the number of words to skip forward or backward from the current PC (the PC points to the instruction *following* the jump instruction).

To calculate the displacement for a jump instruction, evaluate

1/2 (target location-(instruction address+2)).

If the target address is less than the instruction address, add 10000_{16} to the target address and perform the subtraction. Note that a forward branch must generate a positive displacement and a backward branch must generate a negative displacement to be in range.

Example 1:

Patch location $17A_{16}$ with a jump to location $1FE_{16}$.

The source address is equal to the instruction address +2, which is 17A+2 = 17C.

The target location minus the source address is 1FE - 17C = 82. Continuing,

1/2 (target location - source address) = 41

The displacement, 41, is positive. The patch value is therefore 1041_{16}, where 10 is the hexadecimal op code for the JMP instruction and 41 is the displacement value.

Example 2:

Patch Location $1FE_{16}$ with a jump to location $17A_{16}$.

The source address is equal to the instruction address+2, which is $1FE_{16}+2_{16} = 200_{16}$. The sum of the target location plus 10000_{16}, minus the source address, is $1017A_{16}-200_{16} = FF7A_{16}$. Continuing

1/2 (target location - source address) = 7FBD = BD (dropping the first two digits)

The displacement, BD, is negative. The patch value is therefore $10BD_{16}$, where 10 is the hexadecimal op code for the JMP instruction and BD_{16} is the displacement value, negative in this case.

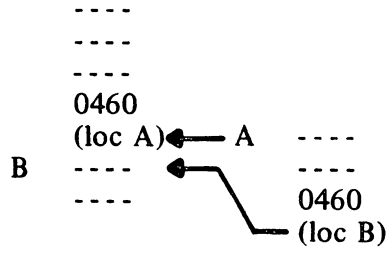
Note that the 7F is generated from the addition of 2_{16} (10000_{16}) and may be discarded. If the high order eight bits of the destination are not equal to 7F, the branch distance is too great to reach with a JMP instruction.

9.7.3.5 Use of Spin and No-operation. It is sometimes convenient to patch a spin (branch to itself) into a location to intercept control in unexpected situations (the alternate path of a conditional jump, for example). That instruction is a JMP to itself and is a value of $10FF_{16}$. (The corresponding assembly language code is JMP $.)

Unwanted instructions can be replaced with a No-Operation (NOP) which is a JMP to the next instruction. The value for an NOP is 1000_{16}. Strings of NOPs may also be placed at various locations in the program source to reserve space for temporary debug patches.

9.7.3.6 Out-of-Line Patches. It is often necessary to patch more instructions into a program than there is room, requiring an out-of-line patch. The simplest mechanism is to use a symbolic address branch instruction to a specific location where the patch is placed. After the patch, use a branch instruction back to the original code.

Example:



Be careful to see that code which is overlayed is moved to the patch area, that it is not a PC relative jump, and that the return pointer comes to the beginning of an instruction.

9.8 ERROR MESSAGES

TXDBUG may issue any of the following error messages:

Message	Meaning
MX01	Unrecoverable I/O error
MX06	Invalid memory address or instruction
MS01	Invalid command
MS05	Required parameter missing
MP00	Parameter specification error
DP00	Invalid hexadecimal number input
DP03	Parameter value is greater than the allowed maximum
DP04	Snapshot is already defined
DP10	Invalid trace region index
DP12	CRU bit width parameter invalid
DP13	Invalid range of registers or memory addresses
DP20	Breakpoint specification error
DP23	Syntax error in trace format character string
DP26	Invalid trace format index number

In addition, during the initial TXDBUG load, the TX990 program loader may issue the following error messages:

Message	Meaning
LDFE	Load bias error
LDFF	Get common error (system error)
LDXX	All other load errors are of the form (LD(XX) where XX is the TXDS I/O error code received

SECTION X

TXDS PROM (TXPROM) PROGRAMMER UTILITY PROGRAM

10.1 INTRODUCTION

This section describes the TXPROM programmer utility program along with the required hardware and software . In addition, it describes the function and use of control files, bit string mapping of PROMs, and examples of the use of the utility, as well as instructions for loading and operating the utility. For standard operations, refer to the loading and operating procedures contained in paragraphs 10.4 and 10.5, plus the description of standard control files found in paragraph 10.7. The description portion of this section is also helpful for operation of the utility. For custom mapped PROMs or PROMs with nonstandard data configurations, read all the information contained in this section. For further information regarding PROM programming with a 990 Computer System, refer to the following related publications:

Title	Part Number
Model 990 Computer TMS9900 Microprocessor Assembly Language Programmer's Guide	943441-9701
Model 990 Computer PROM Programming Module Installation and Operation	945258-9701
Model 990 Computer AMPL Microprocessor Prototyping Laboratory Operation Guide	946244-9701

10.2 REQUIRED CONFIGURATION

The TXPROM programmer utility program requires the following configuration for proper operation:

- An FS990 System
- A Model 990 PROM Programming Unit.

The TXPROM programmer utility software is part of the TX990/TXDS system software and is packaged on a diskette. TXPROM includes the following files:

- :TXPROM/ – contains the PROM programming software.
- A set of standard control files – :S288, :S287, :S471, :S472, :E2704B, :E2704, :E2708B, :E2708, :E2716B and :E2716.

10.3 DESCRIPTION

The TXPROM programmer utility is a software module that controls a computer hardware system to create custom Read Only Memories (ROMs). The hardware system can program either Programmable Read Only Memory devices (PROMs) or Erasable Programmable Read Only Memory devices (EPROMs). Throughout this section, the term PROM refers to either of these devices unless it specifically excludes one of them. TXPROM is part of the Terminal Executive Development System that runs under the TX990 Operating System.

Functions performed by TXPROM include:

- Copying data from a file to a PROM
- Storing data from a PROM into memory or a file
- Displaying a disc file in PROM format
- Comparing data contained in a PROM with that contained in a file, and indicating any discrepancies.

TXPROM uses predefined control information to store data in or read data from PROM devices. Included with the utility is a set of standard control files that contain the control information for reading and programming PROMs that employ the memory configuration used in the 990 Computer Family. For other applications, the user can modify these control files or create new files using the information supplied in this section.

10.3.1 PROM BURN AND VERIFY. Three steps are required to transfer data from a data file into a PROM. As illustrated in figure 10-1, these steps are:

1) Load control and data information from a diskette into separate areas in computer memory.

2) Use the information contained in the control area of memory to direct the transfer of data to the PROM to burn-in the data.

3) Use the information in the control area of memory to read the contents of the newly programmed ROM and compare the contents of the ROM with the contents of the data in the memory buffer area in memory.

The TXPROM software performs the second and third steps after having been instructed to do so by the operator. The user must, therefore, adequately prepare both the data file and the control file to ensure that TXPROM accurately transfers the data to the PROM. The requirements of each of these files are explained later in this section of the manual.

10.3.2 PROM READ OPERATION. A PROM read operation requires TXPROM to perform two steps, as illustrated in figure 10-2:

1) Use the information contained in a control file to read data from a ROM and store the data in the memory buffer area.

2) Store the information from the memory buffer area into a diskette data file as directed by the user.

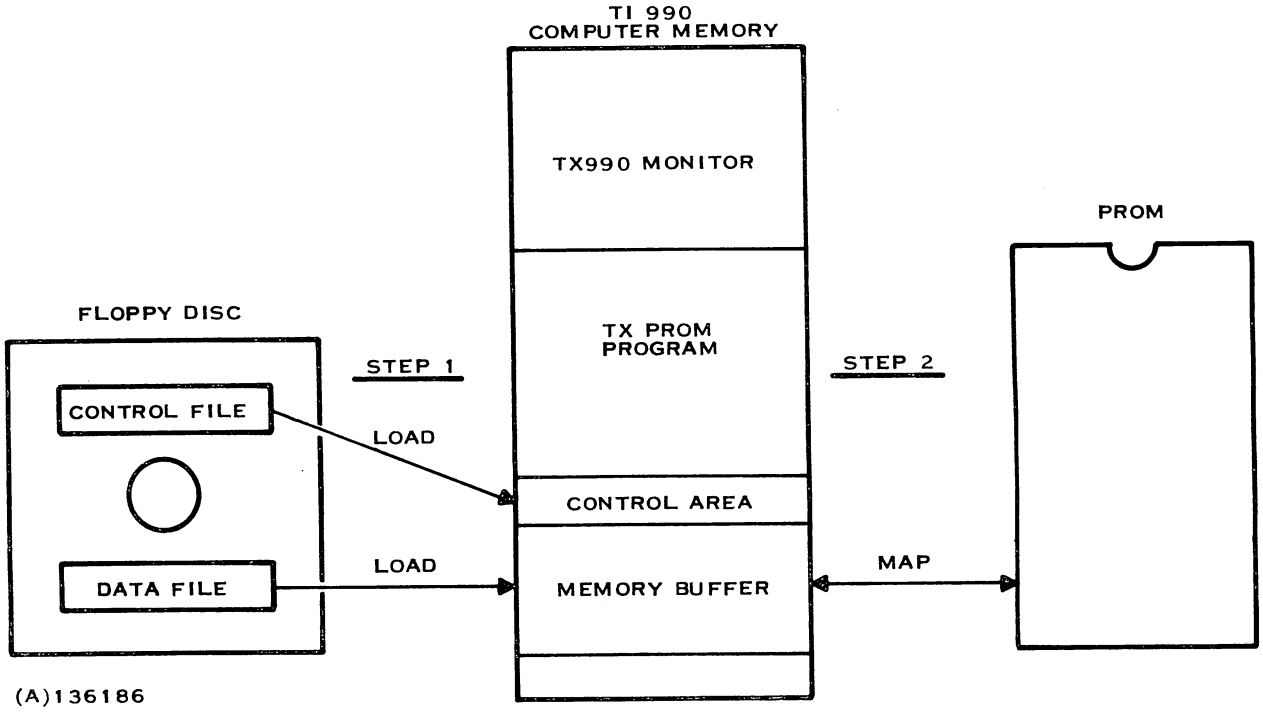


Figure 10-1. PROM Burn, Compare Operation

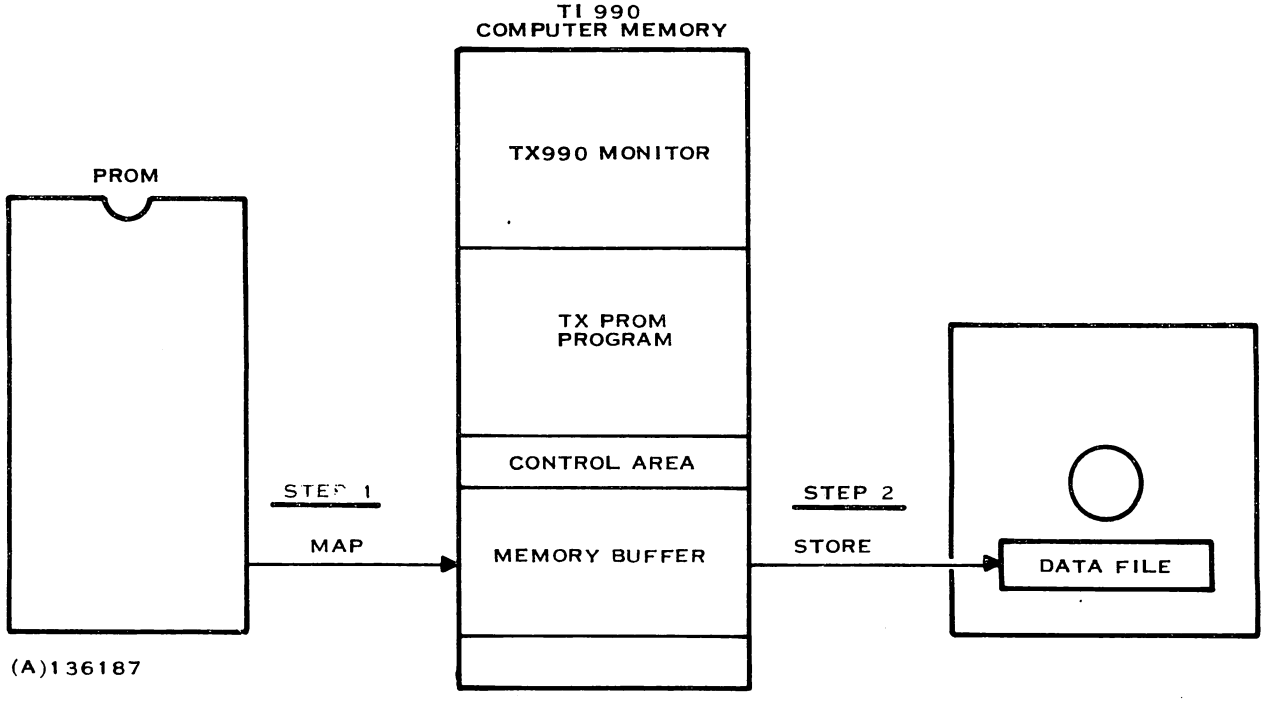


Figure 10-2. PROM Burn, Compare and Read Operation

The transfer of data from the memory buffer to the data file is performed on a word-for-word basis in binary-object format. The previous contents of the data file are lost. The data from the PROM can be read into memory without being stored in an output file by specifying "DUMY" as the output file. This method can be used for preliminary inspection of ROM data, as well as for data file formatting. For example, to read data from four 4 X 256 PROMs and store it in a 256-word file, the following steps could be used:

1) Read the first three half-bytes (4-bit transfers) into the memory buffer using a read operation with DUMY as the output file. This stores the first twelve bits in memory.

2) Read the fourth half-byte into the memory buffer with a read operation that specifies the desired output file. The complete 16-bit word is transferred to the output file.

10.3.3 LUNOS USED. TXPROM assigns LUNO AA_{16} to the control file (see paragraph 10.5).

10.4 LOADING TXPROM

TXPROM is loaded under direction of the TXDS control program. Before loading TXPROM, the diskette containing the software must be inserted into a drive unit and that unit prepared for operation. Since the control program searches all system drives for the requested file, the diskette need not be loaded on a specific drive in multiple drive systems. When initiated, the control program produces the following prompt on the system console:

```
PROGRAM:
```

To load TXPROM, respond to this prompt as follows:

```
PROGRAM: :TXPROM/*<carriage return>
```

The control program then locates the file containing TXPROM, loads it into memory, and begins execution of TXPROM. Input and output operations will then be directed to the system console during execution of TXPROM. Any other interactive device supported by TXDS may be used. To specify a different device for interaction with TXPROM, respond to the PROGRAM: prompt as follows:

```
PROGRAM: :TXPROM/*[device]*<carriage return>
```

10.5 TXPROM OPERATION

When TXPROM is successfully loaded, it prints the following identification and prompt on the selected interactive device:

```
TXPROM PROM PROGRAMMER UTILITY 937569  *A
CONTROL FILE =
```

The response to this prompt determines which of three modes of operation that TXPROM will enter: control file creation, control file modification, or control file execution. The following paragraphs describe the three modes of operation, provide a general procedure for performing each function, and illustrate each mode with an example.

10.5.1 CONTROL FILE CREATION. The control file creation mode allows the user to create a new control file for a custom application after determining that none of the standard control files satisfies the requirements. The mode is entered by pressing the carriage return key in response to the CONTROL FILE = prompt. In this mode, TXPROM outputs each control file parameter prompt in order, followed by an asterisk (*). The asterisk indicates that the parameter is a variable that must be supplied when the control file is executed. If the parameter is to remain a variable,

press the carriage return key on the terminal to move to the next parameter prompt. If the parameter is to be a predetermined value, type that value and press the carriage return key. The entered value becomes the default value for that parameter. Entered values must be in decimal unless specified otherwise by one of the following prefixes:

<	binary
!	octal
>	hexadecimal

For example, to enter an octal loop count of 400_8, the prompt and response appear as:

MEM LEV 1 LOOP CNT* !400

At any point the remaining parameter prompts may be bypassed, leaving them as variable parameters, by pressing the ^ (caret) key. TXPROM then proceeds with the file creation mode termination sequence. This sequence is entered either by pressing the caret key or by completing consideration of all parameter prompts. TXPROM then produces the prompt:

SAVE UNDER FILE NAME =

Entering a floppy disc file name in response to this prompt and then pressing the carriage return causes TXPROM to create a control file with the specified name. That file name can than be used to call the newly created control file for execution. Standard control files are write-protected and cannot be altered. Therefore, choose a file name other than a standard control file name for newly created or modified files. Entering only a carriage return in response to the above prompt creates no new file. The parameters remain in memory until modified, a control file name other than DUMY is specified, or TXPROM is terminated. The parameters can be accessed by referencing DUMY as the desired control file for execution or modification.

