OS/8

TECO Reference Manual

Order No. AA-H608A-TA

ABSTRACT

This document describes the Text Editing and Correcting Program for OS/8 users.

SUPERSESSION/UPDATE INFORMATION: This manual supersedes the TECO chapter of the OS/8 Handbook (DEC-S8-OSHBA-A-D).

OPERATING SYSTEM AND VERSION: OS/8 V3D

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation • maynard, massachusetts

First Printing, March 1979

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

CONTENTS

Page

FIGURES

TABLES

PREFACE

TECO is a text editing program that runs under the PDP8 operating system. TECO may be used to edit any form of ASCII text such as program listings, manuscripts, correspondence and the like. Since TECO is a character-oriented editor rather than a line editor, text edited with TECO does not have line numbers associated with it, nor is it necessary to replace an entire line of text in order to change one character.

This manual is divided into four parts. Chapter 1, which contains basic information, introduces enough TECO commands to allow the novice TECO user to begin creating and editing text files after only a few hours of instruction. The introductory commands are sufficient for any editing application; however, they are less convenient, in most cases, than the advanced commands presented later.

Chapter 2 discusses the concepts underlying TECO. This discussion is relatively command independent; instead, the emphasis is on how TECO works.

Chapter 3 explains how to run TECO under the OS/8 monitor and presents an overview of many of the TECO commands. These commands are grouped by function.

Chapter 4 examines all TECO commands. The commands are listed in alphabetic order. Where applicable, commands appear together in the discussion or in the examples.

DOCUMENTATION SET FOR OS/8

OS/8 SYSTEM GENERATION NOTES (AA-H606A-TA)

The System Generation Notes provide the information you need to get a new OS/8 system running.

OS/8 SYSTEM REFERENCE MANUAL (AA-H607A-TA)

The System Reference Manual describes OS/8 system conventions, keyboard commands, and utility programs.

OS/8 TECO REFERENCE MANUAL (AA-H608A-TA)

The TECO Reference Manual describes the OS/8 version of this character-oriented text editing and correcting program.

OS/8 LANGUAGE REFERENCE MANUAL (AA-H609A-TA)

The Language Reference Manual describes all languages supported by OS/8, including BASIC, FORTRAN IV, and the PAL8 assembly language.

OS/8 ERROR MESSAGES (AA-H610A-TA)

This manual lists in alphabetical order all error messages generated by OS/8 system programs and languages.

CHAPTER 1

INTRODUCTORY COMMANDS

1.1 FUNDAMENTALS

TECO considers text to be any string of ASCII codes. Text is divided into units of pages, lines, and characters. A page of text consists of all the ASCII codes between two form feed characters, including the second form feed. A character is one ASCII code. Thus, every page of text contains one form feed character, which is the last character on the page. Every line of text contains one line feed, which is the last character on the line.

TECO maintains a text buffer in which text is stored. The buffer usually contains one page of text consisting of up to several thousand characters, but the terminating form feed character never appears in the buffer. TECO also maintains a buffer pointer. The pointer is simply a movable position indicator which is always located between two characters in the buffer, or before the first character in the buffer, or after the last character in the buffer. The pointer is never located on a character.

Line feed and form feed characters are inserted automatically by TECO. A line feed is automatically appended to every carriage return entered into the buffer, and a form feed is appended to the content of the buffer by certain output commands. Additional line feed and form feed characters may be entered into the buffer as text. If a form feed character is entered into the buffer, it will cause a page break upon output. That is, all text preceding the form feed will appear on one page, and the text following the form feed will appear on the next page.

Finally, TECO also maintains an input file and an output file, both of which are selected by the user through the use of file specification commands. The input file is any device from which text may be accepted. For example, if a block of text is stored in a disk file, the disk file would be specified as an input file when the text is edited.

The output file is any device on which edited text may be written. If the disk file mentioned above were to be edited, it could be written, for example, onto another disk file.

TECO functions as a "pipeline" editor. Text is read from the input file into the text buffer, and is written from the buffer onto the output file. Once a portion of text has been written to the output file, it cannot be accessed again without closing the output file and re-opening it as an input file.

TECO may be called from command level by typing:

```
.R TECO                                          For RT-11
```

(terminated with a carriage return). TECO will respond by printing an asterisk at the left margin to indicate that it is ready to accept user commands. At this point, one or more commands may be typed at the terminal, and TECO will execute the commands upon receipt of two consecutive ESCAPE characters. The ESCAPE is a non-printing character which may be labelled ESC, ALT, or PREFIX on some keyboards. TECO echoes a dollar sign ($) whenever an ESCAPE is received. The dollar sign character is used in examples throughout this manual to represent ESCAPE.

You can also summon TECO with the MAKE and TECO CCL commands.

To create a new file, use the MAKE command. The format is

```
MAKE filespec
```

where filespec is the name of the new file and the device you wish to store it on. TECO opens the file you specify for output.

To edit an existing file, use the TECO command. The format is

```
TECO filespec
```

where filespec is the name of an already existing file you wish to edit. TECO opens the file for input.

A TECO command consists of one or two characters which cause a specific operation to be performed. Some TECO commands may be preceded or followed by arguments. Arguments may be either numeric or textual. A numeric argument is simply an integer value which might be used to indicate such things as the number of times a command should be executed. A text argument is a string of ASCII characters which might be words of text, for example, or a file specification.

If a command requires a numeric argument, the numeric argument always precedes the command. If a command requires a text argument, the text argument always follows the command. All text arguments are terminated by a special character (usually an ESCAPE) which indicates to TECO that the next character typed will be the first character of a new command.

If more than one command is typed in response to the asterisk generated by TECO, the command string will be executed from left to right until either all commands have been executed or a command error is recognized. If an error is encountered, a message is printed and the rest of the command string is ignored. In any case, TECO prints another asterisk at the left margin as soon as it finishes execution of a command string, so that additional commands may be entered.

The extensive text editing capability of TECO implies a large and versatile command set. However, the novice TECO user will find that little more than a dozen basic commands suffice for most editing requirements. The following section introduces the basic TECO commands. The full command set will be described later in this manual.

1.2 FILE SELECTION COMMANDS

Input and output files may be specified to TECO in several ways. The following commands permit flexible file selection with TECO.

NOTE

All of the following file selection commands are shown with a general argument of "filespec". The actual contents of this filespec argument are operating system dependent.

TECO will accept input text from any input device in the operating system. The input device may be specified by means of an ER command terminated by an ESCAPE. The ER command causes TECO to open the specified file and print an error message if the file is not found. This command does not cause any portion of the file to be read into the text buffer, however. The following examples illustrate use of the ER command:

Command	Function
ERfilespec$	General form of the ER command where "filespec" is the designation of the input file. The command is terminated by an ESCAPE, which echoes as a dollar sign.

The following examples illustrate the use of the ER command.

ERPR:$	Prepare to read an input file from the paper tape reader.
ERPROG.MAC$	Prepare to read input file PROG.MAC from the system's default device.
ERRXA1:PROG.FOR$	Prepare to read input file PROG.FOR from RXA1:.

TECO will write output text onto any device in the operating system. The output file may be specified by means of an EW command terminated by an ESCAPE. If the output device is a file-structured device (e.g., a disk), a file name and extension (if any) must be supplied. If a file name is specified but no device is explicitly defined, the system's default device is assumed. The following examples illustrate use of the EW command, which has the same format as the ER command:

Command	Function
EWfilespec$	General form of the EW command where "filespec" is the designation of the output file. The command is terminated by an ESCAPE, which echoes as a dollar sign.
EWSYS:TEXT.LST$	Prepare to write output file TEXT.LST on SYS:.
EWPROG$	Prepare to write output file PROG on the system's default device.
EWRXA1:TEXT.LST$	Prepare to write output file TEXT.LST on RXA1:.

It is not always necessary to specify an input file. If the user desires to create a file without using any previously edited text as input, he may type commands to insert the necessary text directly into the text buffer from the keyboard and, at the end of each page, write the content of the buffer onto an output file. Since all input is supplied from the keyboard, no input file is necessary.

An output file is unnecessary if the user desires only to examine an input file, without making permanent changes or corrections. In this case, the content of the input file may be read into the text buffer page by page and examined at the terminal. Since all output is printed on the user terminal, no output file is needed.

When the user is finished editing a file, he may use the EX command to close out the file and exit from TECO. The current contents of the text buffer, and any portion of the input file that has not been read yet, are copied to the output file before TECO exits. Note that the EX command takes no arguments.

Command	Function
EX	Move the remainder of the current input file to the current output file, close the output file, then return to the monitor.

The following examples illustrate the use of the EX command.

ERINPUT.MAC$EWOUTPUT.MAC$$	Open an input file "INPUT.MAC" and open an output file named "OUTPUT.MAC". The double ESCAPE ($$) terminates the command string and causes the string to be executed. Note that the ESCAPE which terminates the EW command may be one of the two ESCAPEs which terminates the command string.
ERFILE.MAC$EWCOPY.MAC$EX$$	Open an input file "FILE.MAC" and open an output file named "COPY.MAC", then copy all the text in the input file to the output file, close the output file and exit from TECO.

TECO will only keep one input and one output file open and selected at a time. The current input file may be changed by simply using the ER command to specify a new file. The EX command or one of the other file closing commands presented later may be used to close the output file.

1.3 INPUT AND OUTPUT COMMANDS

The following commands permit pages of text to be read into the TECO text buffer from an input device or written from the buffer onto an output device. Once a page of text has been written onto the output file, it cannot be recalled into the text buffer unless the output file is closed and reopened as an input file.

Command	Function
Y	Clear the text buffer, then read the next page of the input file into the buffer. Since the Y command causes the contents of the text buffer to be lost, it is not permitted if an output file is open and there is text in the buffer.

Command	Function
P	Write the content of the text buffer onto the next page of the output file, then clear the buffer and read the next page of the input file into the buffer.
nP	Execute the P command n times, where n must be an integer in the range 1<=n<65535. If n is not specified, a value of 1 is assumed.

1.4 POINTER POSITIONING COMMANDS

The buffer pointer provides the only means of specifying the location within a block of text at which insertions, deletions or corrections are to be made. The following commands permit the buffer pointer to be moved to a position between any two adjacent characters in the buffer. TECO positions the pointer before the first character in the buffer after every Y or P command.

Command	Function
J	Move the pointer to the beginning of the buffer.
L	Move the pointer forward to a position between the next line feed and the first character of the next line. That is, advance the pointer to the beginning of the next line.
nL	Execute the L command n times, where n may be any integer. A positive value of n moves the pointer to the beginning of the nth line following the current pointer position. A negative value moves the pointer backward n lines and positions it at the beginning of the nth line preceding the current position. If n is zero, the pointer is moved to the beginning of the line on which it is currently positioned.
C	Advance the pointer forward across one character.
nC	Execute the C command n times, where n must be an integer in the range -32768<=n<=32767. A positive value of n moves the pointer forward across n characters (carriage return/line feed counts as two characters). A negative value of n moves the pointer backward across n characters. If n is zero, the pointer position is not changed.

These commands may be used to move the buffer pointer across any number of lines or characters in either direction; however, they will not move the pointer across a page boundary. If a C command attempts to move the pointer backward beyond the beginning of the buffer or forward past the end of the buffer, an error message is printed and the command is ignored.

If an L command attempts to exceed the page boundaries in this manner, the pointer is positioned at the boundary which would have been exceeded. Thus the command "-10000L" would position the pointer before the first character in the buffer. The command "10000L" would position the pointer after the last character in the buffer. No error message is printed in either case.

1.5 TYPE OUT COMMANDS

The following commands permit portions of the text in the buffer to be printed out for examination. These commands do not move the buffer pointer.

Command	Function
T	Type the content of the text buffer from the current position of the pointer through and including the next line feed character.
nT	Type n lines, where n must be an integer in the range -32768<=n<=32767. A positive value of n causes the n lines following the pointer to be typed. A negative value of n causes the n lines preceding the pointer to be typed. If n is zero, the content of the buffer from the beginning of the line on which the pointer is located up to the pointer is typed. This facilitates locating the buffer pointer.
HT	Type the entire content of the text buffer.
V	Type the current line. Equivalent to the sequence "0TT".

The 0T command is particularly useful for determining the position of the buffer pointer. This command should be used frequently to determine that the pointer is actually located where the user expects it to be.

1.6 TEXT MODIFICATION COMMANDS

The following commands permit the user to insert or delete text from the buffer.

Command	Function
Itext$	Where "text" is a string of ASCII characters terminated by an ESCAPE, which echoes as a dollar sign. The specified text is inserted into the buffer at the current position of the pointer, with the pointer positioned immediately after the last character of the insertion.
K	Delete the content of the text buffer from the current position of the pointer through and including the next line feed character.

Command	Function
nK	Execute the K command n times, where n may be any integer in the range -32768<=n<=32767. A positive value of n causes the n lines following the pointer to be deleted. A negative value of n causes the n lines preceding the pointer to be deleted. If n is zero, the content of the buffer from the beginning of the line on which the pointer is located up to the pointer is deleted.
HK	Delete the entire content of the text buffer.
D	Delete the character following the buffer pointer.
nD	Execute the D command n times, where n may be any integer in the range -32768<=n<=32767. A positive value of n causes the n characters following the pointer to be deleted. A negative value of n causes the n characters preceding the pointer to be deleted. If n is zero, the command is ignored.

Like the L command, the D and K commands may not execute across page boundaries. If a K command attempts to delete text up to and across the beginning or end of the buffer, text will be deleted only up to the buffer boundary and the pointer will be positioned at the boundary. No error message is printed. A D command attempting to delete text across a page boundary will produce an error and the command is ignored.

1.7 SEARCH COMMANDS

The following commands may be used to search for a specified string of characters which may occur somewhere in the input file. They cause the buffer pointer to be positioned immediately after the last character in the specified string, if found.

Command	Function
Stext$	Where "text" is a string of ASCII characters terminated with an ESCAPE which echoes as a dollar sign. This command searches the text buffer for the next occurrence of the specified character string following the current pointer position. If the string is found, the pointer is positioned after the last character on the string. If it is not found, the pointer is positioned immediately before the first character in the buffer and an error message is printed.
Ntext$	Performs the same function as the S command except that the search is continued across page boundaries, if necessary, until the character string is found or the end of the input file is reached. If the end of the input file is reached, an error message is printed and it is necessary to close the output file and reopen it as an input file before any further editing may be done on that file.

Both the S command and the N command begin searching for the specified character string at the current position of the pointer. Therefore, neither command will locate any occurrence of the character string which precedes the current pointer position, nor will they locate any character string which continues across a page boundary.

Both commands execute the search by attempting to match the command argument, character for character, with some portion of the buffer contents. If an N command reaches the end of the buffer without finding a match for its argument, it writes the content of the buffer onto the output file, clears the buffer, reads the next page of the input file into the buffer, and continues the search.

1.8 SUMMARY

At this point, the basic TECO commands have been introduced. Recall that TECO indicates it is ready to accept user commands by printing an asterisk (*). Once TECO has printed an asterisk, one or more commands may be typed at the terminal. Errors may be corrected by typing the DELETE key to delete characters. The DELETE key may be labeled DEL or RUBOUT on some keyboards. Each depression of the DELETE key deletes one character, beginning with the last character typed, and then prints the deleted character at the terminal. An entire command string may be deleted in this manner, if necessary. Once the correct command(s) have been entered, typing a double ESCAPE ($$) causes TECO to execute the command(s) in the order in which they were entered, and to print another asterisk so that additional commands may be typed. Note that this manner of operation is different from most other editors. In particular, carriage return has no special significance to TECO. Only the double ESCAPE forces execution of the command string.

If TECO encounters an erroneous command, it prints an error message and ignores the erroneous command as well as all commands which follow it. All error messages are of the form:

```
?XXX     Message
```

where XXX is an error code and the message is a self explanatory message relating to the command that generated the error. Every error message is followed by an asterisk at the left margin, indicating that TECO is ready to accept additional commands. If the first command entered after a TECO-generated error message is a single question mark character (?), TECO will print the erroneous command string up to and including the character which caused the error message. This facilitates locating errors in long command strings and determining how much of a command string was executed before the error was encountered.

At the conclusion of an editing job, the user may type EX to exit TECO. If an input and output file are open at the time the EX command is encountered, the remainder of the input file, including the current contents of the text buffer, is copied to the output file, and the output file is closed before TECO exits.

1.9 SAMPLE EDITING JOB

The following sample editing job is included to help the new user to achieve a greater understanding of the basic TECO commands. The entire terminal output from the editing run has been reproduced

intact, and numbers have been added in the left margin referencing the explanatory paragraphs which follow.

1. At this point, the user calls TECO into memory. TECO responds by printing an asterisk at the left margin. The user then enters an EW commmand, opening an output file called "FILE1.TXT" on DT1. There is no input file. Upon receipt of the double ESCAPE ($$), TECO created the designated output file, then printed another asterisk at the left margin.

2. The user then enters a command string consisting of two commands. The HK command clears the text buffer (not really necessary, since it was already empty), and the I command inserts 18 lines of text into the buffer, including 8 blank lines. TECO executes these commands upon receipt of the second double ESCAPE. At this point, the buffer pointer is positioned at the end of the buffer, following the last line feed character in the text. Note that the user made an error while typing the word "MASSACHUSETTS". He typed "MASA", then realized his mistake and struck the DELETE key once to delete the second "A". TECO echoes the deleted character. The user then types the correct character and continues the insertion.

3. The user then types -20L to move the pointer to the beginning of the buffer and SETTS$ to position the pointer immediately after the character string "ETTS", which terminates the word "MASSACHUSETTS". He then uses an I command to insert one space and a five-digit zip code. A second S command positions the pointer after the word "INFORMATION". The 2C command moves the pointer to the beginning of the next line (carriage return and line feed count as two characters), and the user deletes the words "PERTAINING TO" and replaces them with the word "REGARDING".

4. The user continues editing by positioning the pointer after the word "GUIDE". He then deletes this word and replaces it with the word "MANUAL". Finally, he searches for the word "SINCERELY", types 0T to determine that the pointer was correctly positioned between the Y and the comma which follows it, and types OK to delete everything on the line except the comma. He then inserts "VERY TRULY YOURS" in place of the word "SINCERELY". An HT command causes the edited text to be printed at the terminal.

5. The command string EX$$ causes the content of the buffer to be written onto the output file and closes the output file. The user then reenters TECO and reopens the file "FILE1.TXT" as an input file and specifies the line printer as an output file.

6. This command string reads the first (and only) page of "FILE1.TXT" into the buffer, deletes the first 5 lines, replaces them with a different address and salutation, then prints the content of the buffer on the terminal for verification and finally prints the new version of the letter onto the line printer. Note that the previous version of the letter still resides in file "FILE1.TXT" on DT1.

```
1< *EWDT1:FILE1.TXT$$
2< *HKIMR. JOHN P. JONES
  ! COMPUTER ELECTRONICS CORPORATION
  ! BOSTON, MASSACHUSETTS
  !
  ! DEAR MR. JONES:
  !
  ! I WAS PLEASED TO RECEIVE YOUR REQUEST FOR INFORMATION
  ! PERTAINING TO THE NEW TECO-11 TEXT EDITING AND CORRECTING
  ! PROGRAM.
  !
  ! ENCLOSED IS A COPY OF THE TECO-11 USER'S GUIDE, WHICH
  ! SHOULD ANSWER ALL OF YOUR QUESTIONS.
  !
  ! SINCERELY,
  !
  !
  !
  !
  ! $$
3< *-20LSETTS$I 02150$$
  ! *STION$2C13DIREGARDING$$
4< *SGUIDE$-5DIMANUAL$$
  ! *SELY$0T$$
  ! SINCERELY*0KIVERY TRULY YOURS$$
  ! *HT$$
  ! MR. JOHN P. JONES
  ! COMPUTER ELECTRONICS CORPORATION
  ! BOSTON, MASSACHUSETTS 02150
  !
  ! DEAR MR. JONES:
  !
  ! I WAS PLEASED TO RECEIVE YOUR REQUEST FOR INFORMATION
  ! REGARDING THE NEW TECO-11 TEXT EDITING AND CORRECTING
  ! PROGRAM.
  !
  ! ENCLOSED IS A COPY OF THE TECO-11 USER'S MANUAL, WHICH
  ! SHOULD ANSWER ALL OF YOUR QUESTIONS.
  !
  ! VERY TRULY YOURS,
  !
  !
  !
  !
  !
5< *EX$$
  ! (TECO is rerun, operating system dependent)
  ! *ERDT1:FILE1.TXT$EWLP:$$
6< *Y5KIMR. JAMES B. SMITH
  ! DATEK ASSOCIATES, INC.
  ! 122 MAIN STREET WEST
  ! AUSTIN, TEXAS
  !
  ! DEAR MR. SMITH:
  ! $$
  ! *HT$$
  ! MR. JAMES B. SMITH
  ! DATEK ASSOCIATES, INC.
  ! 122 MAIN STREET WEST
  ! AUSTIN, TEXAS
  !
  ! DEAR MR. SMITH:
  !
  ! I WAS PLEASED TO RECEIVE YOUR REQUEST FOR INFORMATION
```

```
! REGARDING THE NEW TECO-11 TEXT EDITING AND CORRECTING
! PROGRAM.
!
! ENCLOSED IS A COPY OF THE TECO-11 USER'S MANUAL, WHICH
! SHOULD ANSWER ALL OF YOUR QUESTIONS.
!
! VERY TRULY YOURS,
!
!
!
!
!
! *EX$$
```

CHAPTER 2

CONCEPTS

2.1 INTRODUCTION

This chapter presents information describing those concepts of TECO that are relatively independent of the commands you may enter. This information, where applicable, is applied to specific commands in Chapter 3.

