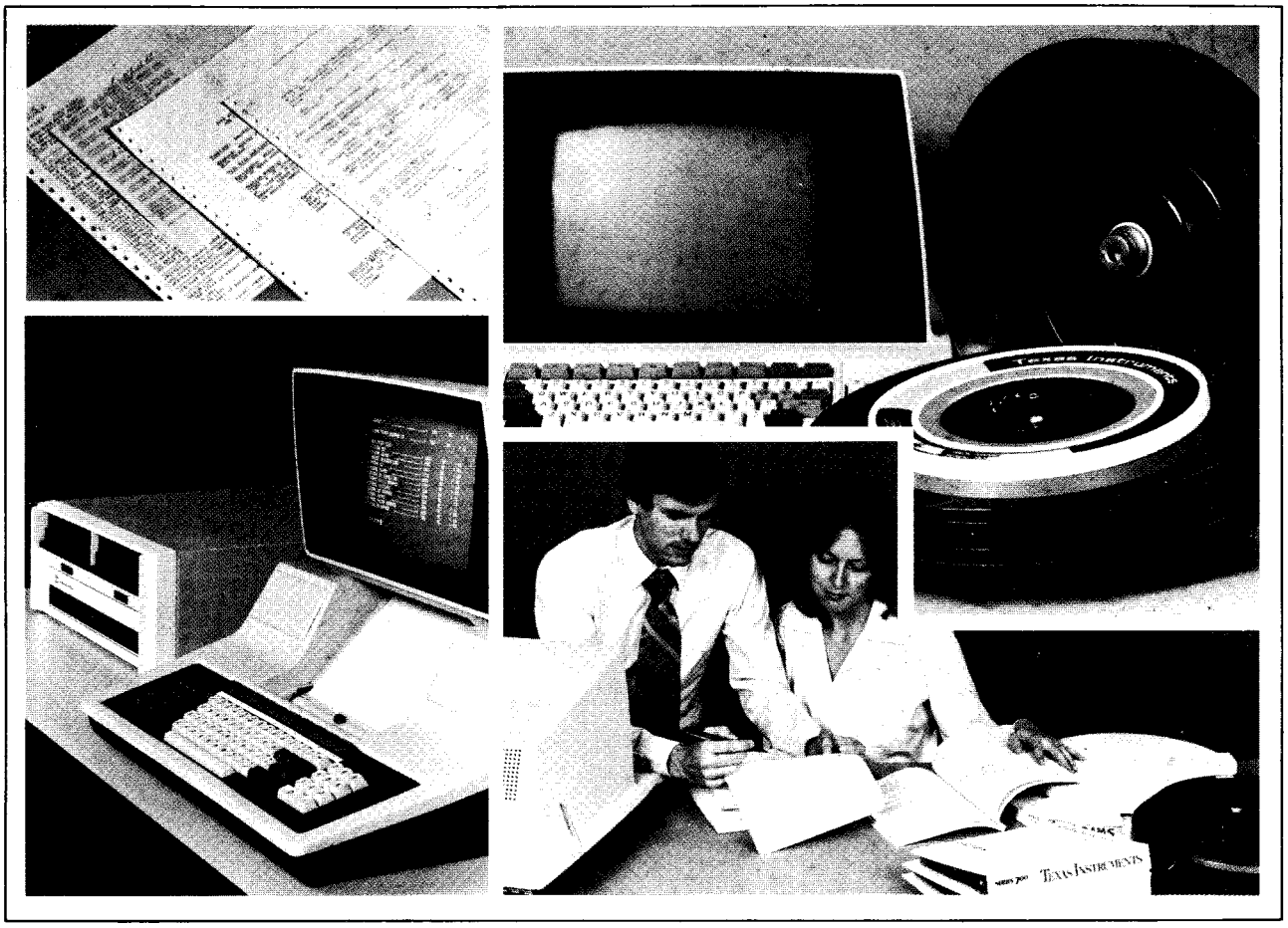
# Model 990 Computer
# DX10 Operating System Release 3
# Reference Manual



## Developmental Operation
## Volume IV

Part No. 946250-9704 *A
15 December 1979

TEXAS INSTRUMENTS

# LIST OF EFFECTIVE PAGES

INSERT LATEST CHANGED PAGES DESTROY SUPERSEDED PAGES

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

Model 990 Computer DX10 Operating System Release 3 Reference Manual, Volume IV, Developmental Operation (946250-9704)

Original Issue .....15 August 1977
Revised .....15 March 1978 (ECN 419811)
Revised .....15 October 1978 (ECN 003584)
Revised .....15 December 1979 (MCR 000208)

Total number of pages in this publication is 172 consisting of the following:

| PAGE NO. | CHANGE NO. | PAGE NO. | CHANGE NO. | PAGE NO. | CHANGE NO. |
|---|---|---|---|---|---|
| Cover | 0 | | | | |
| Effective Pages | 0 | | | | |
| iii - x | 0 | | | | |
| 1-1 - 1-2 | 0 | | | | |
| 2-1 - 2-22 | 0 | | | | |
| 3-1 - 3-18 | 0 | | | | |
| 4-1 - 4-2 | 0 | | | | |
| 5-1 - 5-8 | 0 | | | | |
| 6-1 - 6-4 | 0 | | | | |
| 7-1 - 7-30 | 0 | | | | |
| 8-1 - 8-8 | 0 | | | | |
| A-1 - A-2 | 0 | | | | |
| B-1 - B-8 | 0 | | | | |
| Index-1 - Index-54 | 0 | | | | |
| User's Response | 0 | | | | |
| Business Reply | 0 | | | | |
| Cover Blank | 0 | | | | |
| Cover | 0 | | | | |

# PREFACE

The *Model 990 Computer DX10 Operating System Reference Manual* describes the features of the 990 Disk Executive, DX10. The DX10 operating system reference manual is divided into five volumes plus a sixth volume devoted entirely to error reporting and recovery. Each volume 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 volumes to enhance the development and application of programs. The volumes are organized with both the applications programmer and the production operation in mind, as well as providing the system or application programmer details to allow the extension and/or modification of DX10.

The user should consult all six volumes to become thoroughly familiar with all facets and capabilities of the operating system. Each volume serves a particular purpose and is designed to meet a specific goal. No one volume is intended to stand alone as a complete system tutorial.

General DX10 background information and its features, concepts, and facilities are described in this volume. The other five volumes discuss detailed information specific to production operation, developmental operations, applications programming, system programming and errors. The titles and part numbers of the six volumes along with a brief comment regarding content and level of each are as follows:

*Volume I, Model 990 Computer DX10 Operating System Release 3 Reference Manual, Concepts and Facilities,* part number 946250-9701 — Includes features, concepts, and general background information describing the DX10 Operating System.