When the new file name is determined and the carriage return is entered, TXPROM issues the following prompt:

EXECUTE, BEGIN OR TERMINATE

The responses to this prompt are as follows (letters in parentheses are optional):

EX(ECUTE)	Switch to execution mode and use the newly entered control file parameters for the operation.
BE(GIN)	Restart the TXPROM sequence by returning to the CONTROL FILE = prompt.
TE(RMINATE)	Return to the TXDS control program.

10.5.2 **CONTROL FILE MODIFICATION**. The control file modification mode allows the user to modify the contents of a previously created control file. The mode is entered by responding to the CONTROL FILE = prompt with the name of an existing control file without including a parameter list. TXPROM then responds with the prompt:

MODIFY OR EXECUTE?

Entering the following response places TXPROM in the modification mode (letters in parentheses are optional):

MO(DIFY)

TXPROM then produces the parameter prompts for the control file information as it does in control file creation mode, except that the prompts are followed by the existing values for the parameters. Asterisks indicate variable parameters that must be defined at execution time. The parameters can be changed by typing in the desired value in place of the existing value following each prompt and then pressing a carriage return. The resulting modified control file can replace the original file (if the original file is not a standard control file), can be saved in a new control file, or can be saved in memory only for immediate execution depending upon the response to the SAVE UNDER FILE NAME = prompt. As in the creation mode, pressing the ^ (caret) key at any time skips over the remaining parameters without changing their values. When the SAVE UNDER FILE NAME = prompt has been satisfied and a carriage return entered, TXPROM again produces the following prompt:

EXECUTE, BEGIN OR TERMINATE

Responses to this prompt are identical to those for creation mode.

10.5.3 CONTROL FILE EXECUTION. The control file execution mode allows the user to program a PROM using the parameters in an existing control file. The control file may be one of the supplied standard control files, or a custom generated file produced using the file creation mode of TXPROM. The execution mode is entered by responding to the CONTROL FILE = prompt with the name of an existing control file, or by responding with the name of an existing control file and its parameter list. If only a control file name is specified (without the parameter list), TXPROM responds with the prompt:

MODIFY OR EXECUTE?

Entering the following response places TXPROM in the execute mode (letters in parentheses are optional):

EX(ECUTE)

A parameter list is not required because TXPROM generates prompts for all missing parameters. However, if parameters are included, they must be in the order specified in table 10-1. The parameter list contains a string of values separated by commas and enclosed in parentheses following the file name. For example, the following reply to the CONTROL FILE = prompt illustrates the parameter list:

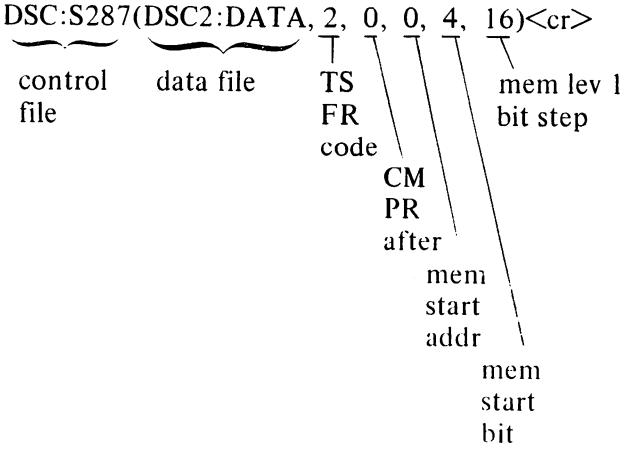


Table 10-1. Table of Control File Parameter Prompts

Parameter Prompt	Possible Value	Description
DATA FILE =	TX990 Pathname	Name of data file
DATA BIAS =	*	Value to add to relocatable code in object modules
TSFR CODE =	0 to 2	Transfer code: 0 nothing, 1 burn PROM, 2 read PROM
CMPR AFTER =	0, 1	Compare after: 0 nothing, 1 compare PROM and memory
MEM DISP =	0, 1	Memory display: 0 nothing, 1 display memory
PROM DISP =	0, 1	PROM display: 0 nothing, 1 display PROM
MEM START ADDR =	*	Memory bound low (address)
# MEM BYTES =	*	Memory bytes to be transferred
MEM START BIT =	*	Memory beginning bit
PROM START ADDR =	*	PROM bound low (address)
# PROM WORDS =	*	PROM words to be transferred
PROM START BIT =	*	PROM beginning bit
**MEM MAP LEVELS =	1 to 3	Number of memory mapping levels
MEM LEV 1 BIT STEP =	0 to $7FFF_{16}$	Number of bits skipped between loops
LOOP COUNT =	1 to 32,767	Number of repetitions of loop 1
2 BIT STEP =	0 to $7FFF_{16}$	
LOOP COUNT =	0 to 32,767	
3 BIT STEP =	0 to $7FFF_{16}$	
LOOP COUNT =	1 to 32,767	
**PROM MAP LEVELS =	1 to 3	Number of PROM mapping levels
PROM LEV 1 BIT STEP =	0 to $7FFF_{16}$	Number of bits skipped between loops
LOOP COUNT =	1 to 32,767	Number of repetitions of loop 1
2 BIT STEP =	0 to $7FFF_{16}$	
LOOP COUNT =	1 to 32,767	
3 BIT STEP =	0 to $7FFF_{16}$	
LOOP COUNT =	1 to 32,767	
TSFR BIT WIDTH =	1 to 8	Transfer bit string width
PROM BITS/WORD =	1 to 8	
PROG 0's OR 1's =	0, 1	Program zero's or one's
PULSE WIDTH =	1 to 6	Programming pulse width
DUTY CYCLE =	0 to 100	% of time used in program device
NO. RETRIES =	0 to $FFFF_{16}$	Number of retries
SIMUL PROG'BLE BITS =	1 to 8	Number of simultaneously programmable bits
CRU BASE =	0 to $1FFE_{16}$	Base CRU address for PROM interface card

*Any value 0 to $7FFF_{16}$; however, some parameters interact with each other to create other limitations. See text.

All numeric parameters are expected to be in decimal notation, but can be in octal, binary or hexadecimal if preceded by the proper prefix as described in the file creation description. When TXPROM enters the execute mode, it scans the contents of the control file and selects the required variable parameters. If a variable parameter list was supplied, the supplied values are filled into the control file data. If additional values are required or if no list was supplied, then TXPROM generates prompts for each required parameter.

TXPROM then checks all parameters for boundary violations. If any value is out of bounds, TXPROM generates a prompt for that value to be changed by the user.

When all parameters have been verified, the control file is executed. All interrupts are disabled during actual data transfer between PROM and memory. When execution is complete, TXPROM generates the following messages:

```
SUCCESSFUL EXECUTION
REPEAT, BEGIN OR TERMINATE?
```

Proper responses to this message are as follows (letters in parentheses are optional):

RE(PEAT)	Repeat the execution process (for burning more than one PROM)
BE(GIN)	Return to the CONTROL FILE = prompt at the start of TXPROM
TE(RMINATE)	Return to the TXDS Control program.

10.6 DATA FILES

TXPROM uses data files to store formatted data on diskette or in computer memory. The data is in object format as described in the Assembly Language Programmer's Guide. The data in the files may be burned into a PROM or compared to the data already in a PROM. Data files are created either by an assembly, by the link editor, or by reading a PROM and storing the contents in a file. When the data is transferred from the file to a PROM, the data is treated as a series of ascending addressed locations each 16 bits long. Each 16-bit word is selected from the file according to control parameters in the control file (Memory Starting Address Number of Memory Bytes, Memory Start Bit, Memory Level n Bit Step, and Memory n Level Loop Count). The data may then be transferred to the PROM according to other control file parameters so that each bit in the data file can be stored separately in the PROM.

10.7 CONTROL FILES

TXPROM uses control files to determine the pattern that data in data files will be stored in a PROM. The data is not necessarily transferred to the PROM as an exact image of the data file. Instead, the parameters of the control file allow each bit, or group of bits, of the data file to be mapped to a separate location in the PROM. Table 10-1 lists each of the parameters in the control file along with the range of values for each parameter. In the file creation phase, TXPROM produces control file parameter prompts for user response. No default values exist during creation mode. In the file modification phase of a PROM programming sequence, TXPROM allows the user to change the control file parameters after issuing the prompts listed in the table. The user can then select the default value with a carriage return or enter a new value. The default values for each standard control file are listed later in this section. The following paragraphs describe the use and function of each control file parameter.

10.7.1 DATA FILE NAME. The data file name is an alphanumeric parameter that specifies the name of the floppy disc data file to be used during the current operation. The file name may also be the name of an input file from a 733 ASR cassette drive; however, the cassette cannot be used as an output file to store information from a PROM read operation because the output format is not cassette compatible. To indicate that no data file is to be used, enter the file name, DUMY.

This specification allows information to be read from a PROM into memory without being stored in a data file. During the initiation sequence, TXPROM allows the user to enter the data file name after it issues the following prompt:

DATA FILE =

No default value exists for this parameter in the standard control files.

10.7.2 **DATA BIAS.** The data bias parameter allows a pre-existing object module to be loaded into memory at a simulated load point that is displaced (biased) from the normal load point of zero. The actual load point in memory of the file is unaffected by this parameter. Typically, the data bias is the same as the base address of the data in the target system in which the PROM is to be used. The value of the data bias is added to each word that is marked as relocatable by the assembler or link editor. TXPROM accesses the data as if it were loaded in memory, starting at the data bias value. Therefore, the memory starting address parameter must be consistent with the addressing used in the biased file. For example, a program that is 1000_{16} bytes long and is loaded with a data bias of 500_{16} must have its memory starting address parameter within the range of 500_{16} to $14FF_{16}$. TXPROM allows the user to enter the data bias after it issues the following prompt:

DATA BIAS =

The default value for this parameter in the standard control files is zero (no displacement).

10.7.3 **TRANSFER CODE.** The transfer code parameter defines the operation to be performed with the PROM device. The code is one of the following three values:

0 No operation
1 Transfer data from specified data file to PROM
2 Read data from PROM and store in specified data file

TXPROM allows the user to enter the transfer code parameter after it issues the following prompt:

TSFR CODE =

The default value for this parameter in control files :E2704B, :E2708B, and :E2716B is 1 (PROM burn operation). There is no default value for the other standard control files.

10.7.4 **COMPARE AFTER.** The compare after parameter allows the user to enable (1) or disable (0) a comparison of the PROM data with the data file data following either a burn or a read operation. If the comparison is successful, TXPROM proceeds to the next operation. If the comparison fails, TXPROM displays the memory byte address, the PROM address, and the two bit strings in the following format:

>Mxxxx.yy=zz Raaaa.bb=cc

Refer to the description of memory display and PROM display later in this section for an explanation of the display formats. TXPROM allows the user to enter the compare after parameter after it issues the following prompt:

CMPR AFTER =

The default value for this parameter in control files :E2704B, :E2708B, and :E2716B is 0 (disable comparison). There is no default value for the other standard control files.

10.7.5 **MEMORY DISPLAY.** The memory display parameter allows the user to select a display of the memory data file on the data terminal being used. This parameter may be either a 1 to enable memory display, or a 0 to inhibit memory display. If the memory display parameter is equal to a 1, the memory region containing the data file is displayed in the following format:

Mxxxx. yy=zz

In this notation, the letters have the following significance:

M = Designates a memory display

xxxx = Memory byte address

yy = Displacement of start of bit string within memory byte ($0 \leqslant yy \leqslant 7$)

zz = The value of the bit string in hexadecimal notation when right-justified within an 8-bit field.

A maximum of four entries are displayed on each output line of the terminal. For example, a memory display value of:

M000B. 00=5A

indicates that the bit string at byte address $000B_{16}$ that begins with the first bit of that byte has a value of $5A_{16}$. TXPROM allows the user to enter the memory display parameter after it issues the following prompt:

MEM DISP =

The default value for this parameter in the standard control files is zero (no display).

10.7.6 **PROM DISPLAY.** The PROM display parameter allows the user to select a display of the PROM contents being burned or read. The display appears on the data terminal being used to initiate TXPROM during the execution of the program. This parameter may be either a 1 to enable PROM display, or a 0 to inhibit PROM display. If the PROM display parameter is equal to a 1, the PROM region is displayed in the following format:

Raaaa. bb=cc

In this notation, the letters have the following significance:

R = Designates a ROM or PROM display

aaaa = PROM/ROM word address

bb = Displacement of start of bit string within PROM word ($0 \leqslant bb \leqslant 7$)

cc = The value of the bit string in hexadecimal notation when right-justified within an 8-bit field.

A maximum of four entries are displayed on each output line of the terminal. For example, a PROM display of

R00E1. 00=7A

indicates that the bit string at PROM word address $00E1_{16}$, that begins the first bit of that byte, has a value of $7A_{16}$. TXPROM allows the user to enter the PROM display parameter after it issues the following prompt:

PROM DISP =

The default value for this parameter in the standard control files is zero (no display).

10.7.7 MEMORY STARTING ADDRESS. The memory starting address parameter indicates the starting address in memory of the first bit string to be transferred to the PROM or to be read from the PROM. If the object module is relocatable, the memory starting address is an absolute memory address. TXPROM allows the user to change the memory starting address parameter by producing the following prompt:

MEM START ADDR =

No default value exists for this parameter in the standard control files.

10.7.8 NUMBER OF MEMORY BYTES. This parameter indicates the number of bytes to be transferred from or to memory during the PROM operation. TXPROM adds this value to the memory starting address to create a range of addresses in memory for the transfer operation. If TXPROM tries to access a bit string outside this range of addresses, an error is indicated. TXPROM allows the user to change the memory bytes parameter by producing the following prompt:

MEM BYTES =

The default value for this parameter varies with the particular standard control file.

10.7.9 MEMORY STARTING BIT. This parameter indicates the starting bit address relative to the starting byte (indicated by memory starting address) of the bit string to be transferred during the operation. The value of this parameter may be any positive magnitude; however, if the value exceeds 7, the starting bit will be located beyond the starting byte indicated by the memory starting address. TXPROM allows the user to enter the memory starting bit after it issues the following prompt:

MEM START BIT =

No default value exists for this parameter in the standard control files.

10.7.10 PROM STARTING ADDRESS. The PROM starting address parameter indicates the starting word address in PROM of the first bit string to be burned or to be read. TXPROM allows the user to enter the PROM starting address after it issues the following prompt:

PROM START ADDR =

The default value for this parameter in the standard control files if 0.

10.7.11 **NUMBER OF PROM WORDS.** This parameter indicates the number of PROM words that will be processed during the current operation. TXPROM adds this value to the PROM starting address to create a range of addresses in PROM for the transfer operation. If TXPROM tries to access a bit string outside this range of addresses, an error is indicated. TXPROM allows the user to enter the PROM words parameter after it issues the following prompt:

PROM WORDS =

The default value for this parameter varies with the particular standard control file.

10.7.12 **PROM STARTING BIT.** This parameter indicates the starting bit address relative to the starting word address (indicated by PROM starting address) of the bit string to be processed. The value of this parameter may be any positive magnitude; however, if the value exceeds the word size for the PROM device type being used, the starting bit is located beyond the starting word indicated by the PROM starting address. TXPROM allows the user to enter the PROM starting bit after it issues the following prompt:

PROM START BIT =

The default value for this parameter in the standard control files is 0.