2.2 MEMORY USE

TECO operates most efficiently on systems with at least 16K of memory. However, TECO will run on 8K or 12K systems, albeit slower.

In all configurations, TECO allocates sufficient storage to contain a 4000 (decimal)-character editing buffer. In 8K, TECO allocates a Q-register storage area which contains 2944 (decimal) characters. (See Section 2.14 for a discussion of Q-registers.) In 12K, TECO expands this by 2K. In 16K, TECO allocates space to store 1-line error messages. These messages exist only in systems with 16K or more memory and reside in a file internal to and accessible only by TECO.

In 8K and 12K systems, TECO uses a series of overlays. In 16K, these overlays are permanently resident.

The overlay system is:

Level	Mnemonic	Commands
Overlay 1	I/O	ER, EB, EW (initially resident)
Overlay 2	Quote	" Conditionals, O, and <>
Overlay 3	Error	Error messages
Overlay 4	Exit	EC, EF, EG, EK, EX
Overlay 5	Flags	EH, EO, ET, EU, =

You will be able to increase the efficiency of TECO in 8K and 12K configurations by grouping commands so as to minimize overlay swapping.

2.3 DATA FILES

You must specify the file from which TECO is to extract information and the file into which TECO places edited or examined information. When you enter an input command, TECO normally brings part of the input file into the editing buffer. (The editing buffer is discussed in Section 2.5.) An output command then tells TECO to write the contents of the editing buffer onto the output file.

NOTE

An output command is different from an exit command. Generally, an output command writes the contents of the editing buffer onto the output file, which remains open. An exit command may or may not write to an output file. In contrast, it normally closes the output file.

TECO can only process a file sequentially; that is, TECO can access the nth page in a file only after it reads the previous (n-1) pages. Also, you cannot reaccess the information TECO places in the output file until you close both the input and the output files. Then, and only then, can you declare the former output file to be the new input file. You may subsequently use an input command to bring the portion of a file you wish to reaccess into the editing buffer.

When using such hard-copy devices as card readers and paper-tape readers, you need only specify a device name to open a file for input or output. For disk and DECtape files, you must specify filenames as well as the device name. If you omit the device name, TECO assumes DSK:.

NOTE

If OS/8 is not configured for the devices you wish to use, consult the BUILD section of the OS/8 System Reference Manual for a listing of available devices and the OS/8 Software Support Manual for instructions on adding other device handlers to your configuration.

Filenames for file-structured devices consist of two parts: the first part, the filename proper, consists of from one to six alphanumeric characters; the second part, which is optional, is called the "extension." If present, the extension consists of either one or two alphanumeric characters. You must separate the filename from the extension by a period. For example,

```
RXA1:MYFILE.FT
```

is a file specification which designates the FORTRAN file called MMYFILE. This file resides on the floppy in drive 1.

If you do not enter an extension, CCL (the OS/8 Concise Command Language Utility Program) supplies .PA. If you do not want the file to have an extension, you must type a period (.) after the filename.

For example,

```
.TECO MYFILE.
```

is the command which retrieves the file MYFILE. with no extension from DSK: (the system device).

NOTE

The term "file specification" is abbreviated as filespec throughout this manual. For a complete discussion of file specifications, see the OS/8 System Reference Manual.

When you use floppies (diskettes), DIGITAL strongly recommends that you keep your files small. If a file is large and you store other files on the device, you may have difficulties with some of TECO's I/O commands because there may be insufficient room to write the file onto the device. Consequently, if you are having problems, you should segment the file using EF commands (see Section 4.14).

2.4 CHARACTER SET

TECO uses the entire ASCII character set. ASCII characters work on two levels in TECO: the command level and the data level.

You can use every ASCII character from control-A (decimal value 01) through delete (decimal value 127) as data in TECO. They can all be read, written, and inserted. (The ASCII character set is listed in Appendix A.)

The only character that is not completely legal as data is the null character (decimal value 0). If you insert a null character, TECO writes it to the output file. However, when TECO reads a null character into the editing buffer (through the reopening of the file as input), TECO removes it from the file. The null character echoes as ^@ on the terminal.

TECO interprets many of the ASCII characters as commands. When you use them as commands, the lower-case characters have the same meaning as their upper-case equivalents.

NOTE

In this manual, commands always appear as capital letters.

Appendix C is a list of TECO commands.

2.4.1 Special Characters

Because of their use as special immediate-action commands (e.g., monitor commands or erasing commands), you may only implicitly enter certain characters into a command string. All of them, however, are legal as data (except the null character) and you may insert them

using commands designed for this purpose. For example, you may insert a <^C>, which has a decimal ASCII value of 3, into the editing buffer using a 3I$ command, where 3 is the argument to the I$ command.

The characters to which this restriction applies are called "special characters" and are listed in Table 2-1.

Table 2-1
Special Characters

Command	Decimal ASCII Value	Function
<CTRL/C>	3	A monitor Command
<CTRL/G><sp>	7-32	A retyping command (causes TECO to retype line showing editorial corrections)
<CTRL/G>*	7-42	A typing command (types current command string)
<CTRL/G><^G>	7-7	An erasing command (erases command string)
<CTRL/O>	15	A monitor command (ends terminal output if TECO is printing)
<CTRL/U>	21	An erasing command (deletes current line)
DELETE	127	An erasing command (deletes last character typed)
ESCAPE	27	Standard text argument terminator (two successive ESCAPEs terminate a command string)

2.4.2 Control Characters

You can enter control characters (ASCII decimal values 0-31) by holding down the CONTROL key while typing a character key.

NOTE

TECO prints a control character as a circumflex followed by the character that you type to produce the control character. For example, <CTRL/A> prints as ^A.

On some terminals, the circumflex prints as an up-arrow. For example, <CONTROL/A> prints as ^A.

You can enter many of the control characters into command strings by typing a circumflex (or up-arrow) followed by the desired character.

For example, <CONTROL/D> is equivalent to <^D>. You should use this method only when typing the control character as a command and not as text. That is, in an alphanumeric argument, TECO interprets the circumflex-letter combination as two characters. In a command string, however, TECO interprets the circumflex-letter combination as one of the control characters.

NOTE

Throughout this manual, entering a control character as a circumflex followed by a letter is called the circumflex construction.

2.4.3 Carriage Control Functions and Responses

A few of the control characters have special echoes to the terminal, namely, bell, tab (<CONTROL/I>), line feed, form feed (<CONTROL/L>), and carriage return. These characters echo on your terminal by performing their particular action.

When you type a carriage return, the TECO monitor automatically generates a line feed following it. The echo to the carriage return type-in is a carriage return followed by a line feed. As TECO appends a line feed character to every carriage return you type, you must type a carriage return followed by a <DELETE> to enter a carriage return without a line feed. However, if you enter a carriage return as <CONTROL/J>, TECO does not append a line feed to it.

Except as text arguments, TECO ignores the characters "carriage return" and "line feed" in command strings. TECO also ignores spaces within command strings. Consequently, you may insert spaces, carriage returns, and line feeds into command strings to improve their readability.

ESCAPE echoes and prints as a dollar sign ($).

2.5 DATA FORMAT -- LINES AND PAGES

TECO lines can be of any length, subject to the limitations of available memory. The characters that define the end of a line are the line feed, the vertical tab, and the form feed. (These are the characters which cause vertical movement.) The end of the editing buffer may also be an end-of-line character if no line feed is present. When TECO counts lines, it does so by counting these end-of-line characters.

An end-of-line character belongs to the line that it terminates.

A text does not have to contain end-of-line characters; however, if you do not include them, the TECO commands that are line-dependent will not be useful. DIGITAL recommends that you enter data as a series of lines.

Pages are delimited in TECO by form feed characters. Thus, a page of text consists of all the ASCII code between two form feed characters. A form feed character does not belong to either of the two pages that it separates. Two consecutive form feed characters delimit a null page.

A form feed character at the end of a file has no effect in TECO. Thus, you may omit it.

TECO operates most efficiently if you divide files into pages of approximately fifty or fewer lines. You may edit files with longer pages or files containing no form feeds, but this process requires more care because you may use more memory than is available. (See Section 4.2 for a list describing how TECO determines when to stop reading characters into the editing buffer.)

2.6 EDITING BUFFER

You can edit a program by:

1. Reading text into the editing buffer
2. Making changes to the text in this buffer
3. Writing the modified text from the editing buffer out to the new file.

This editing buffer is a block of memory within TECO. When reading or inserting data, TECO places it into the buffer, and the data remains there while you are editing it. It leaves the buffer only when you type an output command.

The buffer usually contains one page of text consisting of up to 4000 characters. However, the terminating form feed character never appears in the buffer. If TECO terminates input to the editing buffer because it reads a form feed character, TECO sets the ^E form feed flag to -1.

TECO normally passes data into and out of the buffer one page at a time.

2.7 BUFFER POINTER

The "buffer pointer" construct is fundamental to TECO because the place indicated by this pointer determines the effect of many editing commands. For example, character insertion and deletion always take place at this position.

The buffer pointer is a position indicator. It always points to a position between two characters in the editing buffer (unless it points to a position immediately before or after the buffer's contents).

Either you or TECO can move this pointer to any position in the buffer. It cannot, however, point to a position beyond the boundaries of the buffer; that is, you cannot move it farther forward than the position immediately preceding the first character in the buffer or farther backward than the position following the last character in the buffer.

Although TECO must sequentially process pages in a file, it can address characters randomly within a page.

2.8 GENERAL COMMAND STRING SYNTAX

You can type TECO commands by typing a command string (a command string is a sequence of commands, one immediately after the other, and concluding with two consecutive ESCAPEs).

The only formal delimiter in TECO is the ESCAPE character, which terminates some alphanumeric arguments (see Section 2.9.1). However, if you use the @ command modifier (see Section 4.68) to a command accepting a textual argument, you may specify any delimiter you wish. For example, Itext$ and @I/text/ (where / is an arbitrary delimiter) are equivalent.

The action that tells TECO to begin the execution of a command string is the double ESCAPE ($$). The double ESCAPE is the only command string terminator that TECO accepts.

You may type a command string after TECO prints a prompting asterisk. An example of a command string is:

```
*YIheading$2K4DNtag$2LT$$
```

Execution of this command string begins only after you type two consecutive ESCAPEs. TECO indicates that it is beginning execution by printing a carriage return/line feed. At this time, TECO starts executing each command in the command string in turn, starting at the command immediately following the asterisk. When TECO has executed all commands in the string, it prints another asterisk to indicate that it is ready to accept another command string.

NOTE

If you enter a command which causes TECO to type out information, the asterisk prints immediately after the type-out rather than at the left margin if the sequence does not end with a carriage return/line feed.

If TECO cannot execute a command, execution of the command string stops at that point and TECO prints an error message. However, because TECO executes one operation before proceeding to the next, it executes all commands preceding an error. Thus, TECO does not execute the erroneous command (and any command that may follow). (Errors, error messages, and recovery techniques are discussed in Section 3.16.)

The only exceptions to the rule that TECO does not execute commands until you enter the double ESCAPE are:

1. The special characters in Table 2-1
2. The ? command, which types out all commands executed before an error occurred.
3. The <^S> command, which stores a command string in Q-register Z.

2.9 ARGUMENTS

If a command requires a numeric argument, the argument always precedes the command. If a command requires a text argument, the argument always follows the command.

2.9.1 Text Arguments

Text arguments are character strings that follow a command. Some examples of text arguments are: search strings, command string tags, and file specifications.

Text that you place into the buffer is usually as the argument to an insertion string.

A text argument always follows the command to which it applies. Commands that take text arguments require that you terminate the argument with an ESCAPE; however, the @ command modifier allows you to choose an alternate delimiter to a text argument.

TECO uses the ESCAPE terminating a text argument as one of the two ESCAPEs necessary to terminate a command string.

Example:

*ITEXT$STEXT1$$	An ESCAPE terminates the alphanumeric argument TEXT. An ESCAPE also terminates the second argument TEST; however, TECO also uses this as one of the ESCAPEs terminating the command string.

All ASCII characters are legal as text arguments. However, the special characters listed in Table 2-1 may only be indirectly inserted (for example, the nI$ command can insert any character).

2.9.2 Numeric Arguments

Numeric arguments always precede the command to which they apply. In some cases, you need to enter only a single numeric argument; in others, the command may require a pair of arguments.

NOTE

No TECO command accepts more than two numeric arguments.

In many cases, numeric arguments must be positive; however, some commands allow a numeric argument to be negative or zero. The number and type of numeric arguments that you may use with a command are described in Chapter 4.

When you use a numeric argument to specify a buffer position, the number represents the number of characters in the buffer to the left of that position. Thus, the nth position means the position to the right of the nth character in the buffer, that is, between the nth and (n+1)st characters.

Paired numeric arguments are always buffer-positioning arguments. When you use two numeric arguments, separate them with a comma. Such

a pair indicates all the characters in the editing buffer that lie between the two buffer positions indexed by the two arguments. This definition is precise because the term "buffer position" always points to a position before or after a given character, not "on" or "at" the character.

Example:

12,20	This argument specifies the 13th through the 20th characters in the buffer. These characters are specified because the 12 indicates the position between the 12th and 13th characters, and the 20 indicates the position between the 20th and 21st characters.

You may construct a numeric operator from arithmetic/logical combinations. Table 2-2 lists the TECO operators.

Table 2-2
Arithmetic/Logical Operators

+	Ignored, if used before the first term in a string	+2=2
+	Addition	5+6=11
-	Negation, if used before the first term in a string	-2=-2
-	Subtraction	8-2=6
*	Multiplication	8*2=16
/	Integer divide (and drop the remainder)	8/2=4 8/3=2
&	Bitwise logical AND of the binary representations of two terms	12&10=8
#	Bitwise logical OR of the binary representations of two terms	12#10=14

When you use more than one arithmetic/logical operation in a single numeric argument, TECO performs the operations from left to right.

You may override this sequence through the use of parentheses (). TECO evaluates all operations within parentheses before those outside the parentheses. TECO also accepts nested parentheses.

TECO numbers are normally decimal integers. However, you may change the radix to octal by typing ^O. TECO will then remain in octal radix unless you change it by entering a ^D command.

Example:

```
^0177=127 (decimal)
3*^O10=30 (octal)
2+3*4=20
2+(3*4)=14
2+(3*(16/(e-1))/2+(2*5)=24
2&(3#5)#16=18
-((2+(3*4)-1&(6+8))/2)=-6
```

You may use the arithmetic/logical operators and parentheses to form one or both of the numeric arguments in a pair.

Example:

```
260-(3*42),250+(77/3)P
```

is identical to:

```
144,275P
```

2.9.3 Commands That Return a Value

Generally speaking, there are two main categories of TECO commands: 1) those that perform some operation, such as inserting text; and 2) those that "return" a value, such as the number of characters in the editing buffer. (Some commands do both.)

A command is said to return a value if the command both causes TECO to calculate the current value of some quantity and then takes on this value. You may subsequently use this value as an argument to the next command in the command string. If the command does not take a numeric argument, TECO ignores the number. For example, the ^N flag is equal to -1 when TECO reads in the end of a file. Otherwise, it is equal to 0. Thus, the command string segment ...^N"E... causes the value of the ^N flag to be an argument for the "E (execute if equal to o) command. If it equals 0, TECO executes the commands immediately after the E. Consequently, using such a command is equivalent to typing the particular number that the command returns as a value, except that the value is not usually known in advance.

You may use commands that return values with each other and with explicit numbers. All the same rules apply. Each command that returns a value has all the properties of a number that you explicitly type.

If you concatenate commands that return values with each other (or with digits), the value that TECO returns is not defined. For example, you should not use such arguments as:

```
ZZ
Z48
-2Z
3+ZZ
```

2.10 SUPER TECO

Inevitably, you will lose valuable files or directories because of your error, a hardware error, or an operating system error. To retrieve files whose names have been accidentally deleted from the directory of a device, you can create a version of TECO called SUPER TECO by typing a 2-word patch using ODT. (See the OS/8 System Reference Manual for a complete description of ODT.) This patch is:

```
.GET SYS TECO

.ODT

2034/7420 7610
2117/7450 7410
^C
.SAVE SYS STECO
```

To use STECO, mount the device on which you want to retrieve the file, and then type

```
.R STECO
*ERDEV:$$
*_STRING$$
```

where STRING is a part of the first page of the desired file (e.g., the title line). STECO then searches the entire device for the first occurrence of STRING. Because the device may contain older or similar copies of the desired file, examine the text following each occurrence of STRING. If the file is not the one you want, continue searching until you find the correct file. Once you find the file, type:

```
*ERDEV:$EWDEV2:FILE$$
*N<STRING$NENDSTRING$EC$$
```

where

n is the number of times you had to search for STRING

ENDSTRING is a string at the end of the file.

This command operation retrieves your file and copies it onto another device.

If, as sometimes occurs, meaningless characters precede the first good line of your file, simply delete them.

CHAPTER 3

USING TECO

3.1 INTRODUCTION

This chapter describes how you call TECO from the OS/8 monitor and presents the commands that TECO recognizes. Because these commands are grouped by function, a command may be mentioned in more than one context.

3.2 CALLING TECO

You can call (or load) TECO by typing one of four commands to the monitor.

You may load TECO whenever the monitor prints a dot. The dot indicates that TECO is waiting for a new command.

If you do not type an extension after a filename, CCL (the OS/8 Concise Command Language Utility Program) assumes .PA.

3.2.1 R TECO Command

You can call TECO by typing:

```
.R TECO
*
```

This command brings TECO into memory. It does not automatically initialize any particular device or file for input or output.

After OS/8 loads TECO into memory, TECO prints an asterisk indicating it is ready to receive a command. This state, in which TECO waits for a command string type-in, is the command mode. At this point, you can enter a file specification command such as EB, ER, or EW.

If you are creating a file, you need only type an EWfilespec command to create the file at the keyboard.

3.2.2 General Purpose Initialization Commands

TECO is used chiefly to create new files and to edit existing files. These two functions are so common that there are special OS/8 monitor commands for calling TECO. These commands are MAKE and TECO.

You can follow both TECO and MAKE commands with a file specification. If you do not use a filespec with a TECO command, CCL uses the name of the last ASCII file specified in a TECO or MAKE command. If no filename is given in a TECO command and no previous MAKE or TECO command has been entered on that day, CCL prints the error message "BAD RECOLLECTION". CCL also displays this message if you have not typed an OS/8 DATE command. (See the OS/8 System Reference Manual for a description of the DATE command.)

NOTE

TECO will remember a filespec only between bootstraps or during the same date of operations as entered in an OS/8 monitor DATE command.

3.2.2.1 **MAKE Command** - You type:

```
.MAKE filespec
```

to instruct TECO to create a new file. The filename.ext parameter of the filespec is the name that you give the new file. The dev: parameter of the filespec is the device on which the file will be written; it can be any output device. If you omit dev: from the filespec, TECO assumes DSK:.

The command

```
.MAKE filespec
```

is equivalent to:

```
.R TECO
*EWfilespec$$
```

The MAKE command opens an output file and gives it the name you specify. Once TECO opens a file, you can create it with insert and output commands.

You should choose the filename carefully when using the MAKE command. If a file is already on the device with the same name, the MAKE command overwirtes the old file. However, CCL prints the warning message %SUPERSEDING. If you do not wish to supersede the file, you must type <^C> to return to the monitor.

After the OS/8 monitor loads TECO, TECO prints an asterisk indicating its readiness to receive a command string. Usually, you would then create the file by using an insert command.

Example:

```
.MAKE EARNNG.FT
*
```

This command calls TECO for the creation of a FORTRAN file named EARNNG.FT.

```
.MAKE MYFILE.MA
*
.
.
*EX$$
.MAKE
%SUPERSEDING
```

TECO is called and the output file MYFILE.MA on DSK: IS CREATED. AFTER YOU CLOSE THE FILE WITH THE EX command, the monitor accepts the second MAKE. This second command overwrites the first unless you type <^C>.

```
.MAKE MYFILE.MA
*Ipage of text$$
*PI2nd page of text$$
.
.
.
*PIlast page of text$$
*EC$$
```

This is the way in which you create most files.

3.2.2.2 **TECO Command** - Use the command

```
.TECO filespec
```

to call TECO for editing an existing file on a file-structured device. TECO interprets the filespec in the same way as it does for the MAKE command, except that the device must be a file-structured device (disk, diskette, or DECtape).

The filename and extension must be exactly the same as those of the file that you are editing.

The TECO command opens the specified file for input and reads in the first page of that file. After TECO outputs the new version, it renames the original (input) version of the file "filename.BK", and gives the new version the name of the original file. This operation is identical to that used for the EB command; that is,

```
TECO filespec
```

is equivalent to:

```
*EBfilespec$Y$$
```

(See Section 4.12.) Also,

```
.TECO filespec2<filespec1
```

is equivalent to:

```
.R TECO
*ERfilespec1$EWfilespec2$Y$$
```

You cannot use the TECO command with a file having the extension BK.

