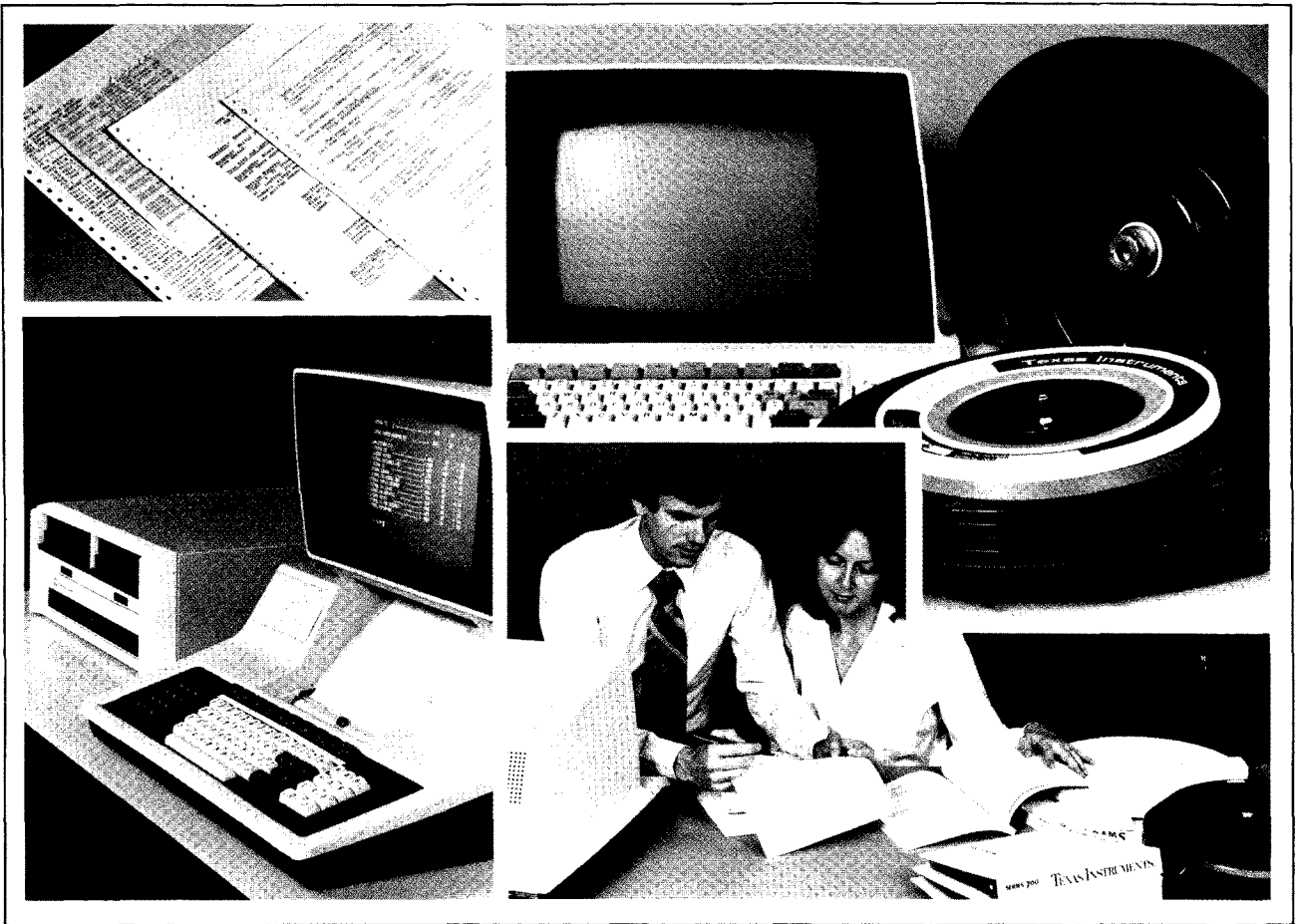

Model 990 Computer DX10 Operating System Developmental Operation Manual



Volume IV

Part No. 946250-9704*B
15 April 1981



TEXAS INSTRUMENTS

© Texas Instruments Incorporated 1977, 1979, 1981

All Rights Reserved, Printed in U.S.A.

The information and/or drawings set forth in this document and all rights in and to inventions disclosed herein and patents which might be granted thereon disclosing or employing the materials, methods, techniques or apparatus described herein, are the exclusive property of Texas Instruments Incorporated.

MANUAL REVISION HISTORY

Model 990 Computer DX10 Operating System Developmental Operation
Manual, Volume IV (946250-9704)

Original Issue	15 August 1977
Revision.....	15 December 1979
Revision.....	15 April 1981

The total number of pages in this publication is 140.



PREFACE

The *Model 990 Computer DX10 Operating System Manuals* describe the features of the 990 Disk Executive, DX10. Each of the six DX10 operating system manuals includes a specific level of discussion related to a particular aspect of the DX10 operating system. All phases of system operation are treated throughout the manuals to enhance the development and application of programs. The manuals are organized with both the applications programmer and the production operation in mind and provide details to the system or application programmer to allow the extension and/or modification of DX10.

The user should consult all six manuals to become thoroughly familiar with all facets and capabilities of the operating system. Each manual serves a particular purpose and is designed to meet a specific goal. No single manual is intended to stand alone as a complete system tutorial. The titles and part numbers of the six manuals along with a brief comment regarding content and level of each are as follows:

Concepts and Facilities, Model 990 Computer DX10 Operating System — Includes features, concepts, and general background information describing the DX10 operating system.

Production Operation, Model 990 Computer DX10 Operating System — Describes the user interface to the DX10 operating system and application programs. It contains information required to start-up, query, control, and maintain DX10 and describes the steps needed to run application programs on Texas Instruments 990/10 and 990/12 computer systems under control of DX10. Accordingly, this manual describes the commands an operator may use to enter, access, execute, and control the execution of application programs. Also included is an introduction to terminal operation, details of log-on/log-off and operation of each specific user terminal.

Application Programming Guide, Model 990 Computer DX10 Operating System — Includes information required by the application programmer in the preparation or modification of application programs. It is primarily intended for assembly language programming under DX10 but also supplies a reference for high-level language programmers. Included is a discussion of program design and structure under DX10. It provides detailed information describing all calls for system services including input/output (I/O) processing. Information is provided describing the DX10 file structures and detailed information describing calls to DX10 for file I/O processing.

Developmental Operation Manual, Model 990 Computer DX10 Operating System — Includes operating instructions for the programmer who is creating new application programs. This volume describes software packages provided as a part of DX10 to support program development and includes a description of the Text Editor, the debug commands, and program installation/deletion functions.

Systems Programming Guide, Model 990 Computer DX10 Operating System — Includes information required by the systems or application programmer for extending or modifying DX10. It provides detailed discussion in such areas as system generation, support of nonstandard devices, and privileged supervisor service calls available within DX10. Also included are detailed instructions and descriptions of how to add commands to the System Command Interpreter (SCI) and how to customize DX10 for a particular configuration and application.

Error Reporting and Recovery, Model 990 Computer DX10 Operating System — Includes information describing error reporting within DX10. It documents task errors, command errors, supervisor call (SVC) errors, SCI errors, magnetic tape and other I/O errors. This manual documents all DX10 errors in cross-reference table form and includes a resolution of each message and suggested recovery techniques.

**NOTE**

Additional, in-depth descriptions related to specific languages including FORTRAN, COBOL, BASIC, RPG II, TI Pascal, Assembly Language, and Query are found in manuals dedicated to the appropriate programming language. A Link Editor manual is provided as a separate volume that describes the application of the link edit function in a DX10 environment. Separate manuals describe the use of an optional Sort/Merge package and the DBMS package.

The following documents contain additional information related to the DX10 operating system:

Title	Part Number
<i>Model 990 Computer DX10 Operating System Concepts and Facilities Manual, Volume I</i>	946250-9701
<i>Model 990 Computer DX10 Operating System Production Operation Manual, Volume II</i>	946250-9702
<i>Model 990 Computer DX10 Operating System Application Programming Guide, Volume III</i>	946250-9703
<i>Model 990 Computer DX10 Operating System Systems Programming Guide, Volume V</i>	946250-9705
<i>Model 990 Computer DX10 Operating System Error Reporting and Recovery Manual, Volume VI</i>	946250-9706
<i>Model 990 Computer DX10 Operating System Release 3.4, System Design Document</i>	939153-9701
<i>Model 990 Computer DX10 Operating System — Release 3 COBOL Programmer's Guide</i>	946266-9701
<i>Model 990 Computer FORTRAN Programmer's Reference Manual</i>	946260-9701
<i>Model 990 Computer Report Program Generator (RPG II) Programmer's Guide</i>	939524-9701
<i>Model 990 Computer TI 990 BASIC Reference Manual</i>	2250304-9701
<i>Model 990 Computer TI Pascal User's Manual</i>	946290-9701
<i>Model 990 Computer TIFORM User's Guide</i>	2250374-9701
<i>Model 990 Computer Model 911 CRT Display Terminal Installation and Operation</i>	945423-9701
<i>Model 990 Computer Model 913 CRT Display Terminal Installation and Operation</i>	943457-9701



<i>Model 733 Terminal User's Guide</i>	945259-9701
<i>Model 743 KSR Data Terminal Operating Instructions</i>	984030-9701
<i>Model 820 KSR Terminal Operator's Manual</i>	2208225-9701
<i>Model 781 RO Terminal Operator's Manual</i>	2265935-9701
<i>Model 783 KSR Terminal Operator's Manual</i>	2265936-9701
<i>Model 785 Communications Terminal Operator's Manual</i>	2265937-9701
<i>Model 787 Communications Terminal Operator's Manual</i>	2265938-9701
<i>Model 743 KSR Terminal Operator's Manual</i>	984030-9701
<i>Model 745 Portable Terminal Operator's Manual</i>	984024-9701
<i>Models 763/765 Operating Instructions</i>	2203664-9701
<i>Models 763/765 Memory Terminals Systems Manual</i>	2203665-9701
<i>TIBOL Programmer's Guide</i>	2263354-9701
<i>Model 990 Computer System 990/10 and /12 Assembly Language Reference Manual</i>	2270509-9701
<i>990/10 Minicomputer Hardware Reference Manual</i>	945417-9701
<i>DX10 3270 User's Guide</i>	2250954-9701
<i>Model 990 Computer DX10 Remote Terminal Subsystem (RTS) System Generation and Programmer's Reference Manual</i>	2272034-9701
<i>Model 990 Computer DX10 Remote Terminal Subsystem (RTS) Operator's Guide</i>	2272055-9701
<i>Model 990 Computer DX10 Remote Terminal Subsystem (RTS) Hardware Installation Manual</i>	2272053-9701
<i>DX10 3780/2780 Emulator Object Installation Manual</i>	2250918-9701
<i>Model 990 Computer DX3780/2780 Emulator User's Guide</i>	946289-9701
<i>Model 990 Computer 306 and 588 Line Printers Installation and Operation</i>	945261-9701
<i>Model 990 Computer PROM Programming Module Installation and Operation</i>	945258-9701
<i>Model 990 Computer Communications System Installation and Operation</i>	945409-9701



<i>Model 990 Computer Communications Systems Software</i>	946236-9701
<i>Model 990 Computer Terminal Executive Development System (TXDS) Programmer's Guide</i>	946258-9701
<i>Model 990 Computer Model FD800 Floppy Disk System with International Chassis Installation and Operation</i>	2250697-9701
<i>Model 990 Computer Model FD1000 Flexible Disk System with International Chassis Installation and Operation</i>	2250698-9701
<i>Model 990 Computer Model 733 ASR/KSR Data Terminal Installation and Operation</i>	945259-9701
<i>Model 990 Computer Model 804 Card Installation and Operation</i>	945262-9701



TABLE OF CONTENTS

Paragraph	Title	Page
SECTION 1. INTRODUCTION		
1.1	General	1-1
SECTION 2. TEXT EDITOR		
2.1	Introduction	2-1
2.2	Terminal Usage	2-1
2.2.1	Hard Copy Terminal Operation	2-2
2.2.2	Video Display Terminal Usage	2-2
2.3	Text Editor Walk Throughs	2-4
2.3.1	Creating a New File	2-4
2.3.2	Editing an Existing File	2-5
2.4	Command Usage	2-8
2.4.1	Execute Text Editor — XE Command	2-12
2.4.2	Execute Text Editor With Scaling — XES Command	2-12
2.4.3	Quit Edit — QE Command	2-12
2.4.4	Recover Edit — RE Command	2-14
2.4.5	Copy Lines — CL Command	2-14
2.4.6	Delete Lines — DL Command	2-15
2.4.7	Move Lines — ML Command	2-15
2.4.8	Insert File — IF Command	2-15
2.4.9	Delete String — DS Command	2-16
2.4.10	Find String — FS Command	2-17
2.4.11	Replace String — RS Command	2-17
2.4.12	Modify Roll — MR Command	2-18
2.4.13	Modify Right Margin — MRM Command	2-18
2.4.14	Modify Tabs — MT Command	2-18
2.4.15	Show Line — SL Command	2-19
2.4.16	Save Lines — SVL Command	2-19
2.5	Edit Control Functions	2-20
2.5.1	Enter Command Mode	2-20
2.5.2	Edit/Compose Mode	2-20
2.5.3	New Line Function	2-20
2.5.4	Display/Suppress Line Numbers	2-21
2.5.5	Duplicate to Tab Function	2-21
2.5.6	Clear to Tab Function	2-21
2.5.7	Tab Function	2-21
2.5.8	Back Tab Function	2-21
2.5.9	Roll Up Function	2-21
2.5.10	Roll Down Function	2-22
2.5.11	Insert Line Function	2-22
2.5.12	Delete Line Function	2-22
2.5.13	Insert Character Function	2-22
2.5.14	Delete Character Function	2-22
2.5.15	Cursor Up Function	2-22
2.5.16	Cursor Down Function	2-23
2.5.17	Cursor Right Function	2-23



TABLE OF CONTENTS (Continued)

Paragraph	Title	Page
2.5.18	Cursor Left Function (Backspace)	2-23
2.5.19	Home Function	2-23
2.5.20	Erase Function	2-23

SECTION 3. ACTIVATING LANGUAGE PROCEDURES

3.1	Assembly Language Program Generation Runthrough	3-1
3.2	FORTRAN Program Generation Runthrough	3-6
3.3	COBOL Program Generation Runthrough	3-9
3.4	BASIC Program Generation Runthrough	3-13
3.5	RPG Source Program Entry	3-14
3.5.1	Compilation	3-14
3.5.2	Binding	3-16
3.5.3	Execution	3-16
3.6	Pascal Program Generation Runthrough	3-17

SECTION 4. LINK EDITOR USE ON DX10

4.1	Supported Features	4-1
4.2	Link Editor Operation With DX10	4-1

SECTION 5. INSTALLING, DELETING, AND MODIFYING PROGRAMS

5.1	Introduction	5-1
5.2	IT — Install Task	5-1
5.2.1	IRT — Install Real-Time Task	5-3
5.3	IP — Install Procedure	5-4
5.4	IO — Install Overlay	5-4
5.5	DT — Delete Task	5-5
5.6	DP — Delete Procedure	5-5
5.7	DO — Delete Overlay	5-5
5.8	Modifying Program File Entries	5-6
5.8.1	Modify Task Entry (MTE)	5-6
5.8.2	Modify Procedure Entry (MPE)	5-7
5.8.3	Modify Overlay Entry (MOE)	5-8

SECTION 6. EXECUTING PROGRAMS

6.1	Introduction	6-1
6.2	Executing an Assembly Language Task	6-1
6.2.1	Execute Task — XT	6-1
6.2.2	Execute Task and Suspend SCI — XTS	6-2
6.2.3	Execute and Halt Task — XHT	6-2
6.3	Executing Language Processors, Tasks from Language Processors, and Subsystems	6-2



TABLE OF CONTENTS (Continued)

Paragraph	Title	Page
SECTION 7. DEBUGGING SUPPORT		
7.1	General	7-1
7.2	Modes of Debugging	7-1
7.2.1	Unconditional Suspend	7-2
7.2.2	Command Parameter Syntax	7-2
7.2.3	Symbols	7-3
7.2.4	Expressions	7-4
7.3	Commands for All Tasks	7-6
7.3.1	AB — Assign Breakpoints	7-8
7.3.2	DB — Delete Breakpoints	7-8
7.3.3	PB — Proceed From Breakpoint	7-9
7.3.4	DPB — Delete and Proceed From Breakpoint	7-9
7.3.5	MM — Modify Memory	7-10
7.3.6	MSM — Modify System Memory	7-10
7.3.6.1	MSM Command Format	7-10
7.3.6.2	MSM Command User Responses	7-11
7.3.6.3	MSM Command Example	7-11
7.3.7	LM — List Memory	7-11
7.3.8	LSM — List System Memory	7-12
7.3.8.1	LSM Command Format	7-12
7.3.8.2	LSM Command User Responses	7-12
7.3.8.3	LSM Command Example	7-12
7.3.9	FW — Find Word	7-13
7.3.10	FB — Find Byte	7-13
7.3.11	AT — Activate Task	7-13
7.3.12	HT — Halt Task	7-14
7.3.13	RT — Resume Task	7-14
7.3.14	MIR — Modify Internal Registers	7-15
7.3.15	SIR — Show Internal Registers	7-16
7.3.16	MWR — Modify Workspace Registers	7-16
7.3.17	SWR — Show Workspace Registers	7-16
7.3.18	SP — Show Panel	7-17
7.3.19	LB — List Breakpoints	7-18
7.3.20	SV — Show Value	7-18
7.3.21	SPI — Show Program Image	7-19
7.3.22	MPI — Modify Program Image	7-19
7.3.23	SAD — Show Absolute Disk	7-20
7.3.24	MAD — Modify Absolute Disk	7-21
7.3.25	SADU — Show Allocable Disk Unit	7-22
7.3.26	MADU — Modify Allocable Disk Unit	7-23
7.3.27	SRF — Show Relative to File	7-24
7.3.28	MRF — Modify Relative to File	7-25
7.3.29	LLR — List Logical Record	7-25
7.4	Commands for Controlled Tasks	7-26
7.4.1	XD — Debug Command	7-26
7.4.2	ST — Simulate Task	7-27
7.4.3	ASB — Assign Simulated Breakpoint	7-28
7.4.4	DSB — Delete Simulated Breakpoint	7-29

**TABLE OF CONTENTS (Continued)**

Paragraph	Title	Page
7.4.5	RST — Resume Simulated Task	7-29
7.4.6	LSB — List Simulated Breakpoints	7-29
7.4.7	QD — Quit Debug	7-30
7.5	Station Dependents Displays	7-31

SECTION 8. EXAMPLE PROGRAM

8.1	Runthrough of an Assembly Language Example Program	8-1
-----	--	-----

APPENDIXES

Appendix	Title	Page
A	Standard Device Names	A-1
B	Command Summary	B-1

LIST OF ILLUSTRATIONS

Figure	Title	Page
2-1	Listing of Text Editor Example	2-6
2-2	Modified File Example	2-9
2-3	Modifications File Example	2-9
3-1	Example Assembly Language Program	3-4
3-2	Example COBOL Program	3-11
3-3	Compiler Output Listing	3-15
3-4	Input File	3-17
3-5	Output Listing	3-17
3-6	Example Pascal Program Source	3-20
3-7	Example Pascal Program Output	3-20
7-1	Debug Panel Display	7-18
7-2	Simulated Breakpoints Display	7-30

LIST OF TABLES

Table	Title	Page
2-1	Edit Control Functions	2-3
6-1	Locating Instructions for Executing Subsystems, Language Processors, and Tasks from Language Processors Available with DX10	6-3
7-1	SCI Debug Commands	7-6
7-2	Command Displays	7-31



SECTION 1

INTRODUCTION

1.1 GENERAL

This manual describes the steps involved in creating and executing a working computer program. These steps are illustrated in subsequent paragraphs by examples taken from each of the programming languages. To gain familiarity with DX10, it is suggested that each programmer perform the complete runthrough for at least one of the supported languages.

NOTE

The methods below do not use the full capabilities of DX10, but only illustrate one of several ways to develop software.

Steps in program generation include the following:

1. Planning and initial coding. This is a very important step, especially for long and complex procedures. Modern programming techniques, such as structured programming, can drastically reduce the number of errors committed at this step.
2. Enter the program into the computer. Under DX10, this is usually done using the Text Editor. Alternately, the program may be prepared on an external media (such as punched cards) and read into the computer. The program is called source code at this step.
3. Invoke the appropriate language processor. DX10 supports FORTRAN, COBOL, Pascal, BASIC, and assembly language. For interpretive languages like BASIC, steps 2, 3, and 5 are often combined. The output of the language processor is called object code.
4. Link edit the output of the language processor. This step ties segmented programs together. The output of the Link Editor is called linked object code. This step is omitted in BASIC programs. In some cases, it is also omitted in COBOL programs. Consult the COBOL and BASIC manuals for more information.
5. The executable program now resides on a disk file or on a sequential device such as cassette or magnetic tape. The next step is to execute the program and debug it. Historically, the most time consuming phase of program generation is program debugging. Proper attention to design and coding in step 1 can drastically reduce this time. If errors are detected, use the debug tools provided by DX10 to debug the program. The program should be corrected at the source code level (i.e., go back to step 2 and use the Text Editor to correct the program.).
6. All programs which are developed for use in a production system should include a standardized documentation form. Some characteristics of a good form are as follows:
 - a. A brief description of all globally defined routines:
 - Title of routine
 - Abstract description of process



- Calling sequence(s)
 - Error conditions
 - External references (including name of source file)
 - Stack requirements (assembly language)
- b. A brief description of all data structures:
- Abstract description of structure
 - List of all routines which have access to the structure
- c. A documentation of systems of interacting modules:
- A cross reference of globally-defined symbols
 - A description of the flow of control within the program
 - A user oriented introduction to the use and applications of the program.



SECTION 2

TEXT EDITOR

2.1 INTRODUCTION

The Text Editor provides the user with the means to create and modify files of textual data interactively. The data in these files may be assembly language source code, high-level language source code, or material that is to be printed, such as software documentation, memos, or drafts.

The interactive user invokes and operates the Text Editor from a Model 911 or 913 VDT, or a hard copy terminal, such as the Silent 700 Model 733 ASR/KSR or 820 KSR. Most of the editing functions are available at both the VDT and hard copy terminals, but the means of invoking a particular function may vary depending on the terminal type and its current mode of operation. While a hard copy terminal can operate only in the 'TTY' mode, the TI VDTs can operate in 'TTY' or 'VDT' mode. The Text Editor supports only the VDT mode when called from a VDT. The *Model 990 Computer DX10 Operating System Production Operation Manual*, contains the descriptions of these two modes and explains how the VDT is placed in either mode. The methods of data display and entry of the two types also differ so that the advantages of both may be fully utilized. These differences and methods are discussed in subsequent paragraphs within this document.

The Text Editor is invoked by use of the XE command. The prompt presented allows the user either to specify that data contained in an existing file be edited, or to indicate that new data is to be created.

When the Text Editor is used to modify the data in an existing file, the user specifies the file access name when the Text Editor is entered. Each of the records in the input file is numbered, relative to the start of the file. When the user makes modifications, the modifications are written to another file (the modifications file) and are not made directly to the input file. When records from the source file are deleted, the Text Editor indicates the deletions in the modifications file, but does not delete any records from the input file. When records are inserted, they are inserted in the modifications file, but not in the input file. Whenever the user requests display of data, the Text Editor builds the display by applying any modifications to the input file data, but the modifications are not made to the input file. When the editor is terminated by use of the QE command, the user may abort the edit, in which case all modifications and new data are discarded. If the user requests that a new file be created, the modifications are applied to the input file as it is written to the new output file. The original input file remains intact, unless the output file access name specified is the same as the input file access name, in which case the modified file replaces the input file.

Errors detected by the Text Editor are defined in the *Model 990 Computer DX10 Operating System Error Reporting and Recovery* manual.

2.2 TERMINAL USAGE

Text editing consists of four major types of operations, as defined in the following:

- Command selection and specification
- Edit control



- Data display
- Data entry

Command selection and specification include the selection of Text Editor functions that assist the operator with the management of the text in the source file. In addition, any SCI command may be called during Text Editor operation. SCI commands also include the entry of parameters, which are used by the command to perform the desired operation. Examples of the commands are Find String, Modify Tabs, and Copy Lines. These commands always have parameters that are supplied by the operator, or, in many cases, can be defaulted. After entry of each parameter, the TAB, NEW LINE, RIGHT FIELD, RETURN, or CARRIAGE RETURN key (depending on the terminal) is pressed to terminate the parameter entry. The terminal must be in the command mode before selecting any command. The terminal is returned to the command mode by pressing the appropriate Enter Command Mode function key listed in table 2-1.

Edit control consists of the operations that control the immediate editing of the data. All of these operations are available in VDT mode while only some are available in TTY mode (i.e., on hard copy devices). The operations available on hard copy devices are: altering the current file position, adding data by line, and deleting data by line. Additional operations allowed in VDT mode are altering cursor position, adding data by character, and deleting data by character. Edit control operations have no parameters.

Data entry operations control the actual entering of data into the file, and data display manages the display of data on the device.

2.2.1 HARD COPY TERMINAL OPERATION. A hard copy terminal is considered to be a relatively slow input/output device, as compared to a VDT. In addition, the function keys on hard copy terminals differ from those on the 911 and 913 VDTs. Each function available at a hard copy terminal is invoked by the operator selecting a specific character while simultaneously pressing the CONTROL key, thereby causing a unique control shift character to be generated.

Edit control operations are also selected by use of the control shift method. These operations require no parameters and are usually followed by a printout of a data record, which identifies the current position in the file.

Data display on hard copy terminals is on a line by line basis, and the line number or record number may be displayed. Data entry on a hard copy device is generally accomplished by retyping the desired segment of the displayed line, or by preceding the new line with a specified control shift character that indicates that the new line is to be inserted in the file immediately following the displayed line. The tab, clear to tab and duplicate to tab functions may be used to speed up this operation.

The specific control shift uses are defined along with each command in subsequent paragraphs.

2.2.2 VIDEO DISPLAY TERMINAL USAGE. The Text Editor supports the Models 911 and 913 VDTs. The Model 913 provides a 12-line display screen, while Model 911 provides a 24-line display screen.

Editing on the VDT occurs on a page basis, with a page being either 12 lines (Model 913 VDT) or 24 lines (Model 911 VDT). Any record displayed on the screen may be edited by positioning the cursor anywhere within the line that contains the record to be edited. Records may be inserted between any lines, and may be inserted or deleted in any order. In addition, characters within a line may be inserted, deleted, or modified. Positioning of the file for display is accomplished by use of the Show Line command, and the Roll Up, Roll Down, Cursor Up, and Cursor Down edit control functions.



Table 2-1. Edit Control Functions

Function	913 VDT Keypad	911 VDT Keypad	820 KSR Teleprinter Control
Enter Command Mode	HELP	CMD	X
Edit Compose Flip (1)	F7	F7	V
Disp/Suprs Line No (2)	F6	F6	F
Clear To Tab	F5	F5	E
Roll Up	ROLL UP	F1	A
Roll Down	ROLL DOWN	F2	B
Dup To Tab	F4	F4	D
New Line	NEW LINE	RETURN	CARG RETN
Tab (3)	TAB	TAB SKIP	I
Back Tab	BACK TAB	FIELD	T
Insert Line	INSERT LINE	Unlabeled Gray Key	O
Delete Line	DELETE LINE	ERASE INPUT	N
Insert Character	INSERT CHAR	INS CHAR	*****
Delete Character	DELETE CHAR	DEL CHAR	CNTLP
Cursor Up	↑	↑	U
Cursor Down	↓	↓	J
Cursor Right	→	→	*****
Cursor Left (Backspace)	←	←	H
Home	HOME	HOME	*****
Erase	CLEAR	ERASE FIELD	*****

- (1) Alternates modes on succeeding hits
(2) Alternates display of numbers (74 data characters) with no display of numbers (80 data characters)
(3) The SHIFT Key must be pressed simultaneously with the TAB SKIP key to achieve the tab function on the 911 VDT.

***** Function not supported

Character positions are numbered 1 through 80. This keeps the cursor position on the line being edited. However, it prevents any data from being entered in column 80. When line numbers are visible, 74 characters are available for text editing. When line numbers are suppressed, 80 characters are available. In the text edit mode (with line numbers suppressed) entering a character when the cursor is at position 80 of a line, automatically moves the cursor to the next line. The entered character is not displayed, and no additional characters can be entered. Pressing the cursor left character function key repositions the cursor over the character in the previous line, position 80. Text editing of that line can continue once the cursor is positioned to the left of or on position 80. Pressing the RETURN key positions the cursor in the first input position of line 3. Text editing on line 3 can begin.

NOTE

If you press the ERASE FIELD key after entering a character in position 80 of a line, (which moves the cursor automatically to the next line), all characters in the previous line are erased. The Modify Right Margin (MRM) command can be used to set the right margin at column 79 or less. This eliminates the line overflow problem. (Refer to the *Model 990 Computer DX10 Operating System Systems Programming Guide* for a description of the MRM command.)



Command selection from either VDT is accomplished by keying in the command and responding to the prompts presented on the display screen.

Edit control is performed by using the cursor control keys and some of the function keys. Although the functions are available on both VDT types, the method of selection varies in some cases due to the physical differences in the keyboards. These differences are defined along with the command definitions in this section.

2.3 TEXT EDITOR WALK THROUGHS

All the commands available under the Text Editor are described in paragraph 2.4, and all the edit control functions are described in paragraph 2.5. In this paragraph, simple walk throughs of the Text Editor are provided for creating a new file and for modifying an existing file. The purpose of these examples is to provide a quick reference for the more common uses of the Text Editor.

2.3.1 CREATING A NEW FILE. The following procedure applies to creating a new file using the Text Editor on a Model 911 Video Display Terminal in the VDT mode. The example assumes that you are properly logged on, and that the System Command Interpreter (SCI) is active. Refer to the *Model 990 Computer DX10 Operating System Production Operation* manual for details about logging on and activating SCI.

Key in XE and press the RETURN key to activate the Text Editor. The following prompt is presented:

```
INITIATE TEXT EDITOR
FILE ACCESS NAME:
```

Press the TAB/SKIP key to indicate that no input file is to be edited. The screen is cleared and the following display is presented in the first four columns of line one, with the cursor in column one, line one:

```
* EOF
```

This display indicates that the only record in the file is the end-of-file record. Because no file access was specified, the Text Editor begins in the compose mode. Press the RETURN key. The following display is then presented:

```
* EOF
```

Note that the end-of-file record is now in line two and that the cursor is in line one, column one, with the rest of the line blank-filled. You may now begin entering data by simply keying the data and pressing the RETURN key whenever you wish to terminate a line. In the example (shown in figure 2-1), 35 records were entered with each record containing ten As.

Any of the edit control functions may be used during data entry, as may any of the commands (prior to entering a command, press the CMD key). To begin entering new data, position the cursor to the last data line; to do so, use the cursor control keys, such as Up Arrow (↑) and Down Arrow (↓). Then, press the RETURN key to insert a line. You may now begin entering data. Once all the data has been entered, the Text Editor is terminated by calling the QE (Quit Editor) command. First,



press the CMD key to enter the command mode, then enter QE and press the RETURN key. The following prompt appears:

```
QUIT EDIT
  ABORT?: NO
```

The reply to the ABORT prompt allows you to either accept (N response) or discard (Y response) the data you entered. Since you want to accept the data, press the RETURN key to accept the initial (N) value. The following display is presented:

```
QUIT EDIT
OUTPUT FILE ACCESS NAME:
                      REPLACE?  NO
MOD LIST ACCESS NAME:
```

The cursor appears after the colon in the first prompt of the display. Enter the pathname of the file to which the entered data is to be written. The pathname can be an existing file or a new file name. You must make an entry since there is no input file. If you had used an input file, its pathname would be displayed and you could accept that pathname. Enter .EXAMPLE as the pathname.

If the file specified by the pathname currently exists and you want to replace it, respond to the REPLACE?: prompt with a Y. If you are creating a new file, respond with an N. The N response allows you to avoid accidentally destroying an existing file. The example uses an N response.

Press the RETURN key in response to the MOD LIST ACCESS NAME prompt. Since a new file is being created, there are no modifications.

Once the file has been created/replaced, the Text Editor is no longer active and the terminal returns to command mode. The initial SCI menu is displayed.

The file created in the example is shown in figure 2-1.

2.3.2 EDITING AN EXISTING FILE. The example described in this paragraph gives the general procedures for editing an existing file by using the Text Editor. The file used as input is the one created in paragraph 2.3.1 and shown in figure 2-1. The editing takes place at a 911 VDT in the VDT mode.

First, enter the command mode (press the CMD key) and key in XE, followed by pressing the RETURN key. The following is displayed:

```
INITIATE TEXT EDITOR
  FILE ACCESS NAME:
```

The pathname .EXAMPLE is repeated automatically from the previous example. Press the RETURN key to accept the pathname and display the first 24 records from the file. The Text Editor is in the edit mode, the cursor is in column one, line one, and line numbers are displayed. This example changes lines one and two, inserts one line after line nine, inserts one line after line 19, changes line 20, and deletes lines 30 through 35.



When the prompt appears, key in SL, which invokes the Show Line command, and then press RETURN. The following display appears:

```
SHOW LINE
      LINE: 1
```

Enter the number of the line that the inserted line is to precede. For the example, enter 10 and press the RETURN key. The display then appears with the first line being line 10, and the cursor in column one of that line. Press the unlabeled gray key and the display is rolled down one line and the line containing the cursor is blank filled. The new line, **BBBBBBBBBB**, is then entered and the RETURN key is pressed. The cursor goes to column one of line ten. Note that no line number is displayed along with the inserted line.

To get to line 20 to insert the next line, use the other method of advancing the cursor, which consists of using the down arrow (\downarrow) key. Each depression of the key causes the cursor to go down one line in the display. The cursor remains in the same column. To get to line 20, press the \downarrow key ten times. Press the unlabeled gray key, which causes the lines below and including line 20 to be rolled down one line and the line containing the cursor to be blank filled. Key in the inserted line as follows:

```
BBBBBBBBBB
```

and press the RETURN key to return the cursor to column one of line 20.