10.7.13 **MEMORY MAPPING LEVELS.** The memory mapping levels parameter specifies the number of loop levels to be used in mapping data from memory into the PROM device. The number of levels may be 1, 2 or 3. Refer to the discussion of Bit String Mapping later in this section for complete information about the use of this parameter. If this parameter is 1, then the loop count for levels 2 and 3 are automatically set to 1. TXPROM allows the user to enter the memory mapping levels parameter after it issues the following prompt:

**MEM MAP LEVELS =

Enter a value of 1 for all standard control files except :E2704B, :E2708B, and :E2716B. For these files, enter a value of 2.

10.7.14 **MEMORY LEVEL n BIT STEP.** This parameter determines the number of bits that are skipped between successive bit addresses when performing a level n (n = 1, 2, or 3) mapping loop. For example, to access only the even-numbered bits (or the odd-number bits) this parameter is set to a value of 1. This value causes a skip of one bit between each bit accessed. TXPROM allows the user to enter this parameter for each of the three possible mapping levels after it issues the following prompt(s) (only the prompts for the number of levels selected in the memory mapping levels parameter are produced):

MEM LEV 1 BIT STEP =

or

MEM LEV 2 BIT STEP =

or

MEM LEV 3 BIT STEP =

The default value for the level 2 and 3 parameters in the standard control files is zero. There is no default value for the level 1 parameter in the standard control files.

10.7.15 MEMORY LEVEL n LOOP COUNT. This parameter determines the number of iterations that are performed of the level n (n=1, 2 or 3) mapping loop. The value may be within the range of 0 to $FFFF_{16}$. TXPROM allows the user to enter this parameter for each of the three possible mapping levels after it issues the following prompts (only the prompts for the number of levels selected in the memory mapping levels parameter are produced; all other levels are set to one:

MEM LEV 1 LOOP COUNT =

or

MEM LEV 2 LOOP COUNT =

or

MEM LEV 3 LOOP COUNT =

The default value for each of these parameters in the standard control files varies with the selected control file.

10.7.16 PROM MAPPING LEVELS. The PROM mapping levels parameter specifies the number of loop levels to be used when mapping data into the PROM. The number of levels may be 1, 2 or 3. Refer to the discussion of Bit String Mapping later in this section for complete information about the use of this parameter. If this parameter is 1, then the loop count for levels 2 and 3 are automatically set to 1. TXPROM allows the user to enter the PROM mapping levels parameter after it issues the following prompt:

**PROM MAP LEVELS =

When responding to this prompt, enter a value of 1 for each standard control file except :E2704B, :E2708B, and :E2716B. These files require a response of 2.

10.7.17 PROM LEVEL n BIT STEP. This parameter determines the number of bits that are skipped between successive bit addresses when performing a level n (n = 1, 2 or 3) mapping loop. For example, to burn every other bit in a PROM (either the odd or even bit addresses) this parameter is set to a value of 1. This value causes a skip of one bit between each bit operated on in the PROM. TXPROM allows the user to enter this parameter for each of the three possible mapping levels after it issues the following prompt(s) (only the prompts for the number of levels selected in the PROM mapping levels parameters are produced):

PROM LEV 1 BIT STEP =

or

PROM LEV 2 BIT STEP =

or

PROM LEV 3 BIT STEP =

The default value for level 1 in the standard control file :S287 is 4; all other standard control files have a default value of 8.

The default value for levels 2 and 3 for this parameter in the standard control files is zero. This value causes TXPROM to access every consecutive bit in the PROM.

10.7.18 PROM LEVEL n LOOP COUNT. This parameter determines the number of iterations that are performed of the level n (n = 1, 2, or 3) mapping loop. The value may be any number from 1 to 32767. TXPROM allows the user to enter this parameter for each of the three possible PROM mapping levels after it issues the following prompt(s) (only the prompts for the number of levels selected in the PROM mapping levels parameter are produced; all other levels are set to one):

PROM LEV 1 LOOP COUNT =

or

PROM LEV 2 LOOP COUNT =

or

PROM LEV 3 LOOP COUNT =

The default value for each of these parameters in the standard control files varies with the selected control file.

10.7.19 TRANSFER BIT WIDTH. The transfer bit width designates the number of bits that are to be transferred in each bit string. This parameter applies to both the memory and the PROM portions of the operation. TXPROM allows the user to change the transfer bit width by producing the following prompt:

TSFR BIT WIDTH =

The default value for this parameter in standard control file :S287 is 4; all other standard control files have a default value of 8.

10.7.20 PROM BITS PER WORD. This parameter specifies the number of bits in each word of the PROM device being used. It should match the architecture of the PROM device. TXPROM allows the user to change this parameter by producing the following prompt:

PROM BITS/WORD =

The default value for this parameter in standard control file :S287 is 4; all other standard control files have a default value of 8.

10.7.21 PROGRAM ZEROS OR ONES. This parameter indicates whether the PROM device begins as all zeros and must be programmed by burning ones, or if it begins as all ones and must be programmed by burning zeros. The *PROM Programmer Installation and Operation Manual* contains a table of initial conditions for all devices that can be programmed with that unit. This parameter should be set to a 1 if a high-level programming pulse (programmed with ones) is required, and to a 0 if a low-level programming pulse (programmed with zeros) is required. TXPROM allows the user to change this parameter by producing the following prompt:

PROG 0'S OR 1'S =

The default value for this parameter in the standard control files varies with the control file selected.

10.7.22 PULSE WIDTH. The pulse width parameter is a code that designates the duration of the programming pulse to be used with the selected PROM device. Table 10-2 lists and defines these codes. Table 10-3 lists the programming pulses required for some commonly used PROM devices. The pulse width is the length of time that power is applied to the PROM device to burn simultaneously programmable bits. TXPROM allows the user to change the value of this parameter by producing the following prompt:

PULSE WIDTH =

The default value for this parameter in the standard control files varies with the control file selected.

Table 10-2. Pulse Widths

Pulse Width Code	Pulse Width (ms)
1	0.5
2	1.0
3	2.0
4	4.0
5	8.0
6	16.0

Pulse Width=2^{code}*(.25)ms

Table 10-3. Minimum, Standard and Maximum Pulse Widths and Duty Cycles

PROM Types	Pulse Width (ms) Minimum	Standard	Maximum	Duty Cycle Minimum	Standard	Maximum
TTL						
188A, S188, S288, S287, S387, S470, S471, S472, S473	1	2	20		25%	35%
EPROMs						
2704, 2708, 2716	0.1	0.1	1		50%	50%

Note: TTL PROM types have the prefix SN74.

10.7.23 **DUTY CYCLE.** The duty cycle parameter indicates the percentage of the programming cycle time that it actually used for burning the PROM. The total programming cycle consists of a programming (burn) phase and a rest phase. The duty cycle value (between 0 and 100) represents the maximum percentage of total time that the programming pulse can be active. Table 10-3 lists the duty cycle requirements of some commonly used PROM devices. TXPROM allows the user to change this parameter to match the requirements of the PROM device being used by producing the following prompt:

DUTY CYCLE =

The default value for this parameter in the standard control files is 25 for PROMs and 50 for EPROMs.

10.7.24 **NUMBER OF RETRIES.** This parameter indicates the number of times that TXPROM will try to program a specific set of bits without success using the normal pulse width. If the first attempt to program a set of bits in a PROM device fails. TXPROM repeats the programming cycle for that set of bits until the correct data is transferred or the number of retries count is depleted. TXPROM allows the user to change this parameter by producing the following prompt:

NO. RETRIES =

The default value for this parameter in the standard control files is zero.

10.7.25 **SIMULTANEOUSLY PROGRAMMABLE BITS.** This parameter indicates the number of bits in the PROM device that can be programmed with the same programming pulse. This parameter is a physical restriction of the type of PROM device. Bipolar devices require that only one bit be programmed at a time; EPROMs require that an entire EPROM word be programmed simultaneously. TXPROM allows the user to change this parameter by producing the prompt:

SIMUL PROG'BLE BITS =

The default value for this parameter in the standard control files is 1 for PROMs and 8 for EPROMs.

10.7.26 **CRU BASE.** The CRU base parameter of the control file defines the CRU base address to be used to select the PROM Programmer interface card. For standard applications, the interface card responds to base address 20_{16}. If the interface card is installed in a chassis location other than the standard slot, the CRU base parameter must be changed. TXPROM allows the user to enter the CRU base after it issues the following prompt:

CRU BASE =

The default value for this parameter in the standard control files is 20_{16}.

10.8 BIT STRING MAPPING

The software uses the memory and PROM mapping parameters to determine the addresses of the bit strings to be used in the programming cycle. When specifying mapping parameters, the PROM or memory words within the defined bounds are considered to be a continuous string of bits. The memory file is further divided into 16-bit words, while the PROM string is divided into words whose length is determined by the architecture of the device. Mapping is required so that portions of the 16-bit memory words may be programmed into PROMs that have smaller word widths. The mapping parameters include bit step and loop count, as defined previously in this section.

TXPROM allows three levels of bit string mapping: level 1, level 2 and level 3. Level 1 determines successive bit strings in memory or PROM. When the level 1 loop count is exhausted, the initial bit

is incremented as determined by the level 2 bit step and the level 1 mapping is repeated. Each time that the level 1 loop count is exhausted, the level 2 loop count is decremented, the initial bit incremented, and the mapping repeated until the level 2 loop count is exhausted. At that point, the level 3 increment is added to the beginning address, the level 3 loop count is decremented, the loop counts for levels 1 and 2 are restored, and the entire cycle is repeated. When the level 3 loop count is exhausted, cycling is complete. A map cycle for memory bits is completely independent of a map cycle for PROM bits; however, the total number of bits that are mapped in the memory cycle must be equal to the number of bits mapped to a PROM.

10.8.1 **LEVEL 1 MAPPING EXAMPLE.** Figure 10-3 illustrates an example of level 1 mapping. In the example, the first four bits of each memory word are mapped into the odd-numbered addresses of a 256 X 4 PROM, (a 128 half-byte transfer). Table 10-4 lists the mapping parameters for both memory and PROM to accomplish the transfer.

10.8.2 **LEVEL 2 MAPPING EXAMPLE.** Figure 10-4 illustrates an example of level 2 mapping. In the example, the first and the last four bits of each memory word are mapped into a 256 X 8 PROM (a 256-byte transfer). Table 10-5 lists the mapping parameters for both memory and PROM to accomplish the transfer. The example combines level 2 memory looping with level 1 PROM looping.

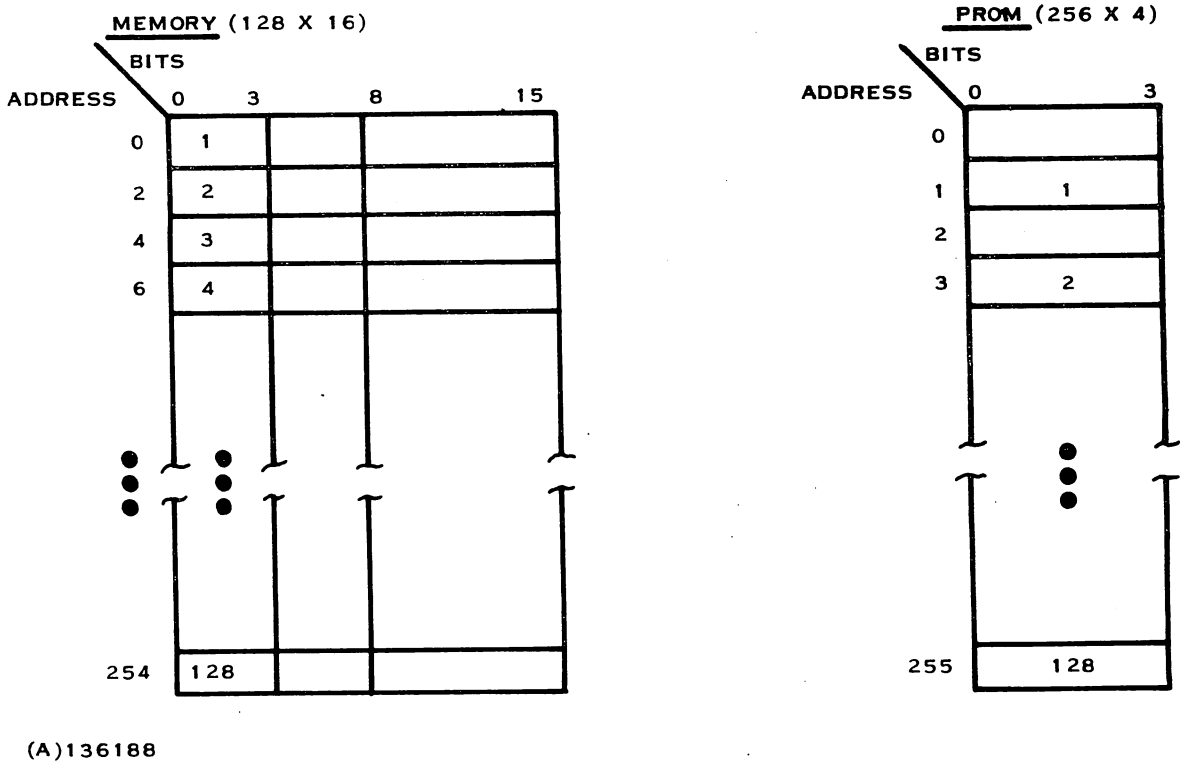


Figure 10-3. Level 1 Mapping Example

Table 10-4. Level 1 Mapping Example Parameters

File (Memory) Mapping Parameters	PROM Mapping Parameters
MEM START ADDR = 0	PROM START ADDR = 0
# MEM BYTES = 256	# PROM WORDS = 256
MEM START BIT = 0	PROM START BIT = 4
MEM LEV 1 BIT STEP = 16	PROM LEV 1 BIT STEP = 8
MEM LEV 1 LOOP CNT = 128	PROM LEV 1 LOOP CNT = 128
MEM LEV 2 BIT STEP = 0	PROM LEV 2 BIT STEP = 0
MEM LEV 2 LOOP CNT = 1	PROM LEV 2 LOOP CNT = 1
MEM LEV 3 BIT STEP = 0	PROM LEV 3 BIT STEP = 0
MEM LEV 3 LOOP CNT = 1	PROM LEV 3 LOOP CNT = 1

TRANSFER BIT WIDTH = 4

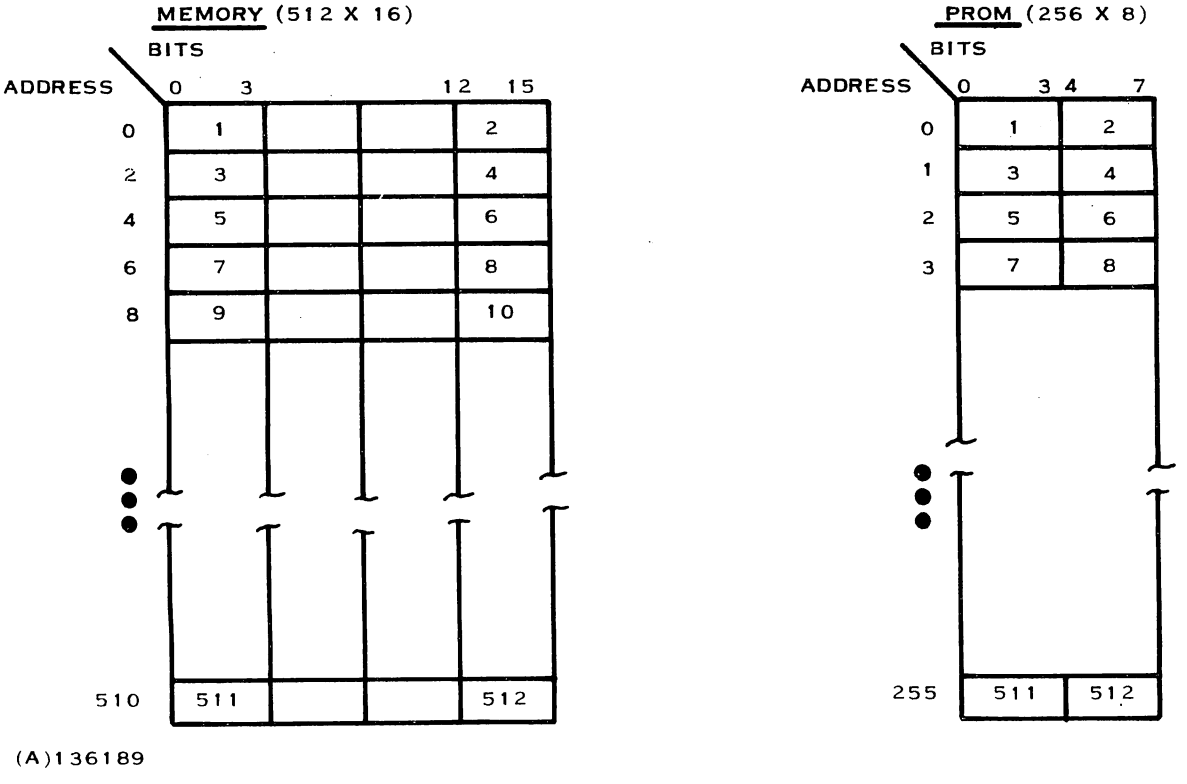


Figure 10-4. Level 2 Mapping Example

Table 10-5. Level 2 Mapping Example Parameters

File (Memory) Mapping Parameters	PROM Mapping Parameters
MEM START ADDR = 0	PROM START ADDR = 0
#MEM BYTES = 512	#PROM WORDS = 256
MEM START BIT = 0	PROM START BIT = 0
MEM LEV 1 BIT STEP = 12	PROM LEV 1 BIT STEP = 4
MEM LEV 1 LOOP CNT = 2	PROM LEV 1 LOOP CNT = 512
MEM LEV 2 BIT STEP = 16	PROM LEV 2 BIT STEP = 0
MEM LEV 2 LOOP CNT = 256	PROM LEV 2 LOOP CNT = 1
MEM LEV 3 BIT STEP = 0	PROM LEV 3 BIT STEP = 0
MEM LEV 3 LOOP CNT = 1	PROM LEV 3 LOOP CNT = 1

TRANSFER BIT WIDTH = 4

10.8.3 LEVEL 3 MAPPING EXAMPLE. Figure 10-5 illustrates an example of level 3 mapping. In the example, the first and the last four bits of each memory word are mapped into the first 256 words of a 1024 X 8 PROM. The mapping of the memory words is then repeated three more times to fill the 1024 words of the PROM. Table 10-6 lists the mapping parameters for both memory and PROM to accomplish the transfer.

