

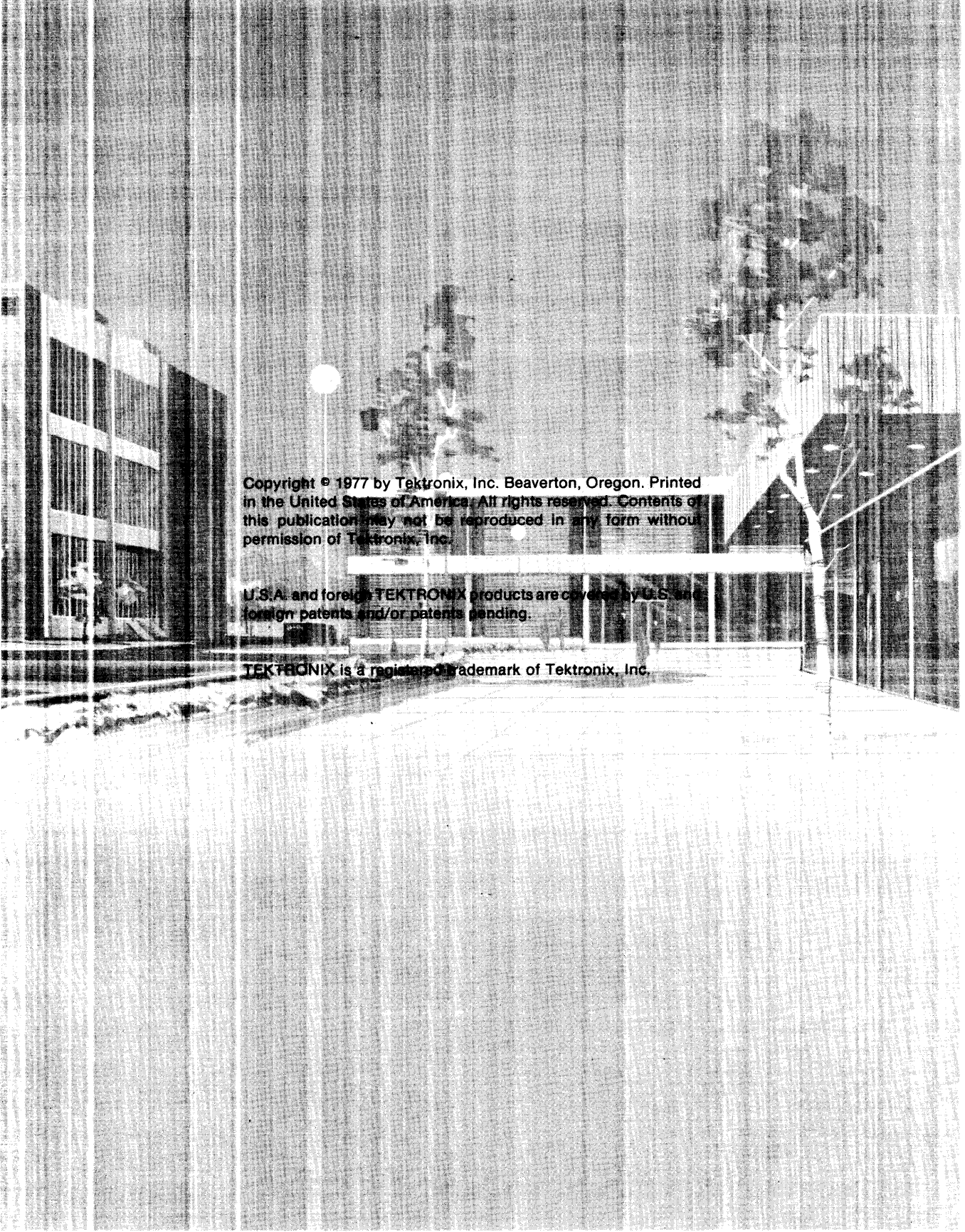
**TEKTRONIX®**

**4051**  
**EDITOR**

4051R06

Tektronix, Inc.  
P.O. Box 500  
Beaverton, Oregon 97077  
070-2170-00

First Printing FEB 1977



Copyright © 1977 by Tektronix, Inc. Beaverton, Oregon. Printed  
in the United States of America. All rights reserved. Contents of  
this publication may not be reproduced in any form without  
permission of Tektronix, Inc.

U.S.A. and foreign TEKTRONIX products are covered by U.S. and  
foreign patents and/or patents pending.

TEKTRONIX is a registered trademark of Tektronix, Inc.

# CONTENTS

SECTION 1	GENERAL DESCRIPTION	Page
	Introduction	1-1
	Specifications	1-3
	Installation Instructions	1-4
	An Overview of the EDITOR Commands	1-6
	Terms Used in the Manual	1-8
	How the Manual is Organized	1-11
SECTION 2	GENERAL INFORMATION	
	Introduction	2-1
	Getting Started	2-6
	Getting the Most Workspace for the EDITOR	2-6
	Calling the EDITOR	2-6
	Returning Control of the System to the BASIC Interpreter	2-6
	The Text Buffer	2-6
	The Size of the Text Buffer	2-7
	Display Format	2-7
	Using an EDITOR Command	2-8
	The Command Keyword	2-8
	Delimiters in the Command	2-9
	Edit Delimiters	2-10
	String Delimiters	2-13
	Edit Line Numbers in the Command	2-15
	How Much Text is Affected by an EDITOR Command	2-23
	Omitting a Parameter in a Command: Default Values	2-27
	Optional Edit Line Numbers	2-27
	Optional Edit Delimiters	2-29
	Other Optional Parameters	2-32
	EDITOR Command Syntax	2-35
SECTION 3	SPECIAL KEYS	
	Introduction	3-1
	The User-Definable Keys	3-1
	RETURN TO BASIC	3-1
	RUBOUT CHARACTER	3-2
	MARGIN OFF	3-2
	MARGIN 1	3-2
	MARGIN 2	3-2
	Resetting the MARGIN Parameter	3-3

# CONTENTS (cont)

## SECTION 3 SPECIAL KEYS (cont)

Keyboard Keys	3-3
The Alphanumeric Keys	3-3
The RETURN Key	3-3
The BREAK Key	3-3
The LINE EDITOR Keys	3-4
The COMPRESS/EXPAND Key	3-4
The REPRINT/CLEAR Key	3-7
The RECALL NEXT LINE/RECALL LINE Key	3-8
The STEP PROGRAM Key (RECALL PREVIOUS LINE)	3-9
Peripheral Control Keys	3-9

## SECTION 4 EDITING COMMANDS

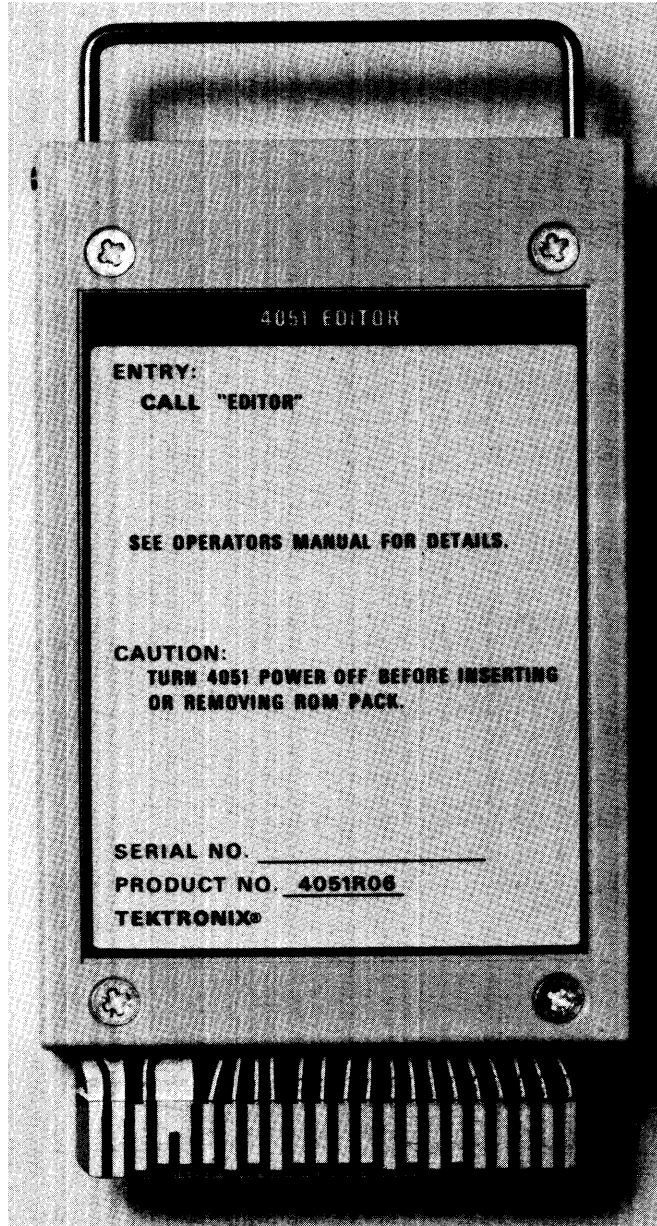
The CARD Command	4-3
The CASE Command	4-9
The COPY Command	4-13
The DELETE Command	4-19
The INSERT Command	4-23
The LIST Command	4-29
The MOVE Command	4-35
The NLSEARCH Command	4-39
The NLSEARCH and Delete Line Command	4-41
The NLSEARCH and Replace String Command	4-47
The SEARCH Command	4-55
The SEARCH and List Line Command	4-59
The SEARCH and Edit Line Command	4-65
The SEARCH and Delete Line Command	4-73
The SEARCH and Replace String Command	4-79
The SORT Command	4-87
The REVSORT Command	4-95

## SECTION 5 ENVIRONMENTAL COMMANDS

The LASTLINE Command	5-3
The LOWERCASE Command	5-9
The RENUMBER Command	5-15
The UPPERCASE Command	5-21
The #= Command	5-27
The ~= Command	5-33
The _= Command	5-41
The ]= Command	5-47

# CONTENTS (cont)

SECTION 6	Input/Output Commands	
	Introduction	6-2
	I/O Addresses	6-2
	Default Values	6-2
	The APPEND Command	6-5
	The FIND Command	6-11
	The INPUT Command	6-17
	The MARK Command	6-21
	The OLD Command	6-25
	The PRINT Command	6-29
	Special PRINT Commands	6-33
	The SAVE Command	6-39
	The SKIP Command	6-43
	The SWN Command	6-49
	The WRITE Command	6-55
SECTION 7	APPENDIX A	
	Error Messages	
SECTION 8	APPENDIX B	
	Tables	
SECTION 8	APPENDIX C	
	Index	



2170-1

Fig. 1-1. 4051 EDITOR.

## Section 1

# GENERAL DESCRIPTION

## INTRODUCTION

The Tektronix 4051R06 EDITOR is a Read-Only Memory device designed to be used with the Tektronix 4050-Series Graphic Systems. The EDITOR ROM Pack, shown in Fig. 1-1, contains firmware routines that allow ASCII magnetic tape files to be altered or edited.

## How To Call the EDITOR

After the ROM Pack is plugged into the Graphic System backpack or into a ROM Expander Unit, the EDITOR routines are made available by entering the statement `CALL "EDITOR"` from the keyboard. Executing the `CALL` statement gives control of the Graphic System to the EDITOR, and until the `RETURN TO BASIC` overlay key is pressed, BASIC commands are not available.

## EDITOR Commands and Special Keys

When the EDITOR is in control of the Graphic System, 29 commands may be used to bring lines of text into memory, alter them, and send the edited lines to an internal or external storage device. The commands include text editing commands, Input/Output commands, and environmental commands. The keyboard LINE EDITOR keys may be used while the EDITOR is in control, and six of the user-definable keys have special meanings.

## What to Use the EDITOR For

The EDITOR is used to edit already existing files, or to create new files. The text acted upon by the EDITOR may be any set of ASCII characters; that is, programs, data, or "free" text such as letters or textbooks. The EDITOR is not restricted to editing BASIC programs; it may be used to write programs in FORTRAN, COBOL, ALGOL, or any other programming language that uses ASCII characters. The EDITOR may be used to edit programs written in other versions of BASIC, to make the programs compatible with 4051 BASIC.

## The EDITOR and the Data Communications Interface

The TEKTRONIX 4051 Data Communications Interface is used to send the contents of internal magnetic tape files to a host computer, or to pull files from a host and place them in an internal tape file. This means that programs created or updated under EDITOR control may be stored on the internal magnetic tape unit, then sent to a host computer using the Data Communications Interface. Or, programs stored on the host may be pulled back into the internal tape unit using the Interface, then edited off-line using the EDITOR.

## **General Description**

Writing, updating, and documenting programs are most economically done off-line, under EDITOR control. Using the EDITOR saves costly connect time during these processes, and frees the host computer to work on other jobs.

## **Summary**

In summary, the EDITOR is a line-oriented and string-oriented text editor consisting of 29 commands. The EDITOR allows the 4050-Series Graphic Systems to be used off-line for creating, editing, and storing free text, as well as for writing, updating, documenting, and saving programs in any programming language based on the ASCII character set.



## SPECIFICATIONS

### POWER REQUIREMENTS

The 4051R06 EDITOR draws all necessary power from the 4051 power supplies. Connections to the power supplies are made through the backpack on the rear panel of the main chassis. The EDITOR ROM Pack must be inserted into a slot in the backpack, or into a ROM Expander Unit, before power is applied to the system.

Voltage Supplies	Current Limit
+5 Vdc	100 mA
+12 Vdc	40 mA
-12 Vdc	4 mA

### MEMORY REQUIREMENTS

When inserted into a backplot slot, the EDITOR immediately reserves 592 bytes of RAM memory for its own use. This space is used to store status information, and is not the same as the workspace reserved for storing text.

The 4051 Option 1 (Data Communications Interface) also uses 592 bytes as soon as it is plugged into the system. However, if the EDITOR and a Data Communications Interface are both plugged into backpack slots, they share the same 592 bytes of storage.

### ALTITUDE

Non-operating: 50,000 feet maximum  
 Operating: 15,000 feet maximum

### TEMPERATURE

Non-Operating: -40°C to +65°C.  
 Operating: +10°C to +40°C.

### HUMIDITY

95% non-condensing (storage)  
 80% non-condensing (operating)

## General Description

### VIBRATION (NON-OPERATING)

0.015" DA-10-50-10

### SHOCK (NON-OPERATING)

1/2 Sine 11 ms duration, 30 G's

### PHYSICAL DIMENSIONS (INCLUDING EDGE BOARD CONNECTOR)

Length: 4.662 inches (11.84 centimeters)

Width: 2.620 inches (6.65 centimeters)

Depth: 0.875 inches (2.22 centimeters)

### WEIGHT

8 oz. (227 grams)

### STANDARD ACCESSORIES

1—Operators Manual (Tektronix Part Number 070-2170-00)

5—User-definable key overlays

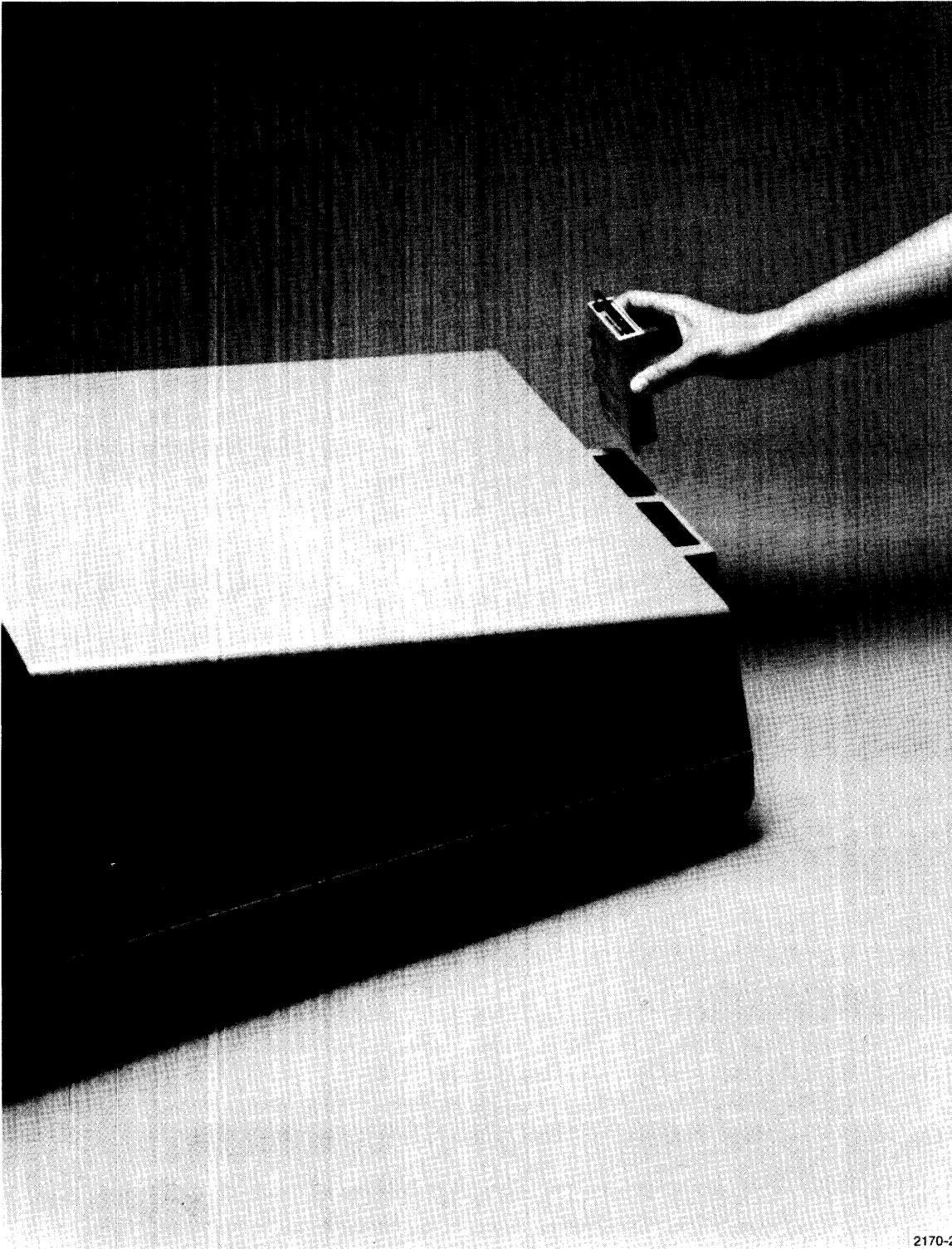
### INSTALLATION INSTRUCTIONS

1. Set the 4051 power switch to the OFF position.



*Inserting any device into the backpack when the 4051 power is ON may cause memory to be erased. Make sure that important information in the RAM is stored on magnetic tape before turning the power OFF and proceeding.*

2. With the power removed from the system, insert the EDITOR ROM Pack into a slot as shown in Fig. 1-2. Press down, and at the same time gently rock the plastic housing from side to side until the ROM Pack edgeboard connector is firmly seated in the receptacle connector.
3. Set the 4051 power switch to the ON position. After a few seconds of warm-up, the system is ready for use.



2170-2

**Fig. 1-2. Insert the EDITOR ROM pack into a backpack slot.**

## **AN OVERVIEW OF THE EDITOR COMMANDS**

### **EDITING COMMANDS**

The EDITOR provides 11 editing commands for use in changing text. The commands are CARD, COPY, CASE, DELETE, INSERT, LIST, MOVE, NLSEARCH, SEARCH, SORT, and REVSORT.

The commands are used to format text into lines of a specified length (CARD), to copy portions of text at a specified location (COPY), and to change the case of the characters in the text (CASE). Lines may be deleted (DELETE), new lines may be created (INSERT), and lines may be listed on the display or sent to an external device (LIST). Lines may be moved to a new location (MOVE) or rearranged according to the ASCII value of characters found in specified positions within a line (SORT and REVSORT).

NLSEARCH and SEARCH together have six syntax forms which are variations of searching for a specified string. NLSEARCH is used to delete lines found to contain a certain string, or to search for a string and replace it with another string. SEARCH performs these same two functions, but at the same time lists all lines containing the specified string.

SEARCH has two more variations. The first one seeks out and lists on a device all lines found to contain a certain string. The second one finds a line containing the string, reprints and recalls the line to the line buffer for editing, then continues the search.

### **INPUT/OUTPUT COMMANDS**

The EDITOR has 10 Input/Output commands used for transmitting text to and from storage devices, for positioning the READ/WRITE head, and for marking new files. The commands are APPEND, FIND, INPUT, MARK, OLD, PRINT, SAVE, SWN (Save With Number), SKIP, and WRITE.

Although many of the commands have the same keyword as a BASIC Input/Output command, remember that these are EDITOR commands, not BASIC commands. EDITOR commands are not necessarily identical to their BASIC equivalents; there are some differences in the syntax forms and in how the commands work. For example, the command WRITE in EDITOR may have two line numbers following the keyword, but WRITE in BASIC requires a data item to be specified in the command. Also, WRITE in BASIC is used to send data to a peripheral device in machine dependent binary code, but WRITE in EDITOR is used to store text on a device in ASCII code.

The differences between EDITOR Input/Output commands and the equivalent BASIC commands are included in the explanations of the commands in Section 6.

## ENVIRONMENTAL COMMANDS

There are eight EDITOR commands that are called "environmental" because they do not transmit or edit text, but alter the working environment or change status bytes within the system. The commands are LASTLINE, LOWERCASE, RENUMBER, UPPERCASE, ]=, #=, ~=, and \_=.

The LASTLINE command examines the text buffer and returns information about the size of its current contents. LOWERCASE and UPPERCASE set flags that affect the result of executing the commands CASE, SORT, REVSORT, NLSEARCH, and SEARCH. The RENUMBER command assigns new edit line numbers to lines of text.

The next four environmental commands allow specified ASCII characters to take on special meanings. The ]= command gives the specified ASCII character the meaning "END-OF-RECORD." The #= command defines a character that can stand for any digit, and ~= assigns a "wildcard" character. The command \_= lets the specified character be used as a prefix meaning "all but the character following this one."

## Special PRINT Commands

In addition to the environmental commands described above, three special PRINT commands may be executed while under EDITOR control to change the processor status or internal magnetic tape status. The first special PRINT command prepares the microprocessor for requests for an alternate Input/Output format. This command is the same as its BASIC equivalent and is reviewed briefly in Section 5.

The second special PRINT command instructs the microprocessor to change the end-of-record character from a CR to a CR and LF when inputting text, and when listing text on an external device. The command differs from its BASIC counterpart in that it only affects the end-of-record character during the two Input operations APPEND and OLD, and the two Output operations LIST and SEARCH.

The third special PRINT command changes the internal magnetic tape status by specifying whether physical records are to be 128 or 256 bytes long, whether or not the checksum error checking technique should be used, and whether or not to use file header format. The command is the same as its BASIC equivalent.

## SPECIAL KEYS

### Alphanumeric Keys, LINE EDITOR Keys, and Peripheral Control Keys

Most of the keys on the Graphic System keyboard operate under EDITOR control as they do in BASIC. All of the alphanumeric keys still function after the EDITOR is called, performing the same or similar functions as in BASIC.

## General Description

The LINE EDITOR keys also work under EDITOR control. The functions COMPRESS, EXPAND, RUBOUT→, RUBOUT←, BACKSPACE, SPACE, REPRINT, CLEAR, RECALL LINE, and RECALL NEXT LINE, may be performed under EDITOR control. Several of these keys function in a slightly different manner or serve different purposes under EDITOR than in BASIC. The differences are explained in Section 3.

The AUTO NUMBER key is not used by the EDITOR, and the STEP PROGRAM key is assigned the special meaning "RECALL PREVIOUS LINE" while the EDITOR is in control. Peripheral control keys REWIND and MAKE COPY may be used under EDITOR, but not AUTO LOAD.

## USER-DEFINABLE KEYS

Five user-definable keys have predefined uses while the EDITOR is in control. The specially assigned keys are shown on the user-definable key overlay. MARGIN OFF, MARGIN 1, and MARGIN 2 control how many columns of text appear on the display and what happens when a PAGE FULL condition occurs. The RUBOUT CHARACTER user-definable key provides the symbol ↓, and the RETURN TO BASIC key is pressed to return control of the system to the BASIC Interpreter.

## TERMS USED IN THE MANUAL

The terms defined below are used throughout the manual. It may be helpful to know what these words mean before reading Section 2.

### Text

The word *text* refers to any set of ASCII characters. Text may consist of one line of characters, or of many lines separated by the end-of-record character. The text acted upon by the EDITOR may be programs, data, or "free text."

### Text Buffer

The *text buffer* is a portion of memory reserved for use by the EDITOR. The text buffer is the workspace where editing occurs. Normally, text is brought into the text buffer from a peripheral device using the command OLD or APPEND. The text is held in the text buffer and modified using the editing commands, then sent from the text buffer to a storage device using Output commands SAVE, SWN, or WRITE.

## Line Buffer

The EDITOR *line buffer* is a portion of memory used to store up to 396 characters. The EDITOR line buffer is more than five times longer than BASIC's line buffer. Acting as an intermediary between the keyboard and the text buffer, the EDITOR line buffer holds characters entered from the keyboard, until BREAK or the RETURN key is pressed. When BREAK or RETURN is pressed, all characters currently in the line buffer are sent to the text buffer.

In BASIC, lines cannot be longer than 72 characters. However, under EDITOR control you may create lines up to 396 characters long, simply by continuing to enter characters from the keyboard. Lines having more than 66 characters appear in "wrap-around" form on the display:

```
: *****
*****
*****
*****
*****
*****
*****
```

The line shown above begins after the colon (:), and consists of 396 asterisks. After the 396 asterisks have been entered, the cursor stops moving, and any additional characters entered from the keyboard are lost.

Once a line has been sent to the text buffer, it may be recalled to the line buffer by pressing the RECALL NEXT LINE/RECALL LINE or STEP PROGRAM key. A line that has been recalled to the line buffer in this manner may be edited using the LINE EDITOR keys.

Although the line buffer can only hold 396 characters, you may create lines longer than 396 characters under EDITOR control. For instance, you may use the SEARCH and Replace String command to locate and delete an end-of-record character between two lines that have 300 characters each. Deleting the end-of-record character in this way concatenates the two lines into one line of 600 characters. The long line may be held in the text buffer, listed, and saved under EDITOR. Once saved, the line can be brought back into the text buffer using the Input commands OLD and APPEND.

However, only part of the line can be brought back into the line buffer, because the line buffer holds a maximum of 396 characters. Pressing RECALL LINE, RECALL NEXT LINE, or STEP PROGRAM brings the first 396 characters of the line into the line buffer for editing. The remaining 204 characters are not accessible for editing by the LINE EDITOR keys.<sup>1</sup>

<sup>1</sup> If you must access these characters, insert an end-of-record character again to split the line into two lines that are both short enough to be recalled to the line buffer for editing. Later, the end-of-record character may be removed once more to obtain the longer line. Thus, although it is more convenient to work with lines having 396 or fewer characters, you may actually use the EDITOR to create a line of any length (provided the line does not exceed the current length of the text buffer).

## General Description

### Line of Text

A line is a string of ASCII characters that fall between two end-of-record characters. In this manual, *line of text* refers to a line found in the text buffer.

### Logical Record

A *logical record* is a string of ASCII characters that lie between two end-of-record characters on a storage device. A string of characters between two end-of-record characters is called a *line of text* while found in the text buffer, but becomes a *logical record* when stored on magnetic tape.

### Edit Line Number

When the RENUMBER command is executed under EDITOR control, each line of text is assigned a special line number called an *edit line number*. Edit line numbers appear to the left of the lines of text, and are separated from the text by a colon (:).

Edit line numbers are assigned by the EDITOR. They are not considered to be part of the text, and cannot be altered except by the RENUMBER command. Edit line numbers should not be confused with program line numbers, which are part of the text and may be edited using EDITOR commands.

### String

The word *string* is used in command syntax forms to mean any ASCII character string. Quotation marks are shown in the syntax form if delimiters are required around a string. For instance, the EDITOR command PRINT does not require delimiters around the parameter string, so no quotation marks appear in the syntax form. However, SEARCH and NLSEARCH require parameter strings to be enclosed in string delimiters, as indicated by the quotation marks in the syntax forms.

### String Delimiter

A *string delimiter* is a character used to enclose a parameter string in an EDITOR command. Although the command syntax forms show quotation marks around strings, any ASCII character except SPACE and CR may serve as a string delimiter.

### End of Record Character

An *end-of-record* character marks the beginning of a logical record or line of text. An end-of-record character can be entered from the keyboard by pressing RETURN while inserting new text, or by entering a specially assigned "END-OF-RECORD" character. (Refer to the ]= command for an explanation of how to use the special "END-OF-RECORD" character.)



End-of-record characters are not actually stored in the text buffer. Pressing RETURN while inserting new text, or entering the special "END-OF- RECORD" character, sets a "flag" in the appropriate location in the text. When outputting text to a peripheral device, the EDITOR disables the flag and inserts a CR character (or a previously specified alternate end-of-record character). Conversely, when bringing text into the text buffer from a peripheral device, the EDITOR removes the CR character (or the alternate end-of-record character) and sets the flag.

### Edit Delimiter

An *edit delimiter* is a character used to separate strings, edit line numbers, or other numeric constants in an EDITOR command. Although the command syntax usually shows a comma (,) as the edit delimiter, any of the following characters may serve as an edit delimiter:

,  
.  
=  
;  
:  
space

Edit delimiters and string delimiters are explained in greater detail in Section 2.

## HOW THE MANUAL IS ORGANIZED

The manual is organized into six sections and two appendices. Section 2 provides general information about the EDITOR and about using EDITOR commands. Section 3 describes operation of the keyboard and user-definable keys under EDITOR control.

Sections 4, 5, and 6 discuss the EDITOR commands in detail. Examples are given to illustrate uses for the commands, and if an EDITOR command resembles a BASIC command, similarities and differences are noted.

Section 4 describes editing commands. Two of the commands, LIST and SEARCH, allow an I/O address to be specified as a parameter. Although technically this makes them Input/Output commands, LIST and SEARCH are included in Section 4 because of the important role they play during text editing.

Section 5 discusses environmental commands, and special PRINT commands used to change processor or magnetic tape status. Section 6 explains the Input/Output commands. Appendix A describes error messages, and Appendix B contains tables, including a command summary, tables of default parameter values, and an ASCII code chart.

## Section 2

# GENERAL INFORMATION

## INTRODUCTION

Section 2 is designed to help you get started using the EDITOR. General principles are explained here in order to simplify the explanations of individual EDITOR commands in later sections. After reading this section, you will be ready to begin using the EDITOR, and will be familiar with most of the rules for using EDITOR commands.

This section tells you how to call the EDITOR, how to return to BASIC control after using the EDITOR, and how to use the commands; including how to specify command keywords, delimiters, and edit line numbers. You will find out how much text is affected by an EDITOR command, and what happens when parameters are omitted in the command.

### NOTE

*Since some of the details in this section will make more sense after you have worked with the commands, it is a good idea to read this material quickly, and return to it if questions or problems arise when trying the EDITOR commands.*

The section begins with a COMMAND SUMMARY chart, showing each command keyword, an example of how to use the command, and a short description of what happens when the command is executed. You may want to glance at this chart before reading about the commands, then use it later as a reference while using the EDITOR. A table of the EDITOR commands and their default parameter values is also included in the section, so that you may quickly check for a default value without looking up the individual command.

4051R06 EDITOR  
COMMAND SUMMARY

Command	Example	Action Taken
<b>EDITING COMMANDS</b>		
<b>CARD</b>	<b>CA 80,42</b>	Formats the text buffer into lines that are 80 characters long, splitting lines of text having more than 80 characters into 2 or more lines, and using the character * (decimal equivalent 42) to fill out any lines of text that have less than 80 characters.
<b>CASE</b>	<b>CAS 1,100</b>	In lines 1 through 100 of the text buffer, lower case characters a-z are replaced by their uppercase equivalents A-Z, if the UPPERCASE flag has been set. If the LOWERCASE flag has been set, uppercase characters A-Z are replaced by their lowercase equivalents a-z.
<b>COPY</b>	<b>C 1,3,10</b>	Duplicates lines 1 through 3 of the text buffer, placing the copied text immediately before line 10.*
<b>DELETE</b>	<b>D 1,10</b>	Deletes lines 1 through 10 from the text buffer.
<b>INSERT</b>	<b>I 100</b>	Responds with five spaces and a colon (:) to prompt the entry of new text from the keyboard. Lines of text entered after the colon are placed in the text buffer immediately before the line 100.
<b>LIST</b>	<b>L @29:200,300</b>	Lists lines 200 through 300 of the text buffer on peripheral device 29 on the General Purpose Interface Bus.
<b>MOVE</b>	<b>M 1,3,10</b>	Moves lines 1 through 3 of the text buffer, placing them immediately before line 10.**
<b>NLSEARCH and Delete Line</b>	<b>NL 0,1000 "REM"*</b>	Searches lines 0 through 1000 of the text buffer, and deletes lines found to contain the string REM .
<b>NLSEARCH and Replace String</b>	<b>NL 1,100 "ON ERROR","ON SIZE"</b>	Searches lines 1 through 100 of the text buffer, and replaces occurrences of the string ON ERROR with the string ON SIZE .
<b>SEARCH and List Line</b>	<b>S @29:0,2000 "638-"</b>	Searches lines 0 through 2000 of the text buffer, and lists on device 29 all lines found to contain the string 638-
<b>SEARCH and Edit Line</b>	<b>S 1,100 "Mr. ",</b>	Searches lines 1 through 100 of the text buffer for the string Mr. . One by one, lines found to contain the string are recalled to the line buffer, and wait to be edited.
<b>SEARCH and Delete Line</b>	<b>S @29:0,3000 "pd."*</b>	Searches lines 0 through 3000 of the text buffer for the string pd. . Lines found to contain the string are listed on device 29, and deleted from the text buffer.
<b>SEARCH and Replace String</b>	<b>S @29:1,1000 "PRI","PRI @3:"</b>	Searches lines 1 through 1000 of the text buffer and replaces occurrences of the string PRI with the string PRI @3: . Changed lines are listed on device 29.

\*See end of table

\*\*See end of table

**4051R06 EDITOR  
COMMAND SUMMARY (cont.)**

<b>Command</b>	<b>Example</b>	<b>Action Taken</b>
<b>SORT</b>	<b>SO 1,10:1,2,3</b>	Rearranges lines 1 through 10 in the text buffer, sorting "alphabetically" according to the ASCII values of characters found in the first three character positions within each line.
<b>REVSORT</b>	<b>REV 1,10:1,2,3</b>	Rearranges lines 1 through 10 in the text buffer, sorting "alphabetically in reverse" according to the ASCII values of characters found in the first three character positions within each line.
<b>ENVIRONMENTAL COMMANDS</b>		
<b>LASTLINE</b>	<b>LA</b>	Returns the following information about the current status of the text buffer: <ul style="list-style-type: none"> <li>— The last edit line number in the text buffer (including offset if the line has no number).</li> <li>— The total number of lines in the text buffer.</li> <li>— The number of bytes needed to save the current contents of the buffer on a storage device.</li> <li>— The number of unused bytes remaining in the text buffer.</li> </ul>
<b>LOWERCASE</b>	<b>LO</b>	Enables the EDITOR to distinguish between lowercase characters a-z and their uppercase equivalents A-Z during searching and sorting operations. Prepares the EDITOR to change uppercase characters into lowercase characters if the CASE command is executed.
<b>UPPERCASE</b>	<b>U</b>	Causes the EDITOR to treat lowercase characters a-z in the text buffer as uppercase characters A-Z during searching and sorting operations. Prepares the EDITOR to change lowercase characters into uppercase characters if the CASE command is executed.
<b>RENUMBER</b>	<b>R 100,10,3</b>	Renumsbers all lines in the text buffer starting with the line currently numbered 3. The new edit line numbers start at 100 and increase with an increment of 10.
<b>] =</b>	<b>] = /</b>	Makes the character / stand for "END-OF-RECORD." The character / may be used during line editing to insert end-of-record characters into the text, and may appear in target and replacement strings during searching operations.
<b># =</b>	<b># = *</b>	Makes the character * stand for "any digit 0 to 9," so that any of the digits 0 through 9 satisfy a search for the character*.
<b>~ =</b>	<b>~ = ?</b>	Makes the character ? stand for "any character." Any ASCII character satisfies a search for the character ? . When used in a replacement string, ? indicates that the ASCII character found in that position should remain unchanged by the replacement procedure.
<b>_ =</b>	<b>_ = +</b>	Allows the character + to be used as a prefix meaning "all but, " so that all characters satisfy a search except the one immediately following +. For instance, the command S "+A" searches the text buffer for all ASCII characters except A.

**4051R06 EDITOR  
COMMAND SUMMARY (cont.)**

Command	Example	Action Taken
<b>Special PRINT Commands</b>		
<b>Processor Status</b>	<b>PRI @37,0:10,4,13</b>	Prepares the microprocessor for requests for an alternate Input/Output format. When a % sign is used in an I/O command instead of @, the special format is as follows: end-of-record = LF (ASCII 10) end-of-file = EOT (ASCII 4) character to ignore = CR (ASCII 13)
	<b>PRI @37,26:1</b>	Tells the microprocessor to send a line feed (LF) character after each CR in the text when listing text on external devices (LIST and SEARCH commands).
	<b>PRI @37,26:0</b>	Tells the microprocessor to send a CR (instead of CR and LF) after each line when listing text on external devices.
<b>Magnetic Tape Status</b>	<b>PRI @33,0:1,1,1</b>	Sends the following status information to the microprocessor: — Format the tape into 128-byte physical records. — Do not use the checksum error checking technique. — Do not use file header format.
	<b>PRI @33,0:0,0,0</b>	Sends the following status information to the microprocessor: — Format the tape into 256-byte physical records. — Use the checksum error checking technique. — Use file header format.
<b>I/O COMMANDS</b>		
<b>APPEND</b>	<b>A @29:50</b>	Adds logical records to the text buffer from the file currently open on device 29. The incoming text is added immediately before line 50 of the text buffer.
<b>FIND</b>	<b>F @29:4</b>	Positions the READ/WRITE heads to the beginning of file 4 on device 29, and opens the file for access by Input/Output operations.
<b>INPUT</b>	<b>INP @29:</b>	Displays one logical record from the file currently open on device 29, and positions the READ/WRITE heads over the next record on file.
<b>MARK</b>	<b>MA 2,5120</b>	Reserves space on the internal magnetic tape for 2 new files, starting at the current position of the tape heads. 5120 bytes of storage are reserved for each file.
<b>OLD</b>	<b>0 @29:</b>	Clears the text buffer, then brings logical records into the text buffer from the current file on device 29.
<b>PRINT</b>	<b>P @29:List of File 3</b>	Prints List of File 3 on device 29.
<b>SAVE</b>	<b>SA @29:100,500</b>	Stores an unnumbered copy of text buffer lines 100 through 500 on the file currently open on device 29. Once the lines are saved, the file is closed to access by Input/Output operations.
<b>SWN</b>	<b>SWN @29:100,500</b>	Stores a numbered copy of text buffer lines 100 through 500 on the file currently open on device 29. Once the lines and edit line numbers are saved, the file is closed to access by Input/Output operations.