The cursor is now in line 20. Change the **AAAAAAAAAA** to **CCCCCCCC** by keying the new value and pressing the RETURN key.

The final change is to delete lines 30 through 35. To do this, press the CMD key to enter the command mode. When the command prompt ([]) is displayed, enter DL and press the TAB key to call the Delete Lines command. The following display is presented:

```
DELETE LINES
  START LINE:
  END LINE:
```

The responses to the prompts are 30 and 35, as follows:

```
DELETE LINES
  START LINE: 30
  END LINE: 35
```

Press the RETURN key after you enter each line number. Since line 35 is the last input line from the file **.EXAMPLE**, the following display is presented:

```
* EOF
```

Since all modifications are completed, call the QE command to terminate processing. When you press the CMD key, the command prompt ([]) appears. Enter QE, press the RETURN key, and the following appears:

```
QUIT EDIT
  ABORT?: NO
```



To save the modifications, press the RETURN key to accept the N value. The following is displayed:

```
QUIT EDIT
  OUTPUT FILE ACCESS NAME:  . EXAMPLE
                        REPLACE?:  N
  MOD LIST ACCESS NAME :
```

Note that the value entered in response to the FILE ACCESS NAME: prompt of the XE command is used as the default here. Accept this value by pressing the RETURN key.

Enter Y in response to the REPLACE parameter to indicate that the input file is to be replaced. Press the RETURN key and enter the value .LISTA in response to the MOD LIST ACCESS NAME prompt. Press the RETURN key. The Text Editor now terminates and the initial SCI menu is displayed.

Figure 2-2 illustrates the file after the changes and figure 2-3 lists the modifications. On the modifications list, the flags at the left margin have the following meanings:

- O — Old value
- N — New value
- I — Inserted line
- D — Deleted line.

To review the modifications file, use the Show File (SF) command with .LISTA as the pathname. To prepare a printed listing, use the Print File (PF) command.

2.4 COMMAND USAGE

The Text Editor is initiated when the operator selects and completes the XE command; the Text Editor terminates when the operator enters and completes the QE command. Whenever the terminal is in the command mode, the Text Editor is suspended and the operator may select any command. Note that the commands selected when the terminal is in the command mode and the Text Editor is suspended do not have to be Text Editor commands. The Text Editor remains suspended until the XE command or another Text Editor command is selected, at which point the Text Editor is re-activated, the state that existed at the time of suspension is restored, and the entered command is processed. Any Text Editor command entered after the Text Editor has been terminated with the QE command causes the following message to be displayed:

COMMAND ALLOWED ONLY WHILE TEXT EDITING:

If the operator quits the command interpreter (by entering the Q command) while the editor is suspended, the QE command is automatically invoked.



```
BAAAAAAAAA
AAAAAAAAAB
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
BBBBBBBBBB
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
BBBBBBBBBB
CCCCCCCCCC
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
```

Figure 2-2. Modified File Example

```
O 1 AAAAAAAAAA
N 1 BAAAAAAAAA
O 2 AAAAAAAAAA
N 2 AAAAAAAAAAB
I   BBBBBBBBBB
I   BBBBBBBBBB
O 20 AAAAAAAAAA
N 20 CCCCCCCCCC
D 30 AAAAAAAAAA
D 31 AAAAAAAAAA
D 32 AAAAAAAAAA
D 33 AAAAAAAAAA
D 34 AAAAAAAAAA
D 35 AAAAAAAAAA
```

Figure 2-3. Modifications File Example



Within the command descriptions in this section, the following conventions are used in the formats of the commands:

Underscore (<u> </u>)	Indicates an initial value supplied by the command.
Braces ({ })	Indicates that a selection of the entries must be made, or the default value used. The default value is underlined.
Brackets ([])	Indicates that the entry is optional.
acnm	Stands for Access Name for file or device.
int	Indicates that the entry may be an integer or an expression.
prev val	The previously entered value for the prompt. SCI assigns the previous value as the default for many of the prompts.
Boldface Type	Indicates the operator keyed and system displayed command.
Uppercase	System displayed data.
Angle Brackets (< >)	Data which must be entered by the operator.

Many commands have prompts that request start line and end line values. The int expression indicates these values. The following format applies to start line entries:

<i>i</i>	=	A numbered line in the input file. This value refers to an existing numbered line. A zero value is not allowed.
<i>i±j</i>		A line referenced by its offset ($\pm j$) from a numbered line (<i>i</i>). This value is used to reference a newly entered line. Neither <i>i</i> nor <i>j</i> may be zero. When a negative value is entered and the cursor is on line 1, the default is to the first line.
$\pm j$	=	A line referenced by its offset ($\pm j$) from the current line. In the VDT mode, the current line is the line the cursor was in when the command was selected. A zero value is not allowed.
Default	=	The current line.

When the int expression refers to an ending line parameter value, the following conventions apply:

<i>i</i>	=	A numbered line in the input file. A zero value is not allowed.
<i>i±j</i>	=	The ending line is offset ($\pm j$) from a numbered line (<i>i</i>) in the input file. Neither <i>i</i> nor <i>j</i> may be zero.



- +j = The ending line is offset (+j) from the specified starting line (a negative or zero value is not permitted).
- Default = The value of the starting line parameter.

NOTE

If a negative value is given for the end line parameter value, the display goes back the amount of the requested value from where the cursor is located.

Within the command descriptions, current file is the input file with all modifications. Note, however, that the modifications are not actually applied to the input file until the Text Editor is terminated, and the input file is replaced, if so specified.

The commands defined in this section are the following:

- XE — Execute Text Editor
- XES — Execute Text Editor with Scaling
- QE — Quit Editor
- RE — Recover Edit
- CL — Copy Lines
- DL — Delete Lines
- DS — Delete String
- FS — Find String
- IF — Insert File
- ML — Move Lines
- MR — Modify Roll
- MRM — Modify Right Margin
- RS — Replace String
- MT — Modify Tabs
- SL — Show Line
- SVL — Save Lines



2.4.1 EXECUTE TEXT EDITOR — XE COMMAND. The XE command activates the Text Editor. When the command is entered, the system generates the following display:

INITIATE TEXT EDITOR

FILE ACCESS NAME: { <acnm> }
 { prev val }

Response to the FILE ACCESS NAME prompt is optional and, if entered, identifies the existing file that is to provide the data input for the Text Editor. If no file name is supplied, a new file is created which is named when the Quit Editor command is processed. The FILE ACCESS NAME entry cannot reference a device.

If the Text Editor is active when the XE command is entered, the system restores the state that existed when the Text Editor was suspended (that is, a non-Text Editor command was called) and the data displayed by the last Text Editor command or function is restored. The file selected when the Text Editor was originally entered remains the input file.

NOTE

Do not use the XE command to text edit from a logging or attention device.

2.4.2 EXECUTE TEXT EDITOR WITH SCALING — XES COMMAND. The XES command is identical to the XE command except that in VDT mode scaling information is displayed on the bottom line of the screen. The scaling information helps the user determine the column numbers associated with the data on the screen. The information has the following format: a 1 is displayed in column 10, a 2 in column 20, a 3 in column 30, etc. Use XES to initiate the Text Editor when editing source code in a language with column-oriented input.

NOTE

Do not use the XES command to text edit from a logging or attention device.

2.4.3 QUIT EDIT — QE COMMAND. The QE command is entered when all processing desired under control of the Text Editor has been completed. When the QE command is keyed, the following display is presented:

QUIT EDIT

ABORT?: { <Y> }
 { NO }

The response to the ABORT prompt allows the operator to immediately terminate the Text Editor without any modification to the file specified in the XE command, or, if no file was specified in the XE command, without any new file being created. Note that any modifications



made or data entered are lost when the ABORT prompt is answered with a Y. When the response to the ABORT prompt is Y, the Text Editor is terminated and no other prompts are presented. If the default response N is used, the following display is presented:

```
OUTPUT FILE ACCESS NAME:  {<acnm>
                           input acnm}
REPLACE?:                 {<Y>}
                           N}
MOD LIST ACCESS NAME:    [<acnm>]
```

The default response to the OUTPUT FILE ACCESS NAME prompt is the value entered when the XE command was processed. If no name was specified in the XE command, the user must enter the name of the output file in response to the OUTPUT FILE ACCESS NAME. This response is also entered if the user wishes to create a new file that contains the contents of the original file (input file specified in the XE command) as modified by the Text Editor. The response can also be the access name of another existing file that is to contain the modified file. The OUTPUT FILE ACCESS NAME entry cannot reference a device.

The response to the REPLACE prompt allows the user to specify whether the file specified in response to the OUTPUT FILE ACCESS NAME prompt, which may be the input file, is to be replaced by the modified version of the file. If the response is Y, the modified file replaces the specified file, or, if no file exists by that name, a new file is created. If the default entry N is made and the specified file does not exist, the Text Editor creates a new file containing the modified file, which is identified by the OUTPUT FILE ACCESS NAME entry. If the file specified by the OUTPUT FILE ACCESS NAME entry does exist and the response to the REPLACE prompt is N, an error message is displayed indicating that a file I/O error has occurred. The user should then reenter the QE command and either replace the existing file by responding Y to the REPLACE prompt or create a new file by entering a new OUTPUT FILE ACCESS NAME.

The method by which the characteristics of the output file are chosen is as follows:

1. If no input file was specified and the output file does not exist, the output file is created as a sequential file with an initial allocation equal to one-fourth of the number of records created with the Text Editor, with an incremental allocation value of one-sixteenth of the number of records created. This algorithm assures efficient use of disk space with minimum execution overhead.
2. If the output file is to replace the input file, the characteristics of the input file are used, with the exception of the allocation values, which are described in step 1.
3. If the input file exists and the output file does not exist (the output file is not to replace the input file), the characteristics of the input file are used, with the exception of the allocation values, which are as described in step 1.
4. If the output file exists, its characteristics are used, except for the allocation values.

To create a relative record file, the relative record file must first be created by use of the Create File (CF) SCI command procedure, and its assigned access name specified when the Text Editor is initiated and terminated.

**NOTE**

Key indexed files cannot be edited by the Text Editor.

The user specifies an access name in response to the MOD LIST ACCESS NAME prompt if a printable copy of the modifications is to be prepared. If the name of a device is entered, the listing is written to that device. If no entry is made, no modifications listing file or printout is provided.

2.4.4 RECOVER EDIT — RE COMMAND. The RE command allows partial and sometimes total recovery of text edits after system crashes or power failures. This command must be entered as the first command after a system has been rebooted and initialized and must be entered from the terminal at which the text edit was originally in progress. When the command is entered, the system displays the following:

```
RECOVER EDIT
OUTPUT FILE ACCESS NAME:
                        REPLACE?: NO
MOD LIST ACCESS NAME:
```

These prompts are identical to those that appear for the Quit Edit (QE) command after the user responds ABORT? = NO. Response to the OUTPUT FILE ACCESS NAME prompt is the value entered when the XE command was processed prior to the system crash or power failure. If no name was specified in the XE command, the user must enter the name of the output file in response to the OUTPUT FILE ACCESS NAME. This response is also entered to create a new file that contains the contents of the original file (input file specified in the XE command) as modified by the Text Editor. The response can also be the access name of another existing file that is to contain the modified file. The OUTPUT FILE ACCESS NAME entry cannot reference a device.

The user specifies an access name in response to the MOD LIST ACCESS NAME prompt if a printable copy of the modifications is to be prepared. If the name of a device is entered, the listing is written to that device. If no entry is made, no modifications listing file or printout is provided.

WARNING

If the user enters XE after a system crash or power failure, the Text Editor deletes and recreates the work files. Consequently, recovery of the previous edit session is no longer possible.

If the user enters an RE command while an edit session is in progress, an error message is returned. Also, if the user enters an RE and no edit was in progress prior to the system crash or power failure, an error message is returned.

2.4.5 COPY LINES — CL COMMAND. The CL command specifies which lines are to be copied from the current file to another position in the modifications file. The following display is presented when the CL command is entered:

```
COPY LINES
START LINE:  { <int>
               current line }
END LINE:    { <int>
               current line }
INSERT AFTER LINE: { <int>
                     current line }
```



The command copies the lines from the START LINE to the END LINE, inclusive, from the current file to a temporary area, positions the modification file to the line specified in the INSERT AFTER LINE entry and inserts the copied lines after this line. Note that the copied lines are not deleted from the input file. Any modifications previously made to the lines being copied are applied to the input file data as the lines are copied. No more than 380 lines may be copied in a single CL command.

2.4.6 DELETE LINES — DL COMMAND. The DL command specifies those lines that are to be deleted from the current file. The format of the DL command display is as follows:

```
DELETE LINES
  START LINE:  { <int>
                { current line }
  END LINE:    { <int>
                { current line }
```

This command causes the lines from the START LINE to the END LINE, inclusive, to be deleted from the current file. The new current line is the line immediately following the deleted block.

2.4.7 MOVE LINES — ML COMMAND. The ML command moves lines from one position in the current file to another position in the modifications file and deletes the specified lines from their original positions. The format of the ML command is as follows:

```
MOVE LINES
  START LINE:  { <int>
                { current line }
  END LINE:    { <int>
                { current line }
  INSERT AFTER LINE: { <int>
                       { current line }
```

This command copies the lines from the START LINE to the END LINE, inclusive, from the current file to a temporary area, deleting each line from the current file as it is copied. The modifications file is positioned to the line specified by the INSERT LINE entry, and the copied lines are inserted after this line. After the ML command is executed, the lines moved are displayed on the VDT screen, with the cursor positioned at the first line on the screen. No more than 380 lines may be moved in a single ML command.

2.4.8 INSERT FILE — IF COMMAND. The IF command inserts an entire file into the edited file after a specified record. The format of the command is as follows:

```
INSERT FILE
  INSERT AFTER LINE:  { <int>
                       { current line }
  FILE PATHNAME      <acnm>
```



In response to the **INSERT AFTER LINE** prompt, the user enters the line number after which the file is to be inserted in the edited file. If the file is to be inserted before the first line, place the cursor on the first line and perform an insert line function. The cursor is positioned on the newly created line, which becomes the new first line of the edit file. Perform an Insert File command, entering the desired file pathname but leaving the **INSERT AFTER LINE** prompt blank. The file is inserted after the blank line. Delete the blank line at the beginning of the file.

In response to the **FILE PATHNAME** prompt, the user enters the access name of the file that is to be inserted in the edited file. Note that the file specified in response to this request is unchanged. There is no default for the **FILE PATHNAME** prompt.

2.4.9 DELETE STRING — DS COMMAND. The DS command causes the system to search each line in the current file, beginning with the current line, for the specified string of characters and to delete the string when it is found. The format of the command is as follows:

```
DELETE STRING
  NUMBER OF OCCURRENCES:  { <int>
                           { 1 }
  START COLUMN:           { <int>
                           { prev val }
  END COLUMN:             { <int>
                           { prev val }
  STRING:                 <string>
```

The user enters the number of times the string is to be deleted in response to the **NUMBER OF OCCURRENCES** prompt. The system searches each line, beginning with the current line, for the specified string and deletes it each time it is found until the number of occurrences specified has been reached. The default response for the **NUMBER OF OCCURRENCES** is one.

The user enters the column in which the string begins in response to the **START COLUMN** prompt, and the ending column of the string in response to the **END COLUMN** prompt. For example, if the entries were 5 and 20, respectively, the system would delete the string in positions 5 through 20 of each line, beginning with the current line.

In response to the **STRING** prompt, the user enters the actual string of characters that is to be deleted. There is no default to the **STRING** prompt. When the string is found and deleted, all characters to the right of the string are shifted left to fill the area occupied by the string and the remainder of the record is blank filled. When the specified number of occurrences of the string have been deleted, the file is positioned at the last record from which the string was deleted and that record is displayed. If the specified number of occurrences is greater than the actual number of occurrences, the file is positioned at the end-of-file position.

As a rule, leading and trailing blanks are stripped from the string. If the desired string contains one or more blanks (or blanks only), the entire string must be enclosed in double quotes.



2.4.10 FIND STRING — FS COMMAND. The FS command allows the user to specify a character string to be found by the system, beginning with the current line. When the specified occurrence of the string is found, the system displays the line that contains the string. The format of the FS command is as follows:

```
FIND STRING
OCCURRENCE NUMBER:  { <int>
                      {  1
START COLUMN:        { <int>
                      { prev val }
END COLUMN:          { <int>
                      { prev val }
STRING:              <string>
```

In response to the OCCURRENCE NUMBER prompt, the user enters a number which corresponds to the number of times the string is to be found before the line containing the string is displayed. For example, if the parameter entered is 5, the system bypasses the first four occurrences of the string and only displays the fifth line that contains the string. The default value for the prompt is one. If the specified number of occurrences is greater than the actual number of occurrences, the file is positioned at end-of-file.

The user enters the column in which the string begins in response to the START COLUMN prompt, and the ending column of the string in response to the END COLUMN prompt. For example, if the entries were 5 and 20, respectively, the system would search for the string in positions 5 through 20 of each line, beginning with the current line.

The user enters the actual string of characters to be found in response to the STRING request. There is no default for the STRING prompt. Double quotes must enclose strings containing leading or trailing blanks, or blanks only.

2.4.11 REPLACE STRING — RS COMMAND. The RS command allows the user to replace a specified number of occurrences of a string of characters with a new string, beginning with the current line. The format of the RS command is as follows:

```
REPLACE STRING
NUMBER OF OCCURRENCES: { <int>
                       {  1
START COLUMN:          { <int>
                       { prev val }
END COLUMN:            { <int>
                       { prev val }
STRING:                { <string>
                       { prev change string }
CHANGE:                <string>
```

In response to the NUMBER OF OCCURRENCES prompt, the user enters the number of times that the existing string is to be replaced by the new string. The search for the specified existing string begins with the current line and ends when the specified number of occurrences have been found and replaced. The file is positioned at the last line in which the string was found and replaced. The default value for this request is one. If the specified number of occurrences is greater than the actual number of occurrences, the file is positioned at end-of-file.



In response to the START COLUMN prompt, the user enters the position (column) in which the search begins.

In response to the END COLUMN prompt, the user enters the position (column) in which the search ends.

In response to the STRING prompt, the user enters the string that is to be modified (search string). In response to the CHANGE prompt, the user enters the new string (replacement string). There are no defaults for these entries. Double quotes must enclose strings containing leading or trailing blanks, or blanks only.

Once all the parameters have been entered, the Text Editor examines each line in the edited file, beginning with the current line, and checks between the specified columns for the search string. Each time the string is found, it is replaced by the replacement string. Note that if the new string has fewer characters than the existing string, the characters to the right of the ending column are shifted to the left and the record is filled with trailing blanks. If the new string is larger than the existing string, the characters to the right of the ending column and to the left of the right margin are shifted right and some may be lost.

2.4.12 MODIFY ROLL — MR COMMAND. The MR command changes the increment or decrement applied on a ROLL command or ROLL UP or ROLL DOWN function. The format of the MR command is as follows:

MODIFY ROLL

NUMBER OF LINES TO ROLL: $\left\{ \begin{array}{c} \langle \text{int} \rangle \\ \underline{12} \end{array} \right\}$

In response to the NUMBER OF LINES TO ROLL prompt, the user enters the number of lines by which the display is to be rolled up (incremented) or rolled down (decremented) each time the ROLL UP or ROLL DOWN function key is pressed on the terminal. The default value is 12 lines.

2.4.13 MODIFY RIGHT MARGIN — MRM COMMAND. The MRM command allows the user to change the edit line right margin. The system does not allow characters to be entered to the right of the right margin position. The format of the MRM command is as follows:

MODIFY RIGHT MARGIN

RIGHT MARGIN POSITION: $\left\{ \begin{array}{c} \langle \text{column} \rangle \\ \underline{\text{prev val}} \end{array} \right\}$

The value of column may be any integer from 1 to 80, with a value of 0 being accepted but changed to 1. Note that the initial value is 80, but once a new value is entered, it becomes the default.

2.4.14 MODIFY TABS — MT COMMAND. The MT command changes or clears the tab settings of the device. The format of the MT command is as follows:

MODIFY TABS

TAB COLUMNS: $\left\{ \begin{array}{c} \langle \text{int} \rangle \\ \underline{\text{prev val}} \end{array} \right\}$



The values entered in response to TAB COLUMNS specify the columns where the cursor (on a VDT) or the print head (on a hard copy terminal) is to be positioned when the TAB key (on a VDT) or the CONTROL key and the I key (on a hard copy terminal) are pressed. If no value is entered, the current tab position remains unchanged. If only the value 0 is entered, all tabs are cleared except for the implied tab at column one. The values entered for the tab positions are separated by commas and may be in any order. Note that if values are entered, there is no implied tab stop at column one. The initial values are 1, 8, 13, 26, and 31, but once new values are entered, they become the defaults. This means that when the cursor or print head is repositioned to the left margin during text editing, it is positioned at the first tab position specified in the MT command.

2.4.15 SHOW LINE — SL COMMAND. The SL command positions the current file to any specified line (record). When the user enters SL, the system presents the following:

SHOW LINE:

LINE: $\left(\begin{array}{c} \langle B \rangle \\ \langle E \rangle \\ \langle \text{int} \rangle \\ \underline{1} \end{array} \right)$

In response to the LINE prompt, the operator may enter any one of the following:

- B — The display presents the first line (record) from the edited file.
- E — The display presents the last line (record) from the edited file.
- int — The entry is presented to the system in the format for the int parameter, as previously defined in this section. The display is of the specified line (record). For example, entering 8 positions the file at record 8.
- Default — The default parameter is '1'.

2.4.16 SAVE LINES — SVL COMMAND. The SVL command copies selected lines from the current edit file to an external file specified by the user. The format of the SVL command is as follows:

SAVE LINES

START LINE: <int>

END LINE: <int>

SAVE FILE PATHNAME: <int>

REPLACE (YES/NO): NO <Yes or No>

All edit lines between the lines specified in the START LINE and END LINE parameters are written to the file specified in the SAVE FILE PATHNAME parameter. If no value is input for the START LINE parameter, the SVL command defaults to the line of the edit file on which the screen's cursor is located. This rule also applies for the END LINE parameter.



2.5 EDIT CONTROL FUNCTIONS

Edit Control functions are those that permit the operator to specify to the Text Editor precisely where within the file the modifications are to be made. Refer to table 2-1 for the edit control functions supported and the keys by which the functions are called on the 911 VDT, the 913 VDT, and the teleprinter devices. The user should note that not all of the edit control functions are supported by hard copy devices.

Within the following descriptions of the edit control functions, all references to the file imply the input file, as modified by previous operations. Modifications are effective only as an option of the QE command.

2.5.1 ENTER COMMAND MODE. The enter command mode function is called by pressing the ENTER COMMAND MODE function key specified in table 2-1 for the appropriate terminal type. Calling this function causes the Text Editor to be suspended. The system then prompts the operator in the selection of a command, which need not be a Text Editor command.

2.5.2 EDIT/COMPOSE MODE. The two modes, edit and compose, under which the Text Editor operates are selected by the use of a toggle key. Successive depressions of the toggle key causes the mode to switch from edit to compose to edit and so forth. For example, if the compose mode is active and the toggle key is pressed, the mode switches to edit. If the key is pressed again, the mode becomes compose. The Text Editor is initiated in the edit mode when a file access name is specified and in the compose mode when no file access name is specified.

The compose mode of the Text Editor is used to enter a large volume of data into a file being edited, or to create a new file. The edit mode is used to make modifications to existing records in the current file, to delete records, or to insert relatively few records at the current position in the file. All edit control functions operate the same way in the edit and compose modes, with the exception being the new line function, which is discussed in a subsequent paragraph. Refer to table 2-1 for the correct EDIT/COMPOSE MODE KEY of the different terminal types.

2.5.3 NEW LINE FUNCTION. The new line function is called by pressing the appropriate new line function key (refer to table 2-1) of the terminal type in use. Note that the new line function operates differently in the compose and edit modes, and it operates differently on the VDT and hard copy device.

In both modes on the hard copy device, selection of the new line function causes the current typed line to be passed to the Text Editor as a new record. In the compose mode, the record is inserted at the current file position, or following previously inserted records at the current file position. A carriage return line feed also occurs. In the edit mode, the current typed line replaces the current record in the file up to the printhead position, and the next record from the file is printed with a carriage return line feed taking place.

On either VDT, the new line function causes the current keyed line to be passed to the Text Editor as a new record. In the compose mode, the record is inserted at the current file position, or following previously inserted records at the current file position. In addition, all lines above and including the one containing the cursor are rolled up one line, and the one containing the



cursor is blank filled. In the edit mode, the current keyed line replaces the current file record and the cursor is positioned at the first position of the next line. In either mode, if the cursor is in the bottom line when the new line function is called, all data lines are rolled up one line and the next record is displayed on the bottom line.

2.5.4 DISPLAY/SUPPRESS LINE NUMBERS. Displaying or suppressing line numbers on the display is selected by a toggle key. Successive depressions of the toggle key cause the line numbers to be suppressed, displayed, suppressed, and so forth. For example, if line numbers are currently being displayed and the toggle key is pressed, the display is refreshed without the line numbers. If the toggle key is pressed again, the display is refreshed with the line numbers. Note that inserted lines are always displayed without line numbers.

When line numbers are displayed, only 74 data characters of each record can be displayed. When line numbers are suppressed, a full 80-characters can be displayed. Refer to table 2-1 for the correct DISPLAY/SUPPRESS LINE NUMBERS key of the terminal type in use.

2.5.5 DUPLICATE TO TAB FUNCTION. The duplicate to tab function is called by pressing the appropriate duplicate to tab function key (refer to table 2-1) of the terminal type in use. Calling this function causes the data from the previous record to be copied into the current line, from the current cursor (printhead) position to the next tab stop or end of record. For example, if the cursor (printhead) is in position 35 and the next tab stop is in position 65, the data from the previous record positions 35 through 64, inclusive, is copied into the corresponding positions of the current line.

2.5.6 CLEAR TO TAB FUNCTION. The clear to tab function is called by pressing the appropriate clear to tab function key (refer to table 2-1) of the terminal type in use. Calling this function causes the data from the current cursor (printhead) position to the next tab stop or end of record position to be cleared. For example, if the cursor (printhead) is in position 35 and the next tab stop is position 65, calling the clear to tab function causes the data from position 35 through 64, inclusive, to be cleared.

2.5.7 TAB FUNCTION. The tab function is called by pressing the appropriate TAB key (refer to table 2-1) of the terminal type in use. When the Tab function is called, it causes the cursor on a VDT, or the print carriage on a hard copy device, to be positioned at the next tab stop to the right of the current position within the line. If there is no tab stop between the current position and the end of the line, the first tab stop of the line is used. On a VDT, the edit line remains unchanged, while on a hard copy device the data from the current line is printed from the current position to the next tab stop. The tab settings are set by use of the Modify Tabs Text Editor command.

2.5.8 BACK TAB FUNCTION. The back tab function is called by pressing the BACK TAB key. On VDTs, the function causes the cursor to be repositioned at the first tab position to the left of the current position within the line. The data in the line remains unchanged. On a hard copy terminal, the current line is reprinted to the tab stop immediately to the left of the current position. If there is no tab stop, a carriage return/line feed occurs. Refer to table 2-1 for the BACK TAB FUNCTION key for the terminal in use.

2.5.9 ROLL UP FUNCTION. The roll up function is called by pressing the appropriate roll up function key of the terminal type in use (refer to table 2-1). On the VDTs, this function causes the display to be repositioned forward the number of lines specified by the roll parameter (this parameter may be changed by the Modify Roll Text Editor command), with the initial roll value being 12 lines on all devices. Note that on a hard copy device, only the record at the current file position (after the roll) is printed.



2.5.10 ROLL DOWN FUNCTION. The roll down function is called by pressing the appropriate roll down function key of the terminal type in use (refer to table 2-1). On the VDTs, this function causes the display to be repositioned backward within the file the number of records specified by the roll parameter, as defined by the Modify Roll Editor command, with the initial value being 12 records on all devices. The records are displayed as newly entered, deleted, or modified. Note that on a hard copy device, only the record at the current file position (after the roll) is printed.

2.5.11 INSERT LINE FUNCTION. The insert line function is called by pressing the appropriate insert line function key of the terminal type in use (refer to table 2-1). When the insert line function is called from a VDT, all lines below and including the line containing the cursor are rolled down one line, and the one containing the cursor is blank filled. This allows for the entry of a new line at the line in which the cursor is resident. Note that if the Text Editor is in the compose mode and the cursor is in the bottom line of the display when the insert line function is called, all data lines are rolled up one line and the line containing the cursor is blank filled.

On a hard copy device, the insert line function allows the operator to enter a line of data that will be inserted into the file at the file position immediately preceding that of the current printed line.

2.5.12 DELETE LINE FUNCTION. The delete line function is called by pressing the appropriate delete line function key of the terminal type in use (refer to table 2-1). This function causes the line that contains the cursor (or the last displayed line on a hard copy device) to be deleted and all lines below it rolled up one line. On the hard copy device, the current line is deleted from the file and the next record is printed.

2.5.13 INSERT CHARACTER FUNCTION. The insert character function is only available on the Model 911 and Model 913 VDTs and is not available on the hard copy devices. The insert character function is called from either VDT by pressing the INSERT CHAR key and keying the new character(s); thereby causing the character at the cursor position and all characters to right of the cursor to be shifted to the right, and the inserted character is written over the cursor. As many characters as desired may be inserted after the INSERT CHAR key has been pressed. Data characters at the right margin of the display are lost as they are shifted to the right. Only data characters may be inserted.

2.5.14 DELETE CHARACTER FUNCTION. The delete character function is only available on the Model 911 and Model 913 VDTs and is not supported on the hard copy devices. The delete character function is called from either VDT by pressing the DELETE CHAR key, which causes the character at the current cursor position to be deleted. In addition, all characters to the right of the deleted character and to the left of the right margin are left shifted one position and the rightmost position is filled with a blank. Care must be used when columns 72 through 80 contain text or information that should also be left shifted. Only characters visible when the delete character function is requested will be left shifted. Any characters not visible will be retained in their current positions. Columns 72 through 80 are visible only when line numbers are suppressed (paragraph 2.5.4).

2.5.15 CURSOR UP FUNCTION. The cursor up function is called by pressing the appropriate cursor up function key of the terminal type in use (refer to table 2-1). This causes the cursor to be moved to the previous line, into the same position as it was in the current line when the cursor up function was called. If the cursor is currently in the top line, all lines are rolled down one line and the cursor remains in the same position.

On a hard copy device, the cursor up function causes the previous line to be printed.



2.5.16 CURSOR DOWN FUNCTION. The cursor down function is called by pressing the appropriate cursor down function key of the terminal type in use (refer to table 2-1). This causes the cursor to be moved to the next line and placed in the same position within the line as it was in the previous line before the cursor down function was called. If the cursor is in the bottom line of the display when the cursor down function is called, all lines are rolled up one line and the cursor remains in the same position.

On a hard copy terminal, the cursor down function causes the next line to be printed.

2.5.17 CURSOR RIGHT FUNCTION. The cursor right function is called by pressing the appropriate cursor right function key of the terminal type in use (refer to table 2-1). This causes the cursor to be moved one position to the right. The data within the line remains unchanged. If the cursor is in the rightmost position of the line, it remains there when the cursor right function is requested.

2.5.18 CURSOR LEFT FUNCTION (BACKSPACE). The cursor left function is called by pressing the appropriate cursor left function key of the terminal type in use (refer to table 2-1). This causes the cursor to be backspaced one position in the line. The data within the line remains unchanged by this function. If the cursor is in the leftmost position, it remains there when the cursor left function is requested.

2.5.19 HOME FUNCTION. The home function is called by pressing the appropriate HOME function key of the terminal type in use (refer to table 2-1). This causes the cursor to be positioned in line one, column one in the display. The displayed data is unchanged by the home function.

2.5.20 ERASE FUNCTION. The erase function is called by pressing the appropriate erase function key of the terminal type in use (refer to table 2-1). All characters in the line are replaced by blanks and the cursor is positioned in the first column of the line.



SECTION 3**ACTIVATING LANGUAGE PROCEDURES****3.1 ASSEMBLY LANGUAGE PROGRAM GENERATION RUNTHROUGH**

This paragraph describes a simple procedure for creating and executing an assembly language program using DX10. Subsequent paragraphs describe similar procedures for creating and executing programs in higher level languages. For more detailed information on how to execute a program using DX10, consult the reference manual for the language in which the program is written.

The brief assembly language program given in this section displays a message and requests the input of three numbers. Since this program is already designed and since it is unlikely to have a long lifetime, step 1 (design) as described in Section 1 may be omitted. The procedure given here assumes that a 911 VDT is used. If another terminal is used, see table 2-1. It is also assumed that a printer (LP01) is used.

1. Enter the program into the computer.
 - a. Power up the computer and terminal and log on using the procedures given in the *Model 990 Computer DX10 Operating System Production Operation* manual.

FILE ACCESS NAME: (Press TAB/SKIP)

- b. Invoke the Text Editor by entering the XE command. Then select the following parameters:

FILE ACCESS NAME: (Press TAB/SKIP)

- c. Press the RETURN key to move the cursor one line above *EOF, and type in the program shown in figure 3-1.
 - d. Press the CMD key to leave the compose mode.
 - e. Enter QE to quit the Text Editor. Select the following parameters:

ABORT?: NO
OUTPUT FILE ACCESS NAME: .SOURCE
REPLACE?: NO
MOD LIST ACCESS NAME: (Press RETURN)

2. Assemble the program.
 - a. Invoke the macro assembler by entering XMA. Select the following parameters:

SOURCE ACCESS NAME: .SOURCE
OBJECT ACCESS NAME: .OBJECT
LISTING ACCESS NAME: LP01
ERROR ACCESS NAME: (Press RETURN)
OPTIONS: (Press RETURN)
MACRO LIBRARY PATHNAME: (Press RETURN)
PRINT WIDTH: 80
PAGE LENGTH: 60



- b. The assembler runs in background mode. Enter a WAIT command to wait for completion of the assembly. When the assembly completes, the following message is displayed:

MACRO ASSEMBLY COMPLETE 0 0 0 0 ERRORS, 0 0 0 0 WARNINGS

Press the RETURN key to return to the command mode.