If a TECO filespec2<filespec1 command would cause TECO to overwirte an existing file, CCL prints the message %SUPERSEDING. If you do not wish TECO to overwrite the file, type a <^C>.

If you have entered a TECO filespec command, CCL remembers the filespec the next time you type a TECO command. If you have entered TECO filespec2<filespec1, CCL remembers only filespec2.

After the OS/8 monitor loads TECO into memory, TECO prints a prompting asterisk to indicate its readiness to receive a command string.

Example:

.TECO LIB40.MA	This command initializes TECO for editing the existing file LIB40.MA. At the completion of editing, TECO automatically changes the name of the original version of LIB40.MA to LIB40.BK and gives the name LIB40.MA to the new version.
.TECO	This command initializes TECO for editing the disk file last referenced in a TECO or MAKE command. If the last file referenced was LIB40.MA, then this would be the file initialized.

3.2.3 MUNG Command

The MUNG Command performs the actions defined in a TECO macro. The MUNG command can take two forms, the first of which:

```
.MUNG filespec
```

is equivalent to:

```
.R TECO
*ERfilespec$Y HXY HK MY$$
```

The second form:

```
.MUNG filespec,text
```

is equivalent to:

```
.R TECO
*ERfilespec$Y HXY HK Itext$MY$$
```

When you enter a MUNG command, TECO places the first page of the file specified in filespec into Q-register Y. If you omit the extension in the filespec, CCL assumes TE.

If you enter a text argument, TECO places it into the text buffer.

Example:

The following demonstrates one way you could use the MUNG command. The action that the macro performs is the formatting of a file into 50-line page segments. If the macro

```
J IEB$ ZJ 27I$ HXA HK MA Y
<!ST!^N;50S
     $"FA OST$ '0,.P0,.K>EX
```

is in the file FIFTY and if you wish to format the file TEXT.TX into 50-line pages, then the command

```
.MUNG FIFTY,TEXT.TX
```

causes the operation to be performed. The following is a description of how this macro operates (see Section 4.4 for a description of the formatting portion of this macro):

1. The MUNG command performs an ER on file FIFTY.
2. The Y command brings the text into the buffer.
3. HXY stores the page in Q-register Y and the HK kills the page.
4. MY places the string TEXT.TX into the buffer.
5. Control now passes to the macro, in which the JIEB$ command string inserts an EB before TEXT.TX and ZJ27I$ inserts an ESCAPE after it.
6. HXA inserts the string EBTEXT.TX$ into Q-register A.
7. HK kills the buffer.
8. The MA command executes the EBTEXT.TK command.
9. The remainder of the macro now formats the file TEXT.TX into 50-line pages.

3.3 FILE SELECTION COMMANDS

File selection is the specifying of both the device from which input is to be taken and the device to which output is to go. In the case of file-structured devices, you must specify a filename as well as the device.

If you want only to create a file or to edit an existing file-structured device, you may use either of the previously described loading commands, that is,

```
.MAKE filespec
```

or

```
.TECO filespec
```

When you load TECO with the R TECO command, one or more of the file selection commands must be used.

The file specification commands are:

```
EBfilespec      Edit creating backup
ERfilespec      Edit read from
EWfilespec      Edit write to
```

3.4 INPUT COMMANDS

You can enter input commands to bring data from a previously opened input file into the editing buffer. However, you must use an input command only after entering an ER command (or its equivalent). Input always starts at the beginning of the input file. Successive input commands bring other parts of the input file into the editing buffer.

The amount of data TECO brings into the buffer depends on the buffer size, the input commands, and the data itself.

After TECO has written a page of text onto the output file, you cannot recall the page into the text buffer unless you close the output file and then reopen it as an input file.

The input commands are:

A	Append next page to end of buffer
P	Page input and output
Y	Yank, bring next page into buffer

The N, FN, and _ search commands may also input data.

3.5 BUFFER POSITION NUMERIC ARGUMENTS

In many cases, you may use numeric arguments to specify buffer positions. Because such arguments tend to be large and not easily countable, the buffer positions which you may often use as numeric arguments are represented by special characters. You may also use these characters as values in arithmetic/logical operations. The special characters are:

B	Beginning of buffer, always zero
H	Whole buffer, always equal to B,Z
Z	End of buffer
.	Current position in buffer

3.6 BUFFER POINTER POSITIONING COMMANDS

You may use a buffer pointer positioning command to move the buffer pointer. The buffer pointer positioning commands are:

C	Continue forward movement
J	Jump to character
L	Line move
R	Reverse character movement

In addition to these commands, the search (S, N, and _) commands and search/replace commands (FS and FN) also move the buffer pointer.

3.7 TEXT TYPE-OUT COMMANDS

You can type text type-out commands to display information in the buffer, to type messages while executing a TECO macro, or to suppress the typing of data. The text type-out commands are:

^A	Type following text
T	Type text
=	Type-out value equal to expression
\	Type ASCII value of number

The two commands <^S> and <^Q> are not type-out commands; instead, they allow you to stop the display and later resume at the place where you stop. The command <^O> tells TECO to omit printing the remainder of the output on the terminal.

TECO also contains the EU flag which, depending on its value, can flag upper- or lower-case output.

3.8 DELETION COMMANDS

The deletion commands remove characters from the editing buffer. The deletion commands are:

D	Delete character
K	Kill line

The K command preceded by a single numeric argument is a line-oriented deletion command. However, if you enter it with a pair of numeric arguments, it becomes character-oriented.

A search/replace command (FS or FN) with a null second text argument also deletes text from the buffer.

3.9 INSERTION COMMANDS

The insertion commands place characters into the editing buffer. The insertion commands are:

I	Insert
nI$	Insert number n into buffer
<^I>	Insert and include tab
<TAB>	Equivalent to <^I>
n\	Insert a number as individual characters into the buffer

3.10 OUTPUT AND EXIT COMMANDS

Output commands transfer data from the editing buffer to the output file. An exit command generally transfers data to the output file. However, output commands also terminate a TECO job and return to the OS/8 monitor. One exit command, <^C>, jumps to the OS/8 monitor without performing output.

The four output commands are:

EC	Exit and close after output of file
EF	End file at page
P	Page output and input next
PW	Page write and append form feed

The three exit commands are:

EG	Exit and go to executing program
EX	Exit to OS/8 monitor after output of file
^C	Jump to OS/8 monitor

3.11 SEARCH COMMANDS

In many cases, you may find that the simplest way to reposition the buffer pointer is to use a character string search. A search command causes TECO to scan through the text until a specified string of characters is found, and then to position the pointer at the end of the string.

The string of characters for which you are searching is the text argument following the search command. This search string can be from 1 to 31 characters long.

If TECO finds an exact match for the search string in the text, it positions the buffer pointer immediately after the last character in this match. If TECO cannot find a match for the string, it positions the pointer at the beginning of the buffer and notifies you of the failure if you had typed an S or FS. If you use an N or FN search, TECO places the buffer pointer after all text in the file. The entire file will have then been written to the output file.

If you type a colon modifier to a search command, then the search will return a -1 if the search succeeds or a 0 if the search fails.

All searches begin at the current position of the buffer pointer.

If you do not include a text argument with a search command (for example, S$$ or N$$), TECO executes the search using the last previous search command argument.

The search commands are:

FN	Search all pages until found, then replace
FS	Search current page until found, then replace
N	Search all pages until found
S	Search current page until found
_	Search, but discard pages until found (that is, no output)

Four match control characters are allowed in a search string:

<^N>	Match any character except the one following
<^Q>	Quote the next character (that is, accept command as a character)
<^S>	Match on separator
<^X>	Match any character

3.12 ITERATION COMMANDS

An iteration command enumerates how many times TECO will execute a command string or it may determine whether a command within an iteration loop has failed so that the looping may end. The iteration commands are:

<...>	Loop
;	Exit loop upon search failure
n;	Exit loop if n is negative

3.13 FLOW CONTROL COMMANDS

TECO contains commands that enable you to write editing and character manipulation programs. The iteration command <...> is a specialized example. In addition, TECO has an unconditional branch command, O, and a set of conditional execution commands. The flow control commands are:

O	Goto
"C	Execute if alphanumeric character
"E	Execute if equal
"F	Execute if false
"G	Execute if greater than zero
"L	Execute if less than zero
"N	Execute if nonzero
"R	Execute if alphanumeric range
"S	Execute if successful
"T	Execute if true
"U	Execute if unsuccessful
"<	Execute if less than zero
">	Execute if greater than zero

3.14 Q-REGISTERS

Q-registers are essentially data storage areas. These registers are the means by which you perform programmed editing and text block movement. Data that you store in Q-registers is not disturbed by the flow of data into and out of the editing buffer. Thus, you may preserve this data throughout an entire TECO job, and also retrieve or change this data at any time.

There are 36 Q-registers, each of which has a single character name. The name is either one of the digits 0 through 9 or one of the letters A through Z. Each Q-register is divided into two storage areas: The first area stores numbers and the second stores strings.

You can store a single positive, negative, or zero decimal integer in the range -4095<n<4095 in a Q-register. You can also increment, test, or recall numbers in a Q-register. Hence, you may use Q-registers as switches and counters, as well as data-save areas.

The Q-registers can hold from 3000 to 5000 characters, depending upon your configuration. However, no single Q-register can hold more than 2047 characters. You can store two types of character strings: ordinary text and TECO command strings. TECO stores both identically. The use you make of them determines if a string will be used as a text or as a macro. (A macro is a series of TECO commands that are stored in a Q-register and can be executed upon command.)

Text that you store in a Q-register is copied into the Q-register from the editing buffer without destroying the copy in the buffer.

Storing text in a Q-register is useful for such tasks as making many copies of a given segment of text throughout a file without retyping it each time, moving a block of text from one position to another in a file, or moving a block of text to another file.

You may store textual data representing TECO command strings in Q-registers. You can then execute such a command string many times throughout an editing job, much like calling a subroutine. You may edit such command strings as you would any other text.

The Q-register commands are:

Gq	Get text
Mq	Execute macro
Qq	Return Q-register number
Uq	Put number in Q-register
^Uq	Put text immediately into Q-register
Xq	Extract text
%q	Postfix, that is, return number incremented, then store

3.15 ERASING COMMANDS

If you make an error while typing a command string, you may correct the error (if you terminate the command string with a double ESCAPE) by using the DELETE key to individually delete characters. Other erasing commands that you may find useful are: <^G><^G>, which erases the entire command; and <^U>, which erases the last line typed.

Typing <^G><sp> will cause TECO to echo the last line typed. <^G>* will echo the entire command. The <^G><sp> and <^G>* commands do not move the buffer pointer and they do not close the command storage register. Consequently, after you type one of these commands, you may continue typing as if you had not entered one of these commands.

3.16 ERROR MESSAGES

TECO error messages are listed in Appendix B.

When TECO encounters an illegal command or a command that it cannot execute, it prints an error message on the terminal. An error message may consist of two parts: The first is a question mark followed by a 3-letter mnemonic code for the error message, and the second is a brief, 1-line statement of the error condition. The 1-line message is available only on systems with a configuration of at least 16K.

TECO normally prints both parts of the error message. The EH command explaains how you may suppress the 1-line statement.

When an error occurs:

1. The command to which it refers is not executed
2. The remainder of the command string is ignored
3. TECO prints an error message
4. TECO returns to command mode.

After TECO prints an error message, you may type the special command ?. TECO then prints all commands that it has executed in the command string.

NOTE

This command displays the executed portion of the command string immediately after an error has occurred. If you type any other command following the printing of an error message, TECO assumes that you are typing a new command string, and you lose the ability to use the ? command for this error.

Also note that the <^S> command is often useful after an error occurs.

After TECO finishes executing a command string (or if you abort a command string by means of the <^G><^G> command), you can store the command string in Q-register Z by entering a <^S> command as the first command after the prompting asterisk. The <^S> has this function only when you use it as the first command in a command string. The <^S> command is especially useful when an error occurs in a long command string.

3.17 TECHNIQUES AND EXAMPLES

TECO may be used in several ways. The most elementary application involves using TECO to create and edit ASCII files on-line. The user enters short command strings, often consisting of a single command, and proceeds from task to task until the file is completely edited.

Since every edited job is simply a sequence of TECO commands, an entire job may be accomplished with one long command string consisting of all the short command strings placed end to end with the intervening double ALT MODE characters removed. This leads to the concept of a TECO editing program, which is simply a long command string that performs a certain editing task. Editing programs may be written (using TECO) and stored in the same manner as any other ASCII file. Whenever the program is needed, it may be read into the buffer as text, stored in a Q-register, and executed by an Mq command (where "q" is the Q-register name).

This is fine for clear-cut editing assignments, such as converting from one format to another or editing certain characters out of a file, but many editing jobs are so complex that a given editing program will only solve a small class of problems. The solution, in this case, is to write very specialized "editing subroutines." TECO subroutines might perform such elementary functions as replacing every occurrence of two or more consecutive spaces with a tabulation character, for example, or ensuring that words are not hyphenated across a page boundary. When an editing problem arises, the right combination of subroutines may be loaded into various Q-registers, augmented with additional commands if necessary, and called by a "mainline" command string.

Editing subroutines are essentially macros; that is, sequences of commands which perform commonly required editing functions. Thus, another application of TECO is in the creation and use of a macro library. As each editing job is undertaken, the user may look for

sequences of operations which might be required in future editing assignments. All of the TECO commands required to perform such an operation may be loaded into a Q-register and executed by means of an Mq or nMq command. When the job is finished, the content of any Q-register which contains a useful macro may be written onto an output file (via the buffer) and saved in the macro library. The nMq command, which was designed to facilitate use of macros, permits one run-time numeric argument to be passed to the macro.

The following examples illustrate some of the techniques discussed earlier. It would not be practical to include examples of the use of every TECO command, since most of the commands can be used for many applications. Instead, you are encouraged to experiment with the individual commands.

Example 1: Splitting, Merging, and Rearranging Files

Assume that a user has a file named PGM.PA on the system device and that this file contains data in the following form:

AB FORM CD FORM EF FORM GH FORM IJ FORM KL FORM MN FORM OP

where each of the letters A, B, C, etc., represents 20 lines of text and FORM represents a form feed character. The user intends to rearrange the file so that it appears in the following format:

AOB FORM D FORM MN FORM EF FORM ICJ FORM KL FORM P FORM GH

The following sequence of commands will achieve this rearrangement. (Search command arguments are not listed explicitly.)

.R TECO	Call TECO.
*EBPGM.PA$Y$$	Specify input file and get first page.
*NC$$	Search for a character string in C to write A and B on the output file.
*J20X1$$	Save all of C in Q-register 1.
*20K$$	Delete C from the buffer.
*NG$$	Search for a character string in G to write D, E and F on the output file.
*HX2$$	Save G and H in Q-register 2.
*Y$$	Delete GH from the buffer and read IJ.
*20L$$	Move pointer to the beginning of J.
*G1$$	Insert C, which was stored in Q-register 1.
*NMM$$	Search for a character string in M to write ICJ and KL on the output file.
*HX1$$	Save MN in Q-register 1 (the previous content is overwritten).
*Y$$	Delete MN and read OP.
*J20X3$$	Save all of O in Q-register 3.
*20K$$	Delete O from the buffer.
*P$$	Write P onto the output file, leaving the buffer cleared (the input file is exhausted).
*G2$$	Bring GH into the buffer from Q-register 2.
*HPEF$$	Write GH on the output file and close it.
*EBPGM.PA$Y$$	Open the partially revised file.
*20L$$	Move the pointer to the beginning of B.
*G3$$	Insert all of O from Q-register 3.
*ND$$	Search for a character string in D to write AOB on the output file.
*PWHK$$	Write D on the output file and clear buffer.
$G1$$	Bring all of MN from Q-register 1 into the buffer.
*EX$$	Write MN onto the output file, then close the file and exit to the OS/8 monitor.

At this point, the file has been rearranged in the desired format. Of course, this rearrangement could have been accomplished in fewer steps if the commands listed above had been combined into longer command strings. Note that the asterisks shown at the left margin in this example are generated by TECO, and not typed by the user.

Assume, now, that the same input file mentioned earlier, containing data in the form:

AB FORM CD FORM EF FORM...FORM OP

is to be split into two separate files, with the first file containing AB FORM CD and the second file containing KL FORM M, while the rest of the data is to be discarded. The following commands could be used to achieve this rearrangement:

.R TECO	Call TECO.
*ERFILE$EWFILE.1$$	Open the input file and the first output file.
*Y$$	Read AB into the buffer.
*P$$	Write AB FORM onto the output file and read CD into the buffer.
*HPEF$$	Write CD onto the output file (without appending a form feed), and close the first output file.
*<-K$$	Search for a character string in K. After this command has been executed, the buffer will contain KL. No output is generated by the search.
*EWFILE.2$P$$	Open the second output file and write KL onto it. Read MN into the buffer.
*2OL0,.P$$	Move the pointer to the end of M, then write M onto the output file.
*EF^C$$	Close the second output file and exit to the OS/8 monitor.

As a final example of file manipulation techniques, assume that the user has two files. One file is MATH.BK, which contains information in the form:

AB FORM CD FORM EF FORM GH FORM IJ FORM KL

and the other is MATH.FT, which contains:

MN FORM OP FORM QR

If both of these files are stored on DECtape unit 1, the following sequence of commands may be used to merge the two files into a single file, MATH.NW, which contains all of MATH.FT followed by the latter half of file MATH.BK in the following format:

MN FORM OP FORM QR FORM GH FORM IJ FORM KL

*R TECO	Call TECO.
*ERDTA1:MATH.FT$$	Open the first input file.
*EWMATH.NW$$	Open the output file on the OS/8 default device.
*Y$$	Read MN into the text buffer.
*NR$$	Search for a character string in R to write MN and OP onto the output file.
*PW$$	Write QR onto the output file, appending a form feed.
*ERDTA1:MATH.BK$$	Open the second input file.
*Y$$	Read AB into the buffer. QR is overwritten.

*<-G$$	Search for a character string in G to delete AB, CD and EF, leaving GH in the buffer.
*NK$$	Search for a character string in K to write GH and IJ on the output file, leaving KL in the buffer.
*HPEF^G$$	Write KL onto the output file (without appending a form feed) and close the file, then exit to the OS/8 monitor.

Example 2: Alphabetizing by Binary Search

Assume that TECO is running and the buffer contains many short lines of text, each beginning with an alphabetic character at the left margin (that is, immediately following a line feed). The lines might consist of names in a roster, for example, or entries in an index. Figure 3-1 shows a command string which will rearrange the lines into rough alphabetical order. This command string groups all lines which begin with the character "A" at the beginning of the page, followed by all lines beginning with "B," and so on.

```
!START! J 0AUA!

!!CONT! L 0AUB!

!QA-QB"G XA K -L GA 1UZ'!

!QBUA!

!L Z-."G -L 0CONT$'!

!QZ"G 0UZ 0START$'!

!$$
```

Figure 3-1 Command String for Example 2

Example 3: An Elementary TECO Macro

Figure 3-2 shows a TECO macro which right justifies the content of the text buffer on a 60-space line. This macro assumes that the buffer contains paragraphs of text in manuscript form and that every line which is not the last line of a paragraph contains between 40 and 60 characters.

When the macro is run, it counts the number of spaces and the number of characters in each line. It then adds spaces between words until the line contains a total of 60 characters. Lines which contain fewer than 40 characters are assumed to be paragraph terminators. These lines are not justified. Figure 3-3 shows how the macro may be stored, loaded and executed using DECtape unit 1 as the storage device. In this example, DECtape file 'TEXT.AS" is the file to be justified.

```
J!1!OUN 0US!
!<QNA-32"E 1%S $ '!
!QNA-13"E OJUSTIFY$'!
!1%N$>!
!!JUSTIFY! QN-40"G!
!60-QN-QS<S $I $S^N $>
!OL QS%N$ Q5%S$ OJUSTIFY$'!
!60-QN"G 60-QN<S $1 $S^N $>''!
!L Z-. "G 01$'$$
```

Figure 3-2 An Elementary TECO Macro for Example 3

```
.R TECO
*ERDTA1:MACRO.TE$ Y HXI HK$$
*ERDTA1:TEXT.AS$ Y MIS$$
```

Figure 3-3 A Second Macro for Example 3

Example 4: Managing a Macro Library

A TECO macro library is most conveniently stored with TECO on the OS/8 system device. Macros are usually short enough to require a small amount of storage space; however, it is impractical to store each macro in a separate named file, because a large macro library stored in this manner would make the device unmanageably large and might even exhaust the available directory entries.