**4051R06 EDITOR  
COMMAND SUMMARY (cont.)**

Command	Example	Action Taken
<b>SKIP</b>	<b>SK @29:10</b>	Moves the READ/WRITE heads 10 logical records forward on the current file on device 29. The portion of the file beyond the new position of the READ/WRITE heads remains open for access by Input/Output operations.
<b>WRITE</b>	<b>W @29:100,500</b>	Stores an unnumbered copy of text buffer lines 100 through 500 on the file currently open on device 29. Once the lines are stored, the file remains open for access by Output operations.

\*If a non-existent line number is specified in an EDITOR command, the EDITOR automatically uses the line having the next largest edit line number.

\*\*A portion of the text buffer may include lines which do not have edit line numbers. EDITOR commands act upon all lines of the text buffer (numbered or not) which fall between the specified starting and ending line numbers.

## GETTING STARTED

### Getting the Most Workspace for the EDITOR

Unless you have important variables and programs stored in memory, you should donate more workspace to the EDITOR by performing the following two functions before calling the EDITOR:

```
DEL ALL
```

```
MEM
```

Deleting BASIC programs and compressing memory immediately before calling the EDITOR, allows more space for text to be stored while it is being altered by EDITOR commands. A maximum amount of space is automatically reserved for text if the EDITOR is called immediately after power-up.

### Calling the EDITOR

The next step in getting started is to call the EDITOR by entering the following statement from the keyboard:

```
CALL "EDITOR"
```

When this statement is executed, the Graphic System is placed under EDITOR control. You may now use the EDITOR commands. Regular BASIC commands are no longer available to you, however, and variables or programs stored in memory prior to the CALL cannot be accessed while the system is under EDITOR control.

### Returning Control of the System to the BASIC Interpreter

When you are finished using the EDITOR commands, press the RETURN TO BASIC overlay key to return control of the system to the BASIC Interpreter. You must now use BASIC commands. All of the variables and programs stored in memory prior to calling the EDITOR, are available to you again.

### The Text Buffer

The EDITOR's workspace is the text buffer, an area in memory reserved for holding lines of text. Newly created lines of text are held in the text buffer, and lines previously stored on magnetic tape are always brought into the text buffer before being changed by editing commands.

## The Size of the Text Buffer

The size of the text buffer depends on the memory option you chose for your Graphic System, and on the number and size of BASIC variables and programs left in memory before calling the EDITOR. To find the maximum size of your text buffer, call the EDITOR immediately after turning the system power on. Then execute the LASTLINE command, by entering the following statement from the keyboard:

```
LASTLINE
```

The EDITOR returns three lines of information. The last line tells you how many bytes of space are "free" (unused) in the text buffer. Since the text buffer is currently empty, and no BASIC variables or programs are stored in memory, this number represents the maximum number of bytes the text buffer may contain.

For example, here is what happens for a Graphic System having 32K bytes of memory:

```
CALL "EDITOR"
LASTLINE
0 Lines
0 Characters
30210 Free
```

For this particular Graphic System, the maximum amount of space reserved for the text buffer is 30210 bytes.

## Display Format

Commands entered from the keyboard appear on the display starting in the first character position. To help you distinguish commands from text, the EDITOR begins displaying text in the seventh character position, and precedes each line of text with a colon (:). Edit line numbers up to four digits long may appear before the colon in each line of text. Here is a sample of the display format:



```
INSERT
      :THESE LINES
      :WERE CREATED
      :USING THE
      :INSERT COMMAND.

RENUMBER 3000,500,0

LIST
3000:THESE LINES
3500:WERE CREATED
4000:USING THE
4500:INSERT COMMAND.
```

In this example, EDITOR command keywords INSERT, RENUMBER, and LIST appear on the display starting in the left-most character position, but lines of text are indented and preceded by a colon, or a line number and a colon.

When the command INSERT is entered, the EDITOR responds by spacing five character positions to the right, and typing a colon. The colon prompts you to enter a line of text. When you press RETURN, the EDITOR moves five spaces on the next line, marks another colon, and waits for you to create another line. When you are through inserting new lines of text, press the BREAK key.

The RENUMBER command assigns edit line numbers to lines of text. Here, each line receives a number, starting with 3000 and incrementing by 500. Then the EDITOR command LIST causes the line numbers and lines of text to appear on the display as shown above.

The commands INSERT, RENUMBER, and LIST are explained in detail in Sections 4 and 5.

## Using an EDITOR Command

### THE COMMAND KEYWORD

#### Abbreviating a Keyword

When entering an EDITOR command, you may specify the complete command name, or use an abbreviation. For instance, the command COPY may be entered as COPY, COP, CO, or C.

Most command keywords can be shortened to one letter. However, if two or more commands begin with the same letter, more of the keyword may be required for the EDITOR to know which command to execute. Examples of this are MOVE and MARK, and the commands COPY, CARD, and CASE. MOVE may be abbreviated to M, but MARK requires at least MA to be specified; COPY may be entered as C, but CARD and CASE can only be shortened to CA and CAS, respectively.

The minimum number of letters required when entering each command keyword is shown in the syntax forms in Sections 4, 5, and 6.

Abbreviating command keywords speeds the process of entering EDITOR commands. It is much more convenient to enter A, O, and L than APPEND, OLD, and LIST. However, you should be aware that when you return to BASIC, some of these abbreviations are not valid. For instance, the BASIC commands PRINT, LIST and OLD cannot be abbreviated to a single letter as the corresponding EDITOR commands can. Attempting to enter keywords P, L, or O while the system is under BASIC control, causes an UNDEFINED VARIABLE error message to appear on the display. To avoid forming habits that are hard to break, you might want to use the same abbreviations for EDITOR commands as for BASIC commands of the same name.

### Uppercase and Lowercase Letters in the Keyword

An EDITOR command may be entered from the keyboard in uppercase, lowercase, or a combination of both. For example, the following command is valid:

```
Search 1,100 "Alpha"
```

If a syntax error is made when entering a command keyword in lowercase, the EDITOR returns a syntax error message. The third line of the message reprints the incorrect command, capitalizing all letters in the keyword up to and including the first incorrect character. For example:

```
serch 1,100 "Alpha"
EDITOR ERROR
Syntax - error number 138
SERch 1,100 "Alpha"
```

As in BASIC, a command containing a syntax error is automatically recalled to the line buffer, and may be corrected by typing over the incorrect characters, or by using the LINE EDITOR keys.

### DELIMITERS IN THE COMMAND

Two types of delimiters may appear in an EDITOR command. The first type is called an edit delimiter, and is used to separate strings or numeric constants such as edit line numbers.

## General Information

### Edit Delimiters

The following characters may be used as edit delimiters:

,  
.  
=  
;  
:  
space

Although the command syntax forms show a comma (,) as the edit delimiter, any of the above characters may be used to separate the command parameters. For instance, the following commands are equivalent:

**MOVE 1,100,250**

**MOVE 1,100:250**

**MOVE 1 100 250**

If a comma appears in a command syntax form, it can be replaced by any other edit delimiter. In some cases, using an edit delimiter other than a comma helps you remember what the numbers mean, and what the command does. For example, you may enter the following command, using commas to separate all of the parameters:

**SEARCH 1,100,"THIS","THAT"**

Or, you may replace some of the commas with other edit delimiters, and enter this command:

**SEARCH 1,100:"THIS"="THAT"**

This second form seems to express the meaning of the command, that the EDITOR is to search lines 1 through 100 of the text buffer, then change occurrences of THIS into THAT .

Some similar examples are given in Sections 4, 5, and 6. Keep in mind that you can substitute other edit delimiters for the ones shown in examples, and that commas appearing in a syntax form may be replaced by other edit delimiters.

**Using a Space as an Edit Delimiter**

A blank space in an EDITOR command is taken to be an edit delimiter, except in these instances:

—When the space occurs immediately before or after a keyword. For example:

**LIST 1,1000**

—When the space immediately follows another edit delimiter, such as a comma or another blank space.

Except in the two cases listed above, blank spaces in a command are considered to be edit delimiters. This means that you must be careful when putting blank spaces in a command, or the EDITOR may misinterpret your command.

For instance, the following two commands are **not** equivalent:

**LIST 0,0+100**

**LIST 0,0+ 100**

The first command tells the EDITOR to list lines 0 through 100 on the display. However, the extra blank spaces between + and 100 cause the EDITOR to misinterpret the second command.

In some cases, adding one more blank space can completely change the meaning of a command. For instance, the commands shown below are not the same:

**SEARCH "97062"**

**SEARCH "97062" ␣**

The character ␣ appearing at the end of the second command indicates that a space was entered by pressing the SPACE bar before the RETURN key. The space is an edit delimiter, and completely changes the meaning of the command. Without the extra space, the command asks the EDITOR for a list of all lines found to contain the string 97062. But when the space (␣) is added onto the end of the command, the EDITOR reprints the lines one by one, recalling each one to the line buffer and stopping each time to allow the line to be edited.

## General Information

When entering the command, do not press the SPACE bar before RETURN unless you want the SEARCH command to perform a different function. This shows how important the addition of one edit delimiter can be.

### When a Space Follows an Edit Delimiter

The EDITOR ignores spaces that immediately follow an edit delimiter. For instance, a comma followed by a space in a command is interpreted as one delimiter, a comma. Similarly, a space followed by one or more blank spaces is taken to be one edit delimiter, a space. This can be important when two successive edit delimiters are required. For example, if starting and ending line numbers are omitted in the SORT command, two successive edit delimiters are required after the keyword:

```
SORT,,1
```

This command tells the EDITOR to rearrange all lines in the text buffer according to the ASCII value of the character found in the first position in each line. The two commas placed side by side within the command are counted as two edit delimiters by the EDITOR.

Normally, commas can be replaced by other edit delimiters. However, because the EDITOR ignores spaces that follow edit delimiters, the following versions of the command are not correct and cause a syntax error:

```
SORT, 1
```

```
SORT. 1
```

```
SORT; 1
```

```
SORT: 1
```

```
SORT= 1
```

In each case, the space within the command is ignored because it immediately follows another edit delimiter. The EDITOR counts only one delimiter, and returns an error if any of these commands are executed.

### When a Space Follows a Command Keyword

The EDITOR ignores spaces that immediately follow a command keyword. This means that for the SORT command described above, the following versions are not correct:

```
SORT ,1
```

```
SORT .1
```

```
SORT ;1
```

```
SORT :1
```

```
SORT =1
```

In each case, the space following the keyword SORT is ignored. The EDITOR only sees one edit delimiter, and returns a syntax error.

### String Delimiters

The second type of delimiter that may appear in an EDITOR command is called a string delimiter, and is used to enclose character strings specified in a command. Although the command syntax forms show quotation marks (") as the string delimiter, any keyboard character that causes a printed character to appear on the display may be used to enclose strings.

The same string delimiter must be used to mark the beginning and the end of a string. For example:

```
SEARCH ~VARIANCE~
```

In this example, the delimiter ~ is used to mark the beginning and the end of the word VARIANCE.

When two strings are specified in an EDITOR command, the delimiter for the first string need not be the same as the delimiter for the second string. For instance, both of the commands shown below are valid:

```
SEARCH "ON ERROR", "ON SIZE"
```

```
SEARCH *ON ERROR*, /ON SIZE/
```

A character used to delimit a string must not occur within the string. For example, the character ' cannot be used to delimit the string WON'T WORK :

## General Information

```
SEARCH 'WON'T WORK'  
EDITOR ERROR  
Syntax - error number 138  
SEARCH 'WON'T WORK'
```

When the command `SEARCH 'WON'T WORK'` is executed, the EDITOR returns a syntax error, and reprints the incorrect command. The syntax error occurs because the delimiter `'` is one of the characters enclosed in the string.

### Using an Edit Delimiter Character as a String Delimiter

Normally, any edit delimiter character except a space may be used as a string delimiter. For example, the following command uses an edit delimiter (`:`) to enclose a string:

```
SEARCH 1,100 :END:
```

However, if the edit line numbers are omitted in the command, and only a blank space appears in their place, a syntax error occurs. The EDITOR returns an error message and reprints the incorrect command:

```
SEARCH :END:  
EDITOR ERROR  
Syntax - error number 148  
SEARCH :END:
```

The rule to remember is that when entering the `SEARCH` and `NLSEARCH` commands with only a space between the keyword and the first string delimiter, do not choose the string delimiter to be one of the characters set aside as edit delimiters. For instance, all of the commands shown below cause syntax errors:

```
SEARCH .END.  
SEARCH =END=  
NLSEARCH ;END;  
NLSEARCH ,END,
```

The above commands cause syntax errors because the delimiter used to enclose the target string is one of the edit delimiters.

### Using a Number as a String Delimiter

If the edit line numbers are omitted in a SEARCH or NLSEARCH command, the delimiter used to enclose the target string must not be a number. For example, the following command causes a syntax error:

```
SEARCH 11END11
EDITOR ERROR
Syntax - error number 138
SEARCH 11END11
```

The error shown above occurs because the number 11 is used to enclose the string END.

## EDIT LINE NUMBERS IN THE COMMAND

### What Edit Line Numbers Are

The EDITOR allows line numbers to be specified as parameters in most commands. The line numbers are called edit line numbers, and are assigned to lines of text by the EDITOR when the RENUMBER command is executed. Edit line numbers appear to the left of the lines of text, and are separated from the text by a colon:

```
LIST
1:100 DIM A(10,10)
2:110 FOR I=1 TO 10
3:120 FOR J=1 TO 10
4:130 A(I,J)=1/(I+J-1)
5:140 NEXT J
6:150 NEXT I
7:160 PRINT "THE 10X10 HILBERT MATRIX IS:"
8:170 PRINT A
9:180 END
```

Edit line numbers are not the same as program line numbers. In the sample text shown above, program line numbers 100, 110, 120, ... are part of the text, and may be edited using EDITOR commands. But the edit line numbers 1, 2, 3, ... are not considered to be part of the text: they are used to refer to specific lines of text, and cannot be changed except by the RENUMBER command.

Edit line numbers may be four digits long, ranging from 0 to 9999. The parameters of the RENUMBER command determine the line numbers that are assigned to the text. For instance, the command RENUMBER 50,5,0 gives edit line numbers 50, 55, 60, ... to each line of text, beginning with the first line in the text buffer.



### When Lines of Text Do Not Have Edit Line Numbers

Lines in the text buffer only receive edit line numbers when the RENUMBER command is executed. This means that newly created lines inserted into the text using the INSERT command do not have edit line numbers. Also, previously stored lines brought into the text buffer using OLD or APPEND, do not have edit line numbers.

The Output commands PRINT, SAVE, and WRITE remove edit line numbers before sending lines to a device. Although SWN (Save With Number) saves edit line numbers along with the text, the numbers are stored as part of the text, and do not function as edit line numbers when brought back into the text buffer.

### How Edit Line Numbers Are Affected by EDITOR Commands

Although edit line numbers can only be created or altered by the RENUMBER command, some EDITOR commands remove edit line numbers from parts of the text. Copied or moved lines (lines sent to a new location by the COPY or MOVE command) lose their edit line numbers upon arriving at their destination. The commands SORT and REVSORT also strip line numbers from lines of text that are sent to a new location.

The following is an example of how this happens:

```
LIST
 1:EDITOR I/O COMMANDS
 2:SAVE
 3:SEARCH
 4:SKIP
 5:SWN
 6:WRITE
 7:APPEND
 8:FIND
 9:INPUT
10:LIST
11:OLD
12:PRINT
```

```
MOVE 7,12,2
```

```
LIST
 1:EDITOR I/O COMMANDS
 :APPEND
 :FIND
 :INPUT
 :LIST
 :OLD
 :PRINT
 2:SAVE
 3:SEARCH
 4:SKIP
 5:SWN
 6:WRITE
```

The first listing shows a piece of text consisting of twelve numbered lines. The command `MOVE 7,12,2` tells the EDITOR to move lines 7 through 12, and place them just before line 2. After the command is executed, a new listing shows that the moved lines no longer have edit line numbers.

The EDITOR removes the numbers in order to keep the edit line numbers in an increasing sequence. Even though some lines are unnumbered, you may continue to execute EDITOR commands, referring to the unnumbered lines as 1+1, 1+2, 1+3, and so on.

### How to Use Edit Line Numbers in a Command

Edit line numbers appear after the command keyword, and are separated by edit delimiters. Many EDITOR commands allow a starting edit line number and an ending edit line number to be specified. The commands APPEND, INSERT, MOVE, and COPY ask for a third edit line number, a destination line number. For example, the following command tells the EDITOR to copy lines 1 through 10 of the text buffer and place the copied lines of text just before line 18:

```
COPY 1,10,18
```

### How to Refer to an Unnumbered Line of Text

Because some EDITOR commands remove edit line numbers from parts of the text, the text buffer may include both numbered and unnumbered lines. For example:

```
LIST
 1: OUTPUT COMMANDS
  : LIST
  : PRINT
  : SAVE
 4: SEARCH
  : SWN
 6: WRITE
```

When a portion of the text is unnumbered, you may execute the RENUMBER command to obtain an edit line number for each line of text. Or, you may continue to perform editing functions, referring to the unnumbered lines by using an "offset."

An offset is a count of the number of lines between a numbered and an unnumbered line. The count begins after any numbered line and continues until the desired unnumbered line is reached. An offset is positive or negative, depending on whether the unnumbered line falls before or after the numbered line.

## General Information

For example, in the preceding text, the unnumbered line consisting of the word SAVE may be referred to in an EDITOR command as line number 1+3, or 4-1:

```
LIST 1+3
      :SAVE
```

```
LIST 4-1
      :SAVE
```

In this example, the EDITOR responds with the same line when asked to list edit line numbers 1+3 and 4-1.

Any line in the text buffer may be identified according to its distance from edit line number 1:

```
LIST 1+1
      :LIST
```

```
LIST 1+2
      :PRINT
```

```
LIST 1+3
      :SAVE
```

```
LIST 1+4
      4:SEARCH
```

```
LIST 1+5
      :SWN
```

```
LIST 1+6
      6:WRITE
```

Or, you may count upward from edit line number 6, using negative values for the offset:

```

LIST 6-1
  :SWN

LIST 6-2
  4:SEARCH

LIST 6-3
  :SAVE

LIST 6-4
  :PRINT

LIST 6-5
  :LIST

LIST 6-6
  1:OUTPUT COMMANDS

```

An offset may be used when referring to a numbered line. For instance, in the last example, the line consisting of OUTPUT COMMANDS has an edit line number (1), but can also be identified as line 6-6:

```

LIST 6-6
  1:OUTPUT COMMANDS

```

A line number with an offset is not a sum. That is 3+1 does not necessarily refer to line 4. Likewise, in the last example shown above, 6-6 refers to edit line number 1, not 0.

#### Using an Offset in an EDITOR Command.

Whenever the term "edit line number" appears in a syntax form, you may specify a line number with an offset of the kind shown above. Here are some sample commands that use offsets to refer to unnumbered lines of text:

```

DEL 0,0+10

SAVE 0+1,1000

LIST 0+1,0+8

COPY 0,0+10,32

MOVE 0+2,0+6,0+10

SORT +7,+10,1

```

## General Information

In the last command shown above, the numbers +7 and +10 are equivalent to 0+7 and 0+10, where 0 refers to the first line of text in the buffer.

### NOTE

*The RENUMBER command can be executed at any time. If you prefer not to use offsets to refer to unnumbered lines of text, execute a RENUMBER command and obtain an edit line number for each line in the text buffer.*

### Specifying a Non-Existent Line Number in a Command

The commands DELETE, NLSEARCH and Delete Line, and SEARCH and Delete Line remove lines of text and their edit line numbers from the buffer. For this reason, some line numbers do not exist that you normally expect to be in the text. For example:

```
LIST
100:REM THIS IS A TABLE OF SQUARE ROOTS
120:PRI "NUMBER","SQUARE ROOT"
130:FOR I=1 TO 100
140:PRI I, SQR(I)
170:NEXT I
180:END
```

In this example, edit line numbers 110, 150, and 160 have been previously deleted from the text buffer and are missing from the normal sequence 100, 110, 120, ... If you specify any of these non-existent line numbers as a parameter in a command, the EDITOR automatically substitutes the line of text having the next largest edit line number. If no larger edit line number exists, the EDITOR assumes that you are referring to some location beyond the end of the text.

Often it is faster and more convenient to specify a non-existent line number when referring to the first line of text, the last line of text, or a location after the last line of text.

**The First Line of Text.** Edit line number 0 always refers to the first line in the text buffer, whether or not the line is numbered. For example:

#### Example 1

```
LIST
: .ODD
60: .EVEN
65: .BLKW
70: .BLKB
75: .RADIX

LIST 0
: .ODD
```

**Example 2**

```
LIST
500:.WORD
501:.BYTE
502:.ASCII
```

```
LIST 0
500:.WORD
```

**Beyond the Last Line of Text.** When specifying edit line numbers in a command, you sometimes need to refer to a location beyond the last line of the text. For instance, you may want moved, copied, or inserted lines to be placed after the last line of text. In this case, you can always refer to a point beyond the end of the text by specifying edit line number 10000. (Edit line number 10000 is always beyond the end of the text, and never exists in the text, because edit line numbers can only be four digits long.)

Any other number that is larger than the largest edit line number in the text may also be used to indicate a point past the end of the text. For instance, if the largest edit line number in the text is 50, any number from 51 to 10000 may be used to refer to a point beyond the last line of text.

For example:

**Example 1**

```
LIST
1:1.1 INTRODUCTION
2:1.2 EMPIRICAL DISTRIBUTIONS
3:1.3 MEASURES OF CENTRAL TENDENCY

INSERT 10000
:1.4 MEASURES OF VARIATION

LIST
1:1.1 INTRODUCTION
2:1.2 EMPIRICAL DISTRIBUTIONS
3:1.3 MEASURES OF CENTRAL TENDENCY
:1.4 MEASURES OF VARIATION
```

## General Information

### Example 2

```
LIST
  1:1.1 INTRODUCTION
  2:1.2 EMPIRICAL DISTRIBUTIONS
  3:1.3 MEASURES OF CENTRAL TENDENCY
  :1.4 MEASURES OF VARIATION

INSERT 4
  :1.5 COMPUTATION OF THE MEAN

LIST
  1:1.1 INTRODUCTION
  2:1.2 EMPIRICAL DISTRIBUTIONS
  3:1.3 MEASURES OF CENTRAL TENDENCY
  :1.4 MEASURES OF VARIATION
  :1.5 COMPUTATION OF THE MEAN
```

When the INSERT command is executed, the EDITOR responds by typing a colon and waiting for a new line to be entered. In both examples shown above, the inserted line is placed after the last line of text, because line numbers 10000 and 4 both refer to a point beyond the end of the text.

**The Last Line of Text.** While 10000 always refers to a location after the last line of text, 10000-1 always refers to the last line of text. This is true whether the last line is numbered or not. An example is shown below:

```
LIST
  50:1.1 INTRODUCTION
  55:1.2 EMPIRICAL DISTRIBUTIONS
  60:1.3 MEASURES OF CENTRAL TENDENCY
  65:1.4 MEASURES OF VARIATION
  70:1.5 COMPUTATION OF THE MEAN

LIST 10000-1
  70:1.5 COMPUTATION OF THE MEAN
```

Any number larger than the largest edit line number in the text, may be used with an offset of -1 to indicate the last line of the text:

```

LIST
:1.1 INTRODUCTION
:1.2 EMPIRICAL DISTRIBUTIONS
:1.3 MEASURES OF CENTRAL TENDENCY
:1.4 MEASURES OF VARIATION
:1.5 COMPUTATION OF THE MEAN

LIST 12-1
:1.5 COMPUTATION OF THE MEAN

```

## HOW MUCH TEXT IS AFFECTED BY AN EDITOR COMMAND

### Starting and Ending Lines

When a starting line number and an ending line number are specified in a command, the EDITOR acts upon the portion of the text buffer from the beginning of the starting line to the end of the ending line. All lines of text between and including the starting line and ending line are affected by the command. Unnumbered lines are affected as well as numbered lines. For example:

```

LIST
1:C      COMPUTE THE SQUARES AND PRINT
:        DIMENSION X(10),Y(10)
:        DO 20 I=1,10
4:        READ (5,100) X(I)
:        Y(I)=X(I)**2
:        WRITE (6,200) X(I),Y(I)
7:20     CONTINUE
8:100    FORMAT (F10.2)
9:200    FORMAT (1H, 15X, 2(F20.5))

DELETE 1,7

LIST
8:100    FORMAT (F10.2)
9:200    FORMAT (1H, 15X, 2(F20.5))

```

In this example, the command `DELETE 1,7` deletes all numbered and unnumbered lines between and including lines 1 and 7 of the text buffer.



## General Information

The parameters of the following EDITOR commands include a starting line and an ending line:

```
CASE
DELETE
LIST
MOVE
NLSEARCH
SEARCH
SORT
REVSORT
SAVE
SWN
WRITE
```

### Destination Lines

The four EDITOR commands INSERT, COPY, MOVE, and APPEND allow a destination line to be specified. The destination line number tells the EDITOR where to place lines that are being repositioned or added to the text buffer. Inserted, copied, moved, or appended lines of text are always placed immediately before the destination line. For example:

```
LIST
1:100 REM ** SUBROUTINE
2:110 PRINT "ENTER CONSTANTS:"
3:120 FOR I=1 TO N
4:130 PRINT "B(";I;")=";
5:140 INPUT B(I)
6:170 RETURN
7:150 NEXT I
8:160 PRINT "END OF INPUT"
```

```
MOVE 7,8,6
```

```
LIST
1:100 REM ** SUBROUTINE
2:110 PRINT "ENTER CONSTANTS:"
3:120 FOR I=1 TO N
4:130 PRINT "B(";I;")=";
5:140 INPUT B(I)
:150 NEXT I
:160 PRINT "END OF INPUT"
6:170 RETURN
```

In this example, the command `MOVE 7,8,6` specifies 7 as a starting line number, 8 as an ending line number, and 6 as a destination line number. After the command is executed, a listing of the text shows that lines 7 and 8 have been moved and placed just before line number 6.

## What Happens When a Non-Existent Line Number is Specified in a Command

It is best not to deliberately specify a non-existent line number in an EDITOR command, except when referring to the first line of text, the last line of text, or a location beyond the last line of text. However, knowing how the EDITOR handles non-existent line numbers can be important, especially if you inadvertently specify a non-existent line number in a command.

When a non-existent edit line number is specified as a starting line, ending line, or destination line, the EDITOR searches for a larger edit line number. The next largest edit line number found in the text is automatically substituted as the command parameter.

This influences the amount of text that is affected by the command, and in what manner. The following examples illustrate what happens when a non-existent starting, ending, or destination line number is specified:

### Example 1

```
LIST
  :BUCKNER, STEVEN
  :SHERMAN, NANCY
  3:SMITH, JACK L.
  4:TURNER, NELL B.
```

```
CASE 1,4
```

```
LIST
  :BUCKNER, STEVEN
  :SHERMAN, NANCY
  3:smith, jack l.
  4:turner, nell b.
```

### Example 2

```
LIST
  1:BUCKNER, STEVEN
  :CALLAHAN, LISA
  3:SHERMAN, NANCY
  4:SMITH, JACK L.
  5:TURNER, NELL B.
```

```
LIST 1,2+1
  1:BUCKNER, STEVEN
  :CALLAHAN, LISA
  3:SHERMAN, NANCY
  4:SMITH, JACK L.
```

## General Information

### Example 3

```
LIST
      :BUCKNER, STEVEN
      :SHERMAN, NANCY
      3:SMITH, JACK L.
      4:TURNER, NELL B.

INSERT 2
      :CALLAHAN, LISA

LIST
      :BUCKNER, STEVEN
      :SHERMAN, NANCY
      :CALLAHAN, LISA
      3:SMITH, JACK L.
      4:TURNER, NELL B.
```

In the first example, the command `CASE 1,4` is given. The EDITOR is unable to find the starting line number 1, and begins looking for an edit line number larger than 1. The next largest line number in the text is 3, so the command acts on the portion of the text that starts with edit line number 3. After the command is executed, a listing shows that lines 3 and 4 have been changed from upper case to lower case.

The second example shows the LIST command entered with starting line number 1 and ending line number 2+1. However, 2 does not appear as a line number in the text. Since the next largest edit line number in the text is 3, the EDITOR uses 3+1 for the ending line number, and lists the first four lines on the display.

In the third example, edit line number 2 is given as the destination line for the INSERT command. The EDITOR is unable to find line number 2, and goes on to use line 3 as the destination line. A later listing shows that the line of text created using the INSERT command, is inserted just before line 3 in the text buffer.

### Special Cases

The EDITOR handles non-existent line numbers in the manner described above, except in the following instances:

- The number 0, which never appears as an edit line number in the text, always refers to the first line in the text buffer. (See "The First Line of Text" on the preceding pages.)
- If a non-existent line number is specified in a command and no larger edit line number can be found in the text, the EDITOR assumes the specified line refers to a location beyond the end of the text. (See "Beyond the Last Line of Text" on the preceding pages.)

—If a non-existent ending line number is specified without an offset in the DELETE command, the EDITOR deletes all numbered and unnumbered lines up to, but not including, the next largest line number found in the text.

## OMITTING A PARAMETER IN A COMMAND: DEFAULT VALUES

EDITOR command parameters include I/O addresses, character strings, edit line numbers, and other numeric constants. Most of these parameters are optional, and if one or more of the parameters is omitted when entering a command, the EDITOR supplies a predetermined value by default. A table of the EDITOR commands and their default parameter values appears on the following pages.

### Optional Edit Line Numbers

More than half of the EDITOR commands allow edit line numbers to be given as parameters. All of these edit line numbers are optional, and may be omitted when entering a command. The only exception to this rule is the DELETE command, which requires at least a starting line number to be specified.

#### Default Values for Optional Edit Line Numbers

The default values for optional starting, ending, and destination line numbers are described in the following paragraphs. The information also appears on the table of default parameter values on the following pages.

**The Starting Line Number.** If the starting line number is omitted in a command, the command begins acting on the first line of text in the buffer. For example, the command `SAVE, 100` saves lines of text from the first line through line 100.

**The Ending Line Number.** If the ending line number is not specified, the last line affected by the command is the last line in the text buffer. For instance `LIST 3,` lists the contents of the text buffer from line 3 to the end of the text.

The only exception to this rule is the DELETE command. The command `DELETE 3,` deletes only line 3, instead of deleting all lines in the text buffer from line 3 on. This is for your protection, to prevent you from accidentally wiping out large portions of the text buffer. If you want to delete all lines in the text buffer from line 3 on, you may do so by entering a command such as `DELETE 3,10000`.

## General Information

**When the Starting and Ending Line Numbers Are Both Omitted.** When the starting line number and the ending line number are both omitted, the command acts upon the entire text buffer. For example, the command `LIST` tells the EDITOR to list all lines in the text buffer, and the command `SEARCH "A$="` or `SEARCH ,, "A$="` tells the EDITOR to search all lines in the text buffer for the string `A$=`.

**The Destination Line Number.** If a destination line is omitted in an EDITOR command, the default destination is before the first line of text for the `INSERT` command, and beyond the end of the text for the `MOVE`, `COPY`, and `APPEND` commands. For example, the command `INSERT` places newly created lines just before the first line of text; but the command `COPY 1,10,` duplicates line 1 through 10 and places the copy after the last line of text.

### Differences between BASIC and EDITOR Commands

BASIC commands do not allow an ending line number to be specified by itself. For instance, the command `SAVE ,100` causes a syntax error in BASIC. Likewise, BASIC does not allow a starting line number to be followed only by a delimiter: the command `SAVE 3,` causes a syntax error.

The EDITOR, however, allows you to give a starting line number by itself, or a starting line number followed by a delimiter. You may specify an ending line number alone, or give a destination line number without any starting or ending line number.

### How to Take Advantage of the Default Values

Being able to specify a starting, ending, or destination line number by itself helps speed the entry of EDITOR commands. If you want to move the first 10 lines of text to the end of the text buffer, for instance, you need only enter `MOVE ,10,` from the keyboard. To save the contents of the text buffer from line 10 on, enter `SAVE 10,` or `WRITE 10,`. Relying on the default values in this way saves you from having to either keep track of the edit line numbers assigned to the first and last line of text or specify 0 and 10000 in the command.

Starting, ending, and destination line numbers can appear in any combination in a command. That is, the following commands are all syntactically correct:

```

COPY
COPY ,,
COPY 2,,
COPY ,5,
COPY ,,10000
COPY 2,5,
COPY ,5,30
COPY 2,,10000
COPY 2,5,30

```

These COPY commands show that an edit line number can appear alone or in combination with the other edit line numbers.

### Optional Edit Delimiters

In any EDITOR command, the number and placement of delimiters is important. Especially when omitting edit line numbers in a command, you must be careful to leave in enough correctly placed delimiters to make the meaning of the command clear to the EDITOR. For example, the SORT commands shown below have different meanings:

```

SORT ,,1,2
SORT ,1,2

```

Both commands are valid, but one of the edit delimiters (,) is omitted in the second command. The first command tells the EDITOR to sort all lines in the text buffer according to the ASCII value of characters found in the first two positions in each line. The second command, however, tells the EDITOR to sort lines up to and including edit line number 1, according to the ASCII value of characters found in the second character position in each line.

### When Edit Delimiters May Be Omitted

Because of their effect on the meaning of commands, you must be careful when leaving out edit delimiters. However, it is useful and convenient to omit edit delimiters in the situations

## General Information

described below:

- No edit delimiters are required when all command parameters are omitted and only the keyword is entered.
- When omitting the starting and ending line numbers in the SEARCH and NLSEARCH commands, the two edit delimiters may be replaced by a blank space. For example:

**SEARCH, , "TEKTRONIX"**

can be shortened to

**SEARCH "TEKTRONIX"**

- When a command ends in a line number followed by an edit delimiter, the delimiter may be omitted without altering the meaning of the command:

### Example 1

**COPY 4,20** is equivalent to **COPY 4,20,**

### Example 2

**SAVE 5** is equivalent to **SAVE 5,**

The commands in Example 1 tell the EDITOR to place a copy of lines 4 through 20 after the last line of text. The commands in Example 2 cause the contents of the text buffer from line 5 on to be stored on magnetic tape.

The EDITOR does not interpret the command **SAVE 5** in the same manner as BASIC. In BASIC the command **SAVE 5** saves program line 5 on the internal magnetic tape. Only one line is saved, line 5. But when executed under EDITOR control, **SAVE 5** saves all lines of text from edit line 5 through the end of the text.

### **When a Starting Line is Given Alone and is Not Followed by an Edit Delimiter.**

Whenever an EDITOR command is entered with only a starting line number, the command affects the text buffer from the starting line number through the end of the text. The following commands act upon all lines in the text buffer from line 50 on:

```

CASE 50
WRITE 50
COPY 50

```

If you want an EDITOR command to affect only one line of text, you must enter that line as the starting and ending line numbers:

```

SAVE 5,5
WRITE 11,11
CASE 30,30
SEARCH 1,1 "*"
COPY 18,18,22

```

All of these commands act upon only one line of text, the one given as the starting and ending line number.

**Exceptions.** Normally, commands entered with only a starting line number affect all lines of text from the starting line number on. However, there are two exceptions to this rule. When a starting line number is given alone and is not followed by an edit delimiter, the LIST and DELETE commands only affect the starting line:

```

LIST
1:100 INIT
2:110 WBYTE @80,108:
3:120 RBYTE X
4:130 WBYTE @63,95:
5:140 M$=CHR(X)
6:150 PRINT M$
7:160 END

```

```

LIST 3
3:120 RBYTE X

```

```

DELETE 3

```

```

LIST
1:100 INIT
2:110 WBYTE @80,108:
4:130 WBYTE @63,95:
5:140 M$=CHR(X)
6:150 PRINT M$
7:160 END

```



## General Information

The above example shows that the command `LIST 3` lists only line 3, not all text from line 3 on. Likewise, after the command `DELETE 3` is executed, a listing shows that only one line has been deleted, line 3.

In summary, when only a starting line number is specified, and no edit delimiter follows the line number, the `LIST` and `DELETE` commands only act upon one line of text. This is to prevent you from accidentally listing or deleting a large portion of the text buffer.

## Other Optional Parameters

Other `EDITOR` command parameters include I/O addresses, ASCII characters, and numeric constants. Most of these parameters are optional. A table of default values for all of the `EDITOR` commands is presented on the following pages. The table gives each `EDITOR` command name, a descriptive form of its syntax showing how the parameters are used in the command, and default values for each optional parameter.



**TABLE OF EDITOR COMMANDS  
AND DEFAULT PARAMETER VALUES  
(cont)**

Command	Syntax (Descriptive Form)	Default Values
<b>RENUMBER</b>	RENUMBER [ new starting line number ] [ , [ increment between new line numbers ] [ , [ line in the current text where renumbering is to begin ] ]	new starting line number = 1 increment between new edit line numbers = 1 where to begin renumbering = first line of text
<b>REVSORT</b>	REVSORT [ starting line number ] , [ ending line number ] , character position [ , character position [ . . . . ] ]	starting line number = first line of text ending line number = last line of text
<b>SAVE</b>	SAVE [ I/O address ] [ starting line number ] [ , [ ending line number ] ]	I/O address = @33,12: starting line number = first line of text ending line number = last line of text
<b>SEARCH</b>	SEARCH [ I/O address ] [ [ starting line number ] , [ ending line number ] , ] space " target string " [ { * , [ " replacement string " ] } ]	I/O address = @32,19: starting line number = first line of text ending line number = last line of text
<b>SKIP</b>	SKIP [ I/O address ] [ number of logical records to advance the READ/WRITE heads ]	number of logical records = 65536
<b>SORT</b>	SORT [ starting line number ] , [ ending line number ] , character position [ , character position [ . . . . ] ]	starting line number = first line of text ending line number = last line of text
<b>SWN (Save With Number)</b>	SWN [ I/O address ] [ starting line number ] [ , [ ending line number ] ]	I/O address = @33,12: starting line number = first line of text ending line number = last line of text
<b>UPPERCASE</b>	UPPERCASE	LOWERCASE
<b>WRITE</b>	WRITE [ I/O address ] [ starting line number ] [ , [ ending line number ] ]	I/O address = @33,12: starting line number = first line of text ending line number = last line of text
<b>] =</b>	] = [ ASCII character ]	initial value: ] = ] default: no assignment
<b># =</b>	# = [ ASCII character ]	initial value: # = # default: no assignment
<b>~ =</b>	~ = [ ASCII character ]	initial value: ~ = ~ default: no assignment
<b>_ =</b>	_ = [ ASCII character ]	initial value: _ = _ default: no assignment

## EDITOR COMMAND SYNTAX

The explanation of each command in Sections 4, 5, and 6 begins with the command syntax and a descriptive form of the syntax. These expressions are constructed in the same manner as the syntax and descriptive forms of BASIC commands described in the 4051 Graphic System Reference Manual.