3. Link edit the object code.

- a. First create a command file for the Link Editor. Invoke the Text Editor by entering XE. Specify the following parameter:

INPUT ACCESS NAME: (Press TAB/SKIP)

- b. Press the RETURN key.
c. Enter the following lines:

```
TASK LANGTST
INCLUDE .OBJECT
END
```

- d. Leave the compose mode by pressing CMD.
e. Quit the Text Editor by entering QE. Select the following parameters:

```
ABORT?: NO
OUTPUT FILE ACCESS NAME: .CNTRLINK
REPLACE?: N
MOD LIST ACCESS NAME: (Press RETURN)
```

- f. Invoke the Link Editor by entering XLE. Select the following parameters:

```
CONTROL ACCESS NAME: .CNTRLINK
LINKED OUTPUT ACCESS NAME: .LNKOUT
LISTING ACCESS NAME: LP01
PRINT WIDTH: 80
```

Press RETURN after the PRINT WIDTH response. When the SCI prompt ([]) appears, enter WAIT and press RETURN. The following display appears:

WAITING FOR BACKGROUND TASK TO COMPLETE

When the Link Editor terminates, the following is displayed:



LINK EDITOR COMPLETED, 0 ERRORS, 0 WARNINGS

Press CMD.

4. Execute the program.
 - a. Enter the IT (Install Task) command to place the program on the system program file. Specify the following parameters:

```
PROGRAM FILE OR LUNO: 0
TASK NAME: LANGTST
TASK ID: 0
OBJECT PATHNAME OR LUNO: .LNKOUT
PRIORITY: 4
DEFAULT TASK FLAGS?: YES
ATTACHED PROCEDURES: NO
```

The installed ID is displayed in the following form when the installation is completed:

```
TASK NAME=LANGTST
TASK ID=id
```

Press CMD to return to the command mode.

- b. Since the program uses LUNO >20, that LUNO must be assigned to the VDT. Call the Assign LUNO (AL) command and respond as follows:

```
LUNO: >20
ACCESS NAME: ME
PROGRAM FILE?: NO
```

The message ASSIGNED LUNO = >20 is then displayed. Press CMD to return to the command mode.

- c. Execute the program using the Execute Task and Suspend SCI (XTS) command. Select the following parameters:

```
PROGRAM FILE OR LUNO: 0
TASK NAME OR ID: LANGTST
PARM1: 0
PARM2: 0
STATION ID: ME
```



```

*****
*          BEGINNING OF DATA SECTION          *
*****
IDT  'RESPONSE'
*****
*          OPENING DATA WORDS                  *
*          1. WORKSPACE POINTER                *
*          2. PC VALUE AT START OF PROGRAM    *
*          3. END ACTION ITEM (IF ANY)        *
*****
DATA WSP          WORKSPACE POINTER ADDRESS
DATA START       PC AT PROGRAM BEGINNING
DATA 0           END ACTION (NONE SPECIFIED)
WSP  BSS 32      WORKSPACE REGISTERS
OPEN DATA 0     I/O REQUEST
          BYTE >0,>20  OPEN LUN0 >20
          DATA 0
          DATA 0
          DATA 0
MSSG0 DATA 0    I/O REQUEST
          BYTE >B,>20  WRITE ASCII ON LUN0 >20
          DATA 0
          DATA GREET  MESSAGE LOCATION
          DATA 0
          DATA MSSG1-GREET MESSAGE LENGTH
*****
*          SPECIFY THE FIRST MESSAGE          *
*****
GREET DATA >0A0D
          TEXT 'HELLO, PLEASE INPUT NUMBER OF ITEMS '
          TEXT 'SOLD TODAY. USE 4-DIGIT NUMBERS.'
          DATA >0A0D
MSSG1 DATA 0    I/O REQUEST
          BYTE >B,>20  WRITE TO LUN0 >20
          BYTE 0,>40  RESPONSE FOLLOWS MESSAGE
          DATA ITEM1
          DATA 0    CHARACTERS SPECIFIED IN INPUT ROUTE
          DATA 10  MESSAGE LENGTH
          DATA STR1  LOCATION OF INPUT PARAMETERS
STR1  DATA STORE  SAVE PARAMETERS IN STORE
          DATA 4    STORE FOUR CHARACTERS
          DATA 0    CHARACTER COUNT AFTER INPUT
STORE BSS 12
ITEM1 DATA >0A0D
          TEXT 'ITEM 1 '
MSSG2 DATA 0    I/O REQUEST
          BYTE >B,>20  WRITE TO LUN0 >20
          BYTE 0,>40  READ AFTER WRITE
          DATA ITEM2  MESSAGE LOCATION
          DATA 0
          DATA 10  MESSAGE LENGTH
          DATA STR2
STR2  DATA STORE+4  2ND ITEM CHARACTERS STORE LOCATION
          DATA 4    STORE FOUR DIGITS
          DATA 0
ITEM2 DATA >0A0D
          TEXT 'ITEM 2 '
MSSG3 DATA 0    I/O REQUEST
          BYTE >B,>20  WRITE TO LUN0 >20

```

Figure 3-1. Example Assembly Language Program (Sheet 1 of 2)



```

        BYTE 0,>40          READ AFTER WRITE
        DATA ITEMS
        DATA 0
        DATA 10
        DATA STR3
STR3    DATA STORE+3      3RD ITEM STORE LOCATION
        DATA 4
        DATA 0
ITEM3   DATA >0A0D
        TEXT 1 ITEM 3
MSG4    DATA 0            I/O REQUEST
        BYTE >B,>20        WRITE TO LUN0 >20
        DATA 0
        DATA GOODBY      MESSAGE LOCATION
        DATA 0
        DATA CLOSE-GOODBY MESSAGE LENGTH
*****
*      FINAL MESSAGE DISPLAYED      *
*****
GOODBY DATA >0A0D
        TEXT 1 THANK YOU. THAT COMPLETES TODAY'S
        TEXT 1 TRANSACTIONS.
        DATA >0A0D
CLOSE   DATA 0            I/O REQUEST
        BYTE 1,>20        CLOSE LUN0 >20
        DATA 0
        DATA 0
        DATA 0
        DATA 0
EOP     BYTE >16,0        TERMINATE TASK
*
START  XOP @OPEN,15       OPEN LUN0 >20
        XOP @MSG0,15      OPENING MESSAGE
        XOP @MSG1,15      INPUT 1
        XOP @MSG2,15      INPUT 2
        XOP @MSG3,15      INPUT 3
        XOP @MSG4,15      EXIT MESSAGE
        XOP @CLOSE,15     CLOSE FILE, UNLOAD/REWIND
        XOP @EOP,15       TERMINATE TASK
        END START

```

Figure 3-1. Example Assembly Language Program (Sheet 2 of 2)

The test program now executes and displays the following on the screen:

```
HELLO, PLEASE INPUT NUMBER OF ITEMS SOLD TODAY. USE 4-DIGIT
NUMBERS.
```

```
ITEM 1
```

Enter a four-digit number. The following is then displayed:

```
ITEM 2
```

Enter a four-digit number. The following is then displayed:

```
ITEM 3
```




Enter a four-digit number. The following is then displayed:

THANK YOU. THAT COMPLETES TODAY'S TRANSACTIONS.

The screen is then blank-filled and the task's runtime ID is displayed.

Press the CMD key to return to the initial SCI menu. Delete the task entry by using the Delete Task (DT) command as follows:

```
DELETE TASK
PROGRAM FILE OR LUNO:    0
TASK NAME OR ID:       LANGTST
```

Use the Delete File (DF) command to delete the following:

```
.SOURCE
.OBJECT
.LNKOUT
.CNTRLINK
```

3.2 FORTRAN PROGRAM GENERATION RUNTHROUGH

The brief FORTRAN program given in this section obtains (from the system) and prints the date in military format. To implement this demonstration one must have the optional FORTRAN compiler and runtime libraries. Since this FORTRAN program is already designed and since it is unlikely to have a long lifetime, step one, design (as defined in Section 1), may be omitted. The following procedure assumes the use of a 911 VDT and a line printer (LP01). If another terminal is used, see table 2-1.

1. Enter the program into the computer.
 - a. Power up the computer and terminal and log on using the procedures given in the *Model 990 Computer DX10 Operating System Production Operation* manual.
 - b. Invoke the Text Editor by entering XE. Select the following parameters:

FILE ACCESS NAME: (Press TAB/SKIP)

- c. Press the RETURN key to move the cursor one line above *EOF, and type in the following FORTRAN program:

```
INTEGER*2 DATE(4)
REWIND 17
REWIND 18
CALL MDATE (DATE)
WRITE (17, 100) DATE
100  FORMAT (1X, 'TODAYS DATE IN MILITARY FORMAT IS ', 4A2)
WRITE (17, 130)
130  FORMAT (1X, 'FORTRAN TEST COMPLETE')
READ (18, 140) IDUMY
140  FORMAT (A1)
STOP
END
```



- d. Press the CMD key to leave the compose mode.
- e. Enter QE to quit the Text Editor. Select the following parameters:

```
                ABORT?:      NO
OUTPUT FILE ACCESS NAME:  .TSTFORT
                REPLACE:    NO
MOD LIST ACCESS NAME:    (Press RETURN)
```

2. Compile the program:

- a. Invoke the FORTRAN compiler by entering XFC. Select the following parameters:

```
SOURCE ACCESS NAME:  .TSTFORT
OBJECT ACCESS NAME:  .TSTOBJ
LISTING ACCESS NAME:  LP01
                OPTIONS:  (Press RETURN)
PRINT WIDTH:        80 (Press RETURN)
```

Enter WAIT and press the RETURN key. The following message appears:

```
WAITING FOR BACKGROUND TASK TO COMPLETE
```

When the FORTRAN compile completes, the message

```
FORTRAN COMPILER NORMAL COMPLETION
```

is displayed. Press CMD; the message

```
FORTRAN COMPILER COMPLETED  0 ERRORS,  0 WARNINGS:
```

is displayed. Press CMD to return to the SCI command mode.

3. Link Edit the object code.
 - a. First, create a control file for the Link Editor. Invoke the Text Editor by entering XE. Specify the following parameters:

```
FILE ACCESS NAME:  (Press TAB/SKIP)
```

- b. Press the RETURN key.



- c. Enter the following lines (assuming the FORTRAN libraries are on the system disk):

```
LIBRARY .FORTRN.OSLOBJ
LIBRARY .FORTRN.STLOBJ
TASK ROOT
INCLUDE .TSTOBJ
END
```

- d. Enter the command mode by entering CMD.
- e. Quit the Text Editor by entering QE. Select the following parameters:

```
ABORT?: NO
OUTPUT FILE ACCESS NAME: .CNTRLINK
REPLACE?: N
MOD LIST ACCESS NAME: (Press RETURN)
```

- f. Invoke the Link Editor by entering XLE. Select the following parameters:

```
CONTROL ACCESS NAME: .CNTRLINK
LINKED OUTPUT ACCESS NAME: .TSTFORTO
LISTING ACCESS NAME: .LSTFORTO
PRINT WIDTH: 80 (Press RETURN)
```

Enter WAIT and press RETURN. The message

```
WAITING FOR BACKGROUND TASK TO COMPLETE
```

is displayed.

When the Link Editor completes, the message

```
LINK EDITOR COMPLETED, 0 ERRORS, 0 WARNINGS
```

is displayed.

Press CMD.



4. Install and execute the program
- a. Enter the IT (Install Task) command to place the program on the system program file. Specify the following parameters:

```

PROGRAM FILE OR LUNO:    0
TASK NAME:              ROOT
TASK ID:                0
OBJECT PATHNAME OR LUNO: .TSTFORTO
PRIORITY:               4
DEFAULT TASK FLAGS?:   YES
ATTACHED PROCEDURES?:  NO

```

The message

```

TASK NAME    =    ROOT
TASK ID     =    id

```

is then displayed. Press CMD.

- b. Use the Assign Synonym (AS) command to define FORTRAN I/O units 17 and 18 to the system. Specify the following parameters:

```

SYNONYM:    UNIT17    SYNONYM:    UNIT18
VALUE:      ME        VALUE:      ME

```

- c. Execute the program by using the XFTF command. Select the following parameters:

```

PROGRAM FILE LUNO:    0
TASK ID:              <id>

```

The test program now executes and displays the date in military format. Press RETURN and the message "STOP 0 NORMAL PROGRAM COMPLETION" is displayed. After the program finishes, delete the test program files to preserve disk space. Use the DF (Delete File) command to delete:

```

.CNTRLINK
.TSTFORTO
.TSTFORT
.TSTOBJ
.LSTFORTO

```

3.3 COBOL PROGRAM GENERATION RUNTHROUGH

The COBOL program shown in figure 3-2 is used in the following demonstration of COBOL program generation techniques in DX10. To implement this demonstration, one must have the optional COBOL language processors. The program presents the operator with a choice of cities. When the operator selects a city, the program displays the current time of day in that city. Since the



program is already designed and documented, steps 1 and 6, respectively, may be omitted. The run-through assumes the use of a 911 VDT and a line printer (LP01). If another terminal is used, see table 2-1.

1. Enter the program into the computer.

a. Power up the computer and terminal and log on using the procedures given in the *Model 990 Computer DX10 Operating System Production Operation* manual.

b. Invoke the Text Editor by entering XE. Specify the following parameters:

FILE ACCESS NAME: (Press TAB/SKIP)

c. Press the RETURN key to position the cursor above the displayed *EOF.

d. Press CMD to enter command mode. Enter MRM and press RETURN to modify the right margin. Respond as follows:

MODIFY RIGHT MARGIN
RIGHT MARGIN POSITION: 72

Press RETURN or NEW LINE.

e. Type the program as shown in figure 3-2. Do not press the RETURN key after keying the last line since this would cause a blank line to be inserted at the end of the source.

f. Press the CMD key to enter the command mode.

g. Terminate the Text Editor by entering the QE command.

1) Enter QE.

2) Specify the following parameters:

ABORT?: N
OUTPUT FILE ACCESS NAME: .TSTCBL
REPLACE?: N
MOD LIST ACCESS NAME: (Press RETURN)



Program Instructions
begin in column 8:

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. TIME-AROUND-THE-WORLD.  
***** THIS PROGRAM CALCULATES TIMES BY OBTAINING THE  
** LOCAL TIME FROM THE SYSTEM CLOCK AND ADDING A TIME  
** WHICH IS THE CHOSEN CITY'S TIME RELATIVE TO CENTRAL  
** STANDARD TIME. THEREFORE, THE VALUES IN THE  
** TIME TABLE MUST BE ADJUSTED IF THE PROGRAM IS TO BE  
***** RUN IN A TIME ZONE OTHER THAN CST.  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER. TI-990.  
OBJECT-COMPUTER. TI-990.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 TIME-VALUES.  
03 FILLER PIC S99 VALUE +1.  
03 FILLER PIC S99 VALUE -1.  
03 FILLER PIC S99 VALUE +8.  
03 FILLER PIC S99 VALUE +12.  
03 FILLER PIC S99 VALUE +6.  
03 FILLER PIC S99 VALUE +14.  
03 FILLER PIC S99 VALUE +16.  
03 FILLER PIC S99 VALUE +7.  
03 FILLER PIC S99 VALUE +15.  
03 FILLER PIC S99 VALUE -4.  
03 FILLER PIC S99 VALUE +3.  
03 FILLER PIC S99 VALUE 0.  
03 FILLER PIC S99 VALUE +13.  
03 FILLER PIC S99 VALUE +1.  
03 FILLER PIC S99 VALUE +9.  
03 FILLER PIC S99 VALUE +13.  
01 TIME-TABLE REDEFINES TIME-VALUES.  
03 DTIME OCCURS 16 TIMES PIC S99.  
01 CLOCK.  
03 HOUR PIC 99.  
03 MIN PIC 99.  
03 SEC PIC 99.  
03 FILLER PIC 99.  
77 HOUR-TEMP PIC S99.  
77 BLANK-LINE PIC X(80) VALUE SPACES.  
77 ANSWER PIC X.  
77 CHOICE PIC 99.  
01 CLOCK-EDITED.  
03 HOUR-EDITED PIC XX.  
03 FILLER PIC X VALUE ":".  
03 MIN-EDITED PIC XX.  
03 FILLER PIC X VALUE ":".  
03 SEC-EDITED PIC XX.  
/  
PROCEDURE DIVISION.  
GIVE-INSTRUCTIONS.  
DISPLAY "COBOL TIME-AROUND-THE-WORLD-PROGRAM" LINE 1 ERASE.  
DISPLAY "KEY IN THE NUMBER OF THE CITY WHOSE TIME YOU WANT."  
LINE 2.  
DISPLAY "TO KNOW:" POSITION 0.  
DISPLAY " 1 NEW YORK " LINE 3.  
DISPLAY " 9 TOKYO " POSITION 0.  
DISPLAY " 2 DENVER " LINE 4.  
DISPLAY " 10 HONOLULU " POSITION 0.  
DISPLAY " 3 MOSCOW " LINE 5.
```

Figure 3-2. Example COBOL Program (Sheet 1 of 2)



```
DISPLAY " 11 BUENOS AIRES           " POSITION 0.
DISPLAY " 4  OACCA                   " LINE 6.
DISPLAY " 12 OAXACA                   " POSITION 0.
DISPLAY " 5  ACCRA                    " LINE 7.
DISPLAY " 13 BANGKOK                  " POSITION 0.
DISPLAY " 6  PEKING                   " LINE 8.
DISPLAY " 14 MONTREAL                 " POSITION 0.
DISPLAY " 7  SYDNEY                   " LINE 9.
DISPLAY " 15 BAGHDAD                 " POSITION 0.
DISPLAY " 8  COPENHAGEN               " LINE 10.
DISPLAY " 16 JAKARTA                 " POSITION 0.
ACCEPT-CHOICE.
ACCEPT CHOICE LINE 2 POSITION 65 PROMPT CONVERT.
IF CHOICE > 16 OR CHOICE = 0 GO ACCEPT-CHOICE.
ACCEPT CLOCK FROM TIME.
MOVE HOUR TO HOUR-TEMP.
ADD DTIME (CHOICE) TO HOUR-TEMP.
IF HOUR-TEMP > 24 SUBTRACT 24 FROM HOUR-TEMP.
IF HOUR-TEMP < 0 ADD 24 TO HOUR-TEMP.
MOVE HOUR-TEMP TO HOUR.
MOVE HOUR TO HOUR-EDITED.
MOVE MIN TO MIN-EDITED.
MOVE SEC TO SEC-EDITED.
DISPLAY "TIME : " LINE 11 POSITION 15.
DISPLAY CLOCK-EDITED POSITION 0.
DISPLAY "WOULD YOU LIKE TO KNOW ANOTHER CITY'S TIME? (Y/N)"
      LINE 12.
ACCEPT ANSWER POSITION 0 PROMPT.
IF ANSWER = "N" GO END-PROGRAM.
DISPLAY BLANK-LINE LINE 11.
DISPLAY BLANK-LINE LINE 12.
GO GIVE-INSTRUCTIONS.
/
END-PROGRAM.
STOP RUN.
END PROGRAM.
```

Figure 3-2. Example COBOL Program (Sheet 2 of 2)

2. Invoke the COBOL compiler by entering the XCC command.
 - a. Enter XCC.
 - b. Specify the following parameters:

```
SOURCE ACCESS NAME: .TSTCBL
OBJECT ACCESS NAME: .CBLOBJ
LISTING ACCESS NAME: .CBLST
      OPTIONS: (Press RETURN)
      PRINT WIDTH: 80 (Press RETURN)
      PAGE SIZE: 55
PROGRAM SIZE (LINES): 1000
```



- c. Enter WAIT and press RETURN. The following is displayed:

WAITING FOR BACKGROUND TASK TO COMPLETE

When the compiler completes, the following is displayed:

COBOL COMPILER COMPLETED, 0 ERRORS, 0 WARNINGS:

- d. Press CMD to return to the command mode.*
- e. Execute COBOL Program in Foreground by entering the XCPF Command. Specify the following parameter:

```
OBJECT ACCESS NAME: .CBLOBJ
DEBUG MODE: NO
MESSAGE ACCESS NAME: (Press RETURN)
SWITCHES: 00000000
FUNCTION KEYS: NO
```

The test program now executes. When the program prompts, select one of the listed cities. The program then displays the current time of day in that city. Enter N to halt program execution.

To preserve disk space, use the Delete File (DF) command to delete the files .TSTCBL, .CBLOBJ, and .CBLIST.

3.4 BASIC PROGRAM GENERATION RUNTHROUGH

The program given below illustrates the program development cycle for BASIC programs. To implement this demonstration, one must have the optional BASIC language processor. Since BASIC is an interactive programming language, steps 2, 3, and 4 are combined. Since the program is already designed and documented, steps 1 and 5 (design and documentation) may be eliminated.

The program given below accepts a number from the operator, calculates the factorial and displays the result. (For example, 6 factorial = $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$.)

1. Enter the program into the computer.
 - a. Power up the computer and terminal and log on using the procedures given in the *Model 990 Computer DX10 Operating System Production Operation* manual.
 - b. Activate BASIC by entering the command BASIC following the SCI prompt ([]).
 - c. When the BASIC screen is displayed, enter the amount of core required (WORKSPACE SIZE (KB)) prompt and press the TAB key to enter the default value on the INITIAL PROGRAM NAME prompt.
 - d. When BASIC responds with its prompt (.), enter the following program:

```
10 PRINT "FACTORIAL PROGRAM", TIMES$, DAT$
15 PRINT "ENTER THE NUMBER WHOSE FACTORIAL YOU WANT:",
```




```
20 INPUT N
30 IF N<=0 THEN 140
40 IF N<=70 THEN 70
50 PRINT "THE MAXIMUM FACTORIAL FOR THIS ROUTINE IS 69!"
60 GOTO 20
70 F=1
80 FOR I=1 TO N
90     F=F*I
100    NEXT I
110 PRINT N,F
120 PRINT
130 GOTO 20
140 END
```

*Steps 3 and 4, linking and installing the program, are combined for COBOL.

2. The program is now in the computer. To execute the program, enter:

RUN

- a. The program responds by requesting an input number. Enter a number and the program displays the factorial.
- b. To exit the program, enter zero (0) in response to the (?) prompt.
- c. To quit the BASIC interpreter, enter BYE. BASIC programs may be stored on disk so that they need not be reentered every time.

For more information about BASIC program storage, consult the *TI990 BASIC Reference Manual*.

3.5 RPG SOURCE PROGRAM ENTRY

There are two means of entering RPG source programs from the keyboard. The DX10 Text Editor (XE) allows for entry of a program or the modification of an existing program. The RPG editor (RPGEDIT) guides the user through each statement type and ensures that fields start in the proper columns. Once the program has been entered, the following commands are used to compile, bind (link), and execute the program.

3.5.1 COMPILATION. Enter either the XRP GC or XRP GC F command to compile a source program. The following values are entered at the appropriate prompts:

```
EXECUTE RPGII COMPILER IN FOREGROUND
SOURCE ACCESS NAME: .EXAMPLE.SOURCE.TEST17
OBJECT ACCESS NAME: .EXAMPLE.OBJECT.TEST17
LISTING ACCESS NAME: .EXAMPLE.LIST.TEST17
MESSAGE ACCESS NAME: .EXAMPLE.MSG
PRINT WIDTH: 120
```

Figure 3-3 shows the listing file created by the compiler.



```

DIRPG      2.0.0  79.180  04/20/79  11:54:12          PAGE  1
          0  1  1  2  2  3  3  4  4  5  5  6  6  7  7
          ...5...0...5...0...5...0...5...0...5...0...5...0...5...0...5...
0001      H                                          TEST17
0002      FINREL  IP  F    80          DISK          TEST17
0003      FRELREC UC  F   80R         DISK          TEST17
0004      FOUTREL O  F    80          DISK          TEST17
0005      IINREL  AA  01
0006      I                                          1  20NEWID    TEST17
0007      I                                          6  30 NAME     TEST17
0008      IRELREC NS  04
0009      I                                          1  20ID       TEST17
0010      I                                          6  30 FIELD    98 TEST17
0011      C          NEWID  CHAINRELREC          99    TEST17
0012      C  98          EXCPT
0013      C          END    TAG
0014      ORELREC  E    01 98
0015      O          NEWID   2
0016      O          NAME   30
0017      OOUTREL  D    01
0018      O          NEWID   3
0019      O          NAME   35
0020      O          98      47 'ADDED'
0021      O          04M98   51 'DUPLICATE'
0022      O          99      45 'EOF'
0023      /*

```

THE FOLLOWING INDICATORS APPEARED IN THIS PROGRAM

01 04 98 99

SORTED LABEL NAMES

```

NAME      LINE
END       13

```

SORTED FIELD NAMES

```

NAME      TYPE  LENGTH  DP
FIELD     A     25
ID        N     2      0
NAME      A     25
NEWID     N     2      0
UPDATE    N     6      0
UDAY      N     2      0
UMONTH    N     2      0
UYEAR     N     2      0

```

Figure 3-3. Compiler Output Listing



3.5.2 BINDING. The XRPGB command is used to bind the RPG II program. The following values are entered at the appropriate prompts:

```
BIND RPG II PROGRAM
  OBJECT ACCESS NAME: .EXAMPLE.OBJECT.TEST17
  LISTING ACCESS NAME: .EXAMPLE.LNKLST.TEST17
  PROGRAM FILE NAME: .EXAMPLE.PROG
  NAME OF TASK: TEST17
  REPLACE TASK: YES
```

3.5.3 EXECUTION. Execution of an RPG II program requires three steps:

1. Synonym assignment to filenames
2. LUNO assignment to the program file
3. Execution of the program.

Filename synonyms must be assigned via the Assign Synonym (AS) command as follows:

```
ASSIGN SYNONYM VALUE
  SYNONYM: INREL
  VALUE: .EXAMPLE.DATA.INREL

ASSIGN SYNONYM VALUE
  SYNONYM: RELREC
  VALUE: .EXAMPLE.DATA.RELREC

ASSIGN SYNONYM VALUE
  SYNONYM: OUTREL
  VALUE: .EXAMPLE.DATA.OUTREL
```

Assignment of a LUNO to the program file is accomplished via the Assign LUNO (AL) command as follows:

```
ASSIGN LUNO
  LUNO: OBB
  ACCESS NAME: .EXAMPLE.PROG
  PROGRAM FILE?: YES
```

The RPGII program is now ready for execution. Because a station-local LUNO was assigned, the XRPGETF command, which executes in foreground, must be used. The following values are entered:

```
EXECUTE RPG II PROGRAM IN FOREGROUND
  TASK NAME/ID: TEST17
  PROGRAM FILE LUNO: OBB
  INDICATORS: NONE
  TERMINAL CONTROL: NO
  TRACE OUTPUT ACCESS NAME:
  MESSAGE ACCESS NAME: .EXAMPLE.MSG
```

Figure 3-4 displays the contents of the input file. The output from the execution of the program is shown in figure 3-5.



```
05 JOHN Q. PUBLIC
03 SEMORE PEOPLE
07 ROBERT A. JONES
08 BARBARA C. GOOD
04 RACHEL R. SOMETHING
09 HOPE R. RIGHT
```

Figure 3-4. Input File

```
05 JOHN Q. PUBLIC      ADDED
03 SEMORE PEOPLE      ADDED
07 ROBERT A. JONES    ADDED
08 BARBARA C. GOOD    ADDED
04 RACHEL R. SOMETHING ADDED
09 HOPE R. RIGHT      ADDED
```

Figure 3-5. Output Listing

3.6 PASCAL PROGRAM GENERATION RUNTHROUGH

The Pascal program shown in figure 3-6 is used in the following demonstration of Pascal program generation techniques in DX10. This program finds the factor by which the dollar is devalued after a certain number of years. Since the program is already designed and since it is unlikely to have a long lifetime, step 1 (design) as described in Section 1 may be omitted. The procedure given here assumes that a 911 VDT is used. Refer to table 2-1 for the function keys of the other terminal types supported by DX10. This procedure also assumes that a printer (LP01) is used.

1. Enter the program into the computer.
 - a. Power up the computer and terminal and log on using the procedures in the *Model 990 Computer DX10 Operating System Production Operation* manual.
 - b. Create the directory TEST by entering the CFDIR command. Specify the following parameters:

```
CREATE DIRECTORY FILE
                PATHNAME: .TEST
                MAX ENTRIES: 10
DEFAULT PHYSICAL RECORD SIZE: (Press RETURN)
```

- c. Invoke the Text Editor by entering the XE command. Specify the following parameters:

```
FILE ACCESS NAME: (Press TAB)
```
 - d. Press the RETURN key.
 - e. Enter the program shown in figure 3-6.
 - f. Press the CMD key to leave the compose mode.



- g. Enter QE to quit the Text Editor. Select the following parameters:

```
                ABORT?:  NO
OUTPUT FILE ACCESS NAME:  .TEST.PASSRC
                REPLACE?:  NO
MOD LIST ACCESS NAME:  (Press RETURN)
```

2. Compile the program.

- a. Invoke the Pascal compiler by entering the XTIP command. Specify the following parameters:

```
SOURCE:  .TEST.PASSRC
OBJECT:  .TEST.PASOBJ
LISTING:  .TEST.PASLST
MESSAGES:  (Press RETURN)
OPTIONS:  (Press RETURN)
MEM1:  (Press RETURN)
MEM2:  (Press RETURN)
MEM3:  (Press RETURN)
```

- b. Enter WAIT and press the RETURN key. The following is displayed:

```
— WAITING FOR BACKGROUND TASK TO COMPLETE —
```

When the compiler completes, the following is displayed:

```
TIP COMPILATION COMPLETE:
```

- c. Press the CMD key to return to the command mode.

3. Link edit the object code.

- a. Create a link control file for the Link Editor. Invoke the Text Editor by entering XE. Specify the following parameters:

```
FILE ACCESS NAME:  (Press TAB/SKIP)
```

- b. Press the RETURN key.

- c. Enter the following link control file:

```
FORMAT IMAGE,REPLACE
LIBRARY .TIP.OBJ
TASK INFLATE
INCLUDE (MAIN)
INCLUDE .TEST.PASOBJ
END
```

- d. Enter the command mode by pressing the CMD key.



- e. Quit the Text Editor by entering QE. Specify the following parameters:

```
ABORT?: NO
OUTPUT FILE ACCESS NAME: .TEST.PASLINK
REPLACE?: NO
MOD LIST ACCESS NAME: (Press RETURN)
```

- f. Invoke the Link Editor by entering XLE. Specify the following parameters:

```
CONTROL ACCESS NAME: .TEST.PASLINK
LINKED OUTPUT ACCESS NAME: .TEST.PASPROG
LISTING ACCESS NAME: .TEST.LINKLIST
PRINT WIDTH: 80 (Press RETURN)
```

- g. Enter WAIT and press the RETURN key. The following message is displayed:

```
— WAITING FOR BACKGROUND TASK TO COMPLETE —
```

- h. When the Link Editor completes, the following message is displayed:

```
LINK EDITOR COMPLETED, 0 ERRORS, 0 WARNINGS
```

- i. Press the CMD key.

4. Execute the Pascal task by entering the XPT command. Specify the following parameters:

```
PROGRAM FILE: .TEST.PASPROG
TASK NAME OR ID: INFLATE
INPUT:
OUTPUT: .TEST.PASOUT
MESSAGES:
MODE (F,B,D): FOREGROUND
MEMORY:
```

5. Show the file created (.TEST.PASOUT) with the Show File (SF) command. Figure 3-7 illustrates the results of execution of the program.

The test program now executes displaying the message INFLATE EXECUTION BEGINS.

To preserve disk space, use the Delete Directory (DD) command to delete the files in the directory .TEST and the directory .TEST.

**Program INFLATION:**

(* Find the factor by which the dollar is devalued after N years for N = 1, 2, ..., 10. Use annual inflation rates of 7, 8, and 10 percent*)

Const N = 10;

Var YEAR : Integer;

R1, R2, R3 : Real;

Begin

Writeln(' YEAR 7% 8% 10%');

R1 := 1.0; R2 := 1.0; R3 = 1.0;

For YEAR := 1 To N Do

Begin

R1 := R1*1.07;

R2 := R2*1.08;

R3 := R3*1.10;

Writeln(YEAR, ' ',R1:6:3, R2:6:3, R3:6:3)

End

End.

Figure 3-6. Example Pascal Program Source

YEAR	7%	8%	10%
1	1.070	1.080	1.100
2	1.145	1.166	1.210
3	1.225	1.260	1.331
4	1.311	1.360	1.464
5	1.403	1.469	1.611
6	1.501	1.587	1.772
7	1.606	1.714	1.949
8	1.718	1.851	2.144
9	1.838	1.999	2.358
10	1.967	2.159	2.594

Figure 3-7. Example Pascal Program Output



SECTION 4

LINK EDITOR USE ON DX10

4.1 SUPPORTED FEATURES

The Link Editor is used to link separate object modules together to form a single program that runs under DX10.

The disk-based operating system for the Models 990/10 and 990/12 computers, DX10, is a multitasking operating system that supports all of the features of the Link Editor. As a disk-resident system, DX10 is well-suited for overlay structured programs. These programs are supported by the automatic overlay loading feature described in the *Link Editor Reference Manual*.

The following Link Editor features are supported by DX10:

- Automatic overlay loading
- Random libraries
- Sequential libraries
- COBOL program linking
- FORTRAN program linking
- Pascal program linking
- ASCII, compressed and image format
- Absolute memory partitioning

For more information about these features, consult the *Link Editor Reference Manual*.

4.2 LINK EDITOR OPERATION WITH DX10

The first step in performing a Link Edit run is to develop a control file that defines the Link Edit functions. The control file can be developed using the DX10 Text Editor, or it can be developed as a card, tape, or cassette file. The control file contains Link Edit commands as well as the names of any object modules.

The Link Editor is executed at a station by entering the XLE command. Note that the station must be in the command mode prior to entering the command (refer to table 2-1 for the appropriate command key of the terminal type in use). When XLE is entered, the following display is presented at a VDT (on a hard copy device, the prompts are printed one at a time).