10.9 STANDARD CONTROL FILES

The TXPROM software includes a set of standard control files. The files contain parameters that can be used without modification to program most PROM devices commonly used with the PROM programming system. Table 10-7 lists the standard control files along with their contents. For special applications, these files can also be used as the basis for building a custom control file, rather than creating a new file. EPROM devices have two control files: the file with the letter B suffix is for burn cycles and the file with no suffix is for reads. EPROM devices require repeated programming cycles to implant the charge. The EPROM "B" files automatically repeat the programming cycle to allow for this requirement. The standard control files reside on the same diskette as the TXPROM software.

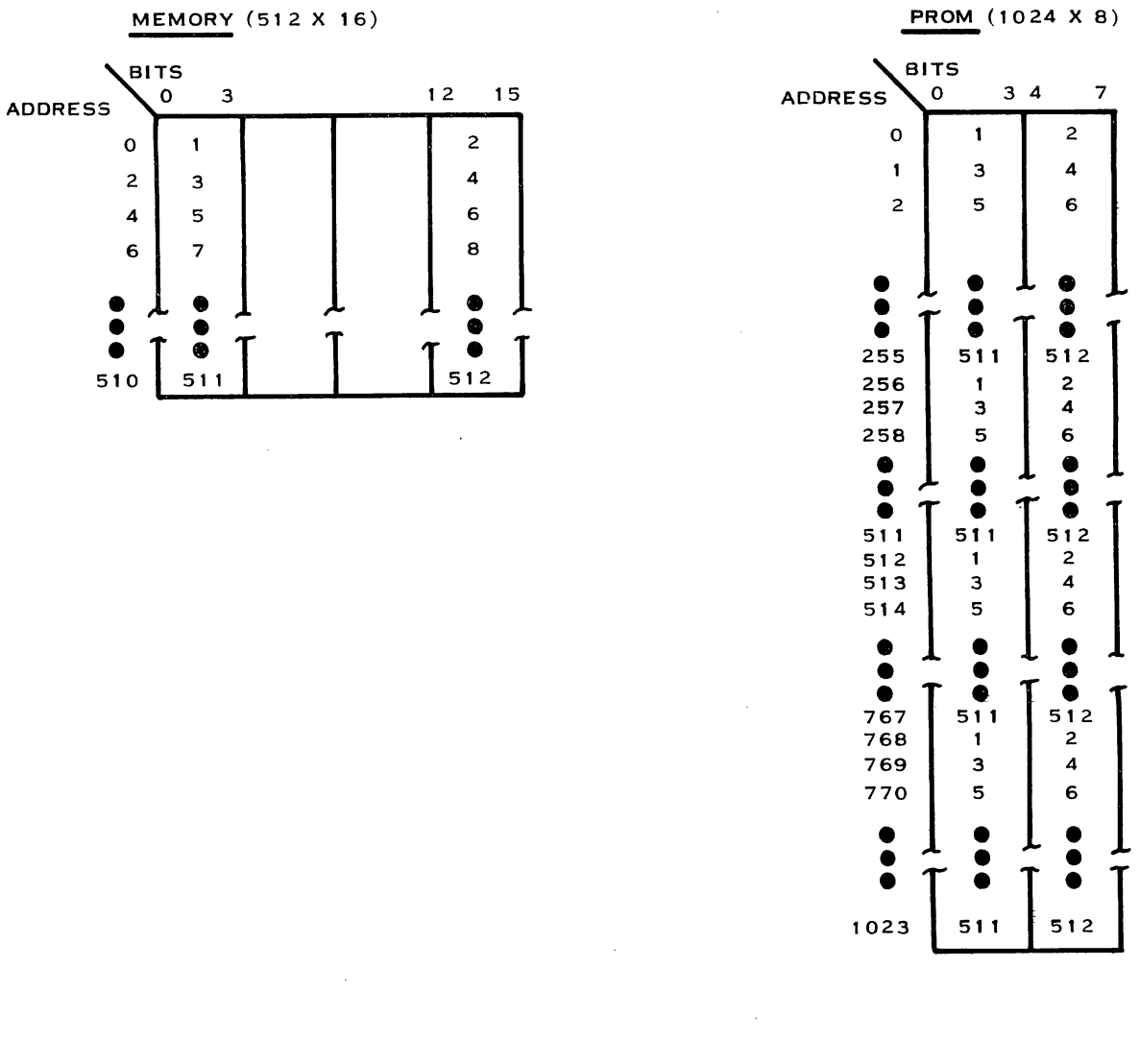
10.10 VARIABLE PARAMETERS

None, any, or all of the parameters in a control file can be made into variable parameters by entering a value of * for each parameter prompt when the control file is created or modified. The values for these parameters are not stored in the control file but must be entered in at execution time (whenever the control file name is requested). Variable parameters allow frequently changed parameters (like DATA FILE name) to be easily inserted, nonpermanently, into control file parameters.

For example, the Standard Control file for 74287 bipolar TTL PROM devices was created with the following variable parameters: DATA FILE, TSFR CODE, CMPR AFTER, MEM START ADDR, MEM START BIT, MEM LEV 1 BIT STEP. To use the standard control file, the user must respond to the control file prompt with:

DSC:S287(<data file>,<tsfr code>,<cmpr after>,<mem start addr>,<mem start bit>, <mem lev 1 bit step>).

The parameters inside the angle brackets, < >, must be supplied with the desired values. If no parameters are entered, TXPROM generates prompts to ask for the information.



(A)136190

Figure 10-5. Level 3 Mapping Example

Table 10-6. Level 3 Mapping Example Parameters

File (Memory) Mapping Parameters	PROM Mapping Parameters
MEM START ADDR = 0	PROM START ADDR = 0
# MEM BYTES = 512	# PROM WORDS = 1024
MEM START BIT = 0	PROM START BIT = 0
MEM LEV 1 BIT STEP = 12	PROM LEV 1 BIT STEP = 4
MEM LEV 1 LOOP CNT = 2	PROM LEV 1 LOOP CNT = 1024
MEM LEV 2 BIT STEP = 16	PROM LEV 2 BIT STEP = 0
MEM LEV 2 LOOP CNT = 256	PROM LEV 2 LOOP CNT = 1
MEM LEV 3 BIT STEP = 0	PROM LEV 3 BIT STEP = 0
MEM LEV 3 LOOP CNT = 4	PROM LEV 3 LOOP CNT = 1

TRANSFER BIT WIDTH = 4

10.11 PROGRAMMING EPROMS

Since EPROMs are metal-oxide-semiconductor (MOS) devices, they must be programmed in a different manner than TTL PROM devices. EPROMs are charge-storage devices that must be programmed by repetively transferring charge to EPROM bits. This repetition may be accomplished by looping through the programming process defined by the data configurations. The number of required repetitions to transfer sufficient change to each bit or bit string is defined by the following formula:

100 ms = pulse width x repetitions.

Therefore, using a pulse width of 0.5 ms, 200 repetitions must be used to successfully program the EPROM. A delay must occur after each attempt to program a bit string before trying to program the same bit string again. This delay allows the charge to diffuse into the EPROM device without a buildup of charge on the surface.

Because of this delay, each bit string of the EPROM should be attempted once before repeating the programming cycle. To ensure this delay, the number of retries parameter for programming each bit string (defined in the control file) must be set to zero. Each bit of the EPROM will not appear to have the correct value (0 or 1) until sufficient charge has been transferred to it.

In the early stages of programming, the bits may not have acquired sufficient charge to have the correct value. This appears as a programming failure if the number of retries is set to a nonzero value, and the bit string will be programmed again without the required delay time. For the same reason, the compare after parameter (defined in the control file) should not be set during the programming cycle, since compare errors will be found in the early stages of programming an EPROM.

Since the programming cycle for an EPROM repeats many times, the display parameter (defined by the control file) should not be set during the programming cycle. Setting the display parameters prints the memory or PROM data for each repetition. Therefore, to program, compare and display, the process must be done in two steps. First, the parameters must be set to zero to program, and after completion of EPROM programming, the parameters may be set to enable compare and/or display. The number of repetitions defined must be changed to one before the second step in order to compare and/or display.

Table 10-7. Standard Control Files

Standard Control File	:S288	:S287	:S471	:S472	:E2704B	:E2704	:E2708B	:E2708	:E2716B	:E2716
Data File	*	*	*	*	*	*	*	*	*	*
Data Bias	0	0	0	0	0	0	0	0	0	0
TSFR Code	*	*	*	*	1	*	1	*	1	*
CMPR After	*	*	*	*	0	*	0	*	0	*
MEM Disp	0	0	0	0	0	0	0	0	0	0
PROM Disp	0	0	0	0	0	0	0	0	0	0
MEM Start Addr	*	*	*	*	*	*	*	*	*	*
# MEM Bytes	64	512	512	1024	1024	1024	2048	2048	4096	4096
MEM Start Bit	*	*	*	*	*	*	*	*	*	*
PROM Start Addr	0	0	0	0	0	0	0	0	0	0
#PROM Words	32	256	256	512	512	512	1024	1024	2048	2048
PROM Start Bit	0	0	0	0	0	0	0	0	0	0
MEM Map Levels	‡	‡	‡	‡	‡	‡	‡	‡	‡	‡
MEM LEV 1 BIT STEP	*	*	*	*	*	*	*	*	*	*
Loop Count	32	256	256	512	512	512	1024	1024	2048	2048
MEM LEV 2 BIT STEP	0	0	0	0	0	0	0	0	0	0
Loop Count	1	1	1	1	200	1	200	1	200	1
MEM LEV 3 BIT STEP	0	0	0	0	0	0	0	0	0	0
Loop Count	1	1	1	1	1	1	1	1	1	1
PROM Map Levels	‡	‡	‡	‡	‡	‡	‡	‡	‡	‡
PROM LEV 1 BIT STEP	8	4	8	8	8	8	8	8	8	8
Loop Count	32	256	256	512	512	512	1024	1024	2048	2048
PROM LEV 2 BIT STEP	0	0	0	0	0	0	0	0	0	0
Loop Count	1	1	1	1	200	1	200	1	200	1
PROM LEV 3 BIT STEP	0	0	0	0	0	0	0	0	0	0
Loop Count	1	1	1	1	1	1	1	1	1	1
TSFR Bit Width	8	4	8	8	8	8	8	8	8	8
PROM Bits/Words	8	4	8	8	8	8	8	8	8	8
PROG 0's, PROG 1's	1	0	1	1	0	0	0	0	0	0
Pulse Width	2	2	2	2	1	1	1	1	1	1
Duty Cycle	25	25	25	25	50	50	50	50	50	50
Number Retries	0	0	0	0	0	0	0	0	0	0
SIMUL Prog'ble Bits	1	1	1	1	8	8	8	8	8	8
CRU Base	>20	>20	>20	>20	>20	>20	>20	>20	>20	>20

*Indicates variable parameters; i.e., value must be entered at execution time.
‡A response is required. Enter 2 for :E27048B, :E2708B, and :E2716B. Enter 1 for all other standard control files.

If level n bit string mapping is used to burn an EPROM, the level (n+1)'s bit step should be set to 0 and the loop count set to the desired number of repetitions. Note that the standard control files for EPROM burns have level 2 loop counts of 200.

10.12 PROGRAMMING EXAMPLES

The following paragraphs illustrate the control file requirements to successfully program a PROM or EPROM using TXPROM.

10.12.1 EPROM EXAMPLE. The following example programs an 8-word data file vertically to the first 16 locations of a 512 X 8 EPROM (2704) as illustrated in figure 10-6.

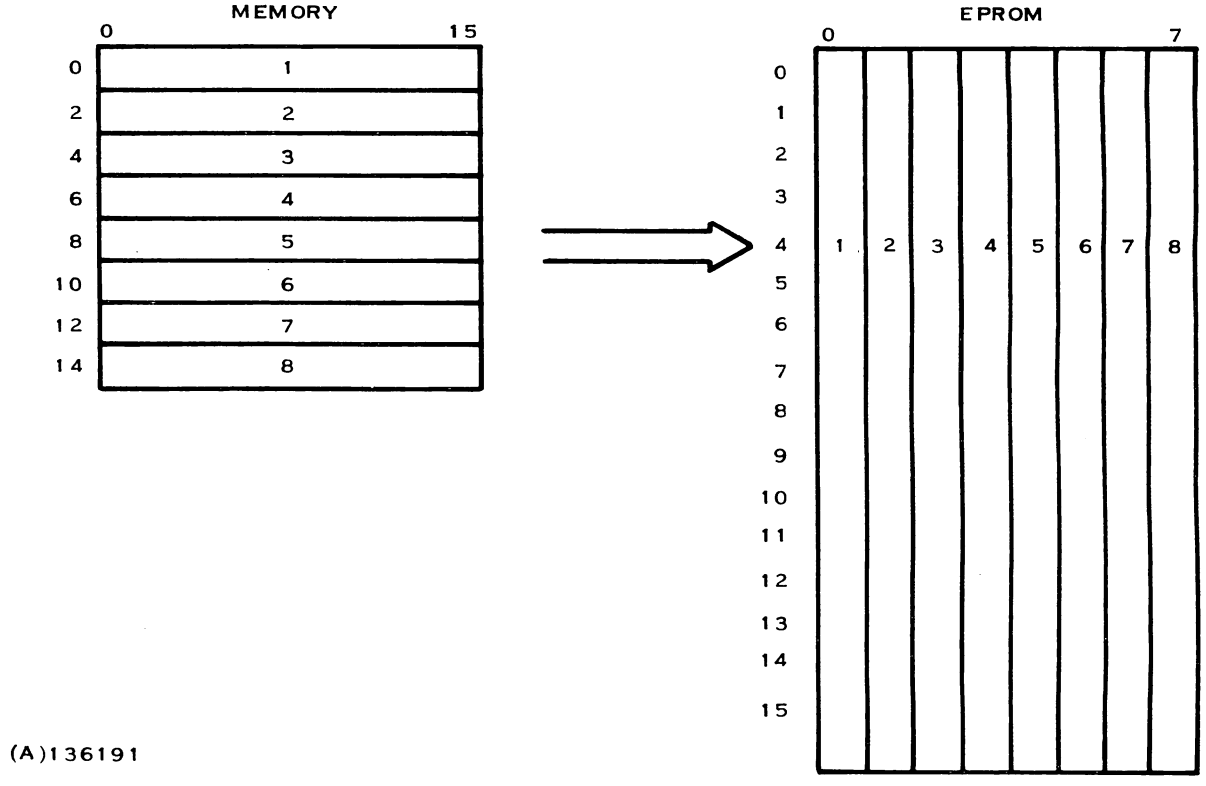


Figure 10-6. EPROM Programming Example

Bits are transferred one at a time from memory to the EPROM. The user creates a control file by modifying the E2704B standard control file. The following parameters are modified:

```
MEM MAP LEVELS = 2
MEM START ADDR = 0
# MEM BYTES = 16
MEM START BIT = 0
MEM LEV 1      BIT STEP = 1
               LOOP COUNT = 256
MEM LEV 2      BIT STEP = 0
               LOOP COUNT = 200
PROM LEVELS = 3
PROM START ADDR = 0
# PROM WORDS = 16
PROM START BIT = 0
PROM LEV 1     BIT STEP = 8          Burns 1 word of memory vertically
               LOOP COUNT = 16
PROM LEV 2     BIT STEP = 1          Positions to next column
               LOOP COUNT = 16
PROM LEV 3     BIT STEP = 0          200 repetitions since EPROM
               LOOP COUNT = 200

                TRANSFER BIT WIDTH = 1
```

10.12.2 **PROM PROGRAMMING EXAMPLE.** Twenty-four 4-bit fields are arranged in 16-bit words of a data file, as shown in figure 10-7. These 24 fields are to be programmed repetitively in the first 384 four-bit words of a 512 X 4 PROM with characteristics similar to a TI SN74S287 (two 287s with a programming adaptor card to make them appear as a 512 X 4 device) as illustrated in figure 10-7.