### Optional Entries

Items enclosed in square brackets are optional. For instance:

<p><b>Syntax Form:</b></p> <p>SK [ I/O address ] [ numeric constant ]</p> <p><b>Descriptive Form:</b></p> <p>SKIP [ I/O address ] [ number of logical records to advance the READ/WRITE heads ]</p>
---

This command can be entered in any of the following forms:

SKIP  
SKIP @29:  
SKIP @29:3  
SKIP 3

### Choices

Items enclosed in braces make up a selection list from which one item must be selected. For example:

## General Information

### Syntax Forms:

```
NL [ [ edit line number ] , [ edit line number ] , ] b " string " { * , " string " }
```

### Descriptive Forms:

```
NLSEARCH [ [ starting line number ] , [ ending line number ] , ] space " target string "  
{ * , " replacement string " }
```

Two ways this command may be used are:

```
NLSEARCH "pd.*"  
NLSEARCH "pd.", "PAID"
```

## Embedded Optional Entries

Embedded optional entries cannot be entered by themselves. For example:

### Syntax Form:

```
S [ I/O address ] [ [ edit line number ] , [ edit line number ] , ] b " string "
```

### Descriptive Form:

```
SEARCH [ I/O address ] [ [ starting line number ] , [ ending line number ] , ] space " target string "
```

These commands are valid:

```
SEARCH 1,1000 "RETURN"  
SEARCH 1,, "RETURN"  
SEARCH ,1, "RETURN"
```

However, the following command is not valid:

**SEARCH 3 "RETURN"**

This command causes a syntax error, because the optional starting line number cannot appear without the two edit delimiters (commas) that are enclosed in the larger set of brackets.

### An Example of EDITOR Syntax

The commands having the most optional entries are COPY, MOVE, and RENUMBER. The syntax and descriptive forms for the COPY command appear below:

<p><b>Syntax Form:</b></p> <p>C [ edit line number ] [ , [ edit line number ] [ , [ edit line number ] ] ]</p> <p><b>Descriptive Form:</b></p> <p>COPY [ starting line number ] [ , [ ending line number ] [ , [ destination for copied text ] ] ]</p>
--

Using the rules for interpreting optional entries and embedded optional entries, the COPY command may be entered in any of these ways:

**COPY**

**COPY 5,, or COPY 5, or COPY 5**

**COPY ,10, or COPY ,10**

**COPY ,,30**

**COPY 5,10, or COPY 5,10**

**COPY ,10,30**

**COPY 5,,30**

**COPY 5,10,30**

## Section 3

# SPECIAL KEYS

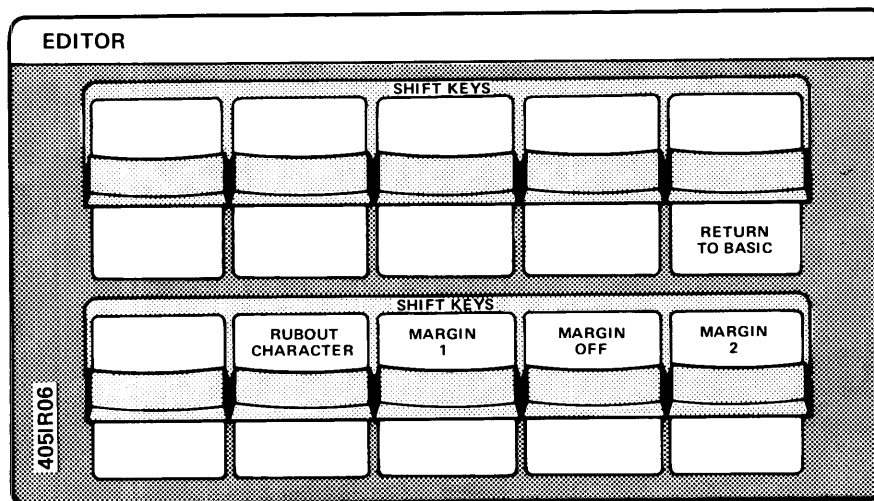
## INTRODUCTION

This section describes the operation of the keys on the Graphic System keyboard while the system is under EDITOR control. Certain keys function in a slightly different manner under EDITOR control than in BASIC; these are explained fully in this section. Other keyboard keys that function under EDITOR control as in BASIC are mentioned briefly in this section; a more complete description may be found in the Graphic System Operator's Manual.

All of the keyboard keys except AUTO LOAD and AUTO NUMBER still function after the EDITOR is called. The alphanumeric keys, LINE EDITOR keys, and peripheral control keys perform the same or similar functions as they do in BASIC. In addition, some of the user-definable keys have special meanings to the EDITOR.

## THE USER-DEFINABLE KEYS

Five of the user-definable keys have predefined uses while the EDITOR is in control. The keys are labeled on the user-definable key overlay as shown below:



2170-3

## RETURN TO BASIC

Pressing the RETURN TO BASIC overlay key returns control of the system to the BASIC Interpreter. EDITOR commands can no longer be executed. The text buffer is cleared, and becomes part of the system RAM space.

## Special Keys

Because the text buffer disappears when control is returned to BASIC, you should save important lines of text on magnetic tape before pressing the RETURN TO BASIC key.

### RUBOUT CHARACTER

Pressing the RUBOUT CHARACTER key causes the symbol ↓ to appear on the display. The symbol ↓ represents the ASCII character "RUBOUT," and is not provided by any of the other keyboard keys. It is an extra character that may be used in lines of text.

### The MARGIN Keys: MARGIN OFF, MARGIN 1, and MARGIN 2

User-definable keys MARGIN OFF, MARGIN 1, and MARGIN 2 control the number of columns of text that appear on the display and what happens when a PAGE FULL condition occurs.

#### MARGIN OFF

Once the MARGIN OFF key has been pressed, information entered from the keyboard or listed on the display appears in two columns. The first column may fill all 72 character positions of the 35 lines on the display. The second column may occupy character positions 37 through 72 of each line on the display.

The first column is filled first, then the second column. Characters begin to fill the second column after the last line of the display is filled, or a CARRIAGE RETURN occurs while the cursor is in the last line. If text is being displayed, lines of text must be less than 36 characters long, to keep the second column of text from overwriting the first column.

When a PAGE FULL condition occurs, *no* blinking F appears in the upper left corner of the screen. Instead, the cursor returns to the HOME position and begins overwriting what is already on the screen.

Upon powering up and calling the EDITOR, the system is automatically set to MARGIN OFF.

#### MARGIN 1

Once MARGIN 1 has been pressed, information from the keyboard or listed on the display appears in one column. When the screen is full, a blinking F appears in the upper left corner of the display, and the HOME/PAGE key must be pressed for writing to continue.

#### MARGIN 2

Once the MARGIN 2 key is pressed, information appears on the display in two columns, as it



does after MARGIN OFF. However, when the screen is full, the blinking F appears in the upper left corner, prompting you to press HOME/PAGE.

### **Resetting the MARGIN Parameter**

The EDITOR "remembers" the choice of MARGIN OFF, MARGIN 1, or MARGIN 2 until the system is powered down, or a different MARGIN key is pressed. Returning control of the system to BASIC does not affect the MARGIN choice. If you press MARGIN 1, for example, returning to BASIC does not reset the MARGIN parameter: the next time you call the EDITOR, it "remembers" your selection of MARGIN 1.

Although returning control of the system to BASIC does not affect the MARGIN choice, turning the system power off resets the MARGIN parameter to MARGIN OFF by default.

## **KEYBOARD KEYS**

### **THE ALPHANUMERIC KEYS**

The alphanumeric keys operate under EDITOR control, including special keys HOME/PAGE, ESC, TAB, TTY LOCK, CTRL, SHIFT, BACKSPACE, LF, RETURN, RUBOUT, BREAK, and the SPACE bar. All of these keys function under EDITOR control as they do in BASIC, with the exception of RETURN and BREAK.

### **The RETURN Key**

When RETURN is pressed, the EDITOR examines the current contents of the line buffer. If the information contained in the line buffer is a command, the EDITOR immediately executes the command. However, if the data is a line of text the EDITOR sends it to the text buffer.

### **The BREAK Key**

The BREAK key has several uses. The functions provided by BREAK are summarized as follows:

- After the INSERT command is executed or one of the RECALL keys is used, pressing BREAK causes the current contents of the line buffer to be loaded into the text buffer. At the same time, the EDITOR is removed from insert mode (prompt mode) and returned to normal command mode.
- BREAK may be used to interrupt listing (LIST and SEARCH commands) or searching operations (NLSEARCH and SEARCH commands).

## Special Keys

- While the Search and Edit Line command is executing, pressing BREAK tells the EDITOR to begin searching for the next occurrence of the target string.
- Pressing BREAK twice provides an immediate exit from a BUSY condition. This operation should be used with caution and only as a last resort.

## THE LINE EDITOR KEYS

All of the LINE EDITOR keys may be used while the system is under EDITOR control. Although RUBOUT←/BACKSPACE and RUBOUT→/SPACE function exactly as they do in BASIC, the other three LINE EDITOR keys work somewhat differently under EDITOR control.

### The COMPRESS/EXPAND Key

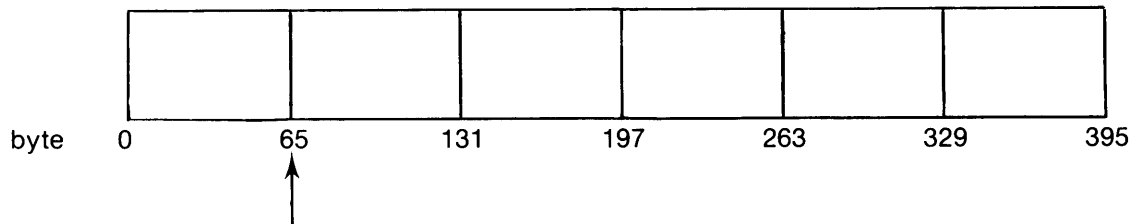
#### The EXPAND Function

The EXPAND function is used to create space for inserting new characters within a line of text. As in BASIC, you bring the line into the line buffer using the RECALL keys, position the cursor over the first character to be shifted to the right side of the display, and press the COMPRESS/EXPAND key.

What happens next depends on the number of characters currently being held in the line buffer, and on the position of the "end-of-line marker."

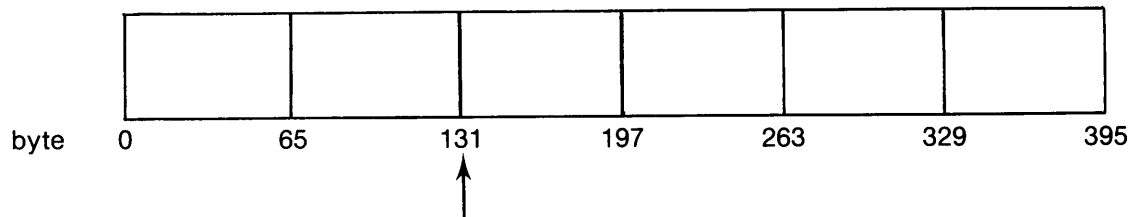
**The End-of-Line Marker.** Unlike BASIC's 72-byte line buffer, the EDITOR line buffer is 396 bytes long, and holds up to 396 characters (a character occupies 1 byte of space). The EDITOR line buffer consists of six units of 66 bytes each, with an "end-of-line marker" positioned over the last byte in one of the six units. The end-of-line marker plays an important role in how the EXPAND and COMPRESS functions operate.

Immediately after the EDITOR is called, the end-of-line marker is at the end of the first unit:

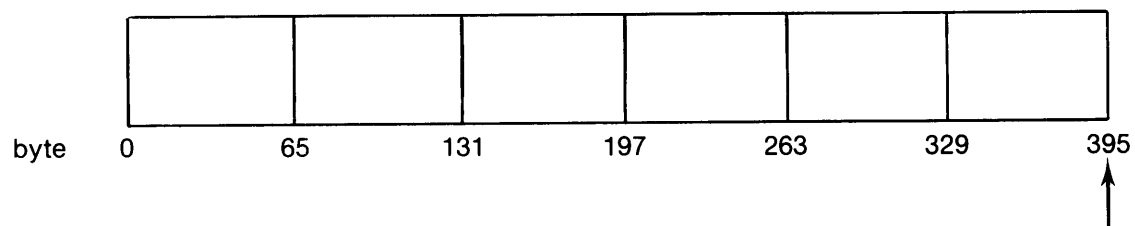


However, the position of the marker may change, depending on the number of characters entered in the line buffer.

**What Causes the End-of-Line Marker to Move.** The position of the end-of-line marker depends on the length of the largest line that has been entered in the line buffer since the EDITOR was called. If all of the lines entered in the line buffer have been less than 66 characters long, the marker remains at the end of the first unit. However, as soon as a line having 66 or more characters is placed in the line buffer, the marker jumps to the end of the second unit:



If a line containing 132 or more characters is placed in the line buffer, the end-of-line marker moves to the end of the third unit. And if a line having 330 or more characters is placed in the line buffer, the end-of-line marker moves to the end of the buffer:



The EDITOR "remembers" the length of the largest line entered in the line buffer. That is, once the marker is moved to the right by a long line, subsequent shorter lines do not affect the marker.

Pressing the COMPRESS/EXPAND key can also cause the end-of-line marker to change position. The marker may move to the right after EXPAND, or to the left after COMPRESS.

**What Happens When EXPAND is Performed.** When the COMPRESS/EXPAND key is pressed without pressing the SHIFT key, the EDITOR checks the line buffer and notes the current position of the end-of-line marker. Then the EDITOR examines the 10 character positions (bytes) immediately preceding the marker. If there are any non-blank characters in these positions, the EDITOR moves the end-of-line marker to the end of the next unit.

Next the EDITOR performs the expansion. All characters to the right of the cursor, including the character underneath the cursor, are moved to the right, until the end-of-line marker is reached. On the screen the line appears to be split into left and right portions separated by a gap. Additional characters may now be inserted into the gap.

**The COMPRESS Function**

Once you have finished inserting characters into an expanded line, you may close the gap by performing the COMPRESS function. Just as for EXPAND, what happens when COMPRESS is performed depends on the number of characters currently in the line buffer, and on the position of the end-of-line marker.

**What Happens When COMPRESS is Performed.** When the COMPRESS/EXPAND key and the SHIFT key are pressed at the same time, the EDITOR checks the line buffer and notes the current position of the end-of-line marker. Then the EDITOR examines the unit (66 bytes) that immediately precedes the marker. If there are only blank characters in these positions, the EDITOR moves the end-of-line marker one unit to the left, and repeats the procedure for the next unit.

Once a unit containing some non-blank characters is reached, the EDITOR performs the compression. The portion of the line that appears to the right on the screen is shifted to the current position of the cursor.

**An Example**

Suppose you want to add a character after the word "OF" in line 2 shown below:

```
LIST
1:*****
*****
2:SOLUTION OF LINEAR EQUATIONS
```

The first step is to enter 2 from the keyboard and press RECALL LINE. Line 2 is reprinted and recalled to the line buffer with the cursor over the first character in the line:

```
2:SOLUTION OF LINEAR EQUATIONS
```

Next you position the cursor after the word "OF," and press EXPAND.

The EDITOR looks for the end-of-line marker. Suppose in this example that the end-of-line marker is at the end of the second unit. The EDITOR then examines the 10 bytes of the line buffer that precede the end-of-line marker, bytes 122 through 131 in this case. Since bytes 122 through 131 are blank, the end-of-line marker does not move.

The EDITOR expands the line, moving the right portion to the right until the end-of-line marker is reached. Since the end-of-line marker is at the end of the second unit, the last character in

the line moves to the 131st position in the line buffer. On the screen the right portion of the line appears to have moved. Since the end-of-line marker is at the end of the second unit, the last character in the line moves to the 131st position in the line buffer. On the screen the right portion of the line appears to have moved to the right, and down one line:

**SOLUTION OF**

**LINEAR EQUATIONS**

You now insert characters into the line:

**SOLUTION OF N**

**LINEAR EQUATIONS**

Finally, you reposition the cursor and press the COMPRESS key. Because the end-of-line marker is still at the end of the second unit, the EDITOR examines the second unit (bytes 66 through 131) for non-blank characters. The second unit does contain some non-blank characters, so the EDITOR does not move the marker, and goes on to perform the compression:

**SOLUTION OF N LINEAR EQUATIONS**

## The REPRINT/CLEAR Key

### The REPRINT Function

Pressing the REPRINT/CLEAR key and the SHIFT key at the same time causes the current contents of the line buffer to appear on the screen immediately below the last line displayed. The cursor moves down one line, retaining its position in the line.

If the line buffer currently contains a line of text, pressing REPRINT causes the line to be reprinted without an edit line number, and without the colon (:) that normally precedes a line of text. After the line of text is reprinted, the EDITOR remains in "insert" mode. Pressing RETURN causes the prompt character (:) to appear, asking you to enter a new line of text. If you do not want to create a new line of text, pressing BREAK removes the EDITOR from "insert" mode.

If the line buffer has been cleared by pressing RETURN, BREAK, or CLEAR, the REPRINT function reprints the line most recently held in the line buffer. After a line of text is reprinted, pressing RETURN causes the prompt character (:) to appear if the EDITOR was in "insert" mode before performing REPRINT. (Refer to the INSERT command for an explanation of the "insert" mode.)

## Special Keys

As in BASIC, the REPRINT function is most useful for displaying a clear copy of a line that has been made unreadable by type-over corrections and RUBOUT symbols.

### The CLEAR Function

Pressing the REPRINT/CLEAR key without pressing the SHIFT key causes the contents of the line buffer to be erased (cleared). If CLEAR is performed after a previously stored line is brought back into the line buffer by using RECALL or RECALL NEXT LINE, the contents of the line is lost. The line is erased from both the line buffer and the text buffer. A listing of the text buffer shows a blank line (a colon followed by blanks) where the erased line was previously stored. Likewise, if CLEAR is pressed when the Search and Edit Line command pauses for you to edit a line, the line disappears from the line buffer and the text buffer. A blank line remains in the text buffer where the erased line previously appeared.<sup>1</sup>

If you inadvertently delete a line of text by pressing CLEAR while the line is in the line buffer, you may recover the line by immediately pressing REPRINT, then BREAK or RETURN. Immediately pressing REPRINT recalls a "backup" copy of the deleted line to the line buffer. Pressing BREAK or RETURN then returns the line to the text buffer.

### The RECALL NEXT LINE/RECALL LINE Key

#### The RECALL LINE Function

The RECALL LINE function is performed after entering an edit line number from the keyboard. Pressing the RECALL NEXT LINE/RECALL LINE key without pressing the SHIFT key causes the specified line of text to be pulled from the text buffer and loaded into the line buffer. The cursor is positioned at the beginning of the recalled line.

The line may now be altered or edited using the keyboard keys and the other LINE EDITOR keys. As in BASIC, you simply space forward or backspace to incorrect characters, and type over the errors with correct information. You may use the RUBOUT, RUBOUT←, and RUBOUT→ keys to delete characters, and the COMPRESS/EXPAND key to allow additional characters to be inserted within the line.

When you have finished editing, press BREAK to return the corrected line to the text buffer.

Pressing RETURN also returns the corrected line to the text buffer, but leaves the EDITOR in insert mode (prompt mode) until BREAK is pressed. This provides an alternate way to insert new lines of text into the text buffer. For instance, to add a new line of text immediately after line 50 in the text buffer, enter 50 from the keyboard, press RECALL LINE, and then RETURN. The EDITOR returns a colon, prompting the entry of new text. You then enter a line of text, press BREAK, and the newly created line is inserted in the text buffer immediately following edit line number 50.

<sup>1</sup>This is slightly different from the way the CLEAR key works in BASIC. In BASIC, if the CLEAR key is pressed after a previously stored line is recalled to the line buffer, the line is cleared from the line buffer, but not deleted from memory.

### **The RECALL NEXT LINE Function**

Like RECALL LINE, RECALL NEXT LINE is performed after entering an edit line number from the keyboard. However, instead of returning the line having the specified edit line number, RECALL NEXT LINE returns the next line in the text buffer (whether it is numbered or not). For instance, entering 50 and then pressing the SHIFT and RECALL NEXT LINE/RECALL LINE keys, recalls the line of text that immediately follows line 50 in the text buffer.

After a RECALL LINE or RECALL NEXT LINE is performed and before BREAK is pressed, either RECALL LINE or RECALL NEXT LINE may be performed again. In either case, the current contents of the line buffer are returned to the text buffer, and the next line in the text buffer is recalled. RECALL LINE or RECALL NEXT LINE may be performed repeatedly in this manner without entering any edit line number, in order to recall consecutive lines for editing.

### **The STEP PROGRAM Key (RECALL PREVIOUS LINE)**

The STEP PROGRAM key functions in a completely different manner under EDITOR control than in BASIC. The STEP PROGRAM key is pressed after an edit line number is entered from the keyboard. The line that immediately precedes the specified line in the text buffer is recalled to the line buffer. For example, entering 50 and pressing STEP PROGRAM causes the line of text immediately preceding line 50 in the text buffer, to be recalled to the line buffer for editing.

A line that has been recalled to the line buffer using the STEP PROGRAM key can be altered with the help of the keyboard and LINE EDITOR keys. As when using RECALL LINE or RECALL NEXT LINE, you press BREAK to send the corrected line to the text buffer. Pressing RETURN also sends the corrected line to the text buffer, but leaves the EDITOR in insert mode.

After RECALL LINE, RECALL NEXT LINE, or STEP PROGRAM and before BREAK is pressed, STEP PROGRAM may be pressed again. The line currently in the line buffer is returned to the text buffer, and the preceding line is recalled to the line buffer. STEP PROGRAM may be pressed repeatedly in this way without entering an edit line number, in order to recall lines one by one for editing.

### **Peripheral Control Keys**

Peripheral control keys REWIND and MAKE COPY function under EDITOR control as they do in BASIC. Pressing REWIND causes the EDITOR to rewind the tape cartridge in the internal magnetic tape unit, and pressing MAKE COPY causes an attached Hard Copy Unit to make a copy of the information currently displayed on the screen.

## Section 4

# EDITING COMMANDS

<b>CONTENTS</b>	<b>PAGE</b>
The CARD Command .....	4-3
The CASE Command .....	4-9
The COPY Command .....	4-13
The DELETE Command .....	4-19
The INSERT Command .....	4-23
The LIST Command .....	4-29
The MOVE Command .....	4-35
The NLSEARCH Command .....	4-39
The NLSEARCH and Delete Line Command .....	4-41
The NLSEARCH and Replace String Command .....	4-47
The SEARCH Command .....	4-55
The SEARCH and List Line Command .....	4-59
The SEARCH and Edit Line Command .....	4-65
The SEARCH and Delete Line Command .....	4-73
The SEARCH and Replace String Command .....	4-79
The SORT Command .....	4-87
The REVSORT Command .....	4-95



## NOTES

## The CARD Command

### Syntax Form:

CA [ numeric constant ] [ , [ numeric constant ] ]

### Descriptive Form:

CARD [ number of characters ] [ , [ fill character (decimal equivalent) ] ]

## PURPOSE

The CARD command formats text into lines of a specified length. Lines of text that are longer than the desired length are split into two or more lines, and lines of text that are too short are filled out to the desired length with a specified ASCII character.

## EXAMPLES

```
.  
CA  
CA 50,  
CA ,46  
CA 50,46
```

## EXPLANATION

The CARD command formats all lines in the text buffer to a uniform length. Two parameters may be entered after the keyword CARD. The first parameter specifies the number of characters each line is to contain. The second parameter is the decimal code number for an ASCII character. The ASCII character is called the "fill character," because it is used when needed to fill out lines of text to the desired length.

For example, the command CA 50,46 tells the EDITOR to format the current contents of the text buffer into lines that are 50 characters long. Lines having fewer than 50 characters are "filled out" using the ASCII character . (decimal equivalent 46). That is, when a line is found

## Editing Commands

### CARD

to have fewer than 50 characters, character positions to the right of the last character in the line are filled with the character . until the length of the line is increased to 50 characters.<sup>1</sup>

If any line has more than the specified number of characters, the CARD command is not immediately executed. Instead, the EDITOR lists on the display the edit line numbers of all lines found to be longer than the desired length. (If the lines are unnumbered, no list appears. A blank line is printed on the display for each line found to be too long.) Then the EDITOR prints a message on the screen, instructing you to type "C" if you wish to continue.

If you enter an uppercase C from the keyboard, the CARD command is executed. Lines shorter than the specified line length are filled out to the desired length using the ASCII fill character. However, lines previously found to be longer than the specified length, are split into two or more lines. When two or more lines are created in this way, the fill character is used if needed to fill out the last of the new lines.

If you do not want long lines of text to be split in the manner described above, respond to the message **Type "C" to continue** by pressing RETURN or entering any character other than uppercase C. This prevents the EDITOR from executing the CARD command. The text remains unchanged, and all lines are intact.

### Default Values

Both parameters for the CARD command are optional. The examples listed above show that both parameters may be omitted, as in the command CA . The number of characters may be specified and the fill character omitted, as in CA 50, . Or, the fill character may be given alone, as in the command CA ,46 .

When the number of characters is omitted in a CARD command, the EDITOR supplies the value 80 by default. When the fill character is omitted, the EDITOR supplies the value 32, the ASCII code for a space. Thus the command CA is equivalent to CA 80,32 . The commands CA 50, and CA 50,32 are the same; and CA ,46 is equivalent to CA 80,46 .

### Notes on the Command Syntax

As indicated in the syntax form, the edit delimiter (,) is optional when the fill character is omitted. For example, the commands CA 50, and CA 50 are the same. Both commands tell the EDITOR to format the text into lines that are 50 characters long, using spaces as needed to fill lines to 50 characters.

<sup>1</sup> Not including the end-of-record character at the end of the line.

## An Editing Example

The following examples show the CARD command being used to format a sample piece of text.

### Example 1

```
LIST
1:Machine dependent binary code
2:Magnetic tape format compatibility
3:Magnetic tape statements
4:Math functions
5:Matrix addition
6:Matrix assignment
7:Minus print fields
8:Modular design of basic statements
9:Nesting
10:Numeric constants
11:Numeric variables
12:Operators, logical
13:Output to printer, formatted
14:Peripheral device numbers
```

CARD 43,45

```
LIST
1:Machine dependent binary code-----
2:Magnetic tape format compatibility-----
3:Magnetic tape statements-----
4:Math functions-----
5:Matrix addition-----
6:Matrix assignment-----
7:Minus print fields-----
8:Modular design of basic statements-----
9:Nesting-----
10:Numeric constants-----
11:Numeric variables-----
12:Operators, logical-----
13:Output to printer, formatted-----
14:Peripheral device numbers-----
```

Example 1 shows the CARD command being used to format lines of free text. The command CA 43,45 tells the EDITOR to format the text into lines of 43 characters, using a hyphen (ASCII code 45) as a fill character.

Since all of the lines have fewer than 43 characters, no message appears on the display, and the CARD command is immediately executed. After the command is executed, a listing of the text shows that hyphens (-) have been added to fill each line to a uniform length of 43 characters.

**Editing Commands**  
**CARD**

**Example 2**

```
LIST
  1:Machine dependent binary code
  2:Magnetic tape format compatibility
  3:Magnetic tape statements
  4:Math functions
  5:Matrix addition
  6:Matrix assignment
  7:Minus print fields
  8:Modular design of program statements
  9:Nesting
 11:Numeric variables
 12:Operators, logical
 13:Output to printer, formatted
 14:Peripheral device numbers
```

```
CARD 34,45
      8
Type "C" to continue
```

```
LIST
  1:Machine dependent binary code-----
  2:Magnetic tape format compatibility
  3:Magnetic tape statements-----
  4:Math functions-----
  5:Matrix addition-----
  6:Matrix assignment-----
  7:Minus print fields-----
  8:Modular design of program statemen
   :ts-----
  9:Nesting-----
 11:Numeric variables-----
 12:Operators, logical-----
 13:Output to printer, formatted-----
 14:Peripheral device numbers-----
```

Example 2 illustrates what happens when one or more of the lines of text is longer than the line length specified in the CARD command. The same sample piece of text is used, but this time the command CA 34,45 is entered.

Because line 8 has more than 34 characters, the CARD command is not immediately executed. Instead, the EDITOR prints the number 8 on the display, and a message appears, asking for a "C" as a signal to continue.

Entering an uppercase C causes the cursor to reappear at the beginning of the next line. The character C does not appear on the display. The CARD command has been executed.

A listing of the text shows that lines having fewer than 34 characters are filled out to the 34th character position using hyphens (-). Line 8 is now split into two lines of 34 characters each, with the second line filled to the 34th position by the character - . The newly created line that appears after line 8 has no edit line number.

If an uppercase C had not been entered from the keyboard after the message appeared, the CARD command would not have been executed.

### Special Uses For the CARD Command

A CARD command that specifies 80 as the first parameter is used to format text into "card images," lines having 80 characters each. Lines formatted into card images may be sent to a device that asks for 80-character records and accepts the 80th character as the end-of-record character.<sup>2</sup>

The CARD command may also be executed before text is rearranged by SORT or REVSORT commands. A CARD command that specifies a fill character having a decimal equivalent less than 33, formats text so that the REVSORT command is the inverse of the SORT command. (Refer to the explanations of the SORT and REVSORT commands later in this section.)

An ASCII code chart is included in Appendix B, for use in finding the decimal equivalents of ASCII characters.

<sup>2</sup>After a CARD command formats text into lines of 80 characters, a CR (CARRIAGE RETURN) character remains in the 81st position of each line. The final CR character can be removed before sending the lines to an external device. The method of removing CR characters depends on how the lines are sent to the device. For instance, if the lines are transmitted using the 4051 Data Communications Interface, a parameter of the CALL "TCRLF" routine signals the interface to remove the final CR character from each line.

## NOTES

## The CASE Command

### Syntax Form:

```
CAS [ edit line number ] [ , [ edit line number ] ]
```

### Descriptive Form:

```
CASE [ starting line number ] [ , [ ending line number ] ]
```

## PURPOSE

The CASE command changes lowercase text characters a-z into their uppercase equivalents A-Z if the uppercase flag has been set. If the lowercase flag has been set, the CASE command changes uppercase text characters A-Z into their lowercase equivalents a-z.

## EXAMPLES

```
CAS  
CAS 50,  
CAS ,50  
CAS 1,100  
CAS 0,0+50
```

## EXPLANATION

The CASE command allows a starting edit line number and an ending edit line number to be specified. The starting and ending line numbers may be entered with an offset as in the command `CAS 0,0+50`, or without an offset as in `CAS 1,100`. The command acts upon all lines of text between and including the starting and ending lines. For instance, the command `CAS 0,0+50` affects the first 51 lines (lines 0 through 0+50) of the text buffer.

What happens when the CASE command is executed depends on whether the uppercase or lowercase flag is set (refer to the explanations of the UPPERCASE and LOWERCASE commands). If the uppercase flag is set, the EDITOR examines the current contents of the text



## Editing Commands

### CASE

buffer from the starting line through the ending line, and changes lowercase characters a-z into their uppercase equivalents A-Z. If the lowercase flag is set, the EDITOR changes uppercase characters A-Z into lowercase characters a-z.

#### Default Values

Both the starting and ending lines are optional. The examples listed above show that the starting line number, ending line number, or both, may be omitted when entering the command. When the starting line number is omitted, the CASE command begins acting on the first line in the text buffer. When the ending line number is omitted, the last line affected by the command is the last line in the text buffer.

For example, the command `CAS` acts upon all lines in the text buffer. The command `CAS 50`, acts upon all lines from line 50 on, and `CAS ,50` affects all lines up to and including edit line 50.

#### Notes on the Command Syntax

As indicated in the syntax form, the edit delimiter (,) is optional when the ending line number is omitted. For example, the command `CAS 50`, and `CAS 50` are the same. Both commands tell the EDITOR to check for a lowercase or uppercase flag, then make appropriate changes in all lines of text from edit line 50 on.

#### Command Semantics

The ending line number should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command either has no effect, or causes a semantic error as in the following example:

```
CAS 7,2
EDITOR ERROR
Semantic - error number 139
CAS 7,2
```

The correct way to enter the command is as follows:

```
CAS 2,7
```

## An Editing Example

The following examples show how the CASE command changes characters in a sample piece of text.

### Example 1

```
LIST
1:CRT and shield assembly removal
2:This procedure requires two people and should
3:be carried out only by qualified service
4:personnel. A crt is a high vacuum device and
5:is dangerous if not handled properly.
7:      WARNING
8:The crt may implode if it is scratched or
9:struck severely. Do not handle the crt by its
10:neck. Wear protective clothing and a face
11:shield when handling the crt.
```

UPPERCASE

CASE 1,1

CASE 7,11

```
LIST
1:CRT AND SHIELD ASSEMBLY REMOVAL
2:This procedure requires two people and should
3:be carried out only by qualified service
4:personnel. A crt is a high vacuum device and
5:is dangerous if not handled properly.
7:      WARNING
8:THE CRT MAY IMplode IF IT IS SCRATCHED OR
9:STRUCK SEVERELY. DO NOT HANDLE THE CRT BY ITS
10:NECK. WEAR PROTECTIVE CLOTHING AND A FACE
11:SHIELD WHEN HANDLING THE CRT.
```

The first listing in Example 1 shows 11 lines of text containing lowercase characters. The command UPPERCASE is entered to set the uppercase flag. Next the commands CASE 1,1 and CASE 7,11 are executed.

Because the uppercase flag is set, the EDITOR changes lowercase characters a-z into uppercase characters A-Z. A listing of the text after the two CASE commands are executed shows only uppercase characters in line 1 and in lines 7 through 11 of the text buffer.

## Editing Commands

### CASE

#### Example 2

```
LIST
:INDICATORS 1-9, 1-11
:INDICATOR, SPARE 1-13
:INITIAL CONDITION 1-14
:INPUT FIELD 2-7
:INPUT/OUTPUT 3-1
:INSERT CHARACTER 1-22, 2-3
:INSERT LINE 1-23, 2-3, 2-6
:INSTALLATION 2-14
:INTERFACE 1-3, 1-4, 1-6, 2-16, 2-21
:JOIN TYPES, RULINGS 2-10
:KEYBOARD FUNCTION (PURPOSE OF) 1-10
:KEYBOARD, NUMERIC 1-21
:LINE FEED 1-24, 2-1, 2-3, 2-7
:LOCAL OPERATION 1-7

LOWERCASE

CASE

LIST
:indicators 1-9, 1-11
:indicator, spare 1-13
:initial condition 1-14
:input field 2-7
:input/output 3-1
:insert character 1-22, 2-3
:insert line 1-23, 2-3, 2-6
:installation 2-14
:interface 1-3, 1-4, 1-6, 2-16, 2-21
:join types, rulings 2-10
:keyboard function (purpose of) 1-10
:keyboard, numeric 1-21
:line feed 1-24, 2-1, 2-3, 2-7
:local operation 1-7
```

The first line in Example 2 shows unnumbered lines of text containing only uppercase characters. The command `LOWERCASE` is entered to set the lowercase flag, then the command `CASE` is executed.

Because the lowercase flag is set, the EDITOR changes uppercase characters into lowercase characters. The command `CASE` tells the EDITOR to change characters to lowercase in all lines of text. A listing after the command is executed shows that all uppercase characters have been changed to lowercase.

### Special Uses for the CASE Command

Some specialized systems are designed to accept only uppercase characters. The `CASE` command may be used to convert characters to uppercase before sending them to a specialized system.

## The COPY Command

### Syntax Form:

C [ edit line number ] [ , [ edit line number ] [ , [ edit line number ] ] ]

### Descriptive Form:

COPY [ starting line number ] [ , [ ending line number ] [ , [ destination for copied text ] ] ]

## PURPOSE

The COPY command duplicates specified lines of text, and sends the copied lines to a desired location in the text buffer.

## EXAMPLES

C

C 5

C ,5

C 5,5

C 5,5,250

C 100,500,1235

## EXPLANATION

The COPY command allows a starting line number, an ending line number, and a destination line number to be specified. When the COPY command is executed, all lines of text between and including the starting and ending lines are copied, and placed immediately before the specified destination line. For instance, the example C 100,500,1235 listed above copies text lines 100 through 500, and inserts the copy immediately before line 1235. The command C 5,5,250 copies line 5, placing the copy immediately before line 250.

## Editing Commands

### COPY

#### Default Values

The starting, ending, and destination line numbers are optional, and may be omitted or entered in any combination. Several examples are listed above.

When the starting line number is omitted, the first line copied by the COPY command is the first line of text. When the ending line number is omitted in the command, the last line copied is the last line of text. When the destination line is omitted, copied text is placed after the last line of text.

In the examples listed above, the command C copies all of the current text. The copy appears below the original text. The command C 5 makes a copy of all text from line 5 on, placing the copy after the last line of text. C ,5 copies all text up to and including line 5, placing the copy after the last line of text.

#### Notes on the Command Syntax

As indicated by the brackets in the syntax form, final edit delimiters (,) are optional. When an edit delimiter is the last entry in the command, it may be omitted without changing the meaning of the command. That is, the commands C 5,, and C 5, and C 5 are equivalent, and C 1,3, is the same as C 1,3 .

#### Command Semantics

The ending line number should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command makes no sense, and has no effect on the text buffer. For example, the command C 4,2,8 has no effect on the current text. The correct way to enter the command is C 2,4,8 .

When only a destination line is specified, the line number must be larger than the largest line number currently assigned to the text. For instance, the command C ,,3 causes a semantic error if the text buffer contains lines numbered 1 through 8:

```
LIST
1:      REAL FUNCTION RSCALE(IFACT)
2:      COMMON /BTEST/ ERROR,DEBUG,EOM,HNDSHK,
3:      &  COMPB(259),SUBSTR(11),MASTER,OPSEQ(5),
4:      &  LINTRM,BLKTRM,INBIT,OUTBIT,TRCOUN,BUFP,
5:      &  NAMBUF(23),FAULT
6:      INTEGER ERROR,DEBUG,EOM,HNDSHK,COMPB,
7:      &  SUBSTR,MASTER,OPSEQ,LINTTM,BLKTRM,
8:      &  TRCOUN,BUFP,FAULT,INBIT,OUTBIT,NAMBUF
```

```
C ,,3
EDITOR ERROR
Semantic - error number 139
C ,,3
```

The reason for the semantic error is that the command C ,,3 asks the EDITOR to copy the entire text, and insert the copy before line 3. However, line 3 lies within the portion of text to be copied.

Although the command C ,,3 is syntactically correct, it does not make sense in this particular example. The EDITOR assumes the command does not convey the meaning you intend, and returns a semantic error.

The above example illustrates a general rule. The destination line must not lie between the starting and ending lines in the text buffer, or a semantic error occurs. For example, the command C 1,10,3 causes a semantic error, because the destination line (edit line number 3) falls between the starting and ending lines (edit lines 1 and 10).

### An Editing Example

The following example shows the COPY command being used to duplicate parts of a sample FORTRAN program.