*Volume II, Model 990 Computer DX10 Operating System Release 3 Reference Manual, Production Operation,* part number 946250-9702 — Describes the user interface to the DX10 Operating System and application programs. It also contains information required to start-up, query, control, and maintain the DX10 Operating System. This volume describes the steps needed to run application programs on Texas Instruments 990/10 and 990/12 computer systems under control of the DX10 operating system. Accordingly, it 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. It details commands available to terminal users both in interactive and in batch mode and describes how to enter a command.

*Volume III, Model 990 Computer DX10 Operating System Release 3 Reference Manual, Application Programmer's Guide,* part number 946250-9703 — 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 processing. Information is provided describing the DX10 file structures and detailed information describing calls to DX10 for file Input/Output processing.

*Volume IV, Model 990 Computer DX10 Operating System Release 3 Reference Manual, Developmental Operation,* part number 946250-9704 — 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 the program installation/deletion functions.

**NOTE**

Additional, in-depth descriptions related to specific languages including FORTRAN (part number 946260-9701), COBOL (part number 946266-9701), BASIC (part number 2250304-9701), RPGII (part number 939524-9701), TI Pascal (part number 946290-9701), and ASSEMBLY LANGUAGE (part number 943441-9701) is found in manuals dedicated to the appropriate programming language. A Link Editor manual (part number 949617-9701) is provided as a separate volume that describes the application of the link edit function in a DX10 environment. A separate manual (part number 946252-9701) describes the use of an optional Sort/Merge package. There are also two manuals which describe the DBMS package (part numbers 2250425-9701 and 2250426-9701).

*Volume V, Model 990 Computer DX10 Operating System Release 3 Reference Manual, System Programming Guide,* part number 946250-9705 — Includes information required by the Systems or Application Programmer in extending or modifying DX10. It provides detailed discussion in such areas as System Generation, support of non-standard devices, and privileged supervisor service calls available within DX10. Also included are detailed instructions and descriptions of how to add a command to the System Command Interpreter (SCI) Package. In addition, it contains specific information necessary to customize DX10 for a particular configuration and application.

*Volume VI, Model 990 Computer DX10 Operating System Release 3 Reference Manual, Error Reporting and Recovery,* part number 946250-9706 — Includes information describing error reporting within DX10. It documents task errors, command errors, Supervisor Call (SVC) errors, SCI errors, magnetic tape and other Input/Output errors. Volume VI documents all DX10 errors in cross reference table form and includes a resolution of each message and suggested recovery techniques.

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

| Title | Part Number |
|---|---|
| *Model 990 Computer DX10 Operating System Release 3, System Design Document* | 939153-9701 |
| *Model 990 Computer Model 911 CRT Display Terminal Installation and Operation* | 945423-9701 |
| *Model 990 Computer Model 913 CRT Display Terminal Installation and Operation* | 943457-9701 |
| *Model 733 Terminal User's Guide* | 945259-9701 |
| *Model 743 KSR Data Terminal Operating Instructions* | 984030-9701 |
| *Model 990 Computer Interim Addendum to the TMS 9900 Microprocessor, Assembly Language Programmer's Guide for Model 990/12 Computer* | 2250077-9701 |
| *990/10 Minicomputer Hardware Reference Manual* | 945417-9701 |

| | |
|---|---|
| *DX10 3270 User's Guide* | 2250954-9701 |
| *Model 990 Computer DX10 Remote Terminal Subsystem (RTS) for Model 915 Remote Terminal Installation and Operation Manual* | 2250356-9701 |
| *DX10 3780/2780 Emulator Object Installation Manual* | 2250918-9701 |
| *Model 990 Computer DX10 3270 Interactive Communication Software (ICS) Object Installation Manual* | 2250954-9701 |
| *Model 990 Computer 306 and 588 Line Printers Installation and Operation* | 945261-9701 |
| *Model 990 Computer PROM Programming Module Installation and Operation* | 945258-9701 |
| *Model 990 Computer Communications System Installation and Operation* | 945409-9701 |
| *Model 990 Computer Communication System Software* | 946236-9701 |
| *Model 990 Computer Terminal Executive Development System (TXDS) Programmer's Guide* | 946258-9701 |
| *Model 990 Computer Model FD800 Floppy Disk System with International Chassis Installation and Operation* | 2250697-9701 |
| *Model 990 Computer Model FD1000 Flexible Disk System with International Chassis Installation and Operation* | 2250698-9701 |
| *Model 990 Computer Model 733 ASR/KSR Data Terminal Installation and Operation* | 945259-9701 |
| *Model 990 Computer Model 804 Card Installation and Operation* | 945262-9701 |

# TABLE OF CONTENTS

## TABLE OF CONTENTS (Continued)

## TABLE OF CONTENTS (Continued)

## TABLE OF CONTENTS (Continued)



## APPENDIXES



## LIST OF ILLUSTRATIONS



## LIST OF TABLES

# 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 interactively to create and modify files of textual data. 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, either of the TI VDTs can operate in 'TTY' or 'VDT' modes. The Text Editor supports only the VDT mode when called from a VDT. *Volume II, DX10 Operating System Production Operations Guide,* 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 Volume VI, Error Reporting, of the DX10 documentation set.

## 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 includes 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. It also includes 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 Model 911 and 913 VDTs. The Model 913 provides a twelve line display screen, while Model 911 provides a twenty-four line display screen.

Editing on the VDT occurs on a page basis, with a page being either twelve lines (Model 913 VDT) or twenty-four 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

**Table 2-1. Edit Control Functions**

| Function | 913 VDT Keytop | 911 VDT Keytop | TTY/820 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 | ****** |
| 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

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.

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 in, and that the System Command Interpreter (SCI) is active. Refer to *Volume II, DX10 Operating System Production Operations Guide* for details on Log In and activating SCI.

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

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

Press the RETURN 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 row one, with the cursor in column one, row one:

```
* EOF
```

This display indicates that the only record in the file is the end-of-file record. To begin entering data, press the F7 key to enter the compose mode, and then press the unlabeled grey 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 row 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 A's.

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). Note, however, that all of the commands return the Text Editor to the edit mode once they have completed. To begin entering new data, first position the cursor to the last data line (using the cursor control keys, such as Up Arrow (↑) and Down Arrow (↓), and then press the F7 key and then press the unlabeled gray 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 then 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 example uses the N response. The following display is then presented:

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

The cursor appears after the colon in the first line 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. The example uses .EXAMPLE as the pathname for the new file.