The user starts with the S287 standard control file and makes the following modifications:

File (Memory) Mapping Parameters

```
MEM START ADDR = 0
# MEM BYTES = 16
MEM START BIT = 0
MEM MAP LEVELS = 3
MEM LEV 1      BIT STEP = 6
MEM LEV 1      LOOP COUNT = 3
MEM LEV 2      BIT STEP = 16
MEM LEV 2      LOOP COUNT = 8
MEM LEV 3      BIT STEP = 0
MEM LEV 3      LOOP COUNT = 16
```

PROM Mapping Parameters

```
PROM START ADDR = 0
# PROM WORDS = 384
PROM START BIT = 0
PROM LEVELS = 1
PROM LEV 1     BIT STEP = 4
PROM LEV 1     LOOP COUNT - 384
```

TRANSFER BIT WIDTH = 4

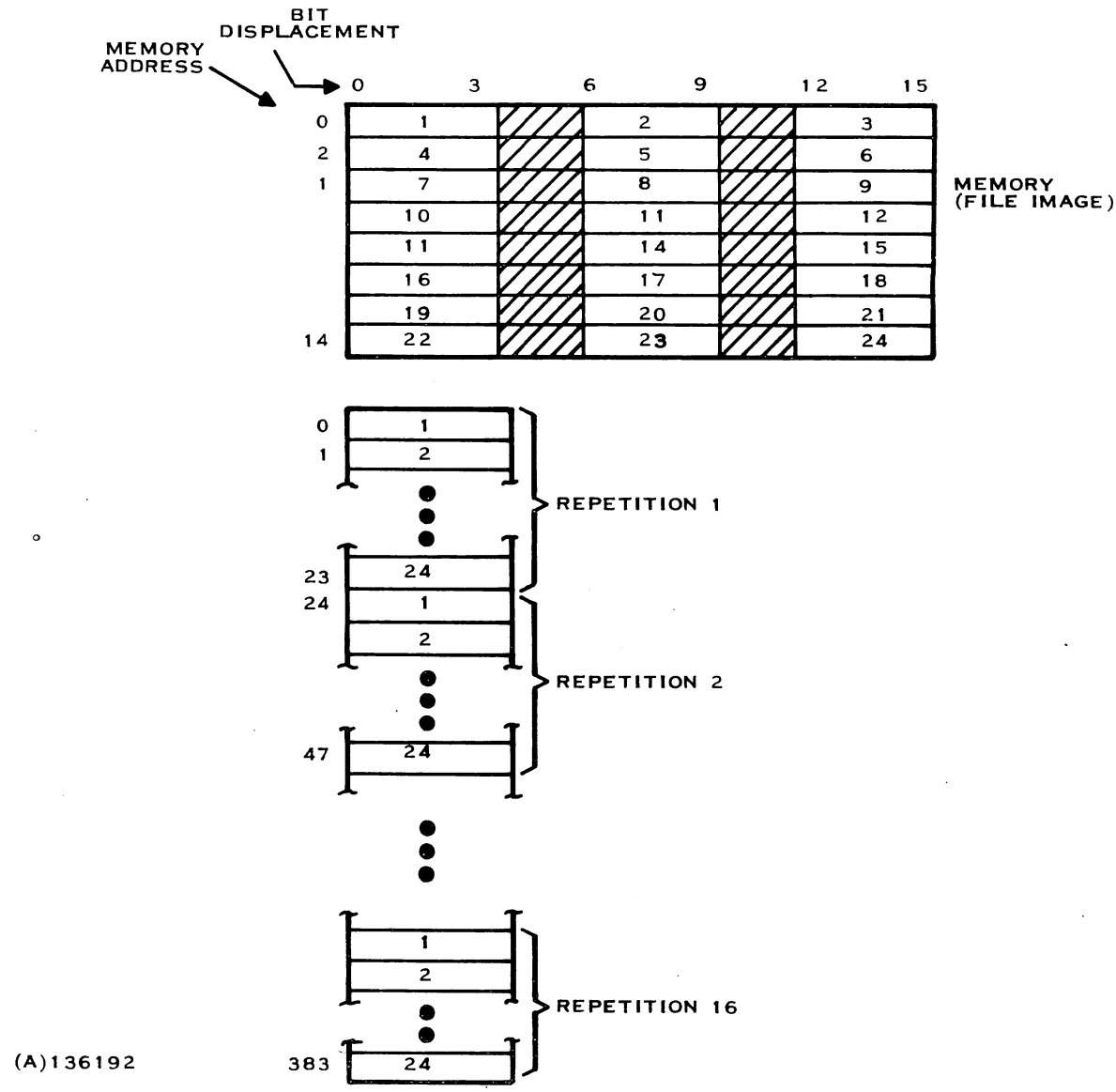


Figure 10-7. PROM Programming Example

10.12.3 **CONTROL FILE CHANGE EXAMPLE.** The user wishes to change the S471 standard control file so that the parameter DATA BIAS is a variable parameter. The user does the following:

```
CONTROL FILE = DSC:S471<cr>
MODIFY OR EXECUTE?     MO<cr>
MODIFICATION MODE

DATA FILE = *<cr>
DATA BIAS = 0   * <^cr>

SAVE UNDER CONTROL FILE NAME=DSC:S471/MOD <cr>
EXECUTE, BEGIN or END? END
```

The user updates the 0 value to "*" and uses the shift ^ to skip the remaining prompts.

The modified control file DSC:S471/MOD can now be used instead of the standard control file. The parameters that need be entered for its use are now: DATA FILE, DATA BIAS, TSFR CODE, CMPR AFTER, MEM START ADDR, MEM START BIT, MEM LEV 1 BIT STEP.

10.12.4 EXECUTING A CONTROL FILE EXAMPLE. This example uses the control file created in the previous example to burn a pair of S471 ROMS (256 X 8) from a 256-word relocatable object module named DATA on the disc in drive 2. The ROMS eventually will be stationed at address F000 on a computer memory card.

```
CONTROL FILE =
DSC:S471/MOD(DSC2:DATA, F000, 1, 1, 0, 0, 16)
DSC:S471/MOD(DUMY,F000, 1, 1, 0, 8, 16)
```

10.13 NONRECOVERABLE ERROR MESSAGES

The following is a list of nonrecoverable error messages issued by TXPROM. These errors cause abortion of all action and return to the CONTROL FILE = prompt:

```
DATA FILE OPEN ERROR
DATA FILE I/O ERROR
CONTROL FILE OPEN ERROR
CONTROL FILE I/O ERROR
HARDWARE MALFUNCTION
HARDWARE OFFLINE
NO. STRING COUNT ERROR
STRING ADDRESS OUT OF BOUNDS
CAN'T GET MEMORY
```

SECTION XI

TXDS BNPF AND HIGH-LOW (BNPFHL) DUMP UTILITY PROGRAM

11.1 INTRODUCTION

The BNPFHL utility program provides the capability of converting a 990 Computer module in standard object code format (i.e., in compressed or noncompressed format) to a module in BNPF format (figures 11-1 and 11-2) or to a module in High-Low format. The conversion from standard object code format to BNPF format is presented in figure 11-1.

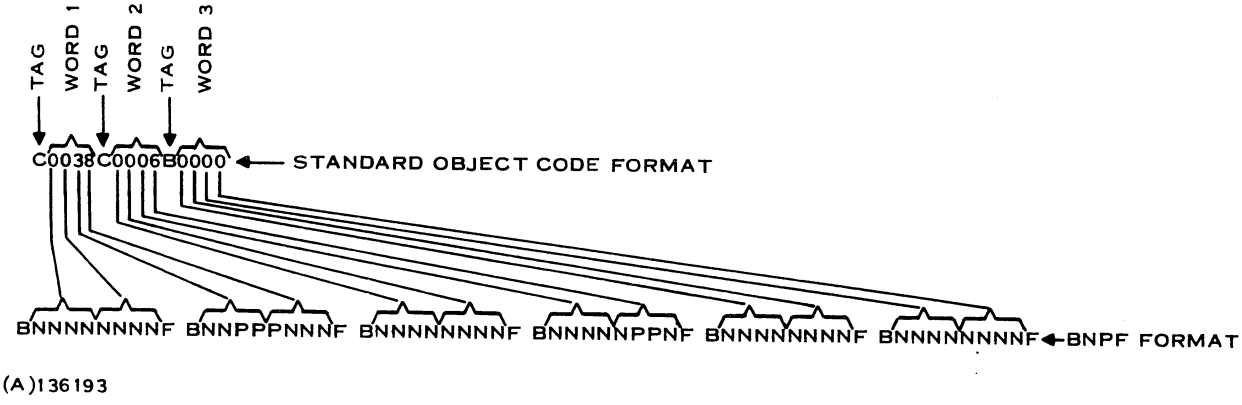


Figure 11-1. Standard Object Code Format to BNPF Format Conversion

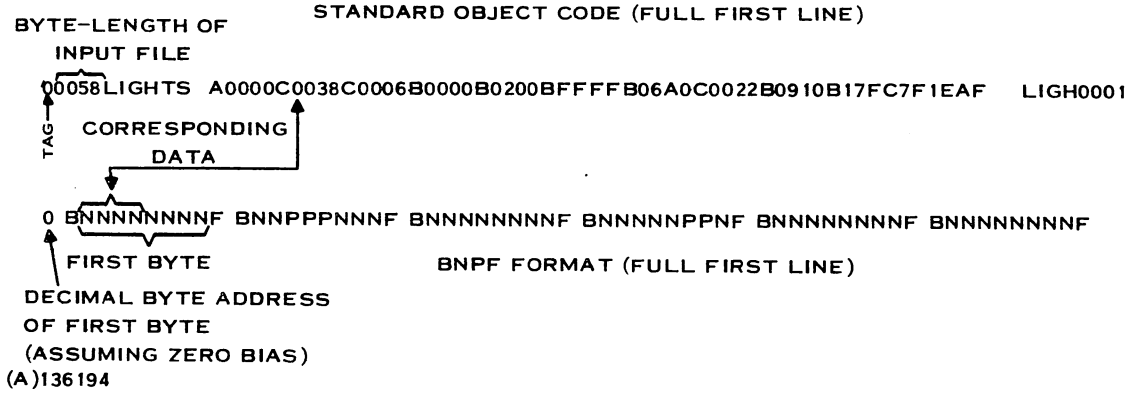


Figure 11-2. Standard Object Code Format to BNPF Format, Full First Line Conversion

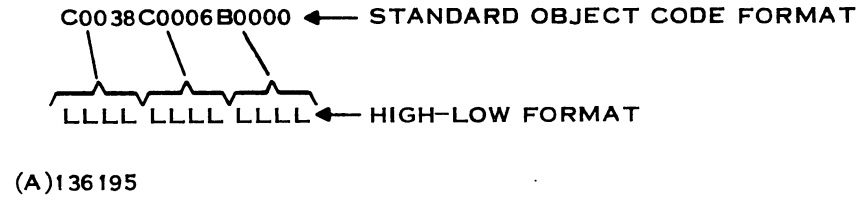


Figure 11-3. Standard Object Code Format to High-Low Format Conversion

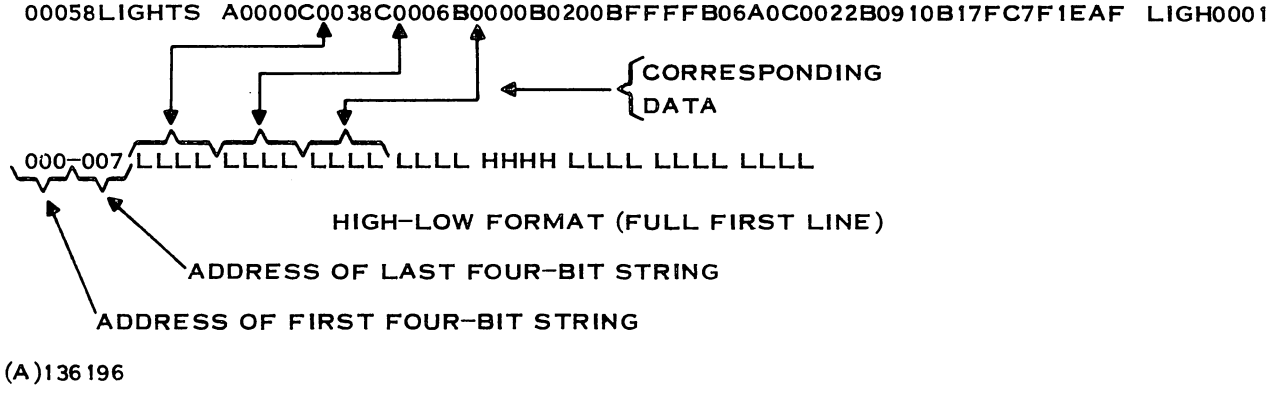


Figure 11-4. Standard Object Code Format to High-Low Format, Full First Line Conversion

When a module in standard object code format is converted to BNPF format, each byte of the standard object code is converted into a string of Ns and Ps (as shown above) preceded by a B (denoting the beginning of the byte) and followed by an F (denoting the end or finish of the byte). Each N corresponds to a negative or zero bit value and each P corresponds to a positive or one bit value. The output module in the BNPF format begins with the decimal byte address (up to five digits) of the first byte contained on the line (as presented in figure 11-2). This decimal byte address has no leading zeros and begins in column one. Each of the lines in the BNPF formatted module contains no more than six bytes of information. An example of a full first line of standard object code is presented in figure 11-2 with the full first line of a converted module in BNPF format. The numbers being converted in figure 11-2 are identical to those shown in figure 11-1.

The conversion from standard object code format to High-Low format is presented in figure 11-3.