#### Example 1

```

LIST
 1:      REAL FUNCTION RSCALE(IFACT)
 2:      COMMON /BTEST/ ERROR,DEBUG,EOM,HNDSHK,
 3:      &  COMPB(259),SUBSTR(11),MASTER,OPSEQ(5),
 4:      &  LINTRM,BLKTRM,INBIT,OUTBIT,TRCOUN,BUFP,
 5:      &  NAMBUF(23),FAULT
 6:      INTEGER ERROR,DEBUG,EOM,HNDSHK,COMPB,
 7:      &  SUBSTR,MASTER,OPSEQ,LINTTM,BLKTRM,
 8:      &  TRCOUN,BUFP,FAULT,INBIT,OUTBIT,NAMBUF
 9:      RSCALE=FLOAT(IFACT)/256.0
10:      RETURN
11:      END
12:      REAL FUNCTION RANGLE(IDEGR)
13:      RANGLE=FLOAT(IDEGR)*180.0/32767.0
14:      SUBROUTINE MOVE(X,Y)
15:      INTEGER X,Y
16:      CALL OPAIR(28,X,Y)

```

C 10,11,14

C 10,11,17

C 2,8,13

C 2,8,15

(continued on next page)

## Editing Commands

### COPY

```
LIST
1: REAL FUNCTION RSCALE(IFACT)
2: COMMON /BTEST/ ERROR,DEBUG,EOM,HNDSHK,
3: &  COMPB(259),SUBSTR(11),MASTER,OPSEQ(5),
4: &  LINTRM,BLKTRM,INBIT,OUTBIT,TRCOUN,BUFP,
5: &  NAMBUF(23),FAULT
6: INTEGER ERROR,DEBUG,EOM,HNDSHK,COMPB,
7: &  SUBSTR,MASTER,OPSEQ,LINTTM,BLKTRM,
8: &  TRCOUN,BUFP,FAULT,INBIT,OUTBIT,NAMBUF
9: RSCALE=FLOAT(IFACT)/256.0
10: RETURN
11: END
12: REAL FUNCTION RANGLE(IDEGR)
: COMMON /BTEST/ ERROR,DEBUG,EOM,HNDSHK,
: &  COMPB(259),SUBSTR(11),MASTER,OPSEQ(5),
: &  LINTRM,BLKTRM,INBIT,OUTBIT,TRCOUN,BUFP,
: &  NAMBUF(23),FAULT
: INTEGER ERROR,DEBUG,EOM,HNDSHK,COMPB,
: &  SUBSTR,MASTER,OPSEQ,LINTTM,BLKTRM,
: &  TRCOUN,BUFP,FAULT,INBIT,OUTBIT,NAMBUF
13: RANGLE=FLOAT(IDEGR)*180.0/32767.0
: RETURN
: END
14: SUBROUTINE MOVE(X,Y)
: COMMON /BTEST/ ERROR,DEBUG,EOM,HNDSHK,
: &  COMPB(259),SUBSTR(11),MASTER,OPSEQ(5),
: &  LINTRM,BLKTRM,INBIT,OUTBIT,TRCOUN,BUFP,
: &  NAMBUF(23),FAULT
: INTEGER ERROR,DEBUG,EOM,HNDSHK,COMPB,
: &  SUBSTR,MASTER,OPSEQ,LINTTM,BLKTRM,
: &  TRCOUN,BUFP,FAULT,INBIT,OUTBIT,NAMBUF
15: INTEGER X,Y
16: CALL OPAIR(28,X,Y)
: RETURN
: END
```

Four COPY commands are executed in Example 1. The commands C 10,11,14 and C 10,11,17 tell the EDITOR to copy lines 10 and 11, inserting the copies before lines 14 and 17, respectively. Next the commands C 2,8,13 and C 2,8,15 tell the EDITOR to place a copy of lines 2 through 8 immediately before lines 13 and 15, respectively.

A second listing of the text shows that copies of the specified lines have been sent to the desired locations. The lines generated by the four COPY commands are the unnumbered lines in the new listing. (Newly created lines are not given edit line numbers until a RENUMBER command is executed.)

The new listing shows that the RETURN and END statements (lines 10 and 11) now appear in three places in the text, as do the COMMON and INTEGER statements (lines 2 through 8). Using the COPY command to generate the lines is quicker and simpler than entering 18 lines from the keyboard.

### Special Uses for the COPY Command

COMMON and INTEGER statements like those shown in Example 1 often appear in FORTRAN programs. COMMON and INTEGER statements can be many lines long, and the same statements are usually repeated many times in one program. Using the COPY command is the easiest way to duplicate groups of lines such as COMMON and INTEGER statements, which must appear in many locations.



## NOTES

## The DELETE Command

### Syntax Form:

D edit line number [ , [ edit line number ] ]

### Descriptive Form:

DELETE starting line number [ , [ ending line number ] ]

## PURPOSE

The DELETE command erases specified lines from the text buffer.

## EXAMPLES

DEL 5

DEL 50,60

DEL 0,10000

DEL 0,0+40

## EXPLANATION

The DELETE command allows a starting and ending line number to be specified. When the DELETE command is executed, all lines of text between and including the starting and ending lines are deleted from the text buffer. For instance, the example DEL 50,60 listed above deletes lines 50 through 60 from the text buffer. Just as for the other EDITOR commands, starting and ending line numbers may be expressed using an offset, as in the example DEL 0,0+40 listed above.

Since edit line number 0 always refers to the first line of text, and 10000 always refers to the last line of text, the command DEL 0,10000 listed above deletes the entire contents of the text buffer. This command is equivalent to the BASIC statement DEL ALL , which cannot be executed under EDITOR control.

## Editing Commands

### DELETE

#### Default Values

The DELETE command is the only EDITOR command that requires at least a starting line number to be specified. Entering only the keyword DEL causes a syntax error. This is for your protection, so that you cannot wipe out the entire text buffer by entering DEL and accidentally pressing RETURN.

The ending line number is optional. When the ending line number is omitted in a DELETE command, the EDITOR deletes only the starting line. This is also for your protection, to keep you from inadvertently deleting large portions of text.

#### Notes on the Command Syntax

As indicated by the brackets in the syntax form, the edit delimiter (,) may be omitted when the ending line is not specified. The meaning of the command is not changed: that is, both DEL 5 and DEL 5, delete only line 5 from the text buffer.

If a negative number is entered as the starting or ending line number, the EDITOR supplies the value 0. Thus the command DEL -3 deletes the first line of text.

#### Command Semantics

If the ending line number is larger than the starting line number, the EDITOR normally returns a semantic error. For example:

```
DEL 8,6
EDITOR ERROR
Semantic - error number 139
DEL 8,6
```

The command DEL 8,6 is syntactically correct. However, the meaning of the command is not clear, because the ending line precedes the starting line in the text buffer.

## An Editing Example

The following example shows the DELETE command being used to erase specified lines from a sample BASIC program.

### Example 1

```
LIST
550:REM ** END OF INITIALIZATION **
560:REM ** BEGINNING OF MAIN CODE **
570:REM ** ROW INTERCHANGE IF L<>K **
580:TRACE ON
590:FOR P=1 TO N+1
600:T=A(K,P)
610:A(K,P)=A(L,P)
620:A(L,P)=T
630:NEXT P
640:REM ** KEEP TRACK OF ROW PERMUTATIONS **
650:T=P1(K)
660:P1(K)=P1(L)
670:P1(L)=T
680:TRACE OFF
690:REM ** REDUCTION SUBROUTINE **

DEL 550,580

DEL 640

DEL 680,

LIST
590:FOR P=1 TO N+1
600:T=A(K,P)
610:A(K,P)=A(L,P)
620:A(L,P)=T
630:NEXT P
650:T=P1(K)
660:P1(K)=P1(L)
670:P1(L)=T
690:REM ** REDUCTION SUBROUTINE **
```

Three DELETE commands are executed in Example 1. The command DEL 550,580 deletes lines 550 through 580. The commands DEL 640 and DEL 680, delete line 640 and line 680, respectively. A new listing shows that the REM, TRACE ON, and TRACE OFF statements have been deleted.

## NOTES

## The INSERT Command

**Syntax Form:**

I [ edit line number ]

**Descriptive Form:**

INSERT [ destination for inserted lines of text ]

### PURPOSE

The INSERT command prepares the EDITOR for new lines to be entered from the keyboard and sent to a specified location in the text. Executing the INSERT command places the EDITOR in "insert" mode. A colon (:) appears on the display, signaling that the EDITOR is ready to receive new lines of text. After the new lines are entered, pressing the BREAK key returns the EDITOR to normal command mode.

### EXAMPLES

I

I 50

I +10

I 10000

I 10000-1

### EXPLANATION

The INSERT command is used to enter new lines from the keyboard and add them to the text buffer. The INSERT command has only one parameter, a destination line number. When an INSERT command is executed, the EDITOR is placed in "insert mode" and is ready to receive new text. Lines entered from the keyboard after the command is executed and before the BREAK key is pressed, are inserted in the text buffer immediately before the specified destination line.

## Editing Commands

### INSERT

For example, the command `I 50` listed above prepares the EDITOR to receive new text from the keyboard. Lines entered after the command is executed and before `BREAK` is pressed, are inserted immediately before line 50 in the current text. Similarly, the command `I +10` listed above prepares the EDITOR to receive new lines and insert them before the 11th line of text (line `+10` is the same as line `0+10`, the 11th line of text).

### Insert Mode

Executing an INSERT command places the EDITOR in insert mode. When the EDITOR is in insert mode, pressing `RETURN` causes a colon (`:`) to appear in the sixth character position of the next line on the display. The cursor appears one character position to the right of the colon.

The colon is the EDITOR's prompt character, and signals that you may begin entering new text from the keyboard. Characters entered immediately after the colon are part of a new line of text. Pressing the `RETURN` key ends the line. When `RETURN` is pressed, the new line is sent from the line buffer to the text buffer, where it is inserted just before the destination line specified in the INSERT command.

After `RETURN` is pressed, another colon appears on the display immediately below the previous one. This means that the EDITOR is ready to receive another new line. You may continue to create new lines in this manner, pressing `RETURN` to mark the end of a line and cause another colon prompt to appear on the display. Each time `RETURN` is pressed, the current line is inserted immediately before the destination line in the text buffer.

The `LINE EDITOR` keys `COMPRESS/EXPAND`, `RUBOUT--/BACKSPACE`, `RUBOUT--/SPACE`, and `REPRINT/CLEAR` function while the EDITOR is in insert mode. That is, if you make an error while entering a new line of text, you may correct the line using these keys, and you may backspace and type over incorrect characters.

The `RECALL NEXT LINE/RECALL LINE` and `STEP PROGRAM` keys do not function while the INSERT command is being used to create new lines of text.

### Removing the EDITOR from Insert Mode

When you are finished entering new lines, press the `BREAK` key to remove the EDITOR from insert mode. Pressing `BREAK` causes the cursor to reappear (without a colon) at the beginning of the next line on the display. All newly created lines have now been inserted in the text buffer, and the system is prepared to receive EDITOR commands.

When BREAK is pressed, the current line is sent to the text buffer, and placed before the destination line in the text buffer. Thus, although you can press RETURN before BREAK, you are not required to press RETURN after entering the last of the newly created lines. If the BREAK key is pressed immediately after the last character of the line is entered, the EDITOR automatically adds an end-of-record character to the end of the line.

Normally, pressing BREAK after the colon prompt appears removes the EDITOR from insert mode without adding a blank line after the inserted lines. However, if an INSERT command is executed and BREAK is pressed immediately after the colon appears for the first time, a blank line is inserted in the text buffer just before the specified destination line.

### The CLEAR Function

Pressing the REPRINT/CLEAR key while the EDITOR is in insert mode, clears the contents of the current line from the line buffer and the text buffer. A blank line remains in the text. (If the line was numbered, the edit line number remains in the text buffer.) The cursor reappears in the seventh character position of the next line on the display. New characters can be entered from the keyboard, and inserted into the empty line by pressing RETURN or BREAK. As always when the EDITOR is in insert mode, pressing RETURN causes another colon prompt to appear on the display, and pressing BREAK ends execution of the INSERT command.

### Default Values

The destination line may be omitted when entering the INSERT command. When no destination line is specified, newly created lines are inserted at the beginning of the current text. That is, the command `I` listed in the examples above places the EDITOR in insert mode, and tells the EDITOR to insert new lines before the first line of the current text.

The command `I` tells the EDITOR to place newly created lines of text before the beginning of the text. If you want to add lines onto the end of the text, you must specify a destination line that is beyond the end of the text. For instance, the command `I 10000` in the examples listed above, may be used to insert lines after the last line of text, because 10000 is always larger than the largest edit line number in the text. (See "How to Use Edit Line Numbers in a Command" in Section 2 for a more complete description.)

By contrast, the command `I 10000-1` inserts new lines just *before* the last line of the current text, because 10000-1 always refers to the last line of text.



## Editing Commands

### INSERT

#### An Editing Example

The following example shows the INSERT command being used to add new statements to a BASIC program.

#### Example 1

```
LIST
1:160 INPUT W
2:180 INPUT F$
3:200 INPUT N
4:220 FIND W
5:240 PRINT @33:"DEF FNF(X)=";F$
6:260 FIND W
7:280 APPEND 290

I
:150 PRINT "ENTER WORK FILE NUMBER:";

I2
:170 PRINT "ENTER FUNCTION, F(X):";

I3
:190 PRINT "ENTER # OF POINTS TO BE GRAPHED:";

I8
:290 REM *** USER FUNCTION APPENDED HERE ***
:300 PRINT "ENTER BEGINNING X VALUE:";
:310 INPUT X1

LIST
:150 PRINT "ENTER WORK FILE NUMBER:";
1:160 INPUT W
:170 PRINT "ENTER FUNCTION, F(X):";
2:180 INPUT F$
:190 PRINT "ENTER # OF POINTS TO BE GRAPHED:";
3:200 INPUT N
4:220 FIND W
5:240 PRINT @33:"DEF FNF(X)=";F$
6:260 FIND W
7:280 APPEND 290
:290 REM *** USER FUNCTION APPENDED HERE ***
:300 PRINT "ENTER BEGINNING X VALUE:";
:310 INPUT X1
```

Four INSERT commands are executed in Example 1. First the command `I` places the EDITOR in insert mode, and causes the colon prompt to appear on the next line of the display. The characters that appear on the same line and to the right of the colon are entered from the keyboard before pressing BREAK.

Next the command `I2` is executed, and another colon appears on the display. As before, characters are entered from the keyboard and BREAK is pressed to remove the EDITOR from insert mode. Then `I3` is executed, another new line entered, and BREAK pressed again.

Finally the command `I8` is executed. This time, the INSERT command is used to add three new lines to the text buffer. After the first and second lines are entered, RETURN is pressed, causing the colon prompt to reappear in the next line. After the third line is entered, however, BREAK is pressed to remove the EDITOR from insert mode.

After the INSERT commands are completed, a listing shows the six newly created lines in the desired locations in the text buffer. The line created using the command `I` appears at the beginning of the text; the lines created using the commands `I2` and `I3` appear before lines 2 and 3, respectively.

The three lines inserted using the command `I8` appear at the very end of the text. This is because edit line number 8 is larger than the largest line number in the current text, and therefore refers to a location beyond the end of the text.

All of the newly created lines of text remain unnumbered until a RENUMBER command is executed.

## NOTES

## The LIST Command

### Syntax Form:

```
L [ I/O address ] [ edit line number ] [ , [ edit line number ] ]
```

### Descriptive Form:

```
LIST [ I/O address ] [ starting line number ] [ , [ ending line number ] ]
```

## PURPOSE

The LIST command lists lines of text on the specified peripheral device. If a peripheral device is not specified, the list is printed on the display.

## EXAMPLES

```
LIS
```

```
LIS 50
```

```
LIS 50,
```

```
LIS ,50
```

```
LIS 400,500
```

```
LIS@29:
```

```
LIS@29:400,
```

```
LIS@29:,500
```

```
LIS@29:400,500
```

## EXPLANATION

The LIST command allows an I/O address, a starting line number, and an ending line number to be specified. When the LIST command is executed, edit line numbers and text between and including the starting and ending line are listed on the specified peripheral device. For instance, the example LIS@29:400,500 shown above causes lines 400 through 500 of the current text to be listed on peripheral device 29.

## Editing Commands

### LIST

When listing text on the display or any peripheral device, the EDITOR inserts a colon (:) before the first text character in each line. The colon is used to mark the beginning of each line and to separate edit line numbers from text.

Executing a LIST command does not change the current contents of the text buffer. When a copy of the text is transmitted to the display or another device, the EDITOR places a colon before the first text character in each line, and inserts an end-of-record character for each line of text. If the device is a magnetic tape, the EDITOR also places an end-of-file mark after the last line recorded on the tape.

A list of the current text can be sent to any device on the General Purpose Interface Bus by specifying the appropriate primary address in the LIST command. For example, the command `LIS@29:` sends a copy of the current edit line numbers and text to device 29 on the GPIB (General Purpose Interface Bus). A primary address is the only requirement for an I/O address: the EDITOR automatically issues secondary address 19, which tells the peripheral device that the incoming ASCII strings are lines of text to be listed.

Specifying a tape device in a LIST command when the tape head is positioned to the beginning of the open file, changes the file header name to `ASCII TEXT`.

### Default Values

All of the LIST command parameters are optional. When no I/O address is specified, edit line numbers and text are listed on the display. When the starting line number is omitted, the first line listed is the first line of text, and when the ending line number is omitted, the last line listed is the last line of text.

For instance, the command `LIS` shown above lists all of the current text on the display. `LIS 50,` lists from line 50 on, and `LIS,50` lists all lines up to and including line 50. Likewise, `LIS@29:400,` sends a list to device 29 of the current text from line 400 on, and `LIS@29:,500` lists on device 29 all text up to and including line 500.

The command `LIS 50` shown above, causes only line 50 to be listed on the display (see "Notes on the Command Syntax").

### Omitting the Keyword

Even the keyword LIST is optional, as long as at least one element of the syntax is specified. An I/O address, an edit line number, or an edit delimiter suffices when entering a LIST command.<sup>3</sup> For instance, the command `LIS 400,` may be shortened to `400,` and the command `LIS ,500` may be shortened to `,500`. Similarly, `LIS 400,500` is the same as `400,500`.

<sup>3</sup>Since a space is an edit delimiter, this means that pressing the SPACE bar, then RETURN, is the same as entering the keyword LIS and pressing RETURN. Both commands cause all of the current text to be listed on the display.

To list one line of text on the display, you need only enter the desired line number, then press RETURN. Thus the command 50 is equivalent to LIS 50 .

### Notes on the Command Syntax

As for many other EDITOR commands, the edit delimiter (,) may be omitted if the ending line number is omitted. However, omitting the edit delimiter changes the meaning of the command. That is, the commands LIS 50 and LIS 50, are not the same.

When a starting line is given alone and is not followed by a delimiter, the LIST command lists only one line of text, the starting line. Thus LIS 50, lists from line 50 through the end of the text, but LIS 50 lists only line 50 on the display.

**I/O Addresses.** When specifying a peripheral device, do not enter blank spaces within the I/O address, or immediately before or after the colon (:) that ends the I/O address. For example, the command LIS@29: ,500 causes an error, because a space appears immediately after the colon:

```
LIS@29: ,500
EDITOR ERROR
Syntax - error number 138
LIS@29: ,500
```

The correct way to enter the above command is LIS@29:,500 .

### Command Semantics

The ending line number specified in the LIST command should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command makes no sense, and has no effect on the text buffer. For example, the command LIS 5,1 has no effect on the current text. The correct way to enter the command is LIS 1,5 .

## Editing Commands

### LIST

### An Editing Example

The following example shows the LIST command being used to display lines of text on the screen.

#### Example 1

```
LIS
1:      DIMENSION A(20),B(20)
2:      L=0
3:      WRITE (6,10)
4:10    FORMAT (' ENTER DEGREE OF POLYNOMIAL:')
5:      READ (5,*) K
6:C     ENTER COEFFICIENTS OF POLYNOMIAL, P(X)
7:      WRITE (6,20)
8:20    FORMAT (' ENTER COEFFICIENTS OF X:')
9:      N=K+1
10:     DO 100 I=1,N
11:100  READ (5,*) A(N-I+1)
12:C   ENTER BEST GUESS APPROXIMATION FOR ROOT
13:    WRITE (6,30)
14:30   FORMAT (' ENTER APPROXIMATE ROOT:')
15:    READ (5,*) X0

LIS ,4
1:      DIMENSION A(20),B(20)
2:      L=0
3:      WRITE (6,10)
4:10    FORMAT (' ENTER DEGREE OF POLYNOMIAL:')

LIS 13,
13:    WRITE (6,30)
14:30   FORMAT (' ENTER APPROXIMATE ROOT:')
15:    READ (5,*) X0

LIS 7,11
7:      WRITE (6,20)
8:20    FORMAT (' ENTER COEFFICIENTS OF X:')
9:      N=K+1
10:     DO 100 I=1,N
11:100  READ (5,*) A(N-I+1)
```

In the example shown above, the command LIS causes the entire text to be displayed on the screen. Next, the command LIS ,4 tells the EDITOR to list all lines up to and including line 4. Then LIS 13, tells the EDITOR to list all lines from 13 on. Finally, the command LIS 7,11 causes lines 7 through 11 to be listed on the display.

## Listing Text on a Magnetic Tape Device

Before specifying a tape device in a LIST command, a file must be opened on the device by executing a FIND command. Once a file is open, the tape may be repositioned using SKIP or INPUT commands. (Refer to the SKIP and INPUT commands for detailed explanations.)

Whether the tape is positioned to the beginning of the file or to a particular logical record, the LIST command stores text on the tape beginning at the current position of the tape head. Any previously recorded information that lies beyond the tape head is lost.

After the LIST command is executed, the file remains open and available to Output operations. Subsequent Output commands overwrite the end-of-file mark left by the last command, and insert a new end-of-file mark when the operation is finished.

After executing a LIST command and before turning the system power off, it is advisable to close the file by executing a FIND command or pressing the RETURN TO BASIC overlay key. Closing the file ensures that all transmitted text reaches the tape.

**Format.** When text is listed on a magnetic tape file, edit line numbers stored by the LIST command become part of the text and no longer serve as edit line numbers. When the text is brought into the text buffer and listed on the display, the display format is as follows: each line on the display begins with a colon followed by a blank space. The next four character positions in the line contain the edit line number stored by the LIST command. One or more of these positions may be blank, depending on the number of digits in the line number, and all four positions are blank if the line was unnumbered. A colon follows, then text characters begin in the seventh position. For example:

```
LIST
:   1:CLIFFORD
:   2:ETHERIDGE
:   3:GLINES
:   4:KINTZ
:   5:LENZ
:   6:NOAKES
:   7:PARKER
```

**Error Messages.** If a LIST command attempts to output text to the internal magnetic tape and no file is open on the tape, a MT File error message appears on the display.

If a LIST command specifies a magnetic tape device and the file is not large enough to hold all of the text, a Device at EOF error occurs. When this happens, text is written on the tape file until the last byte before the physical end of the file is reached. The EDITOR places a logical end-of-file mark in the last byte of the file, and returns the Device at EOF error message. The remainder of the text transmitted by the LIST command is not stored on the tape.



## **Editing Commands**

### **LIST**

After a LIST command is executed, the file remains open to Output operations only. Attempting to execute an APPEND, OLD, INPUT, or SKIP command without reopening the file causes an error. A Device Access or Buffer Access error message appears on the display. Both messages mean that the specified device is not available for access by Input operations, and that the magnetic tape buffer cannot receive information from the device.

## The MOVE Command

### Syntax Form:

M [ edit line number ] [ , [ edit line number ] [ , [ edit line number ] ] ]

### Descriptive Form:

MOVE [ starting line number ] [ , [ ending line number ] [ , [ destination for moved text ] ] ]

## PURPOSE

The MOVE command moves specified lines of text to a new location in the text buffer.

## EXAMPLES

M 1,3,10

M 5,5,30

M 1,3,

M ,50,125

M ,100,

## EXPLANATION

The MOVE command allows a starting line number, an ending line number, and a destination line number to be specified. When the MOVE command is executed, all lines of text between and including the starting and ending lines are moved, and placed immediately before the specified destination line. For instance, the example M 1,3,10 listed above moves text lines 1 through 3, and inserts them immediately before line 10. The command M 5,5,30 moves line 5, and places the line immediately before line 30.

#### The Difference Between MOVE and COPY

The MOVE and COPY commands are alike in syntax and perform similar functions. Both commands send an already existing portion of text to a new location in the text buffer. The only difference between MOVE and COPY is in how the original lines are affected by the command. The COPY command leaves the original lines intact, sending a copy of the lines to the new location. The MOVE command, however, actually moves the original lines by rearranging the text buffer and placing the lines in the new location. The moved lines appear in the new location, but no longer appear in their former position.

#### Default Values

The starting, ending, and destination line numbers for the MOVE command are optional, and may be omitted or entered in any combination. Several examples are listed above.

When the starting line number is omitted, the first line relocated by the MOVE command is the first line of text. When the ending line number is omitted in the command, the last line moved is the last line of text. When the destination line is omitted, moved lines are placed after the last line of text.

In the examples listed above, the command `M 1,3` moves lines 1 through 3 to the end of the current text. The command `M ,50,125` moves all lines up to and including line 50, placing the moved text just before line 125. Finally, the command `M ,100` sends all lines up to and including line 100 to the end of the text.

#### Notes on the Command Syntax

Just as for the COPY command, final edit delimiters (,) are optional. When an edit delimiter is the last entry in the command, it may be omitted without changing the meaning of the command. That is, the commands `M 1,3`, and `M 1,3` are equivalent.

#### Command Semantics

Just as for the COPY command, the ending line number should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command makes no sense, and has no effect on the text buffer. For example, the command `M 7,1,10` has no effect on the current text. The correct way to enter the command is `M 1,7,10` .

A semantic error occurs if the specified destination line lies between the starting and ending lines. For example, the command `M 1,10,3` causes a semantic error, because the destination line (edit line number 3) falls between the starting and ending lines (edit lines 1 and 10).

### An Editing Example

The following example shows the MOVE command being used to move statements of a sample FORTRAN program.

#### Example 1

```

LIST
1:C  DEFINE FORTRAN FUNCTION DF(X)
2:    FUNCTION DF(X)
3:    DF = 3*X**2-2.946*X-5.738
4:    END
5:    RETURN
6:C  DEFINE FORTRAN FUNCTION F(X)
7:    FUNCTION F(X)
8:    F=X**3-1.473*X**2-5.738*X+6.763
9:    RETURN
10:   END
MOVE 6,10,1
LIST
:C  DEFINE FORTRAN FUNCTION F(X)
:  FUNCTION F(X)
:  F=X**3-1.473*X**2-5.738*X+6.763
:  RETURN
:  END
1:C  DEFINE FORTRAN FUNCTION DF(X)
2:  FUNCTION DF(X)
3:  DF = 3*X**2-2.946*X-5.738
4:  END
5:  RETURN
MOVE 5,5,4
LIST
:C  DEFINE FORTRAN FUNCTION F(X)
:  FUNCTION F(X)
:  F=X**3-1.473*X**2-5.738*X+6.763
:  RETURN
:  END
1:C  DEFINE FORTRAN FUNCTION DF(X)
2:  FUNCTION DF(X)
3:  DF = 3*X**2-2.946*X-5.738
:  RETURN
4:  END

```

Two MOVE commands are executed in Example 1. The command M 6,10,1 tells the EDITOR to move lines 6 through 10, inserting them just before line 1. After the command is executed, a new listing shows that the statements defining FUNCTION F(X), originally lines 6 through 10, have been moved to the beginning of the text. The lines relocated by the MOVE command are the unnumbered lines in the new listing. (Moved lines are not given edit line numbers until a RENUMBER command is executed.)

## Editing Commands

### MOVE

Next the command `M 5,5,4` tells the EDITOR to move line 5 and place it immediately before line 4. A new listing shows that the RETURN statement (originally line 5) has been moved, and now appears as an unnumbered line immediately before the END statement (line 4).

### The MOVE Command and BASIC Programs

After using the MOVE command to rearrange statements in a BASIC program, you must edit the program line numbers to be consistent with the changes. This is important if you plan to return to BASIC and execute the program, because BASIC's OLD command reorders statements according to program line numbers. In other words, the program line numbers must be in ascending order, or BASIC's OLD command will rearrange the statements again.

## The NLSEARCH (No List SEARCH) Command

### Syntax Forms:

```
NL [ [ edit line number ] , [ edit line number ] , ] b " string " { * , " string " }
```

### Descriptive Forms:

```
NLSEARCH [ [ starting line number ] , [ ending line number ] , ] space " target string "
{ * , " replacement string " }
```

## PURPOSE

The NLSEARCH command searches a portion of the current text for a specified "target" string. What happens when an occurrence of the string is found, depends on which form of the command is entered. If the target string is immediately followed by an asterisk ( \* ), lines found to contain the string are deleted from the text buffer. However, if a replacement string is specified after the target string, occurrences of the target string in the text are overwritten by the replacement string.

Because the two functions performed by NLSEARCH are different, each is treated as a separate command on the following pages. The two commands are named according to the function they perform: NLSEARCH and Delete Line, and NLSEARCH and Replace String.

## INTRODUCTION

### How Searching Occurs

When an NLSEARCH and Delete Line or NLSEARCH and Replace String command is executed, the EDITOR scans the current text from the starting line through the ending line for an occurrence of the specified "target" string. The first character scanned is the end-of-record character that immediately precedes the starting line, and the last character scanned is the end-of-record character that immediately follows the ending line.

**What Happens When an Occurrence of the Target String is Found**

As soon as an occurrence of the target string is found, the EDITOR performs the required function. That is, for an NLSEARCH and Delete Line command, the EDITOR deletes the line of text that contains the occurrence of the target string. For an NLSEARCH and Replace String command, the EDITOR overwrites the occurrence of the target string in the text with the "replacement" string specified by the command.

**Continuing the Search**

The search continues after the line containing the target string is deleted, or after the target string is overwritten by a replacement string. For an NLSEARCH and Delete Line command, the new search begins after the deleted line. That is, the first character scanned is the end-of-record character that precedes the next line of text.

For an NLSEARCH and Replace String command, the search resumes after the replacement procedure. The first character scanned is 1) the character that immediately follows the replacement string in the text, if the target string was shorter than the replacement string; or 2) the last character of the replacement string, if the target string was longer than the replacement string.

The EDITOR continues to search the text in the manner described above, deleting lines or replacing occurrences of the target string. When the end-of-record character that immediately follows the ending line is reached, the EDITOR stops searching and ends execution of the command.

## The NLSEARCH and Delete Line Command

### Syntax Form:

```
NL [ [ edit line number ] , [ edit line number ] , ] b " string " *
```

### Descriptive Form:

```
NLSEARCH [ [ starting line number ] , [ ending line number ] , ] space " target string " *
```

## PURPOSE

The NLSEARCH and Delete Line command searches a portion of the current text for a specified target string. Lines found to contain the string are deleted from the text buffer.

## EXAMPLES

```
NL "MOVED, LEFT NO ADDRESS"*
```

```
NL "INSUFFICIENT FUNDS"*
```

```
NL 100, , "U0"*
```

```
NL 200, , ":"*
```

```
NL ,5000 "MARGIN"*
```

```
NL 50,1250 "TRACE"*
```

```
NL 1000,5000 "REM"*
```

## EXPLANATION

The NLSEARCH and Delete Line command allows a starting line number, an ending line number, and an ASCII character string to be specified. The string is enclosed in quotation marks (or another string delimiter) and the last entry in the command is an asterisk ( \* ). When the NLSEARCH and Delete Line command is executed, the EDITOR searches the current text from the starting line through the ending line for the specified "target" string, and deletes lines found to contain the string.



## Editing Commands

### NLSEARCH and Delete Line

For example, the command `NL 1000,5000 "REM" *` searches lines 1000 through 5000 for occurrences of the target string `REM`. Lines found to contain the string `REM` are deleted from the text buffer. Similarly, the command `NL 50,1250 "TRACE" *` searches lines 50 through 1250, and deletes lines found to contain the string `TRACE`.

The `NLSEARCH` and Delete Line command is the same as the `SEARCH` and Delete Line command, except that no I/O address is specified in the command, and no list is made of the deleted lines.

### Default Values

When the starting line number is omitted in an `NLSEARCH` and Delete Line command, the EDITOR starts the search at the beginning of the first line of text. For example, the command `NL ,5000 "MARGIN" *` searches all lines up through 5000 and deletes lines found to contain the target string `MARGIN`. When the ending line number is omitted, the search ends after the last line of text. For example, the command `NL 200,, "V0" *` searches from the beginning of line 200 through the end of the text, and deletes line found to contain `V0`.

When both the starting and ending line numbers are omitted, the EDITOR searches the entire text. For example, the command `NL "INSUFFICIENT FUNDS" *` deletes all lines containing the string `INSUFFICIENT FUNDS`.

### Notes on the Command Syntax

When entering an `NLSEARCH` and Delete Line command, do not enter any blank spaces immediately before the asterisk (`*`). For example:

```
NL "X2" *  
EDITOR ERROR  
Syntax - error number 138  
NL "X2" *
```

This command causes a syntax error because a blank space appears between the second quotation mark (`"`) and the asterisk (`*`).

**Edit Delimiters in the Command.** The syntax and descriptive forms show three edit delimiters, represented by two commas (`,`) and a space (`␣`). The commas may be replaced by another edit delimiter (such as `.` or `:` or in some circumstances, a space<sup>4</sup>). The space (`␣`) shown in the syntax and descriptive forms is not meant to be replaced by another edit delimiter: for example, the command `NL,,,"REM" *` causes a syntax error.

<sup>4</sup>The second edit delimiter cannot be a space if the ending line number is omitted. This is because spaces immediately following another edit delimiter are ignored. For more information about edit delimiters, refer to "Edit Delimiters" in Section 2.

You can specify the edit delimiters in the manner indicated in the syntax and descriptive forms, or abbreviate the NLSEARCH and Delete Line command according to the following rules:

—If an ending line number is specified, you need only enter one edit delimiter between the ending line number and the target string. For example, the following commands are valid:

```
NL 1,3,"REM"*
```

```
NL 1,3 "REM"*
```

Several sample commands in this explanation follow the format of the last command shown above. The space immediately following the keyword NL is ignored by the EDITOR, and is entered only for the sake of appearance.

—When omitting both the starting line number and the ending line number, you may enter two non-blank edit delimiters between the keyword and the target string. For example:

```
NL , , "REM"*
```

Or, you may replace the two edit delimiters with a blank space:

```
NL "REM"*
```

This last command illustrates the simplest way to enter an NLSEARCH and Delete Line command without specifying edit line numbers. Several sample commands in this explanation are of this format.

**String Delimiters in the Command.** The target string of an NLSEARCH and Delete Line command must be enclosed in a string delimiter, or a syntax error occurs. A string delimiter can be any keyboard character that causes a printed character to appear on the display except 1) a space; 2) one of the characters in the target string; or 3) a number or an edit delimiter, if no edit line numbers are specified and a space appears in their place. For more detailed explanations, refer to "String Delimiters" in Section 2.

## Editing Commands

### NLSEARCH and Delete Line

#### Command Semantics

Omitting the asterisk ( \* ) at the end of an NLSEARCH and Delete Line command causes a semantic error. Replacing the asterisk with a comma or any other edit delimiter such as a space also causes a semantic error.

The EDITOR returns a semantic error in these two instances because the incorrect command resembles a SEARCH and List Line or SEARCH and Edit Line command. Since the NLSEARCH command makes no list of affected lines, it makes no sense to tell the EDITOR to perform a function similar to that of a SEARCH and List Line or SEARCH and Edit Line command. The EDITOR assumes you have misunderstood the function of the NLSEARCH command, and returns a semantic error.

The ending line number specified in an NLSEARCH and Delete Line command should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command either has no effect, or causes a semantic error as in the following example:

```
NL 10,2 "REM"*  
EDITOR ERROR  
Semantic - error number 139  
NL 10,2 "REM"*
```

The correct way to enter the command is as follows:

```
NL 2,10 "REM"*
```

#### Differences Between NLSEARCH and Delete Line and SEARCH and Delete Line

The NLSEARCH and Delete Line command is the same as the SEARCH and Delete Line command, except that no I/O address is specified in the command, and no list is made of the deleted lines.

### An Editing Example

The following example illustrates the use of the NLSEARCH and Delete Line command.

#### Example 1

```
LIST
1:340 REM ** BEGIN LOOP **
2:350 SET TRACE
3:360 U0=0
4:370 FOR K=1 TO 2↑N-1 STEP 2
5:380 X=A+K*H
6:390 S4=S4+FNF(X)
7:400 NEXT K
8:410 I2=H/3*(S0+2*S2+4*S4)
9:420 SET NORMAL
10:430 REM ** USE CORRECTION FACTOR **
11:440 I2=(16*I2-10)/15
12:450 U0=I2
13:460 PRINT
14:470 PRINT I2
15:480 IF ABS(I2-I0)<E THEN 540
16:490 I0=I2
17:500 NEXT N
18:510 REM ** PRINT FINAL VALUE **
19:520 PRINT
20:530 PRINT "FINAL VALUE IS";I2
```

```
NL "REM"*
NL 2,14 "U0"*
NL 12,, "PRINT]"*
NL ,10 "SET"*
```

```
LIST
4:370 FOR K=1 TO 2↑N-1 STEP 2
5:380 X=A+K*H
6:390 S4=S4+FNF(X)
7:400 NEXT K
8:410 I2=H/3*(S0+2*S2+4*S4)
11:440 I2=(16*I2-10)/15
14:470 PRINT I2
15:480 IF ABS(I2-I0)<E THEN 540
16:490 I0=I2
17:500 NEXT N
20:530 PRINT "FINAL VALUE IS";I2
```

## Editing Commands

### NLSEARCH and Delete Line

The initial listing in Example 1 shows that the text consists of twenty BASIC program statements. Then four NLSEARCH and Delete Line commands are executed.

The command `NL "REM"*` tells the EDITOR to delete all lines in the current text that contain the string `REM`. The command `NL 2,14 "V0"*` tells the EDITOR to search lines 2 through 14 and delete lines found to contain the string `V0`.

The command `NL 12,, "PRINT]"*` searches from line 12 through the end of the text for the word `PRINT` followed by an end-of-record character. (Refer to the explanation of the `] =` command.) When such a string is found, the line containing the word `PRINT` is deleted from the text buffer.

The command `NL ,10 "SET"*` tells the EDITOR to search through line 10 and delete lines containing the string `SET`.

After the four commands are executed, a new listing shows that the REMARK statements (BASIC program lines 340, 430, and 510) have been deleted, as well as statements containing the variable name `V0` (program lines 360 and 450), statements ending with the keyword `PRINT` (program line numbers 460 and 520), and statements containing `SET` (statements 350 and 420, `SET TRACE` and `SET NORMAL`).

## The NLSEARCH and Replace String Command

### Syntax Form:

```
NL [ [ edit line number ] , [ edit line number ] , ] b " string " , " string "
```

### Descriptive Form:

```
NLSEARCH [ [ starting line number ] , [ ending line number ] , ] space " target string " ,  
" replacement string "
```

## PURPOSE

The NLSEARCH and Replace String command searches a portion of the current text for a specified target string, and replaces occurrences of the string with the desired replacement string.

## EXAMPLES

```
NL "INPUT#1,"="INPUT@33:"  
NL "END #1","EOF(0)"  
NL 100,, "CHANGE A$ TO A","A=ASC(A$)"  
NL 45,, "CHANGE A TO A$","A$=CHR(A)"  
NL ,200 "RND","INT(21*RND+10)"  
NL ,5000 "MAT A=B*C","A=B MPY C"  
NL 45,200 "ON ERRDR","ON SIZE"  
NL 100,5000 "MAT ~=INU(~)"," ~~~~~"
```

## EXPLANATION

The NLSEARCH and Replace String command allows a starting line number, an ending line number, and two ASCII character strings to be specified. The first string is the "target" string and the second string is the "replacement" string. Both strings are enclosed in quotation marks or another string delimiter.

## Editing Commands

### NLSEARCH and Replace String

When the NLSEARCH and Replace String command is executed, the EDITOR searches the current text from the starting line through the ending line for the target string. Occurrences of the target string in the text are overwritten by the replacement string in the following manner: the first character of the replacement string overwrites the first character of the target string, the second character of the replacement string overwrites the second character of the target string, and so on.

For example, the command `NL 45,200 "END #1","EOF(0)"` tells the EDITOR to search lines 45 through 200 of the current text for the string `END #1`. Occurrences of this target string are overwritten by the replacement string `EOF(0)`.