```
EXECUTE LINKAGE EDITOR
CONTROL ACCESS NAME:
LINKED OUTPUT ACCESS NAME:
LISTING ACCESS NAME:
PRINT WIDTH: 80
```




In response to the CONTROL ACCESS NAME: prompt, the user must enter the pathname of the device or file from which the control stream is to be read. The control file can be on a sequential disk file, or any sequential device such as a tape unit, cassette unit, or cards. The following is an example of the pathname entry for a sequential disk file:

```
CONTROL ACCESS NAME:  VOL2.EDITOR.CONFILE
```

There is no default for the CONTROL ACCESS NAME:. Tabbing out of the field is not possible.

In response to the LINKED OUTPUT ACCESS NAME: prompt, the user enters the access name of the sequential device or file to which the output of the Link Editor is to be written. If the object output is not desired, the user may specify DUMMY, which suppresses the generation of the output. Use of the DUMMY value allows for a trial run to ensure that no errors occur. The following is an example of an access name entry for a sequential disk file:

```
LINKED OUTPUT ACCESS NAME:  VOL2.LINK.OUT1
```

If the FORMAT command specifies the IMAGE option, the entry made in response to the LINKED OUTPUT ACCESS NAME prompt must be a DX10 program file or a DX10 system image file.

In response to the LISTING ACCESS NAME: prompt, the user enters the access name of the device or file to which the load map listing is to be written. If the listing output is not desired, the user may specify DUMMY, which will suppress the generation of the output. The value entered in response to the prompt can be any valid DX10 access name, synonym, or device name. The following example causes the listing to be written to a line printer.

```
LISTING ACCESS NAME:  LP01
```

For a description of the load map listing, refer to the *Link Editor Reference Manual*.

The last prompt, PRINT WIDTH:, allows the user to either specify the width of the print line, or to accept the default value — 80 characters.

The following example shows the responses for the prompt when the control file is on VOL1.EDITOR.CONFILE, the listing device is line printer one (LP01), and the default PRINT WIDTH value is accepted:

```
XLE
EXECUTE LINKAGE EDITOR
CONTROL ACCESS NAME:  VOL1.EDITOR.CONFILE
LINKED OUTPUT ACCESS NAME:  VOL1.LINK.OUT1
LISTING ACCESS NAME:  LP01
PRINT WIDTH:  80
```



SECTION 5

INSTALLING, DELETING, AND MODIFYING PROGRAMS

5.1 INTRODUCTION

Under DX10, programs are called tasks. A task may be segmented to include sharable procedures and may also include overlays. After Link Edit, and before program execution, the task and its procedures and overlays must be installed on a program file (unless this step is bypassed by use of the IMAGE format option of the Link Editor). For further information about task segmentation, refer to the *Model 990 Computer DX10 Operating System Application Programmer's Guide*.

All of the install commands in this section allow the program file and the object file to be specified by file name or by LUNO. The manner in which the program file is selected is arbitrary. There is an important difference between selecting the object file by LUNO and selecting the object file by pathname: files specified by pathname are rewound when opened, but files specified by LUNO are *not* rewound when opened. Thus, if the same object file contains procedures, tasks, and overlays, it *must* be specified by LUNO for the commands to install correctly all of the object in a program.

Tasks, procedures, and overlays must be installed in the following order:

1. Procedures, if any, must be installed first.
2. The task is installed after the procedures.
3. Overlays are installed last.

Thus, object files containing more than one object (task, procedure, overlay) must be ordered with the procedures first, task second, and overlays last.

The following paragraphs discuss the commands that install, delete, and modify programs.

Installing or modifying a task or procedure to be memory resident requires that the system be re-booted before the task or procedure is usable.

5.2 IT – INSTALL TASK

The Install Task command places an executable task on a program file. If the task has attached procedures, the procedures must be installed before the IT command. For an explanation of the task attributes (priority, privileged, system, memory-resident, and replicative), consult Section 2 of the *Model 990 Computer DX10 Operating System Application Programmer's Guide*.

NOTE

The user should not install a task on the \$\$\$SDS\$ program file. If the \$\$\$SDS\$ program file is used, the user's IDs and names will be destroyed upon each new release of DX10. It is recommended that the user install tasks in his own library. This recommendation also applies to installing real-time tasks, procedures, and overlays.



Syntax:

```

IT
INSTALL TASK          { < acnm > }
                     { < int > }

TASK NAME:           <string>

TASK ID:              { < int > }
                     { < 0 > }

OBJECT PATHNAME OR LUNO: { < acnm > }
                       { < int > }

PRIORITY:             { < int > }
                       { 4 }

DEFAULT TASK FLAGS?: { YES }
                     { <NO> }

ATTACHED PROCEDURES?: { <YES> }
                       { NO }

PRIVILEGED?:          { <YES> }
                       { NO }

SYSTEM TASK?:         { <YES> }
                       { NO }

MEMORY RESIDENT?:    { <YES> }
                       { NO }

REPLICATABLE?:       { YES }
                       { <NO> }

DELETE PROTECTED?:   { <YES> }
                       { NO }

EXECUTE PROTECTED?:  { <YES> }
                       { NO }

OVERFLOW CHECKING?: { <YES> }
                       { NO }

WRITABLE CONTROL STORAGE?: { <YES> }
                             { NO }

ATTACH TASK PROCEDURES

1ST PROCEDURE ID:    { < int > }
                     { 0 }

P1 FROM TASKS PROGRAM FILE?: { YES }
                              { <NO> }

2ND PROCEDURE ID:    { < int > }
                     { 0 }

P2 FROM TASKS PROGRAM FILE?: { YES }
                              { <NO> }

```

These questions are asked only if the answer to DEFAULT TASK FLAGS?: is NO.

These questions are asked only if the answer to ATTACHED PROCEDURES?: is YES.



The TASK NAME: and TASK ID: parameters need not be entered. If they are not, the system assigns values. These parameters cannot be the same as an existing task. The P1 FROM TASKS PROGRAM FILE?: and P2 FROM TASKS PROGRAM FILE?: prompts ask whether the attached procedures reside in the same program file as the task.

5.2.1 IRT — INSTALL REAL-TIME TASK. The Install Real-Time Task command places an executable real-time task on a program file. If the task has attached procedures, the procedures must be installed before the IRT command. For an explanation of the task attributes (priority, privileged, system, memory-resident, and replicative), consult the *Model 990 Computer DX10 Operating System Application Programmer's Guide*. Before installing a real-time task, refer to the NOTE in the paragraph for the Install Task command.

Syntax:

IRT
INSTALL REAL-TIME TASK

PROGRAM FILE OR LUNO:	{ <acnm> } { <int> }	
TASK NAME:	{ <string> }	
TASK ID:	{ <int> } { <u>0</u> }	
OBJECT PATHNAME OR LUNO:	{ <acnm> } { <int> }	
PRIORITY:	{ <int> } { 1 - 127 }	
DEFAULT TASK FLAGS?:	{ <u>YES</u> } { <NO> }	
ATTACHED PROCEDURES?:	{ <YES> } { <u>NO</u> }	
PRIVILEGED?:	{ <YES> } { <u>NO</u> }	} These questions are asked only if the answer to DEFAULT TASK FLAGS?: is NO.
SYSTEM TASK?:	{ <YES> } { <u>NO</u> }	
MEMORY RESIDENT?:	{ <YES> } { <u>NO</u> }	
REPLICATABLE?:	{ <u>YES</u> } { <NO> }	
DELETE PROTECTED?:	{ <YES> } { <u>NO</u> }	
EXECUTE PROTECTED?:	{ <YES> } { <u>NO</u> }	
OVERFLOW CHECKING?:	{ <YES> } { <u>NO</u> }	
WRITABLE CONTROL STORAGE?:	{ <YES> } { <u>NO</u> }	



ATTACH TASK PROCEDURES

1ST PROCEDURE ID:	{ <int> }	} These questions are asked only if the answer to ATTACHED PROCEDURES is YES.
P1 FROM TASKS PROGRAM FILE?	{ <u>YES</u> } { <NO> }	
2nd PROCEDURE ID:	{ <int> }	
P2 FROM TASKS PROGRAM FILE?	{ <u>YES</u> } { <NO> }	

The TASK NAME: and TASK ID: parameters need not be entered. If they are not, the system assigns values. These parameters cannot be the same as an existing task. The P1 FROM TASKS PROGRAM FILE?: and P2 FROM TASKS PROGRAM FILE?: prompts ask whether the attached procedures reside in the same program file as the task.

5.3 IP – INSTALL PROCEDURE

The IP command places a procedure on a program file and assigns a procedure ID for use by subsequent IT calls. Before installing a procedure, refer to the NOTE in the paragraph for the Install Task command.

Syntax:

IP
INSTALL PROCEDURE

PROGRAM FILE OR LUNO:	{ <acnm> }
	{ <int> }
PROCEDURE NAME:	<string>
PROCEDURE ID:	{ <int> }
	{ <u>0</u> }
OBJECT PATHNAME OR LUNO:	{ <acnm> }
	{ <int> }
MEMORY RESIDENT?:	{ <YES> }
	{ <u>NO</u> }
DELETE PROTECT?:	{ <YES> }
	{ <u>NO</u> }
EXECUTE PROTECT?:	{ <YES> }
	{ <u>NO</u> }
WRITE PROTECT?:	{ <YES> }
	{ <u>NO</u> }
WRITABLE CONTROL STORAGE?:	{ <YES> }
	{ <u>NO</u> }

If the PROCEDURE NAME: and PROCEDURE ID: prompts are not specified, the values are assigned by the system. If specified, they cannot be equal to existing names or IDs.

5.4 IO – INSTALL OVERLAY

The IO command places an overlay associated with a task on the program file with the task. The task must be installed before the overlay and may be specified by name or by installed ID. Before installing an overlay, refer to the NOTE in the paragraph for the Install Task command.



Syntax:

IO
INSTALL OVERLAY

PROGRAM FILE OR LUNO:	{<acnm>}
	{<int> }
OVERLAY NAME:	<string>
OVERLAY ID:	<int>
OBJECT PATHNAME OR LUNO:	{<acnm>}
	{<int> }
RELOCATABLE?:	{<YES>}
	{ <u>NO</u> }
DELETE PROTECT?:	{<YES>}
	{ <u>NO</u> }
ASSOCIATED TASK NAME OR ID:	{<string>}
	{<int> }

If the OVERLAY NAME: and OVERLAY ID: values are not specified, they are assigned by the system. If specified, they cannot duplicate existing names or IDs. The ID must be less than 255 and greater than 0.

5.5 DT – DELETE TASK

This command removes a previously installed task from a program file. The task can be specified either by name or by installed ID. If associated overlays exist, they are also deleted.

Syntax:

DT
DELETE TASK

PROGRAM FILE OR LUNO:	{<acnm>}
	{<int> }
TASK NAME OR ID:	{<string>}
	{<int> }

5.6 DP – DELETE PROCEDURE

This command removes a previously installed procedure from a program file. The procedure may be specified by name or by installed ID.

Syntax:

DP
DELETE PROCEDURE

PROGRAM FILE OR LUNO:	{<acnm>}
	{<int> }
PROCEDURE NAME OR ID:	{<string>}
	{<int> }

5.7 DO – DELETE OVERLAY

This command removes a previously installed overlay from a program file. The overlay may be specified by name or by installed ID.



Syntax:

DO
DELETE OVERLAY

PROGRAM FILE OR LUNO: {<acnm>}
 {<int>}
OVERLAY NAME OR ID: {<string>}
 {<int> }

5.8 MODIFYING PROGRAM FILE ENTRIES

SCI provides commands to change the information supplied when an overlay, task, or procedure was installed. The modifications can also be made to modules installed by the Link Editor. These commands allow the user to modify the runtime environment of a program without reinstalling the module.

5.8.1 MODIFY TASK ENTRY (MTE). The Modify Task Entry (MTE) command allows the user to alter the data supplied when the task was installed. When the MTE command is called, the following display is presented:

MTE
MODIFY TASK ENTRY

PROGRAM FILE PATHNAME: <acnm>
MODULE NAME OR ID: {<string>}
 {<int> }

In response to the PROGRAM FILE PATHNAME: prompt, the user enters the pathname of the program file within which the task, as identified by the response to the MODULE NAME or ID: prompt, resides.

Enter either the task name or the installed ID in response to the MODULE NAME or ID: prompts. Once these responses have been entered, the following is displayed:

 ID:
 NAME:
 REAL TIME:
 PRIORITY:
 MODIFY FLAGS?:
ATTACHED PROCEDURES?:

The values displayed for ID:, NAME:, REAL TIME:, and PRIORITY: are the values that were defined when the task was installed. The values displayed for MODIFY FLAGS?: and ATTACHED PROCEDURES?: are YES and NO, respectively. The cursor is set in the first position of the NAME field. Any of the entries may be changed or they may be accepted by pressing the RETURN key. There are some limitations on changing the REAL TIME: and PRIORITY: prompts. If REAL TIME: is YES, the value for PRIORITY: must be between 1 and 127₁₀ (inclusive). If REAL TIME: is NO, the value must be between 0 and 4 (inclusive).



After the user enters the responses to the above prompts, the task flags are displayed if MODIFY FLAGS?: is YES:

```

SYSTEM:
PRIVILEGED:
MEMORY RESIDENT:
REPLICABLE:
DELETE PROTECTED:
EXECUTE PROTECTED:
OVERFLOW:
WRITABLE CONTROL STORAGE:

```

The values defined when the task was installed are the values displayed. Any of the entries may be changed, or they may be accepted by pressing the RETURN key. If, however, the user wishes to change the SYSTEM: prompt to YES, two conditions must be met. The ATTACHED PROCEDURES?: prompt in the first display must have been changed to YES (to specify that the task is linked with procedure 1 is a dummy procedure on S\$PROGA file), and the task's load address must be $\geq C000_{16}$. Otherwise, the user cannot make the change. After responses have been entered, the procedure prompts are displayed if the ATTACHED PROCEDURES?: prompt was changed to YES.

```

1ST PROCEDURE ID:
P1 FROM TASKS PROGRAM FILE:
2ND PROCEDURE ID:
P2 FROM TASKS PROGRAM FILE:

```

The values displayed are the values defined when the task was installed. The user can change or can accept them by pressing the RETURN key.

5.8.2 MODIFY PROCEDURE ENTRY (MPE). The Modify Procedure Entry (MPE) command allows the user to modify the data supplied when the procedure was installed. When the MPE command is called, the following display is presented:

```

MPE
MODIFY PROCEDURE ENTRY
PROGRAM FILE PATHNAME:  <acnm>
MODULE NAME OR ID:      {<string>}
                        {<int>  }

```

In response to the PROGRAM FILE PATHNAME: prompt, the user enters the pathname of the program file within which the procedure, as identified by the response to the MODULE NAME OR ID: prompt, resides. Enter either the procedure name or the installed ID in response to the MODULE NAME OR ID: prompt. Once these responses have been entered, the following is displayed:

```

ID:
NAME:
MEMORY RESIDENT:
DELETE PROTECTED:
EXECUTE PROTECTED:
WRITE PROTECTED:
WRITABLE CONTROL STORAGE:

```




The values displayed are those that were defined when the procedure was installed. The cursor is in the first position of the NAME: field. The user can change displayed values or can accept each one by pressing the RETURN key.

5.8.3 MODIFY OVERLAY ENTRY (MOE). The Modify Overlay Entry (MOE) command allows the user to alter the data supplied when the overlay was installed. When the MOE command is called, the following display is presented:

```
MOE
MODIFY OVERLAY ENTRY
PROGRAM FILE PATHNAME:  <acnm>
MODULE NAME OR ID:     {<string>}
                       {<id>  }
```

The PROGRAM FILE PATHNAME: prompt is responded to with the pathname of the program file upon which the overlay, as identified by the response to the MODULE NAME OR ID: prompt, is resident. Enter either the overlay name or the installed ID of the overlay in response to the MODULE NAME OR ID: prompt. Once these responses have been entered, the following is displayed:

```
          ID:
        NAME:
    RELOCATABLE:
DELETE PROTECTED:
```

The values defined when the overlay was installed are displayed, with the cursor in the first position of the NAME: field. The user can change any of the entries or can accept them by pressing the RETURN key.



SECTION 6

EXECUTING PROGRAMS

6.1 INTRODUCTION

Many commands are provided to execute tasks. Three of these commands are used for assembly language tasks, while the others are used for executing tasks of the various language processors available for the Models 990/10 and 990/12 computers. Subsystems and language processors are also invoked by SCI commands.

The symbol [] is the SCI command prompt. When the symbol is shown in the command formats, it immediately precedes the command entry. Entry of the command is followed by pressing the RETURN key, as is the entry of each parameter requested. The RETURN key is also used to accept a displayed parameter value, or to indicate no entry.

6.2 EXECUTING AN ASSEMBLY LANGUAGE TASK

The three commands for executing assembly language tasks each serve a particular function. These commands are described and their syntax given in the following paragraphs.

6.2.1 EXECUTE TASK – XT. The XT command is used to execute a task and to leave SCI active during task execution. This command is used for most tasks, except those being debugged and terminal interactive tasks. The format of the command is as follows:

```

XT
EXECUTE TASK
PROGRAM FILE OR LUNO:    <string>
TASK NAME OR ID:        <string>
PARM1:                   { <int> }
                           {  0  }
PARM2:                   { <int> }
                           {  0  }
STATION ID:              { <string> }
                           { ME  }

```

In response to the PROGRAM FILE OR LUNO: prompt, the user enters either the pathname associated with the program file on which the task resides or the LUNO assigned to the program file. Either the name or the installed ID of the task is entered in response to the TASK NAME OR ID: prompt. One or two parameters, each two bytes in size, may be passed to the task being called by entering the desired values in response to the PARM1: and PARM2: prompts. The default for both prompts is zero. In response to the STATION ID: prompt, enter the number of the station (i.e., 01, 02, etc.) with which the task is to be associated.



6.2.2 EXECUTE TASK AND SUSPEND SCI – XTS. The XTS command activates the specified task and suspends SCI until the task terminates. This command should be used for terminal interactive tasks to avoid contention between SCI and the task for terminal access. The format of the command is as follows:

```
XTS
EXECUTE TASK AND SUSPEND SCI
PROGRAM FILE OR LUNO: <string>
TASK NAME OR ID: <string>
PARM1: { <int> }
        { 0 }
PARM2: { <int> }
        { 0 }
STATION ID: { <string> }
             { ME }
```

The prompts are as described for the XT command.

6.2.3 EXECUTE AND HALT TASK – XHT. The XHT command places a task in memory in a suspended state so that it can be debugged. Typically, the user places the task to be debugged in memory using XHT, establishes the debug environment (including breakpoints), and then activates the task using the Activate Task (AT) or Resume Task (RT) command. The format of the command is as follows:

```
XHT
EXECUTE AND HALT TASK
PROGRAM FILE OR LUNO: <string>
TASK NAME OR ID: <string>
PARM1: { <int> }
        { 0 }
PARM2: { <int> }
        { 0 }
STATION ID: { <string> }
             { ME }
```

The responses to the prompts are as described for the XT command.

6.3 EXECUTING LANGUAGE PROCESSORS, TASKS FROM LANGUAGE PROCESSORS, AND SUBSYSTEMS

Table 6-1 lists the different subsystem and language processors available with the DX10 operating system and the corresponding location in the associated manual where execution instructions may be found.



Table 6-1. Locating Instructions for Executing Subsystems, Language Processors, and Tasks From Language Processors Available with DX10

Processor/Subsystem	Execution Instructions Location
BASIC	<i>Model 990 Computer TI 990 BASIC Reference Manual, 2250304-9701</i>
COBOL	Section 8 of <i>Model 990 Computer COBOL Programmer's Guide, 2270521-9701</i>
DBMS	Section 6 of <i>Model 990 Computer Data Base Administrator User's Guide, 2250426-9701</i>
FORTRAN	Appendix H of <i>Model 990 Computer FORTRAN Reference Manual, 946260-9701</i>
FORTRAN-78	Section 2 of <i>Model 990 Computer FORTRAN-78 User's Guide for the DX Operating Systems, 2268679-9701</i>
Pascal	Appendix C of <i>Model 990 Computer TI Pascal User's Manual, 946290-9701</i>
RPGII	Section 2 of <i>Model 990 Computer Report Program Generator (RPGII) Programmer's Guide, 939524-9701</i>
SORT/MERGE	Section 8 of <i>Model 990 Computer SORT/MERGE User's Guide, 946252-9701</i>



SECTION 7

DEBUGGING SUPPORT

7.1 GENERAL

Flaws in software are commonly called bugs. The process of removing flaws from software is called debugging. Modern programming techniques can greatly reduce the number of bugs in a program; however, the bugs that remain tend to be subtle and hard to find. Several levels of debugging support are provided:

- High-level language (FORTRAN, COBOL, BASIC, RPG, Pascal) programs are provided with two levels of debugging:
 1. The compilers and interpreters for these languages provide error messages that pinpoint syntax errors in the source programs.
 2. The runtime packages provide error-tracing information in addition to error messages that describe the nature of the error.
- Several System Command Interpreter (SCI) commands provide debugging capabilities without requiring a special mode of operation.
- A special mode of operation allows a single task to be examined in detail during the execution process.

Detailed information about debugging high-level language programs is contained in the appropriate high-level language programmer's guide. Detailed information about the SCI debug commands and the special mode of operation is provided in the following paragraphs.

Since all of the debug commands interact with the terminal, special care must be taken when debugging a program that uses the terminal, since two processes requesting terminal support can be confusing. If the program being debugged requires use of a terminal, two terminals should be used – one for the program and one for debugging.

7.2 MODES OF DEBUGGING

There are two sets of debug commands. One set can be used only on controlled tasks, which are tasks that have been put into debug mode through the use of the Execute Debug (XD) command. The other set of commands may be used on all tasks. In either case, beware of tasks that unconditionally suspend themselves, since some of the debug commands may inadvertently reactivate these tasks.

NOTE

Putting a task into controlled mode affects the execution of all debug commands as follows:

1. Symbolic expressions may be used in place of integer expressions on commands involving the controlled task.
2. Every command expects the controlled task to be unconditionally suspended.



3. Every command leaves the controlled task unconditionally suspended.
4. During the commands, Proceed from Breakpoint (PB), Delete and Proceed from Breakpoint (DPB), and Resume Task (RT), the command key automatically suspends the controlled task.

7.2.1 UNCONDITIONAL SUSPEND. Most of the debugging commands require that the task being debugged be unconditionally suspended either before or during the debug command. The unconditional suspend task state under DX10 (task state 6) is the state in which the task is dormant until activated by a command. There are several ways for a task to become unconditionally suspended:

1. The task is bid with the suspend option selected, either with a supervisor call (see the *Model 990 Computer — DX10 Operating System Application Programmer's Guide*), the Execute and Halt Task (XHT) SCI command, or the .DBID SCI primitive.

The .DBID primitive is used for tasks that interface through SCI, such as command processors which are normally bid using the .BID and .QBID primitives, as described in the *Model 990 Computer DX10 Operating System Systems Programming Guide*, and high-level language programs (e.g., FORTRAN or COBOL programs). When the .DBID primitive is executed through SCI, the task is bid and immediately placed in a suspended state. The run ID of the task is saved in the synonym \$\$BT or it may be obtained by issuing a Show Task Status (STS) command.

The XHT command is used for tasks that are normally executed directly by an Execute Task (XT) command. XHT places the task in a suspended state for debugging and displays the run ID of the task to the user. If the user desires to execute and halt the task, and simultaneously place it in controlled mode, the Execute Debug (XD) command may be used with no input for the RUN ID prompt. The XD command performs the XHT and saves the run ID as the default for the debugger commands.

2. The task suspends itself.
3. The task executes a breakpoint (XOP 15, 15).
4. The task is suspended by the SCI debug commands.

Once the task has been placed in a suspended state, the debugger may be used to assign breakpoints, simulate execution, display memory, and perform other debugging functions. When the debugging session is over, the task may be terminated via the Kill Task (KT) command. If the task was put into controlled mode by an XD command, it may be killed by responding 'YES' to the KILL TASK? prompt of the Quit Debug (QD) command.

7.2.2 COMMAND PARAMETER SYNTAX. In the following discussion of the commands, the syntax shown represents the actual display. The user enters the appropriate command when the SCI command prompt, [], appears. The following conventions are used in the discussion of the commands:

previous — Indicates the last value entered for the prompt.

acnm — Indicates an access name (either a device name, or a file pathname).

string — Indicates a character string.



constant exp – Indicates a decimal or hexadecimal integer or an expression composed of decimal or hexadecimal integers and the operators +, -, *, and /.

full exp – Indicates a constant expression with the additional operators <, >, and (). String operands are also permitted. In controlled mode, symbolic names and the symbols #PC, #WP, #ST, and #Rn are permitted.

full exp list – Indicates a list of full expressions separated by commas. The list may contain a single item.

constant exp list – Indicates a list of constant expressions separated by commas. The list may contain a single item.

Y

N – Indicates a character string beginning with Y or N.

Underscore (_) – Indicates the default value.

Braces { } – Indicates that a selection of the entries must be made.

Angle brackets < > – Indicates an operator entry.

Brackets [] – Indicate an optional parameter.

Boldface Type – Indicates an operator command entry.

Upper Case – Indicates system displayed data.

References to RETURN within the command descriptions refer to different function keys, depending on the terminal type in use (see table 2-1). To enter the command mode, press the appropriate enter command mode key for the terminal in use (table 2-1).

7.2.3 SYMBOLS. The debug support provided allows for symbolic debugging. Symbolic debugging allows the user to specify labels within the task being debugged rather than memory addresses. This allows for more convenient and meaningful debugging since the source code list can be used as reference for the symbolic labels used. Symbolic constants consist of the Link Edit phase name, a period (.), the module identifier name (IDT), a period (.), and the symbol, an assembly language label. The syntax is defined as:

<phase name>.<IDT name>.<symbol>

To have full symbolic capability, both the assembler and Link Editor must have used the SYMT option. If the assembler did not use the SYMT option but the Link Editor did, the following symbols are available:

<phase name>.<IDT name>

If either the phase name or the IDT name of a symbol is omitted, the immediately preceding corresponding value is used. The syntax is as follows:

.<IDT name> .<symbol> (no phase name)

<phase name> ..<symbol> (no IDT name)

..<symbol> (no phase or IDT name)



Examples:

PHASE1.MOD1.XYZ	References Phase = PHASE1 IDT = MOD1 Label = XYZ
.MOD2.MNO	References Phase = PHASE1 IDT = MOD2 Label = MNO
..ABC	References Phase = PHASE1 IDT = MOD2 Label = ABC

Four words of memory per symbol are required to store symbol values.

If the task being debugged is a single routine that was installed without being linked, then a symbolic constant consists of a period (.), the characters of the module identifier name, a period (.), and the characters of the symbol.

.<IDT name>.<symbol>

As with the linked module, the SYMT option of the assembler must have been selected to have full symbolic capability. If the IDT name of a symbolic constant is omitted, the immediately previous corresponding value is used.

Examples:

.PROG.XYZ
..SYM

NOTE

Symbols may only be used for commands affecting a task which has been placed in the controlled mode by the Execute Debug (XD) command.

The method used to encode the symbol does *not* guarantee unique representation of the symbols. An error message appears whenever two symbols are encoded to the same value. The second symbol cannot be used.

7.2.4 EXPRESSIONS. Constants (and symbolic constants for tasks in the controlled mode) may be combined using the operators +, -, *, /, <, >, and () to form expressions which may be used as command operands. The operators have the following meanings:

+	unary plus or addition
-	unary minus or subtraction
*	multiplication
/	division
()	evaluation order
<>	the contents of the indicated memory location



In the syntax definitions in this section, the use of angle brackets in expressions might be confused with the use of angle brackets to indicate items that must be supplied by the user. To avoid confusion, items supplied by the user are shown as lowercase letters enclosed in angle brackets. Angle brackets enclosing numerals or uppercase letters indicate the contents of an address.

Expressions are evaluated according to the following rules:

1. Subexpressions delimited by () and < > are evaluated first, with the innermost expression evaluated before any other levels.
2. Unless directed otherwise by parentheses or angle brackets, unary + and - are evaluated first, multiplication and division are evaluated second, and addition and subtraction last.
3. For operators at the same level, evaluation proceeds left to right.

For example, if .IDTNAM.BEGIN is memory address 7A, and if memory address 7F contains 3B, then the expression FF/(IDTNAM.BEGIN+5 + 2+3* F) is evaluated as follows:

```
>FF/((<IDTNAM.BEGIN+5>+2+3*>F)
>FF/((<>7A+5>+2+3*>F)
>FF/((<>7F>+2+3*>F)
>FF/(>3B+2+>2D)
>FF/(>3B+(-2)+>2D)
>FF/(>39+>2D)
>FF/>66
2
```

NOTE

The right angle bracket, >, will be regarded as a hexadecimal number indicator rather than the right part of < > whenever there are hexadecimal digits immediately following. Thus, no conflict arises.

Several special symbols are allowed in expressions. These special symbols are:

- #PC – represents the contents of the Program Counter
- #WP – represents the contents of the Workspace Pointer
- #ST – represents the contents of the Status Register
- #Rn – where n has the value 0-15, and #Rn represents the contents of the corresponding workspace register.

NOTE

These special symbols may only be used for commands affecting a task which has been placed in the controlled mode by the Execute Debug (XD) command.



Character strings are also allowed in expressions. A character string is of the form 'XXXX_' where 'X' is any valid ASCII character. The apostrophe can be represented in a character string by using double apostrophes. A character string may be any length, but only the leftmost four characters are significant. Strings shorter than four characters are right-justified. The value of a character string is an expression in the ASCII hexadecimal representation of the characters expressed as a 32-bit number.

Example:

String	Value
'ABCD'	41424344
'A'	00000041
'ABCDE'	41424344
' '	00000000
'A' 'B'	00412742

These symbols may be used in expression lists in the same way as constants or symbolic constants. For example, the following is a valid expression:

```
#PC + NAME.IDT - #R15
```

7.3 COMMANDS FOR ALL TASKS

The SCI commands listed in table 7-1 may be used for all tasks. These commands are classified as commands for reference since their most frequent use is expected to be program debugging. Nothing, however, prohibits the use of these commands for purposes other than debugging; they may be used whenever SCI is active. To activate SCI, perform the procedure given in the *Model 990 Computer DX10 Operating System Production Operation* manual. Many of the debug commands require the runtime task ID returned by the XT or XHT commands. Make note of the runtime task ID when the task is placed in execution. The Show Task Status (STS) command may be used to recover the runtime ID (which identifies the task to DX10).

Table 7-1. SCI Debug Commands

Command	Meaning	Section
DATA DISPLAY COMMANDS		
LB	List Breakpoints	7.3.19
LLR	List Logical Record	7.3.29
LM	List Memory	7.3.7
LSM	List System Memory	7.3.8
SAD	Show Absolute Disk	7.3.23
SADU	Show Addressable Disk Unit	7.3.25
SIR	Show Internal Registers	7.3.15
SP	Show Panel	7.3.18
SPI	Show Program Image	7.3.21
SRF	Show Relative to File	7.3.27
SV	Show Value	7.3.20
SWR	Show Workspace Registers	7.3.17

**Table 7-1. SCI Debug Commands (Continued)**

Command	Meaning	Section
DATA MODIFICATION COMMANDS		
MAD	Modify Absolute Disk	7.3.24
MADU	Modify Addressable Disk Unit	7.3.26
MIR	Modify Internal Registers	7.3.14
MM	Modify Memory	7.3.5
MPI	Modify Program Image	7.3.22
MRF	Modify Relative to File	7.3.28
MSM	Modify System Memory	7.3.6
MWR	Modify Workspace Registers	7.3.16
BREAKPOINT COMMANDS		
AB	Assign Breakpoints	7.3.1
DB	Delete Breakpoints	7.3.2
DPB	Delete and Proceed from Breakpoint	7.3.4
LB	List Breakpoints	7.3.19
PB	Proceed from Breakpoint	7.3.3
TASK CONTROL COMMANDS		
AT	Activate Task	7.3.11
HT	Halt Task	7.3.12
RT	Resume Task	7.3.13
XD	Change Task to Debug mode (only one task per station at any given time)	7.4.1
SEARCH COMMANDS		
FB	Find Byte	7.3.10
FW	Find Word	7.3.9
CONTROLLED TASK COMMANDS		
ASB	Assign Simulated Breakpoint	7.4.3
DSB	Delete Simulated Breakpoint	7.4.4
LSB	List Simulated Breakpoint	7.4.6
QD	Quit Debug	7.4.7
RST	Resume Simulated Breakpoint	7.4.5
ST	Simulate Task	7.4.2



7.3.1 AB – ASSIGN BREAKPOINTS. This command may be issued from any terminal. The contents of the specified address(es) in the specified task are replaced by a breakpoint (an XOP 15,15). This effectively stops execution of the task at that location. Thus, the task may be suspended at any location in its execution. The contents of this location are saved and can be restored by the Delete Breakpoints command. A maximum number (specified at system generation) of breakpoints can be in effect on a DX10 system at any one time; an attempt to use more than this number of breakpoints generates an error message. If the runtime ID specifies a system task, the user must be a privileged user or the command is aborted. Moreover, breakpoints may not be set in the DX10 system area. A task need not be memory-resident to be breakpointed. The task to be breakpointed is temporarily suspended while the breakpoints are inserted and its original state is restored. Unless the task is the controlled task, the user must monitor the task with the Show Internal Registers (SIR) or the Show Panel (SP) command to determine when it reaches a breakpoint. When the task reaches a breakpoint, it is placed in state 6 (unconditional suspend). To proceed, use the Proceed from Breakpoint (PB) command, the Delete and Proceed from Breakpoint (DPB) command, or the Delete Breakpoint (DB) and Resume Task (RT) commands.

Syntax:

AB
ASSIGN BREAKPOINTS

RUN ID:	{ <constant exp> previous ID }	runtime ID
ADDRESS(ES):	{ <full exp list> }	address of breakpoint

Example:

ASSIGN BREAKPOINTS

RUN ID:	>A0
ADDRESS(ES):	>200, >30C, >41A

7.3.2 DB – DELETE BREAKPOINTS. If a breakpoint (XOP 15,15) exists in the specified task at the specified address(es), it is replaced with the original value at the location. The parameters are interpreted as in the Assign Breakpoints command with the following exceptions. If no address is specified, the default is the breakpoint at which the task is currently stopped. If “ALL” is specified, all breakpoints for that task are deleted. If the indicated breakpoint does not exist, or a breakpoint within a list of breakpoints does not exist, the user is warned with an error message and the panel is displayed to show the breakpoint status. Deleting a breakpoint at which a task is stopped does not cause the task to resume execution.

The task is temporarily suspended while the breakpoints are deleted and its original state restored.

Syntax:

DB
DELETE BREAKPOINTS

RUN ID:	{ <constant exp> previous ID }	runtime ID
ADDRESS(ES):	{ <full exp list> <ALL> current breakpoint }	breakpoint address



Example:

DELETE BREAKPOINTS

RUN ID: >A0
ADDRESS(ES): >200, >41A, >AC, >506

DELETE BREAKPOINTS

RUN ID: 80
ADDRESS(ES): ALL

7.3.3 PB – PROCEED FROM BREAKPOINT. This command assigns new breakpoints in the specified task at the destination addresses, if any are indicated, and the task is resumed, bypassing the breakpoint at which it is currently stopped. The breakpoint remains active. If the task is not currently at a breakpoint, the new breakpoints are assigned and the user is notified, by a warning message, that the task was not at a breakpoint. The runtime ID is interpreted as in the Assign Breakpoint command.

Syntax:

PB
PROCEED FROM BREAKPOINT
RUN ID: $\left\{ \begin{array}{l} \text{<constant exp>} \\ \text{previous} \end{array} \right\}$ runtime ID of task
DESTINATION ADDRESS(ES): <full exp list> address of breakpoint(s)
to assign

Example:

PROCEED FROM BREAKPOINT

RUN ID: >9A
ADDRESS(ES): >A0, >10, >A14, >2B

7.3.4 DPB – DELETE AND PROCEED FROM BREAKPOINT. This command is exactly like the Proceed from Breakpoint command except that the breakpoint at which the task is currently stopped is deleted. If this breakpoint has already been deleted, the command functions as if it were a PB command.

Syntax:

DPB
DELETE AND PROCEED FROM BREAKPOINT
RUN ID: $\left\{ \begin{array}{l} \text{<constant exp>} \\ \text{previous} \end{array} \right\}$ runtime ID
DESTINATION ADDRESS(ES): <full exp list> address of breakpoint(s) to
assign

Example:

DELETE AND PROCEED FROM BREAKPOINT

RUN ID: >4E
DESTINATION ADDRESS(ES): >1A, >2FE, >340



7.3.5 MM – MODIFY MEMORY. The memory image of the specified task is modified using the input data, starting at the address specified. Roll-in/roll-out does not affect the modification process. A runtime ID of S specifies the DX10 system area ROOT segment. Only users with privileged user IDs are allowed to modify the system area or system tasks. Modify System Memory (MSM) is available to modify the other parts of the system. If the task is not unconditionally suspended, it is temporarily suspended while the command is interacting. This command is interactive and may be terminated at any time by pressing the CMD key (for the 911 VDT). See table 2-1 for equivalent keys on other terminals.

Syntax:

MM
MODIFY MEMORY

RUN ID:	{ <constant exp> previous }	runtime ID
ADDRESS:	<full exp list>	address to modify

The command displays the specified address followed by its value. When the user enters a new value followed by a return, the next consecutive address and its value are displayed. The CMD (911 VDT), HELP (913 VDT), or CNTL and X (hard copy) key returns the user to the command mode.

Example:

```

MODIFY MEMORY
      RUN ID:  >10
      ADDRESS:  >10
0010:  >04C0          (Press RETURN)
0012:  >0500      >0502 (Press RETURN)
0014:  >0100          (Enter Command Mode)

```

7.3.6 MSM – MODIFY SYSTEM MEMORY. This command is used to modify the memory occupied by the DX10 operating system. This command is similar to the MM (MODIFY MEMORY) debugger command (see paragraph 7.3.5) except that an overlay ID is specified instead of a run ID. This command is intended for use only by someone who is very familiar with the DX10 source.

7.3.6.1 MSM Command Format.

MSM
MODIFY SYSTEM MEMORY

OVERLAY ID:	<INT>
ADDRESS:	<INT>



7.3.6.2 MSM Command User Responses.

System Prompts	Response Required or Optional	User Responses
Overlay ID:	R	Number of overlay whose memory is to be modified
Address:	R	Address at which to begin modifying memory

7.3.6.3 MSM Command Example. The MSM command modifies the memory of system overlay 2, starting at address 05900₁₆.