When a module in standard object code format is converted to High-Low format, one of the four hexadecimal numbers in each word of the standard object code is converted into a four-bit string of Hs and Ls (where each H corresponds to a high or one-bit value and each L corresponds to a low or zero bit value). The hexadecimal number in each word to be converted is selected by use of the Position option entry. (Refer to paragraph 11.3.3.5 for a description of the Position option entry function.) This Position option entry may be used to specify a 0, 4, 8, or 12, respectively, for the first, second, third, or fourth hexadecimal numbers in the word. The conversion of the hexadecimal number in the first word, into a four-bit string of Hs and Ls, is followed by a conversion of the corresponding hexadecimal number in the same position of the second word (of the standard object code). The process is continued for each corresponding hexadecimal number in each of the words specified in the response to the MEMORY: prompt. (Refer to paragraph 11.3.4 below for a description of the response to the MEMORY: prompt). The Position option entry may also be used to enter any one of the numbers from 0 through 12 and thereby, specify the bit position in the 16-bit word at which the four-bit conversion is to begin. This means, for example, that specifying a 3 would result in converting bits 3, 4, 5, and 6 of the 16-bit word (which is represented in hexadecimal standard object code format) to a four-string of Hs and Ls. The output module in the High-Low format begins each line with the beginning and end address (in decimal) of each of the four-bit strings presented on the line, using three digits for the address of the first four-bit string on the line and another three digits for the address of the last four-bit string on the line. (See figure 11-4.) Each of the lines contains no more than eight four-bit strings. An example of full first line of standard object code is presented in figure 11-4 with the full first line of converted module in High-Low format. The numbers being converted are identical to those shown in figure 11-3.

NOTE

All HILO conversions begin on a word boundary. Therefore, the response to the MEMORY: prompt requires an even-numbered address entry for the beginning and end address.

The following paragraphs describe how to employ this utility program.

11.2 LUNOs

The BNPFHL utility program uses LUNOs 10 and 11, which are assigned to the input and output pathnames, respectively.

11.3 LOADING THE BNPFHL UTILITY PROGRAM

Proceed as follows:

1. Load the TXDS Control Program in accordance with the step-by-step procedure presented in Section II in this manual.

2. Place the TXDS diskette containing the BNPFHL utility program in an available disc drive.

3. Respond to the PROGRAM:, INPUT:, OUTPUT:, and OPTIONS: prompts as follows:

```
PROGRAM: :BNPFHL/SYS
  INPUT: Input Pathname
 OUTPUT: Output Pathname
OPTIONS: {BNPF} , {DUMP   }
         {HILO}  , {COMPARE}  [,B<bias>,I<init>,P<pos>]
                  {LOAD   }
```

BNPF HIGH/LOW UTILITY 937660 *A
(where a number is entered for <bias>, <unit>, or <pos>

MEMORY: <beg addr>,<end addr>

(The MEMORY: prompt is printed or displayed on the system console after the BNPFHL utility program is loaded as described in paragraph 11.3.4.)

The responses to the INPUT:, OUTPUT:, OPTIONS:, and MEMORY: prompts are described in the following subparagraphs.

NOTE

All numerical input values in response to any of the prompts are assumed to be decimal. However, another base may be specified by using the following prefixes:

Prefix	Base	Example
!	Octal	!23 (equals decimal 19)
>	Hexadecimal	>23 (equals decimal 35)

11.3.1 RESPONSE TO THE INPUT: PROMPT. The response to the INPUT: prompt is either the pathname of a file or the pathname of a device. One of these two responses must be specified. When a DUMP or COMPARE option is specified, the file or device should contain a standard object code module. When a LOAD option is specified, the input file or device should contain either a BNPF of High-Low formatted module to correspond with the BNPF or HILO response to the OPTIONS: prompt.

11.3.2 RESPONSE TO THE OUTPUT: PROMPT. The response to the OUTPUT: prompt is either the pathname of a file or the pathname of a device. One of these two responses must be specified. When the COMPARE option is specified, the response to the OUTPUT: prompt should be a file which contains a BNPF or a HILO formatted module, depending upon whether a BNPF or a HILO file is to be compared to the input standard object code. The output device should not be a hard copy device because no carriage control is included in the output.

11.3.3 RESPONSE TO THE OPTIONS: PROMPT. The response to the OPTIONS: prompt is described in the following subparagraphs.

NOTE

All options must be separated by commas. The Bias, Initialization, and Position options can be defaulted as explained below, but, when used, must be specified in the following sequence: Bias, Initialization, and Position.

11.3.3.1 BNPF and HILO Options. The BNPF option specifies a BNPF formatted input or output module and the HILO option specifies a High-Low formatted input or output module. Either the BNPF or HILO option must be specified. The abbreviations BN and HI may be used, respectively, instead of the full four characters. When neither the BNPF option or the High-Low option is specified, an error results.

11.3.3.2 DUMP, COMPARE, and LOAD Options. The use of these options is described in the following subparagraphs.

NOTE

1. One of these options must be specified or an error will result.
2. Each of the option names may be abbreviated by using the first two letters in the option name

DUMP (DU) Option. The DUMP option causes the input file in 990 standard object code to be dumped to the output file in the specified BNPF or High-Low format.

COMPARE (CO) Option. The COMPARE option is used to verify the results of a DUMP by comparing the output BNPF or HILO formatted file to the input file in standard object code format.

When there is no discrepancy in a BNPF COMPARE, the beginning and end address of the compared information or data are printed on the system console. The following printout is an example of a BNPF COMPARE without errors:

```
TXDS   936215 *A        1/0        00:02

          PROGRAM:              :BNPFHL/SYS
            INPUT:              DSC2:LIGHTS/OBJ
           OUTPUT:              DSC2:TEMP/OBJ
          OPTIONS:              BN, CO
BNPF HIGH/LOW UTILITY 937660 *A
          MEMORY:               0, 24
          BEG ADDR=0000
          END ADDR=0018
          TXDS 936215 *A    1/0  00:03

                PROGRAM:
```

When there is no discrepancy in a HILO COMPARE, no printout or display is presented on the system console.

When a BNPF COMPARE is discrepant, a presentation of the discrepancy is printed out or displayed on the system console. An example of a typical printout or display of a discrepancy is:

T0064=9C M0064=38

where:

T represents the BNPF output file; 0064 represents the decimal address of the byte; and 9C represents the hexadecimal value of the byte, and

where:

M represents the input file in standard object code; 0064 represents the decimal address of the byte; and 38 represents the hexadecimal value of the byte.

The discrepancy is noted by the difference in hexadecimal byte-values 9C and 38. When there exists no discrepancy, both hexadecimal byte-values are 38 and, as a result, are not printed out or displayed on the system console.

When a HILO COMPARE is discrepant, a presentation of the discrepancy is printed out or displayed on the system console. An example of a typical printout or display of a discrepancy is:

M0003.<0,3>=0000 T0003.<0,3>=2000

where:

M represents the input file in standard object code; 003 represents the decimal address of the input file word; 0,3 represents the beginning and ending bit positions of the four-bit string in the input object file; and 0000 represents the hexadecimal value of the input file word; and

where:

T represents the HILO output file; 0003 represents the decimal address of the output file word; 0,3 represents the beginning and ending bit positions of the discrepant four-bit string in the output file; and 2000 represents the hexadecimal value of the output file word.

The discrepant output is presented in hexadecimal word format but, nevertheless, represents the High-Low formatted output from the HILO DUMP program execution. In addition, the discrepancy is noted by the difference in hexadecimal word-values 0000 and 2000. When there exists no discrepancy, both hexadecimal word-values are 0000 and, as a result, are not printed out or displayed on the system console.

LOAD (LO) Option. Selection of the LOAD option causes a previously created BNPF or High-Low formatted file to be converted into an output file which can be used to program PROMs using the PROM Programming Module (i.e. the hardware module). Refer to the TXDS (TXPROM) Programmer Utility Program section in the TXDS Programmer.s Guide, manual number 946258-9701.

11.3.3.3 **Bias (B<bias>) Option.** The Bias option supplements the DUMP and COMPARE options. It defines the number to be added to the address of the relocatable data in the input file as well as to the relocatable data itself for the purpose of producing the output file or for the purpose of comparing the input file to the output file. The BIAS option has no effect on nonrelocatable object module data. The default-substitute produced by the utility program is 0. An example of the use of the Bias option is presented in paragraph 11.5.5. The bias must immediately follow the B.

11.3.3.4 **Initialization (I<init>) Option.** The Initialization option is used to initialize the buffer area into which the input file's standard object code is to be read. This initialization is done prior to converting the input file to the BNPF or HILO format so that each bit position initially contains a 1 or 0. Unused sections of the buffer are also initialized. The default substitute provided by the utility program for the Initialization option is 0. The operator must enter a 0 or 1 immediately after the I. Whenever a number other than 0 or 1 is specified an error results. An example of the use of the Initialization option is presented in paragraphs 11.5.3 and 11.5.4.

11.3.3.5 **Position (P<pos>) Option.** The Position option specifies the first bit of the four-bit string from each of the input-file-words (which are in the format of standard object code) that are to be converted to the HILO format. A Position option number from 0 through 12 is selected by the operator to supplement the HILO option selection. The selected number specifies the position number of the start-bit of the four-bit string of the 16-bit word from the input file's standard object code. An example of the use of the Position option is presented in paragraph 11.5.3. The bit position parameter must immediately follow the P.

11.3.4 **RESPONSE TO MEMORY: PROMPT.** The operator's response to the MEMORY: prompt is used to sepecify the bounds for a BNPF or HILO format. These bounds must be word addresses (even) for HILO format, but may be byte addresses for BNPF format. The first boundary is the address of the first word or byte to be formatted. The second boundary is the address of the last word or byte to be formatted. The boundaries must be separated by a comma. When the addresses are hexadecimal, they must be preceded by a ">" character. It is not necessary to format a whole object module.

Following is an example of a BNPF format MEMORY: prompt. The module was assembled using an ADRG >A0 directive which causes the assembler to generate absolute addresses starting at >A0. The module is >6C bytes long.

MEMORY: >A0, >10B

>A0 is the address of the first byte to be formatted and >10B is the address of the last byte to be formatted.

Following is an example of a HILO format MEMORY: prompt. The module was assembled as a relocatable module. The module is >5E bytes long.

MEMORY: 0, >5C

The address of the first word is zero, the address of the last word is >5C.

NOTE

The starting and ending address may be obtained from an assembly listing that is generated with the object module.

In the event the end address entered in response to the MEMORY: prompt exceeds the capacity of memory, the CANNOT GET MEMORY error message is printed or displayed on the system console.

NOTE

1. The HILO option produces an error message when a byte boundary is specified.

2. The MEMORY: prompt is not issued when the LOAD option is used.

11.4 ERROR MESSAGES

The error messages that result from misuse of the BNPFHL utility are listed in table 11-1 with an explanation of the cause of each error.

11.5 EXAMPLES OF USAGE OF THE BNPFHL UTILITY PROGRAM

Six examples of usage of the BNPFHL utility program are presented in the following subparagraphs. The standard object code used in each of the examples is presented below.

```
TX990  SYSTEM
MEMORY SIZE(WORDS): 24576     AVAILABLE: 12844
!

. EX,16.TE.
TXDS  936215 ♦A      1/ 0    00:01

PROGRAM: :TXCCAT/SYS
  INPUT: DSC2:LIGHTS/OBJ
 OUTPUT: LOG
OPTIONS: SL01,LF55
TXCCAT 937543 ♦♦
```

```
00058LIGHTS. A0000C0038C0006B0000B0200BFFFFB06A0C0022B0910B17FC7F1EAF    LIGH0001
B06A0C0022B0A10B17FCB16F7B06A0C0022B10F2B020CB1FE0B3200B06C0B32007F193F LIGH0002
B06C0B0201B1000B0601B16FEB045BA00387F840F                               LIGH0003
50006LIGHTS7FD03F                                                       LIGH0004
:      LIGHTS     02/25/77  07:23:01     SDSMAC 947075 ♦D              LIGH0005
```

Table 11-1. BNPFHL Error Messages

Message	Cause
UNABLE TO OPEN FILE	The specified input file does not exist.
I/O ERROR ON INPUT FILE	The input file cannot be read.
I/O ERROR ON OUTPUT FILE	The output file cannot be opened and/or written to.
BIT VALUE TOO LARGE. MUST BE > C OR LESS	The position parameter exceeds C_{16}.
ILLEGAL FUNCTION COMMAND	The first parameter after the OPTIONS: prompt is not HILO or BNPF.
REQUIRED PARAMETER MISSING	The second parameter after the OPTIONS: prompt is not DUMP, COMPARE, or LOAD.
INIT VALUE GREATER THAN 1	The initialization parameter is not 0 or 1.
UNABLE TO OPEN OUTPUT FILE	The specified output file does not exist.
ILLEGAL NUMBER INPUT	One of the numeric parameters is not a legal number.
ADDRESS WAS NOT ON WORD BOUNDARY	One of the addresses after the MEMORY: prompt does not begin on a word boundary. This error occurs only with the HILO option.
BAD OBJECT FORMAT	The input files does not contain legal object code.
ABORT; SYSTEM ERROR FROM XOP	A system error flag was returned from an XOP. The flag value is printed above the error.
START ADDRESS GREATER THAN END ADDRESS	The first memory parameter is larger than the second memory parameter.
CANNOT GET MEMORY	Cannot get memory to run.
START GREATER THAN END OR LENGTH > 256 WORDS	The starting address after the memory: prompt is larger than the ending address or the difference between the two is greater then 256. (This applies only to HILO format.)

11.5.1 EXAMPLE OF BNPF FORMATTED DUMP USING DEFAULT SUBSTITUTE PARAMETERS

```
TXDS 936215 *A       1/0              00:01
                PROGRAM:              :BNPFHL/SYS
                  INPUT:              DSC2: LIGHTS/OBJ
                 OUTPUT:              DSC2: TEMP/OBJ
                OPTIONS:              BN, DU
BNPF HIGH/LOW UTILITY 937660 *A
                 MEMORY:              0.>58
TXDS 936215 *A                        00:02
```

```
0 BNNNNNNNNF BNNPPPNNNF BNNNNNNNNF BNNNNNPPNF BNNNNNNNNF BNNNNNNNNF
6 BNNNNNNPNF BNNNNNNNNF BPPPPPPPPF BPPPPPPPPF BNNNNNPPNF BPNPNNNNNF
12 BNNNNNNNNF BNNPNNNPNF BNNNNPNNPF BNNNPNNNNF BNNNPNPPPF BPPPPPPNNF
18 BNNNNNPPNF BPNPNNNNNF BNNNNNNNNF BNNPNNNPNF BNNNNPNPNF BNNNPNNNNF
24 BNNNPNPPPF BPPPPPPNNF BNNNPNPPNF BPPPPNPPPF BNNNNNPPNF BPNPNNNNNF
30 BNNNNNNNNF BNNPNNNPNF BNNNPNNNNF BPPPPNNPNF BNNNNNNPNF BNNNNPPNNF
36 BNNNPPPPPF BPPPNNNNNF BNNPPNNPNF BNNNNNNNNF BNNNNNPPNF BPPNNNNNNF
42 BNNPPNNPNF BNNNNNNNNF BNNNNNPPNF BPPNNNNNNF BNNNNNNPNF BNNNNNNNPF
48 BNNNPNNNNF BNNNNNNNNF BNNNNNPPNF BNNNNNNNPF BNNNPNPPNF BPPPPPPPNF
54 BNNNNNPNNF BNPNPPNPPF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
60 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
66 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
72 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
78 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
84 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
$
```

11.5.2 EXAMPLE OF HILO FORMATTED DUMP USING DEFAULT-SUBSTITUTE

```
TXDS  936215 ♦A      1/ 0    00:04

PROGRAM: :BNPFHL/SYS
  INPUT: DSC2:LIGHTS/OBJ
 OUTPUT: DSC2:TEMP/OBJ
OPTIONS: HI,DU
MEMORY:0,>58
TXDS  936215 ♦A      1/ 0    00:05

PROGRAM: :TXCCAT/SYS
  INPUT: DSC2:TEMP/OBJ
 OUTPUT: LOG
OPTIONS: SL01,LF55
TXCCAT 937543 ♦♦




000-007  LLLL LLLL LLLL LLLL HHHH LLLL LLLL LLLL
008-015  LLLH LLLL LLLL LLLL LLLH LLLH LLLL LLLL
016-023  LLLH LLLL LLLH LLHH LLLL LLHH LLLL LLLL
024-031  LLLH LLLL LLLH LLLL LLLL LLLL LLLL LLLL
032-039  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
040-047  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
048-055  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
056-063  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
064-071  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
072-079  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
080-087  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
088-095  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
096-103  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
104-111  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
112-119  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
120-127  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
128-135  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
136-143  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
144-151  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
152-159  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
160-167  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
168-175  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
176-183  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
184-191  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
192-199  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
200-207  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
208-215  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
216-223  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
224-231  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
232-239  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
240-247  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
248-255  LLLL LLLL LLLL LLLL LLLL LLLL LLLL LLLL
```