The replacement string need not have the same number of characters as the target string. If the replacement string is longer than the target string, the NLSEARCH and Replace String command lengthens the text. For example:

```
LIST
  2:Dear Mr. Doe:
  3:It has come to our attention that you recently purchased
NL "Doe","Johnson"
```

```
LIST
  2:Dear Mr. Johnson:
  3:It has come to our attention that you recently purchased
```

In this example the command `NL "Doe","Johnson"` specifies a replacement string that has more characters than the target string. When the command is executed, the EDITOR locates an occurrence of the target string in line 2, and overwrites it with the replacement string. The characters `JOH` replace `DOE` and are immediately followed by `NSON`. A later listing shows that the length of line 2 has increased by four characters, and the word `JOHNSON` appears instead of `DOE`.

If the replacement string is shorter than the target string, the NLSEARCH and Replace String command shortens the text. For example:

```
LIST
  2:Dear Mr. McGrath:
  3:It has come to our attention that you recently purchased
NL "McGrath","Ank"
```

```
LIST
  2:Dear Mr. Ank:
  3:It has come to our attention that you recently purchased
```

In this example the command `NL "McGrath","Ank"` specifies a replacement string that has fewer characters than the target string. When the command is executed, the EDITOR locates an occurrence of the target string in line 2, and overwrites it with the replacement string. The characters `Ank` overwrite `McG`, and the remainder of the target string, `rath` is deleted from the text buffer. A later listing shows that the length of line 2 has decreased by four characters, and the word `Ank` appears instead of `McGrath`.

### **The Difference Between NLSEARCH and Replace String and SEARCH and Replace String**

The NLSEARCH and Replace String command is the same as the SEARCH and Replace String command, except that no I/O address is specified, and no list is made of the changed lines.

### **Default Values**

When the starting line number is omitted in an NLSEARCH and Replace String command, the EDITOR starts the search at the beginning of the first line of text. For example, the command `NL ,5000 "MAT A=B * C","A=B MPY C"` searches all lines up through 5000, and replaces occurrences of the string `MAT A=B * C` with `A=B MPY C`.

When the ending line is omitted, the search ends after the last line of text. For example, the command `NL 45, "CHANGE A TO A$","A$=CHR(A)"` searches from the beginning of line 45 through the end of the text, and overwrites occurrences of `CHANGE A TO A$` with `A$=CHR(A)`.

When both the starting and ending line numbers are omitted, the EDITOR searches the entire text. For example, the command `NL "END #1","EOF(0)"` searches the entire text, and replaces all occurrences of `END#1` with `EOF(0)`.

### **Notes on the Command Syntax**

When entering an NLSEARCH and Replace String command, use only one edit delimiter between the target and replacement strings. Entering more than one delimiter causes a syntax error. For example:

```
NL 1,100 "PRINT" , "PRINT @33:"  
EDITOR ERROR  
Syntax - error number 138  
NL 1,100 "PRINT" , "PRINT @33:"
```

The correct way to enter the command is as follows:

```
NL 1,100 "PRINT","PRINT @33:"
```



## Editing Commands

### NLSEARCH and Replace String

**Edit Delimiters in the Command.** Just as for other searching commands, the syntax and descriptive forms show three edit delimiters, represented by two commas (,) and a space (b). The two commas may be replaced by another edit delimiter, with the following exception: the second comma cannot be replaced by a space if the ending line number is omitted, or a syntax error occurs. This is because the EDITOR ignores spaces that immediately follow another edit delimiter. (For more information, refer to "Edit Delimiters" in Section 2.)

The space (b) in the syntax and descriptive forms is not meant to be replaced by another edit delimiter. For example, the command `NL,,,"@33","@29"` causes a syntax error.

Just as for the other searching commands, you can specify the edit delimiters in the manner indicated in the syntax and descriptive forms, or abbreviate the NLSEARCH and Replace String command according to these rules:

—If an ending line number is specified, you need only enter one edit delimiter between the ending line number and the target string. For example, the commands `NL 1,3,"S1","S2"` and `NL 1,3 "S1","S2"` are valid.

Several of the sample commands in this explanation are of the format of the last command. The space immediately following the keyword `NL` is ignored by the EDITOR, and is entered only for the sake of appearance.

—When omitting both the starting and ending line numbers, you may enter two non-blank edit delimiters between the keyword `NL` and the target string, as in the command `NL,,"S1","S2"`. Or, you may replace the two edit delimiters with a blank space, as in `NL "S1","S2"`. The format of this last command is more convenient, and is emphasized in the examples.

**String Delimiters in the Command.** As for all searching commands, the target and replacement strings must be enclosed in a string delimiter, or a syntax error occurs. The delimiter used to enclose the target string need not be the same as the delimiter used to enclose the replacement string.

A string delimiter can be any keyboard character that causes a printed character to appear on the display except 1) a space; 2) one of the characters in the target or replacement string; or 3) a number or edit delimiter, if no edit line numbers are specified and a space appears in their place. For more information, refer to "String Delimiters" in Section 2.

### Command Semantics

The ending line number specified in an NLSEARCH and Replace String command should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command either has no effect, or causes a semantic error.

## Special Uses for the NLSEARCH and Replace String Command:

### Deleting Specified Strings

If the replacement string is empty (has no characters), the NLSEARCH and Replace String command deletes occurrences of the target string from the text buffer. This is not the same as deleting the *line* in which the target string occurs (the function provided by the NLSEARCH and Delete Line and SEARCH and Delete Line commands). For example:

```
NL "MAT", ""
```

This command locates and deletes all occurrences of the string `MAT` in the current text. The target string is empty—no characters are enclosed in the quotation marks used as the string delimiter. When the command is executed, each occurrence of the string `MAT` in the text is replaced by a "null" string, and disappears from the text. For every occurrence of `MAT` deleted from the text buffer, the text is three characters shorter. For an illustration of how this command acts on a sample piece of text, see "An Editing Example" in this explanation.

#### NOTE

*An empty replacement string may be specified in an NLSEARCH and Replace String or SEARCH and Replace String command. However, specifying an empty target string in any NLSEARCH or SEARCH command makes no sense, and causes a semantic error.*

## An Editing Example

The following example illustrates the use of the NLSEARCH and Replace String command.

## Example 1

```

LIST
1:300 IF END #1 THEN 1500
2:310 REM ** INITIALIZATION **
3:320 DIM A(S1,S2),B(S3,S4),Q(X1,X1),R(X2,X2)
4:330 DIM S(X3,X3),T(X1,X1),U(X2,X2),V(X3,X3)
5:340 MAT A=ZER
6:350 MAT B=ZER
7:360 MAT C=ZER(10,10)
8:370 REM ** USER ENTERS THE THREE MATRICES **
9:380 PRINT "ENTER FIRST MATRIX:"
10:390 MAT INPUT Q
11:400 PRINT "ENTER SECOND MATRIX:"
12:410 MAT INPUT R
13:420 PRINT "ENTER THIRD MATRIX:"
14:430 MAT INPUT S
15:440 REM ** INVERT THE MATRICES **
16:450 MAT T=INU(Q)
17:460 MAT U=INU(R)
18:470 MAT V=INU(S)
19:480 MAT C=ZER(X1,X1)
20:490 PRINT "ENTER A 10 BY 10 MATRIX:"
21:500 MAT INPUT Z(10,10)

NL ,10 "IF END ~~","ON EOF(0)"
NL 21,, "MAT INPUT Z(10,10)","DIM Z(10,10)]510 INPUT Z"
NL 7,7 "MAT C=ZER(10,10)","DIM C(10,10)]365 C=0"
NL "MAT ",""
NL "ZER","0"

```

(continued on next page)

```

LIST
1:300 ON EOF(0) THEN 1500
2:310 REM ** INITIALIZATION **
3:320 DIM A(S1,S2),B(S3,S4),Q(X1,X1),R(X2,X2)
4:330 DIM S(X3,X3),T(X1,X1),U(X2,X2),V(X3,X3)
5:340 A=0
6:350 B=0
7:360 DIM C(10,10)
   :365 C=0
8:370 REM ** USER ENTERS THE THREE MATRICES **
9:380 PRINT "ENTER FIRST MATRIX:"
10:390 INPUT Q
11:400 PRINT "ENTER SECOND MATRIX:"
12:410 INPUT R
13:420 PRINT "ENTER THIRD MATRIX:"
14:430 INPUT S
15:440 REM ** INVERT THE MATRICES **
16:450 T=INV(Q)
17:460 U=INV(R)
18:470 V=INV(S)
19:480 C=0(X1,X1)
20:490 PRINT "ENTER A 10 BY 10 MATRIX:"
21:500 DIM Z(10,10)
   :510 INPUT Z

```

Example 1 shows the NLSEARCH and Replace String command being used to make a "foreign" BASIC program compatible with TEKTRONIX 4051 BASIC. An initial listing shows that the text consists of 21 BASIC program statements. Then five NLSEARCH and Replace String commands are executed.

The first NLSEARCH and Replace String command tells the EDITOR to search the current text up through line 10 for a string consisting of IF END immediately followed by a space, immediately followed by any two ASCII characters. The two unspecified ASCII characters are represented in the target string by the symbol ~, the wildcard character by default. (For an explanation of what the wildcard character means when used in a target or replacement string, refer to the ~= command in Section 5.) The command also specifies that occurrences of the target string in the text are to be overwritten by the replacement string ON EOF(0) .

The second NLSEARCH and Replace String command tells the EDITOR to search the current text from line 21 on for occurrences of the string MAT INPUT Z(10,10) . Occurrences of the string are to be overwritten by a string consisting of DIM Z(10,10) immediately followed by the beginning of a new line and the string 510 INPUT Z . The beginning of the new line is represented in the replacement string by the symbol ] , the current "END-OF-RECORD" character by default. (For an explanation of what the "END-OF-RECORD" character means when used in a target or replacement string, refer to the ]= command in Section 5.)

## Editing Commands

### NLSEARCH and Replace String

The third NLSEARCH and Replace String command is similar to the second, but specifies that the EDITOR is to search only line 7: 7 is both the starting and ending line number. The command tells the EDITOR to overwrite occurrences of the string `MAT C=ZER(10,10)` in line 7 with the string `DIM C(10,10)` followed by the beginning of a new line and the string `365 C=0`. As in the last command, the beginning of the new line is represented in the replacement string by the symbol `]` .

The fourth NLSEARCH and Replace String command tells the EDITOR to search the entire text for the string `MAT`. Occurrences of `MAT` are to be replaced by an empty string. This is the same as telling the EDITOR to delete all occurrences of the string `MAT`.

Finally, the fifth NLSEARCH and Replace String command specifies that all occurrences of `ZER` in the program are to be replaced by the character `0`.

After the NLSEARCH and Replace String commands are executed, a new listing is made to show the changed text. The statement `300 IF END #1 THEN 1500` is now `300 ON EOF(0) THEN 1500`. The statement `500 MAT INPUT Z(10,10)` is replaced by `500 DIM Z(10,10)` and immediately followed by the new statement `510 INPUT Z`. The statement `360 MAT C=ZER(10,10)` is replaced by `360 DIM C(10,10)` and immediately followed by the new statement `510 INPUT Z`.

All occurrences of the string `MAT` have been deleted from the text. For example, the statement `450 MAT T=INV(Q)` has been changed to `450 T=INV(Q)`. Occurrences of `ZER` have been overwritten by `0`: for example, statement 340 is now `A=0`.

## The SEARCH Command

### Syntax Forms:

S [ I/O address ] [ [ edit line number ] , [ edit line number ] , ] b " string " [ { \* , [" string " ] } ]

### Descriptive Forms:

SEARCH [ I/O address ] [ [ starting line number ] , [ ending line number ] , ] space " target string " [ { \* , [" replacement string " ] } ]

## PURPOSE

The SEARCH command searches a portion of the current text for lines containing a specified "target" string. What happens when an occurrence of the string is found, depends on which form of the command is entered. If the command is entered with only the keyword SEARCH and a target string, lines containing the string are listed on a peripheral device. But if an asterisk immediately follows the target string, lines found to contain the string are listed on the device, then deleted from the text buffer.

If the target string is followed by an edit delimiter such as a comma, lines of text found to contain the string are recalled one by one to the line buffer, and wait to be edited from the keyboard. And finally, if the command specifies a replacement string after the target string, occurrences of the target string in the text are overwritten by the replacement string, and changed lines are listed on a peripheral device.

Because such different functions are performed by the four variations of the SEARCH command, each one is treated as a separate command on the following pages. The SEARCH commands are named according to the function they perform: SEARCH and List Line, SEARCH and Edit Line, SEARCH and Delete Line, and SEARCH and Replace String.

## **INTRODUCTION**

### **How Searching Occurs**

When a SEARCH command is executed, the EDITOR scans the text for the target string in the same manner as for the NLSEARCH command. That is, the search begins with the end-of-record character that precedes the starting line, and ends with the end-of-record character that follows the ending line. Upon finding an occurrence of the target string, the EDITOR performs the required function: listing, deleting, or recalling to the line buffer the line that contains the target string (SEARCH and List Line, SEARCH and Edit Line, SEARCH and Delete Line); or overwriting an occurrence of the target string in the text with a replacement string (SEARCH and Replace String).

The search then continues, starting with the end-of-record character that precedes the next line of text (SEARCH and Edit Line, SEARCH and Delete Line) or with the next text character after the target string (SEARCH and List Line). Like the NLSEARCH and Replace String command, the SEARCH and Replace String command resumes the search with 1) the character that immediately follows the replacement string in the text, if the target string was shorter than the replacement string; or 2) the last character of the replacement string, if the target string was longer than the replacement string.

### **The SEARCH Command as an Input/Output Command**

SEARCH and List Line, SEARCH and Delete Line, and SEARCH and Replace String are Input/Output commands. Information concerning I/O addresses, and the syntax and semantics of EDITOR Input/Output commands is provided at the beginning of Section 6. Other important information is summarized below.

Before specifying a magnetic tape device in a SEARCH command, a file on the chosen peripheral device must be opened by executing a FIND command. If no file is open on the specified device, executing a SEARCH command causes a MT File error. After executing a FIND command, you may position the tape to a particular logical record within the file by executing a SKIP or INPUT command. Whether the tape is positioned to the beginning of the file using the FIND command or to a particular logical record using FIND and then SKIP or INPUT, the SEARCH command stores text on the tape beginning at the current position of the tape head. Any previously recorded information that lies beyond the tape head, is lost.

After a SEARCH command is executed, the file remains open for access by Output commands. That is, after executing a SEARCH command, you may execute any EDITOR Output command without reopening the file. However, before turning the system power off or removing the internal magnetic tape unit, you should close the file by executing a FIND command that specifies a different file number. Closing the file in this way is important to ensure that all transmitted text reaches the tape.

Executing a SEARCH command when the tape is positioned to the beginning of a file changes the file header name to ASCII TEXT (if this has not been previously done).

**A Special Note About Syntax.** When specifying an I/O address in a SEARCH command, do not enter any blank spaces between the I/O address and the target string, or a syntax error occurs. That is, if you specify an I/O address, you must omit the space (b) that appears in the syntax and descriptive forms. This is also mentioned in the explanations of the individual SEARCH commands on the following pages.



## NOTES

## The SEARCH and List Line Command

### Syntax Form:

```
S [ I/O address ] [ [ edit line number ] , [ edit line number ] , ] b " string "
```

### Descriptive Form:

```
SEARCH [ I/O address ] [ [ starting line number ] , [ ending line number ] , ] space " target string "
```

## PURPOSE

The SEARCH and List Line command searches a portion of the text for a specified target string, and lists on a peripheral device all lines found to contain the string. If no peripheral device is specified, lines containing the target string are listed on the display.

## EXAMPLES

```
S "ON ~ GO TO"  
S 100,, "DO"  
S ,5000 "CONTINUE"  
S 100,5000 "X1"  
S@33:"JONES"  
S@20:100,, "EOF"  
S@33:,5000 "NEW ADDRESS"  
S@29:100,5000 "638-"
```

## EXPLANATION

The SEARCH and List Line command allows an I/O address, a starting line number, an ending line number, and an ASCII character string to be specified. The string must be enclosed in quotation marks (or another string delimiter). When the SEARCH and List Line command is executed, the EDITOR searches the current text from the starting line through the ending line for the specified "target" string. Lines found to contain the string are listed on the specified peripheral device. The contents of the text buffer are not changed by the command.

## Editing Commands

### SEARCH and List Line

For example, the command `S@33:100,5000 "638-"` searches lines 100 through 5000 of the current text for occurrences of the target string `638-`. Lines found to contain the string `638-` are listed on the file that is currently open on device 33 (the internal magnetic tape).

If a line contains more than one occurrence of the target string, the line is listed more than once. For example, a line containing three occurrences of the target string is listed three times on the specified peripheral device.

### Default Values

When no I/O address is specified in a SEARCH and List Line command, lines found to contain the target string are listed on the system display by default. For example, the command `S 100,5000 "X1"` searches lines 100 through 5000 for the string `X1`. Lines found to contain `X1` are listed on the display.

When the starting line number is omitted in a SEARCH and List Line command, the EDITOR starts the search at the beginning of the first line of text. For example, the command `S ,5000 "NEW ADDRESS"` searches all lines up through 5000 for the string `NEW ADDRESS`, and lists lines found to contain the string on the system display.

When the ending line number is omitted, the search ends after the last line of text. For example, the command `S 100,, "DO"` searches from the beginning of line 100 through the end of the text, and lists on the display the lines found to contain the string `DO`.

When both the starting and ending line numbers are omitted, the EDITOR searches the entire text. For example, the command `S "CONTINUE"` lists on the system display all lines in the current text that contain the string `CONTINUE`.

### Notes on the Command Syntax

When entering a SEARCH and List Line command, do not enter any blank spaces immediately after the I/O address. For example, the following command causes a syntax error:

```
S@33: 100,500 "CONTINUE"
```

The correct way to enter the command is as follows:

```
S@33:100,500 "CONTINUE"
```

**Edit Delimiters in the Command.** Just as for other searching commands, the syntax and descriptive forms show three edit delimiters, represented by two commas (,) and a space (b). The two commas may be replaced by another edit delimiter, with the following exception: the second comma cannot be replaced by a space if the ending line number is omitted, or a syntax error occurs. This is because the EDITOR ignores spaces that immediately follow another edit delimiter. (For more information, refer to "Edit Delimiters" in Section 2.)

The space (b) in the syntax and descriptive forms is not meant to be replaced by another edit delimiter. For example, the command S,,,"GO TO" causes a syntax error.

Just as for the other searching commands, you can specify the edit delimiters in the manner indicated in the syntax and descriptive forms, or abbreviate the SEARCH and List Line command according to these rules:

—If an ending line number is specified, you need only enter one edit delimiter between the ending line number and the target string. For example, the commands S 100,500,"EOF" and S 100,500 "EOF" are valid.

Several of the sample commands in this explanation are of the format of this last command. The space immediately following the keyword S is ignored by the EDITOR, and is entered only for the sake of appearance.

—When omitting both the starting and ending line numbers, you may enter two non-blank edit delimiters between the keyword S and the target string, as in the command S,,,"EOF" . Or, you may replace the two edit delimiters with a blank space, as in S "EOF" . The format of this last command is more convenient, and is emphasized in the examples.

**String Delimiters in the Command.** As for all searching commands, the target string must be enclosed in a string delimiter, or a syntax error occurs. The string delimiter can be any keyboard character that causes a printed character to appear on the display except 1) a space; 2) one of the characters in the target or replacement string; or 3) a number or edit delimiter, if no edit line numbers are specified and a space appears in their place. For more information, refer to "String Delimiters" in Section 2.

## Editing Commands

### SEARCH and List Line

#### Command Semantics

When specifying a peripheral device in a SEARCH and List Line command, do not enter any blank spaces immediately before the colon (:), or a semantic error occurs.<sup>5</sup> For example:

```
S033 : "GO"  
EDITOR ERROR  
Semantic - error number 139  
S033 : "GO"
```

The correct way to enter the command is as follows:

```
S033: "GO"
```

The ending line number specified in a SEARCH and List Line command should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command either has no effect, or causes a semantic error as in the following example:

```
S 10,2 "REM"  
EDITOR ERROR  
Semantic - error number 139  
S 10,2 "REM"
```

The correct way to enter the command is as follows:

```
S 2,10 "REM"
```

#### NOTE

*When entering a SEARCH and List Line command, do not press the SPACE bar immediately before RETURN. If the final entry is a space, the EDITOR interprets the command to be a SEARCH and Edit Line command, not a SEARCH and List Line command. This is because a space is an edit delimiter. (Refer to the explanation of the SEARCH and Edit Line command.)*

<sup>5</sup>This is true for all EDITOR Input/Output commands.

### An Editing Example

In the following example, the SEARCH and List Line command is used to locate and list lines containing a specified target string. The initial listing shows twenty-six fictitious names and telephone numbers. The command S "638-" tells the EDITOR to locate and list all lines containing the string 638- . When the command is executed, the EDITOR lists on the display five lines found to contain telephone numbers that begin with the prefix 638- .

#### Example 1

```
LIST
:Evans, John           638-2582
:Atkins, Bruce        648-8754
:Carney, Jack L       644-0971
:Deitz, Lyle          643-6993
:Gatman, Stanley      357-6785
:Herriott, Thomas W   638-1134
:Estes, Marlene       288-8148
:Augee, Carol         638-3639
:Karp, J Kenneth      646-4792
:Mill, Roger H        649-1695
:Carlson, E W         638-7251
:Donner, Beatrice     641-9364
:Miller, B K          639-4029
:Nelson, Elroy C      644-1114
:Otey, P              620-1355
:Smith, T L           644-0588
:Overby, Pierre       357-5122
:Potter, William T    638-5150
:Robinson C H         643-5640
:Sherby, Kevin        627-6091
:Shea, Francine       635-0683
:Snow, Daniel K       620-4144
:Thompson, Isabel    649-1768
:Trujillo, Leona     646-7891
:Wetzel, Maude        357-4308

S "638-"
:Evans, John           638-2582
:Herriott, Thomas W   638-1134
:Augee, Carol         638-3639
:Carlson, E W         638-7251
:Potter, William T    638-5150
```

## NOTES

## The SEARCH and Edit Line Command

### Syntax Form:

```
S [ [ edit line number ] , [ edit line number ] , ] b " string " ,
```

### Descriptive Form:

```
SEARCH [ [ starting line number ] , [ ending line number ] , ] space " target string " ,
```

## PURPOSE

The SEARCH and Edit Line command searches a portion of the text for a specified target string. One by one, lines found to contain the string are recalled to the line buffer for editing. After a line containing the string is recalled and edited, the BREAK key is pressed to return the corrected line to the text buffer and to tell the EDITOR to begin searching for the next occurrence of the target string.

## EXAMPLES

```
S "IF BREAK THEN",  
S "MAT ~=ZER",  
S 100,, "CALL WARN",  
S 50,, "DATA",  
S ,5000 "PRINT USING",  
S ,5000 "MAT",  
S 100,5000 "MR.",  
S 1,2000 "NUM",  
S 0,100 "'",
```



## Editing Commands

### SEARCH and Edit Line

#### EXPLANATION

The SEARCH and Edit Line command allows a starting line number, an ending line number, and an ASCII character string to be specified. The string must be enclosed in quotation marks or another string delimiter, and the last entry in the command is an edit delimiter (a comma or one of the other five edit delimiters). When the SEARCH and Edit Line command is executed, the EDITOR searches the current text from the starting line through the ending line for the specified "target" string. One by one, lines found to contain the string are listed on the display, and recalled to the line buffer for editing.<sup>6</sup>

For example, the command S 100,5000 "NUM", searches lines 100 through 5000 of the current text for occurrences of the target string NUM . Lines found to contain the string NUM are listed on the display, and recalled to the line buffer for editing. Lines containing more than one occurrence of the target string are listed and recalled to the line buffer only once.

Lines containing the target string are recalled to the line buffer in the following manner: executing the SEARCH and Edit Line command by pressing RETURN causes the line that contains the first occurrence of the target string to appear on the display. The cursor is positioned over the last character of the target string. For example:

```
LIST
  1:370 REM ** USER ENTERS THE THREE MATRICES **
  2:380 PRINT "ENTER FIRST MATRIX:"
  3:390 INPUT Q
  4:400 PRINT "ENTER SECOND MATRIX:"
  5:410 INPUT R
  6:420 PRINT "ENTER THIRD MATRIX:"
  7:430 INPUT S

S 4,7 "PRINT",
  4:400 PRINT "ENTER SECOND MATRIX:"
```

The line may now be edited using the keyboard and LINE EDITOR keys. (Refer to Section 3 for information about how the system keyboard operates under EDITOR control.)

After the line is edited, pressing the BREAK key returns the corrected line to the text buffer. At the same time, the line containing the next occurrence of the target string is recalled to the line buffer, and listed on the display with the cursor positioned over the last character of the target string. The line is now ready for keyboard editing.

<sup>6</sup>No I/O address is specified in the SEARCH and Edit Line command. Recalled lines are always listed on the system display.

You may continue to edit lines in this manner, pressing BREAK each time to return a corrected line to the text buffer and signal the EDITOR to list and recall the next line that contains an occurrence of the target string. After the last line containing the target string is listed on the display and recalled to the line buffer, pressing BREAK ends execution of the SEARCH and Edit Line command, and causes the cursor to reappear at the beginning of the next line on the display.

### **The RETURN and REPRINT/CLEAR Keys**

Pressing RETURN during execution of the SEARCH and Edit Line command causes a colon prompt (:) to appear on the display. After the colon appears you may create a new line of text by entering characters from the keyboard, then pressing RETURN or BREAK. Pressing RETURN or BREAK inserts the new line into the text immediately following the last line recalled by the SEARCH and Edit Line command. Pressing RETURN also causes a new colon prompt to appear on the display; and pressing BREAK signals the EDITOR to search for the next occurrence of the target string.

While the line buffer contains a line recalled by the SEARCH and Edit Line command, pressing the REPRINT/CLEAR key without pressing the SHIFT key clears the line from the line buffer and the text buffer. A blank line remains in the text, and the cursor reappears in the seventh character position of the next line on the display. New characters can be entered from the keyboard, and inserted into the empty line by pressing RETURN or the BREAK key. As always during the execution of the SEARCH and Edit Line command, pressing RETURN causes a colon prompt to appear on the display, and pressing BREAK signals the EDITOR to search for the next occurrence of the target string.

### **The RECALL NEXT LINE/RECALL LINE and STEP PROGRAM Keys**

The RECALL NEXT LINE/RECALL LINE and STEP PROGRAM keys function during the execution of the SEARCH and Edit Line command. This allows you to edit text that precedes or follows lines containing the specified target string. While the line buffer contains a line recalled by the SEARCH and Edit Line command, the RECALL NEXT LINE and RECALL LINE functions return the recalled line to the text buffer. At the same time, the next line of text is recalled to the line buffer and listed on the display. Similarly, pressing the STEP PROGRAM key returns the current line to the text buffer, and recalls the *preceding* line to the line buffer.

### **Default Values**

When the starting line number is omitted in a SEARCH and Edit Line command, the EDITOR starts the search at the beginning of the first line of text. For example, the command S ,500 "PRINT USING", searches all lines up through 500 for the string PRINT USING . Lines found to contain PRINT USING are listed on the system display and recalled to the line buffer for editing.

## Editing Commands

### SEARCH and Edit Line

When the ending line number is omitted, the search ends after the last line of text. For example, the command `S 100,, "CALL WARN"`, searches from the beginning of line 100 through the end of the text for the string `CALL WARN`. Lines found to contain `CALL WARN` are listed on the display, and recalled to the line buffer for editing.

When both line numbers are omitted, the EDITOR searches the entire text. For example, the command `S "IF BREAK THEN"`, lists and recalls all lines in the current text that contain the string `IF BREAK THEN`.

### Notes on the Command Syntax

The last entry in the SEARCH and Edit Line command can be any edit delimiter: a comma, a colon, a semicolon, an equals sign, or a space. If the edit delimiter is *not* a space, do not enter blank spaces immediately before the edit delimiter. For example:

```
S "DATA" ,  
EDITOR ERROR  
Syntax - error number 138  
S "DATA" ,
```

The correct way to enter the command is as follows:

```
S "DATA",
```

**Edit Delimiters in the Command.** Just as for other searching commands, the two commas in the syntax and descriptive forms may be replaced by another edit delimiter, with the following exception: the second comma cannot be replaced by a space if the ending line number is omitted, or a syntax error occurs. Also, the space (Ø) in the syntax and descriptive forms is not meant to be replaced by another edit delimiter.

Just as for the other searching commands, you can specify the edit delimiters in the manner indicated in the syntax and descriptive forms, or abbreviate the SEARCH and Edit Line command as in the following examples:

```
S 1,10,"MAT",  
S 1,10 "MAT",  
S,,"NUM",  
S "NUM",
```

As always, spaces immediately following the keyword are not required, and are ignored by the EDITOR.

**String Delimiters in the Command.** As for all searching commands, the target string must be enclosed in a string delimiter, or a syntax error occurs. The string delimiter can be any keyboard character that causes a printed character to appear on the display except 1) a space; 2) one of the characters in the target or replacement string; or 3) a number or edit delimiter, if no edit line numbers are specified and a space appears in their place. For more information, refer to "String Delimiters" in Section 2.

### Command Semantics

The ending line number specified in a SEARCH and Edit Line command should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command either has no effect, or causes a semantic error.

### An Editing Example

The following example illustrates the use of the SEARCH and Edit Line command.

```

Example 1   LIST
              1:630 PRINT@33:"A=";A
              2:640 TRACE ON
              3:650 FOR J=1 TO N-1
              4:660 FOR I=J+1 TO N
              5:670 A(I,N+1)=A(I,J)*A(J,N+1)
              6:680 NEXT I
              7:690 NEXT J
              8:700 TRACE OFF
              9:710 PRINT@33:"A=";A
             10:730 DIM B(M,M),C(M,M)
             11:740 MAT C=CON / ALL THE ELEMENTS OF C ARE 1
             12:750 MAT B=IDN / B IS THE N×N IDENTITY MATRIX
             13:770 MAT A=INV(A)

S '^",
  11:740 MAT C=CON  █ ███ ███ ██████████ ███ ███ ███
  12:750 MAT B=IDN  █ ███ ███ ██████████ ██████████

S 11,, "MAT ",
  11:740 ███ C=███
      740 C=1
  12:750 █ALL█=IDN",B
  13:770 ███ A=INV(A)
      770 A=INV(A)

S ,10 "PRINT",
  1:630 PRINT█████"A=";A
      630 PRINT "A=";A
  9:710 PRINT█████"A=";A
      710 PRINT "A=";A
  
```

(continued on next page)

## Editing Commands SEARCH and Edit Line

```
S 2,9 "TRACE",
2:640 SRERASE
8:700 SRNORMAL

LIST
1:630 PRINT "A=";A
2:640 SET TRACE
3:650 FOR J=1 TO N-1
4:660 FOR I=J+1 TO N
5:670 A(I,N+1)=A(I,J)*A(J,N+1)
6:680 NEXT I
7:690 NEXT J
8:700 SET NORMAL
9:710 PRINT "A=";A
10:730 DIM B(M,M),C(M,M)
11:740 C=1
12:750 CALL "IDN",B
13:770 A=INV(A)
```

The initial listing in Example 1 shows that the text consists of a portion of a BASIC program. Then four SEARCH and Edit Line commands are executed. The first command `S ""`, tells the EDITOR to list and recall to the line buffer all lines containing the character `'`. When the command is executed, lines 11 and 12 are recalled to the line buffer and listed on the display. As shown in the example, the RUBOUT→ key is pressed repeatedly to delete characters from the right hand portion of line 11. The BREAK key is then pressed to return the edited line to the text buffer and tell the EDITOR to continue the search. Line 12 is recalled and listed, and corrected using the RUBOUT→ key, and BREAK is pressed again. Since all occurrences of the character `'` have been located, pressing BREAK ends execution of the command.

The command `S 11, "MAT"`, tells the EDITOR to search from line 11 on for the string `MAT`. Lines containing `MAT` are to be listed and recalled to the line buffer. When the command is executed, lines 11, 12, and 13 are recalled to the line buffer and listed on the display. As in the last command, keyboard corrections are made to the lines, by pressing the RUBOUT keys and typing over incorrect characters. After lines 11 and 13 are corrected, a REPRINT function is performed to obtain a clean copy of the edited line. (Refer to the explanation of the REPRINT/CLEAR key in Section 3.) After each line is corrected, BREAK is pressed to return the line to the text buffer and resume the search. Execution ends after the BREAK key is pressed the third time.

The command `S ,10 "PRINT"`, tells the EDITOR to search up through line 10 for the string `PRINT`. Lines containing `PRINT` are to be listed and recalled to the line buffer. When the command is executed, lines 1 and 9 are recalled and listed on the display. For each line, the RUBOUT→ key is pressed to delete certain characters; a COMPRESS function is performed to delete blank spaces; and a clean copy of the line is obtained by performing a REPRINT function. The BREAK key is pressed after each line is corrected. Execution ends the second time the BREAK key is pressed.

The command `S 2,9 "TRACE"`, tells the EDITOR to search lines 2 through 9 for the string `TRACE`. Lines containing `TRACE` are to be listed and recalled to the line buffer. When the command is executed, lines 2 and 8 are recalled to the line buffer and listed on the display. Line 2 appears first and is corrected by typing over incorrect characters and by pressing the RUBOUT key. The line is returned to the text buffer by pressing BREAK. Line 8 appears next and is corrected and returned to the text buffer by pressing BREAK again. This time, pressing BREAK ends execution of the command.

Finally, a new listing shows how the text has been modified by the SEARCH and Edit Line commands.

## NOTES

## The SEARCH and Delete Line Command

### Syntax Form:

```
S [ I/O address ] [ [ edit line number ] , [ edit line number ] , ] b " string " *
```

### Descriptive Form:

```
SEARCH [ I/O address ] [ [ starting line number ] , [ ending line number ] , ] space " target string " *
```

## PURPOSE

The SEARCH and Delete Line command searches a portion of the current text for lines containing a specified target string. Lines found to contain the string are listed on a peripheral device, and deleted from the text buffer. If no peripheral device is specified, the lines are printed on the display, then deleted.

## EXAMPLES

```
S "transferred"*  
S 100,, "OPTION"*  
S ,5000 "TRACE"*  
S 100,5000 "pd."*  
S020:"REM"*  
S020:,5000 "X1"*  
S020:100,5000 "~"*
```

## EXPLANATION

The SEARCH and Delete Line command allows an I/O address, a starting line number, an ending line number, and an ASCII character string to be specified. The string is enclosed in quotation marks (or another string delimiter) and the last entry in the command is an asterisk (\*). When the SEARCH and Delete Line command is executed, the EDITOR searches the current text from the starting line through the ending line for the specified "target" string, and deletes lines found to contain the string.



## Editing Commands

### SEARCH and Delete Line

The deleted lines are listed on the peripheral device specified by the command. Lines found to contain more than one occurrence of the target string are only listed once.

For example, the command `S 100,500 "pd." *` searches lines 100 through 500 for occurrences of the target string `pd.`. Lines found to contain the string `pd.` are listed on the system display by default, and deleted from the text buffer. Similarly, the command `S 100,5000 "'" *` searches lines 100 through 5000, listing and deleting lines found to contain the character `'`.

### Default Values

When no I/O address is specified in a SEARCH and Delete Line command, deleted lines are listed on the system display by default. For example, the command `S 100,5000 "REM" *` searches lines 100 through 500 for the string `REM`. Lines found to contain `REM` are listed on the system display, and deleted from the text buffer.

When the starting line number is omitted, the EDITOR starts the search at the beginning of the first line of text. For example, the command `S ,500 "TRACE" *` searches the current text through line 500 for the string `TRACE`. Lines found to contain `TRACE` are listed on the system display and deleted from the text buffer.

When the ending line number is omitted, the search ends after the last line of text. For example, the command `S 100,,"OPTION" *` searches from the beginning of line 100 through the end of the text for the string `OPTION`. Lines found to contain `OPTION` are listed on the system display and deleted from the text buffer.

When both the starting and ending line numbers are omitted, the EDITOR searches the entire text. For example, the command `S "transferred" *` lists and deletes all lines containing the string `transferred`.

### Notes on the Command Syntax

When entering a SEARCH and Delete Line command, do not enter any blank spaces immediately before the asterisk (`*`). For example:

```
S "X2" *  
EDITOR ERROR  
Syntax - error number 138  
S "X2" *
```

This command causes a syntax error because a blank space appears between the string delimiter (`"`) and the asterisk (`*`).

When specifying a peripheral device in a SEARCH and Delete Line command, do not enter any blank spaces immediately after the I/O address. For example, the command `S @29: "REM" *` causes a syntax error. The correct way to enter the command is `S @29:"REM" *`.

**Edit Delimiters in the Command.** Just as for other searching commands, the syntax and descriptive forms show three edit delimiters, represented by two commas (,) and a space (␣). The two commas may be replaced by another edit delimiter, with the following exception: the second comma cannot be replaced by a space if the ending line number is omitted, or a syntax error occurs.

The space (␣) in the syntax and descriptive forms is not meant to be replaced by another edit delimiter. For example, the command `S,,,"REM" *` causes a syntax error.

Just as for the other searching commands, you can specify the edit delimiters in the manner indicated in the syntax and descriptive forms, or abbreviate the NLSEARCH and Replace String command according to these rules:

—If an ending line number is specified, you need only enter one edit delimiter between the ending line number and the target string. For example, the command `S 1,100 "REM" *` is valid. Several of the sample commands in this explanation are of this format. The space immediately following the keyword `S` is not required, and is entered only for the sake of appearance.

—When omitting both the starting and ending line numbers, you may enter two non-blank edit delimiters between the keyword `S` and the target string, as in the command `S,,"REM" *`. Or, you may replace the two edit delimiters with a blank space, as in `S "REM" *`. The format of this last command is more convenient, and is emphasized in the examples.

**String Delimiters in the Command.** As for all searching commands, the target string must be enclosed in a string delimiter, or a syntax error occurs. The string delimiter can be any keyboard character that causes a printed character to appear on the display except 1) a space; 2) one of the characters in the target or replacement string; or 3) a number or edit delimiter, if no edit line numbers are specified and a space appears in their place. For more information, refer to "String Delimiters" in Section 2.

### **Command Semantics**

When specifying a peripheral device in a SEARCH and Delete Line command, do not enter any blank spaces immediately before the colon (:) or a semantic error occurs. For example, the command `S@33 : "OPTION" *` causes a semantic error.

## Editing Commands

### SEARCH and Delete Line

The ending line number specified in a SEARCH and Delete Line command should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command either has no effect, or causes a semantic error.

### An Editing Example

The following example illustrates the use of the SEARCH and Delete Line command.