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, with the initial SCI menu 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 then displayed:

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

Press the RETURN key to display the first 24-records from the file. The Text Editor is in the edit mode, the cursor is in column one, row one, and line numbers are displayed. In this example, we are going to change lines one and two, insert one line after line nine, insert one line after line 19, change line 20, and delete lines 30 through 35.

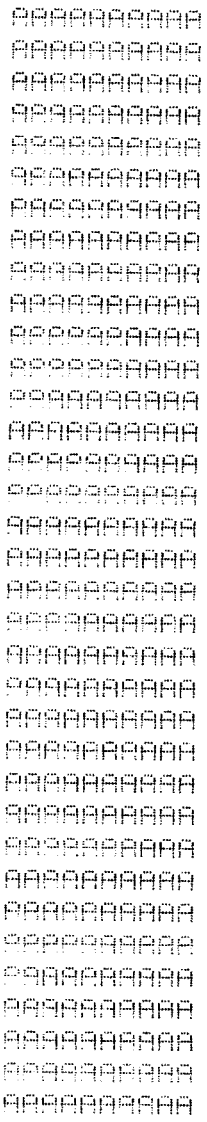


**Figure 2-1. Listing of Text Editor Example**

First, change the first character of line one from A to B. Do this by simply keying the B while the cursor was in row one, column one. Press the RETURN key to place the cursor in row two, column one. To change the last character (in column ten of row two) from A to B, press the right arrow (→) to advance the cursor one position for each stroke. To get to column ten, press the → key nine times. Once the cursor is over the last A, the B is keyed to replace the existing A. The RETURN key is then pressed to move the cursor to column one of line three.

The next modification is to insert a line of data after line nine. There are two ways to move the cursor to line ten (since the insert line function causes the new line to be placed ahead of the line the cursor is in, we have to go to line ten). One way to move the cursor is by using the Show Line command. To use the command, press the CMD key to enter the command mode.

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

```
SHOW LINE
          LINE:  1
```

Enter the number of the line that the inserted line is to precede. For the example, we 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 (↓) 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 ↓ key ten times. Press the unlabeled gray key, which caused the lines below and including line twenty 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, which we want to change from AAAAAAAAAA to CCCCCCCCCC. Do it by simply keying the new value and RETURN key.

Our 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 TAB 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 to perform the Delete Lines function. Since line 35 is the last input line from the file .EXAMPLE, the following display is presented:

```
* EOF
```

Since we have completed all modifications, call the QE command to terminate processing. The CMD key is pressed and the command prompt [ ] is displayed. We then enter QE and press RETURN, and the following is displayed:

```
QUIT EDIT
      ABORT?:  N
```

Since we do not want to discard all our modifications, press RETURN 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. We accept that value by pressing the RETURN key and get the following display:

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

Enter Y in response to the REPLACE parameter to indicate we want to replace the input file. 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.

## 2.4 COMMAND USAGE

The Text Editor is initiated when the operator selects and completes the XE command and 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 reactivated, 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 ONLY ALLOWED 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
BBBBBBBBBB
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
BBBBBBBBBB
CCCCCCCCCC
AAAAAAAAAA
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   AAAAAAAAAB
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 (_) | 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. |
| Upper Case | 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 is used to indicate these values. The following format applies to start line entries:

i = A numbered line in the input file. This value refers to an existing numbered line. A zero value is not allowed.

i±j = A line referenced by its offset (±j) from a numbered line (i). This value is used to reference a newly entered line. Neither i nor j may be zero.

±j = A line referenced by its offset (±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 = A numbered line in the input file. A zero value is not allowed.

i±j = The ending line is offset (±j) from a numbered line (i) in the input file. Neither i nor j 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 the evaluation of the start line 'int' expression is greater than the value of i in the end line 'int' expression (if used), the start line value is used in place of the end line entry.

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

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. An access name 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.

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 to 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' is displayed in column 20, a '3' in column 30, etc. XES should be used to initiate the Text Editor when editing source code in a language which requires column oriented input.

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 **COPY LINES – CL COMMAND.** The CL command is used to specify 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.

2.4.5 **DELETE LINES – DL COMMAND.** The DL command is used to specify 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.

2.4.6 **MOVE LINES – ML COMMAND.** The ML command is used to move lines from one position in the current file to another position in the modifications file, and to delete 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.

2.4.7 **INSERT FILE – IF COMMAND.** The IF command is used to insert an entire file into the edited file after a specified record. The format of the command is as follows:

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

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.

In response to the INSERT AFTER LINE prompt, the user enters the line (record) 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.

**2.4.8 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 leading or trailing blanks, the entire string must be enclosed in double quotes. Quotes must also be used to enclose a string containing blanks.

**2.4.9 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. Quotes must enclose strings containing leading or trailing blanks, or blanks only.

**2.4.10 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.

2.4.11 **MODIFY ROLL – MR COMMAND.** The MR command is used to change 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:  {<int>}
                              {12   }
```

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.

2.4.12 **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. However, editing data past the margin is not lost. The format of the MRM command is:

```
MODIFY RIGHT MARGIN
    RIGHT MARGIN POSITION:  {<column>}
                            {prev val}
```

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

2.4.13 **MODIFY TABS – MT COMMAND.** The MT command is used to change or clear the tab settings of the device. The format of the MT command is as follows:

```
MODIFY TABS
    TAB COLUMNS:  {<int>   }
                  {prev val}
```

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.

**2.4.14 SHOW LINE – SL COMMAND.** The SL command is used to position the current file to any one line (record). When the operator keys in SL, the system presents the following:

SHOW LINE:

LINE: {<B> <E> <int> 1}

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).

Default – The default parameter is '1'.

**2.4.15 SAVE LINES - SVL COMMAND.** The SVL command may be used to copy 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 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 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. However, if no value is input for both the START LINE and END LINE parameters, the entire edit file is saved.

## 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 keys specified on the 911 VDT, the 913 VDT, and the hard copy devices (820 KSR and TTY) by which the functions are called. The user should note that not all of the edit control functions are supported by hard copy devices.

The edit functions in the first group in table 2-1 are implemented as event keys, while the remainder of the edit functions are implemented as task edit keys. The event keys are given a higher priority and are processed before task edit keys. Therefore, if a task edit key is entered many times and an event key is entered before all of the responses to the task edit keys are completed, the event key request is processed before the rest of the task edit requests. An example would be entering the "INSERT LINE" key six times followed by the "ROLLUP" key. If two of the line insertions have been processed when the "ROLLUP" is entered, the page is rolled and then the last four remaining insert lines are processed. This type of response results anytime when edit and event keys are mixed.

Within the following description 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 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.

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 either a 913 VDT or a 911 VDT. Refer to table 2-1 for the function keys of the other terminal types suported by DX10. 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 in using the procedures given in Volume II.

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

      FILE ACCESS NAME: press CLEAR AND NEW LINE (913 VDT) or ERASE FIELD and RETURN (911 VDT)

   c. Place the Text Editor in the compose mode by pressing F7.

   d. Press the INS LINE (913 VDT) or the unlabeled gray key (911 VDT) to move the cursor one line above *EOF and type in the program shown in figure 3-1. (Do not press NEW LINE or RETURN after entering the last line).

   e. Press the CMD (911) or HELP (913) key to leave the compose mode.

   f. 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 (911 VDT) or NEW LINE (913 VDT)

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 or NEW LINE)
      OPTIONS: (Press RETURN or NEW LINE)
      MACRO LIBRARY PATHNAME: (Press RETURN or NEW LINE)
      PRINT WIDTH: 80
      PAGE LENGTH: 60

When the assembly completes, the following message is displayed (after the RETURN key is pressed):

MACRO ASSEMBLY COMPLETE 0000 ERRORS, 0000 WARNINGS

Press RETURN or NEW LINE 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 ERASE FIELD and RETURN (911 VDT)<br>or<br>CLEAR and NEW LINE (913 VDT) |

   b. Place the Text Editor in compose mode by pressing F7 and then press the INS LINE key (913 VDT) or the unlabeled gray key (911 VDT).

   c. Enter the following lines:

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

   d. Leave the compose mode by entering CMD or HELP.

   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 or NEW LINE |

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

WAITING FOR BACKGROUND TASK TO COMPLETE

When the Link Editor terminates, the following is displayed:

LINK EDITOR COMPLETED, 0 ERRORS, 0 WARNINGS

Press HELP or 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 HELP or 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 or HELP 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 LUNO >20
       DATA 0
       DATA 0
       DATA 0
       DATA 0
MSSG0  DATA 0              I/O REQUEST
       BYTE >B,>20         WRITE ASCII ON LUNO >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 LUNO >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 LUNO >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 LUNO >20
```

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

```
        BYTE 0,>40          READ AFTER WRITE
        DATA ITEM3
        DATA 0
        DATA 10
        DATA STR3
STR3    DATA STORE+3        3RD ITEM STORE LOCATION
        DATA 4
        DATA 0
ITEM3   DATA >0A0D
        TEXT 'ITEM 3
MSSG4   DATA 0              I/O REQUEST
        BYTE >B,>20         WRITE TO LUNO >20
        DATA 0
        DATA GOODBY         MESSAGE LOCATION
        DATA 0
        DATA CLOSE-GOODBY MESSAGE LENGTH
*****************************************************
*       FINAL MESSAGE DISPLAYED                     *
*****************************************************
GOODBY DATA >0A0D
        TEXT 'THANK YOU.  THAT COMPLETES TODAY''S'
        TEXT ' TRANSACTIONS.'
        DATA >0A0D
CLOSE   DATA 0              I/O REQUEST
        BYTE 1,>20          CLOSE LUNO >20
        DATA 0
        DATA 0
        DATA 0
        DATA 0
EOP     BYTE >16,0          TERMINATE TASK
*
START   XOP  @OPEN,15       OPEN LUNO >20
        XOP  @MSSG0,15      OPENING MESSAGE
        XOP  @MSSG1,15      INPUT 1
        XOP  @MSSG2,15      INPUT 2
        XOP  @MSSG3,15      INPUT 3
        XOP  @MSSG4,15      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 HELP or 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 or 913 VDT and a line printer (LP01).

1. Enter the program into the computer.

   a. Power up the computer and terminal and log in using the procedure given in Volume II.

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

      FILE ACCESS NAME: press CLEAR and NEW LINE (913 VDT) or ERASE FIELD and RETURN (911 VDT).

   c. Place the Text Editor in the compose mode by pressing F7.

   d. Press the INS LINE (913 VDT) or unlabeled gray key (911 VDT) key to move the cursor one line above *EOF and type in the following FORTRAN program (do not press NEW LINE or RETURN after entering the last line):

```
      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
```

e. Press the CMD (911 VDT) or HELP (913 VDT) key to leave the compose mode.

f. 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 or NEW LINE) |
| PRINT WIDTH: | 80 (Press RETURN or NEW LINE) |

Enter WAIT and press NEW LINE or RETURN. The following message is then displayed.

WAITING FOR BACKGROUND TASK TO COMPLETE

When the FORTRAN compile completes, the message

FORTRAN COMPILER NORMAL COMPLETION

is displayed. Press HELP or CMD and the message

FORTRAN COMPILER COMPLETED 0 ERRORS, 0 WARNINGS:

is displayed. Press HELP or 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 CLEAR, then NEW LINE or ERASE FIELD and RETURN |

b. Place the Text Editor in compose mode by pressing F7 and pressing the INS LINE key (913 VDT) or the unlabeled gray key (911 VDT).

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 (911 VDT) or HELP (913 VDT).

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 or NEW LINE)
```

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 or NEW LINE)
```

Enter WAIT and press NEW LINE or 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 HELP or 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 HELP or 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:   UNIT 18
  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 or NEW LINE and the message "STOP 0 NORMAL PROGRAM COMPLETION" is displayed. After the program finishes, it is recommended that one 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 the city of

interest. Since the program is already designed and documented, steps 1 and 6, respectively, may be omitted. The runthrough assumes the use of a 911 VDT or 913 VDT and a line printer (LP01).

1. Enter the program into the computer.

   a. Power up the computer and terminal, and log in using the procedures given in Volume II.

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

      FILE ACCESS NAME: Press CLEAR and NEW LINE or ERASE FIELD and RETURN.

   c. Press the F7 key to enter the compose mode.

   d. Press the INS LINE key (913 VDT) or the unlabeled gray key (911 VDT) to position the cursor above the displayed *EOF.

   e. Press CMD (911 VDT) or HELP (913 VDT) to enter command mode. Enter MRM and press RETURN or NEW LINE to modify the right margin. Respond as follows:

      MODIFY RIGHT MARGIN
      RIGHT MARGIN POSITION: 72

      Press RETURN or NEW LINE.

   f. Type the program as shown in figure 3-2. Do not press NEW LINE or RETURN after keying the last line since this will cause a blank line to be inserted at the end of the source.

   g. Press the CMD (911 VDT) or HELP (913 VDT) key to enter the command mode.

   h. 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                   " POSISITON 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: | .CBLLST |
| OPTIONS: | (Press RETURN or NEW LINE) |
| PRINT WIDTH: | 80 (press RETURN or NEW LINE) |
| PAGE SIZE: | 55 |
| PROGRAM SIZE (LINES): | 1000 |

c. Enter WAIT and press RETURN or NEW LINE. 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 HELP (913 VDT) or CMD (911 VDT) 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 NEW LINE or 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 .CBLLST.

## 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 = 6x5x4x3x2x1 = 720.)

1. Enter the program into the computer.

   a. Power up the computer and log in, using the procedures given in Volume II.

   b. Invoke one of the two BASIC processors. Business BASIC is used in this example, but Scientific BASIC performs the same for this program. Enter XBB (XSB).

   c. When the BASIC interpreter signifies its readiness by prompting (*), type the following program:

```
10  PRINT "FACTORIAL PROGRAM", CLK$, DAT$
15  PRINT "ENTER THE NUMBER WHOSE FACTORIAL YOU WANT;"
20  INPUT N
25    IF N=0 THEN 140
30    IF N<57 THEN 70
```

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

```
 40   PRINT "AN ATTEMPT TO COMPUTE A VALUE GREATER THAN 56! WILL PRODUCE";
 50     PRINT "MACHINE OVERFLOW"
 60   GOTO 140
 70   F=1
 80   FOR X=1 TO N
 90     LET F=F*X
100   NEXT X
110   PRINT N,F
120   PRINT
130   GOTO 20
140   END
```

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 QU. 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 BASIC 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 XRPGC or XRPGCF 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.

```
DXRPG        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...
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                                               TEST17
0006        I                                        1   20NEWID          TEST17
0007        I                                        6  30 NAME           TEST17
0008        IRELREC  NS  04                                               TEST17
0009        I                                        1   20ID             TEST17
0010        I                                        6  30 FIELD      98   TEST17
0011        C           NEWID     CHAINRELREC            99               TEST17
0012        C   98                EXCPT                                   TEST17
0013        C           END       TAG                                     TEST17
0014        ORELREC  E        01 98                                       TEST17
0015        O                        NEWID      2                         TEST17
0016        O                        NAME      30                         TEST17
0017        OOUTREL  D        01                                          TEST17
0018        O                        NEWID      3                         TEST17
0019        O                        NAME      35                         TEST17
0020        O                  98               47 'ADDED'                TEST17
0021        O                  04N98            51 'DUPLICATE'            TEST17
0022        O                  99               45 'EOF'                  TEST17
0023  /*                                                                  TEST17


THE FOLLOWING INDICATORS APPEARED IN THIS PROGRAM

01  04  98  99


S O R T E D  L A B E L  N A M E S

NAME          LINE

END            13


S O R T E D  F I E L D  N A M E S

NAME      TYPE   LENGTH  DP

FIELD       A      25
ID          N       2     0
NAME        A      25
NEWID       N       2     0
UDATE       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 XRPGTF 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 CREATION OF CONTROL FILES AND EXECUTION OF SORT/MERGE

The SORT/MERGE control file may be created interactively or by creating a batch stream with the text editor. After the control file has been created, the SORT/MERGE program may be executed by means of the XSM or XSMF procedures, as shown in figure 3-6, or by the XBSM, as shown in figure 3-7.

```
                    BACKGROUND EXECUTION

[] XSM

                    NEW CONTROL FILE?: N
                    CONTROL FILE NAME: .XYZ
                    RUN SORT MERGE?: Y
                    LISTING DEVICE NAME: LP01
[]
                    Other Tasks                    SORT/MERGE Execution

    SORT/MERGE NORMAL COMPLETION
[]

                    FOREGROUND EXECUTION
                    (From ST02)

[] XSMF

                    NEW CONTROL FILE?: NO
                    CONTROL FILE NAME: CR01
                    RUN SORT MERGE?: YES
                    LISTING DEVICE NAME: ST02

    SORT/MERGE Executes

[]
```

**Figure 3-6. Interactive Options — No Control File Creation**

**3.6.1 EXECUTING SORT/MERGE IN BATCH MODE.** The command to execute the SORT/MERGE in batch mode is as follows:

XBSM CFN = filename LDN = device (or filename)

If the user has previously created the control file, only the XBSM command is required to execute SORT/MERGE in batch mode. If the control file is being built in the batch stream, the filename on the XBSM statement must match with the filename on the SM$SMC statement. Figure 3-7 shows a file created by the text editor, which is then used as a batch stream with the XB command.

```
BATCH
SM$SMC CFN=DS03.FILNAM
SM$HD SMT=SORTR,TCL=6,ORL=80,MS=8000
SM$OUT FP=DS03. TEST,LRL=80,PRL=560
SM$WKF WFV=DS02,WFT=E
SM$IN FP=DS03.EIGH4
SM$SLC RST=I
SM$REF FT=N,CP=C,BL=32,EL=37
SM$REF FT=D,CP=C,BL=1,EL=80
SM$CLS
XBSM CFN=DS03.FILNAM,LDN=LP01
EBATCH
```

**Figure 3-7. Example of Entries to Create a Batch File and Execute SORT/MERGE**

# 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 which runs under DX10.

The disk-based operating system for the Model 990/10 computer, DX10, is a multitasking operating system, which supports all of the features of the Link Editor. Being a disk-resident system, DX10 is well-suited for overlay structured programs, such as the supported 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
- 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 "DUMY" which will suppress the generation of the output. Use of the DUMY 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 "DUMY" 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 manual.

The last prompt, LINE 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 LINE 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 Volume III, Application Programming Guide, of the DX10 Operating System documentation. 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 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 which install, delete, and modify programs.

Installing or modifying a task or procedure to be memory resident requires that the system be rebooted 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 Volume III, Section 2.

**NOTE**

> The user should not install a task on the S$SDS$ program file. If the S$SDS$ 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:

INSTALL TASK

PROGRAM FILE OR LUNO: {<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}

These questions are asked only if the answer to DEFAULT TASK FLAGS is 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 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 are resident 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 Volume III. Before installing a real-time task, refer to the NOTE in the paragraph for the Install Task command.