11.5.3 EXAMPLE OF HILO FORMATTED DUMP BEGINNING AT POSITION 4 AND OF INITIALIZING THE BUFFER TO ALL BINARY ONES

```
TXDS  936215 ♦A      1/ 0    00:06

PROGRAM: :BNPFHL/SYS
  INPUT: DSC2:LIGHTS/OBJ
 OUTPUT: DSC2:TEMP/OBJ
OPTIONS: HI,DU,I1,P4
MEMORY: 0,>58
TXDS  936215 ♦A      1/ 0    00:07

PROGRAM: :TXCCAT/SYS
  INPUT: DSC2:TEMP/OBJ
 OUTPUT: LOG
OPTIONS: SL01,LF55
TXCCAT 937543 ♦♦
```

```
000-007  LLLL LLLL LLLL LLHL HHHH LHHL LLLL HLLH
008-015  LHHH LHHL LLLL HLHL LHHH LHHL LHHL LLLL
016-023  LLLL LLHL HHHH LLHL LHHL LLHL LHHL LLHL
024-031  LLLL LHHL LHHL LHLL HHHH HHHH HHHH HHHH
032-039  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
040-047  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
048-055  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
056-063  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
064-071  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
072-079  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
080-087  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
088-095  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
096-103  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
104-111  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
112-119  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
120-127  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
128-135  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
136-143  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
144-151  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
152-159  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
160-167  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
168-175  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
176-183  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
184-191  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
192-199  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
200-207  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
208-215  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
216-223  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
224-231  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
232-239  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
240-247  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
248-255  HHHH HHHH HHHH HHHH HHHH HHHH HHHH HHHH
```

11.5.4 EXAMPLE OF A HILO COMPARE WITH DISCREPANT DATA. The file generated in the example in 11.5.3 is compared to the first hexadecimal number in the words of the standard object code file (positions 0-3) instead of the second hexadecimal number in the words of the standard object code file (positions 4-7) that was used in generating the file presented in paragraph 11.5.3.

```
TXDS  936215 •A      1/ 0    00:11

PROGRAM: :BNPFHL/SYS
  INPUT: DSC2:LIGHTS/OBJ
 OUTPUT: DSC2:TEMP/OBJ
OPTIONS: HI,CO,I1
MEMORY: 0,<58
ILLEGAL NUMBER INPUT
TXDS  936215 •A      1/ 0    00:15

PROGRAM: :BNPFHL/SYS
  INPUT: DSC2:LIGHTS/OBJ
 OUTPUT: DSC2:TEMP/OBJ
OPTIONS: HI,CO,I1
MEMORY: 0,>58
M0003.(0,3)=0000  T0003.(0,3)=2000  M0005.(0,3)=0000  T0005.(0,3)=6000
M0007.(0,3)=0000  T0007.(0,3)=9000  M0008.(0,3)=1000  T0008.(0,3)=7000
M0009.(0,3)=0000  T0009.(0,3)=6000  M000B.(0,3)=0000  T000B.(0,3)=A000
M000C.(0,3)=1000  T000C.(0,3)=7000  M000D.(0,3)=1000  T000D.(0,3)=6000
M000E.(0,3)=0000  T000E.(0,3)=6000  M0010.(0,3)=1000  T0010.(0,3)=0000
M0011.(0,3)=0000  T0011.(0,3)=2000  M001C.(0,3)=1000  T0012.(0,3)=F000
M0013.(0,3)=3000  T0013.(0,3)=2000  M0014.(0,3)=0000  T0014.(0,3)=6000
M0015.(0,3)=3000  T0015.(0,3)=2000  M0016.(0,3)=0000  T0016.(0,3)=6000
M0017.(0,3)=0000  T0017.(0,3)=2000  M0018.(0,3)=1000  T0018.(0,3)=0000
M0019.(0,3)=0000  T0019.(0,3)=6000  M001A.(0,3)=1000  T001A.(0,3)=6000
M001B.(0,3)=0000  T001B.(0,3)=4000
TXDS  936215 •A      1/ 0    00:19
```

11.5.5 EXAMPLE OF A BNPF FORMATTED DUMP WITH BIAS 100

```
PROGRAM: :BNPFHL/SYS
  INPUT: DSC2:LIGHTS/OBJ
 OUTPUT: DSC2:TEMP/OBJ
OPTIONS: BN,DU,B100
MEMORY: 0,>58
TXDS  936215 ♦A      1/ 0    00:20

PROGRAM: :TXCCAT/SYS
  INPUT: DSC2:TEMP/OBJ
 OUTPUT: LOG
OPTIONS: SL01,LF55
TXCCAT 937543 ♦♦
```

```
100 BNNNNNNNNNF BPNNPPPNNF BNNNNNNNNF BNPPNPNPNF BNNNNNNNNF BNNNNNNNNF
106 BNNNNNNPNF BNNNNNNNNF BPPPPPPPPF BPPPPPPPPF BNNNNNPPNF BPNPNNNNNF
112 BNNNNNNNNF BPNNNNPPNF BNNNNPNNPF BNNNPNNNNF BNNNPNPPPF BPPPPPPNNF
118 BNNNNNPPNF BPNPNNNNNF BNNNNNNNNF BPNNNNPPNF BNNNNPNPNF BNNNPNNNNF
124 BNNNPNPPPF BPPPPPPNNF BNNNPNPPNF BPPPPNPPPF BNNNNNPPNF BPNPNNNNNF
130 BNNNNNNNNF BPNNNNPPNF BNNNPNNNNF BPPPPNNPNF BNNNNNNPNF BNNNNPPNNF
136 BNNNPPPPPF BPPPNNNNNF BNNPPNNPNF BNNNNNNNNF BNNNNNPPNF BPPNNNNNNF
142 BNNPPNNPNF BNNNNNNNNF BNNNNNPPNF BPPNNNNNNF BNNNNNNPNF BNNNNNNNPF
148 BNNNPNNNNF BNNNNNNNNF BNNNNNPPNF BNNNNNNNPF BNNNPNPPNF BPPPPPPPNF
154 BNNNNNPNNF BNPNPPNPPF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
160 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
166 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
172 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
178 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
184 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF
$
```

11.5.6 EXAMPLE OF A BNPF COMPARE WITH DISCREPANT DATA. The BNPF file which is used is the one created in paragraph 11.5.5 with Bias option 100; however, the COMPARE was performed without the Bias option.

```
TXDS  936215 ♦A      1/ 0    00:21
 -

PROGRAM: :BNPFHL/SYS
  INPUT: DSC2:LIGHTS/OBJ
 OUTPUT: DSC2:TEMP/OBJ
OPTIONS: BN,CO
MEMORY: 0,>58
BEG ADDR=0064
T0064=9C  M0064=38  T0064=6A  M0064=06
T0070=86  M0070=22
T0076=86  M0076=22
T0082=86  M0082=22
END ADDR=0058
TXDS  936215 ♦A      1/ 0    00:22

PROGRAM:
```

SECTION XII

TXDS IBM CONVERSION UTILITY (IBMUTL) PROGRAM

12.1 INTRODUCTION

The IBM Conversion Utility (IBMUTL) Program provides a means of transferring standard IBM formatted diskette datasets to TX990 files and transferring TX990 files to standard IBM formatted diskette datasets. IBMUTL also provides a means of formatting diskettes to standard IBM specification for a single density diskette as designated in "The IBM Diskette For Standard Data Interchange", GA21-9182-0.

12.2 IBMUTL DESCRIPTION

IBMUTL allows the user to read or write datasets on an IBM formatted diskette in a form that can be read and used by systems and devices that are based on IBM sequentially sectored diskettes using the EBCDIC character set. The IBM formatted diskette may already contain datasets created by another process or may have been newly formatted by this utility or other means. All preexisting datasets will be preserved.

12.2.1 FORMATTING IBM DISKETTE. The diskette is formatted to IBM format by entering the format command. If more than two bad tracks are found, or if track zero is bad, the diskette is unuseable and another diskette should be used. Track zero contains the dataset headers (sectors 8-26) and other information about the diskette (sectors 1-7). The dataset headers are written to include name, record length, beginning of extent (BOE), end of extent (EOE), and end of data (EOD) fields only. All others are left in the initialized state (blank).

12.2.2 TRANSFERRING TX990 FILES TO IBM DATASETS. TX990 files that are to be converted to IBM format must be specified by the operator with a standard TX990 pathname. The new dataset will begin with the first available label following the last used label in the IBM diskette directory. Empty labels between used labels are skipped by this directory. The name of the new dataset may be the same as an already existing dataset but the existing dataset will not be replaced.

12.2.3 TRANSFERRING IBM DATASETS TO TX990 FILES. The operator must specify the desired dataset label and the TX990 pathname. The dataset labels from the IBM diskette are displayed when the transfer command is entered. If the TX990 file does not already exist, it will be created as a noncontiguous sequential file. If two datasets have the same name, only the first dataset may be accessed by this utility.

12.3 LUNOS AND THEIR USES

This utility uses the console device assigned to LUNO 15_{16} as the interactive device. If LUNO 15_{16} is not assigned, the system console is used. LUNO 11_{16} is assigned to the drive in which the IBM diskette is mounted; LUNO 10_{16} is assigned to the TX990 file.

12.4 LOADING AND EXECUTING

IBMUTL can be executed using OCP or the Terminal Executive Development System (TXDS). If OCP is used, follow the procedure below:

1. Ready the device which contains the object program for IBMUTL.
2. Load the program into memory using OCP. IBMUTL must be loaded as a privileged task.

LP,CS1,3,P.	if loading the IBMUTL object program from cassette drive one as a privileged task
LP,:IBMUTL/SYS,3,P	if loading the IBMUTL object program from the system diskette drive, file :IBMUTL/SYS, as a priority level 3 privileged task

3. Execute the program, and terminate OCP.

   EX,10.TE.

   If the IBMUTL object program is linked into the system, omit steps 1 and 2 and simply execute the task using the task ID assigned to it at that time.

If the TXDS control program is used, follow the procedure below:

1. Ready the device which contains the object program for IBMUTL.

2. Enter the name of the device or file which contains the object program in response to the PROGRAM: prompt. Specify IBMUTL to be loaded as a privileged task by following the device or file name with ",P".

PROGRAM: CS1,P*	if loading the object program on cassette drive one and executing as a privileged task
PROGRAM::IBMUTL/SYS,P*	if the object program in the file :IBMUTL/SYS on the system diskette drive and executing as a privileged task

12.5 OPERATOR INTERACTION

12.5.1 SPECIAL CHARACTERS. There are two special characters recognized by IBMUTL. They are as follows:

* When entered in response to a prompt and followed by a carriage return, IBMUTL is terminated.

& When entered in response to a prompt and followed by a carriage return, IBMUTL restarts by requesting the IBM diskette drive name.

12.5.2 OPERATOR PROMPTS. When the task has been loaded and executed, the task name and revision level are printed followed by a request for the IBM floppy diskette drive name.

```
TI FLOPPY DISK IBM CONVERSION TASK. PN 936216 **

IBM DISK DRIVE NAME:
```

The required service is selected in response to the next query.

```
SERVICE? F=FORMAT. T=TRANSFER:
```

When format is selected, no further interaction is necessary. The format process is executed to completion and the following messages are displayed.

FORMAT IN PROGRESS . . .

FORMAT COMPLETE.

The utility will then request the IBM diskette drive name again.

When one of the transfer services is selected, the IBM diskette is checked for proper format and a list of the labels is displayed. When the IBM format is found to be incorrect, a message is displayed and the diskette name request is repeated. (Operator responses are underlined; (C/R) represents a carriage return). Upon responding with a "T" for the transfer function, the following messages are output:

FILE1 All dataset labels on the IBM format diskette are listed.

FILE2

FILE3

NOTE

While listing the dataset labels of the IBM formatted diskette, blanks are printed when a dataset is encountered in which the label contains all blanks.

or,

** DISKETTE NOT IBM FORMAT ** Specified diskette is not an IBM formatted diskette.

At this point, if the format is correct, the program is ready to perform the transfer operation. The direction of transfer is established by response to the next query.

FUNCTION? F=FILE TO DATASET, D=DATASET TO FILE: F
or D

Whether or not an "F" or a "D" is selected, the next question asked is:

CHARACTERS PER RECORD? 2 MIN. - 128MAX: 80(C/R)

When only a carriage return is entered, the default value is 80 characters per record.

The TX990 user file pathname and IBM dataset names are requested next:

USER FILE PATHNAME: DSC2:SOURCE/ABC(C/R)

DATASET NAME: SOURCE(C/R)

FILE TRANSFER IN PROGRESS. . .

Pathname can be defaulted to the first six characters of the dataset name when transfer is dataset to file; or when transfer is file to dataset, the dataset name can be defaulted to the file name portion of the pathname. The default substitute is specified by a carriage return (C/R) response to the query. Pathnames that are preceded by a colon are defaulted to the system diskette drive.

When the IBM diskette is not filled at the completion of converting the specified file, IBMUTL requests the record size again.

When it is desired to reverse the transfer function or change the drive on which datasets are being accessed, an ampersand (&) symbol reply returns the program to the point where the diskette name is requested:

USER FILE PATHNAME: &(C/R)

Responding with an ampersand (&) returns program control to the initial user prompt:

IBM DISK DRIVE NAME:

If the user enters an asterisk (*), IBMUTL terminates with the following message;

UTILITY SERVICE TERMINATED

12.6 ERROR REPORTING AND RECOVERY

Errors encountered during execution of IBMUTL are reported to the operator in accordance with table 12-1. Whenever recovery from such errors is possible, the program returns to a logical restart point and continues its function.

Table 12-1. IBMUTL Error Messages

Message	Meaning	Recovery
** DISKETTE NOT IBM FORMAT **	Specified diskette is not an IBM format diskette.	Insert a properly formatted diskette in specified drive, or return to diskette name request (enter "&") and input correct drive name.
UNDEFINED PATHNAME	Illegal pathname has been entered.	Validate pathname and reenter.
DISKETTE DIRECTORY FULL	An attempt to exceed the maximum number of datasets (19) allowable per IBM formatted diskette.	Program control returns to diskette drive request. Install new IBM format diskette and retry transfer function.
TOO MUCH DATA	Data capacity of IBM format diskette has been exceeded. Last file is labeled as an empty dataset and transfer is terminated.	Program control returns to diskette drive request. Install new IBM format diskette and retry transfer function.
MORE THAN 2 BAD TRACKS, FORMAT ABORTED	Bad diskette.	Program control returns to diskette drive request. Install new diskette and retry format function.
FILE SERVICE ERROR nn	Error encountered while accessing TX990 user file. Refer to Error Appendices for error code nn.	Program control returns to diskette drive request. Respond according to individual error code.
FLOPPY DISK ACCESS ERROR nn	Error encountered while accessing IBM dataset. Refer to Error Appendices for error code nn.	Program control returns to diskette drive request. Respond according to individual error code.
I/O ERROR nn	I/O error encountered during program execution. Refer to Error Appendices for error code nn.	Program control returns to diskette drive request. Respond according to individual error code.

SECTION XIII

TXDS ASSIGN AND RELEASE LUNO UTILITY PROGRAM

13.1 INTRODUCTION

The TXDS LUNO Utility provides a means of assigning and releasing Logical Unit Numbers (LUNOs) without having OCP linked in with the TX990 operating system. This capability is especially necessary in order to execute FORTRAN programs without OCP.