#### Example 1

```
LIST
1:340 REM ** BEGIN LOOP **
2:350 SET TRACE
3:360 U0=0
4:370 FOR K=1 TO 2↑N-1 STEP 2
5:380 X=A+K*H
6:390 S4=S4+FNF(X)
7:400 NEXT K
8:410 I2=H/3*(S0+2*S2+4*S4)
9:420 SET NORMAL
10:430 REM ** USE CORRECTION FACTOR **
11:440 I2=(16*I2-10)/15
12:450 U0=I2
13:460 PRINT
14:470 PRINT I2
15:480 IF ABS(I2-I0)<E THEN 540
16:490 I0=I2
17:500 NEXT N
18:510 REM ** PRINT FINAL VALUE **
19:520 PRINT
20:530 PRINT "FINAL VALUE IS":I2

S "PRINT]"*
13:460 PRINT
19:520 PRINT

S 2,15 "U0"*
3:360 U0=0
12:450 U0=I2

S 2,, "SET"*
2:350 SET TRACE
9:420 SET NORMAL

S ,19 "REM"*
1:340 REM ** BEGIN LOOP **
10:430 REM ** USE CORRECTION FACTOR **
18:510 REM ** PRINT FINAL VALUE **
```

(continued on next page)

```
LIST
4:370 FOR K=1 TO 2↑N-1 STEP 2
5:380 X=A+K*H
6:390 S4=S4+FNF(X)
7:400 NEXT K
8:410 I2=H/3*(S0+2*S2+4*S4)
11:440 I2=(16*I2-10)/15
14:470 PRINT I2
15:480 IF ABS(I2-I0)<E THEN 540
16:490 I0=I2
17:500 NEXT N
20:530 PRINT "FINAL VALUE IS";I2
```

Example 1 shows the SEARCH and Delete Line command being used to edit a BASIC program. An initial listing shows that the text consists of twenty BASIC program statements. Then four SEARCH and Delete Line commands are executed.

The command `S "PRINT]" *` tells the EDITOR to list and delete all lines that end with the string `PRINT ]`. The end of the line is represented in the target string by the symbol `]`, the current "END-OF-RECORD" character by default. (For more information, refer to the `]=` command in Section 5.) When the command is executed, the EDITOR lists and deletes two lines found to contain the target string, lines 13 and 19.

The command `S 2,15 "V0" *` searches lines 2 through 15 for the string `V0`. Lines containing `V0` are to be listed on the display, and deleted from the text buffer. When the command is executed, the EDITOR lists and deletes lines 3 and 12.

The command `S 2,, "SET" *` searches from line 2 on for the string `SET`. When the command is executed, the EDITOR lists and deletes two lines found to contain the string `SET`, lines 2 and 9.

The command `S ,19 "REM" *` searches up through line 19 for the string `REM`. When the command is executed, the EDITOR lists and deletes three lines found to contain the target string, lines 1, 10, and 18.

Finally, a complete listing shows how the program has been modified by the four SEARCH and Delete Line commands.

## NOTES

## The SEARCH and Replace String Command

### Syntax Form:

```
S [ I/O address ] [ [ edit line number ] , [ edit line number ] , ] b " string " , " string "
```

### Descriptive Form:

```
SEARCH [ I/O address ] [ [ starting line number ] , [ ending line number ] , ] space " target string "  
 , " replacement string "
```

## PURPOSE

The SEARCH and Replace String command searches a portion of the current text for a specified target string, and replaces occurrences of the string with the desired replacement string. Changed lines are listed on a peripheral device, or on the display if no device is specified.

## EXAMPLES

```
S /MAT B=IDN(3,3)/,/DIM B(3,3)ICALL "IDN",B/  
S 100,, /MAT C=CON(N,N)/,/DIM C(N,N)IC=1/  
S ,5000 "MAT INPUT A_(", "INPUT A"  
S 100,5000 "# ^", "~ REM"  
S033:"TRACE ON", "SET TRACE"  
S033:100,, "PRINT ~; ", "PRINT ~, "  
S033:,5000 "READ#1; ", "READ@33:"  
S033:100,5000 "PRINT", "PRINT@3:"
```

## Editing Commands

### SEARCH and Replace String

#### EXPLANATION

The SEARCH and Replace String command allows an I/O address, a starting line number, an ending line number, and two ASCII character strings to be specified. The first string is the "target" string and the second string is the "replacement" string. Both strings must be enclosed in quotation marks or another string delimiter.

When the SEARCH and Replace String command is executed, the EDITOR searches the current text from the starting line through the ending line for the target string. Occurrences of the target string in the text are overwritten by the replacement string in the following manner: the first character of the replacement string overwrites the first character of the target string, the second character of the replacement string overwrites the second character of the target string, and so on.

Each time an occurrence of the target string is overwritten by the replacement string, the changed line is listed on the peripheral device specified by the command. If a line contains more than one occurrence of the target string, the line is listed more than once. For example, a line containing three occurrences of the target string is listed three times on the specified peripheral device.

The replacement string need not have the same number of characters as the target string. If the replacement string is longer than the target string, the SEARCH and Replace String command lengthens the text; but if the replacement string is shorter than the target string, the command shortens the text.

For example, the command `S 100,5000 "PRINT","PRINT@3:"` tells the EDITOR to search lines 100 through 5000 of the current text for the target string `PRINT`. Occurrences of `PRINT` are overwritten by `PRINT@3:` and changed lines are listed on the system display by default. Each time an occurrence of `PRINT` is replaced by `PRINT@3:` the text increases by three characters.

#### Default Values

When no I/O address is specified in a SEARCH and Replace String command, changed lines are listed on the system display by default. For example, the command `S 100,5000 "TRACE ON","SET TRACE"` searches lines 100 through 5000 for the string `TRACE ON`. Occurrences of the string `TRACE ON` are replaced by `SET TRACE` and the changed lines are listed on the system display.

When the starting line number is omitted, the EDITOR starts the search at the beginning of the first line of text. For example, the command `S ,5000 "MAT INPUT","INPUT"` searches the current text through line 5000 for the string `MAT INPUT`. Occurrences of `MAT INPUT` are replaced by `INPUT` and changed lines are listed on the system display.

When the ending line number is omitted, the search ends after the last line of text. For example, the command `S 100, /MAT C=CON/,/C=1/` searches from the beginning of line 100 through the end of the text for the string `MAT C=CON`. Occurrences of `MAT C=CON` are replaced by `C=1` and changed lines are listed on the system display.

### Notes on the Command Syntax

When entering a SEARCH and Replace String command, use only one edit delimiter between the target and replacement strings. Entering more than one delimiter causes a syntax error. For example:

```
S 1,100 "PRINT" , "PRINT @33:"  
EDITOR ERROR  
Syntax - error number 138  
S 1,100 "PRINT" , "PRINT @33:"
```

The correct way to enter the command is as follows:

```
S 1,100 "PRINT","PRINT @33:"
```

When specifying a peripheral device in a SEARCH and Replace String command, do not enter any blank spaces immediately after the I/O address. For example, the command `S@33: 1,100 "TRACE ON","SET TRACE"` causes a syntax error. The correct way to enter the command is `S@33:1,100 "TRACE ON","SET TRACE"`.

**Edit Delimiters in the Command.** Just as for other searching commands, the syntax and descriptive forms show three edit delimiters, represented by two commas (,) and a space (b). The two commas may be replaced by another edit delimiter, with the following exception: the second comma cannot be replaced by a space if the ending line number is omitted, or a syntax error occurs.

The space (b) in the syntax and descriptive forms is not meant to be replaced by another edit delimiter. For example, the command `S,, "@33","@29"` causes a syntax error.

Just as for the other searching commands, you can specify the edit delimiters in the manner indicated in the syntax and descriptive forms, or abbreviate the NLSEARCH and Replace String command according to these rules:

—If an ending line number is specified, you need only enter one edit delimiter between the ending line number and the target string. For example, the command `S 1,100 "A0","A2"` is valid. Several of the sample commands in this explanation are of this format. The space immediately following the keyword `S` is not required and is entered only for the sake of appearance.



## Editing Commands

### SEARCH and Replace String

—When omitting both the starting and ending line numbers, you may enter two non-blank edit delimiters between the keyword `S` and the target string, as in the command `S,,"A0","A2"` . Or, you may replace the two edit delimiters with a blank space, as in `S "A0","A2"` . The format of this last command is more convenient, and is emphasized in the examples.

**String Delimiters in the Command.** As for all searching commands, the target and replacement strings must be enclosed in a string delimiter, or a syntax error occurs. The delimiter used to enclose the target string need not be the same as the delimiter used to enclose the replacement string.

A string delimiter can be any keyboard character that causes a printed character to appear on the display except 1) a space; 2) one of the characters in the target or replacement string; or 3) a number or edit delimiter, if no edit line numbers are specified and a space appears in their place. For more information, refer to "String Delimiters" in Section 2.

### Command Semantics

The ending line number specified in a SEARCH and Replace String command should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command either has no effect, or causes a semantic error.

When specifying a peripheral device in a SEARCH and Replace String command, do not enter any blank spaces immediately before the colon (`:`) or a semantic error occurs. For example, the command `S@33 : "MAT INPUT","INPUT"` causes a semantic error.

### Special Uses for the SEARCH and Replace String Command:

#### Deleting Specified Strings

If the replacement string is empty (has no characters), the SEARCH and Replace String command deletes occurrences of the target string from the text buffer. This is not the same as deleting the *line* in which the target string occurs (the function provided by the NLSEARCH and Delete Line and SEARCH and Delete Line commands).

For example, the command `S /MAT/,//` locates and deletes all occurrences of the string `MAT` in the current text.

### An Editing Example

The following example illustrates the use of the SEARCH and Replace String command.

#### Example 1

```

LIST
  1:620 'CORRECT MATRIX A
  2:630 PRINT@33:"A=";A
  3:640 TRACE ON
  4:650 FOR J=1 TO N-1
  5:660 FOR I=J+1 TO N
  6:670 A(I,N+1)=A(I,J)*A(J,N+1)
  7:680 NEXT I
  8:690 NEXT J
  9:700 TRACE OFF
 10:710 PRINT@33:"A=";A
 11:720 'INITIALIZE MATRICES B,C
 12:730 MAT C=CON(N,N)
 13:740 MAT B=IDN(N,N)

S ,5 "TRACE ON","SET TRACE"
  3:640 SET TRACE

S 5,, "TRACE OFF","SET NORMAL"
  9:700 SET NORMAL

S "@33:", " "
  2:630 PRINT "A=";A
 10:710 PRINT "A=";A

S "# /", "~ REM "
  1:620 REM CORRECT MATRIX A
 11:720 REM INITIALIZE MATRICES B,C

S /MAT C=CON(N,N)/,/DIM C(N,N)]735 C=1/
  :735 C=1

S /MAT B=IDN(N,N)/,/DIM B(N,N)]750 CALL "IDN",B/
  :750 CALL "IDN",B

LIST
  1:620 REM CORRECT MATRIX A
  2:630 PRINT "A=";A
  3:640 SET TRACE
  4:650 FOR J=1 TO N-1
  5:660 FOR I=J+1 TO N
  6:670 A(I,N+1)=A(I,J)*A(J,N+1)
  7:680 NEXT I
  8:690 NEXT J
  9:700 SET NORMAL
 10:710 PRINT "A=";A
 11:720 REM INITIALIZE MATRICES B,C
 12:730 DIM C(N,N)
  :735 C=1
 13:740 DIM B(N,N)
  :750 CALL "IDN",B

```

## Editing Commands

### SEARCH and Replace String

Example 1 shows the SEARCH and Replace String command being used to locate and replace specified strings, and make a "foreign" BASIC program compatible with 4051 BASIC. An initial listing shows that the text consists of thirteen BASIC program statements. Then six SEARCH and Replace String commands are executed.

The command `S ,5 "TRACE ON","SET TRACE"` tells the EDITOR to search the current text up to and including edit line 5 for the string `TRACE ON`. Occurrences of `TRACE ON` are to be replaced by `SET TRACE` and the changed lines listed on the display.

When the command is executed, an occurrence of the target string is located in line 3 and overwritten with the string `SET TRACE`. A listing of line 3 as changed by the replacement procedure appears on the next line of the display.

The command `S 5, "TRACE OFF","SET NORMAL"` tells the EDITOR to search from line 5 on for the string `TRACE OFF`. Occurrences of `TRACE OFF` are to be replaced by `SET NORMAL` and the changed lines listed on the display.

When the command is executed, an occurrence of the target string is located in line 9 and overwritten with `SET NORMAL`. A listing of line 9 as changed by the replacement procedure appears on the display.

Next the command `S "@33:"," "` tells the EDITOR to locate all occurrences of `@33:` in the text. The string `@33:` is to be replaced by a space (the replacement string consists of one blank space) and the changed lines are to be listed on the display. When the command is executed, occurrences of `@33:` are located and replaced in lines 2 and 10. After the replacement procedure, a listing of lines 2 and 10 appears on the next lines of the display.

Then the command `S "# ","~ REM"` is executed. At the time the command is executed, the symbol `#` is the "any digit" character and `~` is the wildcard character: no `#=` or `~=` commands have been previously executed. The command `S "# ","~ REM"` tells the EDITOR to search the entire text for a string consisting of a digit immediately followed by a space and an apostrophe (`'`). Occurrences of this string are to be overwritten with a five-character string consisting of the first character of the target string, immediately followed by a space and `REM`.

When the command is executed, the EDITOR locates the target string in lines 1 and 11. After the replacement procedure, a new listing of the two lines appears on the display.

Next the command `S /MAT C=CON(N,N)/, /DIM C(N,N)]735 C=1/` tells the EDITOR to replace all occurrences of `MAT C=CON(N,N)` with the string `DIM C(N,N)` immediately followed by a new line that begins with `735 C=1`. The beginning of the new line is represented by the symbol `]`, the current "END-OF-RECORD" character by default. When the command is executed, the EDITOR locates and overwrites an occurrence of the target string in edit line 12. The replacement procedure results in a new line. A listing of the new line `735 C=1` appears on the display.

## **Editing Commands SEARCH and Replace String**

Finally the command `S /MAT B=IDN(N,N)/,/DIM B(N,N)]750 CALL "IDN",B/` is executed. The EDITOR locates an occurrence of the string `MAT B=IDN(N,N)` in line 13, and overwrites the string with `DIM B(N,N)` followed by a new line that begins with `750 CALL "IDN",B` . A listing of the new line appears on the display.

After all of the SEARCH and Replace String commands are executed, a listing of the entire text shows all the changes that have been made.

## NOTES

## The SORT Command

### Syntax Form:

SO [ edit line number ] , [ edit line number ] , numeric constant [ , numeric constant [ , . . . ] ]

### Descriptive Form:

SORT [ starting line number ] , [ ending line number ] , character position [ , character position  
 [ , . . . ] ]

## PURPOSE

The SORT command rearranges lines of text, sorting "alphabetically" according to the ASCII values of characters found in specified positions within each line.

## EXAMPLES

```
SO ,,1
SO ,,1,2,3
SO 1,,1,2
SO ,100,1,2
SO 1,1000,1,2
SO 1,1000,1,2,3,4,5,6
```

## EXPLANATION

The SORT command allows a starting line number, an ending line number, and character positions to be specified. When the SORT command is executed, the EDITOR rearranges the text between and including the starting and ending line numbers using a "selection sort" procedure. Lines are rearranged in the text buffer according to the ASCII value of characters found in the positions specified by the command. For example, the command SO 1,1000,1,2 tells the EDITOR to rearrange lines 1 through 1000 of the text buffer according to the ASCII value of characters found in the first two character positions in each line.

## Editing Commands

### **SORT**

Lines are rearranged in order of increasing ASCII value. For example, the command `SO 500,600,1` rearranges lines 500 through 600 of the text so that the ASCII code values of characters found in the first character position increase. That is, a line beginning with `A` is placed before a line beginning with `C`; and a line beginning with `0` is placed before a line beginning with `3`.

Sorting begins with the last character position specified in the command, and ends with the first character position specified in the command. For example, the command `SO 1,1000,1,2,3` sorts lines 1 through 1000 according to the ASCII value of characters found in the third character position in each line, then the second character position, then the first character position. This has the effect of "alphabetizing" lines on the basis of their first three characters.

The I/O light on the system front panel lights up briefly each time the SORT command moves a line of text. When the text is completely sorted, the I/O light stops blinking and the cursor reappears on the next line of the display.

To stop the execution of a SORT command, you may press the BREAK key twice.

### **What Happens When a Specified Character Position is Empty**

Since text rearranged by the SORT command may consist of lines of varying lengths, one or more lines may be empty (have no character) in the specified character position.<sup>7</sup> When the SORT command rearranges lines according to characters in a certain position, lines that are empty in that position move to the top of the text buffer.

For example, the command `SO 100,500:1,2,3,4` begins sorting on the fourth character position. Any lines having fewer than four characters are moved to the top of the text buffer. When the command sorts on the third character position, lines having fewer than four characters move to the top of the text buffer; and so on.

When sorting is complete, the lines are "alphabetized." For example, the line `CAR` precedes `CART`, the line `TO` precedes `TOP`, and `A` precedes `ART`.

### **Default Values**

When the starting line number is omitted in a SORT command, the first line involved in the sorting process is the first line of the current text. That is, the command `SO ,100,1,2` sorts all lines up to and including line 100 according to the ASCII value of characters found in the first two character positions in each line.

<sup>7</sup> Don't confuse an empty character position with a character position containing a space, the character SP (ASCII 32).

When the ending line number is omitted in a SORT command, the last line involved in the sorting process is the last line of the current text. For example, the command `SO 1,,1,2` sorts from line 1 through the end of the text, according to the ASCII value of characters found in the first two character positions.

When both the starting and ending line numbers are omitted, the command acts upon the entire text. That is, the command `SO,,1,2,3` sorts all lines of text according to the ASCII value of characters found in the first three character positions.

### Notes on the Command Syntax

**Interpreting the SORT Command.** The EDITOR interprets the SORT command in the following way: the number that precedes the first edit delimiter (,) is a starting line number. The number that immediately precedes the second edit delimiter is an ending line number. Numbers that immediately precede any subsequent edit delimiters are character positions.

Default values are supplied if one or both of the starting and ending line numbers are omitted in the command. Some sample SORT commands are shown below, along with their interpretations. The commands specify 5, 6, and 7 as parameters, but differ in their interpretations because of the number and placement of the edit delimiters.

Command	Interpretation
<code>SO 5,6,7</code>	Sort lines 5 through 6 on character position 7.
<code>SO ,5,6,7</code>	Sort up through line 5 on character positions 7 and 6.
<code>SO, ,5,6,7</code>	Sort all lines of text, on character positions 7, 6, and 5.
<code>SO 5, ,6,7</code>	Sort from line 5 on, on character positions 7 and 6.

To help you remember the interpretation of a SORT command, you may separate edit line numbers from character positions using an edit delimiter other than a comma, as in the command `SO 50,60:1,2,3`.

**Syntax Errors.** A minimum of two edit delimiters and one character position must be specified in a SORT command, or a syntax error occurs. For example, all of the following commands cause syntax errors:

```
SO 5,
SO ,100,
SO 1,2
SO 100,500,
```



## Editing Commands

### **SORT**

These commands cause syntax errors because they do not specify at least two edit delimiters (,) and one character position. For instance, the first command `SO 5,` specifies only one edit delimiter and no character position: `5` is interpreted as a starting line number. The last command `SO 100,500,` specifies two edit delimiters, but no character position: `100` and `500` are interpreted as starting and ending line numbers, respectively.

**Character Positions.** A character position can be any digit from 1 through 9999. The character positions specified in a SORT command are generally entered consecutively, and in increasing order, as in the command `SO,,1,2,3,4,5,6`. However, the following commands are syntactically correct:

```
SO 100,200:5,1,2,3,4
```

```
SO 100,200:7,12
```

Commands of this type are useful in certain circumstances. (Refer to "Special Uses for the SORT Command" in this explanation.)

### **Command Semantics**

The ending line number specified in a SORT command should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command either has no effect, or causes a semantic error.

### **How the SORT Command is Affected by the Lowercase or Uppercase Flag**

Calling the EDITOR or executing the LOWERCASE command sets a "lowercase flag." Setting the lowercase flag enables the EDITOR to distinguish between lowercase and uppercase text characters during the sorting process. For example, when sorting lines according to the ASCII value of characters in the first position, the EDITOR places a line beginning with `b` (ASCII 98) after a line beginning with `B` (ASCII 66).

Executing the UPPERCASE command sets an "uppercase flag." Setting the uppercase flag causes the EDITOR to treat all lowercase text characters as uppercase characters during the sorting process. For example, when sorting lines according to the ASCII value of characters in the first character position, the EDITOR treats a line beginning with `b` as if it began with `B` (ASCII 66).

For illustrations of how the lowercase and uppercase flags affect the result of the SORT command, refer to "An Editing Example" in the explanations of the LOWERCASE and UPPERCASE commands.

## Special Uses for the SORT Command

**Alphabetizing Using the SORT Command.** The SORT command may be used to alphabetize lists of names or words that begin in the same character position and contain the characters A-Z and/or a-z. An illustration is provided in "An Editing Example" in this explanation. The SORT command should specify enough character positions to completely alphabetize the list. For example, if the longest name in the list has six characters, the command `SO,,1,2,3,4,5,6` completely alphabetizes the list.

The list may be in lowercase letters, or uppercase, or both. For instance, the first character in each line may be uppercase, and the rest of the characters lowercase. However, keep in mind that when the lowercase flag is set, the EDITOR distinguishes between lowercase and uppercase characters. For example, if the lowercase flag is set, the EDITOR moves the line `art` to a location after the line `Art`, and places `art` after `Zap`. If you do not want lines to be "alphabetized" in this manner, you may execute the UPPERCASE command to make the EDITOR treat all text characters as uppercase.

Words or names to be alphabetized should begin in the same character position. This is because blank spaces preceding the word are the character `SP` (ASCII 32). When the EDITOR sorts on a particular character position, a line having a space in that position is placed above a line having any other alphanumeric character: all ASCII characters except control characters have a higher ASCII code value than `SP`.

**Arranging Numbers in Increasing Order.** The SORT command can be used to arrange numbers in increasing order. The numbers need not all have the same number of digits. For example:

```
LIST
: 11
:5126
: 643
: 0
```

```
SO,,1,2,3,4
```

```
LIST
: 0
: 11
: 643
:5126
```

If the numbers to be arranged in increasing order contain a decimal point (`.`), the decimal point must appear in the same character position in each line.

## Editing Commands

### **SORT**

The numbers to be arranged in increasing order should either all begin in the same character position, or all end in the same character position. If the numbers all end in the same character position, you should execute a SORT command that specifies the appropriate character positions consecutively and in increasing order, as in the command `SO,,1,2,3,4,5,6` .

If the numbers to be arranged in increasing order all begin in the same character position, you should execute a different SORT command. For example:

```
LIST
      :67
      :6112
      :43
      :51260
      :973

SO,,5,4,3,1,2
```

```
LIST
      :43
      :67
      :973
      :6112
      :51260
```

The SORT command in this example first specifies character positions that are empty in one or more of the lines to be sorted. The character positions are entered consecutively and in decreasing order: `5,4,3` . Next the command specifies the remaining character positions consecutively and in *increasing* order: `1,2` .

Only a SORT command of the type described in this example can correctly arrange numbers that begin in the same character position, and do not all have the same number of digits.

## An Editing Example

The following example illustrates the SORT command being used to alphabetize a list of names. Because all of the names begin in the same character position and have nineteen or fewer characters, a SORT command that specifies the first nineteen character positions completely alphabetizes the list.<sup>8</sup>

### Example 1

```
LIST
:Zimmerman, Neil D.
:Hillstrom, A. A.
:Carmichael, David
:Brockway, Marius E.
:Harvey, Richard A.
:Taylor, Owen
:Gardner, Keith W.
:Foster, Alice
:Siebold, William B.
:Lentz, John F.
:Sloan, Irene
:Pollock, Robert
:Kearney, John D.
:Ellis, Terry L.
:Keller, Suzanne

SORT,,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19

LIST
:Brockway, Marius E.
:Carmichael, David
:Ellis, Terry L.
:Foster, Alice
:Gardner, Keith W.
:Harvey, Richard A.
:Hillstrom, A. A.
:Kearney, John D.
:Keller, Suzanne
:Lentz, John F.
:Pollock, Robert
:Siebold, William B.
:Sloan, Irene
:Taylor, Owen
:Zimmerman, Neil D.
```

<sup>8</sup> Actually for this particular list of names, the command SO,,1,2 would suffice to completely alphabetize the list, because no two names are identical in the first three or more positions.

## Editing Commands

### **SORT**

#### **Timing Information**

The SORT command uses a "selection sort" algorithm.<sup>9</sup> The amount of time required to sort the text depends on the number of lines to be sorted, the number of characters per line, and the number of character positions specified in the command.

The execution time for a SORT command may be approximated by the following formula:

$$\text{Time (in seconds)} = (2.02\text{E-}5) * S * C * L^{1.016}$$

where     S = the number of character positions specified in the SORT command  
           C = the average number of characters per line  
           L = the number of lines of randomly ordered text

For example, executing the command `SO,,1` takes about six seconds if the text consists of 100 randomly arranged lines having an average of 63 characters per line.

An alternative formula is as follows:

$$\text{Time (in seconds)} = (2.02\text{E-}5) * S * T * L^{1.016}$$

where     S = the number of character positions specified in the SORT command  
           T = the total number of characters to be sorted  
           L = the number of lines of randomly arranged text

Both of the above formulas assume the text is randomly ordered. Execution times are much shorter, and will be minimal, if the text is already fairly well sorted.

<sup>9</sup> For information about sorting by selection, refer to Donald E. Knuth, *The Art of Computer Programming*, Vol. 3, "Sorting and Searching," Addison-Wesley Publishing Company, 1973, p. 139.

## The REVSORT Command

### Syntax Form:

```
REV [ edit line number ] , [ edit line number ] , numeric constant [ , numeric constant [ , . . . ] ]
```

### Descriptive Form:

```
REVSORT [ starting line number ] , [ ending line number ] , character position [ , character position  
[ , . . . ] ]
```

## PURPOSE

The REVSORT command rearranges lines of text, sorting "alphabetically in reverse" according to the ASCII values of characters found in specified positions within each line.

## EXAMPLES

```
REV , , 1  
REV , , 1, 2, 3  
REV 1, , 1, 2  
REV , 1000, 1, 2  
REV 1, 1000, 1, 2  
REV 1, 1000, 1, 2, 3, 4, 5, 6
```

## EXPLANATION

Like the SORT command, the REVSORT command allows a starting line number, an ending line number, and character positions to be specified. When the REVSORT command is executed, the EDITOR rearranges the text between and including the starting and ending line numbers using a "selection sort" procedure. Lines are rearranged in the text buffer according to the ASCII value of characters found in the positions specified by the command. For example, the command `REV 1,1000,1,2` tells the EDITOR to rearrange lines 1 through 1000 of the text buffer according to the ASCII value of characters found in the first two character positions in each line.

## Editing Commands

### REVSORT

Unlike the SORT command, the REVSORT command rearranges lines in order of *decreasing* ASCII value. For example, the command `REV 500,600,1` rearranges lines 500 through 600 of the text so that the ASCII code values of characters found in the first character position decrease. That is, a line beginning with `A` is placed after a line beginning with `C`; and a line beginning with `0` is placed after a line beginning with `3`.

Sorting begins with the last character position specified in the command, and ends with the first character position specified in the command. For example, the command `REV 1,1000,1,2,3` sorts lines 1 through 1000 according to the ASCII value of characters found in the third character position in each line, then the second character position, then the first character position. This has the effect of "reverse alphabetizing" lines on the basis of their first three characters.

The I/O light on the system front panel lights up briefly each time the REVSORT command moves a line of text. When the text is completely sorted, the I/O light stops blinking and the cursor reappears on the next line of the display.

To stop the execution of a REVSORT command, you may press the BREAK key twice.

#### **What Happens When a Specified Character Position is Empty**

Since text rearranged by the REVSORT command may consist of lines of varying lengths, one or more lines may be empty (have no character) in the specified character position.<sup>10</sup> When the REVSORT command rearranges lines according to characters in a certain position, lines that are empty in that position move to the top of the text buffer.

For example, the command `REV 100,500:1,2,3,4` begins sorting on the fourth character position. Any lines having fewer than four characters are moved to the top of the text buffer. When the command sorts on the third character position, lines having fewer than four characters move to the top of the text buffer; and so on.

#### **When the REVSORT Command is the Inverse of the SORT Command**

The REVSORT command performs the inverse of the SORT command if the lines being sorted all have characters (are non-empty) in the specified positions. For example, if all lines have at least five characters, the command `REV,,1,2,3,4,5` rearranges the text in reverse order from the command `SO,,1,2,3,4,5`.

If one or more lines is empty in a position specified by the command, the REVSORT command may not perform the inverse of the SORT command.<sup>10</sup> For example, if one of the lines has only three characters, the command `REV,,1,2,3,4,5` may not rearrange the text in reverse order from the command `SO,,1,2,3,4,5`.

<sup>10</sup> Don't confuse an empty character position with a character position containing a space, the character SP (ASCII 32).

## Default Values

When the starting line number is omitted in a REVSORT command, the first line involved in the sorting process is the first line of the current text. That is, the command `REV ,100,1,2` sorts all lines up to and including line 100 according to the ASCII value of characters found in the first two character positions in each line.

When the ending line number is omitted in a REVSORT command, the last line involved in the sorting process is the last line of the current text. For example, the command `REV 1,,1,2` sorts from line 1 through the end of the text, according to the ASCII value of characters found in the first two character positions.

When both the starting and ending line numbers are omitted, the command acts upon the entire text. That is, the command `REV,,1,2,3` sorts all lines of text according to the ASCII value of characters found in the first three character positions.

## Notes on the Command Syntax

**Interpreting the REVSORT Command.** The number that precedes the first edit delimiter (,) in a REVSORT command is a starting line number. The number that immediately precedes the second edit delimiter is an ending line number, and numbers that immediately precede any subsequent edit delimiters are character positions. Default values are supplied if one or both of the starting and ending line numbers are omitted in the command. For example, the command `REV 5,6,7` tells the EDITOR to "reverse" sort lines 5 through 6 on character position 7, but the command `REV,,5,6,7` tells the EDITOR to "reverse" sort all lines, on character positions 7, 6, and 5.

To help you remember the interpretation of a REVSORT command, you may separate edit line numbers from character positions using an edit delimiter other than a comma, as in the command `REV 50,60:1,2,3`.

**Syntax Errors.** Like the SORT command, the REVSORT command requires a minimum of two edit delimiters and one character position to be specified, or a syntax error occurs. For example, all of the following commands cause syntax errors:

```
REV
REV 5,
REV ,100,
REV 100,500
```

These commands cause syntax errors because they do not specify at least two edit delimiters (,) and one character position.



## Editing Commands

### REVSORT

**Character Positions.** A character position can be any digit from 1 through 9999. The character positions specified in a REVSORT command are generally entered consecutively and in increasing order, as in the command `REV,,1,2,3,4,5,6` . However, the following commands are syntactically correct:

```
REV 100,200:5,1,2,3,4
```

```
REV 100,200:7,12
```

### Command Semantics

The ending line number specified in a REVSORT command should be at least as large as the starting line number. If the ending line number is smaller than the starting line number, the command either has no effect, or causes a semantic error.

### How the REVSORT Command is Affected by the Lowercase or Uppercase Flag

Calling the EDITOR or executing the LOWERCASE command sets a "lowercase flag." Setting the lowercase flag enables the EDITOR to distinguish between lowercase and uppercase text characters during the sorting process. For example, when the REVSORT command sorts lines according to the ASCII value of characters in the first position, a line beginning with `b` (ASCII 98) is placed before a line beginning with `B` (ASCII 66).

Executing the UPPERCASE command sets an "uppercase flag." Setting the uppercase flag causes the EDITOR to treat all lowercase text characters as uppercase characters during the sorting process. For example, when the REVSORT command sorts lines according to the ASCII value of characters in the first character position, a line beginning with `b` is treated as if it began with `B` (ASCII 66).

For illustrations of how the lowercase and uppercase flags affect the result of the REVSORT command, refer to "An Editing Example" in the explanations of the LOWERCASE and UPPERCASE commands.

### Special Uses for the REVSORT Command

**"Reverse Alphabetizing" Using the REVSORT Command.** The REVSORT command may be used to "reverse alphabetize" lists of names or words that contain the characters A-Z and/or a-z. The REVSORT command should specify enough character positions to completely sort the list. For example, if the longest name in the list has six characters, the command `REV,,1,2,3,4,5,6` completely "reverse alphabetizes" the list.

The list may be in lowercase letters, or uppercase, or both. For instance, the first character in each line may be uppercase, and the rest of the characters lowercase. However, keep in mind that when the lowercase flag is set, the EDITOR distinguishes between lowercase and uppercase characters. For example, if the lowercase flag is set, the REVSORT command places the line `art` before the line `ART`, and places `art` before `Zap`. If you do not want lines to be "reverse alphabetized" in this manner, you may execute the UPPERCASE command to make the EDITOR treat all text characters as uppercase.

Words or names to be "reverse alphabetized" should either all begin in the same character position, or all end in the same character position. If the names all end in the same character position, you should execute a REVSORT command that specifies the appropriate character positions consecutively and in increasing order, as in the command `REV,,1,2,3,4,5,6`. If the names all begin in the same character position, you should first execute a CARD command that specifies 32 (SP) as the fill character and fills out all lines to the same character position. Then execute a REVSORT command that specifies the appropriate character positions consecutively and in increasing order as in the command `REV,,1,2,3,4,5,6`. Executing a CARD command to fill out the lines to a uniform length ensures that the REVSORT command correctly "reverse alphabetizes" the list (in the sense of performing the inverse of the SORT command).

**Arranging Numbers in Decreasing Order.** The REVSORT command can be used to arrange numbers in decreasing order. An illustration is provided in "An Editing Example" in this explanation. Numbers to be arranged in decreasing order by the REVSORT command need not all have the same number of digits, but if the numbers contain a decimal point (.) the decimal point must appear in the same character position in each line.

An example is shown below:

```
LIST
: 43
: 611
: 0
:5126
: 973

REV, , 1, 2, 3, 4, 5

LIST
:5126
: 973
: 611
: 43
: 0
```

## Editing Commands

### REVSORT

Numbers to be arranged in decreasing order should either all begin in the same character position, or all end in the same character position. If the numbers end in the same character position, you should execute a REVSORT command that specifies the appropriate character positions consecutively and in increasing order, as in the command `REV,,1,2,3,4` .

If the numbers to be arranged in decreasing order all begin in the same character position and do not all have the same number of digits, you should first execute a CARD command, then a REVSORT command. The CARD command should use ASCII code value 32 (SP) as the fill character. The REVSORT command should specify the appropriate character positions in the following order: first specify character positions that have no digit in one or more of the lines to be sorted. Enter these positions consecutively and in *decreasing* order. Then specify the remaining character positions consecutively and in *increasing* order.

An example is shown below:

```
LIST
      :67
      :6112
      :43
      :51260
      :973

CA 5,32

REV,,5,4,3,1,2

LIST
      :51260
      :6112
      :973
      :67
      :43
```

In this example the numbers to be arranged in decreasing order all begin in the first character position, and do not all have the same number of digits. Executing the command `CA 5` formats the text into lines of five characters, using SP (a space) as the fill character by default.

Because one or more of the lines has no digit in character positions 5, 4, and 3, the REVSORT command is entered as `REV,,5,4,3,1,2` . After the command is executed, a new listing shows that the five numbers have been correctly arranged in decreasing order.

**An Editing Example**

The following example shows the REVSORT command being used to arrange numbers in decreasing order. The numbers are found in character positions 33 through 41, and all end in the same character position (41). The REVSORT command executed in the example tells the EDITOR to rearrange lines 3 through 14, on the basis of the ASCII value of characters found in positions 33 through 41. After the command is executed, a new listing shows that the lines have been rearranged so that the numbers in the "Amount Sold" column are in decreasing order.

**Example 1**

```
LIST
1: SALESMAN                AMOUNT SOLD
2: -----
3: CLIFFORD                $   920,000
4: ETHERIDGE              $   150,990
5: GLINES                 $   249,800
6: KINTZ                  $   820,000
7: LENZ                   $   950,300
8: NOAKES                 $   433,600
9: PARKER                 $  1,320,500
10: SAUER                  $  1,570,000
11: THOMPSON              $   720,400
12: VAN DYKE              $   565,300
13: VERMIERE              $   450,800
14: WENTWORTH             $  1,102,000
```

```
REV 3, 14, 33, 34, 35, 36, 37, 38, 39, 40, 41
```

```
LIST
1: SALESMAN                AMOUNT SOLD
2: -----
3: SAUER                   $  1,570,000
4: PARKER                 $  1,320,500
5: WENTWORTH              $  1,102,000
6: LENZ                   $   950,300
7: CLIFFORD               $   920,000
8: KINTZ                  $   820,000
9: THOMPSON               $   720,400
10: VAN DYKE              $   565,300
11: VERMIERE              $   450,800
12: NOAKES                $   433,600
13: GLINES                $   249,800
14: ETHERIDGE             $   150,990
```

## **Editing Commands**

### **REVSORT**

#### **Timing Information**

Like the SORT command, the REVSORT command uses a "selection sort" algorithm. The execution time for a REVSORT command depends on the number of lines to be sorted, the number of characters per line, and the number of character positions specified in the command. Execution times may be approximated using the formulas provided for the SORT command. (Refer to "Timing Information" in the explanation of the SORT command.)

## Section 5

# ENVIRONMENTAL COMMANDS

<b>CONTENTS</b>	<b>PAGE</b>
The LASTLINE Command . . . . .	5-3
The LOWERCASE Command . . . . .	5-9
The RENUMBER Command . . . . .	5-15
The UPPERCASE Command . . . . .	5-21
The #= Command . . . . .	5-27
The ~= Command . . . . .	5-33
The _= Command . . . . .	5-41
The  = Command . . . . .	5-47

## NOTES

## The LASTLINE Command

**Syntax Form:**

LA

**Descriptive Form:**

LASTLINE

### PURPOSE

The LASTLINE command returns the following information about the current status of the text buffer:

- The edit line number of the last line in the text buffer (including an offset if the last line does not have a number).
- The total number of lines in the text buffer.
- The number of bytes needed to save the current contents of the text buffer on a storage device.
- The number of unused bytes remaining in the text buffer.

### EXAMPLE

LA

### EXPLANATION

The LASTLINE command has no parameters. Only the keyword LA is entered from the keyboard, as shown in the example above. When the LASTLINE command is executed, the EDITOR returns information about the current contents of the text buffer.

### The Last Line of Text

When the LASTLINE command is executed, information about the text buffer is printed on the next four lines of the display. The first number printed on the display is the edit line number of the last line in the current text. The edit line number may be expressed using an offset, as in +10 or 1200+5. If the text buffer is empty, no number is printed, and a blank line appears on the display.



#### The Number of Lines in the Current Text

The next line on the display shows the number of lines in the current text. For example, if the text buffer currently contains 12 lines of text, the words `12 Lines` appear on the display.

#### The Number of Characters in the Current Text

The next line on the display gives the number of characters in the current text. For example, if the text buffer currently contains 303 characters, the information `303 Characters` appears on the display.

The number of characters includes a count of end-of-record characters (usually CARRIAGE RETURNS) used to delimit lines of text upon output. However, edit line numbers do not contribute to the count, nor do the colons (:) that mark the beginning of text in listings.<sup>1</sup>

Because each character requires one byte of storage space on a peripheral device, the number of characters returned by the LASTLINE command is the amount of space (in bytes) required to store the current text.

This number determines how large a file must be marked to store the current text. To mark a file large enough to store the text using the SAVE or WRITE command, mark the file to have one more byte than the number of characters returned by the LASTLINE command. The extra byte must be reserved for storing an end-of-file mark, which the EDITOR adds after the last character when outputting text.