Figure 3-4 illustrates a macro that packs the user's TECO macro library (or any other set of short ASCII files) into a single file requiring only one directory entry. This macro could be stored on the system device in a file named PACK.TE (the extension indicates a TECO command string file). The user must also create a separate file containing the name of each file to be packed. This file must be formatted as follows:

```
file1.ex
file2.ex
file3.ex
 .
 .
 .
filen.ex
```

where each file specification after the first is preceded and followed by a carriage return/line feed combination. Assume that such a file is created and stored as INDEX.AS on the system device. If macro PACK.TE is also on the system device, the following commands will pack all files listed in INDEX.AS into file MACLIB.PK on the system device.

```
Y 10<A> HX0 HK 0U1 0U2
<G0 Q1J :S
$; .U1 2R 0X4 HK
I ERDSK:$ G4 @I.$. HX3
M3 HK I\$ G4 I\$ 0U5
!A! AZ"N PW HK 0U5 0A$'
%2$> Q2"E 0B$' EF
!B! HK Q2\ I FILES PACKED
$ HT HK
```

Figure 3-4 File-Packing Macro

The packing macro prints a message, as shown, where "n" is the number of files that were packed. The files to be packed will be taken from the system device. Files PACK.TE, INDEX.AS and MACLIB.PK may reside on any file-structured device if the file designations in the above command summary are changed accordingly. (See Figure 3-5.)

```
.R TECO
*ERSYS:PACK.TE$Y HXP HK$$
*ERSYS:INDEX.AS$EWSYS:MACLIB.PK$MP$$
N FILES PACKED
*
```

Figure 3-5 Loading and Running the File-Packing Macro

Once the packing macro has packed all the files into MACLIB.PK, the individual files may be deleted. Alternatively, macros could be saved in individual files on, say, DTA1 and the packing macro could be used to pack the files into one system device file simply by replacing the imbedded "ERDSK:" command in the macro body with "ERDTA1:". If the library index is also saved on the system device, an unpacking macro may be used to create an unpacked copy of the macro library whenever required, and the original library tape may be saved as a backup.

Figure 3-6 illustrates a macro that unpacks the output file produced by the packing macro. This macro accepts a packed ASCII file (such as MACLIB.PK), then unpacks the file and restores each entry as a discrete file with the appropriate specification.

Assume that a user desires to access a macro or other ASCII file that was packed into file MACLIB.PK, as shown in the previous example. If file UNPACK.TE contains the unpacking macro, the following commands will unpack the entries and restore them as individual, named files.

The unpacking macro prints a message, as shown, where "n" is the number of files that were unpacked. Once the files are unpacked, they will be directed to the system disk. Alternately, the unpacked files could be directed to, say, DECtape unit 5 by modifying the "EWDSK:" command in the macro body to read "EWDTA5:". (See Figure 3-7.)

```
0U2 <Y -Z; 0A-92'E
:2S\$'L .-13'L 1,.-1X4
0,.K Q2'E 0A$' EF
!A! %2$ I EWDSK: $ G4
@I.$. 0,.X3 M3 0,.K'''
FW> Q2'E 0B$' EF
!B! Q2 I FILES UNPACKED
$ HT HK
```

Figure 3-6 Unpacking Macro

```
.R TECO
*ERSYS:UNPACK.TE$Y HXP HK$$
*ERINCEX.AS$MP$$
N FILES UNPACKED
*
```

Figure 3-7 Loading and Running the Unpacking Macro

CHAPTER 4

TECO COMMANDS

4.1 INTRODUCTION

This chapter describes all TECO commands used on the PDP-8. The commands are listed in their ASCII order with the following exception: A control character is listed under its alphabetic equivalent. For example, you will find ^F listed user F, and not preceding A as it would in the normal ASCII collating sequence.

Within a letter grouping, the letter predominates over a symbol; for example, A precedes ^A.

The nonalphabetic symbols follow the same rule. However, since the ASCII collating order is difficult to remember, the order for the entry of nonalphabetic symbols is:

```
!
"
%
.
:
;
< >
=
?
@
\
^

_
```

The ASCII character set is listed in Appendix A.

4.2 A APPEND COMMAND

Append next page to end of buffer.

FORMAT: A

The A (append) command reads in the next page of the input file without clearing the current contents of the editing buffer. TECO concatenates this information to that which is already in the buffer; that is, TECO places it in the buffer following buffer position Z. An A command does not change the position of the buffer pointer.

If the editing buffer does not have sufficient space to accommodate the appended data, TECO issues the error message ?MEM STORAGE CAPACITY EXCEEDED.

TECO terminates input begun by an A command when:

1. The end of the input file is reached;
2. A form feed character is read;
3. The buffer is 2/3 full and a line feed character is read;
4. The buffer is filled to within 128 characters of its capacity; or
5. The buffer is full.

If TECO reads a form feed (that is, if input stops because of condition 2), the form feed flag ^E is set to -1. TECO does not place the form feed flag into the buffer with the rest of the text.

The next input command you enter begins input of the character following the form feed. If a form feed is not read, the form feed flag is set to 0. You may test the form feed flag, but this is usually unnecessary.

The A command does not accept a numeric argument. Note that nA, where n is a numeric argument, is a different command. If you wish to append more than one page to the editing buffer, you can type n<A>, where n is the number of pages you wish to append.

If the end of the input file was previously read (that is, if the EOF flag has been set (see ^N)), the A command has no effect.

Examples:

*YA$$	This command deletes the page of text currently in the editing buffer, and reads in the text two pages of the current input file, appending the second page to the first.
*A$$	This command enters the next page of the file into the editing buffer and appends it to the data already in the buffer. The previous contents of the buffer are not altered and the pointer is not moved.
*<A^N;>EX$$	If the contents of a file fit into memory as a single unit, this command string will bring the entire file into memory and remove all form feeds. The ^N; command will cause TECO to exit from the loop when the end-of-file flag is set.

4.3 nA COMMAND

Return the ASCII code of a character.

FORMAT: nA

where n prints to the (n+1)th character following the buffer pointer.

The purpose of the nA command is to return a number equal to the ASCII value for the (.+n+1)th character in the editing buffer.

For example, the expression 1-1A is equivalent to the ASCII code for the character immediately preceding the pointer and 0A is equivalent to the ASCII code of the character immediately following the pointer.

NOTE

You cannot omit n. If you omit the position indicator, TECO will treat your command as an append command. Because TECO cannot determine if you meant to type an nA or A command, no error message can be issued.

If you attempt to reference a position outside of the buffer, TECO prints the error message ?POP ATTEMPT TO MOVE POINTER OFF PAGE.

Example:

```
...0A-^^/'E
1A-^^B'EMQ' '...
```

This command string segment will verify that the string /B immediately follows the buffer pointer. If it does, TECO executes the macro in Q-register q. If not, TECO executes the commands following the second apostrophe. (The purpose of the ^^ command is to return the ASCII value of the character following it.) 0A-^^/ is an argument which is the value of the character following the pointer minus the ASCII value of a / character.

4.4 ^Atest<^A> TYPE-OUT COMMAND

Type delimited text.

FORMAT: <^A>text<^A>

where the first ^A may be a <circumflex-A> or <CONTROL/A> but the second must be a <CONTROL/A>.

The ^Atext<^A> command types the "text" between the ^As on your terminal. The text is usually a message that you wish typed out during the execution of a command string.

The first ^A is the actual command. Enter it as a circumflex-A or a <CONTROL/A>. The second ^A is a delimiter which indicates the last character in the text argument. The second delimiter must be a <CONTROL/A> because a ^ is a legal character within a text string.

The string "text" is the character string that TECO types out when it encounters the ^A command. The text string can contain any character except ^A.

Example:

```
*0U1<!ST^N;50S
                $"FAOST$ '12I$0,.P0,.K^A
OUTPUTTING PAGE <^A> %2=>EC$$
OUTPUTTING PAGE 1
OUTPUTTING PAGE 2
OUTPUTTING PAGE 3
OUTPUTTING PAGE 4
```

1. 0U1 stores 0 in Q-register 1.

2. The angle brackets indicate TECO will perform a repeated sequence of operations.

3. The 50S command searches for the 50th occurrence of a line feed. If the buffer does not contain 50 lines, the seaarch fails and TECO executes the commands after the failure conditional ("F).

4. TECO appends the next page and branches back to search for the fiftieth line (AOST$).

5. If there are now 50 lines, TECO inserts a form feed into the buffer, and the text in the buffer up to and including the form feed is written to the output file.

6. 0,.K removes this text from the buffer.

7. TECO then prints the message.

8. The %1 command increments the value in Q-register 1, and then the = command prints it.

9. This continues until the end-of-file flag is set; that is, ^N=-1.

10. TECO closes the file.

4.5 B POSITION INDICATOR

Position of beginning of buffer, always 0.

FORMAT: B

The B buffer position indicator always equals 0. It represents the position at the beginning of the buffer, that is, the position preceding the first character in the buffer.

You may also use the B command in arithmetic expressions.

Example:

*B,.K$$	Remove all characters from the beginning of the buffer to the character immediately preceding the buffer pointer position.

4.6 C BUFFER POINTER COMMAND

Move pointer forward.

FORMAT: nC

where n may be positive or negative.

You can use the C command to move the buffer pointer. You would normally use this command when the pointer has to be moved across only several characters. The C commands are listed in Table 4-1.

Table 4-1
C Commands

Command	Argument	Function
nC	n>0	Move the pointer forward over n characters in the buffer from the current position of the pointer; that is, nC is equivalent to (.+n)J. nC is also equivalent to -nR.
C	1 is assumed	Move the pointer forward one position. This is equivalent to -R.
0C	0	No movement of the pointer.
-C	-1 is assumed	Move the pointer backward one position. This is equivalent to R.
nC	n<0	Move the pointer backward over n characters in the buffer from the current position of the pointer; that is, nC is equivalent to (.-n)J. nC is also equivalent to nR.

If a C command attempts to move the buffer pointer across either editing buffer boundary, TECO ignores the command and prints the error message ?POP ATTEMPT TO MOVE POINTER OFF PAGE.

Examples:

C	Advance the buffer pointer one space.
L4C	Advance the pointer to the position following the fourth character in the next line.
Q1C	Advance the pointer the number of characters in Q-register 1.

4.7 ^C COMMAND

Abort or exit.

FORMAT: ^C

If TECO is in command mode (that is, awaiting a command), then <^C> is an OS/8 monitor command which causes a jump back to the monitor. However, if TECO is not in command mode, the <^C> is a TECO command which aborts the present action and returns control to TECO. If you abort TECO while it is executing with a <^C>, it prints the message ?XAB EXECUTION ABORTED.

If you wish to insert this command into a text or macro, you can type it using the circumflex convention.

4.8 D DELETE COMMAND

Character deletion.

FORMAT: nD

where n may be positive or negative.

You can use a D command to individually delete characters and short strings. The D commands are listed in Table 4-2.

Table 4-2
D Commands

Command	Argument	Function
nD	n>0	Delete the n characters following the buffer pointer.
D	1 is assumed	Delete the character following the buffer pointer.
0D	0	This is a null command.
-D	-1 is assumed	Delete the character preceding the buffer pointer.
nD	n<0	Delete the n characters preceding the buffer pointer.

After TECO executes the D command, it positions the buffer pointer between the characters that preceded and followed the deletion. The pointer will always be adjacent to one of the characters to which it was adjacent before the deletion.

If you attempt to delete text up to and across the beginning or end of the buffer, no text will be deleted and TECO prints the error message ?POP ATTEMPT TO MOVE POINTER OFF PAGE.

When deleting across carriage return/line feeds, the carriage return/line feed counts as two characters.

Examples:

The following examples assume that the buffer contains the text shown at the right; the buffer pointer points to the position between the M and the N.

```
ABCDE
FGHIJ
KLMNO
PQRST
UVWXY
Z
```

*6D$$ — Deletes NO, the carriage return/line feed, and PQ, changing the third and fourth lines to KLMRST.

*-D$$ — Deletes M.

*-5D$$ — Deletes the carriage return/line feed, and KLM, changing the second and third lines to FGHIJNO.

*-2D2D$$ — Deletes LMNO, changing the third line to K.

*KD$$ — Deletes NO and the carriage return/line feed, and P, changing the third and fourth lines to KLMQRST.

4.9 ^D DECIMAL RADIX COMMAND

Enter decimal mode.

FORMAT: ^D

The entering of a ^D or a <CONTROL/D> changes the current radix to decimal if TECO was not already in decimal. You would use this command after you have changed the radix to octal with a ^O command.

If the radix is in decimal, this command is a no-op.

The initial radix of TECO is decimal.

4.10 <DELETE>

Delete a character.

FORMAT: none.

Typing the DELETE key while entering a command string causes TECO to delete the last character typed. You may continue deleting characters until you have erased the entire command string. An attempt to delete past the prompting asterisk causes TECO to type a carriage return/line feed followed by the printing of another asterisk.

The actual function of the delete character is to delete the last typed character in the command string. Consequently, if the incorrect character is not the last one in the string, you must delete all characters back to that point.

Type <DELETE> twice to erase a carriage return and the TECO-generated line feed following it.

If you have used the SET TTY SCOPE OS/8 monitor command, the <DELETE> will cause an immediate erasure of the character, and the cursor will be moved backwards. You can delete any nonexecuted character; however, you cannot erase <^C> and <^G>, etc., because these execute immediately.

If your terminal is not in "scope mode," TECO echoes a deletion by typing the character deleted; for example, the string "real-time," if followed by five deletions, would appear as "real-timeemit-" on your terminal.

If your terminal is not in scope mode, you should use the <^G><sp> command when, through extensive editing, a command string becomes unintelligible.

Example:

After typing the portion of the command string shown below, you discover that you have misspelled the name "Ericson."

```
*3LKILEIF ERICXON
```

To nullify this error, you would type three successive <DELETE>s. As you do this, TECO responds by retyping the character being deleted.

```
*ILEIF ERICXON<DELETE>N<DELETE>0<DELETE>X<^G><sp>
ILEIF ERIC
```

4.11 ^E END-OF-PAGE FORM FEED FLAG SIGNAL

This flag indicates if a form feed was read when the current buffer's contents were placed into the buffer.

FORMAT: ^E

If, when TECO is loading data into the editing buffer, input stops because a form feed was read, it sets the ^E flag to -1. If input was stopped because:

1. The end of the input file was reached;
2. The buffer is 2/3 full and a line feed character is read;
3. The buffer is filled to within 128 characters of its capacity; or
4. The buffer is full

the flag remains as a 0.

The P and other similar output commands test this flag to determine whether they should append a form feed when they write the contents of the buffer to the output file.

Example:

This macro divides a file into pages of 50 lines each and also preserves original form feeds.

```
<!ST!^N;50S
          $"F^EU1 Q1"LP' Q1"EA' 0ST' 12I$ 0,.P 0,.K> EC$$
```

1. TECO searches for the 50th line terminator.
2. If this search fails, TECO stores the value of the form feed flag in Q-register 1.
3. If this value is less than 0, TECO writes the buffer to the output file and reads in a new page.
4. If it equals zero, TECO appends a new page to the buffer's contents.
5. TECO then branches back to ST to search again for the fiftieth line.
6. If fifty lines were found, TECO appends a form feed. This text is written to the output file and then deleted. TECO again searches for the 50th line.
7. This continues until the end of the file is reached, that is, ^N is set to -1.

4.12 EB EDIT BACKUP COMMAND

Open an input and output file, creating a file with the same name as the input file. The old file has its extension changed to BK.

FORMAT: EBfilespec$
@EB/filespec/

where / is an arbitrary delimiter which is not one of the characters in filespec.

Use the EB command to open a file for editing in a manner similar to the TECO filespec command.

NOTE

You can use this command only for files stored on a directory-structured device.

The EBfilespec command is equivalent to:

*ERfilespec$EWfilespec$

except that the input file has its extension changed to BK.

The operation of the EB command is as follows:

1. The EB command executes an automatic ERfilespec$ command, opening the specified file for input and releasing any previously opened input file.

2. Then, it opens a temporary file to receive the output of the edited version of the input file.

3. The output device is the same as the input device.

4. Finally, the EB command sets an internal flag indicating that special action must be taken when the EB file is closed (by an EC, EF, EX, or EG command). It also prohibits any further EW or EB commands until the file is closed.

When you close a file opened with an EB command, the following action takes place:

1. If a file with the name filename.BK already exists on the device, TECO deletes it.

2. TECO renames the input file filename.ext to filename.BK.

3. Finally, TECO renames the temporary output file filename.ext.

You cannot use the EB command with a file having the extension BK.

The TECO filespec initialization command causes an automatic EBfilespec to be executed (followed by an automatic Y command).

Examples:

*EBAB.FT$$	This command selects the disk file AB.FT for editing. When the editing is completed, the file AB.FT is the new version. TECO changes the old version to the back-up file AB.BK, and deletes any previous back-up file AB.BK.
*@EB/RXA1:TEXT$.TX/	This command selects the file TEXT$.TX, where the $ is an ESCAPE character from device RXA1 (floppy drive 1). If you had created a file with the name TEXT.TX, it could not be accessed through the OS/8 monitor except through the use of # filespec construction.

4.13 EC EXIT CLOSE COMMAND

Transfer the remainder of the input file to the output file and then close the input and output files.

FORMAT: EC

EC commands TECO to write the contents of the editing buffer and any information in the input file not as yet brought into the editing buffer onto the output file. The input and output files are closed. After the execution of an EC command, TECO remains in command mode.

The EC command is similar to the EX command, which produces the identical output and also closes the input and output files. However, the EX command returns control to the OS/8 monitor.

If you use an EC command instead of an EX command, you do not have to reload TECO, and you do not lose the data in the Q-registers.

Example:

*EC$$	This command closes the current file and writes its contents to the appropriate file.

4.14 EF END FILE COMMAND

End the output file with the current page.

FORMAT: EF

The EF command is an output-file-closing command; that is, TECO closes the file you opened with an EW command. You would normally use an EF command to close the output file after all output to it is complete. Furthermore, the EF command is most often used after a P command, which outputs the last page of a file.

NOTE

If you type an EF command in the middle of the file, all succeeding pages that would have been read to it with an EX or EC command are omitted. If you are not creating a file, it is far safer to exit with an EC command rather than an EF command because you could lose data if you erroneously think you are at the end of the file.

Examples:

*PEF$$	Output the current page to the output file, and then close the output file. Use this command string to close a file after writing the last page.
*PWEF$$	Equivalent to the preceding example, except that the buffer is not altered.
*EFEWTEXT.TE$$	Close the current output file and open an output file TEXT.TE on the OS/8 default device DSK:.
*ERPTR:$EFEWRXA0:FILE.MA$$	Read the input file from the paper tape reader, close the current output file, and open FILE.MA on RXA0: as an output file.
*ERFILE.TX$EWFILE1.TX$ 4PEFEWFILE2.TX$ 4PEFEWFILE3.TX$EC$$	This command divides file FILE.TX into three files: The first two files each consists of four pages, while file FILE3.TX contains the remainder of FILE.TX. If you thought FILE.TX consisted of 12 pages of data, you could have concluded with a 4PEF command string, rather than an EC. However, if you were mistaken, you would not have written the remainder to an output file, and thus could lose all text following the 12th page with an EF command.

4.15 EG EXIT AND GO COMMAND

Exit from TECO after output, then either:

1. Re-execute the last compile-class CCL command, or
2. Perform the action specified in the optional text argument.

FORMAT:
```
EG
@EG//
EGtext$
@EG/text/
```

where:

text	is an OS/8 monitor command
/	is an arbitrary delimiter which is not one of the characters in "text".

The EG command is a dual-purpose command.

1. It transfers the contents of the editing buffer and any remaining text in the input file to the output file. It then exits from TECO. This is identical to an EX command.
2. An OS/8 monitor command will then be executed.

If you specify a text argument to an EG command, that text must be an OS/8 monitor command. After the exit from TECO, that command will be executed. Thus,

```
*EGtext$
```

is equivalent to:

```
*EX
.text
```

This command form is often used when the text you have created is a batch command file. If it were, you would type EGSUBMIT filespec$$.

If you do not specify a text argument, TECO causes the last compile-class command (for example, COMPILE, EXECUTE, or LOAD) attempted before TECO was called to be re-executed (with the same arguments). Generally, you would use the EG command to exit from an editing job that was called by an EBfilespec or TECO filespec command.

As an example, suppose you give the command

```
.COMPILE PLOT.FT
```

to request compilation of a FORTRAN source program, but the compiler encounters errors in the code. You would then call TECO to correct these errors with the command

```
.TECO PLOT.FT
```

After you have corrected the errors, you would exit from TECO with the command

```
*EG$$
```

This command causes: 1) the rest of the file PLOT.FT to be output and closed; and 2) the command COMPILE PLOT.FT to be re-executed automatically.

Example:

The following is an example that can be called with a MUNG command. It will compile all FORTRAN programs that are on your disk. You call it with the OS/8 command.

```
.MUNG COMPIL,*.FT

Z"E EBTEMP.TM$ Y3KZJ-3KJ
<FS $$;> <S
$;-L I.COMPILE $L>
I$JOB
$ZJI$END
$
EGSUBMIT TEMP.TM/T/H$
'Z"N EWDIR.TM$
JI$JOB
.DIR TEMP.TM<$
ZJI/F
.MUNG COMPIL
$END
$
EGSUBMIT DIR.TM/H/T$'
```

In this example, the $ symbols indicate dollar signs rather than ESCAPEs.

4.16 EH EDIT HELP COMMAND

Error message printing form command.

FORMAT: nEH

where:

n is an integer such that -1<n<3.