```
MSM
MODIFY SYSTEM MEMORY
      OVERLAY ID:  2
      ADDRESS:    05900
      5900:      >0004 6
      5902:      >0000 1
      5904:      >0429
```

7.3.7 LM — LIST MEMORY. This command lists the specified memory area of a program on the specified output device or file. The runtime ID is interpreted as in the Modify Memory command. The output defaults to the terminal. If the task is not unconditionally suspended, it is temporarily suspended while the listing is being formatted.

In the output, the contents of 16 bytes are printed per line. The address of the first byte is the first entry on the line. The contents of each pair of bytes are shown as four hexadecimal digits. At the right end of the line, the contents are printed as ASCII characters. The bytes that contain values corresponding to printable ASCII characters are translated and then displayed or printed as ASCII characters. The nonprintable ASCII characters are displayed or printed as periods.

Syntax:

```
LM
LIST MEMORY
```

RUN ID:	{ <constant exp> previous }	runtime ID
STARTING ADDRESS:	<full exp>	starting address
NUMBER OF BYTES:	<full exp>	length of display
LISTING ACCESS NAME:	<acnm>	output device or file name

Example:

```
LIST MEMORY
      RUN ID:    >80
      STARTING ADDRESS: >102
      NUMBER OF BYTES:  >14A
      LISTING ACCESS NAME: LP01
```



7.3.8 LSM – LIST SYSTEM MEMORY. This command lists the memory occupied by the DX10 operating system. The command is similar to the List Memory (LM) command (see paragraph 7.3.7) except that an overlay ID is specified instead of a run ID. This command is intended for use only by someone who is very familiar with the DX10 source.

In the output, the contents of 16 bytes are printed per line. The address of the first byte is the first entry on the line. The contents of each pair of bytes are shown as four hexadecimal digits. At the right end of the line, the contents are printed as ASCII characters. The bytes that contain values corresponding to printable ASCII characters are translated and then displayed or printed as ASCII characters. The nonprintable ASCII characters are displayed or printed as periods.

7.3.8.1 LSM Command Format.

```
LSM
LIST SYSTEM MEMORY
      OVERLAY ID:  <int>
STARTING ADDRESS:  <int>
      NUMBER OF BYTES:  [int]
      LISTING ACCESS:  [acnm]
```

7.3.8.2 LSM Command User Responses.

System Prompts	Response Required or Optional	User Responses
OVERLAY ID:	R	Number of the overlay whose memory is to be listed.
STARTING ADDRESS:	R	Address at which to begin listing memory.
NUMBER OF BYTES:	O	Number of bytes to list. <i>Default value is 16.</i>
LISTING ACCESS NAME:	O	Access name of file or device to which output is to be sent. <i>The default is the terminal local file of the terminal.</i>

7.3.8.3 LSM Command Example. The LSM command lists the memory of system overlay 6 from byte 0100₁₆ to byte 0120₁₆.

```
[ ] LSM
LIST SYSTEM MEMORY
      OVERLAY ID:  6
STARTING ADDRESS:  0100
      NUMBER OF BYTES:  020
      LISTING ACCESS NAME:
0100 352E 0006 0006 0000 0000 1DF2 BE03 1C40 5. .. .. .
0110 1000 0000 0000 0000 0000 0000 0000 0000 .. .. .. .
[ ]
```




7.3.9 FW – FIND WORD. The specified memory area in the specified program is searched for the value (or successive values). The search begins on a word boundary. If the value is found, the address at which it was found is displayed. The runtime ID and addresses are interpreted as in the Modify Memory command. If the task is not unconditionally suspended, it is temporarily suspended while the search is performed.

Syntax:

FW
FIND WORD

RUN ID:	{ <constant exp> previous }	runtime ID
VALUE(S):	<full exp list>	value(s) to find
STARTING ADDRESS:	<full exp>	starting address
ENDING ADDRESS:	<full exp>	ending address

Example:

FIND WORD

RUN ID:	>7F
VALUES:	>80A, >80B, >80C, >80D
STARTING ADDRESS:	>AC
ENDING ADDRESS:	>CAE

7.3.10 FB – FIND BYTE. This command performs the same functions as the Find Word command except it applies to bytes. The search starts on a byte boundary and increments forward one byte at a time.

Syntax:

FB
FIND BYTE

RUN ID:	{ <constant exp> previous }
VALUE(S):	<full exp list>
STARTING ADDRESS:	<full exp>
ENDING ADDRESS:	<full exp>

Example:

FB (FIND BYTE)

RUN ID:	>10
VALUE(S):	6, 7, >A, 9, >A, >BC, 2, 1
STARTING ADDRESS:	>2F
ENDING ADDRESS:	>3AC

7.3.11 AT – ACTIVATE TASK. This command causes the specified task to be activated if it is unconditionally suspended. The command is turned into a No Operation command if the task is not unconditionally suspended.



Syntax:

```
AT
ACTIVATE TASK
    RUN ID: {<constant exp>}
             {previous}
```

Example:

```
ACTIVATE TASK
    RUN ID: >A
```

7.3.12 HT — HALT TASK. The specified task is unconditionally suspended at the end of the current time slice. The runtime ID is interpreted as in the Assign Breakpoints command. If the specified task is already unconditionally suspended, this command is turned into a No Operation command. If the task is not in the active state, this command waits five seconds for the task to reach an unconditional suspension and gives the user the option of aborting or continuing to wait. This occurs *every* five seconds.

Syntax:

```
HT
HALT TASK
    RUN ID: {<constrant exp>}
             {previous}
```

Example

```
HALT TASK
    RUN ID: 7
```

If the task cannot be suspended, the following message is displayed:

```
UNABLE TO SUSPEND TASK. CURRENT STATE=XX. CONTINUE COMMAND?
```

If a YES response is entered, another attempt is made to suspend the task. If unsuccessful, the message is displayed again. A NO response to the preceding message causes the following message to be displayed:

```
DO YOU WISH TO LEAVE SUSPENSION PENDING?
```

A YES response leaves the suspension pending, while a NO response terminates the suspension attempt.

7.3.13 RT – RESUME TASK. The specified task is activated at the point at which it was suspended. The runtime ID is assigned when the task is executed. The specified task must be unconditionally suspended when this command is executed or an error is indicated. Either the Delete Breakpoint (DB) and the Resume Task (RT) commands, the Proceed from Breakpoint (PB) command, or the Delete and Proceed from Breakpoint (DPB) command must be used to restart the task halted at a breakpoint. The RT command, rather than the Activate Task (AT) command, should be used to reactivate a task halted by the Halt Task (HT) command.



Syntax:

```

RT
RESUME TASK
      RUN ID: { <constant exp> }
               { previous }

```

Example:

```

RESUME TASK
      RUN ID: 7

```

7.3.14 MIR – MODIFY INTERNAL REGISTERS. The internal registers (Program Counter, Workspace Pointer, and Status Register) for the specified task are modified according to the user inputs. If the task being debugged is not a privileged task, then only bits 0 through 6 of the status register can be modified with this command. This command acts interactively, like the Modify Memory command. If the task is not unconditionally suspended, it is temporarily suspended while the command is interacting. The runtime ID is interpreted as in the Assign Breakpoints (AB) command. This command is interactive and may be terminated at any time by pressing the CMD key (for the 911 VDT). See table 2-1 for key equivalents on other terminals.

Syntax:

```

MIR
MODIFY INTERNAL REGISTERS
      RUN ID: { <constant exp> }
               { previous }

```

Once the RUN ID: is entered, the RETURN key is pressed and the following display is presented:

```

PC:XXXX
WP:XXXX
ST:XXXX

```

where

XXXX represents the contents of each register. Modifications to each register are entered after the contents for the appropriate register.

Example:

```

MODIFY INTERNAL REGISTERS
      RUN ID: >24
      PC: 0106   Press RETURN
      WP: 0040   >60
      ST: E40F   >40F

```

Results:

```

No change to PC
Change WP to >60
Change ST to >40F

```



7.3.15 SIR – SHOW INTERNAL REGISTERS. The internal registers are displayed on the terminal. The runtime ID is interpreted as in the Assign Breakpoint command. The displayed state is the state of the task before it was suspended to capture the internal registers, while the remainder of the display reflects the values in effect after the task was suspended. The character string representation of the status register follows the hexadecimal value and may include the following characters: L = logical greater than, A = arithmetic greater than, E = equal, C = carry, O = overflow, P = parity, X = XOP in progress, S = privileged mode, and M = map file.

Syntax:

```
SIR
SHOW INTERNAL REGISTERS
```

```
RUN ID: { <constant exp>
          previous }
```

Example display:

```
RUN ID=0E STATE=06 (BP) WP=0082 PC=0016 PC=2FCF ST=218F E M
```

7.3.16 MWR – MODIFY WORKSPACE REGISTERS. The specified workspace registers of the specified task are modified according to the user inputs. This command is interactive, like the Modify Memory command. New values must be terminated by pressing the RETURN key. If the task is not unconditionally suspended, it is temporarily suspended while the command is interacting. The runtime ID is interpreted as in the Assign Breakpoint command. Because this is an interactive command, it may be terminated at any time by pressing the CMD key (for the 911 VDT). See table 2-1 for key equivalents on other terminals.

Syntax:

```
MWR
MODIFY WORKSPACE REGISTERS
```

```
RUN ID: { <constant exp>
          previous }
```

```
REGISTER NUMBER: { <constant exp>
                   0 } starting register number
```

Example:

```
MODIFY WORKSPACE REGISTERS
```

```
    RUN ID: (Press Return)
REGISTER: 2
R2: 0200 >FFFF
R3: 0300 >3FFF
R4: 062F (Press RETURN)
R5: 8010
```

Return to command mode

7.3.17 SWR – SHOW WORKSPACE REGISTERS. The current workspace for the specified task is displayed. If the task is not unconditionally suspended, it is temporarily suspended while the workspace is displayed. If the terminal requesting the command is a VDT, the SWR command



has the same effect as the Show Panel command. The runtime ID is interpreted as in the Assign Breakpoint command.

Syntax:

```
SWR
SHOW WORKSPACE REGISTERS
    RUN ID:  { <constant exp> }
              { previous }
```

Example:

```
SHOW WORKSPACE REGISTERS
    RUN ID:  >A
```

7.3.18 SP – SHOW PANEL. The debug panel for the specified task is displayed. The runtime ID is interpreted as in the Assign Breakpoint command. If the task is not unconditionally suspended, it will be temporarily suspended while the panel is being formatted and displayed. The displayed task state is the state of the task before it was suspended. The debug panel consists of the internal registers, the workspace registers, breakpoints, memory display, and task state.

Syntax:

```
SP
SHOW PANEL
    RUN ID:  { <constant exp> }
              { previous }
    MEMORY ADDRESS:  <full exp>
```

Example:

```
SHOW PANEL
    RUN ID:  >80
    MEMORY ADDRESS:  >AC
```



Figure 7-1 is an annotated example of the debug panel display.

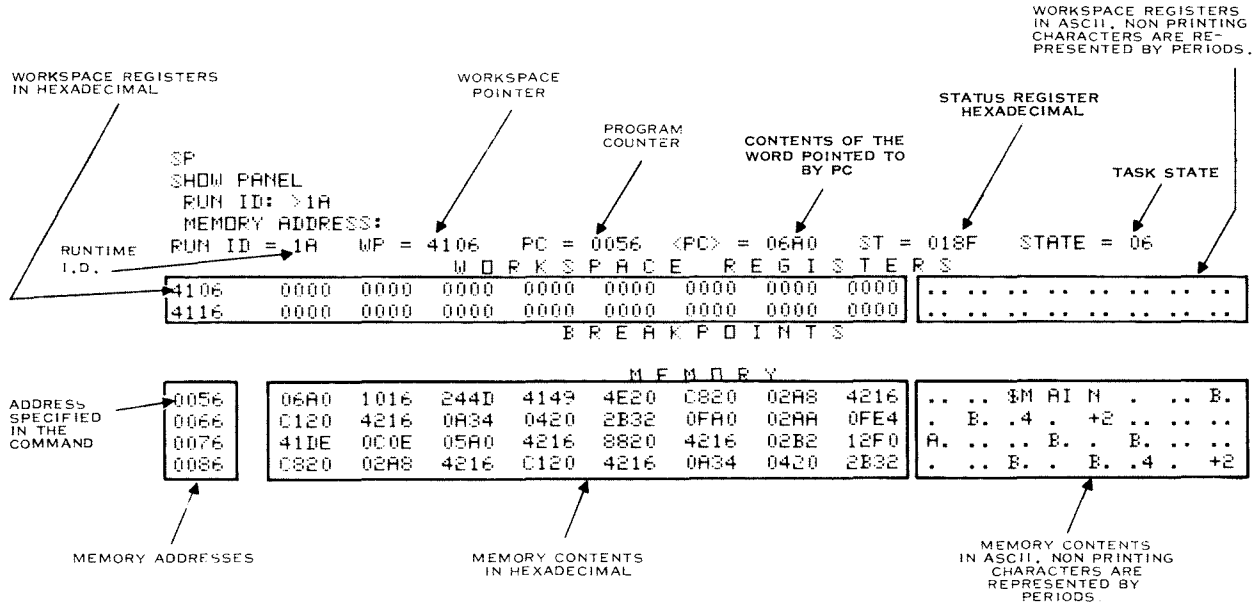


Figure 7-1. Debug Panel Display

7.3.19 LB – LIST BREAKPOINTS. The breakpoints for the specified task are displayed.

Syntax:

```

LB
LIST BREAKPOINTS
RUN ID: { <constant exp>
         previous }

```

Example:

```

LIST BREAKPOINTS
RUN ID: >4C

```

7.3.20 SV – SHOW VALUE. The value of the specified expression is displayed. Its hexadecimal, decimal, and ASCII representations are given.

Syntax:

```

SV
SHOW VALUE
EXPRESSION: <full exp>

```

If a “controlled” task exists, the expression may be symbolic.



Example:

SHOW VALUE
EXPRESSION: >FF/#R8+.NAME.IDT

7.3.21 **SPI — SHOW PROGRAM IMAGE.** The Show Program Image (SPI) command displays the disk-resident memory image of a module (defined as a task, procedure, or overlay) for the specified program file. The display may be directed to a device or a file.

Syntax:

```

SPI
SHOW PROGRAM IMAGE
      PROGRAM FILE:      <acnm>           location of module
      OUTPUT ACCESS NAME: <acnm>           define where to write
                                          output
      MODULE TYPE:      { <Task>
                          <Overlay>
                          <Procedure> }   task, overlay, or procedure
      MODULE NAME OR ID: <string>         name or installed ID
      ADDRESS:          <constant exp>    address to start display
      LENGTH:          <constant exp>    address to end display

```

Examples:

```

[] SPI
SHOW PROGRAM IMAGE
PROGRAM FILE:  DS02.S$PROGA
OUTPUT ACCESS NAME:
MODULE TYPE:  PROCEDURE
MODULE NAME OR ID:  1+2
ADDRESS:  02000
LENGTH:  8
2000  1EFE 1306 9815 1EFA 1605 0585 0606 1220  .. . . . . . . . . . .
[]

```

7.3.22 **MPI — MODIFY PROGRAM IMAGE.** The Modify Program Image (MPI) command modifies a module (defined to be task, procedure, or overlay) in the specified program file using memory addresses and new data supplied by the user.

Syntax:

```

MPI
MODIFY PROGRAM IMAGE
      PROGRAM FILE:      <acnm>           location of module
      OUTPUT ACCESS NAME: <acnm>           define where to write output

```



MODULE TYPE:	{ <TASK> <PROCEDURE> <OVERLAY> }	task, procedure, or overlay
MODULE NAME OR ID:	<string>	name or installed ID
ADDRESS:	<constant exp>	starting memory address
VERIFICATION DATA:	<constant exp>	optional verification data
DATA:	<constant exp list>	New data to be inserted in the module.
CHECKSUM:	<constant exp>	Optional verification data for new data; checksum is an exclusive OR of each word of new data. If the checksum is not known, leaving this field blank causes the checksum to be printed.

RELOCATION OF DATA?: Y data value relocated when task
N loaded in memory for execution

Example:

```
[ ] MPI
MODIFY PROGRAM IMAGE
PROGRAM FILE: DS02.S$PROGA
OUTPUT ACCESS NAME: LP01
MODULE TYPE: PROCEDURE
MODULE NAME OR ID: 1 + 2
ADDRESS: 02000
VERIFICATION DATA: 01EFE
DATA: 01FFF
CHECKSUM:
[ ]
```

7.3.23 SAD — SHOW ABSOLUTE DISK. The Show Absolute Disk (SAD) command prints the contents of a specified absolute address on a disk. The SAD command may be entered only by privileged users. The device is the device name assigned to the disk unit at sysgen time. It normally consists of the characters -DS01- for the system disk and -DS0x-, where “x” is a digit greater than one for other disks on the system.

In the output, the contents of 16 bytes are printed per line. The address of the first byte is the first entry on the line. The contents of each pair of bytes are shown as four hexadecimal digits. At the right end of the line, the contents are printed as ASCII characters. The bytes that contain values corresponding to printable ASCII characters are translated and then displayed or printed as ASCII characters. The nonprintable ASCII characters are displayed or printed as periods.



Syntax:

```

SAD
SHOW ABSOLUTE DISK
      DISK UNIT: <name>           disk drive device name
      TRACK: <constant exp>      starting track address
      SECTOR: <constant exp>     starting sector address
      FIRST WORD: { <constant exp> } starting word address
                   { 0 }
      NUMBER OF WORDS: <constant exp> number of words to show
      OUTPUT ACCESS NAME: <acnm>  define where to write output

```

Example:

```

[] SAD
SHOW ABSOLUTE DISK
      DISK UNIT: DS02
      TRACK: 0
      SECTOR: 0
      FIRST WORD: 0
      NUMBER OF WORDS: 4+4-2
      OUTPUT ACCESS NAME:

TRACK 0000 SECTOR 00 RECORD LENGTH 0120 BYTES (01 SECTORS).
0000 4D41 5931 3030 2020 2610 MA Y1 00 &.
[]

```

7.3.24 MAD — MODIFY ABSOLUTE DISK. The Modify Absolute Disk (MAD) command places specified data on a disk at a specified absolute track, sector, and word address. This command may only be entered by privileged users. The disk unit is the device name given the disk at sysgen time. **FIRST WORD** is the address of the first word on the sector to be loaded with the data being entered. Data is entered in groups of word values to be placed on disk. Each word value must be separated from the next with a comma and values are loaded on disk in successive addresses. The verify parameter allows the user to enter a string of words to be compared against the data at the load address. If a bad compare results, the load does not take place. Since the MAD command can write anything anywhere on the disk and can therefore destroy the DX10 system image, the **VERIFICATION DATA** prompt should always be used.

In the output, the contents of 16 bytes are printed per line. The address of the first byte is the first entry of the line. The contents of each pair of bytes are shown as four hexadecimal digits. At the right end of the line, the contents are printed as ASCII characters. The bytes that contain values corresponding to printable ASCII characters are translated and then displayed or printed as ASCII characters. The nonprintable ASCII characters are displayed or printed as periods.



Syntax:

```

MAD
MODIFY ABSOLUTE DISK
        DISK UNIT:      <name>           disk drive device name
OUTPUT ACCESS NAME:    <acnm>           define where output goes
        TRACK:         <constant exp>   starting track address
        SECTOR:        <constant exp>   starting sector address
        FIRST WORD:    <constant exp>   starting word to modify
VERIFICATION DATA:   <list>           data used to verify
        DATA:         <list>           new data

```

Example:

```

[ ] MAD
MODIFY ABSOLUTE DISK
        DISK UNIT:  DS02
OUTPUT ACCESS NAME:
        TRACK:  0
        SECTOR:  0
        FIRST WORD:  4
VERIFICATION DATA:  03030
        DATA:  02020

0004  2020  2020  2610  0205  0120  0001  0000  0000  &.  . . . . .
[ ]

```

7.3.25 SADU – SHOW ALLOCABLE DISK UNIT. All disks on a DX10 system are addressed in allocable disk units (ADUs). The maximum number of ADUs on a disk is 65,535. Therefore, if a disk contains more than 65,535 sectors, multiple sectors are used as ADUs. ADUs are the basic addressable disk unit in a DX10 system.

This command outputs the contents of the specified ADU to the specified device.

In the output, the contents of 16 bytes are printed per line. The address of the first byte is the first entry of the line. The contents of each pair of bytes are shown as four hexadecimal digits. At the right end of the line, the contents are printed as ASCII characters. The bytes that contain values corresponding to printable ASCII characters are translated and then displayed or printed as ASCII characters. The nonprintable ASCII characters are displayed or printed as periods.

Syntax:

```

SADU
SHOW ALLOCABLE DISK UNIT
        DISK UNIT:      <name>           disk unit
        ADU NUMBER:    <constant exp>   ADU to be shown
        SECTOR OFFSET: <constant exp>   sector offset
        FIRST WORD:    { <constant exp> } first word to show
                        { 0 }

```



NUMBER OF WORDS:	<constant exp>	number of words to show
OUTPUT ACCESS NAME:	<acnm>	output device or file

Example:

```
[ ] SADU
SHOW ALLOCABLE DISK UNIT
      DISK UNIT: DS02
      ADU NUMBER: 1
      SECTOR OFFSET: 0
      FIRST WORD: 0
      NUMBER OF WORDS: 4
      OUTPUT ACCESS NAME:

ADU 0001  SECTOR 00  RECORD LENGTH 0120 BYTES (01 SECTORS).
0000  0000 0000 0000 0000                                .. . . . .
[ ]
```

7.3.26 MADU – MODIFY ALLOCABLE DISK UNIT. All disks on a DX10 system are addressable in addressable disk units (ADUs). (See SADU.)

This command modifies the specified ADU on disk as directed by operator inputs. If verification data does not match the data already on disk, the modification is not performed.

In the output, the contents of 16 bytes are printed per line. The address of the first byte is the first entry of the line. The contents of each pair of bytes are shown as four hexadecimal digits. At the right end of the line, the contents are printed as ASCII characters. The bytes that contain values corresponding to printable ASCII characters are translated and then displayed or printed as ASCII characters. The nonprintable ASCII characters are displayed or printed as periods.

Syntax:

```
MADU
MODIFY ALLOCABLE DISK UNIT
      DISK UNIT:      <name>          disk drive device name
      OUTPUT ACCESS NAME: <name>      define where output is to go
      ADU NUMBER:     <constant exp>  number of ADU to be shown
      SECTOR OFFSET:  <constant exp>  which sector in ADU
      FIRST WORD:     <constant exp>  first word of interest
      VERIFICATION DATA: <constant exp list> data to verify
      DATA:          <constant exp list> data to load
```



Example:

```

[] MADU
MODIFY ALLOCABLE DISK UNIT
      DISK UNIT: DS02
      OUTPUT ACCESS NAME:
      ADU NUMBER: 1
      SECTOR OFFSET: 0
      FIRST WORD: 0
      VERIFICATION DATA: 0
      DATA: 0100+01000

0000 1100 0000 0000 0000 0000 0000 0000 0000 0000  .. . . . . . .
[]

```

7.3.27 SRF – SHOW RELATIVE TO FILE. The Show Relative to File command displays any word or group of words within a file. It assumes that the user has knowledge of the file structure and allows the user to address any word within the file. If a word address is greater than 64K (65,536) bytes, the user *must* supply the record number and a word offset into the record.

In the output, the contents of 16 bytes are printed per line. The address of the first byte is the first entry of the line. The contents of each pair of bytes are shown as four hexadecimal digits. At the right end of the line, the contents are printed as ASCII characters. The bytes that contain values corresponding to printable ASCII characters are translated and then displayed or printed as ASCII characters. The nonprintable ASCII characters are displayed or printed as periods.

Syntax:

```

SRF
SHOW RELATIVE TO FILE
      PATHNAME:      <acnm>           file pathname
      RECORD NUMBER: { <constant exp> } must be specified if word
                      { 0 }           address is over 64K bytes
      FIRST WORD:    <constant exp>   first word to show (must
                                      be an even byte address)
      NUMBER OF WORDS: <constant exp> length of block to show
      OUTPUT ACCESS NAME: <acnm>     output device or file

```

Example:

```

[] SRF
SHOW RELATIVE TO FILE
PATHNAME: .S$PROC.XB
RECORD NUMBER: 0
FIRST WORD: 0
NUMBER OF WORDS: 6
OUTPUT ACCESS NAME:
FILE: .S$PROC.XB RECORD: 000000
0000 0000 0018 1A0B 5842 2028 4558  .. . . . . XB ( EX
[]

```



7.3.28 MRF — MODIFY RELATIVE TO FILE. The Modify Relative to File (MRF) command changes data at an absolute word address within a file. It is assumed that the user has knowledge of the file and disk structure. Addresses greater than 64K (65,536) bytes must have a record number. Words less than 64K bytes can be addressed directly; the sector is located by the program. Verification should be used if the file is critical to a program.

In the output, the contents of 16 bytes are printed per line. The address of the first byte is the first entry of the line. The contents of each pair of bytes are shown as four hexadecimal digits. At the right end of the line, the contents are printed as ASCII characters. The bytes that contain values corresponding to printable ASCII characters are translated and then displayed or printed as ASCII characters. The nonprintable ASCII characters are displayed or printed as periods.

Syntax:

MRF		
MODIFY RELATIVE TO FILE		
PATHNAME:	<acnm>	pathname to file
OUTPUT ACCESS NAME:	<acnm>	output device or file
RECORD NUMBER:	{<constant exp>}	sector offset-required if
	0	word address is over 64K
		bytes
FIRST WORD:	<constant exp>	address of first word (must
		be an even byte address)
VERIFICATION DATA:	<constant exp list>	data to be verified
DATA:	<constant exp list>	data to be loaded
CHECKSUM:	<constant exp>	Optional verification data
		for new data. If the check-
		sum is not known, leaving
		this field blank will cause
		the checksum to be printed
		out. The checksum is an
		exclusive OR of each word
		of new data.

Example:

```
[ ] MRF
MODIFY RELATIVE TO FILE
      PATHNAME: .S$PROC.XB
      OUTPUT ACCESS NAME: LPO1
      RECORD NUMBER: 0
      FIRST WORD: 0
      VERIFICATION DATA: 0
      DATA: 100
      CHECKSUM: 100
[ ]
```

7.3.29 LLR — LIST LOGICAL RECORDS. This command lists selected records from a sequential or relative record file to a device or file. The LLR command differs from the Show File (SF) command in that it allows positioning within the file and also prints both ASCII and hexadecimal formats for the record contents.



Syntax:

```

LLR
LIST LOGICAL RECORD
PATHNAME: <acnm>           pathname to file
STARTING RECORD: {<int>}   relative record number
                        {    }   within the file
NUMBER OF RECORDS: <acnm> length of block to show
LISTING ACCESS NAME: [acnm] listing device or file

```

Example:

```

[ ] LLR
LIST LOGICAL RECORD
      PATHNAME:  .S$PROC.XB
      STARTING RECORD:  0
      NUMBER OF RECORDS:  1
      LISTING ACCESS NAME:

      FILE ACCESS NAME:  .S$PROC.XB
      RECORD:  0 0 0 0 0
0 0 0 0 5842 2028 4558 4543 5554 4520 4241 5443      XB (EX EX UT E BA TC H) ,
0 0 1 0 4829 2C20

```

7.4 COMMANDS FOR CONTROLLED TASKS

These commands may be used for tasks in the controlled mode (see XD command in the next paragraph). The SCI commands may also be used for controlled tasks, and some SCI commands result in additional output. For example, if a breakpoint occurs in a controlled task, then a panel is displayed. Breakpoints in noncontrolled tasks produce no output. If the linked object was specified in the XD command, then symbolic expressions may be used for integer parameters. The controlled mode commands are explained in the following paragraphs.

7.4.1 XD – DEBUG COMMAND. This command places the specified task into controlled mode. The runtime ID is optional, but cannot be that of a system task. If no runtime ID is given, an automatic call is made to the Execute and Halt Task (XHT) command to place the task into execution. The symbol table object file is optional and its presence determines whether symbolic expressions are allowed on any of the subsequent debug commands. If a symbol table was specified to the Link Editor (SYMT option selected) and if the controlled task symbol table object file is specified, then symbolic expressions involving symbols in the object code symbol table may be used in commands which call for string parameters. The debugger may be used to simulate 990/12 object code (when executing on a 990/12) or 990/10 object code (when executing on a 990/10 or a 990/12). The command defaults to object code of the type of the host computer. Only one task for each station may be in debug mode at a given time.

Syntax:

```

XD
EXECUTE DEBUG
      RUN ID:  {<int> }
              { previous }
SYMBOL TABLE OBJECT FILE:  <acnm>
990/12 OBJECT CODE?:  {Y}
                      {N}

```



Example:

```
EXECUTE DEBUG
                                RUN ID:    >9A
SYMBOL TABLE OBJECT FILE:    .OBJ.PROG
                                990/12 OBJECT CODE:    N
```

Example:

```
EXECUTE DEBUG
                                RUN ID:    >9A
SYMBOL TABLE OBJECT FILE:    .OBJ.PROG
```

7.4.2 ST – SIMULATE TASK. The ST command provides controlled and traced execution of the instructions in the controlled task. Controlled execution continues until the execution of a specified number of instructions has been simulated, or until a specified address is placed in the PC, or the occurrence of a breakpoint or simulated breakpoint, whichever occurs first. Simulation may be continued by entering the F3 function key.

Simulated execution continues without operator intervention and locks out further SCI commands. The user can regain SCI capabilities by returning to the command mode. The ST or Resume Simulated Task (RST) commands may be used to reenter simulated execution.

Syntax:

```
ST
SIMULATE TASK
  FOR:    [<full exp>]      symbolic expression
  FROM:   [<full exp>]      symbolic expression
  TO:     [<full exp>]      symbolic expression
```

The FOR operand is an expression that specifies the number of instruction simulations to be performed. The value of the FOR operand is less than or equal to 32,767. When the specified number of simulations has been performed, SCI displays the following message and halts simulation:

TIME OUT

When the FOR operand is omitted, the FOR value specified in the previous ST command is used. If there was no previous ST command, '1' is used. Simulation halts if a breakpoint or simulated breakpoint is encountered, or if the execution of an instruction at a specified address is simulated. If simulation continues without encountering these conditions, the user may regain control of the program by returning to the command mode of SCI.

The FROM address is a constant, variable, or expression that specifies the address of the first instruction to be simulated. When the address is omitted, simulation begins at the instruction whose address is in the PC.

The TO address is a constant, variable or expression that specifies the address of the last instruction to be simulated. Following simulation of that instruction, SCI displays the panel and halts simulation. The TO address may be less than the FROM address. When the TO address is not entered, simulation continues until a breakpoint or simulated breakpoint is encountered, or the user returns to the command mode.



The following example shows an ST command:

SIMULATE TASK	Begin simulation of program
FOR: 25	IDTNAM at location BEGIN
FROM: .IDTNAM.BEGIN	halting after simulating the
TO: . . END	execution of 25 instructions,
	or at location .END.

Single instruction execution is performed by using a FOR parameter equal to 1.

7.4.3 ASB — ASSIGN SIMULATED BREAKPOINT. This command sets up a breakpoint on a range of values for memory alteration (A), CRU access (C), PC value (P), memory references (R), status value (S), or XOPs (X). A memory write operation that does not change the value in memory is *not* a memory alteration. The breakpoints set with this command are valid only during a simulation. Breakpoints, in this case, are conditions that stop execution but allow execution to be resumed on another command, either Resume Simulated Task (RST) or F3 function key on a 911 VDT. See table 2-1 for equivalent keys on other terminals. Each simulated breakpoint is assigned a number that is displayed at the completion of the ASB command. When a breakpoint occurs during simulation, a panel and the breakpoint number are displayed along with the display string. COUNT specifies the number of times the breakpoint will be passed before execution is halted.

Syntax:

ASB

ASSIGN SIMULATED BREAKPOINT

ON (A, C, P, R, S, X):	$\left. \begin{array}{c} \langle A \rangle \\ \langle C \rangle \\ \underline{P} \\ \langle R \rangle \\ \langle S \rangle \\ \langle X \rangle \end{array} \right\}$	Alteration (memory) CRU Access PC Value Reference (memory) ST Value XOP Level
FROM:	<full exp>	symbolic expression for the lower limit for break-pointing
THRU:	<full exp>	symbolic expression for the upper limit for break-pointing
COUNT:	<full exp>	symbolic expression which specifies the number of times this breakpoint is to be encountered before execution is halted
DISPLAY:	<full exp>	symbolic expression for the memory address to be displayed when this breakpoint is reached



Examples:

ASSIGN SIMULATED BREAKPOINT		Set a breakpoint on memory locations 6 through 1C0.
ON (A, C, P, R, S, X):	A	Display the program counter (PC) if the breakpoint is taken.
FROM:	6	Count defaults to 1.
THRU:	>1C0	
COUNT:	(Press RETURN)	
DISPLAY:	(Press RETURN)	
ASSIGN SIMULATED BREAKPOINT		Set a breakpoint on the second occurrence of a PC value between BRANCH and BRANCH+ 24 in the module IDT NAM.
ON (A, C, P, R, S, X):	P	Display location TABLE in IDTNAM if the breakpoint occurs.
FROM:	.IDTNAM.BRANCH	
THRU:	. . BRANCH+ 24	
COUNT:	2	
DISPLAY:	. . TABLE	

7.4.4 DSB – DELETE SIMULATED BREAKPOINT. This command allows the user to delete a list of simulated breakpoints assigned with the ASB command. The only argument is the breakpoint number assigned by the ASB command. The keyword “ALL” deletes all of the simulated breakpoints.

Syntax:

```

DSB
DELETE SIMULATED BREAKPOINT
BREAKPOINT NUMBERS: <full exp>      number of simulated
                     <ALL>          breakpoint

```

Example:

```

DELETE SIMULATED BREAKPOINT
BREAKPOINT NUMBERS: 3

```

7.4.5 RST — RESUME SIMULATED TASK. This command allows the user to resume simulation following a breakpoint, a simulated breakpoint or simulation of a specified number of instructions (time-out). The last entered values for the FOR: and TO: prompts in the Simulate Task (ST) command are used as the RST limits. Upon reaching a terminating condition (breakpoint, simulated breakpoint, time-out, or TO: address), a panel and termination reason are displayed. Simulation may be continued by entering an F3 function key or terminated by pressing the CMD key (on a 911 VDT). See table 2-1 depending on the terminal type in use.

Syntax:

```

RST
RESUME SIMULATED TASK

```

7.4.6 LSB – LIST SIMULATED BREAKPOINTS. The List Simulated Breakpoints command is used to display all active simulated breakpoints. The display is shown in figure 7-2. The first column lists the numbers assigned when the breakpoints were set. In the figure, the numbers start at one and are consecutive, because the breakpoints listed were set consecutively. The TYPE column lists letters of the “on” parameter for Assign Simulated Breakpoints command to



identify the breakpoints shown in the ASB examples. The remaining column lists the current count (the number of times the program has yet to go through the breakpoint) and the COUNT column lists the count operand entered when the breakpoints were set. The DISPLAY column lists the display operand. The FROM and THRU columns list the corresponding operand addresses. When the operands represent CRU addresses, ST register values, or XOP levels, the operands are listed as hexadecimal numbers.

Syntax:

LSB
LIST SIMULATED BREAKPOINTS

Figure 7-2 is an annotated example of the simulated breakpoints display.

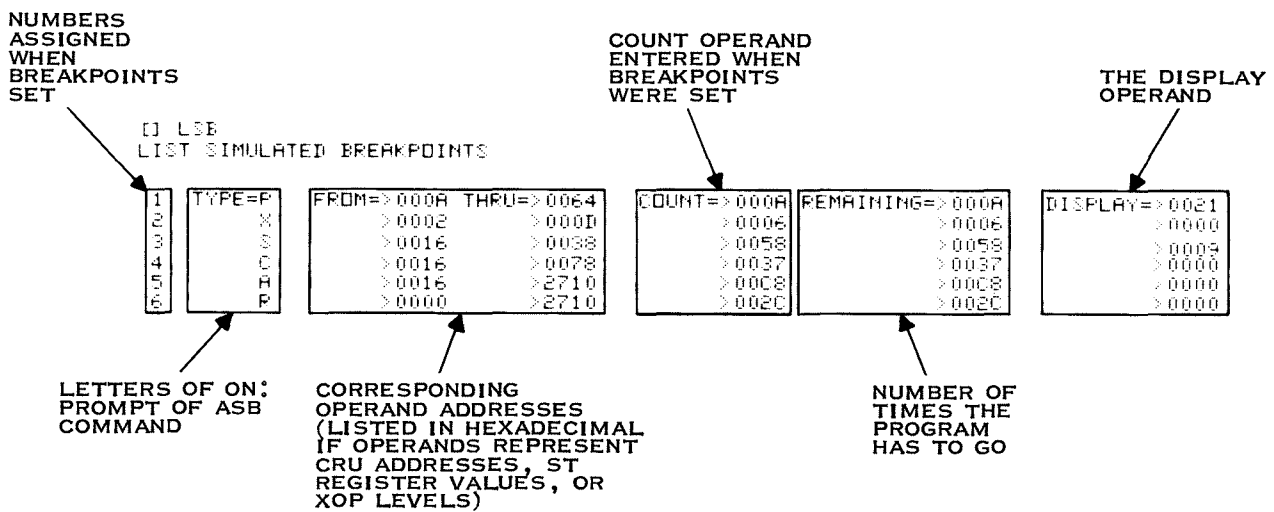


Figure 7-2. Simulated Breakpoints Display

7.4.7 QD — QUIT DEBUG. This command takes a task out of debug mode. The user has the option of keeping the task in debug mode or killing the task. If the user chooses not to kill the task, it will be left unconditionally suspended. The user may still issue any of the general SCI commands. The Resume Task (RT) or Proceed from Breakpoint (PB) commands (depending on whether the task is at a breakpoint) can be used to activate the task.

Syntax:

QD
QUIT DEBUG MODE

KILL TASK?: { <N> }
 { Y }



Example:

```
QUIT DEBUG MODE
      KILL TASK?:    YES
```

7.5 STATION DEPENDENT DISPLAYS

As mentioned previously, the displays generated by debugging SCI commands vary in format and content depending on the display device. High-speed display terminals (such as Video Display Terminals) display more information than hard copy terminals. Table 7-2 lists the display generated by several of the debug commands in varying environments.

Table 7-2. Command Displays

Command	Hard Copy Regular	Hard Copy Debug	VDT Regular	VDT Debug
AB	—	—	—	PANEL
ASB	—	—	—	BRKPT NO. + PANEL
DB	—	—	—	PANEL
DBP	—	—	—	PANEL
DSB	—	—	—	PANEL
FB	MSG OR TLF	MSG OR TLF	MSG OR TLF	MSG + PANEL OR TLF
FW	MSG OR TLF	MSG OR TLF	MSG OR TLF	MSG + PANEL OR TLF
HT	—	—	—	PANEL
LB	BRKPTS	BRKPTS	BRKPTS	BRKPTS
LLR	TLF	TLF	TLF	TLF
LM	TLF	TLF	TLF	TLF
LSB	—	SIMULATED BRKPTS	—	SIMULATED BRKPTS
MM	INTERACT	INTERACT	INTERACT	INTERACT PANEL
MWR	INTERACT	INTERACT	INTERACT	INTERACT PANEL
PB	—	—	—	PANEL
QD	—	—	—	—
RST	—	TRAP # OR 'TIMEOUT'	—	TRAP # OR 'TIMEOUT' + PANEL
RT	—	—	—	PANEL
SIR	INT REG	INT REG	PANEL	PANEL
SP	PANEL	PANEL	PANEL	PANEL
ST	—	TRAP # OR 'TIMEOUT'	—	TRAP # OR 'TIMEOUT' + PANEL
SV	VALUES	VALUES	VALUES	VALUES
SWR	WKSPC	WKSPC	PANEL	PANEL
XD	—	—	—	PANEL

TLF = contents of terminal local file

PANEL = Debug Panel, figure 7-2

INT REG = Internal registers



SECTION 8

EXAMPLE PROGRAM

8.1 RUNTHROUGH OF AN ASSEMBLY LANGUAGE EXAMPLE PROGRAM

The runthrough given in this section uses a demonstration program supplied by Texas Instruments with the DX10 operating system. The source code for the program is contained in a disk file with the pathname DS01.TI.SOURCE.TSTSDS. The program creates a disk file named DS01.TSTMSG and writes a message to the file. You will be able to edit and assemble the source, link the object, and then execute the program by installing the linked object as a task on the system program file and executing that task.

The procedures given in this section follow those given in Section 1 of this document. Since step one, design and initial coding, and step two, entering the program, have been done, the runthrough could begin with step three, assembling the source code. However, to demonstrate the use of Text Editor, begin with step two to modify the source code.

Before beginning the runthrough, power up the computer and the terminal, and log on at the terminal using the procedures described in the *Model 990 Computer DX10 Operating System Production Operation* manual.

The procedures given in this section are for use on a 911 VDT. Refer to table 2-1 for the equivalent key for other terminals.

For this example, it is convenient to create a directory file structure to simplify file references. A suggested pathname format is as follows:

Source files:	<volume name> . <programmer name> .SOURCE. <program name>
Object files:	<volume name> . <programmer name> .OBJECT. <program name>
Listing files:	<volume name> . <programmer name> .LIST. <program name>
Linked Output files:	<volume name> . <programmer name> .LINKED. <program name>
Error files:	<volume name> . <programmer name> .ERROR. <program name>
Link Edit Control files:	<volume name> . <programmer name> .CONTROL. <program name>



The volume name is optional if the system disk (DS01) is used, and it may be omitted. Use either your first or last name for the programmer name entry. The first step is to create a directory file. Enter the CFDIR command, which causes the following to be displayed:

CREATE DIRECTORY FILE

PATHNAME:
MAX ENTRIES:

DEFAULT PHYSICAL RECORD SIZE:

Enter the programmer name you selected, preceded by a period, in response to the PATHNAME: prompt. In response to the MAX ENTRIES: prompt, enter 10. In response to the DEFAULT PHYSICAL RECORD SIZE: prompt, press the RETURN key.

Repeat the command five times to create the following directories (enter these pathnames in response to the PATHNAME: prompt):

```
. <programmer name> .OBJECT
. <programmer name> .LIST
. <programmer name> .ERROR
. <programmer name> .CONTROL
. <programmer name> .LINKED
```

The MAX ENTRIES: prompt is answered with 1 each time. Since it can become tedious to enter the full file pathname every time it is required, it is convenient to assign synonyms to the directory pathnames and use them when required. Synonyms are assigned by use of the Assign Synonym (AS) SCI command. The command must be called for each synonym assignment. Enter AS, and the following is displayed:

ASSIGN SYNONYM VALUE
SYNONYM:
VALUE:

The example uses the following:

```
SYNONYM: S
VALUE: DS01.TI.SOURCE

SYNONYM: O
VALUE: DS01 . <programmer name> .OBJECT

SYNONYM: L
VALUE: DS01 . <programmer name> .LIST

SYNONYM: C
VALUE: DS01 . <programmer name> .CONTROL

SYNONYM: E
VALUE: DS01 . <programmer name> .ERROR

SYNONYM: P
VALUE: DS01 . <programmer name> .PROG
```



SYNONYM: LK
VALUE: DS01 . <programmer name> .LINKED

Call the Text Editor by entering XE. The following is then displayed:

```
INITIATE TEXT EDITOR
FILE ACCESS NAME:
```

Respond to the FILE ACCESS NAME: prompt with the following:

```
FILE ACCESS NAME: S.TSTSDS
```

The response could also be DS01.TI.SOURCE.TSTSDS. To modify the message, perform the following:

1. Enter the command mode by pressing the CMD key.
2. Key in RS to specify the Replace String command and press the RETURN key.
3. The prompts displayed and the responses are as follows:

```
REPLACE STRING
NUMBER OF OCCURRENCES: 1
START COLUMN: 37
END COLUMN: 39
STRING: OLD
CHANGE: NEW
```

Press RETURN to activate the command processor.

When the Text Editor completes the string replacement, the line containing the old string, now changed, is displayed with the cursor in column one.

4. Press the CMD key to enter the command mode.
5. Enter QE and press RETURN to call the Quit Editor command. The following display is then presented:

```
ABORT?: NO
```

6. Press the RETURN key. The following display is then presented:

```
OUTPUT FILE ACCESS NAME: S.TSTSDS
REPLACE?: NO
MOD LIST ACCESS NAME:
```

7. Press RETURN to accept the OUTPUT FILE ACCESS NAME entry. Enter Y in response to the REPLACE prompt and press the RETURN key. Press the RETURN key again to indicate that no modifications list is desired.



Once the text edit is complete, the source code must be assembled. Perform the following steps:

1. Invoke the Macro assembler by entering XMA and pressing the RETURN key. The following display is presented:

```
EXECUTE MACRO ASSEMBLER
SOURCE ACCESS NAME:
OBJECT ACCESS NAME:
LISTING ACCESS NAME:
ERROR ACCESS NAME:
OPTIONS:
MACRO LIBRARY PATHNAME:
PRINT WIDTH: 80
PAGE LENGTH: 60
```

2. Respond to the prompts in the following manner (press RETURN after each entry):

```
SOURCE ACCESS NAME: S.TSTSDS
OBJECT ACCESS NAME: 0.TSTSDS
LISTING ACCESS NAME: L.TSTSDS
ERROR ACCESS NAME: E.TSTSDS
OPTIONS: SYMT
MACRO LIBRARY PATHNAME: (Press RETURN)
PRINT WIDTH: 80
PAGE LENGTH: 60
```

3. Enter WAIT and press the RETURN key. When the assembly completes, the following message is displayed if no errors occur:

```
MACRO ASSEMBLY COMPLETE, 0000 ERRORS, 0000 WARNINGS
```

If errors are indicated, use the Show File command (SF) to view the contents of the E.TSTSDS file, which is the error file. Enter the command mode by pressing CMD.

The output of the Macro Assembler must now be linked by the following procedures:

1. First, call the Text Editor, by entering XE, to create the Link Edit control file. The following display is presented:

```
INITIATE TEXT EDITOR
FILE ACCESS NAME: S.TSTSDS
```

2. Press the TAB/SKIP key to clear the entry.
3. Press the RETURN key to insert a line.
4. Key the following and terminate each line by pressing the RETURN key:

```
TASK TSTSDS
SYMT
INCL 0.TSTSDS
END
```



5. Enter the command mode (CMD) and then enter QE to quit the editor. Respond to the prompts as follows:

```
                ABORT?:      NO
OUTPUT FILE ACCESS NAME:  C.TSTSDS
                REPLACE?:    Y
MOD.LIST ACCESS NAME:     (Press RETURN)
```

6. Call the Link Editor by keying in XLE. Respond to the prompts as follows:

```
EXECUTE LINKAGE EDITOR
CONTROL ACCESS NAME:      C.TSTSDS
LINKED OUTPUT ACCESS NAME: LK.TSTSDS
LISTING ACCESS NAME:      L.TSTSDS
PRINT WIDTH:              80
```

When the SCI command prompt appears, enter WAIT and press the RETURN key. The following message is displayed:

```
WAITING FOR BACKGROUND TASK TO COMPLETE.
```

When the Link Editor completes, the following message is displayed:

```
LINK EDITOR COMPLETED,  0 ERRORS,  0 WARNINGS:
```

Enter the command mode (CMD).

The program must now be installed as a DX10 task by use of the Install Task command. A program file is required for the Install Task command. You can either create your own program file using the Create Program File (CFPRO) command, described in the *Model 990 Computer DX10 Operating System Production Operation* manual, or use the system program file. This runthrough uses the system program file. Perform the following to install the task.

Call the command by entering IT. Respond to the prompts as follows:

```
INSTALL TASK
PROGRAM FILE OR LUNO:  0
TASK NAME:             TSTSDS
TASK ID:               0
OBJECT PATHNAME OR LUNO: LK.TSTSDS
PRIORITY:              3
DEFAULT TASK FLAGS?:  YES
ATTACHED PROCEDURES?: NO
```




The installed ID is returned by the system. Return to the command mode by pressing the CMD key.

To execute the task, use the Execute and Halt Task (XHT) command. This command activates the task but does not begin execution. The XHT command is useful when the debugging commands are to be used for the task. When XHT is entered, the following prompts are displayed. Respond as shown:

```
EXECUTE AND HALT TASK
PROGRAM FILE NAME OR LUNO:      0
TASK NAME OR ID:                TSTSDS
PARM1:                          0
PARM2:                          0
STATION ID:                     ME
```

Note that the runtime ID of the task is returned on the display. Remember the runtime ID for the next step. Return to the command mode by pressing the CMD key.

Place the task in the debug mode by entering the Execute Debug (XD) command. Respond to the following prompts as shown:

```
INITIATE DEBUG MODE
RUN ID: runtime id
SYMBOL TABLE OBJECT FILE: LK.TSTSDS
990/12 OBJECT CODE ?: (Press RETURN)
```

To begin execution of the task, use the Simulate Task (ST) command. The prompts and responses are as follows:

```
SIMULATE TASK
FOR: 1000
FROM: (Press RETURN)
TO: (Press RETURN)
```

The message TIME OUT is displayed. Return to the command mode by pressing the CMD key.

Quit the debugger by entering the QD command. The prompts and responses are as follows:

```
QUIT DEBUG
KILL TASK: YES
```

The Show File command is used to verify that the test program has created a file that contains the text message.

When SF is entered, respond to the prompt as shown:

```
SHOW FILE
FILE ACCESS NAME: DS01.TSTMSG
```

The contents of the file are displayed, and the message should read THIS IS THE NEW MESSAGE. Press the CMD key to get the initial SCI menu. Use the Text Editor, as previously described, to return the original value of the message (change NEW to OLD).



To conserve disk space, it is recommended that the Delete File command be used to delete the DS01.TSTMSG file, and that the Delete Directory command be used to delete the directory that contains the files created in this runthrough. Perform the following steps:

1. Enter DF to delete the test file. Respond as follows:

```
DELETE FILE
FILE ACCESS NAME:    DS01.TSTMSG
```

2. Enter DD to delete the directory. Respond as follows:

```
DELETE DIRECTORY
PATHNAME:            <programmer name>
LISTING ACCESS NAME: (Press RETURN)
ARE YOU SURE:        YES
```

The directory created and all files in it are now deleted. Return the terminal to the command mode by pressing the CMD key.



APPENDIX A
STANDARD DEVICE NAMES

The following list defines the standard DX10 device name format.

Device	Name ¹
Disk	DSxx
Interactive Terminals ²	STxx
Cassette Drives	CSxx
Flexible Diskette (single density)	DKxx
Magnetic Tape	MTxx
Card Readers	CRxx
Line Printers	LPxx
Communications	CMxx
AMPL Emulator	EMxx
AMPL Trace Emulator	TMxx

Notes:

¹The letters xx represent a two-digit number assigned by SYSGEN. Values are sequential within device type and are in the range 01 through 99.

²Includes VDTs, KSRs, TPDs, ASRs, (excluding cassette units), and any other interactive keyboard device.



APPENDIX B**COMMAND SUMMARY****ALIAS COMMANDS**

AA - ADD ALIAS
DA - DELETE ALIAS FROM PATHNAME

BACKGROUND/BATCH COMMANDS

BATCH - BEGIN BATCH EXECUTION
CM - CREATE MESSAGE
EBATCH - END BATCH EXECUTION
EC - ERROR COUNT
KBT - KILL BACKGROUND TASK
SBS - SHOW BACKGROUND STATUS
WAIT - WAIT FOR BACKGROUND
XB - EXECUTE BATCH SCI

BREAKPOINT COMMANDS

AB - ASSIGN BREAKPOINT
DB - DELETE BREAKPOINT
DPB - DELETE AND PROCEED FROM BREAKPOINT
LB - LIST BREAKPOINTS
PB - PROCEED FROM BREAKPOINT

COBOL COMMANDS

XCC - EXECUTE COBOL COMPILER BACKGROUND
XCCF - EXECUTE COBOL COMPILER FOREGROUND
XCP - EXECUTE COBOL PROGRAM BACKGROUND
CFPRO - CREATE PROGRAM FILE
XCPF - EXECUTE COBOL PROGRAM FOREGROUND
XLE - EXECUTE LINKAGE EDITOR
XCT - EXECUTE COBOL TASK BACKGROUND
XCTF - EXECUTE COBOL TASK FOREGROUND

CREATE COMMANDS

CF - CREATE FILE
CFDIR - CREATE DIRECTORY FILE
CFIMG - CREATE IMAGE FILE
CFKEY - CREATE KEY INDEX FILE
CFPRO - CREATE PROGRAM FILE
CFREL - CREATE RELATIVE RECORD FILE
CFSEQ - CREATE SEQUENTIAL FILE
CM - CREATE MESSAGE
CSF - CREATE SYSTEM FILES
CSK - CREATE SEQUENTIAL TO KEY
ENDKEY - END CFKEY SPECIFICATION
KEY - CFKEY KEY SPECIFICATION



COPY COMMANDS

AF	- APPEND FILE
BD	- BACKUP DIRECTORY (DISK)
CC	- COPY/CONCATENATE
CD	- COPY DIRECTORY (DISK)
CKS	- COPY KEY TO SEQUENTIAL FILES
CKSR	- COPY KIF TO SEQUENTIAL RANDOMLY
CPI	- COPY PROGRAM IMAGE
CSM	- COPY SEQUENTIAL MEDIA
CSK	- COPY SEQUENTIAL TO KEY
DCOPY	- DISK COPY/RESTORE UTILITY
IBMUTL	- CONVERT IBM DISKETTE
PF	- PRINT FILE
RD	- RESTORE DIRECTORY (DISK)
VB	- VERIFY BACKUP (DIRECTORY)
VC	- VERIFY COPY (DIRECTORY)

DBMS 990 COMMANDS

ADBF	- ASSIGN DB FILE ID
CPYFIL	- COPY FILE
DDL	- FORMAT DDL
EDBMS	- END DATA BASE MANAGER
ILDFIL	- INITIAL LOAD FILE
LSTDDL	- LIST DDL
PQUERY	- PRIMITIVE QUERY
RDBF	- RELEASE DB FILE ID
RLDFIL	- RELOAD FILE
SDBMS	- START DATA BASE MGR
SUMFIL	- SUMMARIZE FILE STATUS
UDBF	- UNLOCK DB FILE

PASSWORD UTILITIES

ADDPE	- ADD PASSWORD ENTRY
ADDPW	- ADD PASSWORD
CMPSW	- CHANGE MASTER PASSWORD
CPSW	- CHANGE PASSWORD
DELPE	- DELETE PASSWORD ENTRY
DELPW	- DELETE PASSWORD
MPSWF	- MAP PASSWORD FILE

ALIAS UTILITIES

ADDALIAS	- ADD ALIAS ENTRY
DLTALIAS	- DELETE ALIAS ENTRY
LSTALIAS	- LIST ALIAS ENTRIES
MODALIAS	- MODIFY ALIAS ENTRY



LOGGING UTILITIES

CLLOG - CLOSE LOG
OPLOG - OPEN LOG
RECOVR - RECOVER DBMS FILE

DEBUG COMMANDS

/BKPT - BREAKPOINT COMMANDS
/GDEB - GENERAL DEBUG COMMANDS
/PDEB - PASCAL DEBUG COMMANDS
/SDEB - SPECIAL DEBUG COMMANDS

DELETE COMMANDS

DD - DELETE DIRECTORY
DF - DELETE FILE

DEVICE OPERATIONS

/DISK - DISK COMMANDS
/DVICE - DEVICES
/STAT - STATUS COMMANDS
/TERM - TERMINALS

DIRECTORY COMMANDS

BD - BACKUP DIRECTORY
CD - COPY DIRECTORY
CFDIR - CREATE DIRECTORY FILE
DD - DELETE DIRECTORY
LD - LIST DIRECTORY
MD - MAP DISK (DIRECTORY)
RD - RESTORE DIRECTORY
VB - VERIFY BACKUP (DIRECTORY)
VC - VERIFY COPY (DIRECTORY)

DISK COMMANDS

CKD - CHECK DISK FOR CONSISTENCY
DCOPY - DISK COPY/RESTORE UTILITY
IBMUTL - CONVERT IBM DISKETTE
INV - INITIALIZE NEW VOLUME
IV - INSTALL VOLUME
MAD - MODIFY ABSOLUTE DISK
MADU - MODIFY ALLOCATABLE DISK UNIT
MD - MAP DISK
MVI - MODIFY VOLUME INFORMATION
RCD - RECOVER DISK
SAD - SHOW ABSOLUTE DISK
SADU - SHOW ALLOCATABLE DISK UNIT
SVS - SHOW VOLUME STATUS
UV - UNLOAD VOLUME
XCU - EXECUTE 2.2 CONVERSION UTILITY



DEVICES

CC	- COPY/CONCATENATE
HO	- HALT OUTPUT AT DEVICE
KO	- KILL OUTPUT AT DEVICE
PF	- PRINT FILE
RO	- RESUME OUTPUT AT DEVICE
SOS	- SHOW OUTPUT STATUS
RCRU	- READ CONTENTS OF SPECIFIED CRU REGISTER
WCRU	- WRITE VALUE TO SPECIFIED CRU ADDRESS

TEXT EDIT COMMANDS

CL	- COPY LINES
DL	- DELETE LINES
DS	- DELETE STRING
FS	- FIND STRING
IF	- INSERT FILE
ML	- MOVE LINES
MR	- MODIFY ROLL VALUE
MRM	- MODIFY RIGHT MARGIN
MT	- MODIFY TAB SETTINGS
QE	- QUIT EDITOR
RE	- RECOVER EDIT
RS	- REPLACE STRING
SL	- SHOW LINES
SVL	- SAVE LINES
XE	- INITIATE TEXT EDITOR
XES	- INITIATE TEXT EDITOR WITH SCALING

FILE OPERATIONS

/DIR	- DIRECTORY COMMANDS
/EDIT	- TEXT EDIT COMMANDS
/FILEC	- FILE COMMANDS
/LUNO	- LUNO
/STAT	- STATUS COMMANDS

FILE COMMANDS

/AL	- ALIAS COMMANDS
/CF	- CREATE COMMANDS
/COPY	- COPY COMMANDS
/DEL	- DELETE COMMANDS
/LUNO	- LUNO
/MOD	- MODIFY COMMANDS
/SHOW	- SHOW COMMANDS
/SYN	- SYNONYM COMMANDS

FORTRAN COMMANDS

XFC	- EXECUTE FORTRAN COMPILER BACKGROUND
XFCF	- EXECUTE FORTRAN COMPILER FOREGROUND
XLE	- EXECUTE LINKAGE EDITOR
XFT	- EXECUTE FORTRAN TASK BACKGROUND
XFTF	- EXECUTE FORTRAN TASK FOREGROUND
XF78	- EXECUTE FORTRAN-78 COMPILER BACKGROUND
XF78F	- EXECUTE FORTRAN-78 COMPILER FOREGROUND



GENERAL DEBUG COMMANDS

FB	- FIND BYTE
FW	- FIND WORD
HT	- HALT TASK
KT	- KILL TASK
LLR	- LIST LOGICAL RECORD
LM	- LIST MEMORY
LSM	- LIST SYSTEM MEMORY
MIR	- MODIFY INTERNAL REGISTERS
MM	- MODIFY MEMORY
MSM	- MODIFY SYSTEM MEMORY
MWR	- MODIFY WORKSPACE REGISTERS
RT	- RESUME TASK
SIR	- SHOW INTERNAL REGISTERS
SP	- SHOW PANEL
SV	- SHOW VALUE
SWR	- SHOW WORKSPACE REGISTERS
XHT	- EXECUTE AND HALT TASK

LANGUAGE PROCESSORS

XLE	- EXECUTE LINKAGE EDITOR
XMA	- EXECUTE MACRO ASSEMBLER

LUNO

AGL	- ASSIGN GLOBAL LUNO
AL	- ASSIGN (TERMINAL LOCAL) LUNO
BL	- BACKSPACE LOGICAL UNIT
FL	- FORWARD SPACE LOGICAL UNIT
MLP	- MODIFY LUNO PROTECTION
RAL	- RELEASE ALL LUNO'S
RGL	- RELEASE GLOBAL LUNO
RL	- RELEASE LUNO
RWL	- REWIND LOGICAL UNIT
SIS	- SHOW I/O STATUS
WEOF	- WRITE EOF TO LUNO

MISCELLANEOUS COMMANDS

CM	- CREATE MESSAGE
EC	- ERROR COUNT
IDT	- INITIALIZE DATE AND TIME
LC	- LIST COMMANDS
LDC	- LIST DEVICE CONFIGURATION
MSG	- WRITE MESSAGE TO TERMINAL
Q	- QUIT/LOGOUT
RCRU	- READ CONTENTS OF SPECIFIED CRU REGISTER
SDT	- SHOW DATE AND TIME
SMS	- SHOW MEMORY STATUS
SV	- SHOW VALUE
WAIT	- WAIT FOR BACKGROUND
WCRU	- WRITE VALUE TO SPECIFIED CRU ADDRESS
XANAL	- ANALYZE DX10 CRASH FILE
XCU	- EXECUTE 2.2 CONVERSION UTIL
XGEN	- EXECUTE GEN990 - AUTO SYSGEN



MODIFY COMMANDS

MAD	- MODIFY ABSOLUTE DISK
MADU	- MODIFY ABSOLUTE DISK UNIT
MDS	- MODIFY DEVICE STATUS
MFN	- MODIFY FILE NAME
MFP	- MODIFY FILE PROTECTION
MKL	- MODIFY KEY INDEXED FILE LOGGING
MLP	- MODIFY LUNO PROTECTION
MM	- MODIFY MEMORY
MOE	- MODIFY OVERLAY ENTRY
MPE	- MODIFY PROCEDURE ENTRY
MPI	- MODIFY PROGRAM IMAGE
MRF	- MODIFY RELATIVE TO FILE
MSM	- MODIFY SYSTEM MEMORY
MTE	- MODIFY TASK ENTRY
MVI	- MODIFY VOLUME INFORMATION
STI	- SHOW TERMINAL INFORMATION
XE	- INITIATE TEXT EDITOR
XES	- INITIATE TEXT EDITOR WITH SCALING

OVERLAY COMMANDS

DO	- DELETE OVERLAY
IO	- INSTALL OVERLAY
ISO	- INSTALL SYSTEM OVERLAY
MOE	- MODIFY OVERLAY ENTRY

PASCAL COMMANDS

ABP	- ASSIGN PASCAL BREAKPOINT
DBP	- DELETE PASCAL BREAKPOINT
DPBP	- DELETE AND PROCEED FROM PASCAL BREAKPOINT
LBP	- LIST PASCAL BREAKPOINTS
LPS	- LIST PASCAL STACK
PBP	- PROCEED FROM PASCAL BREAKPOINT
SPS	- SHOW PASCAL STACK
XTIP	- EXECUTE TI PASCAL COMPILER
XSILT	- EXECUTE TI PASCAL PARSER
XCODE	- EXECUTE TI PASCAL CODE GENERATOR
XPT	- EXECUTE TI PASCAL TASK
XPX	- EXECUTE TI PASCAL CROSS REFERENCE
XCONFIG	- EXECUTE CONFIGURATION PROCESSOR
XCONFIGI	- EXECUTE CONFIGURATION INTERACTIVELY
XNESTER	- EXECUTE SOURCE PROGRAM NESTER
XRASS	- EXECUTE REVERSE ASSEMBLER
XSPLIT	- EXECUTE SPLIT OBJECT
P\$DELETE	- DELETE TI PASCAL TEMPORARY FILES
P\$SYN	- DELETE TI PASCAL GLOBAL SYNONYMS

PROCEDURE COMMANDS

DP	- DELETE PROCEDURE
IP	- INSTALL PROCEDURE
MPE	- MODIFY PROCEDURE ENTRY



PROGRAM DEVELOPMENT

/COBOL	- COBOL
/DBMS	- DATA BASE MANAGEMENT SYSTEM
/DEBUG	- DEBUG COMMANDS
/EDIT	- TEXT EDIT COMMANDS
/LANG	- PROGRAMMING LANGUAGES
/OVER	- OVERLAY COMMANDS
/PROC	- PROCEDURE COMMANDS
/PFIL	- PROGRAM FILE COMMANDS
/SM	- SORT/MERGE
/SYN	- SYNONYM COMMANDS
/TASK	- TASK COMMANDS
/TX	- TX/DX CONVERSION
/MISC	- MISCELLANEOUS COMMANDS

PROGRAM FILE COMMANDS

BD	- BACKUP DIRECTORY
CD	- COPY DIRECTORY
CFPRO	- CREATE PROGRAM FILE
CPI	- COPY PROGRAM IMAGE
MOE	- MODIFY OVERLAY ENTRY
MPE	- MODIFY PROCEDURE ENTRY
MPF	- MAP PROGRAM FILE
MPI	- MODIFY PROGRAM IMAGE
MTE	- MODIFY TASK ENTRY
RD	- RESTORE DIRECTORY
SPI	- SHOW PROGRAM IMAGE

PROGRAMMING LANGUAGES

/BASIC	- BASIC COMMANDS
/COBOL	- COBOL COMMANDS
/FORTRN	- FORTRAN COMMANDS
/MAC	- MACRO ASSEMBLER COMMANDS
/QUERY	- QUERY COMMANDS
/RPG	- RPG COMMANDS
/TIP	- TI PASCAL COMMANDS

REMOTE TERMINAL ANSWER INCOMING CALL

ANS	- ANSWER INCOMING CALL
CALL	- CALL TERMINAL
DISC	- TERMINAL DISCONNECT
LHPC	- LIST HARDCOPY TERMINAL PORT CHARACTERISTICS
MHPC	- MODIFY HARDCOPY TERMINAL PORT CHARACTERISTICS



RPG COMMANDS

XRPGB	- BID RPG PROGRAM BACKGROUND
XRPGC	- EXECUTE REPG COMPILER BACKGROUND
XRPGCF	- EXECUTE REPG COMPILER FOREGROUND
XRPGT	- EXECUTE RPG TASK BACKGROUND
XRPGTF	- EXECUTE RPG TASK FOREGROUND

SPECIAL DEBUG COMMANDS

ASB	- ASSIGN SIMULATED BREAKPOINT
DSB	- DELETE SIMULATED BREAKPOINTS
LSB	- LIST SIMULATED BREAKPOINTS
QD	- QUIT DEBUG MODE
RST	- RESUME SIMULATED TASK
ST	- SIMULATE TASK
XD	- INITIATE DEBUG MODE

SYSGEN OPERATIONS

ALGS	- ASSEMBLE/LINK GENERATED SYSTEM
CSF	- CREATE SYSTEM FILES
IBMUTL	- CONVERT IBM DISKETTE
IGS	- INSTALL GENERATED SYSTEM
ISO	- INSTALL SYSTEM OVERLAY
LDC	- LIST DEVICE CONFIGURATION
MDS	- MODIFY DEVICE STATUS
MVI	- MODIFY VOLUME INFORMATION
PGS	- PATCH GENERATED SYSTEM
SMS	- SHOW MEMORY STATUS
TGS	- TEST GENERATED SYSTEM
XANAL	- ANALYZE DX10 CRASH FILE
XCU	- CONVERT DX10 2.2 DISK
XGEN	- GEN990-AUTO SYSGEN
XPS	- PATCH SYNONYM GENERATOR

SHOW COMMANDS

LD	- LIST DIRECTORY
LLR	- LIST LOGICAL RECORD
LM	- LIST MEMORY
LSM	- LIST SYSTEM MEMORY
MD	- MAP DISK
MKF	- MAP KEY INDEXED FILE
MPF	- MAP PROGRAM FILE
PF	- PRINT FILE
SF	- SHOW FILE
SPI	- SHOW PROGRAM IMAGE
SRF	- SHOW RELATIVE TO FILE
STI	- SHOW TERMINAL INFORMATION

SORT/MERGE

XBSM	- EXECUTE BATCH SORT/MERGE
XSM	- EXECUTE SORT/MERGE
XSMF	- EXECUTE SORT/MERGE IN FOREGROUND



DX10 OPERATION

/BGB	- BACKGROUND/BATCH COMMANDS
/COPY	- COPY COMMANDS
/DIR	- DIRECTORY COMMANDS
/DVICE	- DEVICES
/MTERM	- MEMORY TERMINAL COMMANDS
/RTERM	- REMOTE TERMINAL COMMANDS
/SM	- SORT/MERGE
/STAT	- STATUS COMMANDS
/TASK	- TASK COMMANDS
/TX	- TX/DX CONVERSION
/VOL	- VOLUME COMMANDS
/WARM	- WARMSTART COMMANDS
/MISC	- MISCELLANEOUS COMMANDS

STATUS COMMANDS

LTS	- LIST TERMINAL STATUS
MDS	- MODIFY DEVICE STATE
MTS	- MODIFY TERMINAL STATUS
SBS	- SHOW BACKGROUND STATUS
SIS	- SHOW I/O STATUS
SMS	- SHOW MEMORY STATUS
SOS	- SHOW OUTPUT STATUS
STI	- SHOW/MODIFY TERMINAL INFO
STS	- SHOW TASK STATUS
SVS	- SHOW VOLUME STATUS

SYNONYM COMMANDS

AS	- ASSIGN SYNONYM VALUE
LS	- LIST SYNONYMS
MS	- MODIFY SYNONYMS
Q\$SYN	- DELETE SYSTEM SYNONYMS
STI	- SHOW TERMINAL INFORMATION

TASK COMMANDS

AT	- ACTIVATE TASK
DT	- DELETE TASK
HT	- HALT TASK
IRT	- INSTALL REAL TIME TASK
IT	- INSTALL TASK
KT	- KILL TASK
MTE	- MODIFY TASK ENTRY
RT	- RESUME TASK
STS	- SHOW TASK STATUS
XHT	- EXECUTE AND HALT TASK
XT	- EXECUTE TASK
XTS	- EXECUTE TASK AND SUSPEND SCI

TERMINALS

LTS	- LIST TERMINAL STATUS
MDS	- MODIFY DEVICE STATUS
MTS	- MODIFY TERMINAL STATUS



TX/DX CONVERSION

DXTX	- DX10 FILE TO DISKETTE FILE
IBMUTL	- CONVERT IBM DISKETTE
TXCM	- COMPRESS DISKETTE FILE
TXCP	- CHANGE DISKETTE FILE PROTECT
TXDF	- DELETE DISKETTE FILE
TXDX	- DISKETTE FILE TO DX10 FILE
TXFD	- FORMAT DISKETTE
TXMD	- MAP DISKETTE
TXSF	- SET SYSTEM FILE

USER ID COMMANDS

AUI	- ASSIGN USER ID
DUI	- DELETE USER ID
LUI	- LIST USER ID'S
MUI	- MODIFY USER ID

VOLUME COMMANDS

CDS	- CHECK DISK STRUCTURES
CKD	- CHECK DISK FOR CONSISTENCY
INV	- INITIALIZE NEW VOLUME
IV	- INSTALL VOLUME
MVI	- MODIFY VOLUME INFORMATION
RCD	- RECOVER DISK
SVS	- SHOW VOLUME STATUS
UV	- UNLOAD VOLUME

WARMSTART COMMANDS

IDT	- INITIALIZE DATE AND TIME
INV	- INITIALIZE NEW VOLUME
IS	- INITIALIZE THE SYSTEM
ISL	- INITIATE SYSTEM LOG
IV	- INSTALL VOLUME
MDS	- MODIFY DEVICE STATUS
XANAL	- ANALYZE DX10 CRASH FILE



ALPHABETICAL INDEX

INTRODUCTION

This index lists the commands and concepts covered in the DX10 manuals. It presents the entries in alphabetical order with numerical items last. Each entry indicates where the command or concept receives major coverage in the DX10 manuals. These are the reference types used:

- Paragraphs — These references point to a discussion in the main body of this manual. They follow the same format as the paragraph numbers shown in the Table of Contents: section number (S), then top-level paragraph number (P), then lower level paragraph numbers (p and q) as needed. A reference to an entire section uses only the section number (S).

S.P.p.q
Section S

- Appendixes — These references guide the reader to information in one of the appendixes in the back of this manual. Appendix references look like paragraph references, except that appendixes are lettered instead of numbered: appendix letter (X), then paragraph numbers (P and p) as needed. A reference to an entire appendix uses only the appendix letter.

X.P.p
Appendix X

- Figures — Figure references guide the reader to figures mentioned in the List of Illustrations at the front of this manual. The reference begins with the letter F and gives the figure number: the number of the section (S) or letter of the appendix (X) that contains the figure, a dash (-), and a sequence number (n).

FS-n
FX-n

- Tables — These entries refer to tables found in this manual. They consist of the letter T followed by the table number.

TS-n
TX-n

- Manuals — These entries refer to other DX10 manuals in the set. The indexes in those books indicate the exact location of the information needed.

CONCEPTS	Concepts and Facilities Manual, Volume I
PROD OPER	Production Operation Manual, Volume II
APPL PROG	Application Programming Guide, Volume III
SYS PROG	Systems Programming Guide, Volume V
ERROR	Error Reporting and Recovery Manual, Volume VI



AA Command	PROD OPER
AB Command	7.3.1
Abort I/O on Specified LUNO	
SVC Codes >0F	APPL PROG
Absolute Memory Partitioning, Link	
Editor Supported Feature	4.1
Access Privilege	APPL PROG
Activate Suspended Task	
SVC Codes >07	APPL PROG
Activate Task Command	7.3.11
Activate Time Delay Task	
SVC Codes >0E	APPL PROG
AF Command	PROD OPER
AGL Command	PROD OPER
AL Command	PROD OPER
ALGS Command	SYS PROG
ANALZ Command	SYS PROG
Append File Command	PROD OPER
Application Program Generation	1.1
Application Programming	
Environment	CONCEPTS
AS Command	PROD OPER
ASB Command	7.4.3
ASCII Compressed and Image Format,	
Link Editor Supported Feature	4.1
Assemble and Link Generated	
System Command	SYS PROG
Assembly Language Programs	APPL PROG
Example	8.1, F3-1
Generation Runthrough	3.1
Assign Alias Command	PROD OPER
Assign Breakpoint Command	7.3.1
Assign Global LUNO Command	PROD OPER
Assign LUNO Command	PROD OPER
Assign Simulated Breakpoint Command	7.4.3
Assign Space on Program File	
SVC Codes >37	SYS PROG
Assign Synonym Command	PROD OPER
Assign User ID Command	PROD OPER
AT Command	7.3.11
AUI Command	PROD OPER
Automatic Overlay Loading, Link Editor	
Supported Feature	4.1
Back Tab, Text Editor Control Function	2.5.8
Backspace Logical Unit	
Command	PROD OPER
Backspace, Text Editor Control Function	2.5.18
Backup Directory Command	PROD OPER
Backup Disk	PROD OPER
BASIC Program Generation Runthrough	3.4
BATCH Command	PROD OPER
Batch Mode SCI	PROD OPER
Batch Stream Programming	SYS PROG
BD Command	PROD OPER
Begin Batch Execution Command	PROD OPER
Bid Task SVC Codes >1F	APPL PROG
BL Command	PROD OPER
Breakpoints, Display Simulated	F7-2
BUILD Command, GEN990	SYS PROG
Building DX10	PROD OPER
CC Command	PROD OPER
CD Command	PROD OPER
CF Command	PROD OPER
CFDIR Command	PROD OPER
CFIMG Command	PROD OPER
CFKEY Command	PROD OPER
CFPRO Command	PROD OPER
CFREL Command	PROD OPER
CFSEQ Command	PROD OPER
CHANGE Command, GEN990	SYS PROG
Change Diskette File	
Protection Command	PROD OPER
Change Priority SVC Codes >11	APPL PROG
Changing:	
Disk	PROD OPER
Menus	SYS PROG
Chassis Configuration	PROD OPER
Check Disk for Consistency	
Command	SYS PROG
CKD Command	SYS PROG
CKS Command	PROD OPER
CL Command	2.4.5
Clear Secret Synonym Command	SYS PROG
Clear to Tab, Text Editor Control	
Function	2.5.6
CM Command	PROD OPER
COBOL Program:	
Example	F3-2
Generation Runthrough	3.3
Linking, Link Editor Supported Feature	4.1
Command:	
AA	PROD OPER
AB	7.3.1
Activate Task	7.3.11
AF	PROD OPER
AGL	PROD OPER
AL	PROD OPER
ALGS	SYS PROG
ANALZ	SYS PROG
Append File	PROD OPER
AS	PROD OPER
ASB	7.4.3
Assemble and Link Generated	
System	SYS PROG
Assign Alias	PROD OPER
Assign Breakpoint	7.3.1
Assign Global LUNO	PROD OPER
Assign LUNO	PROD OPER
Assign Simulated Breakpoint	7.4.3
Assign Synonym	PROD OPER
Assign User ID	PROD OPER
AT	7.3.11
AUI	PROD OPER
Backspace Logical Unit	PROD OPER
Backup Directory	PROD OPER
BATCH	PROD OPER
BD	PROD OPER
Begin Batch Execution	PROD OPER
BL	PROD OPER
CC	PROD OPER
CD	PROD OPER
CF	PROD OPER
CFDIR	PROD OPER
CFIMG	PROD OPER
CFKEY	PROD OPER
CFPRO	PROD OPER



CFREL	PROD OPER	DS	2.4.9
CFSEQ	PROD OPER	DSB	7.4.4
Change Diskette File		DT	5.5
Protection	PROD OPER	DUI	PROD OPER
Check Disk for Consistency	SYS PROG	DXTX	PROD OPER
CKD	SYS PROG	EBATCH	PROD OPER
CKS	PROD OPER	EC	SYS PROG
CL	2.4.5	End Batch Execution	PROD OPER
Clear Secret Synonym	SYS PROG	End Key Specifications	PROD OPER
CM	PROD OPER	ENDKEY	PROD OPER
Compress Diskette File	PROD OPER	Entry Control Keys:	
Constants, Debug	7.2.4	TTY Mode	PROD OPER
Convert DX10 File to TX990		VDT Mode	PROD OPER
Diskette File	PROD OPER	Error Counting	SYS PROG
Convert TX990 Diskette File		Execute and Halt Task	6.2.3
to DX10 File	PROD OPER	Execute Batch	PROD OPER
Copy Concatenate	PROD OPER	Execute Conversion Utility	SYS PROG
Copy Directories	PROD OPER	Execute Crash Analyzer	SYS PROG
Copy Key Indexed Files to		Execute Debug	7.4.1
Sequential Files	PROD OPER	Execute GEN990	PROD OPER
Copy Line Text Editor	2.4.5	Execute Link Editor	APPL PROG
Copy Program Image	PROD OPER	Execute Macro Assembler	APPL PROG
Copy Sequential Files to Key		Execute Patch Synonym	
Indexed Files	PROD OPER	Processor	SYS PROG
Copy Sequential Media	PROD OPER	Execute System Generation	SYS PROG
CPI	PROD OPER	Execute Task	6.2.1, PROD OPER
Create File	PROD OPER	Execute Task and	
Create File Directory	PROD OPER	Suspend SCI	6.2.2, PROD OPER
Create Image File	PROD OPER	Execute Text Editor	2.1, 2.4.1
Create Key Indexed File	PROD OPER	Execute TX990 Link Editor	PROD OPER
Create Message	PROD OPER	Execute with Scaling Text Editor	2.4.2
Create Program File	PROD OPER	Expressions, Debug	7.2.4
Create Relative Record File	PROD OPER	FB	7.3.10
Create Sequential File	PROD OPER	Find Byte	7.3.10
Create System File	PROD OPER	Find String Text Editor	2.4.10
CSF	PROD OPER	Find Word	7.3.9
CSK	PROD OPER	FL	PROD OPER
CSM	PROD OPER	Format Description	PROD OPER
DA	PROD OPER	Format Diskette	PROD OPER
DB	7.3.2	Forward-Space LUNO	PROD OPER
DCOPY	PROD OPER	FS	2.4.10
DD	PROD OPER	FW	7.3.9
Definitions, SCI	PROD OPER, SYS PROG	GEN990	SYS PROG
Delete Alias	PROD OPER	BUILD	SYS PROG
Delete and Proceed from Breakpoint	7.3.4	CHANGE	SYS PROG
Delete Breakpoint	7.3.2	GENERATE	SYS PROG
Delete Directory	PROD OPER	HELP	SYS PROG
Delete Diskette	PROD OPER	LIST	SYS PROG
Delete File	PROD OPER	NEXT	SYS PROG
Delete Line Text Editor	2.4.6	SAVE	SYS PROG
Delete Overlay	5.7	STOP	SYS PROG
Delete Procedure	5.6	WHAT	SYS PROG
Delete Simulated Breakpoint	7.4.4	Halt Output	PROD OPER
Delete String Text Editor	2.4.9	Halt Task	7.3.12
Delete Task	5.5	HO	PROD OPER
Delete User ID	PROD OPER	HT	7.3.12
Description Message Returned		IBMUTL	PROD OPER
by DX10	PROD OPER	IDS	PROD OPER
DF	PROD OPER	IDT	PROD OPER
Disk Copy/Restore	PROD OPER	IF	2.4.8
Displays, Debug	7.5	IGS	SYS PROG
DL	2.4.6	Index, SCI	APPL PROG
DO	5.7	Initialize Date and Time	PROD OPER
DP	5.6	Initialize Disk Surface	PROD OPER
DPB	7.3.4	Initialize New Volume	PROD OPER



Initialize System	PROD OPER	MLP	PROD OPER
Initialize System Log	PROD OPER	MM	7.3.5
Insert File Text Editor	2.4.8	Modify Absolute Disk	7.3.24
Install Generated System	SYS PROG	Modify Allocable Disk Unit	7.3.26
Install Overlay	5.4	Modify Device State	SYS PROG
Install Procedure	5.3	Modify Disk Volume	
Install Real-Time Task	5.2.1	Information	PROD OPER
Install System Overlay	PROD OPER	Modify File Name	PROD OPER
Install Task	5.2	Modify File Protection	PROD OPER
Install Volume	PROD OPER	Modify Internal Registers	7.3.14
INV	PROD OPER	Modify Key Indexed File	
IO	5.4	Logging	PROD OPER
IP	5.3	Modify LUNO Protection	PROD OPER
IRT	5.2.1	Modify Memory	7.3.5
IS	PROD OPER	Modify Overlay Entry	5.8.3
ISL	PROD OPER	Modify Procedure Entry	5.8.2
ISO	PROD OPER	Modify Program Image	7.3.22
IT	5.2	Modify Relative to File	7.3.28
IV	PROD OPER	Modify Right Margin Text Editor	2.4.13
KBT	PROD OPER	Modify Roll Text Editor	2.4.12
KEY	PROD OPER	Modify Synonyms	PROD OPER
Key Parameters	PROD OPER	Modify System Memory	7.3.6
Kill Background Task	PROD OPER	Modify Tabs Text Editor	2.4.14
Kill Output	PROD OPER	Modify Task Entry	5.8.1
Kill Task	PROD OPER	Modify Terminal Status	PROD OPER
KO	PROD OPER	Modify the IS	PROD OPER
KT	PROD OPER	Modify User ID	PROD OPER
LB	7.3.19	Modify Workspace Registers	3.3.16
LD	PROD OPER	MOE	5.8.3
LDC	PROD OPER	Move Line Text Editor	2.4.7
Link Editor LOAD	APPL PROG	MPE	5.8.2
Link Editor TX Program File	PROD OPER	MPF	PROD OPER
List Breakpoints	7.3.19	MPI	7.3.22
List Directory	PROD OPER	MR	2.4.12
List Logical Record	7.3.29, PROD OPER	MRF	7.3.28
List Memory	7.3.7	MRM	2.4.13
List Simulated Breakpoint	7.4.6	MS	PROD OPER
List Synonym Values	PROD OPER	MSM	7.3.6
List System Device		MT	2.4.14
Configuration	PROD OPER	MTE	5.8.1
List System Memory	7.3.8	MTS	PROD OPER
List Terminal Status	PROD OPER	MUI	PROD OPER
Listing User ID	PROD OPER	MVI	PROD OPER
LLR	7.3.29, PROD OPER	MWR	3.3.16
LM	7.3.7	Patch Generated System	SYS PROG
LS	PROD OPER	PB	7.3.3
LSB	7.4.6	PF	PROD OPER
LSM	7.3.8	PGS	SYS PROG
LTS	PROD OPER	Print File	PROD OPER
LUI	PROD OPER	Procedure	SYS PROG
MAD	7.3.24	Proceed from Breakpoint	7.3.3
MADU	7.3.26	Processor	SYS PROG
Map Disk	PROD OPER	Q	PROD OPER
Map Diskette	PROD OPER	QD	7.4.7
Map Key Indexed		QE	2.1, 2.4.3
File	APPL PROG, PROD OPER	Quit Debug	7.4.7
Map Program File	PROD OPER	Quit Text Editor	2.1, 2.4.3
MD	PROD OPER	Q\$SYN	SYS PROG
MDS	SYS PROG	RAL	PROD OPER
MFN	PROD OPER	RCD	SYS PROG
MFP	PROD OPER	RCRU	PROD OPER
MIR	7.3.14	RD	PROD OPER
MKF	APPL PROG, PROD OPER	RE	2.4.4
MKL	PROD OPER	Read CRU	PROD OPER
ML	2.4.7	Recover Disk	SYS PROG



Recover Edit Text Editor	2.4.4	TXCP	PROD OPER
Release All LUNO	PROD OPER	TXDF	PROD OPER
Release Global LUNO	PROD OPER	TXDX	PROD OPER
Release LUNO	PROD OPER	TXMD	PROD OPER
Replace String Text Editor	2.4.11	TXSF	PROD OPER
Restore Directory	PROD OPER	TXXLE	PROD OPER
Resume Output	PROD OPER	Unload Volume	PROD OPER
Resume Simulated Task	7.4.5	UV	PROD OPER
Resume Task	7.3.13	VB	PROD OPER
Rewind Logical Unit	PROD OPER	VC	PROD OPER
RGL	PROD OPER	Verify Backup	PROD OPER
RL	PROD OPER	Verify Copy	PROD OPER
RO	PROD OPER	WAIT	PROD OPER
RS	2.4.11	Wait for Background	
RST	7.4.5	Termination	PROD OPER
RT	7.3.13	WCRU	PROD OPER
RWL	PROD OPER	Write CRU	PROD OPER
SAD	7.3.23	XANAL	SYS PROG
SADU	7.3.25	XB	PROD OPER
Save Line Text Editor	2.4.16	XCT	APPL PROG
SBS	PROD OPER	XCTF	APPL PROG
SDT	PROD OPER	XCU	SYS PROG
Set System File	PROD OPER	XD	7.4.1
SF	PROD OPER	XE	2.1, 2.4.1
Show Absolute Disk	7.3.23	XES	2.4.2
Show Allocable Disk Unit	7.3.25	XFC	APPL PROG
Show Background Status	PROD OPER	XFCF	APPL PROG
Show Date and Time	PROD OPER	XFT	APPL PROG
Show File	PROD OPER	XFTF	APPL PROG
Show Internal Registers	7.3.15	XGEN	PROD OPER, SYS PROG
Show I/O Status	PROD OPER	XHT	6.2.3
Show Line Text Editor	2.4.15	XLE	APPL PROG
Show Memory Map	PROD OPER	XMA	APPL PROG
Show Memory Status	PROD OPER	XPS	SYS PROG
Show Output Status	PROD OPER	XSB	APPL PROG
Show Panel	7.3.18	XT	6.2.1, PROD OPER
Show Program Image	7.3.21	XTS	6.2.2, PROD OPER
Show Relative to File	7.3.27	Commands:	
Show Task Status	PROD OPER	Controlled Task	7.4
Show Terminal Information	PROD OPER	Debug	7.3, T7-1
Show Value	7.3.20	GEN990	SYS PROG
Show Volume Status	PROD OPER	Usage, Text Editor	2.2
Show Workspace Registers	7.3.17	Selection, Text Editor	2.2
Simulate Task	7.4.2	Communications:	
SIR	7.3.15	Hardware	CONCEPTS
SIS	PROD OPER	Software	CONCEPTS
SL	2.4.15	Compatibility, DX10	SYS PROG
SMM	PROD OPER	Compiler Listing, RPG	F3-3
SMS	PROD OPER	Compiling Programs	APPL PROG
SOS	PROD OPER	Compress Diskette File	
SP	7.3.18	Command	PROD OPER
SPI	7.3.21	Configuration, Chassis	PROD OPER
SRF	7.3.27	Constants, Debug Command	7.2.4
ST	7.4.2	Control Function:	
STI	PROD OPER	Back Tab, Text Editor	2.5.8
STS	PROD OPER	Backspace, Text Editor	2.5.18
Summary	Appendix B	Clear to Tab, Text Editor	2.5.6
SV	7.3.20	Cursor Down, Text Editor	2.5.16
SVL	2.4.16	Cursor Left, Text Editor	2.5.18
SVS	PROD OPER	Cursor Right, Text Editor	2.5.17
SWR	7.3.17	Cursor Up, Text Editor	2.5.15
Test Generated System	SYS PROG	Delete Character, Text Editor	2.5.14
TGS	SYS PROG	Delete Line, Text Editor	2.5.12
TXCM	PROD OPER		



Display/Suppress Line Numbers, Text Editor	2.5.4	CSM Command	PROD OPER
Duplicate to Tab, Text Editor	2.5.5	Cursor Down, Text Editor Control Function	2.5.16
Edit/Compose Mode, Text Editor	2.5.2	Cursor Left, Text Editor Control Function	2.5.18
Enter Command Mode, Text Editor	2.5.1	Cursor Right, Text Editor Control Function	2.5.17
Erase, Text Editor	2.5.20	Cursor Up, Text Editor Control Function	2.5.15
Home, Text Editor	2.5.19	DA Command	PROD OPER
Insert Character, Text Editor	2.5.13	Data Display: Hard Copy	2.2.1
Insert Line, Text Editor	2.5.11	Text Editor	2.2
New Line, Text Editor	2.5.3	Data Entry, Text Editor	2.2
Roll Down, Text Editor	2.5.10	Date and Time SVC Codes >03	APPL PROG
Roll Up, Text Editor	2.5.9	DB Command	7.3.2
Tab, Text Editor	2.5.7	DCOPY Command	PROD OPER
Text Editor	2.5, T2-1	DD Command	PROD OPER
Control Keys, Text Editor	T2-1	Debug: Commands	7.3, T7-2
Controlled Tasks, Commands for	7.4	Constants	7.2.4
Conventions: Debug Syntax	7.2.2	Displays	7.5, T7-2
Text Editor Syntax	2.4	Expressions	7.2.4
Conversion: DX10	SYS PROG	Display	F7-1
IBM Utility	PROD OPER	Modes of	7.2
TX/DX	PROD OPER	Support	7.1
Convert Binary to Decimal SVC Codes >0A	APPL PROG	Symbolic	7.2.3
Convert Binary to Hexadecimal SVC Codes >0C	APPL PROG	Syntax Conventions	7.2.2
Convert Decimal to Binary SVC Codes >0B	APPL PROG	Unconditional Suspend Task	7.2.1
Convert DX10 File to TX990 Diskette File Command	PROD OPER	Definitions, SCI Command	PROD OPER, SYS PROG
Convert Hexadecimal to Binary SVC Codes >0D	APPL PROG	Delete Alias Command	PROD OPER
Convert TX990 Diskette File to DX10 File Command	PROD OPER	Delete and Proceed from Breakpoint Command	7.3.4
Copy Concatenate Command	PROD OPER	Delete Breakpoint Command	7.3.2
Copy Directories Command	PROD OPER	Delete Character, Text Editor Control Function	2.5.14
Copy Key Indexed Files to Sequential Files Command	PROD OPER	Delete Directory Command	PROD OPER
Copy Line Text Editor Command	2.4.5	Delete Diskette Command	PROD OPER
Copy Program Image Command	PROD OPER	Delete File Command	PROD OPER
Copy Sequential Files to Key Indexed Files Command	PROD OPER	Delete Line, Text Editor Command	2.4.6
Copy Sequential Media Command	PROD OPER	Control Function	2.5.12
CPI Command	PROD OPER	Delete Overlay: Command	5.7
Crash: Analysis	ERROR, SYS PROG	SVC Codes >2A	SYS PROG
Codes	ERROR	Delete Procedure: Command	5.6
Front Panel	ERROR	SVC Codes >29	SYS PROG
Create File Command	PROD OPER	Delete Simulated Breakpoint Command	7.4.4
Create File Directory Command	PROD OPER	Delete String Text Editor Command	2.4.9
Create Image File Command	PROD OPER	Delete Task: Command	5.5
Create Key Indexed File Command	PROD OPER	SVC Codes >28	SYS PROG
Create Message Command	PROD OPER	Delete User ID Command	PROD OPER