Syntax:

INSTALL REAL-TIME TASK

| | |
|---|---|
| PROGRAM FILE OR LUNO: | {<acnm> / <int>} |
| TASK NAME: | <string> |
| TASK ID: | {<int> / 0} |
| OBJECT PATHNAME OR LUNO: | {<acnm> / <int>} |
| PRIORITY: | {<int> / 1 - 127} |
| 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} |

These questions are asked only if the answer to DEFAULT TASK FLAGS is 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 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 are resident 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:

INSTALL PROCEDURE

PROGRAM FILE OR LUNO: {<acnm> | <int>}

PROCEDURE NAME: <string>

PROCEDURE ID: {<int> | 0}

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

MEMORY RESIDENT?: {<YES> | NO}

DELETE PROTECT?: {<YES> | NO}

EXECUTE PROTECT?: {<YES> | NO}

WRITE PROTECT?: {<YES> | NO}

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

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:

INSTALL OVERLAY

PROGRAM FILE OR LUNO: {<acnm> | <int>}
OVERLAY NAME: <string>
OVERLAY ID: <int>
OBJECT PATHNAME OR LUNO: {<acnm> | <int>}
RELOCATABLE?: {<YES> | NO}
DELETE PROTECT?: {<YES> | NO}
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 zero.

### 5.5 DT – DELETE TASK

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

Syntax:

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:

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:

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:

MODIFY TASK ENTRY

PROGRAM FILE PATHNAME: <acnm>

MODULE NAME OR ID: {<string> | <int>}

The PROGRAM FILE PATHNAME: prompt is responded to with the pathname of the program file within which the task, as identified by the response to the MODULE NAME OR ID prompt, is resident.

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 which 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 TAB key. There are some limitations on changing the REAL TIME and PRIORITY prompts. If REAL TIME is 'YES', the value for PRIORITY must range between 1 and $127_{10}$ (inclusive). If REAL TIME is 'NO', the value must range 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:
                REPLICTABLE:
            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 TAB 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' (in order to specify that the task is linked with procedure 1), and the task's load address must be $\geq C000_{16}$. Otherwise, the user will not be able to 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. Any of the entries may be changed or they may be accepted by pressing the TAB 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:

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

The PROGRAM FILE PATHNAME: prompt is responded to with the pathname of the program file within which the procedure, as identified by the response to the MODULE NAME OR ID: prompt, is resident. 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. Any of the displayed values may be changed, or the displayed value can be accepted by pressing the TAB 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:

```
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. Any of the entries may be changed, or accepted by pressing the TAB 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 varied language processors available for the Model 990/10 computer. 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 is resident, 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. All tasks are associated with a particular station.

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 AHD 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, Release 3**

| Processor/Subsystem | Execution Instructions Location |
|---|---|
| BASIC | *Model 990 Computer TI 990 BASIC Reference Manual,* 2250304-9701 |
| COBOL | Section 13 of *Model 990 Computer DX10 Operating System, Release 3, COBOL Programmer's Guide,* 946266-9701 |
| DBMS | Section 6 of *Model 990 Computer Data Base Administrator User's Guide,* 2250426-9701 |
| FORTRAN | Appendix H of *Model 990 Computer FORTRAN Programmer's Guide,* 946260-9701 |
| Pascal | Appendix C of *Model 990 Computer TI Pascal User's Manual,* 946290-9701 |
| RPGII | Section 2 of *Model 990 Computer Report Program Generator (RPGII) Programmer's Guide,* 939524-9701 |
| SORT/MERGE | Section 8 of *Model 990 Computer SORT/MERGE User's Guide,* 946252-9701 |

# 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 drastically reduce the number of bugs in a program; however, the bugs which 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 pin point 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 only be used 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 which 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 Volume III, or 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 Volume V) and high-level language programs (e.g., FORTRAN, 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 the "NEW LINE" function key for the terminal type in use (listed in table 2-1). Entry of the command mode is accomplished by pressing the appropriate "ENTER COMMAND" mode key for the terminal in use, also listed in 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, then symbols of the form

<phase name>.<IDT name>

are available. If either the phase name or the IDT name of a symbol is omitted, the immediately previous 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.

**NOTE**

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, use of angle brackets in expressions may be confused with use of angle brackets to indicate items supplied by the user. To avoid confusion, items supplied by the user are shown as lower case letters enclosed in angle brackets. When angle brackets enclose numerals or upper case letters, the contents of an address is indicated.

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,

#PC + NAME.IDT - #R15

is a valid expression.

## 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 Volume II. 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 STS (Show Task Status) 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 | V2-5.5 |
| 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: {<constant exp>}  runtime ID of task
                            { previous      }
                            ----------------
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: {<constant exp>}  runtime ID
                            { previous      }
                            ----------------
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.

Syntax:

```
MM
MODIFY MEMORY
          RUN ID: {<constant exp>}   runtime ID
                  {previous      }
         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 is used to modify the memory of system overlay two starting at address $05900_{16}$.

```
[ ] 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.

Syntax:

```
    LM
LIST MEMORY
                        RUN ID:  {<constant exp>}   runtime ID
                                 {previous      }
                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.

### 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. *Default value is 16.* |
| LISTING ACCESS NAME: | O | Access name of file or device to which output is to be sent. *Default is the terminal local file.* |

**7.3.8.3 LSM Command Example.** The LSM command is used to list the memory of system overlay six from byte $0100_{16}$ to byte $0120_{16}$.

```
[ ] 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 Breakpoint 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 will wait five seconds for the task to reach unconditional suspend and give 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 and the RT commands, the Proceed from Breakpoint command, or the Delete and Proceed from Breakpoint command, must be used to restart the task halted at a breakpoint. RT should be used to reactivate a halted task (HT) rather than AT.

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 command. This command is interactive and may be terminated by entering a CMD (911VDT), HELP (913VDT) or CNTL and X (hard-copy device) key at any time.

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, 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 a return. 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. This is an interactive command and may be terminated at any time by entering the command key.

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.

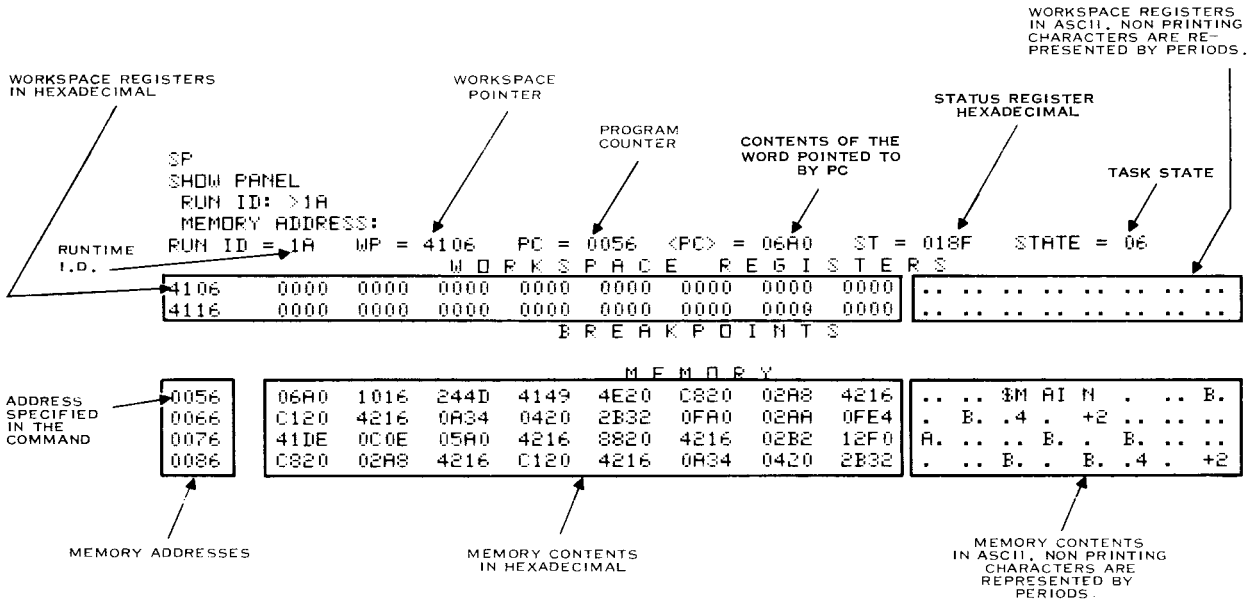



**Figure 7-1. Debug Panel Display**

**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 file (default is the terminal local file).

Syntax:

```
      SPI
SHOW PROGRAM IMAGE
                  PROGRAM FILE:   <acnm>          location of module
           OUTPUT ACCESS NAME:    <acnm>          define where to write
                                                  output

                                  {<Task>     }
                 MODULE TYPE:     {<Overlay>  }   task, overlay, or procedure
                                  {<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 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 operator.

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. |

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 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.

The contents of sixteen bytes are printed per line. The address of the first byte printed 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 that correspond to printable ASCII characters are translated and printed as ASCII characters. The nonprinting ASCII characters are 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> / 0} | starting word address |
| 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 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 has the capability to write anything, anywhere on the disk, and can therefore destroy the DX10 system image, the verify option should always be used.

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.

SADU outputs the contents of the specified ADU to the specified device. Output default is to the terminal making the request.

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> / 0} | first word to show |
| 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.)

MADU modifies the specified ADU on disk as directed by operator inputs. If verification data does not match the data already on disk, the modification will not be performed.

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     .. .. .. .. ..
[]
```

**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 over 64K (65,536) bytes, the user *must* supply the record number and a word offset into the record.

Syntax:

SRF
SHOW RELATIVE TO FILE

| | | |
|---|---|---|
| PATHNAME: | <acnm> | file pathname |
| RECORD NUMBER: | {<constant exp> / 0} | must be specified if word 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 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 above 64K (65,536) bytes must have a record number. Words below 64K bytes can be addressed directly and the sector is located by the program. Verification should be used if the file is critical to a program.

Syntax:

MRF
MODIFY RELATIVE TO FILE

| | | |
|---|---|---|
| PATHNAME: | <acnm> | pathname to file |
| OUTPUT ACCESS NAME: | <acnm> | output device or file |
| RECORD NUMBER: | {<constant exp> / 0} | sector offset-required if 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 checksum 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.XD
     OUTPUT ACCESS NAME: LP01
          RECORD NUMBER: 0
             FIRST WORD: 0
      VERIFICATION DATA: 0
                   DATA: 100
               CHECKSUM: 100
[ ]
```

## 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

```
                                 {<int>   }
                      RUN ID:    { previous}
SYMBOL TABLE OBJECT FILE:        <acnm>

                                 {Y}
           990/12 OBJECT CODE?:  {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
      FOR:   25                 Begin simulation of program
     FROM:   .IDTNAM.BEGIN      IDTNAM at location BEGIN
       TO:   . . END            halting after simulating the
                                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 which does not change the value in memory is *not* a memory alteration. The breakpoints set with this command are only valid during a Simulate command. Breakpoints, in this case, are conditions which stop execution but allow execution to be resumed on an operator command (either RST or 'F3' function key). Each simulated breakpoint is assigned a number which is displayed at the completion of the "ASB" command. When a breakpoint occurs during simulation, a panel and the breakpoint number is 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): | <A> | Alteration (memory) |
| | <C> | CRU Access |
| | P | PC Value |
| | <R> | Reference (memory) |
| | <S> | ST Value |
| | <X> | 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. Display the program counter (PC) if the breakpoint is taken. Count defaults to 1. |
| ON (A, C, P, R, S, X): | A | |
| FROM: | 6 | |
| 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. Display location TABLE in IDTNAM if the breakpoint occurs. |
| ON (A, C, P, R, S, X): | P | |
| 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 "FOR" and "TO" 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 returning to the command mode.

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

```
[] LSB
LIST SIMULATED BREAKPOINTS
 1  TYPE=P   FROM=>000A THRU=>0064   COUNT=>000A REMAINING=>000A   DISPLAY=>0021
 2       X        >0002       >000D        >0006            >0006          >0000
 3       S        >0016       >0038        >0058            >0058          >0009
 4       C        >0016       >0078        >0037            >0037          >0000
 5       A        >0016       >2710        >00C8            >00C8          >0000
 6       R        >0000       >2710        >002C            >002C          >0000
[]
```

**Figure 7-2. Display of Simulated Breakpoints**

7.4.7 **QD – QUIT DEBUG.** This command takes a task out of debug (or "controlled") mode. The user has the option of killing the task at this point. 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" or "proceed from breakpoint" commands (depending on whether the task is at a breakpoint) may 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 slower, 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 |
| DB | – | – | – | PANEL |
| PB | – | – | – | PANEL |
| DBP | – | – | – | PANEL |
| LB | BRKPTS | BRKPTS | BRKPTS | BRKPTS |
| HT | – | – | – | PANEL |
| RT | – | – | – | PANEL |
| MM | INTERACT | INTERACT | INTERACT | INTERACT PANEL |
| LM | TLF | TLF | TLF | TLF |
| FW | MSG OR TLF | MSG OR TLF | MSG OR TLF | MSG + PANEL OR TLF |
| FB | MSG OR TLF | MSG OR TLF | MSG OR TLF | MSG + PANEL OR TLF |
| SIR | INT REG | INT REG | PANEL | PANEL |
| MWR | INTERACT | INTERACT | INTERACT | INTERACT PANEL |
| SWR | WKSPC | WKSPC | PANEL | PANEL |
| SP | PANEL | PANEL | PANEL | PANEL |
| SV | VALUES | VALUES | VALUES | VALUES |
| XD | – | – | – | PANEL |
| ASB | – | – | – | BRKPT NO. + PANEL |
| DSB | – | – | – | PANEL |
| LSB | – | SIMULATED BRKPTS | – | SIMULATED BRKPTS |
| ST | – | TRAP # OR 'TIMEOUT' | – | TRAP # OR 'TIMEOUT'+PANEL |
| RST | – | TRAP # OR 'TIMEOUT' | – | TRAP # OR 'TIMEOUT'+PANEL |
| QD | – | – | – | – |

TLF = contents of terminal local file

PANEL = Debug Panel, figure 5-1

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, we are beginning with step two to modify the source code.

Before beginning the runthrough, power up the computer and the terminal, and log in at the terminal using the procedures described in Volume II.

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 syntax 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 directtory file. Enter the CFDIR command, which causes the following to be displayed:

```
CREATE DIRECTORY FILE

        PATHNAME:
     MAX ENTRIES:
```

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.

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 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:    0
  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 then 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:                Press RETURN
     MACRO LIBRARY PATHNAME:              Press RETURN
                PRINT WIDTH:  80          Press RETURN
               PAGE LENGTH:  60          Press RETURN
```

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 ERASE FIELD to clear the entry. Press RETURN.

3. Press F7 to enter the compose mode.

4. Press the unlabeled gray key to insert line.

5. Key the following and terminate each line by pressing the RETURN key:

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

6. 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)
```

7. 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 command (CFPRO, described in Volume II), 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 PROCEDURE?:   NO
```

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

To execute the task, use the Execute and Halt Task (XHT) command. Use of 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 prompt is 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.

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:  0.TSTSDS
          990/12 OBJECT CODE ?:
```

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.

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.

# 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:

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

[2]Includes: VDTs, KSR, 820, ASR, (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
- EBATCH - END BATCH EXECUTION
- 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

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
- CSF - CREATE SYSTEM FILES
- ENDKEY - END CFKEY SPECIFICATION
- KEY - CFKEY KEY SPECIFICATION

COPY COMMANDS
- AF - APPEND FILE
- CC - COPY/CONCATENATE
- CKS - COPY KEY TO SEQUENTIAL FILE
- CSM - COPY SEQUENTIAL MEDIA
- CSK - COPY SEQUENTIAL TO KEY
- DCOPY - DISK COPY/RESTORE UTILITY
- PF - PRINT FILE
- VB - VERIFY BACKUP (DIRECTORY)
- VC - VERIFY COPY (DIRECTORY)

DEBUG COMMANDS
- /BKPT - BREAKPOINT COMMANDS
- /GDEB - GENERAL 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 |
| RD | - RESTORE DIRECTORY |
| VB | - VERIFY BACKUP (DIRECTORY) |
| VC | - VERIFY COPY (DIRECTORY) |

DISK COMMANDS

| | |
|---|---|
| CKD | - CHECK DISK FOR CONSISTENCY |
| DCOPY | - DISK COPY/RESTORE UTILITY |
| INV | - INITIALIZE NEW VOLUME |
| IV | - INSTALL VOLUME |
| MAD | - MODIFY ABSOLUTE DISK |
| MADU | - MODIFY ALLOCATABLE DISK UNIT |
| MD | - MAP DISK |
| MVI | - MODIFY VOLUME INFORMATION |
| 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 |
| RS | - REPLACE STRING |

```
            SL          - SHOW 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

GENERAL DEBUG COMMANDS
            FB          - FIND BYTE
            FW          - FIND WORD
            HT          - HALT TASK
            KT          - KILL TASK
            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 |

MISCELLANEOUS COMMANDS

| | |
|---|---|
| CM | - CREATE MESSAGE |
| 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 |
| 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

| | |
|---|---|
| MDS | - MODIFY DEVICE STATUS |
| MFN | - MODIFY FILE NAME |
| MFP | - MODIFY FILE PROTECTION |
| MKL | - MODIFY KEY INDEXED FILE LOGGING |
| MLP | - MODIFY LUNO PROTECTION |
| MOE | - MODIFY OVERLAY ENTRY |
| MPE | - MODIFY PROCEDURE ENTRY |
| MPI | - MODIFY PROGRAM IMAGE |
| MRF | - MODIFY RELATIVE TO FILE |
| MTE | - MODIFY TASK ENTRY |
| STI | - SHOW TERMINAL INFORMATION |
| XE | - INITIATE TEXT EDITOR |

OVERLAY COMMANDS

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

PROGRAM DEVELOPMENT

| | |
|---|---|
| /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

| | |
|---|---|
| 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 |
| SPI | - SHOW PROGRAM IMAGE |

PROCEDURE COMMANDS

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

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

| | |
|---|---|
| CSF | - CREATE SYSTEM FILES |
| ISO | - INSTALL SYSTEM OVERLAY |
| MVI | - MODIFY VOLUME INFORMATION |
| XANAL | - ANALYZE DX10 CRASH FILE |
| XCU | - EXECUTE 2.2 CONVERSION UTILITY |
| XGEN | - EXECUTE GEN990-AUTO SYSGEN |

SHOW COMMANDS

| | |
|---|---|
| LD | - LIST DIRECTORY |
| LLR | - LIST LOGICAL RECORD |
| MD | - MAP DISC |
| 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

| | |
|---|---|
| XSM | - EXECUTE SORT/MERGE |
| XSMF | - SORT/MERGE FOREGROUND |

DX10 OPERATION

| | |
|---|---|
| /BGB | - BACKGROUND/BATCH COMMANDS |
| /COPY | - COPY COMMANDS |
| /DIR | - DIRECTORY COMMANDS |
| /DVICE | - DEVICES |
| /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 |
| 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 |
| STI | - SHOW TERMINAL INFORMATION |

```
TASK COMMANDS
          AT          - ACTIVATE TASK
          DT          - DELETE TASK
          HT          - HALT 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
          MTS         - MODIFY TERMINAL STATUS
          STI         - SHOW/MODIFY TERMINAL INFO

TX/DX CONVERSION
          DXTX        - DX10 FILE TO DISKETTE FILE
          TXCP        - CHANGE DISKETTE FILE PROTECT
          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
          INV         - INITIALIZE NEW VOLUME
          IV          - INSTALL VOLUME
          MVI         - MODIFY VOLUME INFORMATION
          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
          XANAL       - ANALYZE DX10 CRASH FILE
```