For instance, if the LASTLINE command returns `1536 Characters`, and you intend to store the entire text on magnetic tape using the SAVE or WRITE command, the file must be marked to have at least 1537 bytes. (Refer to the MARK command for an explanation of how to mark a tape file under EDITOR control.)

If you intend to store the text using the SWN (Save With Number) command, you must mark the file to be somewhat larger. This is because the count of characters returned by the LASTLINE command does not include the number of bytes needed to store edit line numbers (the SWN command saves edit line numbers along with the text). If the SWN command is used to store the text on magnetic tape, you must mark a tape file large enough to hold the edit line numbers and text. To compute the number of bytes required, count approximately seven bytes per line of text, for storing edit line numbers; add the number of characters returned by the LASTLINE command; then add one byte for the end-of-file mark. The result of this calculation is the minimum number of bytes that should be specified when marking a file to hold edit line numbers and text.

For instance, if the LASTLINE command returns `100 Lines` and `1537 Characters`, and you want to save the current text using the SWN command, you should mark the magnetic tape file to have at least  $7 * 100 + 1537 + 1 = 2238$  bytes.

<sup>1</sup> The EDITOR inserts the colons when listing text on the display or another device. The colons appear only in the listing, and are not stored in the text buffer.

## The Number of Free Bytes in the Text Buffer

The last line of information printed on the display is the number of free (unused) bytes in the text buffer. For example, if the LASTLINE command returns the information 318 Free , there are 318 bytes available in the text buffer for storing edit line numbers and text. This helps you keep track of how much more text may be entered without overflowing the text buffer.

When calculating how much space a particular line of text will occupy in the text buffer, count one byte per text character, then add two extra bytes. The two extra bytes precede the line in the text buffer, and are used to store the edit line number. Even if the line is unnumbered, two bytes are still reserved in the text buffer, in case an edit line number is assigned later. Thus any line of text occupies a minimum of two bytes in the text buffer, and a blank line occupies exactly two bytes.

When calculating how much space text occupies in the text buffer, do not count end-of-record characters. End-of-record (CARRIAGE RETURN) characters are not stored in the text buffer. The two-byte pairs used to store edit line numbers serve to delimit the lines of text, so that end-of-record characters need not be stored in the buffer. When the EDITOR outputs text to a peripheral device, it recognizes an edit line number to be the beginning of a new line of text, and automatically inserts the end-of-record character before sending a line to the specified device.<sup>2</sup>

When the LASTLINE command returns the information 318 Free , you may actually only enter a maximum of 316 new text characters. Exactly how many text characters can be entered, depends on how many new lines, and therefore how many pairs of extra bytes, will be stored in the text buffer. For example, if 318 bytes are free in the text buffer, you might only be able to add 212 new text characters, if the 212 characters are entered as 53 lines of 4 characters each. This is because the 53 lines require  $2 * 53 = 106$  additional bytes of space in the text buffer for storing edit line numbers. In this case, the new text occupies  $212 + 106 = 318$  bytes, all of the remaining space in the text buffer.

## An Editing Example

The following examples show how the LASTLINE command provides information about the current contents of the text buffer.

<sup>2</sup> Although end-of-record characters are not stored in the text buffer, you must enter end-of-record characters from the keyboard or press RETURN to end the current line. Entering an end-of-record character marks the end of a line, and prompts the EDITOR to reserve two bytes in the text buffer to precede the next new line of text.

Environmental Commands  
LASTLINE

Example 1

```
CALL "EDITOR"  
  
LA  
  
0 Lines  
0 Characters  
30210 Free
```

Example 2

```
LA  
1354  
1354 Lines  
28838 Characters  
18 Free
```

Example 3

```
LIST  
:  
: DATA DCTAB(1)/1H /  
: DATA DCTAB(2)/1H /  
: DATA DCTAB(3)/1H /  
: DATA DCTAB(4)/1H /  
: DATA DCTAB(5)/1H /  
: DATA DCTAB(6)/1H /  
: DATA DCTAB(7)/1H /  
: DATA DCTAB(8)/1H /  
: DATA DCTAB(9)/1H /  
: DATA DCTAB(10)/1H /
```

```
LA  
+9  
10 Lines  
251 Characters  
29949 Free
```

```
R 2000,100,0
```

```
LA  
2900  
10 Lines  
251 Characters  
29949 Free
```

```
I 3000  
:  
: DATA DCTAB(11)/1H /  
: DATA DCTAB(12)/1H /
```

```
LA  
2900  
12 Lines  
303 Characters  
29895 Free
```

In Example 1, the LASTLINE command is executed immediately after the EDITOR is called. Since the text buffer is empty, the EDITOR prints a blank line on the display, then returns the information that there are no lines currently in the text buffer, no characters to be stored on magnetic tape, and 30210 unused bytes remaining in the text buffer.

In Example 2 the LASTLINE command is executed while the text buffer contains a piece of text. After the command is executed, the EDITOR returns the following information:

- The last line of text has edit line number 1354.
- The current text consists of 1354 lines.
- There are 28838 characters to be stored on magnetic tape.
- 18 unused bytes remain in the text buffer.

In Example 3 the LASTLINE command is executed before and after a RENUMBER command, then is executed a third time after new text is added to the text buffer.

Before the command LA is entered, a listing shows that the text buffer contains ten unnumbered lines of text. Then the LASTLINE command is executed, and the EDITOR returns the information that the last line of text is edit line +9 (the same as edit line 0+9); that there are 10 lines of text consisting of 251 characters; and that there are 29949 unused bytes remaining in the text buffer.

Next a RENUMBER command is executed, and the command LA is entered and executed again. The information returned by the LASTLINE command is the same as before, except for the edit line number of the last line of text. The last line now has edit line number 2900, because the RENUMBER command assigned the numbers 2000, 2100, 2200, ... to the text.

The number of free bytes in the text buffer does not change after a RENUMBER command is executed. This is because two bytes per line have already been automatically reserved in the text buffer, for later use in storing edit line numbers.

Next the INSERT command is used to add two new lines to the text buffer. The new lines are inserted at the end of the text, because the command I 3000 specifies a destination beyond the last line of text (refer to the INSERT command for an explanation of how to specify a destination).

Finally, the LASTLINE command is executed again. Because of the two unnumbered lines of text inserted after line 2900, the number of lines has changed from 10 to 12; the number of characters to be stored on magnetic tape has increased to 303; and the number of unused bytes in the text buffer is now 29895.

## Environmental Commands

### LASTLINE

The number of characters to be stored on magnetic tape has increased to 303 because of the two new lines of text. Each line contributes 25 new text characters (including the 7 blank characters entered by pressing the SPACE bar), and one end-of-record character.

Thus there are  $25 * 2 + 2 = 52$  new characters to be stored on magnetic tape.

The number of free bytes in the text buffer has decreased to 29895 because of the new lines of text. Each of the two new lines reserves 2 bytes in the text buffer for later use in storing an edit line number, and 25 bytes to store the new text characters. Thus the number of free bytes in the text buffer has decreased by  $2 * 2 + 2 * 25 = 54$  bytes.

## The LOWERCASE Command

<p><b>Syntax Form:</b></p> <p>LO</p> <p><b>Descriptive Form:</b></p> <p>LOWERCASE</p>
---

### PURPOSE

The LOWERCASE command enables the EDITOR to distinguish between lowercase characters a-z and their uppercase equivalents A-Z during searching and sorting operations (NLSEARCH, SEARCH, REVSORT, and SORT commands). The LOWERCASE command also prepares the EDITOR to change uppercase characters into lowercase characters if the CASE command is executed.

### EXAMPLE

LO

### EXPLANATION

The LOWERCASE command has no parameters. Only the keyword LO is entered from the keyboard, as shown in the example above. Executing the LOWERCASE command has no immediate effect on the contents of the text buffer. Instead, a system environmental parameter is assigned a value that prepares the EDITOR for subsequent commands.

Changing the value of the environmental parameter by executing the LOWERCASE command, sets a "lowercase flag." Once the lowercase flag is set, the EDITOR is able to recognize the difference between lowercase and uppercase characters in the text buffer. For example, when the lowercase flag is set, the EDITOR recognizes the character b to be different from the character B. This affects the operation of the commands NLSEARCH, SEARCH, REVSORT, SORT, and CASE.

## **Environmental Commands**

### **LOWERCASE**

#### **The NLSEARCH and SEARCH Commands**

When the lowercase flag is set, a lowercase character in the text buffer cannot "match" or satisfy a search for an uppercase character specified in a target string. Likewise, an uppercase character in the text cannot "match" or satisfy a search for a lowercase character.

For example, when the lowercase flag is set the command `NL "A", ""` deletes all occurrences of the character `A` (ASCII equivalent 65) from the text buffer. Occurrences of the character `a` (ASCII equivalent 97) are not changed by the command. Likewise, when the lowercase flag is set the command `NL "a", ""` deletes all occurrences of the character `a` (ASCII equivalent 97), and ignores the character `A` (ASCII equivalent 65).

#### **The REVSORT and SORT Commands**

Setting the lowercase flag enables the EDITOR to distinguish between lowercase and uppercase characters during sorting operations. For example, if a `REVSORT` or `SORT` command is executed while the lowercase flag is set, a line containing the character `B` in a specified character position is not treated the same as a line containing the character `b` in the same position. This is because the uppercase character `B` does not have the same ASCII code value as its lowercase equivalent `b`. (Refer to the ASCII code chart provided in Appendix B.)

For an illustration of how `REVSORT` and `SORT` operate after the `LOWERCASE` command is executed, refer to "An Editing Example" on the following pages.

#### **The CASE Command**

While the lowercase command is set, executing a `CASE` command causes uppercase characters `A-Z` to be replaced by their lowercase equivalents `a-z`. The reverse operation is performed if the uppercase flag is set. (Refer to the explanation of the `UPPERCASE` command.)

#### **Default Value**

Calling the EDITOR automatically sets the lowercase flag. The lowercase flag is set until the `UPPERCASE` command is executed. The `UPPERCASE` command sets an "uppercase flag," by changing the value of the environmental parameter. (Refer to the explanation of the `UPPERCASE` command.)

Because the lowercase flag is set by default, you need only execute the `LOWERCASE` command if the `UPPERCASE` command has been executed since the last time the EDITOR was called.

### An Editing Example

The following examples show how the SEARCH, CASE, SORT, and REVSORT commands operate while the lowercase flag is set.

#### Example 1

```
LIST
:Abernathy, Tod
:Brockway, Marius E.
:Carmichael, David
:Ellis, Terry L.
:Foster, Alice
:Hillstrom, A.
:Kearney, John D.
:Keller, Suzanne
:Lentz, John F.
:Pollock, Robert
:Siebold, William B.
:Taylor, Owen
:Zimmerman, Neil D.

LO
S "A"
:Abernathy, Tod
:Foster, Alice
:Hillstrom, A.

S "a"
:Abernathy, Tod
:Brockway, Marius E.
:Brockway, Marius E.
:Carmichael, David
:Carmichael, David
:Carmichael, David
:Kearney, John D.
:Keller, Suzanne
:Siebold, William B.
:Taylor, Owen
:Zimmerman, Neil D.

CASE
LIST
:abernathy, tod
:brockway, marius e.
:carmichael, david
:ellis, terry l.
:foster, alice
:hillstrom, a.
:kearney, john d.
:keller, suzanne
:lentz, john f.
:pollock, robert
:siebold, william b.
:taylor, owen
:zimmerman, neil d.
```



Environmental Commands  
LOWERCASE

Example 2

```
LO
LIST
:A
:a
:B
:b
:C
:c
:D
:d
:E
:e
:F
:f
```

```
SORT,,1
```

```
LIST
:A
:B
:C
:D
:E
:F
:a
:b
:c
:d
:e
:f
```

Example 3

```
LO
LIST
:A
:a
:B
:b
:C
:c
:D
:d
:E
:e
:F
:f
REUSORT,,1
```

(continued on next page)

```
LIST  
:f  
:e  
:d  
:c  
:b  
:a  
:F  
:E  
:D  
:C  
:B  
:A
```

Example 1 illustrates how the SEARCH and CASE commands operate while the lowercase flag is set. An initial listing shows that the text buffer contains a list of thirteen names. The command LO is executed to ensure that the lowercase flag is set, then the command S "A" tells the EDITOR to search the text and list lines found to contain the character A .

Because the lowercase flag is set, the EDITOR is able to distinguish between lowercase and uppercase characters, and searches only for the uppercase character A (ASCII value 65). A lowercase character a in the text does not "match" the target string, and so does not satisfy the search.<sup>3</sup>

After the command S "A" is executed, the EDITOR lists three lines found to contain the character A .

Next, the command S "a" is executed. Again the EDITOR recognizes the difference between lowercase and uppercase characters in the text, and searches only for the lowercase character a (ASCII value 97). After the command is executed, eight lines found to contain the character a are printed on the display. Several lines are listed more than once, because they contain more than one occurrence of the character a .

Next the command CASE is executed. Because the lowercase flag is set, the EDITOR replaces all uppercase characters in the text with their lowercase equivalents. A new listing shows that the buffer now contains only lowercase characters.

Example 2 illustrates how the SORT command operates while the lowercase flag is set. As in the previous example, the command LO is executed to make sure that the lowercase flag is set. Then a listing shows uppercase characters A-F and the lowercase equivalents a-f in lines consisting of one character each. The command S0,,1 tells the EDITOR to rearrange the lines according to the ASCII value of the character found in the first position in each line. Lines are to be rearranged so that the ASCII code values are in increasing order.

<sup>3</sup>This is not the way the SEARCH command operates if the uppercase flag is set.

## Environmental Commands

### LOWERCASE

Because the lowercase flag is set, the EDITOR is able to distinguish lowercase characters from uppercase characters. After the SORT command is executed, a new listing shows that lines consisting of lowercase characters have been moved to a lower location in the text buffer than those consisting of the uppercase equivalents. This is because lowercase characters have higher ASCII code values than uppercase characters. (Refer to the ASCII code chart provided in Appendix B.)

Example 3 is similar to Example 2, but executes the command `REV,,1`. This time the lines are rearranged in decreasing ASCII code value. Again, lowercase and uppercase characters are seen as different; but this time, uppercase characters are moved to a lower location in the text buffer, because their ASCII values are lower than those of the lowercase equivalents.

## The RENUMBER Command

### Syntax Form:

R [ edit line number ] [ , [ numeric constant ] [ , [ edit line number ] ] ]

### Descriptive Form:

RENUMBER [ new starting line number ] [ , [ increment between new line numbers ] [ ,  
[ line in the current text where renumbering is to begin ] ] ]

## PURPOSE

The RENUMBER command assigns a new set of edit line numbers to some or all of the lines in the current text. The parameters for the command include a new starting edit line number, the increment between new edit line numbers, and the line in the current text where renumbering is to begin. If none of these parameters are specified, the command assigns new edit line numbers 1, 2, 3, ... and renumbers the entire text.

## EXAMPLES

```
R  
R 3,  
R ,5,  
R ,100  
R 100,10  
R 100,,75  
R ,2,200  
R 50,5,100
```

## **EXPLANATION**

The parameters of the RENUMBER command specify which lines of text are to be renumbered, and how they are to be renumbered. The first parameter is the first new edit line number, the second parameter is the increment between new edit line numbers, and the third parameter is the line in the current text where renumbering is to begin. For example, the command `R 50,5,100` listed in the examples above renumbers the text buffer, beginning with the line currently number 100. New edit line numbers are assigned in increments of 5, starting with new edit line number 50.

The RENUMBER command is an important and useful command. Assigning edit line numbers to the text makes it easy to refer to specific lines in subsequent EDITOR commands. Unless edit line numbers are assigned, lines must be referred to using offsets, as in `0+10` or `250+3`. Although you are free to use offsets to identify lines of text, it is simpler to refer to lines using edit line numbers assigned by the RENUMBER command.

For this reason, you will probably want to execute a RENUMBER command immediately after creating a new piece of text using the INSERT command, or after inputting the contents of a tape file into the text buffer.

Once a RENUMBER command has been executed, editing the text may cause some lines to lose their edit line numbers. For instance, lines affected by the MOVE and COPY commands, or rearranged by the SORT and REVSORT commands, are stripped of their edit line numbers. When this happens, you may execute a RENUMBER command again and obtain a new set of edit line numbers. The RENUMBER command can be executed as often as needed during editing, to help avoid confusion and keep the lines numbered.

## **Default Values**

Just as for the COPY and MOVE commands, the three parameters of the RENUMBER command are optional, and may be omitted or entered in any combination. Some examples are listed above.

When the new starting line number is omitted, the RENUMBER command assigns new edit line number 1 to the first renumbered line of text. When the increment between new line numbers is omitted in the command, the value used for the increment is 1. If the line where renumbering is to begin is not specified, the entire text is renumbered.

For instance, in the examples listed above, the command `R` renumbers all of the current text, assigning new edit line numbers 1, 2, 3, ... . The command `R ,5,` also renumbers the entire text, but assigns new line numbers 5, 10, 15, ... . The command `R 100,10` renumbers the entire text with the following numbers: 100, 110, 120, ... .

The command `R ,,100` renumbers the text from the line currently numbered 100 on. The new edit line numbers assigned are 1, 2, 3, ... . `R 100,,75` renumbers the text from line 75 on, assigning the line numbers 100, 101, 102, ... . Finally `R ,2,2000` renumbers from line 2000 on, assigning the numbers 1, 3, 5, ... .

### Notes on the Command Syntax

When an edit delimiter is the last entry in a RENUMBER command, it may be omitted without altering the meaning of the command. That is, the commands R 3,, and R 3 are equivalent. The command R ,5, is the same as R ,5 and R 10,10, may be shortened to R 10,10 .

### The Line # Too Large Error Message

If a RENUMBER command tries to assign a number larger than 9999 to one or more lines of text, the EDITOR returns a Line # Too Large error message. An example is shown below:

```
LIST
  1: REM ** SUBROUTINE
  2: PRINT "ENTER CONSTANTS:"
  3: FOR I= TO N
  4: PRINT "B(";I;")=";
  5: INPUT B(I)
  6: NEXT I
  7: PRINT "END OF INPUT"
  8: RETURN

R 9500,100,1
EDITOR ERROR
Line # Too Large - error number 132
```

```
LIST
9500: REM ** SUBROUTINE
9600: PRINT "ENTER CONSTANTS:"
9700: FOR I= TO N
9800: PRINT "B(";I;")=";
9900: INPUT B(I)
: NEXT I
: PRINT "END OF INPUT"
: RETURN
```

In the above example, the command R 9500,100,1 tells the EDITOR to renumber the current text, assigning edit line numbers in increments of 100, and assigning new edit line number 9500 to the first line of text.

When the RENUMBER command is executed, the EDITOR removes all previous edit line numbers, and begins assigning the new numbers: 9500, 9600, 9700, ... . However, after assigning line number 9900 to the fifth line of text, the EDITOR stops. The next line number in the sequence would be 10000, but edit line numbers can only be four digits long. The EDITOR returns the message Line # Too Large , and stops executing the RENUMBER command.

## Environmental Commands

### RENUMBER

A new listing of the text shows that the EDITOR has renumbered the text as directed, until the next edit line number to be assigned is more than four digits long. To obtain an edit line number for every line of text, you may execute another RENUMBER command that specifies either a smaller increment, or a smaller first new edit line number.

### An Editing Example

The following example shows the RENUMBER command being used to assign new edit line numbers.

#### Example 1

```
LIST
      :100 FOR P=1 TO N+1
      :110 T=A(K,P)
      :120 A(K,P)=A(L,P)
      :130 A(L,P)=T
      :140 NEXT P
```

R

```
LIST
  1:100 FOR P=1 TO N+1
  2:110 T=A(K,P)
  3:120 A(K,P)=A(L,P)
  4:130 A(L,P)=T
  5:140 NEXT P
```

R 40,,

```
LIST
 40:100 FOR P=1 TO N+1
 41:110 T=A(K,P)
 42:120 A(K,P)=A(L,P)
 43:130 A(L,P)=T
 44:140 NEXT P
```

R 50,10,

```
LIST
 50:100 FOR P=1 TO N+1
 60:110 T=A(K,P)
 70:120 A(K,P)=A(L,P)
 80:130 A(L,P)=T
 90:140 NEXT P
```

(continued on next page)

```
R 600,5,50

LIST
600:100 FOR P=1 TO N+1
605:110 T=A(K,P)
610:120 A(K,P)=A(L,P)
615:130 A(L,P)=T
620:140 NEXT P

R 10,,600

LIST
10:100 FOR P=1 TO N+1
11:110 T=A(K,P)
12:120 A(K,P)=A(L,P)
13:130 A(L,P)=T
14:140 NEXT P
```

Five RENUMBER commands are executed in the example shown above. Each time, a new listing of the text shows how the text has been renumbered. The command R assigns numbers 1, 2, 3, 4, and 5 to the previously unnumbered lines. Next, R 40,, rennumbers all the lines, using an increment of 1 by default and assigning 40 as the first new edit line number.

The command R 50,10, rennumbers the lines using 10 as the increment, and assigns edit line number 50 to the first line of text. Then R 600,5,50 rennumbers the text from the current line 50 on, assigning to line 50 the new edit line number 600, and renumbering the rest of the lines in increments of 5.

Finally, the command R 10,,60 rennumbers all the lines, assigning 10 as the first new edit line number, and using an increment of 1 by default.

### Special Uses for the RENUMBER Command

When using the EDITOR to create a BASIC program, you may use the RENUMBER and SWN (Save With Number) commands together to make BASIC program line numbers. This is done by entering BASIC statements from the keyboard without program line numbers, then executing a RENUMBER command such as R 100,10,0 . The command causes edit line numbers 100, 110, 120, ... to appear before the colon in each line of text.

Next, open a magnetic tape file and execute a SWN command. The SWN command saves the edit line number along with each statement. The next time the file is brought back into the text buffer, numbers previously assigned as *edit* line numbers, now appear as *program* line numbers preceding each statement. This method saves you the trouble of entering BASIC program line numbers along with each statement.

For an illustration of the procedure described above, refer to "An Editing Example" in the explanation of the SWN command.



## NOTES

## The UPPERCASE Command

**Syntax Form:**

U

**Descriptive Form:**

UPPERCASE

### PURPOSE

The UPPERCASE command causes the EDITOR to treat lowercase characters a-z in the text buffer as uppercase characters A-Z during searching and sorting operations (NLSEARCH, SEARCH, REVSORT, and SORT commands). The UPPERCASE command also prepares the EDITOR to change lowercase characters into uppercase characters if the CASE command is executed.

### EXAMPLE

U

### EXPLANATION

The UPPERCASE command has no parameters. Only the keyword U is entered from the keyboard, as shown in the example above. Executing the UPPERCASE command has no immediate effect on the text buffer. Instead, a system environmental parameter is assigned a value that prepares the EDITOR for subsequent commands.

Changing the value of the environmental parameter by executing the UPPERCASE command disables the lowercase flag, and sets an "uppercase flag." (Refer to the LOWERCASE command for an explanation of the lowercase flag.) Once the uppercase flag is set, the EDITOR perceives all characters in the text buffer to be uppercase characters. For example, when the uppercase flag is set the EDITOR considers a lowercase b found in the text to be the same as an uppercase B . This affects the operation of the NLSEARCH, SEARCH, REVSORT, SORT, and CASE commands.

## **Environmental Commands**

### **UPPERCASE**

#### **The NLSEARCH and SEARCH Commands**

When the uppercase flag is set, a lowercase character in the text buffer "matches" or satisfies a search for the equivalent uppercase character specified in a target string. Conversely, since all characters in the text are perceived as being uppercase, no character in the text can "match" or satisfy a search for a lowercase character.

For example, when the uppercase flag is set the command `NL "A", "*" *` deletes all occurrences of the character `A` (ASCII equivalent 65) from the text buffer. Occurrences of the character `a` are also deleted from the text buffer, because the EDITOR considers every `a` to be identical to `A`. Conversely, the command `NL "a" *` has no effect on the text, because the EDITOR sees no lowercase characters while the uppercase flag is set.

#### **The REVSORT and SORT Commands**

When the uppercase flag is set, the EDITOR considers all characters in the text buffer to be uppercase characters. If a REVSORT or SORT command is executed while the uppercase flag is set, a line of text containing the character `b` in a specified character position is treated the same as a line containing the character `B` in the same position. This is because when the uppercase flag is set, the characters `b` and `B` are treated as having the same ASCII code value, 66.

For an illustration of how the REVSORT and SORT commands operate after the UPPERCASE command is executed, refer to "An Editing Example" on the following pages.

#### **The CASE Command**

While the uppercase flag is set, executing a CASE command causes lowercase characters a-z to be replaced by their uppercase equivalents A-Z. This is the inverse of the function performed by the CASE command if the lowercase command is set.

#### **Default Value**

Calling the EDITOR automatically sets the lowercase flag. To set the uppercase flag, you must execute the UPPERCASE command.

#### **An Editing Example**

The following examples show how the SEARCH, CASE, SORT, and REVSORT commands operate while the uppercase flag is set. The examples are analogous to those used to illustrate the effect of setting the lowercase flag. To compare the results shown below with the results when the lowercase flag is set, refer to "An Editing Example" in the explanation of the LOWERCASE command.

Example 1

LIST

1:Abernathy, Tod  
2:Brockway, Marius E.  
3:Ellis, Terry L.  
4:Foster, Alice  
5:Hillstrom, A.  
6:Keller, Suzanne  
7:Lentz, John F.  
8:Pollock, Robert  
9:Siebold, William B.  
10:Taylor, Owen

U

S "a"

S "A"

1:Abernathy, Tod  
1:Abernathy, Tod  
2:Brockway, Marius E.  
2:Brockway, Marius E.  
4:Foster, Alice  
5:Hillstrom, A.  
6:Keller, Suzanne  
9:Siebold, William B.  
10:Taylor, Owen

CASE

LIST

1:ABERNATHY, TOD  
2:BROCKWAY, MARIUS E.  
3:ELLIS, TERRY L.  
4:FOSTER, ALICE  
5:HILLSTROM, A.  
6:KELLER, SUZANNE  
7:LENTZ, JOHN F.  
8:POLLOCK, ROBERT  
9:SIEBOLD, WILLIAM B.  
10:TAYLOR, OWEN

Environmental Commands  
UPPERCASE

Example 2

```
LIST  
:a  
:b  
:c  
:d  
:e  
:f  
:A  
:B  
:C  
:D  
:E  
:F
```

```
U  
SO,,1
```

```
LIST  
:a  
:A  
:b  
:B  
:c  
:C  
:d  
:D  
:e  
:E  
:f  
:F
```

Example 3

```
LIST  
:a  
:b  
:c  
:d  
:e  
:f  
:A  
:B  
:C  
:D  
:E  
:F
```

```
U  
REU,,1
```

(continued on next page)

```
LIST
:f
:F
:e
:E
:d
:D
:c
:C
:b
:B
:a
:A
```

Example 1 illustrates how the SEARCH and CASE commands operate while the uppercase flag is set. An initial listing shows that the text buffer contains a list of ten names. The command U is executed to set the uppercase flag, then the command S "a" tells the EDITOR to search the text and list lines found to contain the character a .

Because the uppercase flag is set, the EDITOR perceives all characters in the text to be uppercase characters, and so is unable to find any occurrences of the character a (ASCII value 97). Since no occurrences of the target string are found, no lines are listed on the display after the SEARCH command is executed.

Next the command S "A" tells the EDITOR to search the text and list lines found to contain the character A . Because the uppercase flag is set, the EDITOR does not distinguish between lowercase and uppercase characters in the text. Both the lowercase character a and the uppercase character A "match" the target string and satisfy the search.

After the command is executed, the EDITOR lists seven lines found to contain the characters A or a . Lines 1 and 2 are printed twice, because they contain two occurrences of the target string.

Next the command CASE is executed. Because the uppercase flag is set, the EDITOR replaces all lowercase characters in the text with their lowercase equivalents. A new listing shows that the text buffer now contains only uppercase characters.

Example 2 illustrates how the SORT command operates while the uppercase flag is set. As in the previous example, the command U is executed to set the uppercase flag. Then a listing shows uppercase characters A-F and their lowercase equivalents a-f in lines consisting of one character each. The command S0,,1 tells the EDITOR to rearrange the lines according to the ASCII value of the character found in the first position in each line. Lines are to be rearranged so that the ASCII code values are in increasing order.

## Environmental Commands

### UPPERCASE

Because the uppercase flag is set, the EDITOR treats lowercase characters in the text as uppercase characters. After the SORT command is executed, a new listing shows that lines consisting of lowercase characters are next to those consisting of their uppercase equivalents. This is because lowercase characters are treated as having the same ASCII code values as their uppercase equivalents.

Example 3 is similar to Example 2, but executes the command `REV,,1`. This time the lines are rearranged in decreasing ASCII code value. Again, lowercase characters are seen as uppercase: a new listing shows that each line consisting of a lowercase character is next to the line consisting of its uppercase equivalent.

## The # = Command (A Character to mean "DIGIT")

**Syntax Form:**

```
# = [ ASCII character ]
```

### PURPOSE

The # = command assigns the meaning "any digit 0 through 9" to a specified ASCII character. Once the new meaning is assigned, the specified ASCII character may be used in the target string of any SEARCH or NLSEARCH command. Immediately after the EDITOR is called, the meaning "any digit 0 through 9" is assigned to the character # by default.

### EXAMPLES

```
# =
```

```
# = *
```

```
# = |
```

```
# = #
```

### NOTE

*The most important aspect of the # = command is its effect on subsequent searching operations. It may be helpful to be familiar with the NLSEARCH and SEARCH commands before reading about the # = command.*

### EXPLANATION

The # = command specifies an ASCII character that will represent any digit (0, 1, 2, 3, 4, 5, 6, 7, 8, or 9) in subsequent searching operations. After a # = command is executed, the specified ASCII character can be used to locate lines or strings that contain one or more digits. For example, after the command # = \* is executed, the character \* is assigned the special meaning "any digit 0 through 9," and can be used to locate digits in text.



## Environmental Commands

#=

### ASCII Character Symbols

Any ASCII character may be specified in a #= command except CR (ASCII 13) and SP (ASCII 32). CR and SP cannot be assigned the meaning "any digit": pressing RETURN ends the command, and spaces after the keyboard #= are ignored.

An ASCII chart is provided in Appendix B. As in BASIC, the symbol for each control character in the first or second column of the chart is entered by pressing the CTRL key and the corresponding character in the fifth or sixth column of the chart. For example, the symbol for BEL (ASCII 7) is entered as G, and the symbol for ESC (ASCII 27) is entered as [. The only exception is control character CR (ASCII 13), which cannot be represented by the symbol M. Pressing CTRL and M has the same effect as pressing the RETURN key, and does not cause the symbol M to appear on the display.

The symbol for the character RUBOUT (ASCII 127) is entered from the keyboard by pressing the RUBOUT CHARACTER overlay key.

### Locating Digits in the Text

The ASCII character specified in the #= command can be used in subsequent searching commands to locate digits in the text. If the character appears in the target string of a SEARCH or NLSEARCH command, the EDITOR understands the character to mean "any digit." When scanning the text to find a "match" for that character, the EDITOR looks for a 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. The first occurrence of one of these digits is considered to be a match and ends the search.

For example, after the command #=\* is executed, the character \* can be used in a target string to mean "any digit." When scanning the text to find a match for the target string, the EDITOR considers the first occurrence of a digit to match the character \*.

After the command #=\* is executed, the character \* may appear anywhere in a target string, and may appear more than once in a target string. For example, the command S "V\*" locates and lists lines that contain the character V followed by a digit (such as V0 or V1). The command S "\*\*\*A" searches for lines containing three digits followed by the character A (such as 100A or 201B).

#### NOTE

*The "all but" prefix should not immediately precede the "any digit" character in a target string. For example, if the character \_ currently has the meaning "all but" and the character # currently means "any digit," the character \_ should not immediately precede # in the target string of a SEARCH or NLSEARCH command. (Refer to the explanation of the \_= command.)*

Once a character is assigned the meaning "any digit," only a digit can satisfy a search for that character. For example, after the command `#=*` is executed, the command `S "*"` looks only for digits in the text. Occurrences of the character `*` do not satisfy the search.

The character specified in the `#=` command represents "any digit" until the RETURN TO BASIC key is pressed or another `#=` command is executed that specifies a different ASCII character. Pressing the RETURN TO BASIC key and calling the EDITOR again, has the same effect as executing the command `#=#` (refer to "Default Values" in this explanation).

**Syntax Errors.** The ASCII character that currently represents "any digit" cannot appear in the replacement string of a NLSEARCH and Replace String or SEARCH and Replace String command, or a syntax error occurs. For example:

```
#=*
S "U0","U*"
EDITOR ERROR
Syntax - error number 138
S "U0","U*"
```

The syntax error shown in the example occurs because the character assigned the meaning "any digit" appears in the target string of a SEARCH and Replace String command.

Although the character currently assigned the meaning "any digit" cannot be used in a replacement string, you may insert the character into the text during line editing and while entering new text from the keyboard.

## Default Values

Calling the EDITOR assigns the meaning "any digit" to the character `#` by default. Until a `#=` command is executed that specifies a different ASCII character, the EDITOR understands `#` to mean "digit" in the target string of a SEARCH or NLSEARCH command.

When the parameter of the `#=` command is omitted, no character is assigned the meaning "any digit." That is, entering `#=` and pressing RETURN cancels the previously chosen "any digit" character, and makes no new assignment. You should execute the command `#=` if you want to make sure that no ASCII character means "digit" in a target string. At any time after the command `#=` is executed, you are free to choose a "digit" character again by executing a command such as `#=#` or `#=* .`

## Environmental Commands

#=

### When to Execute the # = Command

Because calling the EDITOR assigns the character # as the "any digit" character, you must execute a # = command that specifies a character other than # if you want to search for occurrences of the symbol # in the text.

### An Editing Example

The following example illustrates the use of the # = command.

#### Example 1

```
LIST
:1220 PRINT "ENTER # OF VALUES:";
:1230 INPUT X
:1240 GOSUB 1660
:1250 IF E=1 THEN 1440
:1260 L=E
:1270 N=0
:1280 N1=0
:1290 GO TO 1030
:1300 E8=2
:1310 GOSUB 2630
:1320 GO TO 1050
:1330 B(4)=B(4)+1

S " #"
:1240 GOSUB 1660
:1250 IF E=1 THEN 1440
:1290 GO TO 1030
:1310 GOSUB 2630
:1320 GO TO 1050

#=*

S "*"
:1240 GOSUB 1660
:1250 IF E=1 THEN 1440
:1290 GO TO 1030
:1310 GOSUB 2630
:1320 GO TO 1050

S "=*"
:1250 IF E=1 THEN 1440
:1270 N=0
:1280 N1=0
:1300 E8=2
```

(continued on next page)

```

S " #"
  :1220 PRINT "ENTER # OF VALUES:";

#=#

S "TO #"
  :1290 GO TO 1030
  :1320 GO TO 1050

#=

S " #"
  :1220 PRINT "ENTER # OF VALUES:";

```

Example 1 shows three `#=` commands and their effect on subsequent searching operations. The initial listing in the example shows that the text consists of twelve BASIC program statements. At the time of the listing, the character `#` is the "any digit" character by default: no `#=` commands have been previously executed.

Then the command `S " #"` is executed. Because the character `#` currently represents "any digit," the EDITOR searches the text for lines containing a digit preceded by a blank space.

The EDITOR finds four occurrences of the target string. A listing appears on the display of the lines found to contain a digit preceded by a space. The four listed lines are the BASIC program statements that specify a "destination" line number: `GOSUB`, `IF...THEN`, and `GO TO` statements. The four occurrences of the target string are shaded below:

```

LIST
  :1240 GOSUB 1660
  :1250 IF E=1 THEN 1440
  :1290 GO TO 1030
  :1310 GOSUB 2630
  :1320 GO TO 1050

```

Next the command `#=*` is executed. After the command is executed, the symbol `*` is the "any digit" character, and the symbol `#` no longer has any special meaning.

Because `*` is the current "digit" character, the command `S " *"` searches for a digit preceded by a space, and produces the same results as the previous command `S " #"` did when `#` was the "digit" character. The command `S "=*"` searches for a digit preceded by the symbol `=`, and lists four lines found to contain equalities.

Next the command `S " #"` is executed a second time. This time, the character `#` has no special meaning, so the EDITOR searches for the symbol `#` preceded by a space. One occurrence of the target string is found and the line in which it occurs is listed on the display.

## Environmental Commands

`#=`

Then the command `#=#` is executed to reassign the meaning "any digit" to the character `#`. The subsequent command `S "TO #"` specifies the character `#` in a target string to mean "digit." The command locates and lists the two GO TO statements of the program.

Finally, the command `#=` is executed. The command `#=` cancels the previous assignment `#=#`, and makes no new assignment. No ASCII character represents "any digit." The command `S "#"` locates and lists the one line that contains the symbol `#`, illustrating that the character `#` no longer means "any digit."

## The ~ = Command (A "WILDCARD" Character)

### Syntax Form:

```
~ = [ ASCII character ]
```

### PURPOSE

The ~ = command assigns the meaning "wildcard" to a specified ASCII character. Once the new meaning is assigned, the specified ASCII character may be used in the target or replacement string of a SEARCH or NLSEARCH command. Immediately after the EDITOR is called, the meaning "wildcard" is assigned to the character ~ by default.

### EXAMPLES

```
~ =
~ = ?
~ = *
~ = ~
```

### NOTE

*The most important aspect of the ~ = command is its effect on subsequent searching operations. It may be helpful to be familiar with the NLSEARCH and SEARCH commands before reading about the ~ = command.*

### EXPLANATION

The ~ = command specifies an ASCII character to be used as a "wildcard" character in subsequent searching operations. For example, the command ~ = ? specifies the character ? to be a "wildcard" character for subsequent searching operations.

The wildcard character may appear in the target string or replacement string of a NLSEARCH or SEARCH command. When specified in a target string, the wildcard character has the special meaning "any character." When specified in a replacement string, the wildcard character means "no change."

## Environmental Commands

~=

The character specified in the ~= command is the wildcard character until the RETURN TO BASIC key is pressed or another ~= command is executed that specifies a different ASCII character. Pressing the RETURN TO BASIC key and calling the EDITOR again, has the same effect as executing the command =~~ (refer to "Default Values" in this explanation).

### ASCII Character Symbols

Any ASCII character may be specified in a ~= command except CR (ASCII 13) and SP (ASCII 32). CR and SP cannot be the wildcard character, because pressing RETURN ends the command, and spaces after the keyword ~= are ignored.

An ASCII chart is provided in Appendix B. As in BASIC, the symbol for each control character in the first or second column of the chart is entered by pressing the CTRL key and the corresponding character in the fifth or sixth column of the chart. For example, the symbol for BEL (ASCII 7) is entered as G, and the symbol for ESC (ASCII 27) is entered as [. The only exception is control character CR (ASCII 13), which cannot be represented by the symbol M. Pressing CTRL and M has the same effect as pressing the RETURN key, and does not cause the symbol M to appear on the display.

The symbol for the character RUBOUT (ASCII 127) is entered from the keyboard by pressing the RUBOUT CHARACTER overlay key.

### Using the Wildcard Character in a Target String

The ASCII character specified in the ~= command can appear in the target string of subsequent SEARCH or NLSEARCH commands. When used in the target string of a SEARCH or NLSEARCH command, the wildcard character means "any character." That is, any ASCII character in the text is a "match" for a wildcard character in the target string.