13.2 LOADING AND EXECUTING

The LUNO utility is executed through the TXDS Control Program, by responding to the PROGRAM: prompt in the following manner:

```
PROGRAM:          :TXLUNO/*<cr>
```

13.3 OPERATOR INTERACTION

When the LUNO utility starts execution, it displays the following identification message:

```
ASSIGN & RELEASE LUNO  939888**
```

13.3.1 OPERATOR PROMPTS. After the heading is displayed, the LUNO utility displays two prompts, in the following order:

```
LUNO?
PATHNAME?
```

The user enters the logical unit number being assigned or released in response to the LUNO? prompt. LUNOs may range from 0 to 255. If the user desires to enter a hexadecimal value, it must be preceded by a ">" sign.

If the LUNO is being assigned, the user enters the pathname of the file or device to which the LUNO is to be assigned in response to the PATHNAME? prompt. If the LUNO is to be released, the user should enter a carriage return in response to the PATHNAME? prompt.

The following example assigns LUNO 33 to a file and releases LUNO A_{16}. User responses are underlined.

```
PROGRAM:   :TXLUNO/SYS  * <cr>
ASSIGN AND RELEASE LUNO 939888**
LUNO?  33 <cr>
PATHNAME?  VOL2:TASK1/SRC <cr>
LUNO?  >A <cr>
PATHNAME?  <cr>
LUNO?  *<cr>
```

13.3.2 **SPECIAL CHARACTERS.** The LUNO utility recognizes two special characters which may be entered in response to a prompt:

* Terminates the utility.

& Restarts the utility at the LUNO? prompt.

13.4 ERROR MESSAGES AND RECOVERY

The LUNO utility may return the following error message:

I/O ERROR, nn

where nn is one of the I/O error codes in APPENDIX D.

To recover from the error, retry the LUNO Assign or Release operation.

APPENDIX A

GLOSSARY

APPENDIX A

GLOSSARY

Boot Program – A program that loads the Operating System into memory and starts the Operating System executing.

COMMON – An area of memory which may be coded by use of the TXDS Control Program and the system console keyboard (e.g., a 733 ASR, a 911 VDT) or by means of a task-specified-code and then made accessible for use by a task through the Get COMMON Data address supervisor call. The size of the system COMMON memory area is determined by a system parameter specified when the system is generated.

Default-substitute – A substitute pathname, or field of a pathname, provided by some utility programs when the program or keyboard-entry does not supply the data.

Device Name Table – A table accessed by the File Management supervisor call to obtain the address of the Physical Device Table (PDT) corresponding to a device name. Contains all device names defined in the system and addresses of the PDTs for the devices.

Device Service Routine – A routine of the TX990 Operating System that controls I/O operations with a device.

DNT – Device Name Table.

DSR – Device Service Routine.

Dynamic Task Area – The area of memory occupied by task 10_{16}. Task 10_{16} can be loaded by using the Operator Communication Package (OCP) or the TXDS control program.

End-of-file – A record in a file (either logically or physically) that marks the end of the file. The character sequences that denote end-of-file for the file-oriented supported devices are shown in Appendix B.

End-of-record – A character of a record that marks the end of the record. The characters that denote end-of-record for supported devices are shown in Appendix B.

EOF – End-of-file.

EOR – End-of-record.

GENTX – The system generation task, which obtains system parameters interactively from the keyboard of the LOG. GENTX builds source statement files from which modules TXDATA and TASKDF are assembled.

IDT – Program identifier of the source module.

Initial Program Load – The loading of a TX990 system placing the module in memory and starting execution of the system.

I/O Supervisor – The portion of TX990 that processes I/O supervisor calls, and passes control to the Device Service Routine (DSR) for the device.

IPL – Initial Program Load.

Keyboard Status Block (KSB) – A data structure in TXDATA used for character mode I/O with a VDT. TXDATA includes a KSB for each VDT.

KSB – Keyboard Status Block.

LDT – Logical Device Table.

Logical Device Table (LDT) – A table in TXDATA that contains a Logical Unit Number (LUNO) and the address of the Physical Device Table (PDT) that corresponds to the device assigned to the LUNO.

Logical Unit Number (LUNO) – A number by which an I/O operation specifies the device for the operation.

LUNO – Logical Unit Number.

OCP – Operator Communication Package.

Operator Communication Package (OCP) – A package of modules that contains the routines for the commands by which the operator or user communicates with TX990.

PC – Program Counter.

PDT – Physical Device Table.

Physical Device Table (PDT) – A table in TXDATA that contains device-related data required by the Device Service Routine (DSR) in an I/O supervisor call for the device.

Program Counter (PC) – A register in the computer hardware that contains the address of the next instruction to be executed.

Status Register – A register in the computer hardware that contains condition bits and the interrupt mask.

Supervisor Call Block – A block of memory that defines a supervisor call, addressed by the supervisor call instruction. The code of the supervisor call is in byte 0 of the supervisor call block. The number of additional bytes (if any) and the content of the additional bytes are defined for each supervisor call.

Supervisor Call Table – A table in TXDATA in which entry points to supervisor call routines are listed in a supervisor call code order.

Task Data Division – One of two logical divisions within a task. The data division contains one or more workspaces, data structures, supervisor call blocks, and data for the task. A data division may or may not be assembled separately from the procedure division of the task, and is not shared with any other task.

Task Management – Task Management maintains a state code for each task. The state codes are listed in Appendix C.

Task Scheduler – Initiates execution of a user task. When the currently executing task completes a time slice, the task scheduler passes control to the oldest task on the active list for the highest priority (0). If there is no task on the active list for priority 0, the oldest task on the active list for the next highest priority receives control.

Task Status Block (TSB) – A data structure in TXDATA used by the TX990 Operating System to control execution of the task.

Task Time Delay – The result of a task executing a Time Delay supervisor call. The Time Delay supervisor call suspends the calling task for a specified number of 50 ms periods.

Task Time Slice – A period of execution of a task having a maximum length defined when the system is generated. A task time slice begins when the task scheduler passes control to the task. A task time slice ends: (1) when the system suspends the task upon expiration of the maximum time period allowed for a task time slice; (2) when the task executes a supervisor call that suspends the task; (3) when the system suspends the task to await completion of an I/O operation. To avoid completely locking out low priority tasks, there is a maximum number of consecutive time slices (weighting factor) for each priority level. When the number of time slices has been used by a priority level, the oldest task on the active list for the next lower priority is allowed a time slice before the higher level again has control. The maximum number of time slices for each priority level are system parameters defined when the system is generated. The maximum period of a time slice may be extended by execution of a Do Not Suspend supervisor call. The time slice is less than the maximum time period when the task suspends itself, or is suspended awaiting completion of an I/O operation.

Task Weighting Factor – A count of task time slices for a priority level. When the number of task time slices specified as the weighting factor for priority level has been used by tasks at that priority level after a task at a lower level has had control, a task at a lower priority level receives control for a time slice.

Task Area, Dynamic – Memory area where task 10 resides (see Dynamic Task Area and Task, Uses, Loading of).

Task, Bid – To start execution of a task causing the TX990 Operating System to enter the task on the active list according to its priority level.

Task, Debugging of a – The process of removing errors from a task.

Task, Diagnostic (DTASK) – A system task that terminates a task when fatal errors occur in the task, and prints an error message.

Task, Executing a – Controlling the processor and the resources of the computer.

Task, Linked – Consists of separately assembled modules that have been combined by resolving external references and definitions in the modules to form a single executable module.

Task, LIST8080 – A utility task that copies 80-character records from one device to another.

Task, Loaded – A task copied from an external storage medium into the memory of the computer in preparation for execution.

Tasks, Multiple – Two or more tasks concurrently active in an operating system.

Task, Procedure Division – One of two logical divisions within a task. The procedure division contains the executable code for the task. A procedure division may or may not be assembled separately from the data division of the task and may be shared with other tasks.

Task, Suspended – A task temporarily removed from the active list and from execution as a result of a supervisor call or during an I/O operation.

Task, Terminated – A task removed from execution and from the active list either at normal completion or at an abnormal termination initiated by the operator or by the diagnostic task when a fatal error is detected.

Task, User, Loading of – The task loaded into the dynamic task area using the OCP LPROG command.

Task, Waiting – A task waiting for completion of an I/O operation or for a system function or resource.

Workspace – A 16-word area of memory addressed as workspace registers 0 through 15. The active workspace is defined by the contents of the workspace pointer register.

Workspace Pointer (WP) – A register that contains the address of workspace register 0.

Workspace Register – A memory word accessible to an instruction of the computer as a general purpose register. It may be used as an accumulator, a data register, an index register, or an address register.

WP – Workspace pointer register.

APPENDIX B

COMPRESSED OBJECT CODE FORMAT

APPENDIX B

COMPRESSED OBJECT CODE FORMAT

The standard object code format under the TX990 Operating System is comprised basically of an ASCII tag character followed by one or two ASCII fields. The first field is numeric in value and the optional second field contains a symbol. (For additional familiarity with standard object code format, refer to the *Model 990 Computer Assembly Language Programmer's Guide*, part number 943441-9701). The first ASCII field in standard object code format is four characters (i.e., four bytes) in length which, when converted to compressed object code format, is changed to binary, two bytes in length. The second field in standard object code format is left unchanged when converting to compressed object code format. Records are terminated with the standard end-of-record tag character, only. The beginning-of-module-tag-character is an ASCII zero in standard object code format and a binary one in compressed object code format. This is used to distinguish between compressed and uncompressed modules. The end-of-module colon record, identified by the colon at the beginning of the last line of the module, is unchanged. The diskette is the only device capable of supporting compressed object code format.

ASCII Standard Object Code Format (e.g., from punched cards)

```
00008TASK          A0000B000AB020000000B00007F7EEF
:       TASK              021/77      12:32:54
```

Hexadecimal Representation of Standard Format

```
3030  3030  3854  4153  4B20  2020  2041  3030
3030  4230  3030  4142  3032  3030  4330  3030
3042  4330  3030  3746  3745  4546  2020  2020
```

ASCII Representation of Standard Format

```
00  00  8T  AS  K       A   00
00  B0  00  AB  02  00  C0  00
0B  C0  00  7F  7E  EF
```

Hexadecimal Representation of Compressed Format

```
0100  0854  4153  4B20  2020  2041  0000  4200
0A42  0200  4300  0042  C000  4600  0000  0000
```

ASCII Representation of Compressed Format

```
..  .T  AS  K       A   ..  B.
.B  ..  C.  .B  ..  F.  ..  ..
```

Colon Record for Both Formats Hexadecimal Representation

```
3A20  2020  2020  2054  4153  4B20  2020  2020
2030  3231  2F37  3720  2020  2031  323A  3332
3A35  3420  2020
```

ASCII Representation

```
:       T   AS  K
 0  21  /77      1  2:  32
:5  4   .
```

APPENDIX C

TASK STATE CODES

APPENDIX C

TASK STATE CODES

The user-task supervisor calls which return one of the task state codes listed in table G-1 to byte 1 of the supervisor call block are:

- Bid Task Supervisor Call
- Activate Suspended Task Supervisor Call
- Activate Time Delay Task Supervisor Call

The user may code his program to read out the task state code to an output device or, using the OCP STate (ST) command, the user can cause a terminal to print out the task state codes.

Table C-1. List of Task State Codes

Code (Hexadecimal)	Significance
00	Active task, priority level 0
01	Active task, priority level 1
02	Active task, priority level 2
03	Active task, priority level 3
04	Terminated task
05	Task in time delay
06	Suspended task
07	Currently executing task
08	Task awaiting VDT character input
09	Task awaiting completion of I/O
0A	Task queued for I/O
0B	Task queued for file utility routine
0C	Task on the diagnostic queue
0D	Task waiting for file management completion
10	Task queued for file management

APPENDIX D

I/O ERROR CODES

APPENDIX D

I/O ERROR CODES

Code (Hexadecimal)	Description
	DSR ERRORS
00	NO ERROR
01	ILLEGAL LUNO
02	ILLEGAL OPERATION CODE
03	LUNO IS NOT YET OPENED
04	RECORD LOST DUE TO POWER FAILURE
05	ILLEGAL MEMORY ADDRESS
06	TIME OUT, OR ABORT
07	ILLEGAL DEVICE
11	DEVICE ERROR
12	NO ADDRESS MARK FOUND
15	DATA CHECK ERROR
19	DISKETTE NOT READY
1A	WRITE PROTECT
1B	EQUIPMENT CHECK ERROR
1C	INVALID TRACK OR SECTOR
1D	SEEK ERROR OR ID NOT FOUND
1E	DELETED SECTOR DETECTED
	FILE MANAGEMENT ERRORS
20	LUNO IS IN USE
21	BAD DISC NAME
22	PATHNAME HAS A SYNTAX ERROR
23	ILLEGAL FUR OPCODE
24	BAD PARAMETER IN PRB
25	DISKETTE IS FULL
26	DUPLICATE FILE NAME
27	FILE NAME IS UNDEFINED
28	ILLEGAL LUNO
29	SYSTEM BUFFER AREA FULL
2A	SYSTEM CAN'T GET MEMORY
2B	FILE MANAGEMENT ERROR
2C	CAN'T RELEASE SYSTEM LUNO
2D	FILE IS PROTECTED
2E	ABNORMAL FUR TERMINATION
2F	SUPPORT FOR OPTION DOES NOT EXIST IN SYSTEM
30	NON-EXISTENT RECORD
3B	INVALID ACCESS PRIVILEGE
3E	FILE CONTROL BLOCK ERROR
3F	FILE DIRECTORY FULL

I/O ERROR CODES (Continued)

Code (Hexadecimal)	Description
	TASK LOADER ERROR
60	I/O ERROR, LOAD NOT COMPLETE
61	OBJECT MODULE CONTAINS NONRELOCATABLE OBJECT CODE
62	CHECKSUM ERROR LOAD ABORTED
63	LOADER RAN OUT OF MEMORY
64	TASK 10 IS BUSY
	VDT ERRORS
80	DEVICE NOT AVAILABLE VDT STATION NOT FOUND

Note:

Error Code >FF is a general error code.

ALPHABETICAL INDEX

ALPHABETICAL INDEX

INTRODUCTION

The following index lists key words and concepts from the subject material of the manual together with the area(s) in the manual that supply major coverage of the listed concept. The numbers along the right side of the listing reference the following manual areas:

- Sections - References to Sections of the manual appear as "Section x" with the symbol x representing any numeric quantity.
- Appendixes - References to Appendixes of the manual appear as "Appendix y" with the symbol y representing any capital letter.
- Paragraphs - References to paragraphs of the manual appear as a series of alphanumeric or numeric characters punctuated with decimal points. Only the first character of the string may be a letter; all subsequent characters are numbers. The first character refers to the section or appendix of the manual in which the paragraph is found.
- Tables - References to tables in the manual are represented by the capital letter T followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the table). The second character is followed by a dash (-) and a number:

  Tx-yy

- Figures - References to figures in the manual are represented by the capital letter F followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the figure). The second character is followed by a dash (-) and a number:

  Fx-yy

- Other entries in the Index - References to other entries in the index are preceded by the word "See" followed by the referenced entry.

USER'S RESPONSE SHEET

Manual Title: Model 990 Computer Terminal Executive Development System (TXDS) Programmer's Guide (946258-9701)

Manual Date: 15 December 1977 **Date of This Letter:** _____

User's Name: _____ **Telephone:** _____

Company: _____ **Office/Department:** _____

Street Address: _____

City/State/Zip Code: _____

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. If there are any other suggestions that you wish to make, feel free to include them. Thank you.

CUT ALONG LINE

Location in Manual	Comment/Suggestion
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), TAPE AND MAIL

FOLD

FIRST CLASS
PERMIT NO. 7284
DALLAS, TEXAS

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

TEXAS INSTRUMENTS INCORPORATED
DIGITAL SYSTEMS DIVISION

P.O. BOX 2909 · AUSTIN, TEXAS 78769

ATTN: TECHNICAL PUBLICATIONS
MS 2146

FOLD

TEXAS INSTRUMENTS
INCORPORATED
DIGITAL SYSTEMS DIVISION
POST OFFICE BOX 2909 AUSTIN, TEXAS 78769