TECO error messages consist of two parts: The first, or code, is always typed, while the second part, the brief message, is also typed only with the 16K version of TECO.

By using the EH command, you may change TECO so that it prints only the 3-letter code preceded by a ? or both the code and a 1-line error message. TECO always prints an error message.

Table 4-3 lists the EH commands.

Table 4-3
EH Commands

Command	Function
1EH	Sets TECO so that it prints only the 3-letter code part of the error message.
2EH	Sets TECO so that it prints both the error message code and the 1-line extended descriptions automatically. On systems with less than 16K, a 2EH command is a null command because there is insufficient memory for the internal file which contains the error messages.
0EH	Resets TECO to the system standard mode of error message type-out (normally equivalent to 2EH).

You may return the current value of the EH setting by typing EH. To be useful, this must be typed as a numeric argument to another command.

4.17 EK EXIT KILL COMMAND

Detach output file from TECO without performing input or output.

FORMAT: EK

The EK command detaches the output file (if there is one). It does not delete any files. If, for example, after typing an EWfilespec1 command and deciding that this is an error, you wish to write to filespec2 instead, then the command

```
EKEWfilespec2
```

will detach filespec1 without causing output.

You can also use the EK command to detach a file if it were going to overwrite another file.

4.18 EO VERSION COMMAND

Return version number of TECO.

FORMAT: EO

An EO command returns a value which specifies the current version number of TECO. This will be either a 5 or a 5xx, where 5 is the current version and xx is any 2-digit number. If EO does not return the 5, this manual may contain inapplicable or erroneous information for your version.

NOTE

This command is included for compatibility with versions of TECO implemented on other DIGITAL computers. It is not used except to see if you have the version of TECO described in this manual.

4.19 ER EDIT READ COMMAND

Initializes a file so that TECO may sequentially extract information from it.

FORMAT: ERfilespec$
@ER/filespec/

where:

/ is an arbitrary delimiter which is not one of the characters in filespec.

The ER command initializes a file so that TECO may read information from it. An ER command also terminates input from any file that may have been previously opened for input, in addition to opening a file for input.

You may open one file for input, read only part of that file, and then, with another ER command, release the first file and open a new file for input. It is not necessary to read the end of one file before opening a second. However, opening the second file does terminate input from the first.

NOTE

OS/8 TECO permits only one input file to be open at any time.

If you are creating a file, then you do not need to enter an ER command. Instead, you enter the text directly into the text buffer from the terminal keyboard.

Examples:

*ERPULSE.FT$$	Select the file PULSE.FT from the OS/8 default device DSK:.
*ERCDR:$EWPTP:$$	Select the card reader for input and the paper tape punch for output.
*ERDTA1:INPUT.TX$EWRKA0:OUTPUT.TX$$	Open an input file INPUT.TX on DECtape unit 1 and an output file OUTPUT.TX on disk unit 0.
*ERRXA1:PROG$$	Prepare to read input file PROG or PROG.MA from RXA1:.

4.20 <ESCAPE> COMMAND

A signal to begin execution or a command delimiter.

FORMAT: none.

An <ESCAPE> character is the only predetermined command delimiter in TECO. Use it to indicate the location of the last character in a text argument. In this case, it is part of the previous command; for example, in IABC$, the ESCAPE is part of the I command.

The <ESCAPE> is sometimes also used after an n%q command. You would use it to ensure that the incremented value in the Q-register is not used as an argument for the following command.

You may insert the <ESCAPE> into a text string if you preface the command with an @ modifier. (See Section 4.68.) Using the @ modifier allows you to choose any delimiter that is not one of the TECO special characters listed in Table 2-1.

You may also insert the <ESCAPE> into a command string using a 27I$ command.

TECO echoes an <ESCAPE> as a dollar sign ($) on a terminal.

A double ESCAPE ($$) signals TECO to begin execution of the command string.

On some terminals, ESCAPE is labeled ALTmode or PREfix. (The term "ALTmode" is the traditional TECO name for ESCAPE.)

4.21 ET EDIT TERMINAL COMMAND

Flag and command to set typeout modes.

FORMAT: nET

where:

n is a combination of the numbers 1, 2, and 8.

The ET flag informs TECO whether you have entered a SET TTY SCOPE or SET TTY NO SCOPE command to the OS/8 monitor. If you have entered SET TTY NO SCOPE, then this flag has a value of 0. If you are in "scope mode" then the flag is set to 2.

Use this command to control how TECO will type information to your terminal. The three values this command can take are listed in Table 4-4. Because these numbers represent bits, they are additive; that is, one ET value does not preclude another.

Table 4-4
ET Commands

Command	Function
1ET	Setting this bit inhibits all of TECO's typeout conversions. All characters are output to the terminal exactly as they appear in the buffer or <^A> command. For example, the changing of control characters in the CONTROL/CHAR form is suppressed. This mode is useful for driving displays.
2ET	Process <DELETE> and <^U> in scope mode. Scope mode processing uses the cursor control features of the CRT terminals to handler character deletion by actually erasing characters from the screen.
8ET	Read without echoing for ^T command. This allows data to be read by the ^T command without having the characters echo at the terminal. Normal command input will echo.

Example:

The following TECO command string erases the entire screen of a VT52.

```
*27^T^^H^TW7^T^^K^T$$
```

4.22 EU EDIT UPPER/LOWER COMMAND

Upper-case and lower-case flagging of output.

FORMAT: nEU

where:

n is a positive, negative, or zero integer.

The upper/lower-case flag determines the manner in which TECO is to transmit characters to your terminal. If you have entered a SET TTY LC command to the OS/8 monitor, TECO, when called into memory, sets this value to -1. Otherwise, it will be set to 0.

Use the nEU command to flag upper-case output characters with an apostrophe ('). For example, if you are displaying a text, a form of the EU command will tell TECO to precede every character entered in lower case with an apostrophe. Although TECO prints the text in upper case, you would clearly be able to distinguish upper- and lower-case characters.

The EU commands are listed in Table 4-5.

Table 4-5
EU Commands

Command	Argument	Function
nEU	n>0	Flag upper-case characters with an apostrophe.
0EU	0	Flag lower-case characters with an apostrophe.
nEU	n<0	Do not flag characters.

4.23 FW EDIT WRITE COMMAND

Initialize the output file.

FORMAT: EWfilespec$
@EW/filespec/

where:

/ is an arbitrary delimiter which is not one of the characters in filespec.

Use an EW command to open a file for output. If an output file is currently open, a second EW command closes that file before opening the new file. TECO permits only one output file to be active at any one time.

If you type an EWfilespec command while a file is open, TECO deletes the previously opened file. However, TECO does not permit you to type an EW command if you had opened the file with an EBfilespec command.

You may not output any information without first entering an EW or equivalent command.

TECO does not permit you to use multiple EW commands without changing the input file.

The MAKE filespec initialization command causes TECO to execute an automatic EWfilespec$ command.

Examples:

*ERRXA2:CREF.2$EWRXA0:CREF.3$$	This command string selects the file CREF.2 on diskette drive 2 for input and opens a file called CREF.3 on diskette drive 0 for output. If there is a file named CREF.3 already on the diskette, it will be overwritten.
*EWPROG.PA$$	Prepare to write the output file PROG.PA on the OS/8 default device DSK:.
*EWRXA1:OCON.TE$$	Prepare to write the output file OCON.TE on RXA1:.

4.24 EX EXIT COMMAND

Exit from TECO to the OS/8 monitor.

FORMAT: EX

The EX command outputs the latter part of the input file and closes the input and output files. Using the EX command is the easiest way to finish editing a job.

For example, you may be editing a 30-page file and the last change you make is on page 10. At this point you can give the command

```
*EX$$
.
```

In this case, the action TECO performs is equivalent to the command string 21PEF, with an automatic exit to the monitor at the end. Thus, TECO:

1. Rapidly moves all the rest of the input file, including the page currently in the buffer, to the output file
2. Closes the output file
3. Returns control to the monitor.

The EX command is equivalent to:

```
*EC$
*^C
.
```

The EX command outputs a form feed character only if, after the output of the editing buffer's contents, it examines the ^E end-of-page form feed flag and finds that a form feed terminated input. In this way, the EX command maintains existing page sizes.

4.25 ^F COMMAND

Return value of console switch register.

FORMAT: ^F

After you enter an ^F or <CONTROL/F> command, TECO returns the number input on the console switch register. (On the PDP-12, this is the right switch register.) This number must be in the range 0<n<4095.

4.26 FN FAST NONSTOP SEARCH/REPLACE COMMAND

Search remainder of the file until text is found and then replace it.

FORMAT:
```
nFNtext1$text2$
nFNtext1$$
n@FN/text1/text2/
n@FN/text1//
```

where:

n	is a positive number. If text2 is omitted, no replacement occurs.
/	is an arbitrary delimiter which is not one of the characters in text1 or text2.

Use an Fn N command to search for a character string in a page of the input file which may not yet have been read into the buffer (function of the N command) and to replace it with another string. The FN command operates like the N command when searching for the string. If the search fails, no replacement occurs.

If you omit text2 in an FN search, TECO deletes text1 and does not insert a string into the buffer to replace it. However, even when text2 is omitted, its terminating delimiter must be present as shown in the form

```
*FNtext1$$
```

The maximum length of a text argument is 31 ASCII characters. The FN command, like all search commands, accepts a colon modifier.

The text1 argument may contain the four match control characters:

^Na	Match anything except "a"
^Q	Use the next command as match character
^S	Match on separator
^X	Exempt position from match

If a search fails, TECO writes the entire input file to the output buffer.

You may preface the FN command with a number n to indicate which occurrence of a string is the object of the search. If a number specifying which occurrence of a string is to be replaced is less than 1, then TECO prints the error message ?NAS NEGATIVE OR ZERO ARGUMENT TO S.

Examples:

*12FNSTRING$TEXT$$	This command replaces the 12th occurrence of STRING with TEXT.
*12<FNSTRING$TEXT$>$$	The first 12 occurrences of STRING are replaced with TEXT. Note the position of the ESCAPEs. The ESCAPE following the TEXT terminates the string TEXT. Because the search is within an iteration, a double ESCAPE cannot be used because it would terminate the command before the iteration were to be entered into the command string.
*12<@ FN/STRING//>$$	This is similar to the above example with the difference being that STRING is not replaced. If the @ modifier were not used, you would have to type $$. This would prematurely terminate the command and you would receive an error message.

4.27 FS FAST SEARCH/REPLACE COMMAND

Search the remainder of the editing buffer until text is found and then replace it.

FORMAT:
```
nFStext1$text2$
nFStext1$$
n@FS/text1/text2/
n@FS/text1//
```

where:

n — is a positive number, text1 and text2 are less than 32 characters long

/ — is an arbitrary delimiter which is not one of the characters in text1 or text2.

The FS command searches for a character string within the current editing buffer (function of the S command) and replaces it with another string. If the string to be replaced is not found after the current pointer position and before the end of the buffer, the search fails and no replacement is made. If a search fails, TECO moves the buffer pointer to the beginning of the editing buffer.

If you omit text2 from a FS command, text1 is deleted without replacement. However, when you omit text2, its terminating ESCAPE must be present as shown in the form

```
*FStext1$$
```

Text1 and text2, like all search commands, may not be longer than 31 ASCII characters.

The FS command may use the colon modifier, while text1 may use the following match control characters:

^Na	Match anything except "a"
^Q	Use the next command as match character
^S	Match on a separator
^X	Exempt position from match

You may preface the FS command with a number n to indicate which occurrence of a string is the object of the search. If the number specifying which occurrence of a string is to be replaced is less than 1, then TECO prints the error message ?NAS NEGATIVE OR ZERO ARGUMENT TO S.

Examples:

*12FSOF$FOR$$	This command causes TECO to search the current buffer for the 12th occurrence of the string OF and to replace it with the string FOR.
*12<FSOFFOR>$$	This command causes TECO to search for, and then to replace, the first 12 occurrences of OF with FOR. Note that the concluding double $$ follows the >.

*12FSINTEREST$$	This command causes TECO to search the current page for the 12th occurrence of the string INTEREST and to delete it. The two ESCAPEs, $$, must be typed following the string to be deleted; the first delimits the string for which you are searching and the second tells TECO that there is no replacement string.
*@12FS/INTEREST//$$	This command is identical to the one described immediately above. It is very useful if a double ESCAPE would prematurely terminate your command string.

4.28 G GET COMMAND

Place Q-register q's contents in the editing buffer.

FORMAT: Gq

where:

q is a Q-register.

The command Gq (where q is one of the 36 Q-registers) fetches a copy of the character string stored in the Q-register and inserts it into the editing buffer at the current buffer pointer position. This command does not alter the contents of the Q-register. TECO positions the buffer pointer at the right end of the character string inserted.

Examples:

```
*ZJ-5XAJ8LGA$$
```

This command string puts a copy of the last five lines of the page into Q-register A and then puts a copy of these five lines immediately after the eight lines in the page. It does not, however, delete the five lines from their position at the end of the page.

```
*STEXT1$0L.U1
STEXT2$0L.U2
Q1,Q2XA
Q1,Q2K
NTEXT3$L GA$$
```

This command string stores all text from TEXT1 to TEXT2 in Q-register A, deletes that text from the page, and then places this text on the line following TEXT3.

If you type a : before a Gq command, TECO types the contents of that register on your terminal without inserting it into the buffer.

Example:

If the second command in the above examples were changed to:

```
STEXT1$0L.U1
STEXT2$0L.U2
Q1,Q2XA
:GA$$
```

then the :GA command would verify that you have placed the proper text into the Q-register.

4.29 <^G> COMMAND

1. Retype all text back to the last line terminator.

   FORMAT: <^G><sp>

   where the command must be typed using a <CONTROL/G>.

2. Retype the entire command string.

   FORMAT: <^G>*

   where the command must be typed using a <CONTROL/G>.

3. Erase all commands that have been entered but not executed.

   FORMAT: <^G><^G>

   where the command must be typed using a <CONTROL/G>.

4.29.1 <^G><sp> COMMAND LINE ECHO COMMAND

The <^G><sp> command prints the line currently being input on your terminal. This command is useful when the correcting of typographical errors causes you to be unable to read easily what you have typed.

If the terminal has a bell, it will ring twice.

You cannot type this command using the circumflex construction.

This is identical in function with typing a line feed during the entering of a command to the OS/8 monitor.

<^G><sp> is normally unnecessary if you are entering information on a CRT and you have entered the OS/8 SET TTY SCOPE command. The <^G><sp> command can be thought of as an erasing commmand although it is actually an echoing command. Its function is to retype the current line, omitting characters erased with other commands. Use it when you type so many <DELETE>s on a line that you cannot determine what you have typed.

After TECO types a line, continue typing the command string just as if the command had not been typed. TECO neither stores this command in nor removes anything from the command string.

Example:

```
*STAET:<DELETE>:<DELETE>T<DELETE>ERT:<TAB>TRZE<^G><sp>
START:    TRZE
$$
*
```

4.29.2 <^G>* COMMAND STRING ECHO COMMAND

The <^G>* command prints all the lines you typed from the last TECO prompt (i.e., the asterisk) to be reprinted. This command differs from the <^G><sp> command in that it types the entire command string, rather than only the last line.

If the terminal has a bell, it will ring.

You cannot use the circumflex construction for this command.

4.29.3 <^G><^G> COMMAND STRING ERASURE COMMAND

Typing two consecutive <^G><^G> characters erases all commands entered but not yet executed.

If the terminal has a bell, it will ring.

You cannot use the circumflex construction for this command.

4.30 H WHOLE POSITION INDICATOR

Incorporate entire buffer (B,Z) limits into a command argument.

FORMAT: H

The H command is equivalent to the numeric pair B,Z. Thus, in those commands that take two numeric buffer position arguments, H represents the combination B,Z (which is the entire buffer). This letter is particularly useful with type-out and output commands.

Examples:

*HK$$	Delete entire editing buffer.
*HT$$	Type entire buffer.
*HXA$$	Insert entire buffer into Q-register A.

4.31 I INSERT COMMAND

Insert text into the buffer.

FORMAT: Itext$
@I/text/
where:

text — is only limited by available command string storage space

/ — is an arbitrary delimiter which is not one of the characters in "text."

The I command followed by a text argument is the basic TECO insertion command. Delimit the text argument by an ESCAPE.

This command inserts the ASCII text string, "text," into the editing buffer just ahead of the buffer pointer. After the insertion, TECO positions the buffer pointer immediately after the last inserted character. TECO does not insert the ESCAPE terminating the text argument. "Text" may contain any character except the special characters listed in Table 1-1.

The amount of core available for command string storage limits the number of characters in the text. During normal editing jobs, DIGITAL recommends that you limit insertions to about 10 to 15 lines each.

If a very long insertion command begins to exceed the TECO command storage capacity, TECO rings the terminal bell once when ten characters of storage remain and once after each additional character entered. The bell also echoes as a ^G. When this occurs, terminate the command string immediately. Entering more than ten additional characters into the current command string causes a fatal error.

The @I/text/ command is slightly more powerful than the I command. It enables you to insert single (but not double) ESCAPE characters in addition to the characters that can be inserted with the I command. The @I form is useful for inserting TECO command strings into the editing buffer.

Delimit the text argument to the @I command, both before and after, by any single character which is not itself a part of the text to be inserted. TECO does not require an ESCAPE to terminate the text string; it is the second occurrence of the delimiting character that terminates the text string. The text is inserted immediately preceding the buffer pointer, as it is with the I command. TECO does not insert the delimiting character.

Examples:

*JIline one line two line three $$ *	This example shows insertion of several lines of text at the beginning of the buffer.
*KI $$ *	Use this command string to delete the tail of a line without removing the carriage return/line feed at the end. If the buffer contains: ABCD EFGH and the buffer pointer is between the B and the C, this command produces: AB EFGH
*I <DELETE> $$ *	Use this command to insert a carriage return without a line feed following it. The single <DELETE> deletes the line feed but not the carriage return.
*@I%TEXT$x<DELETE>$%$$	This is a convenient method for inserting multiple ESCAPEs when using the @I command. Type the sequence x<DELETE>, where x is any character except an ESCAPE, between the successive ESCAPEs. If the x<DELETE> were not typed, TECO would assume that you were terminating the command string.

The following examples assume that the buffer contains ABCDEF with the buffer positioned between the D and E.

*IXYZ$$	Produces ABCDXYZEF with the buffer pointer between the Z and the E.
*I $$	Produces ABCD EF with the buffer pointer positioned before the E.
3RI $4CI $$	Produces A BCDE F

4.32 nI$ INSERT COMMAND

Insert a character into the buffer.

FORMAT: nI$

where:

n is the ASCII value of the character to be inserted.

The nI$ command inserts one character into the editing buffer. The n numeric argument includes all characters that the I and @I commands cannot insert. However, the nI$ command inserts only one character at a time. The command nI$ inserts the character with the ASCII value n into the buffer immediately preceding the pointer.

NOTE

The I$ command inserts a null string into the editing buffer. The nI$ command always inserts a character into the buffer.

The nI$ command is most often used to insert the special characters listed in Table 2-1.

Example:

If you are creating a macro that you wish to abort immediately if a certain condition occurs, then you could type some command string minus the <^C> and then add it with the insert command. For example,

```
*IABCDEFGHIJKILMN$$
```

If the <^C> were to be inserted after the K, then the command string

```
*3R3I$HXA$$
```

would insert the ^C, and then place the command string in Q-register A.

The following command string could also be typed:

```
*IABCDEFGHIJK$ 3I$ ILMN$$
```

4.33 J JUMP COMMAND

Move buffer pointer relative to the beginning of the buffer.

FORMAT: nJ

where n>0

The nJ command moves the buffer pointer to the position immediately after the nth character in the buffer. The J commands are listed in Table 4-6.

Table 4-6
J Commands

Command	Function
nJ	Move the pointer to the position following the nth charater in the text buffer.
0J	move the pointer to the beginning of the buffer.
J	Equivalent to 0J.
ZJ	Move the pointer to the end of the buffer.

4.34 K KILL COMMAND

Line deletion.

FORMAT: nK
m,nK

where:

n may be positive, negative, or zero if there is one argument. If there are two arguments, m<n.

The K commands are described in Table 4-7.

Table 4-7
K Commands

Command	Argument	Function
K	1 assumed	Deletes everything from the buffer pointer through the next line terminator. If the pointer is at the beginning of a line, the K command deletes the entire line. Otherwise, the K command deletes only the portion of the line following the pointer (including the line terminator).
nK	n>0	Deletes everything from the buffer pointer through the nth line terminator following it.
0K	0	Deletes everything from the pointer back to the beginning of the current line.
-K	-1 assumed	Deletes everything from the pointer back to the beginning of the line preceding the current line.
nK	n<0	Deletes everything from the pointer back to the beginning of the nth line preceding the current line.
m,nK	m<n	Deletes the (m+1)st through the nth characters in the buffer and positions the pointer at the point of deletion (that is, the pointer is set equal to m).
HK	B,Z	Deletes the entire contents of the buffer.

If the K command attempts to delete text up to and across the beginning or end of the buffer, TECO deletes text only up to the buffer boundary. The buffer pointer is positioned at the boundary and no error message is printed.

TECO positions the buffer pointer between the character that preceded and followed the deletion.