Create New Text Editor File	2.3.1	Description Message Returned by DX10 Command	PROD OPER
Create Program File Command	PROD OPER	Device: Functions	CONCEPTS
Create Relative Record File Command	PROD OPER	I/O, Programming	APPL PROG
Create Sequential File Command	PROD OPER	Names	PROD OPER
Create System File Command	PROD OPER	Standard	Appendix A
CSF Command	PROD OPER	Service Routines DSR	SYS PROG
CSK Command	PROD OPER	DF Command	PROD OPER



Direct Disk I/O Read by ADU SVC Codes >00>9	SYS PROG	Loading	PROD OPER
Direct Disk I/O Read by Track SVC Codes >00>A	SYS PROG	Operation	PROD OPER
Direct Disk I/O Read Format SVC Codes >00>5 and >F	SYS PROG	Text Editor	2.1
Direct Disk I/O Store Registers SVC Codes >00>E	SYS PROG	EBATCH Command	PROD OPER
Direct Disk I/O Write by ADU SVC Codes >00>B	SYS PROG	EC Command	SYS PROG
Direct Disk I/O Write by Track SVC Codes >00>C	SYS PROG	Edit/Compose Mode, Text Editor Control Function	2.5.2
Direct Disk I/O Write Format SVC Codes >00>8	SYS PROG	Edit Control, Text Editor	2.2
Direct Diskette I/O Read ASCII SVC Codes >00>9	SYS PROG	Edit Existing Text Editor File	2.3.2
Direct Diskette I/O Read Deleted Sector SVC Codes >00>11	SYS PROG	End-Action Status SVC Codes >2F	APPL PROG
Direct Diskette I/O Read Direct SVC Codes >00>A	SYS PROG	End Batch Execution Command ...	PROD OPER
Direct Diskette I/O Read Format SVC Codes >00>5 and >F	SYS PROG	End Key Specifications Command	PROD OPER
Direct Diskette I/O Write ASCII SVC Codes >00>B	SYS PROG	ENDKEY Command	PROD OPER
Direct Diskette I/O Write Deleted Sector SVC Codes >00>10	SYS PROG	End-of-Program SVC Codes >16 ..	APPL PROG
Direct Diskette I/O Write Direct SVC Codes >00>C	SYS PROG	End-of-Task SVC Codes >04	APPL PROG
Direct Diskette I/O Write Format SVC Codes >00>8	SYS PROG	Enter Command Mode, Text Editor Control Function	2.5.1
Directory Creating and Deleting ...	PROD OPER	Erase, Text Editor Control Function	2.5.20
Disk:		Error Control	CONCEPTS
Backup	PROD OPER	Error Counting Command	SYS PROG
Changing	PROD OPER	Error Messages	ERROR
Copy/Restore Command	PROD OPER	Estimating, Memory	SYS PROG
Formats	CONCEPTS	Example:	
Hardware	CONCEPTS	Assembly Language Program	8.1, F3-1
Management	CONCEPTS	COBOL Program	F3-2
Manager SVC Codes >22	SYS PROG	Pascal	F3-7
Pack Installation	PROD OPER	RPG	F3-5
Protection	CONCEPTS	Input File	F3-4
Display:		Text Editor	F2-1, F2-2, F2-3
Debug	F7-1	Execute and Halt Task Command	6.2.3
Command	7.5, T7-2	Execute Batch Command	PROD OPER
Simulated Breakpoints	F7-2	Execute Conversion Utility Command	SYS PROG
Display/Suppress Line Numbers, Text Editor Control Function	2.5.4	Execute Crash Analyzer Command ...	SYS PROG
DL Command	2.4.6	Execute Debug Command	7.4.1
DO Command	5.7	Execute GEN990 Command	PROD OPER
Do Not Suspend SVC Codes >09 ...	APPL PROG	Execute Link Editor Command ...	APPL PROG
DP Command	5.6	Execute Macro Assembler Command	APPL PROG
DPB Command	7.3.4	Execute Patch Synonym Processor Command	SYS PROG
DS Command	2.4.9	Execute System Generation Command ...	6.2.2, PROD OPER, SYS PROG
DSB Command	7.4.4	Execute Task: and Suspend SCI Command	6.2.2, PROD OPER
DSR, Device Service Routines	SYS PROG	Command	6.2.1, PROD OPER
DT Command	5.5	SVC Codes >2B	APPL PROG
DUI Command	PROD OPER	Execute Text Editor Command	2.1, 2.4.1
Duplicate to Tab, Text Editor Control Function	2.5.5	Execute TX990 Link Editor Command	PROD OPER
DXTX Command	PROD OPER	Execute with Scaling Text Editor Command	2.4.2
DX10:		Executing Programs	APPL PROG
Building	PROD OPER	Execution Instructions, Language	T6-1
Compatibility	SYS PROG	Expressions, Debug Command	7.2.4
Conversion	SYS PROG	FB Command	7.3.10
		Features:	
		Hardware	CONCEPTS
		Terminal	CONCEPTS, PROD OPER



File:

Create New Text Editor 2.3.1
 Creating and Deleting PROD OPER
 Edit Existing Text Editor 2.3.2
 Example, RPG Input F3-4
 Features CONCEPTS
 I/O, Programming APPL PROG
 Management PROD OPER
 Programming APPL PROG
 Standard SYS PROG
 Types CONCEPTS

File Operation:

Backward Space SVC
 Codes >00>07 APPL PROG
 Close EOF SVC
 Codes >00>02 APPL PROG
 Close SVC Codes >00>01 APPL PROG
 Close Unload SVC Codes
 >00>04 APPL PROG
 Forward Space SVC Codes
 >00>06 APPL PROG
 Modify Access Privileges SVC
 Codes >00>11 APPL PROG
 Open Extend SVC Codes
 >00>12 APPL PROG
 Open Rewind SVC Codes
 >00>03 APPL PROG
 Open SVC Codes >00>00 APPL PROG
 Read ASCII SVC Codes
 >00>09 APPL PROG
 Read Direct SVC Codes
 >00>0A APPL PROG
 Read File Characteristics SVC
 Codes >00>05 APPL PROG
 Rewind SVC Codes >00>0E APPL PROG
 Rewrite SVC Codes >00>10 APPL PROG
 Unload SVC Codes >00>0F APPL PROG
 Unlock SVC Codes >00>4A APPL PROG
 Write ASCII SVC Codes
 >00>0B APPL PROG
 Write Direct SVC Codes
 >00>0C APPL PROG
 Write EOF SVC Codes
 >00>0D APPL PROG

File Utility Operation:

Add Alias SVC Codes >00>9A .. APPL PROG
 Assign LUNO SVC Codes
 >00>91 APPL PROG
 Create File SVC Codes >00>90 .. APPL PROG
 Define Write Mode SVC Codes
 >00>9C APPL PROG
 Delete Alias SVC Codes
 >00>9B APPL PROG
 Delete File SVC Codes >00>92 .. APPL PROG
 Delete Protect File SVC Codes
 >00>98 APPL PROG
 Release LUNO SVC Codes
 >00>93 APPL PROG
 Rename File SVC Codes
 >00>95 APPL PROG
 Unprotect File SVC Codes
 >00>96 APPL PROG
 Verify Pathname SVC Codes
 >00>99 APPL PROG

Write Protect File SVC Codes
 >00>97 APPL PROG
 Find Byte Command 7.3.10
 Find String Text Editor Command 2.4.10
 Find Word Command 7.3.9
 FL Command PROD OPER
 Format Description Command PROD OPER
 Format Diskette Command PROD OPER
 FORTRAN Program:
 Generation Runthrough 3.2
 Linking, Link Editor Supported Feature ... 4.1
 Forward Space LUNO Command .. PROD OPER
 FS Command 2.4.10
 FW Command 7.3.9

GENERATE Command, GEN990 ... SYS PROG
 Generation Runthrough:
 Assembly Language Program 3.1
 BASIC Program 3.4
 COBOL Program 3.3
 FORTRAN Program 3.2
 Pascal Program 3.6
 RPG Program 3.5

GEN990:

BUILD Command SYS PROG
 CHANGE Command SYS PROG
 Command SYS PROG
 Commands SYS PROG
 GENERATE Command SYS PROG
 HELP Command SYS PROG
 LIST Command SYS PROG
 NEXT Command SYS PROG
 SAVE Command SYS PROG
 STOP Command SYS PROG
 WHAT Command SYS PROG

Get Common Data Address SVC

Codes >10 APPL PROG

Get Event Key:

SVC Codes >30 APPL PROG
 SVC Codes >39 APPL PROG

Get Memory SVC Codes >12 APPL PROG

Get Parameters SVC Codes >17 ... APPL PROG

Get System Pointer Table Address

SVC Codes >32 SYS PROG

Getdata SVC Codes >1D APPL PROG

Global LUNO PROD OPER, SYS PROG

Glossary CONCEPTS

Halt Output Command PROD OPER

Halt Task Command 7.3.12

Hard Copy:

Data Display 2.2.1
 Terminal Operation 2.2.1

Hardware:

Communications CONCEPTS
 Features CONCEPTS
 Required CONCEPTS
 Supported CONCEPTS
 Terminals CONCEPTS

HELP Command, GEN990 SYS PROG

HO Command PROD OPER

Home, Text Editor Control Function 2.5.19

HT Command 7.3.12



IBM Conversion Utility	PROD OPER
IBMUTL Command	PROD OPER
IDS Command	PROD OPER
IDT Command	PROD OPER
IF Command	2.4.8
IGS Command	SYS PROG
Index, SCI Command	APPL PROG
Initialize Date and Time:	
Command	PROD OPER
SVC Codes >3B	SYS PROG
Initialize Disk Surface Command ..	PROD OPER
Initialize New Volume:	
Command	PROD OPER
SVC Codes >38	SYS PROG
Initialize System Command	PROD OPER
Initialize System Log Command ..	PROD OPER
Input File Example, RPG	F3-4
Insert Character, Text Editor Control	
Function	2.5.13
Insert File Text Editor Command ..	2.4.8
Insert Line, Text Editor Control Function ..	2.5.11
Install Disk Volume SVC Codes >20 ..	SYS PROG
Install Generated System Command ..	SYS PROG
Install Overlay:	
Command	5.4
SVC Codes >27	SYS PROG
Install Procedure:	
Command	5.3
SVC Codes >26	SYS PROG
Install Real-Time Task:	
Command	5.2.1
SVC Codes >25	SYS PROG
Install System Overlay Command ..	PROD OPER
Install Task:	
Command	5.2
SVC Codes >25	SYS PROG
Install Volume Command	PROD OPER
International Considerations	APPL PROG
INV Command	PROD OPER
IO Command	5.4
I/O Backward Space SVC Codes	
>00>07	APPL PROG
I/O Close SVC Codes >00>01	APPL PROG
I/O Close Unload SVC Codes	
>00>04	APPL PROG
I/O Close with EOF SVC Codes	
>00>02	APPL PROG
I/O Forward Space SVC Codes	
>00>06	APPL PROG
I/O, Logical	CONCEPTS, PROD OPER
I/O Open Rewind SVC Codes	
>00>03	APPL PROG
I/O Open SVC Codes	
>00>00	APPL PROG
I/O Read ASCII SVC Codes	
>00>09	APPL PROG
I/O Read Device Status SVC	
Codes >00>05	APPL PROG
I/O Read Direct SVC Codes	
>00>0A	APPL PROG
I/O Rewind SVC Codes	
>00>0E	APPL PROG
I/O Unload SVC Codes	
>00>0F	APPL PROG
I/O Write ASCII SVC Codes	
>00>0B	APPL PROG
I/O Write Direct SVC Codes	
>00>0C	APPL PROG
I/O Write EOF SVC Codes	
>00>0D	APPL PROG
IP Command	5.3
IRT Command	5.2.1
IS Command	PROD OPER
ISL Command	PROD OPER
ISO Command	PROD OPER
IT Command	5.2
IV Command	PROD OPER
KBT Command	PROD OPER
KEY Command	PROD OPER
Key Parameters Command	PROD OPER
Keyboard Code Table	APPL PROG
Keyboard Equivalents	PROD OPER
Keys:	
Text Editor Control	T2-1
TTY Mode, Command Entry	
Control	PROD OPER
VDT Mode, Command Entry	
Control	PROD OPER
Kill Background Task Command ..	PROD OPER
Kill Output Command	PROD OPER
Kill Task:	
Command	PROD OPER
SVC Codes >33	SYS PROG
Kit, Software	PROD OPER
KO Command	PROD OPER
KT Command	PROD OPER
Language:	
Execution Instructions	T6-1
SCI	SYS PROG
Programming	APPL PROG, CONCEPTS
LB Command	7.3.19
LD Command	PROD OPER
LDC Command	PROD OPER
Link Editor:	
LOAD Command	APPL PROG
Operation	4.2
Supported Feature:	
Absolute Memory Partitioning	4.1
ASCII Compressed and Image Format ..	4.1
Automatic Overlay Loading	4.1
COBOL Program Linking	4.1
FORTRAN Program Linking	4.1
Random Libraries	4.1
Sequential Libraries	4.1
TX Program File Command	PROD OPER
Linking Programs	APPL PROG
List Breakpoints Command	7.3.19
LIST Command, GEN990	SYS PROG
List Directory Command	PROD OPER
List Logical Record	
Command	7.3.29, PROD OPER
List Memory Command	7.3.7
List Simulated Breakpoint Command ..	7.4.6
List Synonym Values Command ..	PROD OPER
List System Device Configuration	
Command	PROD OPER



List System Memory Command 7.3.8
 List Terminal Status Command PROD OPER
 Listing, RPG Compiler F3-3
 Listing User ID Command PROD OPER
 LLR Command 7.3.29, PROD OPER
 LM Command 7.3.7
 Load Overlay SVC Codes >14 APPL PROG
 Loading DX10 PROD OPER
 Log on/Log off, Terminal PROD OPER
 Log:
 SVC Codes >21, System APPL PROG
 System CONCEPTS, PROD OPER
 Logical I/O CONCEPTS, PROD OPER
 LS Command PROD OPER
 LSB Command 7.4.6
 LSM Command 7.3.8
 LTS Command PROD OPER
 LUI Command PROD OPER
 LUNO CONCEPTS, PROD OPER
 Global PROD OPER, SYS PROG

 MAD Command 7.3.24
 MADU Command 7.3.26
 Map Disk Command PROD OPER
 Map Diskette Command PROD OPER
 Map Key Indexed File
 Command APPL PROG, PROD OPER
 Map Program File Command PROD OPER
 Map Program Name to ID SVC
 Codes >31 APPL PROG
 MD Command PROD OPER
 MDS Command SYS PROG
 Memory Estimating SYS PROG
 Memory Management CONCEPTS
 Menus PROD OPER, SYS PROG
 Changing SYS PROG
 Messages, Error ERROR
 MFN Command PROD OPER
 MFP Command PROD OPER
 MIR Command 7.3.14
 MKF Command APPL PROG, PROD OPER
 MKL Command PROD OPER
 ML Command 2.4.7
 MLP Command PROD OPER
 MM Command 7.3.5
 Modes of Debug 7.2
 Modify Absolute Disk Command 7.3.24
 Modify Allocable Disk Unit Command ... 7.3.26
 Modify Device State Command SYS PROG
 Modify Disk Volume Information
 Command PROD OPER
 Modify File Name Command PROD OPER
 Modify File Protection
 Command PROD OPER
 Modify Internal Registers Command 7.3.14
 Modify Key Indexed File Logging
 Command PROD OPER
 Modify LUNO Protection
 Command PROD OPER
 Modify Memory Command 7.3.5
 Modify Overlay Entry Command 5.8.3
 Modify Procedure Entry Command 5.8.2
 Modify Program Image Command 7.3.22
 Modify Relative to File Command 7.3.28

Modify Right Margin Text Editor
 Command 2.4.13
 Modify Roll Text Editor Command 2.4.12
 Modify Synonyms Command PROD OPER
 Modify System Memory Command 7.3.6
 Modify Tabs Text Editor Command 2.4.14
 Modify Task Entry Command 5.8.1
 Modify Terminal Status
 Command PROD OPER
 Modify the IS Command PROD OPER
 Modify User ID Command PROD OPER
 Modify Workspace Registers Command ... 3.3.16
 MOE Command 5.8.3
 Move Line Text Editor Command 2.4.7
 MPE Command 5.8.2
 MPF Command PROD OPER
 MPI Command 7.3.22
 MR Command 2.4.12
 MRF Command 7.3.28
 MRM Command 2.4.13
 MS Command PROD OPER
 MSM Command 7.3.6
 MT Command 2.4.14
 MTE Command 5.8.1
 MTS Command PROD OPER
 MUI Command PROD OPER
 MVI Command PROD OPER
 MWR Command 3.3.16

New Line, Text Editor Control Function ... 2.5.3
 NEXT Command, GEN990 SYS PROG

Operation:
 DX10 PROD OPER
 Hard-Copy Terminal 2.2.1
 Link Editor 4.2
 Terminal PROD OPER
 Types of Text Editor 2.2
 Video Display Terminal 2.2.2
 Overlays APPL PROG
 IDs SYS PROG

Pascal:
 Example F3-7
 Program Generation Runthrough 3.6
 Source Example F3-6
 Patch Generated System Command .. SYS PROG
 Patching DX10 SYS PROG
 PB Command 7.3.3
 PF Command PROD OPER
 PGS Command SYS PROG
 Poll Status of Task SVC
 Codes >35 APPL PROG
 Print File Command PROD OPER
 Priority, Task APPL PROG
 Privilege, Access APPL PROG
 Privileged SVC SYS PROG
 Procedure:
 Command SYS PROG
 Definitions, SCI ... PROD OPER, SYS PROG
 IDs SYS PROG
 Index, SCI PROD OPER
 Programming SCI SYS PROG
 Proceed from Breakpoint Command 7.3.3



Processor:

- Command SYS PROG
- SCI SYS PROG
- Subroutines SYS PROG

Program:

- Assembling APPL PROG
- Compiling APPL PROG
- Example, Assembly Language 8.1
- Executing APPL PROG, PROD OPER
- Generation, Application 1.1
- Linking APPL PROG
- Management CONCEPTS
- Writing APPL PROG

Programming:

- Batch Stream SYS PROG
- Device I/O APPL PROG
- File APPL PROG
- File I/O APPL PROG
- Languages APPL PROG, CONCEPTS
- SCI Procedure SYS PROG
- Tasks APPL PROG
- Terminal APPL PROG
- Putdata SVC Codes >1C APPL PROG

Q Command PROD OPER

QD Command 7.4.7

QE Command 2.1, 2.4.3

Quit Debug Command 7.4.7

Quit Text Editor Command 2.1, 2.4.3

Q\$SYN Command SYS PROG

RAL Command PROD OPER

Random Libraries, Link Editor

- Supported Feature 4.1

RCD Command SYS PROG

RCRU Command PROD OPER

RD Command PROD OPER

RE Command 2.4.4

Read CRU Command PROD OPER

Read/Write Task SVC Codes >2D SYS PROG

Read/Write TSB SVC Codes >2C SYS PROG

Recover Disk Command SYS PROG

Recover Edit Text Editor Command 2.4.4

Release All LUNO Command PROD OPER

Release Global LUNO Command PROD OPER

Release LUNO Command PROD OPER

Release Memory SVC Codes >13 APPL PROG

Replace String Text Editor Command 2.4.11

Reset End-Action SVC

- Codes >3E APPL PROG

Restore Directory Command PROD OPER

Resume Output Command PROD OPER

Resume Simulated Task Command 7.4.5

Resume Task Command 7.3.13

Return Common Data Address

- SVC Codes >1 B APPL PROG

Rewind Logical Unit Command PROD OPER

RGL Command PROD OPER

RL Command PROD OPER

RO Command PROD OPER

Roll Down, Text Editor Control

- Function 2.5.10

Roll Up, Text Editor Control

- Function 2.5.9

RPG:

- Compiler Listing F3-3
- Example F3-5
- Input File Example F3-4
- Program Generation Runthrough 3.5

RSC Command 2.4.11

RST Command 7.4.5

RT Command 7.3.13

RWL Command PROD OPER

SAD Command 7.3.23

SADU Command 7.3.25

SAVE Command, GEN990 SYS PROG

Save Line Text Editor Command 2.4.16

SBS Command PROD OPER

SCI:

- Batch Mode PROD OPER
- Characteristics PROD OPER
- Command Definitions PROD OPER, SYS PROG
- Command Index APPL PROG, PROD OPER
- Language SYS PROG
- Overview CONCEPTS
- Procedure Definitions PROD OPER, SYS PROG
- Procedure Programming SYS PROG
- Processor SYS PROG
- Subroutines SYS PROG

SDT Command PROD OPER

Self Identification SVC Codes

- >2E APPL PROG

Sequential Libraries, Link Editor

- Supported Feature 4.1

Set System File Command PROD OPER

SF Command PROD OPER

Show Absolute Disk Command 7.3.23

Show Allocable Disk Unit Command 7.3.25

Show Background Status

- Command PROD OPER

Show Date and Time Command PROD OPER

Show File Command PROD OPER

Show Internal Registers Command 7.3.15

Show I/O Status Command PROD OPER

Show Line Text Editor Command 2.4.15

Show Memory Map Command PROD OPER

Show Memory Status Command PROD OPER

Show Output Status Command PROD OPER

Show Panel Command 7.3.18

Show Program Image Command 7.3.21

Show Relative to File Command 7.3.27

Show Task Status Command PROD OPER

Show Terminal Information

- Command PROD OPER

Show Value Command 7.3.20

Show Volume Status Command PROD OPER

Show Workspace Registers Command 7.3.17

Simulate Task Command 7.4.2

Simulated Breakpoints, Display F7-2

SIR Command 7.3.15

SIS Command PROD OPER

SL Command 2.4.15

SMM Command PROD OPER

SMS Command PROD OPER



Software:	
Communications	CONCEPTS
Kit	PROD OPER
SOS Command	PROD OPER
Source Example, Pascal	F3-6
SP Command	7.3.18
SPI Command	7.3.21
SRF Command	7.3.27
ST Command	7.4.2
Standard:	
Device Names	Appendix A
Files	SYS PROG
STI Command	PROD OPER
STOP Command, GEN990	SYS PROG
STS Command	PROD OPER
Subroutines, Processor	SYS PROG
Summary, Command	Appendix B
Support, Debugging	7.1
Supported Feature:	
Absolute Memory Partitioning, Link	
Editor	4.1
ASCII Compressed and Image Format,	
Link Editor	4.1
Automatic Overlay Loading, Link Editor ...	4.1
COBOL Program Linking, Link Editor ...	4.1
FORTRAN Program Linking, Link	
Editor	4.1
Random Libraries, Link Editor	4.1
Sequential Libraries, Link Editor	4.1
Suspend Awaiting Queue Input	
SVC Codes >24	SYS PROG
SV Command	7.3.20
SVC:	
Definition	APPL PROG, SYS PROG
Memory Control	APPL PROG
Miscellaneous	APPL PROG
Overview	CONCEPTS
Privileged	SYS PROG
Program Control	APPL PROG
SVC Codes	SYS PROG
>00>00:	
File Operation Open	APPL PROG
I/O Open	APPL PROG
>00>01:	
File Operation Close	APPL PROG
I/O Close	APPL PROG
>00>02:	
File Operation Close EOF	APPL PROG
I/O Close with EOF	APPL PROG
>00>03:	
File Operation Open Rewind ..	APPL PROG
I/O Open Rewind	APPL PROG
>00>04:	
File Operation Close Unload ..	APPL PROG
I/O Close Unload	APPL PROG
>00>05:	
File Operation Read File	
Characteristics	APPL PROG
I/O Read Device Status	APPL PROG
>00>06:	
File Operation Forward	
Space	APPL PROG
I/O Forward Space	APPL PROG
>00>07:	
File Operation Backward	
Space	APPL PROG
I/O Backward Space	APPL PROG
>00>09:	
File Operation Read ASCII ...	APPL PROG
I/O Read ASCII	APPL PROG
>00>0A:	
File Operation Read Direct ...	APPL PROG
I/O Read Direct	APPL PROG
>00>0B:	
File Operation Write ASCII ...	APPL PROG
I/O Write ASCII	APPL PROG
>00>0C:	
File Operation Write Direct ...	APPL PROG
I/O Write Direct	APPL PROG
>00>0D:	
File Operation Write EOF ...	APPL PROG
I/O Write EOF	APPL PROG
>00>0E:	
File Operation Rewind	APPL PROG
I/O Rewind	APPL PROG
>00>0F:	
File Operation Unload	APPL PROG
I/O Unload	APPL PROG
>00>10:	
Direct Diskette I/O Write	
Delete Sector	SYS PROG
File Operation Rewrite	APPL PROG
>00>11:	
Direct Diskette I/O Read	
Deleted Sector	SYS PROG
File Operation Modify	
Access Privileges	APPL PROG
>00>12, File Operation	
Open Extend	APPL PROG
>00>4A, File Operation	
Unlock	APPL PROG
>00>90, File Utility Operation	
Create File	APPL PROG
>00>91, File Utility Operation	
Assign LUNO	APPL PROG
>00>92, File Utility Operation	
Delete File	APPL PROG
>00>93, File Utility Operation	
Release LUNO	APPL PROG
>00>95, File Utility Operation	
Rename File	APPL PROG
>00>96, File Utility Operation	
Unprotect File	APPL PROG
>00>97, File Utility Operation	
Write Protect File	APPL PROG
>00>98, File Utility Operation	
Delete Protect File	APPL PROG
>00>99, File Utility Operation	
Verify Pathname	APPL PROG
>00>9A, File Utility Operation	
Add Alias	APPL PROG
>00>9B, File Utility Operation	
Delete Alias	APPL PROG
>00>9C, File Utility Operation	
Define Write Mode	APPL PROG



>00>5:	>24, Suspend Awaiting Queue
Direct Disk I/O Read Format ... SYS PROG	Input SYS PROG
Direct Diskette I/O Read	>25:
Format SYS PROG	Install Real-Time Task SYS PROG
>00>8:	Install Task SYS PROG
Direct Disk I/O Write Format ... SYS PROG	>26, Install Procedure SYS PROG
Direct Diskette I/O Write	>27, Install Overlay SYS PROG
Format SYS PROG	>28, Delete Task SYS PROG
>00>9:	>29, Delete Procedure SYS PROG
Direct Disk I/O Read by ADU ... SYS PROG	>2A, Delete Overlay SYS PROG
Direct Diskette I/O Read	>2B, Execute Task APPL PROG
ASCII SYS PROG	>2C, Read/Write TSB SYS PROG
>00>A:	>2D, Read/Write Task SYS PROG
Direct Disk I/O Read by Track .. SYS PROG	>2E, Self Identification APPL PROG
Direct Diskette I/O Read	>2F, End-Action Status APPL PROG
Direct SYS PROG	>30, Get Event Key APPL PROG
>00>B:	>31, Map Program Name to ID .. APPL PROG
Direct Disk I/O Write by ADU .. SYS PROG	>32, Get System Pointer Table
Direct Diskette I/O Write	Address SYS PROG
ASCII SYS PROG	>33, Kill Task SYS PROG
>00>C:	>34, Unload Disk Volume SYS PROG
Direct Disk I/O Write by Track .. SYS PROG	>35, Poll Status of Task APPL PROG
Direct Diskette I/O Write	>36, Wait on Multiple Initiate
Direct SYS PROG	I/O APPL PROG
>00>E, Direct Disk I/O Store	>37, Assign Space on Program
Registers SYS PROG	File SYS PROG
>00>F:	>38, Initialize New Volume SYS PROG
Direct Disk I/O Read Format ... SYS PROG	>39, Get Event Key APPL PROG
Direct Diskette I/O Read	>3B, Initialize Date and Time ... SYS PROG
Format SYS PROG	>3E, Reset End Action APPL PROG
>01, Wait for I/O APPL PROG	SVL Command 2.4.16
>02, Time Delay APPL PROG	SVS Command PROD OPER
>03, Date and Time APPL PROG	SWR Command 7.3.17
>04, End-of-Task APPL PROG	Symbolic Debug 7.2.3
>06, Unconditional Wait APPL PROG	Syntax Conventions:
>07, Activate Suspended Task ... APPL PROG	Debug 7.2.2
>09, Do Not Suspend APPL PROG	Text Editor 2.4
>0A, Convert Binary to	System:
Decimal APPL PROG	Generation SYS PROG
>0B, Convert Decimal to	Overview CONCEPTS
Binary APPL PROG	Log CONCEPTS, PROD OPER
>0C, Convert Binary to	Example ERROR
Hexadecimal APPL PROG	SVC Codes >21 APPL PROG
>0D, Convert Hexadecimal to	
Binary APPL PROG	Tab, Text Editor Control Function 2.5.7
>0E, Activate Time Delay Task .. APPL PROG	TASK:
>0F, Abort I/O on Specified	IDs SYS PROG
LUNO APPL PROG	Priority APPL PROG
>10, Get Common Data	State Codes APPL PROG, PROD OPER
Address APPL PROG	Programming APPL PROG
>11, Change Priority APPL PROG	Terminal:
>12, Get Memory APPL PROG	Access PROD OPER
>13, Release Memory APPL PROG	Extended Support CONCEPTS
>14, Load Overlay APPL PROG	Features CONCEPTS, PROD OPER
>16, End-of-Program APPL PROG	Functions PROD OPER
>17, Get Parameters APPL PROG	Keyboard Code Table APPL PROG
>1B, Return Common Data	Keyboard Equivalents PROD OPER
Address APPL PROG	Log on/Log off PROD OPER
>1C, Putdata APPL PROG	Operation PROD OPER
>1D, Getdata APPL PROG	Hard Copy 2.2.1
>1F, Bid Task APPL PROG	Video Display 2.2.2
>20, Install Disk Volume SYS PROG	Programming APPL PROG
>21, System Log APPL PROG	Test Generated System Command .. SYS PROG
>22, Disk Manager SYS PROG	



Text Editor:

Command:

- Copy Line 2.4.5
- Delete Line 2.4.6
- Delete String 2.4.9
- Execute 2.1, 2.4.1
- Execute with Scaling 2.4.2
- Find String 2.4.10
- Insert File 2.4.8
- Modify Right Margin 2.4.13
- Modify Roll 2.4.12
- Modify Tabs 2.4.14
- Move Line 2.4.7
- Quit 2.1, 2.4.3
- Recover Edit 2.4.4
- Replace String 2.4.11
- Save Line 2.4.16
- Selection 2.2
- Show Line 2.4.15
- Usage 2.4
- Control Function 2.5, T2-1
 - Back Tab 2.5.8
 - Backspace 2.5.18
 - Clear to Tab 2.5.6
 - Cursor Down 2.5.16
 - Cursor Left 2.5.18
 - Cursor Right 2.5.17
 - Cursor Up 2.5.15
 - Delete Character 2.5.14
 - Delete Line 2.5.12
 - Display/Suppress Line Numbers 2.5.4
 - Duplicate to Tab 2.5.5
 - Edit/Compose Mode 2.5.2
 - Enter Command Mode 2.5.1
 - Erase 2.5.20
 - Home 2.5.19
 - Insert Character 2.5.13
 - Insert Line 2.5.11
 - New Line 2.5.3
 - Roll Down 2.5.10
 - Roll Up 2.5.9
 - Tab 2.5.7
- Control Keys T2-1
- Data Display 2.2
- Data Entry 2.2
- DX10 2.1
- Edit Control 2.2
- Example F2-1, F2-2, F2-3
- File:
 - Create New 2.3.1
 - Edit Existing 2.3.2
 - Operation, Types of 2.2
 - Syntax Conventions 2.4
 - Walk Throughs 2.3
- TGS Command SYS PROG
- Time Delay SVC Codes >02 APPL PROG
- TTY Mode, Command Entry
 - Control Keys PROD OPER
- TX Program File Command,
 - Link Editor PROD OPER
- TXCM Command PROD OPER
- TXCP Command PROD OPER
- TXDF Command PROD OPER
- TXDX Command PROD OPER
- TX/DX Conversion PROD OPER
- TXMD Command PROD OPER
- TXSF Command PROD OPER
- TXXLE Command PROD OPER
- Unconditional Suspend Task, Debug 7.2.1
- Unconditional Wait SVC
 - Codes >06 APPL PROG
- Unload Disk Volume SVC
 - Codes >34 SYS PROG
- Unload Volume Command PROD OPER
- UV Command PROD OPER
- VB Command PROD OPER
- VC Command PROD OPER
- VDT Extended Support CONCEPTS
- VDT Mode, Command Entry
 - Control Keys PROD OPER
- VDT Operation PROD OPER
- Verify Backup Command PROD OPER
- Verify Copy Command PROD OPER
- Video Display Terminal Operation 2.2.2
- WAIT Command PROD OPER
- Wait for Background
 - Termination Command PROD OPER
- Wait for I/O SVC Codes >01 APPL PROG
- Wait on Multiple Initiate I/O
 - SVC Codes >36 APPL PROG
- Walk Throughs, Text Editor 2.3
- WCRU Command PROD OPER
- WHAT Command, GEN990 SYS PROG
- Write CRU Command PROD OPER
- Writing Programs APPL PROG
- XANAL Command ERROR, SYS PROG
- XB Command PROD OPER
- XCT Command APPL PROG
- XCTF Command APPL PROG
- XCU Command SYS PROG
- XD Command 7.4.1
- XE Command 2.1, 2.4.1
- XES Command 2.4.2
- XFC Command APPL PROG
- XFCF Command APPL PROG
- XFT Command APPL PROG
- XFTF Command APPL PROG
- XGEN Command PROD OPER, SYS PROG
- XHT Command 6.2.3
- XLE Command APPL PROG
- XMA Command APPL PROG
- XPS Command SYS PROG
- XSB Command APPL PROG
- XT Command 6.2.1, PROD OPER
- XTS Command 6.2.2, PROD OPER

FOLD



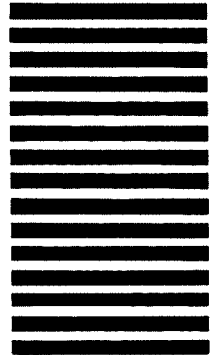
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 7284 DALLAS, TX

POSTAGE WILL BE PAID BY ADDRESSEE

TEXAS INSTRUMENTS INCORPORATED
DIGITAL SYSTEMS GROUP

ATTN: TECHNICAL PUBLICATIONS
P.O. Box 2909 M/S 2146
Austin, Texas 78769



FOLD

Texas Instruments U.S. District Sales and Service Offices

(A complete listing of U.S. offices is available from the district office nearest your location)

California

831 S. Douglas Street
El Segundo, California 90245
(213) 973-2571

100 California Street
Suite 480
San Francisco, California 94111
(415) 781-9470

776 Palomar Avenue
P.O. Box 9064
Sunnyvale, California 94086
(408) 732-1840*

3186 Airway
Suite J
Costa Mesa, California 92626
(714) 540-7311

Colorado

9725 East Hampden Avenue
Suite 301
Denver, Colorado 80231
(303) 751-1780

Florida

1850 Lee Road
Suite 115
Winter Park, Florida 32789
(305) 644-3535

Georgia

3300 Northeast Expressway
Building 9
Atlanta, Georgia 30341
(404) 458-7791

Illinois

515 West Algonquin Road
Arlington Heights, Illinois 60005
(312) 640-2900
(800) 942-0609*

Massachusetts

504 Totten Pond Road
Waltham, Massachusetts 02154
(617) 890-7400

Michigan

24293 Telegraph Road
Southfield, Michigan 48034
(313) 353-0830
(800) 572-8740*

Minnesota

7625 Parklawn Avenue
Minneapolis, Minnesota 55435
(612) 830-1600

Missouri

2368 Schuetz
St. Louis, Missouri 63141
(314) 569-0801*

New Jersey

1245 Westfield Avenue
Clark, New Jersey 07066
(201) 574-9800

Ohio

4124 Linden Avenue
Dayton, Ohio 45432
(513) 258-3877

Pennsylvania

420 Rouser Road
Coraopolis, Pennsylvania 15108
(412) 771-8550

Texas

8001 Stemmons Expressway
P.O. Box 226080
M/S 3108
Dallas, Texas 75266
(214) 689-4460

13510 North Central Expressway
P.O. Box 225214
M/S 393
Dallas, Texas 75265
(214) 238-3881

9000 Southwest Freeway, Suite 400
Houston, Texas 77074
(713) 776-6577

8585 Commerce Drive, Suite 518
Houston, Texas 77036
(713) 776-6531
(713) 776-6553*

Virginia

1745 Jefferson Davis Highway
Crystal Square 4, Suite 600
Arlington, Virginia 22202
(703) 553-2200

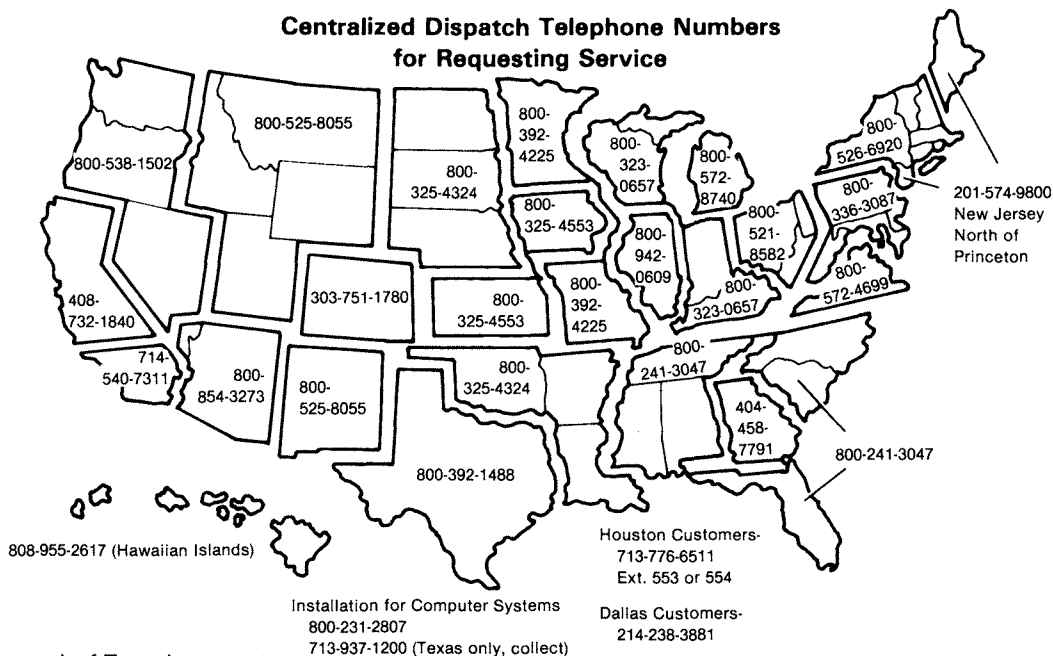
Wisconsin

205 Bishops Way
Suite 214
Brookfield, Wisconsin 53005
(414) 784-1323

*Service telephone number

TI-CARE*

**Centralized Dispatch Telephone Numbers
for Requesting Service**



*Service mark of Texas Instruments

The TI Customer Support Line is available to answer our customers' complex technical questions. The extensive experience of a selected group of TI senior engineers and systems analysts is made available directly to our customers. The TI Customer Support Line telephone number is (512) 250-7407.



TEXAS INSTRUMENTS

INCORPORATED

DIGITAL SYSTEMS GROUP

POST OFFICE BOX 2909

AUSTIN, TEXAS