Examples:

The following examples assume that the buffer contains the text shown at the right; the buffer pointer is positioned between the M and the N.

```
ABCDE
FGHIJ
KLMNO
PQRST
UVWXY
Z
```

*HK$$	Deletes everything in the buffer, but does not delete the form feed (if there is one) marking the end of the page. To delete the form feed as well, type HKA.
*0,.K$$	Deletes everything from A through M.
*.,ZK$$	Deletes everything from N through Z.
*K$$	Deletes NO, changing the third and fourth lines to KLMPQRST.
*0LK$$	Deletes the entire third line.
*L3K$$	Deletes the last three lines (everything from P through Z).
*KD$$	Deletes NO and P, changing the third and fourth lines to KLMQRST.
*0K$$	Deletes KLM.
*-K$$	Deletes FGHIJ KLM.

4.35 L LINE COMMAND

Move the buffer pointer by lines.

FORMAT: nL

where:

n may be positive, negative, or zero.

Use the L command to move the buffer pointer over entire lines. The L command and its arguments are shown in Table 4-8.

Table 4-8
L Commands

Command	Argument	Function
L	1 assumed	Advances the pointer to the beginning of the line following the current line.
nL	n>0	Advances the pointer to the beginning of the nth line following the current line.
0L	0	Moves the pointer back to the beginning of the current line.
-L	-1 assumed	Moves the pointer back to the beginning of the line preceding the current line.
nL	n<0	Moves the pointer back to the beginning of the nth line preceding the current line.

If you attempt to move the buffer pointer backward beyond the position immediately prior to the first character in the buffer or forward beyond the position immediately after the last character in the buffer, TECO does not print an error message; however, TECO moves the pointer to the beginning or end of the editing buffer.

Examples:

*J3L$$ The J command moves the pointer to the beginning of the first line in the buffer. The 3L command then moves it to the beginning of the fourth line.

*ZJ-2L$$ The ZJ command moves the pointer to the end of the last line in the buffer. Then the -2L command moves the pointer to the beginning of the next to last line in the buffer (assuming that the last line is terminated by a line feed).

*L4C$$ Advance the pointer to the position following the fourth character in the next line.

4.36 M MACRO COMMAND

Execute the command string stored in a Q-register.

FORMAT: Mq
nMq
m,nMq

where:

q is a Q-register.

TECO command strings are composed of ASCII characters and, as such, you can insert or read them into the editing buffer just like any other text.

The command string stored in a Q-register is called a macro.

When a command string is in the editing buffer, you can edit it, but you may not execute it because, when it is in the buffer, it appears to be data to TECO. However, if you copy a command string from the editing buffer into a Q-register (using an X command), then this command string can be executed.

The command Mq (where q is one of the 36 Q-registers) executes the text in that register. Thus, entering an Mq command is analogous to calling a subroutine.

You may include any TECO command in the command string which is stored in and executed from the Q-register. The only restriction is that the commands must all be complete within the macro in the Q-register. For example, a command and its argument must not be split apart, one in the command string and the other in the Q-register. If you include iterations and conditional execution strings, these must also be complete within the register. If you use an O command in the macro, the tag to which it branches must also be in the register.

The forms of the M command are:

Mq	Execute the contents of Q-register q.
nMq	Execute the contents of Q-register q and use n as a numeric argument for the first command in the command string.
M,nMq	Execute the contents of Q-register q and use m,n as a numeric argument for the first command in the command string.

Example:

The following shows the creation of a macro to format a file into pages of 50 lines. The macro will be stored in a Q-register, and then be called to operate upon two files.

```
*EWFIFTY .TEC$$
*@I/Y<!ST!^N; 50S
                    $'FA OST$'12I$ 0,.P 0,.K> EC/$$
*HXA$$
*EC$$
*EBFILE1.FT$$
*MA$$
*EBFILE2.FT$$
*MA$$
*
```

1. This macro will be kept after it is created in the file FIFTY.TE.

2. The Y brings in the first page of text.

3. The ^N will signal if the end-of-file flag has been set.

4. If it has been set, ^N; will cause a branching from the iteration.

5. If the page has fewer than 50 lines, a new page is appended to the buffer and the loop is continued.

6. If there are at least 50 lines, a form feed is appended, the 50 lines are output, and then killed.

7. This will continue until the ^N flag is set, indicating the end of the file has been reached.

8. After the macro has been completed, it is stored with the HXA command into Q-register A.

9. FIFTY.TE is then closed.

10. FILE1.FT is then opened, and the Q-register is executed.

11. The procedure is repeated for FILE2.FT.

4.37 N NONSTOP SEARCH COMMAND

Search a file for a string until it is found.

FORMAT: nNtext$
n@N/text/

where n>0 and / is an arbitrary delimiter which is not one of the characters in "text." The N command combines the S command with input/output functions. Use the N command to search for a character string in a page of the input file which may not yet have been read into the editing buffer.

The N command may accept a colon modifier.

"Text" may include the following match control characters:

^Na	Match anything except "a"
^Q	Use the next command as match character
^S	Match on a separator
^X	Exempt position from match

The N command differs from the S command in that the former does not terminate at the end of the page currently in the buffer.

If TECO does not find a match for the search string between the current buffer pointer position and the end of the buffer, the current page is output, the buffer is cleared, and the next page is read in. The search then starts over at the beginning of the new page. This process continues until a match is found or the input file is completely written to the output file.

If an N search fails, the entire input file passes through the buffer and is written to the output file. TECO also clears the editing buffer but does not close the output file. Unless the : modifier was used or the search is within an iteration, an error message is typed to notify the user that the search has failed.

An N search cannot detect a match when the matching characters are split across two buffers. It also cannot detect a match if the characters are on separate lines unless you include the line separator in the search string.

The output function of the N command is exactly like the P command. If a form feed character was encountered when a given page was read in, TECO appends a form feed character to that page when it is output; otherwise, no form feed character is output.

If you do not include a text argument with a search command (for example, N$$), TECO executes the search using the last previous search command argument.

You may use the N command with a single numeric argument. The command nN_, in which n must be greater than 0, causes a search for the nth occurrence of the search string. When you omit n, TECO assumes n=1.

The _ search differs from the N search in that the former produces no output. However, both will search through the entire file following the buffer pointer until a match occurs.

Examples:

*NDIGITAL$$	If page 5 of the text is currently in the buffer and the string DIGITAL does not occur until page 15, this command causes pages 5 through 14 to be output and page 15 to be read in. The pointer will be set immediately after the L in DIGITAL.
*NLAST LIN PG1 1ST LIN PG2 $$?SRH FAILED *	If this string actually exists in the file but the two lines are not read into the same buffer, the N search will fail. In other words, a search can be dependent on how TECO brings information into the buffer.
*NMASSACHUSETTS$$?S?SRHLED *EF$$ *EBOUTPUT.FI$$ *Y$$ *NMASSACHUSETTS$$	An N search should not be used when an S search would suffice, because user errors with the N command, such as the spelling shown here, can cause considerable delay. In this example, the error causes two passes over the entire file instead of just one.

4.38 ^N

1. Do not match on character following the <^N>.

   FORMAT: <^N>a

   where a is an ASCII character and must be entered <CONTROL/N>.

2. Flag that indicates if the end of file has been reached.

   FORMAT: ^N

   where the command can be entered as a <CONTROL/N> or circumflex-N.

4.38.1 <^N> MATCH CONTROL CHARACTER

Use the <^N> match control command to accept any character in a particular position during a search except the character following the command. When TECO is searching for a string, any character except that following the <^N> is acceptable as a match. It differs from the <^X> match control character in that the <^X> will accept any character in that position as a match.

The <^N>-character combination counts as one character in the search string. However, it counts as 2 of the 31 characters permitted in a search string.

The combination <^N><^S> is legal; it commands TECO to accept any character which is not a separator as a match for this string.

The circumflex construction may not be used.

4.38.2 ^n End-of-File Indicator Command

The ^N command returns a value indicating if the end of the file has been reached. It is most often used in conjunction with a command that inputs data. The setting of the flag will then be the cause of jumping from an iteration or the fulfillment of a condition for a conditional execution command. It is initially set to 0. When the end of file has been reached, it is set to -1.

Example:

```
*<YITITLE
$PW^N;>$$
```

This command string inserts TITLE at the top of each page of a file.

```
*<^N;<FSABC$DEF$;>
   <FSXYZ$PQR$;>
   P> EX$$
```

This macro changes all occurrences of ABC to DEF and of XYZ to PQR in an entire file. The ^N terminates the iteration when the ^N flag is set to -1.

4.39 O GOTO COMMAND

Unconditional branch to a location.

FORMAT: Otag$

where:

tag — is composed of ASCII characters delimited by exclamation marks in the command string and by an ESCAPE in the O command.

The purpose of the tag following the O is to name the destination of the unconditional branch instruction. The tag location itself may be either before or after the O command in the command string. However, the branch cannot occur before the beginning of the current iteration. For example, if you enter the following command string:

```
...!tag1!...<...Otag1$...>
```

TECO will produce an error message.

However, the command string segment

```
...<!tag1!...Otag1$...>
```

is legal.

The O command causes the command string execution pointer to be moved to the first character following the exclamation point that terminates the tag, and command execution continues from that point.

Tags are ignored except when an O command forces TECO to scan the command string from them.

There is no restriction on the length of the tag (except that it must fit into the buffer).

The tag must also be in the same macro level; that is, you cannot branch from within a called macro to outside of the contents of the Q-register nor can you branch from outside of a macro into it.

4.40 <^O> COMMAND

1. Stop terminal output during typing out.

   FORMAT: <^O>

   where the command must be typed as <CONTROL/O>.

2. Change to octal radix.

   FORMAT: ^O

4.40.1 <^O> TYPE OUT COMMAND

During the execution of a type-out, you can stop the terminal output by typing <^O>. This command causes TECO to finish execution of the command string, omitting all further type-outs. You can enter this command only while TECO is actually typing out text at the terminal. If it is not, TECO ignores the ^O. The effect of this command does not carry over to the next command string.

Typing a second <^O> will cause the type-out to continue if the command string has not finished executing.

This command cannot be typed as circumflex-O.

Occasionally, the asterisk that TECO prints when a command finishes execution is also suppressed. If this occurs, you can type <CONTROL/U>. TECO then responds with an asterisk if it is waiting for a command.

Example:

The following example assumes the buffer contains the text shown at the right.

```
ABCDE
FGHIJ
KLMNO
PQRST
UVWXY
Z
```

```
*HT3K$$
ABCDE
EG^O
```

The user requests type-out of the whole buffer, but stops it with a <^O> (which echoes as ^O) immediately after the G is typed. However, a second HT command would show that the buffer now contains:

```
PQRST
UVWXY
Z
```

4.40.2 <^O> OCTAL RADIX COMMAND

The entering of a ^O or <CONTROL/O> changes the current radix to octal if TECO was not already in octal.

If the radix is in octal, this command is a no-op.

The initial radix of TECO is decimal.

4.41 P PAGE COMMAND

Write the buffer's contents to the output file.

FORMAT: nP
m,nP

where:

n may be positive, negative, or zero if there is one argument. If there are two arguments, m<n.

The P command outputs the text buffer, clears the buffer, and reads the next page of the input file into the buffer.

The P command performs two operations; when you type it with a single numeric argument (or no argument), the P command performs both output and input.

The P commands are described in Table 4-9.

Table 4-9
P Commands

Command	Argument	Function
P	1 assumed	Similar to PWHKY. Outputs the entire contents of the buffer, and then clears the buffer and reads in the next page of input. The buffer pointer is left at the beginning of the page that is read in. If there is no input file, or if no more data is in the input file, the buffer is left cleared. TECO appends a form feed character to the end of the output data only if the last input command was terminated by a form feed.
nP	n>0	Executes the P command n times. You can use this command to skip over several pages of text when no editing is required. The nP command causes the n pages of the input file -- starting with the page currently in the editing buffer -- to be output, and then the nth page after the current page to be brought in.
m,nP	m<n	When used with a pair of numeric arguments, the P command only outputs data; it neither clears any data from the buffer nor moves the buffer pointer. Also, the m,nP command never appends a form feed to the output unless you have inserted a form feed in the buffer between the mth and nth characters. The only action of m,nP is to output the (m+1)st through the nth characters in the buffer. (m,nP and m,nPW are equivalent.)

(continued on next page)

Table 4-9 (Cont.)
P Commands

Command	Argument	Function
HP	H=B,Z	Writes the buffer without appending a form feed to it; the buffer is not cleared, and no new data is read in. (HP and HPW are equivalent.) Although this form may resemble a single argument command (and thus could output a form feed), it is really a dual argument command.

As discussed above, you can use the nP command to skip over several pages to get to the next page where editing is required. The nP command can also be used with a very large argument, e.g., 1000, in order to skip to the end of the input file without doing any more editing.

Examples:

```
*PT$$
FIRST LINE OF PAGE
```
Output the current page, clear the buffer, read in the next page, and then type out the first line of the new page.

```
*.,ZP0,.P$$
```
This command string outputs the entire contents of the buffer, but it arranges the data as it is output. The .,ZP command outputs first part of the page that follows the buffer pointer. Then, the 0,.P command outputs that part of the data preceding the pointer. No form feed character is appended to either section of the output.

```
*.,ZP12I$0,.P$$
```
This performs the same action as the preceding command string except that it appends a form feed character to the part of the page that is output last. (12 is the ASCII code for form feed.)

```
*HK12I$HP$$
```
This command string produces a single blank page.

```
*8P$$
```
If page 6 of a file is in the editing buffer, this command causes pages 6 - 13 of the file to be output one after the other, and then read in page 14.

```
.MAKE FILE.FT
*Ipage of text$$
*PI2nd page of text$$
.
.
.
*PIlast page of text$$
*EC$$
```
This is the usual method for creating a text file.

4.42 PW PAGE WRITE COMMAND

Write contents of editing buffer to the output file and append a form feed.

FORMAT: nPW m,nPW

where:

n>0 if there is one argument. If there are two arguments, m<n.

The PW command is the basic output command because it does nothing but write data to the output file. Depending on the argument, the PW command writes all or some part of the data in the editing buffer. It does not, however, delete data from the buffer, and it never moves the buffer pointer.

The PW command writes the entire buffer and it always appends a form feed to the contents. The nPW command (n>0) also outputs n copies of the text in the buffer, appending a form feed to each copy. The PW command does not clear the buffer nor does it move the buffer pointer. (The same is also true of a P command used with two arguments.)

Note also that when you use a PW command with a single argument, a form feed character is always automatically sent to the output file immediately following the data from the buffer. TECO appends the form feed character to the outgoing data regardless of whether it encountered a form feed character when the data was read in, that is, regardless of the setting of the ^E form feed flag. This is not true of the P command.

Table 4-10 lists the PW commands.

Table 4-10
PW Commands

Command	Argument	Function
Pw	1 is assumed	Writes the contents of the buffer onto the output file and appends a form feed character. The buffer is not cleared and the pointer position remains unchanged.
nPW	n>0	Same as n<PW>. This executes the PW command n times.
m,nPW	m<n	Writes the contents of the buffer from the (m+1)th character through and including the nth character onto the output file. A form feed character is not appended to this output nor is the buffer cleared. The buffer pointer's position remains unchanged.
HPW	H=B,Z	Equivalent to PW except that a form feed is not appended to the output.

If the editing buffer is empty when TECO executes a PW command, no output of any kind takes place.

Examples:

*300PW$$	This command outputs 300 copies of the current page.
*PWJKIJ.DOE$ PW$$	This command outputs the current buffer, modifies the first line, and then outputs the buffer again.
*HK12I$PW$$	This produces two successive blank pages.

4.43 Q Q-REGISTER COMMAND

Return contents of a Q-register.

FORMAT: Qq

where:

q is a Q-register designator.

Use the Qq command (where q is one of the 36 Q-registers) to read the numeric value in a register. Qq has no function other than to return the value in the specified Q-register as a numeric argument to another command. It does not alter the value in the Q-register. Qq is often used in conjunction with conditional commands.

Examples:

*QR-3UR$$	This command string subtracts 3 from the value in Q-register R and then inserts the new value into the register.

4.44 <^Q>

1. Execution command that resumes a type-out that was frozen by a <^S>.

   FORMAT: <^Q>

   where you must type the command as <CONTROL/Q>.

2. Accept the next command as a character in a search.

   FORMAT: <^Q>c

   where c is a TECO command. This command must be entered as <CONTROL/Q>.

4.44.1 <^Q> TYPE-OUT CONTROL COMMAND

The <^Q> is an execution time command that resumes terminal output which had been stopped by a <^S>.

The circumflex construction may not be used.

4.44.2 <^Q> MATCH CONTROL CHARACTER

<^Q> is a match control character that accepts the next character as text. A <^Q> character in a search command argument indicates that TECO should interpret the character following the command literally rather than as a command. This character may be used to search for ^S, ^N, and ^S characters.

It does not count as one of the maximum 31 characters in the search command argument.

The circumflex construction may not be used.

4.45 R REVERSE COMMAND

Move the buffer pointer backwards.

FORMAT: nR

where n may be positive, negative, or zero.

The R command is a buffer pointer positioning command, and is the reverse of the C command. The forms of the R command are listed in Table 4-11.

Table 4-11
R Commands

Command	Argument	Function
nR	n>0	Move the pointer backwards over n characters in the buffer from the current buffer pointer position. nR is equivalent to -nC.
R	1 is assumed	Move the pointer backward one character.
0R	0	This is a null command.
-R	-1 is assumed	Move the pointer forward one character. This is equivalent to C.
nR	n<0	Move the pointer forward over n characters in the buffer from the current buffer pointer position. nR is equivalent to nC.

If an R command attempts to move the pointer across either buffer boundary, TECO ignores the command and prints the error message ?POP ATTEMPT TO MOVE POINTER OFF PAGE.

Example:

*0L2R$$

The 0L command moves the pointer back to the beginning of the current line. Then, the 2R command moves it back past the last two characters in the preceding line.

4.46 S SEARCH COMMAND

Search for a character string.

FORMAT: nStext n@S/text/

where n>0 and text is less than 32 characters long.

Use the S command to search for a character string within the current editing buffer. If the string is not found between the current buffer pointer position and the end of the buffer, the search fails. After an unsuccessful S search, the buffer pointer is reset to the beginning of the buffer, and, unless you have used the : modifier or the search is within an iteration, TECO prints the message.

Type the search string as an alphanumeric argument following the S. Terminate it with an ESCAPE. "Text" can contain any character except those listed in Table 1-1, Special Characters, but may not contain more than 31 characters.

The S command may be used with a single numeric argument. The command nS causes a search for the nth occurrence of the specified search string. When n is omitted, n=1 is assumed. n must be greater than 0.

If you are searching for text that contains an ESCAPE, you can use an @ modifier to the S command.

If you do not include a text argument with an S command (for example, S$$), TECO executes the search using the last previous search command argument.

The S command may use the following match control characters:

^Na	Do not match on "a"
^Q	Use next command as match character
^S	Match on a separator
^X	Exempt next position from matching

Examples:

*SA<TAB>B$$	This causes the pointer to be positioned immediately after the B, in the first occurrence of the string A<TAB>B after the current position of the pointer.
*SNIX$$?SRH SEARCH FAILED	The string NIX is not found between the current pointer position and the end of the buffer. The error message is typed and the pointer is moved to the beginning of the buffer. You may have typed an incorrect search string, the pointer may have been positioned somewhere in the buffer after the N, or the string NIX may not be in the editing buffer.
*@3S+$+IEF$$	The command @3S+$+ searches for the third occurrence of the ESCAPE character following the buffer pointer. When this ESCAPE is found, the characters EF are inserted immediately after it. The + characters serve as the delimiters for the 1-character search string $. The + characters are not part of the search string.

4.47 ^S

1. Store a completed or aborted command string in Q-register Z.

   FORMAT: <^S>

   where you must type a <CONTROL/S>.

2. Freeze the output display.

   FORMAT: <^S>

   where you must type a <CONTROL/S>.

3. Accept any nonalphanumeric character as a match for this relative position in the search string.

   FORMAT: ^S

4.47.1 <^S> STORE COMMAND STRING COMMAND

After TECO completes execution of a command string (or if you abort it with <^G><^G>), you may store it in Q-register Z by typing a <^S> as the first command in the next command string. The previous contents, if any, of Q-register Z are destroyed.

This command causes the entire previous command string, less one of the two concluding ESCAPEs, to be stored in Q-register Z. If you abort the command string with a <^G><^G>, neither <^G> will be stored with the command string. This command has this function only when it is the first command in a command string.

If you had intended to use the <^S> but instead typed some other command first, you may recover the ability to use this command by typing enough <DELETE>s to cause TECO to respond with a carriage return/line feed and a new asterisk. This technique will not work perfectly if some of the characters typed before the <^S> were break characters (ESCAPE, carriage return, etc.). If you typed some break characters, some of the leading characters of the preceding command string will be overwritten.

This command is especially useful when an error occurs in a long command string.

4.47.2 <^S> FREEZE OUTPUT COMMAND

If you type a <CONTROL/S> while TECO is typing text, the monitor freezes the output. The output resumes at the place of interruption if you type a <CONTROL/Q>.

4.47.3 <^S> MATCH CONTROL CHARACTER

<^S> may also act as a match control character. When you enter it, TECO accepts any separator (that is, any nonalphanumeric character) as a match in a search.

4.48 I TYPE COMMAND

Type out text in the editing buffer.

FORMAT: nT

where n may be positive, negative, or zero.

You can type any part of the text in the editing buffer for examination by using the T command. The text that TECO types depends on the position of the buffer pointer and the argument(s) given. The T command never moves the buffer pointer.

When preceded with a single numeric argument, T is a line-oriented command; when preceded with a pair of numeric arguments, T is a character-oriented command. The T commands are described in Table 4-12.

Table 4-12
T Commands

Command	Argument	Function
T	1 assumed	Types out everything from the buffer pointer through the next line terminator. If the pointer is at the beginning of a line, T causes the entire line to be TECO typed out. If the pointer is in the middle of a line, T causes that portion of the line following the pointer to be typed out.
nT	n>0	Types out everything from the buffer pointer through the nth line terminator following it. If the pointer is at the beginning of a line, this command types out the next n lines (including the current line).
0T	0	Types out everything from the beginning of the current line up to the pointer. This command is especially useful for determining the position of the buffer pointer.
-T	-1 assumed	Types out everything in the line preceding the current line, plus everything in the current line up to the pointer.
nT	n<0	Types out everything in the n lines preceding the current line, plus everything in the current line up to the pointer.
m,nT	m<n	Types out the (m+1)st through the nth characters in the buffer.

(continued on next page)

Table 4-12 (Cont.)
T Commands

Command	Argument	Function
.,.+nT	n>0	Types the n characters immediately following the buffer pointer.
.,.-nT	n<0	Types the n characters immediately preceding the buffer pointer.
HT	H=B,Z	Types out the entire contents of the buffer.

Examples:

The following examples assume the buffer contains the text shown at the right, with the buffer pointer positioned between the M and the N.

```
ABCDE
FGHIJ
KLMNO
PQRST
UVWXY
Z
```

```
*T$$
NO

*3T$$
NO
PQRST
UVWXY
```

```
*0T$$
KLM*
```

Note that no carriage return/line feed exists between the beginning of the line on which the pointer is located and the pointer itself; therefore, none is typed. The second asterisk indicates that TECO is ready for the next command.

```
*0TT$$
KLMNO
```

This pair of commands causes the line to be typed out without moving the pointer.

```
*-2T$$
ABCDE
FGHIJ
KLM*
```

```
*.,.+6T$$
NO
PQ*
*.,.-2T$$
LM
```

The six characters typed are NO, carriage return/line feed, and PQ

```
*0LT$$
KLMNO
```

This pair of commands types out the entire current line and leaves the pointer at the beginning of the line.

4.49 ^T TYPE-IN COMMAND

1. Use the ASCII code for the next character typed at the terminal as an argument.

   FORMAT: ^T

2. Type out the character whose ASCII code precedes the command.

   FORMAT: n^T

   where:

   n is the ASCII code for a character.

4.49.1 ^T INPUT COMMAND

^T is equivalent to the ASCII code for the next character you type at the terminal. When TECO executes a command string, every ^T character encountered causes it to pause and accept one character typed at the terminal. TECO then substitutes the ASCII code for this character for the ^T.

This command is useful only as a numeric argument for another command.

It is often used with an ^Atext<^A> message string preceding it. The message string informs the user that TECO is waiting for a character to be typed in.

Example:

```
*<SFUNCTION $^A                 Here, the ^T command is used as the
FUNCTION LETTER <^A>            argument for an nI$ command. This
:^TI$>$$                        command string inserts the letter typed
FUNCTION LETTER M               in following each occurrence of the
FUNCTION LETTER N               string FUNCTION that is found by the
FUNCTION LETTER C               search command.

*^TUC$$                         Places the ASCII value of the typed
                                character in Q-register C.
```

4.49.2 ^T TYPEOUT COMMAND

The n^T command types out the character whose ASCII code is n.

Example:

```
*J<S            This macro will type out each line of a file. It
$;2R0TL         will then wait for you to type a K to delete the
<^T>-^^K"E      line or anything else to type the next line.
-K'>$$
```

4.50 <TAB> INSERT COMMAND

Insert text and <TAB> into the editing buffer.

FORMAT: <TAB>text$

The tab command is equivalent to the I command, except that the tab command causes TECO to insert the tab itself as well as all the following text up to the ESCAPE. In other words, if the first character of a text string to be inserted by an I command is a tab, you may omit the I.

The numer of characters in the text is limited by the amount of core available for command string storage. During normal editing jobs, DIGITAL recommends that you limit insertions to about 10 or 15 lines each.

If a long insertion command begins to exceed the TECO command storage capacity, TECO rings the terminal bell once when ten characters of storage remain and once after each additional character entered. When this occurs, terminate the command string immediately. Entering more than ten additional characters into the current command string causes a fatal error.

Example:

If the buffer contains ABCDEF with the buffer pointer positioned between the D and E, then:

```
*<TAB>XYX$$    produces ABCD    XYZEF
```

The pointer is now between the Z and E.

4.51 U COMMAND

Insert number into a Q-register.

FORMAT: nUq

where -4095<n<4095 and q is a Q-register designator.

The nUq command stores the integer n in Q-register q (where q is one of the 36 Q-registers). Anything previously in the numeric part of the Q-register is destroyed.

Example:

```
*SLINE 1$0L.U1
SLINE 2$.U2
Q1,12K$$
```

This command string will delete the text beginning at line 1 and preceding line 2.

4.52 <^U> COMMAND

Erase the current line from the command string buffer.

FORMAT: <^U>

where you must type <CONTROL/U>.

When you enter a <^U> while entering text, TECO erases all text in the command string back to the last carriage return/line feed. You may then resume entering the command string as if the deleted text had never been entered; that is, the command string <^U> does not close the command string register.

TECO echoes this command on the terminal as a ^U.

This command may not be entered using the circumflex construction.

You may use the <^U> to erase more than one command string line if you type a <DELETE> after the <^U> has deleted the line.

4.53 ^Uqtext$ COMMAND

Enter the text following the command into a Q-register.

FORMAT: ^Uqtext$
        @^Uq/text/

where:

q	is a Q-register designator
text	is the character string to be inserted into the Q-register
/	is an arbitrary delimiter which is not one of the characters in "text."

Text is normally placed into a Q-register by copying it from the editing buffer (through the use of the Xq command). The ^U command gives you the option of directly inserting a text into a Q-register. Specifically, TECO places the text and the ASCII character string following the Q-register designator and preceding the ESCAPE into the Q-register when it begins executing the command string.

You can enter the command directly only by using the circumflex construction.

Because the text often contains ESCAPE characters, DIGITAL recommends that you use the @ modifier form.

4.54 W WINDOW COMMAND

1. Perform a display cycle.

   FORMAT: W

2. Show n lines before and after the cursor.

   FORMAT: nW

   where n>0

NOTE

These commands are only used with nonrefreshable CRTs.

4.54.1 W Command

Perform a display cycle. Update the display from the current buffer.

4.54.2 nW Command

Set display mode to show n lines before and after the line containing the cursor. The initial display shows three lines above and below the cursor.

4.55 X EXTRACT COMMAND

Insert text into a Q-register.

FORMAT: nXq
m,nXq

where:

n is positive, negative, or zero if there is one argument. If there are two arguments, m<n.

q is a Q-register designator.

The X command copies characters from the editing buffer into a Q-register. TECO does not remove these characters from the editing buffer. Consequently, an X command is often followed with a K command. Any data previously in the data part of the Q-register is destroyed.

The X commands are listed in Table 4-13.

Table 4-13
X Commands

Command	Argument	Function
nXq	n>0	Copies everything, from the current buffer pointer position to the nth following line terminator character, into Q-register q.
Xq	1 is assumed	Copies the text of the current line, from the buffer pointer position to the next line terminator, into Q-register q.
0Xq	0	Copies the text from the beginning current line preceding the buffer pointer to the buffer pointer position into Q-register q.
-Xq	-1 is assumed	Copies the text of the line preceding the current line, and the current line to the buffer pointer position, into Q-register q.
m,nXq	m<n	Copies the (m+1)st through the nth characters into Q-register q.
HXq	H=B,Z	Copies the entire editing buffer into Q-register q.

If an X command requires more memory for storage than is available, TECO prints the error message ?QMO Q-REGISTER MEMORY OVERFLOW and does not execute the command.

Example:

*0,.X10,.KZJG1$$	This command string moves everything to the left of the pointer from its position at the beginning of the page to the end of the page. The 0,.X1 command puts everything from the top of the page to the pointer in Q-register 1. The 0,.K command then deletes this data from the buffer. The ZJ command moves the pointer to the end of the page. At this point, the command G1 copies the contents of Q-register 1 into the buffer at the position of the pointer.

4.56 <^X>

Accept any character in this relative position in a search string as a match.

FORMAT: <^X>

where the command must be typed <CONTROL/X>.

An <^X> character indicates that this position in the character string is unimportant and that TECO will accept any ASCII character in this position as a match for the search string.

The circumflex construction may not be used.

4.57 Y YANK COMMAND

Read a new page into the editing buffer.

FORMAT: Y

The Y command first clears the editing buffer and then reads text into the buffer until one of the following conditions is met:

1. The end of the input file is reached;
2. A form feed character is read;
3. The buffer is two-thirds full and a line feed is read (or filled to within 128 characters of capacity); or
4. The buffer is completely filled.

The usual effect of the Y command is to clear the editing buffer (execute an HK command) and then read the next page of the input file into it. Less than the entire next page is read in only if that page is too large to fit within two-thirds of the buffer's capacity.

If the end of the input file has previously been read, the Y command only clears the buffer.

If a form feed is read (that is, if input stops because of condition 2), the form feed flag ^E is set to -1. TECO does not write the form feed into the buffer with the rest of the text. A succeeding input command begins input at the character following the form feed.

If a form feed is not read, the form feed flag is set to 0, and the next input command begins input at the character following the last character previously read in. You may test the form feed flag, but ordinarily this is not necessary.

TECO automatically executes a single Y command in response to the TECO filespec initialization command. This causes the first page of the input file to be read into the buffer before TECO prints the first asterisk.

The Y command sets the buffer pointer to the position preceding the first character in the buffer.

The Y command does not accept a numeric argument. If you wish to use multiple Y commands, you may type n<y>, where n is the number of pages to be ignored.

The Y command aborts if the text buffer is not empty and an output file is open.

Examples:

*ERREPORT.TE$Y$$	This command string opens the file REPORT.TE for input and reads in the first page of that file.
*ERDTA3:DATA.FT$YYY$$	This command string reads in and discards the first two pages of the DECtape file DATA.FT, and then reads in the third page of that file. If an output file were open, you would have to type 2<HK>Y.

4.58 Z POSITION INDICATOR

Indicates the last position in the editing buffer.

FORMAT: Z

Z is a buffer position indicator that equals the number of characters in the buffer. Thus, Z is a number that is the count of the characters in the buffer.

While Z is often used as an argument to a command, you can also use it in arithmetic expressions.

Example:

One common use of Z is to verify if characters have been read into the editing buffer; for example, the command string segment

```
...ZU1 A Z-Q1"E...
```

stores a number which is the count of the characters in the buffer and appends a new page. The second Z indicates the new number of characters in the buffer. If, when the former count is subtracted from it, the remainder equals zero, then no characters were read in.

4.59 !tag!

A string of ASCII characters used to identify a location in a command string.

FORMAT: !tag!

where ! precedes and follows the ASCII string.

When an O command directs the execution of a command string from the linear order of execution, the place to which TECO is directed is called a tag. The tag is delimited both before and after with exclamation points.

Tags may be thought of in much the same way as labels in other programming languages.

The length of a tag is limited solely by the command string register.

Because you do not have to reference tags, they can serve as comments within a command string or macro. (Normally, comments are included only in lengthy TECO macros that will be maintained.)

4.60 " BRANCHING COMMANDS

Depending upon the argument, skip any command that precedes an apostrophe at the same nesting level.

FORMAT: n"X....'

where:

n	is the argument to the conditional execution command
"	signals that the command is a conditional execution command
X	is the condition to matched
'	represents the place to begin execution if the condition is not satisfied.

In this command, n is the numeric argument on which TECO bases its decision to execute a command string. The quotation mark (") is the first character of all conditional execution commands. Immediately following the quotation mark is a character which must be one of those listed in Table 4-14. Terminate this command string with an apostrophe (').

If n satisfies the condition, TECO executes all the commands in the usual manner until the apostrophe. If there is no branch command within the range "...', then after TECO executes the last command in the range, command execution falls through the apostrophe, and execution begins with the next command following it. If n does not satisfy the condition, then TECO skips all the commands before the apostrophe, and command execution continues with the first command following the apostrophe.

Use the quotation mark and apostrophe only in matching pairs. You may nest them in the same manner that parentheses surrounding arithmetic expressions can be nested.

As TECO scans for the matching single quote, it keeps an iteration count for each level of nested iteration which it finds. TECO then ignores any single quotes which occur at a nesting level greater than 0.

NOTE

All conditionals must end at the same macro level in which they begin.

The conditional execution commands are in Table 4-14.

Table 4-14
Conditional Execution Commands

Command	Function
n"C	Execute the commands that follow if n is an ASCII character (A-Z, a-z, 0-9).
n"E	Execute the commands that follow if n=0.
n"F	Execute the commands that follow if n represents a false (flag is off) (i.e., if n=0).
n"G	Execute the commands that follow if n>0.
n"L	Execute the commands that follow if n<0.
n"N	Execute the commands that follow if not = 0.
n"R	Execute the commands that follow if n is the ASCII code for an alphanumeric character (A-Z, a-z, 0-9).
n"S	Execute the commands that follow if n represents success (i.e., if n<0).
n"T	Execute the commands that follow if n represents true (flag is on) (i.e., if n<0).
n"U	Execute the commands that follow if n represents unsuccessful (i.e., if n=0).
n"<	Execute the commands that follow if n<0. Same as n"G.
n">	Execute the commands that follow if n>0. Same as n"L.

Examples:

```
*!START!J<TAB>PDP-8 TECO
$                                       !INSERT PAGE HEADING!
<S 5K$;R-D16$>                          !CHANGE 5K TO 6K!
<SWAR$;-3DILOVE$>                       !CHANGE WAR TO LOVE!
PZ"NOSTART$                             !GET NEXT PAGE AND!
'EF$$                                   !RESTART IF NOT NULL!
```

This small editing program also contains an example of the O unconditional branching command, that is, the OSTART$ command which causes a jump back to !START!. String tags are also used purely for documentation: for example, !INSERT PAGE HEADING!.

This example also shows how a conditional execution command may be combined with an O command to produce a conditional branch.

1. After TECO performs all three of the editing functions on the page, it executes the P command to write this page and read in the next.

2. The program then tests Z (the number of characters in the buffer) to determine if any data was read in.

3. If Z does not equal 0, data was read in; therefore, a branch is taken to restart the program.

4. When Z-0, the command OSTART$ is skipped, and execution branches to the concluding EF command.

This technique fails when a file contains null pages (consecutive form feed characters). The ^N end-of-file test may be preferred.

```
*YZ"N!##! Z-4000+1"G4000J
0L 12I$ 0,.P0,.K 0###$ 'ZJ
A .-Z"NO###$ ' 'PEF$$
*
```

This slightly more complex command string shows how conditional execution commands may be nested. If the first Y command produces no data, the "N command sends execution to the matching apostrophe on the right. This is the last apostrophe, immediately prior to the PEF. Otherwise, TECO executes the commands following the "N.

The function of this command string is to convert a file with pages of arbitrary length to one with pages of approximately 4000 characters each.

The command string operates as follows:

1. Z-4000+1"G means if Z>4000 (that is, there are at least 4000 characters on the current page), execute the following commands; otherwise, skip to the matching apostrophe (between $ and Z).

2. If Z>4000, 4000J and 0L moves the pointer to the end of the last complete line before the 4000th character in the buffer.

3. Then, 12I$ and 0,.P outputs this much of the buffer with a form feed character after it, and 0,.K deletes that which has been output.

4. Now, go back to !##! and test Z again. Stay in this loop until Z<4000.

5. Execution then skips to the apostrophe. ZJ moves the pointer to the end of the current buffer.

6. A appends another page., but leaves the pointer (.) at the end of the previous page.

7. .-Z"N checks to determine if any data was actually read in. If so, the loop is re-entered at !##!; otherwise, the end of the file has been reached.

8. When .-Z=0, execution skips to the matching apostrophe and then falls through the next apostrophe to the PEF that closes the output file.

NOTE

```
Y<!ST!^N;  Z-4000"L AOST$ '4000J 0L 121$
0,.P 0,.K> EC$$
```

```
*<NSIN$;:SCOS$"S-3DITAN$'ZJ>$$
```

This example shows how the value returned by a colon search can be used as the argument for a conditional execution command.

1. The N command searches through the file for the first occurrence of SIN on any page.

2. When SIN is found, the command :SCOS$ checks for an occurrence of COS following SIN on the same page.

3. The colon search command returns the value -1 if the search is successful, and 0 if there is no COS following SIN on the page.

4. This value is then used as the numeric argument for the "S command.

5. If :SCOS$ has a value of -1, TAN replaces the occurrence of COS that was found.

6. If :SCOS$ has a value of 0, the commands -3DITAN$ are skipped.

7. We then jump to the end of this page, ignoring all further occurrences of SIN and COS on it, and continue the iteration process.

4.61 % COMMAND

Change the number stored in a Q-register by n.

FORMAT: n%q

where:

n is a number

q is a Q-register designator.

The command n%q adds n to the integer in Q-register g and then returns the new value in the same manner as a Qq command. If you omit n, 1 is assumed (that is, 1%q). Note that this command returns a value as well as incrementing the stored value.

If you wish to increment the value in the Q-register but do not want the returned value to be used as an argument for the next command, type an ESCAPE after the n%q command (that is, n%q$).

4.62 . POSITION INDICATOR

Return a number equal to the number of characters preceding the buffer pointer in the editing buffer.

FORMAT: .

A period (.) equals the number of characters, that is, the number of characters from the beginning of the buffer to the present buffer pointer position, to the left of the current position of the buffer pointer, and hence represents the buffer pointer position itself.

You can use the period in arithmetic expressions.

4.63 : MODIFIER

1. Return a value if a search fails.

   FORMAT: :nStext$
           :n@S/text/

   where:

   S represents any search command

   / is an arbitrary delimiter which is not one of the characters in "text."

2. Do not output a carriage return after typing out a number.

   FORMAT: :=
           :=

3 Cause a Gq command to type-out its contents without inserting the contents into buffer.

   FORMAT: 3:Gq

   where:

   q is a Q-register designator.

4.63.1 : S Modifier

Use the colon modifier to alter the execution of a search command in the event a search fails. Normally, a search that fails causes TECO to print an error message; if you use the colon modifier, however, no error message is printed. Instead, every colon search returns a numeric value that TECO can print out, store in a Q-register, or test by a conditional branch.

A colon search command returns the value -1 if the search is successful, and the value 0 if the search fails.

The general form of a colon search command is the same for S, FS, N, and FN searches:

```
*nSstring$
```

The colon precedes the search command letter and its numeric argument, if any. Both the colon and @ modifier may be used on a search command in either order.

Just as the Z command takes on a value that may be used as a numeric argument, so also the command :Sstring$ takes on a value of 0 or -1 after TECO executes the search. If this is the last command in a command string, or if the command following it does not take a numeric argument, the value returned by the colon is discarded. Consequently, a command that takes a numeric argument should follow a colon search.

The colon search commands reposition the buffer pointer in the same manner as other search commands, regardless of whether the returned value is used.

The primary use of the colon search is in programmed editing. The command is usually followed by a conditional command.

4.63.2 : Numerical Type-out Modifier

See the = and == discussion for information on how : modifies numeric type-out commands.

4.63.3 : Q-register Type-out Command

See the G command for information.

4.64 : COMMAND

1. If an argument is positive or zero, jump out of current iteration field.

   FORMAT: n;

   where:

   n is a number, either explicit or returned.

2. If the current search failed, jump out of current iteration field.

   FORMAT: ;

You can terminate repetition of a command string loop before the iteration count is satisfied by using the conditional iteration exit command, semicolon (;). You can use the ; command only within angle brackets. You may use it with or without a numeric argument.

When you use a semicolon without a numeric argument, it evaluates the outcome of the last search (of any kind) that was executed before TECO encountered the semicolon. If this search was successful, command execution continues within the loop, as if no semicolon were present. If, however, the most recent search failed, the ; command causes all those commands that follow the semicolon in the loop to be skipped over, and command execution passes to the first command following the right angle bracket, which closes the innermost loop containing the semicolon.

NOTE

Within a command loop, all searches are colon searches. They do not generate error messages when a failure occurs; instead, they return a value of -1 if successful and 0 if unsuccessful.

TECO executes all unmodified search commands entered within command loops as though they were preceded by a colon and followed by a semicolon.

The semicolon command is most often used with a numeric argument. TECO ignores the command n; if n<0. However, if n>0, the command n; causes command execution to exit from the loop, just as a ; command exits from the loop when a search fails.

4.65 <...> COMMAND

Continue executing the command string until conditions are met.

FORMAT: n<...>

where:

n	is a positive number. If n is omitted, the command is executed an infinite number of times or until an exit condition occurs within the loop.

You can cause a group of commands to be iterated (repeatedly executed) any number of times by placing the commands within angle brackets. The left angle bracket marks the beginning of a command string loop and the right angle bracket marks the end of the loop.

These command string loops can be nested in the same manner that arithmetic expressions are nested within parentheses. Loops should be nested to no more than approximately ten levels; otherwise, a pushdown list overflow may occur.

You may use a numeric argument to type the number of times a given loop is executed. The argument is placed before the left angle bracket in the form n<...>. This causes the group of commands within the brackets to be iterated n times. In a command of the form <...>, if the argument is less than or equal to zero, Teco skips the commands within the angle brackets. If no argument is given, the number of iterations is assumed to be infinite.

You should use only the ; and 0 commands to exit from a loop. The flow control commands (") should not be used.

Example:

```
*J8<<TAB>$L>$$
```

This commmand string inserts a tab at the beginning of the first eight lines in the buffer and leaves the pointer positioned at the beginning of the ninth line. The J command starts the pointer off at the beginning of the first line. The first command in the loop, <TAB>$, inserts a tab. Then the L command moves the pointer to the next line to prepare for the next iteration of the loop.

```
*J<0LIJAN$
FS1969$1970$;>
HT$$
JAN REPORT
DEPT:
JAN 1970 SALES
         WHOLESALE
         RETAIL
JAN 1970 EXPENSES
         OVERHEAD
         ADVERTISING
```

This command string inserts JAN at the beginning of the first line in the buffer and at the beginning of each line that contains 1970. It also changes the 69 in every occurrence of 1969 to 1970. The action is similar to the way in which the J command starts the operation at the beginning of the buffer. The first execution of the 0L does nothing. IJAN$ then

JAN 1970 RETURNS JAN 1970 INVENTORY	inserts JAN at the beginning of the first line. Now a search is made for 1969. When it is found, FS1969$1970$ changes 1969 to 1970. This completes the first iteration; execution loops the first iteration and then loops back to the <. 0L moves the pointer to the beginning of the line where the 1969 was found. Here JAN is inserted and a search is begun for the next 1969. This continues until the search command fails to find another 1969. When the search fails, the pointer is moved to the beginning of the buffer. HT is the next command which is executed. (This macro assumes that no line contains more than one 1969.)
*EBfilespec$50000<yhp>ex$$	This removes all form feeds from a file.
*<FSREAD$WRITE$;>$$	This command causes a search of the current page for all occurrences of the string READ and replacement of them with the string WRITE.
*<@FN/ERROR//;>$$	This command causes TECO to search all of the following pages for the string ERROR and delete every occurrence of it. The @ construction must be used in this case because it allows you to specify a delimiter other than $. The delimiter must be specified twice after the string: the first to delimit the string and the second to indicate that a replacement string is not present. If $ were used as the delimiter, a double $ would be present causing an erroneous termination of the command string.

4.66 = NUMERICAL TYPE-OUT COMMAND

Type out a number in decimal or octal (with or without a following carriage return).

FORMAT:
```
=
:=
==
:==
```

where a single = means type out in decimal, a double = means type out in octal, and the : modifier means do not type a carriage return after typing.

Table 4-15 outlines these commands.

Table 4-15
= Commands

Command	Function
n=	Type the value of n in decimal followed by a carriage return/line feed.
n:=	Same as n= except the carriage return/line feed is suppressed.
n==	Type the value of n as an unsigned octal number followed by a carriage return/line feed.
n:==	Same as n== except the carriage return/line feed is suppressed.

Examples:

```
*YZ==$$
2529
```
This reads in a page and then types out the (decimal) number of characters in the page.

```
*IOA==$$
40
```
This types the octal representation of the next character in the buffer.

```
*YNCHAPTER $=$$
16
```
This command searches for the next chapter heading and then types out the number of the chapter. The buffer pointer points to the location immediately following the 6.

4.67 ? COMMAND

1. Trace the execution of a TECO command string.

   FORMAT: ?

   where:

   ? is inserted into the command string.

2. If an error occurs, type out the commands executed in the command string.

   FORMAT: ?

   where:

   ? must be the first character typed after the prompting asterisk.

4.67.1 ? Trace Command

The use of a question mark causes TECO to enter trace mode. In trace mode, TECO types out each command as it is executed. A second question mark takes TECO out of trace mode.

The ? command traces only the commands that TECO executes; that is, if a branching command causes a command to be skipped over, TECO does not print it.

Example:

```
*JHT?
!L!1A-9"N
          !M!1A-58"NCOM$
                    'CD<TAB>$
       'LOL$$
AB: LINE 1
          LINE 2
C:  LINE 3
          LINE 4
!L!1A-9"N!M!1A-58"NCOM$1A-58"NCO!M!1A-58
"NCD       $'LOL$1A-9"NLO!L!1A-9"N!M!1A-58"
NCO!M!1A-58"NCD $'LO!L!1A-9"NLO!L!1A-9!M
11A-58"NC?POP
*J?HT$$
J?
```

```
AB: LINE 1
    LINE 2
LINE 3
    LINE 4
```

After the first ? command, TECO begins typing out each command as it is executed. This enables you to see exactly what the command string is doing. The attempt to move the pointer beyond the end of the fourth (and last) line with the C command causes the ?POP error message.

The second question mark command turns off the trace feature so that the HT following it is not printed.

4.67.2 ? Error Command

In some cases, you may not be able to determine immediately which command in the string caused an error. This could occur, for example, if the command string had several commands of the same type. In such a case, use the question mark command to obtain more information. The question mark command, when you type it immediately -- and only immediately -- after an error message type-out, causes TECO to type out the entire command string up to and including the bad command.

NOTE

When you use this form of the ? command, it is not necessary to type an ESCAPE or any other delimiting character following the question mark.

TECO always types a second question mark after the last character of the group. The character at which the error was detected is the last character before the question mark.

4.68 @ TEXT DELIMITER MODIFIER

Modifies the next command which takes a text argument to use the delimiter form.

FORMAT: @x/text/

where:

x — is a command that takes a text argument (see Table 4-16)

/ — is an arbitrary delimiter which is not one of the characters in "text."

Use an @ modifier to alter the method by which TECO reads a command's text. The general form and the commands for which the @ modifier is applicable are listed in Table 4-16.

Table 4-16
@ Commands

@nS/text/	Search page
@nN/text/	Search file
@n_/text/	Search file
@nFS/text1/text2/	Search page and replace
@nFS/text1//	Search page and delete
@nFN/text1/text2/	Search file and replace
@nFN/text1//	Search file and delete
@I/text/	Insert text
n@I//	Insert character
@EB/filespec/	Edit backup
@EG//	Exit and execute last
@EG/text/	Exit and execute text
@ER/filespec/	Edit read
@EW/filespec/	Edit write
@^Uq/text/	Insert text in Q-register

Place the @ modifier before the command and before a numeric argument. When you use the @ modifier, the text string is delimited not by the command and an ESCAPE but by the first character you type after the command and the next recurrence of this command. In the above examples, the delimiting character is a slash. The delimiting character may be any character except a character that appears in the text itself.

Using the @ modifier, you may enter single (but not double) ESCAPEs into a text string.

Use the @ modifier in the above commands to separate the strings with a delimiting character other than an ESCAPE. This is useful in cases where a double ESCAPE cannot terminate the command. For example, if you are searching for a string, then deleting it without replacement, you would ordinarily type a double <ESCAPE> after an FS or FN command. However, if this were within a command string, the command would be terminated at this point unless you used an @ modifier.

4.69 \ COMMAND

1. Returns the numbers following the buffer pointer.

   FORMAT: \

2. Insert a character numeric string into the buffer.

   FORMAT: n\

   where:

   n is the number to be inserted.

4.69.1 \ Command

The \ command (without a numeric argument) is equivalent to the value of the digit string (optionally preceded by a + or - sign) immediately following the current position of the buffer pointer. The value is terminated by the first alphabetic character that TECO encounters. If there is no digit, TECO types a 0.

TECO interprets the digit string in the current radix.

Example:

```
*YNCHAPTER $\=$$
16
```

This command searches for the next chapter heading and then types out the number of the chapter. The buffer pointer is positioned immediately following the 6 after this command has been executed.

4.69.2 n\ Insertion Command

Use the n\ command to insert the ASCII representation of a number n into the buffer. TECO inserts the number in the current radix. For example, 349 inserts the ASCII characters 3, 4, and 9 into the buffer immediately preceding the pointer. Note that n does not have to be a number typed in by the user. It can be a value which some other TECO command returns.

4.70 ^^x COMMAND

Return the ASCII value of a character.

FORMAT: ^^x

where:

x is an ASCII character.

The ^^ command, when followed by an arbitrary character x, is equivalent to the ASCII value of that character. For example, in the command ^^A, the character A is an argument for ^^A and TECO does not interpret it as a command.

Example:

*^^MU0$$	This command stores the ASCII value of the letter M (77) into Q-register 0.

4.71 _ COMMAND

Search for a character string and discard all pages before the string is found.

FORMAT: n_text$
n@_/text/

where:

n	indicates what occurrence of the string is to be found
/	is an arbitrary delimiter which is not one of the characters in "text."

The backarrow command is identical to the N command except that a _ search generates no output. Generally, where the N command executes a P, the _ executes a Y. You use the _ search for examination functions and for discarding parts of a file.

You can also use the _ command with a single numeric argument. The command n_ causes a search for the nth occurrence of the search string. When you omit n, TECO assumes n=1. n must always be greater than 0.

"text" may not be longer than 31 ASCII characters.

"text" may include the match control characters:

^Na	No match on character "a"
^Q	Use following command as match character
^S	Match on separator
^X	Exempt position from match

The _ command may use a : modifier.

Example:

*5_VERSION88$$	You can use this command to determine if the string VERSION88 occurs in the input file five times. If it does, the pointer is positioned immediately after the fifth occurrence, and everything in the input file preceding the page on which the fifth occurrence is located is discarded. If it does not, the entire file is discarded.

APPENDIX A

OCTAL & DECIMAL ASCII CHARACTER SET

CHAR	OCT	DEC	CHAR	OCT	DEC	CHAR	OCT	DEC	CHAR	OCT	DEC
NUL	000	000	SP	040	032	@	100	064		140	096
^A	001	001	!	041	033	a	101	065	a	141	097
^B	002	002	"	042	034	B	102	066	b	142	098
^C	003	003	#	043	035	C	103	067	c	144	099
^D	004	004	$	044	036	D	104	068	d	144	100
^E	005	005	%	045	037	E	105	069	e	145	101
^F	006	006	&	046	038	F	106	070	f	146	102
^G	007	007	'	047	039	G	107	071	g	147	103
^H	010	008	(	050	040	H	110	072	h	150	104
TAB	011	009	)	051	041	I	111	073	i	151	105
LF	012	010	*	052	042	J	112	074	j	152	106
VT	013	011	+	053	043	K	113	075	k	153	107
FF	014	012	,	054	044	L	114	076	l	154	108
CR	015	013	-	055	045	M	115	077	m	155	109
^N	016	014	.	056	046	N	116	078	n	156	110
^O	017	015	/	057	047	O	117	079	o	157	111
^P	020	016	0	060	048	P	120	080	p	160	112
^Q	021	017	1	061	049	Q	121	081	q	161	113
^R	022	018	2	062	050	R	122	082	r	162	114
^S	023	019	3	063	051	S	123	083	s	163	115
^T	024	020	4	064	052	T	124	084	t	164	116
^U	025	021	5	065	053	U	125	085	u	165	117
^V	026	022	6	066	054	V	126	086	v	166	118
^W	027	023	7	067	055	W	127	087	w	167	119
^X	030	024	8	070	056	X	130	088	x	170	120
^Y	031	025	9	071	057	Y	131	089	y	171	121
^Z	032	026	:	072	058	Z	132	090	z	172	122
ALT	033	027	;	073	059	[	133	091		173	123
FS	034	028	<	074	060	\	134	092		174	124
GS	035	029	=	075	061	]	135	093		175	125
RS	036	030	>	076	062	^	136	094		176	126
US	037	031	?	077	063	_	137	095	DEL	177	127

APPENDIX B

TECO ERROR MESSAGES

Printout Abbreviation	Printout Message	Meaning
?ARG	IMPROPER ARGUMENTS	Number missing before comma, or two arguments specified to D, or three numeric arguments
?BNI	NOT AN ITERATION	Iteration close (>) without matching open (<)
?CCL	CCL.SV NOT FOUND OR EG ARGUMENT TOO BIG	CCL not found or EG argument too long
?FER	FILE ERROR	FILE ERROR can mean: 1) input file not found on "ER" command 2) cannot enter output file on "EW" or "EB" command 3) device specified for file does not exist 4) "EB" command given on non-file structured device
?FUL	OUTPUT COMMAND WOULD HAVE	Output command would have overflowed output file (panic mode)
?IEC	ILLEGAL CHARACTER X* AFTER E	E followed by an illegal character
?IFC	ILLEGAL CHARACTER X* AFTER F	F followed by an illegal character

TECO ERROR MESSAGES

Printout Abbreviation	Printout Message	Meaning
?IFN	ILLEGAL CHARACTER X*** IN FILE NAME	Illegal file name in "ER", "EW" or command
?ILL	ILLEGAL COMMAND X*	Illegal command
?INP	INPUT ERROR	Parity error on input file
?IQC	ILLEGAL CHARACTER X* AFTER ""	" followed by an illegal command
?IQN	ILLEGAL Q REGISTER NAME X**	Non-alphanumeric Q-register name
?MEM	STORAGE CAPACITY EXCEEDED	Text buffer overflow
?NAC	NEGATIVE ARGUMENT TO,	Negative argument to comma
?NAE	NO ARGUMENT BEFORE =	No numeric argument to the left of an equal sign
?NAP	NEGATIVE OR ZERO ARGUMENT TO P	Negative or zero argument to P
?NAQ	NO ARGUMENT BEFORE QUOTE	No numeric argument to the left of a quote
?NAS	NEGATIVE OR ZERO ARGUMENT TO S	Negative or zero argument with a search
?NAU	NO ARGUMENT BEFORE U	No numeric argument to the left of a U
?NAY	NUMERIC ARGUMENT TO Y	Numeric argument specified with Y command
?NFO	VERSION NUMBER TO FILE FOR OUTPUT	Attempt to output without opening an output file
?NYI	CASE SUPPORT NOT IMPLEMENTED	Case support not implemented (use EO for version)
?NYI	CASE SUPPORT NOT IMPLEMENTED	Case support not implemented (use W for watch)
?OUT	OUTPUT ERROR	Output file too big or output parity error

TECO ERROR MESSAGES

Printout Abbreviation	Printout Message	Meaning
?PDO	INTERNAL PUSHDOWN OVERFLOW	Pushdown overflow (macros and iterations nested too deeply)
?POP	ATTEMPT TO MOVE POINTER OFF PAGE	Attempt to move pointer outside of text buffer
?QMO	Q REGISTER MEMORY OVERFLOW	Q-register storage overflow
?SNI	NOT IN AN ITERATION	Semicolon on command level
?SRH	SEARCH FAILED	Failing search at command level
?STL	SEARCH STRING TOO LONG	Search string too large (greater than 31 characters)
?UTC	UNTERMINATED COMMAND	Incomplete command (PDL not empty at end of command string)
?UTM	UNTERMINATED MACRO	Incomplete command (PDI not empty at end of macro)
?WLO	CANNOT WRITE OUT ERROR MESSAGE OVERLAY	Write locked system device
?XAB		Execution aborted
?YCA	Y COMMAND ABORTED	Y (or .) command aborted because data would be lost

APPENDIX C

TECO COMMAND SUMMARY

Command	Function
ERdev:filnam.ex$	Input file selection.
EWdev:filnam.ex$	Output file selection.
EBdev:filnam.ex$	I/O file selection with backup protection.
Y	Clear buffer and read one page of input file.
A	Read one page of input file and append to current buffer content.
BUFFER POINTER POSITIONS	
B	Before first character.
.	Current pointer position (number of characters to left of pointer).
Z	After last character (number of characters in buffer).
m,n	From m+1th character through and including nth character.
H	Entire buffer; equivalent to B,Z.
ARITHMETIC OPERATORS	
-n	Negation.
m+n	Addition.
m-n	Subtraction.
m*n	Multiplication.
m/n	Divide and truncate.
m&n	Bitwise logical AND.
m#n	Bitwise logical OR.
( )	Perform enclosed operations first.
POINTER POSITIONS	
nJ	Position pointer between nth and n+1th characters.
nC	Move pointer forward across n characters.
nR	Move pointer backward across n characters.
nL	Position pointer at beginning of nth line following current position.
TYPE-OUT COMMANDS	
nT	Type buffer content from pointer position to beginning of nth following line.
m,nT	Type m+1th character through and including nth character.
n=	Type the integer equivalent of expression n.
^Atext^A	Type the enclosed text.
^O	Inhibit typeout.

TECO COMMAND SUMMARY

Command	Function
DELETION COMMANDS	
nD	Delete the n characters following the pointer.
-nD	Delete the n characters preceding the pointer.
nK	Delete the n lines following the pointer.
m,nK	Delete the m+1th character through and including the nth character.
INSERTION COMMANDS	
Itext$	Insert text delimited by I and ALT MODE.
<I>text$	Insert tabulation, then text. <I> is a TAB (control-I) character.
nI	Insert character whose ASCII code is n.
@I/text/	Insert text delimited by arbitrary character shown as a slash.
n\	Insert the ASCII code for integer n.
OUTPUT AND EXIT	
PW	Write current page and append form feed.
P	Write current page, append form feed, clear buffer, and read next page.
m,nP	Write m+1th through nth characters without appending a form feed.
EF	Close the current output file.
^G	Close the current output file and exit to the OS/8 monitor.
^C	Immediate exit to the OS/8 monitor.
^P	Exit to the monitor and do a START 200.
EX	Write the rest of the input file on the output file and exit to the monitor.
EC	Write the remainder of the input file on the output file and close the file.
SEARCH COMMANDS	
nStext$	Begin at the pointer and search for the nth occurrence of the text delimited by the S and the ALT MODE on the current page.
nNtext$	Equivalent to nStext$ except that the search is continued across page boundaries.
n text$	Equivalent to nNtext$ except that no output is generated.
nFNtext1 $text2$	Do nNtext1$ and then replace text1 with text2.
:nStext$	Equivalent to nStext$ except that it returns a value of -1. If the search succeeds, or 0, if the search fails. The colon may be used with N and searches.
n@S/text/$	Equivalent to nStext$ except that the text is delimited by the arbitrary character following the S, instead of ALT MODE.
^X	Accept any character in this position.
^S	Accept any separator in this position. Save last typed command.
^N	Accept any character except the following character in this position.
^Q	Interpret the next character literally, rather than as a command.
FS	Search for a character string in the current buffer and replace with another string.
FN	Search for a character string in a page other than the current buffer and replace with another string.

TECO COMMAND SUMMARY

Command	Function
ITERATION AND FLOW CONTROL	
n< >	Perform enclosed commands n times.
n;	If n is positive, jump out of the current iteration field.
!tag!	Define a position named "tag" at this location.
Otag$	Jump to the position defined by "tag."
n"E	If n=0, execute the following command string.
n"N	If n=0, execute the following command string.
n"L	If n is less than zero, execute the following command string.
n"G	If n is greater than zero, execute the following command string.
n"C	If n is the ASCII code for an alphanumeric character, execute the following commands.
n-m"A	If n is greater than or equal to m, execute the following commands.
n-m"B	If n is less than m, execute the following commands.
Q-REGISTER COMMANDS	
nUq	Store n in Q-register q.
Qq	Equivalent to the value stored in Q-register q.
n%I	Add n to the content of Q-register q and return this value.
^Uqtext$	Insert text into Q-register q.
nXq	Load the n following lines into Q-register q.
m,nXq	Load the m+1th character through the nth character into Q-register q.
Gq	Insert the content of Q-register q into the buffer.
Mq	Execute the content of Q-register q as a command string.
^U	Insert the specified string into the text storage area of the Q-register
NUMERIC VALUES	
nA	ASCII value of nth character following pointer.
^E	Form feed flag.
^F	Console data switches.
^H	Always equals zero.
^^X	Equivalent to the ASCII code for character "X."
^Z	Command and Q-register storage words in use.
^O	Set octal radix.
^D	Set decimal radix.
\	Equivalent to the value of the digit string following the pointer.
^T	Equivalent to the ASCII code for the next character typed.
^V	Equivalent to the number of the version of TECO being run.
PROGRAMMING AIDS	
?	After an error message, identifies erroneous character.
?	Except after an error message, toggles in and out of trace mode.
^G^G	Erases current command string.

INDEX

OS/8
TECO Reference Manual
AA-H608A-TA

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Did you find errors in this manual? If so, specify the error and the page number.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code or Country_____

Please cut along this line.

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

RT/C SOFTWARE PUBLICATIONS ML 5-5/E45
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MASSACHUSETTS 01754

Do Not Tear - Fold Here

Cut Along Dotted Line