For example, after the command ~=? is executed, the character ? means "any character" when it appears in the target string of a SEARCH or NLSEARCH command. When scanning the text to find a match for the target string, the EDITOR considers any ASCII character to match the character ?. Even blank spaces entered by pressing the SPACE bar, CARRIAGE RETURN characters entered by pressing RETURN, and occurrences of the symbol ? in the text satisfy a search for the wildcard character ?.

The wildcard character may be specified anywhere in a target string, and may be specified more than once in a target string. For example, after the command ~=? is executed, the wildcard character ? means "any character" in commands such as S "B(?,L)" or S "B(K,?)" or S "(?,?)".

**NOTE**

*The "all but" prefix should not immediately precede the wildcard character in a target string. For example, if the character \_ currently means "all but" and the character ~ is the wildcard character, \_ should not immediately precede ~ in the target string of a SEARCH or NLSEARCH command. (Refer to the explanation of the \_= command.)*

**Using the Wildcard Character in a Replacement String**

The ASCII character specified in the ~=  
 command can appear in the replacement string of subsequent NLSEARCH and Replace String or SEARCH and Replace String commands. When used in a replacement string, the wildcard character means "no change." That is, a wildcard character in the replacement string tells the EDITOR that the character found in that position should not be overwritten during the replacement procedure. For example, if the character # means "digit" and ? is the wildcard character, the command NL "X#","V?" changes occurrences of X0 to V0 , X1 to V1 , X2 to V2 , and so on. Each time an occurrence of the target string is found in the text, the EDITOR overwrites the first character of the target string with the character V , and leaves the second character of the target string unchanged.

The wildcard character may be specified anywhere in a replacement string, and may be specified more than once in a replacement string. Also, the wildcard character may be specified in both the target and replacement string of a NLSEARCH and Replace String or SEARCH and Replace String command. This is a useful tool for specifying that no matter what ASCII character occupies a certain position in the target string, that character should remain unchanged by the replacement procedure.

For example, after the command ~=? is executed, the command NL "?33","?29" overwrites %33 with %29 , and @33 with @29 . Similarly, NL "GOTO~~~~","GOSUB~~~~" overwrites GO TO 1040 with GOSUB 1040 , GO TO 1580 with GOSUB 1580 , and so on.

A wildcard character in the replacement string actually indicates a *position* in the target string that is to remain unchanged. If the fourth character in a replacement string is the wildcard character, the fourth character in occurrences of the target string should be unchanged by the replacement procedure. For example, if ~ is the wildcard character, the command NL "31~0","1~0" replaces 3120 with 110 , *not* 120 .



## Environmental Commands

~=  
~

### Default Values

Calling the EDITOR assigns the character ~ to be the "wildcard" character by default. Until a ~=  
command is executed that specifies a different ASCII character, the EDITOR understands ~ to mean "any ASCII character" in the target string of a SEARCH or NLSEARCH command, and "no change" in the replacement string of a SEARCH or NLSEARCH command.

When the parameter of the ~=  
command is omitted, no wildcard character is assigned. That is, entering ~=  
and pressing RETURN cancels the previously chosen wildcard character, and makes no new assignment. You should execute the command ~=  
if you want to make sure that no ASCII character means "any digit" when used in a target string, or that no character means "no change" when used in a replacement string. At any time after the command ~=  
is executed, you are free to choose a wildcard character again by executing a command such as ~=  
or ~=? .

### When to Execute the ~= Command

Because calling the EDITOR assigns ~ as the wildcard character, you must execute a ~=  
command that specifies a character other than ~ if you want to search specifically for occurrences of the symbol ~ in the text, or if you want to use the character ~ in a replacement string without the special meaning "no change."

### An Editing Example

The following examples illustrate the use of the ~=  
command.

#### Example 1

```
LIST
: 1760 PRINT "IS THIS A CONTINUATION OF THE SAME PROBLEM?";
: 1770 INPUT A$
: 1780 IF A$="Y" THEN 2140
: 1790 GOSUB 430
: 1800 PRINT "IS DATA ON PROGRAM TAPE?";
: 1810 INPUT B$
: 1820 X=29
: 1830 IF B$="Y" THEN 1950
: 1840 E(?)=1
: 1850 PRINT "IS DATA TO BE READ FROM EXTERNAL DEVICE?";
: 1860 INPUT C$
: 1870 IF C$="N" THEN 1930
```

NL "~\$", "E\$"

(continued on next page)

```

LIST
: 1760 PRINT "IS THIS A CONTINUATION OF THE SAME PROBLEM?";
: 1770 INPUT E$
: 1780 IF E$="Y" THEN 2140
: 1790 GOSUB 430
: 1800 PRINT "IS DATA ON PROGRAM TAPE?";
: 1810 INPUT E$
: 1820 X=29
: 1830 IF E$="Y" THEN 1950
: 1840 E(?)=1
: 1850 PRINT "IS DATA TO BE READ FROM EXTERNAL DEVICE?";
: 1860 INPUT E$
: 1870 IF E$="N" THEN 1930

```

## Example 2

```

LIST
:500 REM ~ SUBROUTINE
:510 X0=0
:520 X1=1
:530 X2=X(1,1)
:540 PRINT "DO YOU NEED INSTRUCTIONS?";
:550 INPUT X$
:560 IF X$="Y" THEN 780

```

~=?

NL "X#", "U?"

```

LIST
:500 REM ~ SUBROUTINE
:510 U0=0
:520 U1=1
:530 U2=X(1,1)
:540 PRINT "DO YOU NEED INSTRUCTIONS?";
:550 INPUT X$
:560 IF X$="Y" THEN 780

```

```

S "~"
:500 REM ~ SUBROUTINE

```

## Environmental Commands

~ =

### Example 3

```
LIST
: 2000 REM ~~~ SUBROUTINE ~~~
: 2100 FOR I=1 TO L(10)
: 2200 T1(1)=T1(1)+B(1,I)
: 2300 T1(2)=T1(2)+B(2,I)
: 2400 T1(3)=T1(3)+B(3,I)
: 2500 T1(4)=T1(4)+B(4,I)
: 2600 NEXT I
: 2700 GOSUB 3790
: 2800 INPUT K
: 2900 B(1,L)=B(1,L)+D3*X1
: 3000 B(2,L)=B(2,L)+D3*X1*X1
: 3100 B(3,L)=B(3,L)+D3
```

NL "B(~,L)","B(~,K)"

```
LIST
: 2000 REM ~~~ SUBROUTINE ~~~
: 2100 FOR I=1 TO L(10)
: 2200 T1(1)=T1(1)+B(1,I)
: 2300 T1(2)=T1(2)+B(2,I)
: 2400 T1(3)=T1(3)+B(3,I)
: 2500 T1(4)=T1(4)+B(4,I)
: 2600 NEXT I
: 2700 GOSUB 3790
: 2800 INPUT K
: 2900 B(1,K)=B(1,K)+D3*X1
: 3000 B(2,K)=B(2,K)+D3*X1*X1
: 3100 B(3,K)=B(3,K)+D3
```

Example 1 illustrates the use of the wildcard character in a target string. The initial listing shows that the text consists of a portion of a BASIC program. At the time of the listing, the character ~ is the wildcard character by default: no ~ = commands have been previously executed.

Then the command NL "~\$", "E\$" is executed. Because the character ~ in a target string means "any character," the EDITOR looks for two-character strings consisting of any ASCII character immediately followed by \$ . Occurrences of the two-character string are overwritten by the string E\$ . After the command NL "~\$", "E\$" is executed, a new listing shows that occurrences of the variable names A\$ , B\$ , and C\$ have been replaced with the variable name E\$ .

Example 2 illustrates the use of the wildcard character in a replacement string. The initial listing shows several statements of a BASIC program. The command `~=?` assigns `?` as the wildcard character, then the command `NL "X#", "V?"` is executed.

At the time the command is executed, the character `#` means "any digit" : no `#=` commands have been previously executed. (Refer to the `#=` command for information about the "any digit" character.) Occurrences of such a two-character pair in the text are replaced by another two-character string consisting of the character `V` followed by the second character of the target string. After the command `NL "X#", "V?"` is executed, a new listing shows that occurrences of the variable names `X0` , `X1` , and `X2` have been replaced by the variable names `V0` , `V1` , and `V2` , respectively.

Finally the command `S "~"` is executed to illustrate that once the command `~=?` assigns the character `?` as the wildcard character, the character `~` no longer means "any character" when used in a target string. The command `S "~"` locates and lists the one line that contains the symbol `~` .

Example 3 illustrates the use of the wildcard character in both the target and replacement string. The initial listing shows that the text consists of a portion of a BASIC program. At the time of the listing `~` is the wildcard character: no `~=` commands have been previously executed.

Then the command `NL "B(~,L)", "B(~,K)"` is executed. Because `~` is the wildcard character, the EDITOR looks for a five-character string consisting of `B(` followed by any ASCII character, followed by `,L)` . Occurrences of such a five-character string are overwritten by a string that is similar but specifies `K)` instead of `L)` as the last two characters.

After the command is executed, a new listing shows that the variable names `B(1,L)` , `B(2,L)` , and `B(3,L)` have been changed to `B(1,K)` , `B(2,K)` , and `B(3,K)` , respectively.

## NOTES

## The \_ = Command (An "ALL BUT" Prefix)

### Syntax Form:

```
_ = [ ASCII character ]
```

### PURPOSE

The \_ = command assigns the meaning "all but" to a specified ASCII character. Once the new meaning is assigned, the specified ASCII character may be used as a prefix to another character in the target string of a SEARCH or NLSEARCH command. Immediately after the EDITOR is called, the meaning "all but" is assigned to the character \_ by default.

### EXAMPLES

```
_ =
_ = '
_ = +
_ = _
```

### NOTE

*The most important aspect of the \_ = command is its effect on subsequent searching operations. It may be helpful to be familiar with the NLSEARCH and SEARCH commands before reading about the \_ = command.*

### EXPLANATION

The \_ = command specifies an ASCII character that will mean "all but the following character" in the target string of subsequent searching commands. For example, the command \_ =' assigns the character ' to be used as an "all but" prefix in the target string of a SEARCH or NLSEARCH command.

Any ASCII character may be specified in a \_ = command except CR (ASCII 13) and SP (ASCII 32). The symbol for the character RUBOUT (ASCII 127) is entered from the keyboard by pressing the RUBOUT CHARACTER overlay key.

### Using the "All But" Prefix in a Target String

When the "all but" prefix precedes another character in the target string of a SEARCH or NLSEARCH command, the EDITOR looks for any ASCII character except the one immediately following the prefix. For example, after the command `_ = '`  assigns the character `'` to be the "all but" prefix, the command `S "'B"` tells the EDITOR to search for all ASCII characters except `B`. The command `S "A'0"` tells the EDITOR to locate all two-character strings consisting of `A` followed by any ASCII character except `0`.

The "all but" prefix may appear anywhere within a target string, but cannot be the last character in the target string (refer to "Syntax Errors" in this explanation). The "all but" prefix may appear more than once in a target string. For example, if `_` is the "all but" prefix, the command `S "5_3_2"` searches for three-character strings consisting of the character `5`, followed by any ASCII character except `3`, followed by any character except `2`. Strings such as `501` or `598` match the target string, but `532`, `538`, or `572` do not satisfy the search.

The "all but" prefix may appear in successive positions in a target string. However, even-numbered occurrences of the character in a set of successive "all but" prefixes do not have the special meaning "all but the character following this one." For example, if `X` is the "all but" prefix, the command `S "XXXA"` searches for a two-character string consisting of any ASCII character except `X` followed by any ASCII character except `A`. Likewise, the command `S "XXXX"` searches for a two-character string consisting of any character except `X` followed by any character except `X`.

#### NOTE

*The "all but" prefix should not immediately precede the "digit," wildcard, or "END-OF-RECORD" character in a target string. For example, if `_` is the "all but" prefix, `#` is the "any digit" character, `~` is the wildcard character and `]` is the "END-OF-RECORD" character, the EDITOR cannot interpret commands such as `S "_#"` or `S "_="` or `S "#"`.*

The character specified in the `_ =` command may be used in the replacement string of a subsequent SEARCH or NLSEARCH command. However, the character has no special meaning when used in a replacement string. For example, after the command `_ = -` is executed, the command `NL "100A","100-A"` replaces occurrences of the string `100A` with the string `100-A`.

The character specified in the `_ =` command is the "all but" prefix until the RETURN TO BASIC key is pressed or another `_ =` command is executed that specifies a different ASCII character. Pressing the RETURN TO BASIC key and calling the EDITOR again, has the same effect as executing the command `_ =_` (refer to "Default Values" in this explanation).

## Syntax Errors

The "all but" prefix must precede another character in the target string. Entering the "all but" prefix as the last character of the target string causes a syntax error. For example, if `_` is the "all but" prefix, attempting to execute `S "_"` or `S "A _"` causes a syntax error message to appear on the display.

## Default Values

Calling the EDITOR assigns the character `_` as the "all but" prefix by default. Until a `_ =` command is executed that specifies a different ASCII character, the EDITOR understands `_` to mean "all but the following character" in the target string of a SEARCH or NLSEARCH command.

When the parameter of the `_ =` command is omitted, no character is assigned to be the "all but" prefix. That is, entering `_ =` and pressing RETURN cancels the previously chosen "all but" prefix, and makes no new assignment. You should execute the command `_ =` if you want to make sure that no ASCII character means "all but the following character" in the target string. At any time after the command `_ =` is executed, you are free to choose an "all but" prefix again by executing a command such as `_ =_` or `_ ='`.

## When to Execute the `_ =` Command

Because calling the EDITOR assigns `_` as the "all but" prefix, you must execute a `_ =` command that specifies a character other than `_` if you want to search for occurrences of the symbol `_` in the text.

## An Editing Example

The following example illustrates the use of the `_ =` command.



\_ =

Example 1 LIST

```
:550 REM __ USER'S INPUT CONTINUES __
:560 REM
:570 PRINT "DO YOU WISH TO GO ON?";
:580 INPUT A$
:590 IF A$="N" THEN 1200
:600 PRINT "ENTER FIRST NUMBER:";
:610 INPUT A0
:620 PRINT "ENTER SECOND NUMBER:";
:630 INPUT A1
:640 DELETE A,B
:650 DIM A(N),B(N)
:660 FOR I=1 TO N
:670 A(I)=I
:680 NEXT I
```

S "A\_("

```
:580 INPUT A$
:590 IF A$="N" THEN 1200
:610 INPUT A0
:630 INPUT A1
:640 DELETE A,B
```

\_ =

NL "\_", "\*"

LIST

```
:550 REM ** USER'S INPUT CONTINUES **
:560 REM *****
:570 PRINT "DO YOU WISH TO GO ON?";
:580 INPUT A$
:590 IF A$="N" THEN 1200
:600 PRINT "ENTER FIRST NUMBER:";
:610 INPUT A0
:620 PRINT "ENTER SECOND NUMBER:";
:630 INPUT A1
:640 DELETE A,B
:650 DIM A(N),B(N)
:660 FOR I=1 TO N
:670 A(I)=I
:680 NEXT I
```

\_ ='

S "A'0"

```
:580 INPUT A$
:590 IF A$="N" THEN 1200
:630 INPUT A1
:640 DELETE A,B
:650 DIM A(N),B(N)
:670 A(I)=I
```

\_ =\_

S " / "

```
:550 REM ** USER'S INPUT CONTINUES **
```

Example 1 shows three `_ =` commands and their effect on subsequent `SEARCH` and `NLSEARCH` commands. The initial listing shows that the text consists of a portion of a BASIC program. At the time of the listing, the character `_` is the "all but" prefix: no `_ =` commands have been previously executed.

Then the command `_ =` is executed. Because no ASCII character is specified in the command, the EDITOR cancels the previous assignment of `_` as the "all but" prefix, and makes no new assignment. Because `_` is no longer the "all but" prefix, the command `NL "_ ", "*"` overwrites occurrences of the symbol `_` with the symbol `*`.

Next the command `_ ='` specifies the character `'` to be the "all but" prefix. Since the character `'` now means "any character but the following one," the command `S "A'0"` tells the EDITOR to locate and list lines containing a two-character string consisting of the character `A` followed by any ASCII character except `0`. The EDITOR locates and lists six lines found to contain a "match" for the target string. The "matches" are shown below:

```
A$
A1
A,
A(
```

Finally the command `_ =_` is executed to re-establish `_` as the "all but" prefix. Since `_` is now the "all but" prefix, the character `'` no longer has any special meaning, as illustrated by the fact that the command `S ""` locates and lists a line that contains the symbol `'`.

## NOTES

## The ]= Command (A Character to mean "END-OF-RECORD")

**Syntax Form:**

```
] = [ ASCII character ]
```

### PURPOSE

The ]= command assigns the meaning "END-OF-RECORD" to a specified ASCII character. Once the new meaning is assigned, the specified ASCII character string may be used to insert end-of-record characters (usually carriage returns) into the text. The character may also be used in target and replacement strings in the SEARCH and NLSEARCH commands. Immediately after the EDITOR is called, the meaning "END-OF-RECORD" is assigned to the character ] by default.

### EXAMPLES

```
]=
```

```
]=/
```

```
]=|
```

```
]=]
```

### NOTE

*An important aspect of the ]= command is its effect on subsequent searching operations. It may be helpful to be familiar with the SEARCH and NLSEARCH commands before reading about the ]= command.*

### EXPLANATION

The ]= command specifies an ASCII character to mean "END-OF-RECORD" during subsequent editing or searching operations. For example, the command ]=/ assigns the special meaning "END-OF-RECORD" to the character / .

## Environmental Commands

]=

The ASCII character specified in a ]= command means "END-OF-RECORD" when used in the target or replacement string of subsequent SEARCH or NLSEARCH commands. The character also means "END-OF-RECORD" if entered from the keyboard while the line buffer contains a recalled line (a line recalled from the text buffer to the line buffer by the SEARCH and Edit Line command or by the RECALL NEXT LINE/RECALL LINE or STEP PROGRAM keys).

Any ASCII character may be specified in a ]= command except CR (ASCII 13) and SP (ASCII 32). The character specified in the ]= command is the "END-OF-RECORD" character until the RETURN TO BASIC key is pressed or another ]= command is executed that specifies a different ASCII character. Pressing the RETURN TO BASIC key and calling the EDITOR again, has the same effect as executing the command ]=] (refer to "Default Values" in this explanation).

### The Difference Between the "END-OF-RECORD" Character and CR (ASCII 13)

The "END-OF-RECORD" character is used to locate, insert, or delete end-of-record characters. However, the "END-OF-RECORD" character does not represent the ASCII character CR (ASCII code 13). The distinction is as follows:

End-of-record characters are not stored in the text buffer. The two bytes that precede each line of text in the text buffer store the edit line number and also serve as a "flag" to mark the beginning of the line. Because the flag serves the same purpose as an end-of-record character, the character CR (ASCII 13) is not needed and not stored in the text buffer.

Pressing RETURN while the system is under EDITOR control sets the flag, but does not insert a CR into the text. Inputting a stored line from a peripheral device removes the CR (ASCII 13) character, and sets the flag: outputting the line disables the flag and reinserts the CR (ASCII 13) character.<sup>4</sup>

This means that the character CR (ASCII 13) can only occur in text brought in by an Input command that specifies a % symbol in the I/O address. If the alternate Input/Output format specifies an end-of-record character other than CR, the symbol M represents CR characters in listings of the inputted text. But even then, the "END-OF-RECORD" character represents the alternate end-of-record character, and cannot locate the character CR : that is, the symbol M does not satisfy a search for the "END-OF-RECORD" character.

In summary, the "END-OF-RECORD" character specified by the ]= command represents a flag and not CR (ASCII 13). Using the "END-OF-RECORD" character to locate, insert, or delete end-of-record characters, actually locates, sets, or disables a flag in the text buffer. For the sake of simplicity the term "end-of-record character" instead of "flag" will be used in the remainder of this discussion: but keep in mind that in this explanation, the term "end-of-record character" does not refer to the ASCII character CR.

<sup>4</sup>If the symbol % instead of @ appears in the I/O address of an Input/Output command, the EDITOR removes or inserts a previously specified "alternate" end-of-record character.

## Using the "END-OF-RECORD" Character in a Target String

The "END-OF-RECORD" character may be specified in the target string of a SEARCH or NLSEARCH command. When scanning the text for the target string, the EDITOR considers an end-of-record character to "match" the "END-OF-RECORD" character.

This allows you to locate lines that begin or end with a specified string. For example, after the command ]=/ is executed, the command S "/1650"\* locates and deletes lines that begin with the string 1650 . The command S "MD/", locates lines that end with the string MD and recalls the lines one by one to the line buffer for editing.

The "END-OF-RECORD" character can be specified anywhere in a target string. For example, if ] is the "END-OF-RECORD" character, the command S "]STOP]" locates and lists lines consisting of the string STOP .

The "END-OF-RECORD" character may appear in successive positions in a target string. For example, if ] is the "END-OF-RECORD" character, the command NL "]]"\* deletes blank lines. (Two successive end-of-record characters constitute a blank line. A blank line can be created by pressing RETURN twice while entering text from the keyboard. )

### NOTE

*The "all but" prefix should not immediately precede the "END-OF-RECORD" character in a target or replacement string. For example, if the character \_ currently means "all but" and the character ] is the "END-OF-RECORD" character, \_ should not immediately precede ] in the target or replacement string of a SEARCH or NLSEARCH command. (Refer to the explanation of the \_= command.)*

If the "END-OF-RECORD" character appears in the target string of a SEARCH and Delete Line or NLSEARCH and Delete Line command, the command deletes the line that contains the last character of the target string. For example, if ] is the "END-OF-RECORD" character, the command NL "0]1"\* deletes lines that begin with 1 and are immediately preceded by a line that ends with 0 .

However, if the "END-OF-RECORD" character is the last character of the target string, the NLSEARCH and Delete Line or SEARCH and Delete Line command deletes the line that contains the next to last character of the target string. For example, if ] is the "END-OF-RECORD" character the command NL "1650]" \* deletes lines that end with the string 1650 .

]=

### Using the "END-OF-RECORD" Character in a Replacement String

The "END-OF-RECORD" character can be specified in the replacement string of a SEARCH and Replace String or NLSEARCH and Replace String command. This allows you to insert or delete end-of-record characters. For example, if ] is the "END-OF-RECORD" character, the command S "STOP","STOP]" inserts an end-of-record character after the string STOP . The command S "P]A","PA" deletes end-of-record characters found immediately after the character P and before the character A .

The "END-OF-RECORD" character may appear anywhere in a replacement string, and may appear more than once in a replacement string. For example, if / is the "END-OF-RECORD" character, the command S "\*\*\*\*\*","//\*\*\*\*\*/" inserts two end-of-record characters before and after a string consisting of six successive \* characters.

The "END-OF-RECORD" character may appear in successive positions in a replacement string. For example, if ] is the "END-OF-RECORD" character and ~ is the wildcard character, the command NL "~]","~]" creates a blank line after each line in the current text.

### Entering the "END-OF-RECORD" Character From the Keyboard

While the INSERT command is being used to create new text, the character specified in the ]= command does not have the special meaning "END-OF-RECORD." For example, if ] is the "END-OF-RECORD" character and the INSERT command is being used to enter new lines, typing ] inserts the symbol ] into the text.

However, if the line buffer currently contains a line recalled to the line buffer by the SEARCH and Edit Line command or by the RECALL NEXT LINE/RECALL LINE or STEP PROGRAM keys, entering the "END-OF-RECORD" character from the keyboard enters an end-of-record character into the text. The "END-OF-RECORD" character appears on the display when it is typed from the keyboard, but is converted to an end-of-record character when the line is placed back in the text buffer. Subsequent listings show the beginning of a new line instead of the "END-OF-RECORD" symbol.

Thus you may split up a recalled line by entering an "END-OF-RECORD" character from the keyboard while the cursor is in the appropriate position.

### Deleting "END-OF-RECORD" Characters From the Keyboard

When certain searching operations locate and list a line containing an end-of-record character, the end-of-record character is represented on the display by the "END-OF-RECORD" character.

For example, if the "END-OF-RECORD" character assigned by the ]= command appears in the target string of a SEARCH and Edit Line command, listings of recalled lines show the "END-OF-RECORD" character in the appropriate position. The end-of-record character may be deleted from the recalled line by typing over the "END-OF-RECORD" symbol with a new character, or by pressing the RUBOUT key while the cursor is positioned over the "END-OF-RECORD" symbol. This allows you to concatenate lines by deleting end-of-record characters from the keyboard.

For example, if # is the "any digit" character and / is the "END-OF-RECORD" character, the command S "#/R", tells the EDITOR to locate lines that end with a digit and are immediately followed by a line beginning with R . When such a pair of lines is found, both the line ending with a digit and the line beginning with R are recalled to the line buffer, and listed on the display as one line separated by an end-of-record character. The end-of-record character is represented on the display by the symbol / . For example:

```
S "#/R",
      :1324/RETURN
      :1650/RETURN
```

The end-of-record character may now be deleted by typing over the symbol / with a new character or by pressing the RUBOUT key while the cursor is positioned over the / . This deletes the end-of-record character, and concatenates the two lines.

## Default Values

Calling the EDITOR assigns the character ] to be the "END-OF-RECORD" character by default. Until a ]= command is executed that specifies a different ASCII character, the EDITOR understands ] to mean "END-OF-RECORD" when used in a target or replacement string or when entered from the keyboard after a line is recalled to the line buffer.

When the parameter of the ]= command is omitted, no "END-OF-RECORD" character is assigned. That is, entering ]= and pressing RETURN cancels the previously chosen "END-OF-RECORD" character, and makes no new assignment. You should execute the command ]= if you want to make sure that no ASCII character means "END-OF-RECORD" when used in a target or replacement string or when inserted into a recalled line. At any time after the command ]= is executed, you are free to choose an "END-OF-RECORD" character again by executing a command such as ]=] or ]=/ .



]=

### When to Execute the ]= Command

Because calling the EDITOR assigns the character ] as the "END-OF-RECORD" character, you must execute a ]= command that specifies a character other than ] if you intend to search for occurrences of the symbol ] in the text, or if you want to use the symbol ] in a replacement string without the special meaning "END-OF-RECORD."

You may also want to assign a different "END-OF-RECORD" character before recalling a line to the text buffer that contains the symbol ]. Otherwise, returning the line to the text buffer would delete the symbol ] and add an end-of-record character in its place.

### An Editing Example

The following examples illustrate the use of the ]= command.

#### Example 1

```
LIST
      :THIS
      :IS
      :A
      :TEST

NL "]" ,"*"

LIST
      :*THIS*IS*A*TEST

NL "]" ,"↓"

LIST
      :↓*THIS*IS*A*TEST
```

#### Example 2

```
LIST
1:500 DO 200 I=1,500
2:    B(N)=A(N)
3:    C=B(N)
4:
5:    DO 250 J=2,N
6:250 B(N-J+1)=A(N-J+1)+X*B(N-J+2)
7:    K1=N-1
8:
9:    DO 300 J=2,K1
10:300 C=B(K1-J+2)+X*C
11:
12:    X0=X-B(1)/C
13:    X0=X
```

NL "]" "\*"

(continued on next page)

```

LIST
1:500 DO 200 I=1,500
2:    B(N)=A(N)
3:    C=B(N)
5:    DO 250 J=2,N
6:250 B(N-J+1)=A(N-J+1)+X*B(N-J+2)
7:    K1=N-1
9:    DO 300 J=2,K1
10:300 C=B(K1-J+2)+X*C
12:    X0=X-B(1)/C
13:    X0=X

```

Example 3

```

LIST
:TAPE VERIFICATION [ SECTION 5 ]
:
:Description
:
:Hardware Requirements
:
:Program Limitations
:
:Operating Instructions
:
:Variables

```

NL "]]", "]] " "

```

LIST
:TAPE VERIFICATION [ SECTION 5 ]
:
:Description
:
:Hardware Requirements
:
:Program Limitations
:
:Operating Instructions
:
:Variables

```

]=/

```

S "]"
:TAPE VERIFICATION [ SECTION 5 ]

```

]=

## Example 4

```

LIST
:TAPE VERIFICATION [ SECTION 5 ]
:
: Description
: Hardware Requirements
: Program Limitations
: Operating Instructions- Methodology

]=/

S "-", "/"
: Methodology

S "Description/", "Description-"
: Description- Hardware Requirements

LIST
:TAPE VERIFICATION [ SECTION 5 ]
:
: Description- Hardware Requirements
: Program Limitations
: Operating Instructions
: Methodology

```

Example 1 shows the location of end-of-record characters in a short sample text. At the time of the initial listing, the character ] is the "END-OF-RECORD" character: no ]= commands have been previously executed.

Because the character ] means "END-OF-RECORD," the command NL "]" "\*" tells the EDITOR to replace all end-of-record characters with the symbol \* . A subsequent listing shows the text concatenated into one line, with the symbol \* in positions previously occupied by end-of-record characters. The \*'s indicate that lines of text are considered to begin with an end-of-record character. This is because the end-of-record character is actually a flag stored in the two bytes that precede the line in the text buffer.<sup>5</sup>

Example 2 shows the "END-OF-RECORD" character being used to delete blank lines. The initial listing shows that the text consists of thirteen lines of FORTRAN code, including two blank lines (lines 4 and 11). At the time of the listing, the character ] is the "END-OF-RECORD" character: no ]= commands have been previously executed.

<sup>5</sup> Although the \* at the beginning of the line in the new listing shows that the initial end-of-record character has been replaced by the character \*, the EDITOR has automatically reinserted an end-of-record character at the beginning of the line. Thus text always begins with an end-of-record character (and never ends with one).

Then the command `NL "]"*` is executed. Because `]` is the "END-OF-RECORD" character, the command tells the EDITOR to locate and delete blank lines (lines consisting of two successive end-of-record characters). After the command is executed, a new listing shows that the two blank lines (lines 4 and 11) have been deleted from the text buffer.

Example 3 shows the "END-OF-RECORD" character being used to indent certain lines of text. At the time of the initial listing, `]` is the "END-OF-RECORD" character. The command `NL "]"","]]` tells the EDITOR to locate two successive end-of-record characters, and overwrite them with two end-of-record characters followed by five blank spaces. This is the same as adding five spaces to the beginning of every line that immediately follows a blank line. After the command is executed, a new listing shows the indented lines are those that immediately follow a blank line in the text.

Next the command `]=/` is executed to assign `/` as the "END-OF-RECORD" character. Since `/` is now the "END-OF-RECORD" character, the character `]` no longer has any special meaning, and may be used to locate the symbol `]` in the text. This is illustrated by the fact that the command `S "]"` locates and lists a line that contains the symbol `]`.

Example 4 shows the "END-OF-RECORD" character being used to insert and delete end-of-record characters. After an initial listing, the command `]=/` is executed to assign `/` as the "END-OF-RECORD" character. The commands `S "-","/"` and `S "Description/","Description-"` are then executed.

The command `S "-","/"` tells the EDITOR to overwrite occurrences of the symbol `-` with an end-of-record character, and to list changed lines on the display. The text contains an occurrence of the symbol `-` in the last line of the text. The EDITOR replaces the symbol `-` with an end-of-record character, splitting the last line into two lines consisting of `Operating Instructions` and `Methodology`. A listing of the new line `Methodology` appears on the display.

The command `S "Description/","Description-"` tells the EDITOR to overwrite occurrences of `Description/` with `Description-`. The third line together with the end-of-record character that precedes the fourth line, matches the target string and is overwritten by the replacement string. This has the effect of concatenating the third and fourth lines and inserting a hyphen (-) between them. The new line is listed on the display, and a complete listing is made to show how the text has been changed.

## NOTES

## Section 6

# INPUT/OUTPUT COMMANDS

<b>CONTENTS</b>	<b>PAGE</b>
Introduction . . . . .	6-2
I/O Address . . . . .	6-2
Default Values . . . . .	6-2
The APPEND Command . . . . .	6-5
The FIND Command . . . . .	6-11
The INPUT Command . . . . .	6-17
The MARK Command . . . . .	6-21
The OLD Command . . . . .	6-25
The PRINT Command . . . . .	6-29
Special PRINT Commands . . . . .	6-33
The SAVE Command . . . . .	6-39
The SKIP Command . . . . .	6-43
The SWN Command . . . . .	6-49
The WRITE Command . . . . .	6-55

### INTRODUCTION

Section 6 describes ten EDITOR commands used for transmitting text to and from storage devices, for positioning the magnetic tape head, and for marking new files. The commands are APPEND, FIND, MARK, OLD, PRINT, SAVE, SKIP, SWN (Save With Number), and WRITE. Many of these commands have the same keyword as a BASIC Input/Output command, but differ slightly from BASIC. Important differences between EDITOR Input/Output commands and their BASIC counterparts are included in the explanation of the commands.

### I/O ADDRESSES

As in BASIC, an I/O address consists of the symbol @ or % , followed by a primary address, a comma, a secondary address, and a colon (:). The symbol @ or % specifies the Input/Output format to be used (refer to "Specifying an Alternate Input/Output Format" in the explanation of the PRINT command).

The primary address is a peripheral device number between 1 and 255. As in BASIC, when an Input/Output command is executed, the peripheral device is converted to a primary talk or listen address, whichever is appropriate for the keyword. The primary address tells the peripheral device whether it has been selected to send or receive information from the text buffer.

Preassigned peripheral device numbers are the same as in BASIC. For example, device 32 is the system display, device 33 is the internal magnetic tape, and device 37 addresses microprocessor status parameters. Device numbers 1 through 30 are used for external devices on the General Purpose Interface Bus.

A secondary address is specified as a number between 0 and 31. Each number has a predefined meaning that tells the EDITOR what type of operation is being performed. For example, secondary address 27 means that the EDITOR is executing a FIND command; secondary address 4 means that the EDITOR is executing an APPEND or OLD command.

The colon (:) following the secondary address marks the end of the I/O address.

### DEFAULT VALUES

I/O addresses in EDITOR commands are optional. If an I/O address is not specified, the EDITOR automatically inserts a default I/O address appropriate for the keyword. If only a primary address is specified, the EDITOR automatically issues a default secondary address.

The following table lists the default primary and secondary addresses for EDITOR Input/Output commands.

TABLE OF DEFAULT PRIMARY AND SECONDARY ADDRESSES  
FOR EDITOR INPUT/OUTPUT COMMANDS

I/O Command	Default I/O Address
APPEND	@33,4:
FIND	@33,27:
INPUT	@33,13:
LIST	@32,19:
OLD	@33,4:
PRINT	@32,12:
SAVE	@33,12:
SEARCH	@32,19:
SKIP	@33,13:
SWN (Save with Number)	@33,12:
WRITE	@33,12:

32 = GS display  
33 = internal magnetic tape

## ENTERING EDITOR INPUT/OUTPUT COMMANDS FROM THE KEYBOARD

When entering an Input/Output command, do not enter any blank spaces within the I/O address or immediately before or after the colon (:) that ends the I/O address. Extra blank spaces can cause a syntax error, semantic error, or INVALID I/O OPERATION error. Some examples are shown below:

```
SAV@29: 100,500
EDITOR ERROR
Syntax - error number 138
SAV@33: 100,500
```

```
SAV@29 :100,500
EDITOR ERROR
Semantic - error number 139
SAV@33 :100,500
```

```
SAV@ 29:100,500
EDITOR ERROR
```

```
INVALID I/O OPERATION IN IMMEDIATE LINE - MESSAGE NUMBER 67
```



## I/O Commands

The correct way to enter the command is as follows:

```
SAU029:100,500
```

Just as for all EDITOR commands, spaces may be entered immediately after the keyword. That is, the following command is also valid:

```
SAU 029:100,500
```

## COMMAND SEMANTICS

The ending line number specified in an Input/Output command should be as large as the starting line number. If the ending line number is smaller than the starting line number, the command either has no effect, or causes a semantic error as in the following example:

```
SAU 7,3  
EDITOR ERROR  
Semantic - error number 139  
SAU 7,3
```

The correct way to enter the command is as follows:

```
SAU 3,7
```

## MAGNETIC TAPE MOVEMENT

EDITOR Input/Output commands that specify a tape device in the I/O address cause the magnetic tape to move (advance or rewind). After tape movement stops, the magnetic tape head points to a particular logical record on the file. For example, the command `FIN6` advances or rewinds the internal magnetic tape until the tape head points to the first logical record on file 6.

For the sake of simplicity in these explanations, the phrase "positioning the tape to a particular logical record" is used instead of "positioning the tape so that the tape head points to a particular logical record." References to the tape head are omitted, except when needed to emphasize the exact position of the magnetic tape.

## The APPEND Command

### Syntax Form:

A [ I/O address ] [ edit line number ]

### Descriptive Form:

APPEND [ I/O address ] [ destination for appended text ]

## PURPOSE

The APPEND command brings logical records into the text buffer from the file currently open on a peripheral device. If no peripheral device is specified, the APPEND command inputs logical records (stored lines) from a file on the internal magnetic tape. Incoming lines are added to the current text immediately before the line number specified in the command.

## EXAMPLES

A

A 50

A@29:

A@29:0

## EXPLANATION

The APPEND command allows an I/O address and a destination line number to be specified. When the APPEND command is executed, the EDITOR inputs logical records from the specified peripheral device, and adds them to the text buffer immediately before the destination line in the current text. For example, the command A@29:0 listed above inputs logical records from the file currently open on device 29, and places the lines before the first line of text (line 0).

Before the APPEND command is executed, a file on the chosen peripheral device must be "opened" by executing a FIND command. The FIND command positions the READ/WRITE (magnetic tape) head to the beginning of the file, and opens the file for access by Input/Output commands. After executing a FIND command, you may position the tape to a particular logical

## I/O Commands

### APPEND

record within the file by executing a SKIP command. Using the SKIP command to position the tape to a particular logical record, allows you to append part of a magnetic tape file instead of the entire file. (Refer to the FIND and SKIP commands for more detailed explanations.)

Whether the tape is positioned to the beginning of the file using the FIND command, or to a particular logical record using FIND and then SKIP, the APPEND command inputs all logical records from the current position of the tape head to the end of the file. After the APPEND command is executed, the file is closed and no longer available for access by Input/Output commands.

### Default Values

Both the I/O address and the destination line number are optional. When the I/O address is omitted in an APPEND command, the peripheral device is the internal magnetic tape by default. When the destination line is omitted, appended lines are added after the last line of the current text.

For instance, the command `A` listed above brings logical records into the text buffer from the file currently open on the internal magnetic tape. Appended lines are inserted at the end of the text. The command `A 50` also inputs logical records from the internal magnetic tape, but places incoming lines immediately before line 50 in the text buffer. The command `A@29:` inputs logical records from the open file on peripheral device 29, and sends incoming lines to the end of the current text.

### Notes on the Command Syntax

When entering an APPEND command from the keyboard, do not enter any blank spaces between the I/O address and the destination line number. For example, a command such as `A@29: 0` causes a syntax error.

### Differences between BASIC APPEND and EDITOR APPEND

The differences between the BASIC command APPEND and the EDITOR command APPEND are as follows:

- When an APPEND command is executed under EDITOR control, incoming lines do not overwrite a "dummy" line as they do in BASIC. Instead, appended lines are placed before the destination line, without destroying the line.
- Partial files may be appended under EDITOR control, by using SKIP or INPUT commands to position the magnetic tape to a particular logical record within a file. In BASIC this can only be done using INPUT commands.

- BASIC's APPEND command assigns new program line numbers to BASIC program statements brought into memory. However, the EDITOR's APPEND command does not alter BASIC program line numbers, nor does it assign edit line numbers to newly appended lines of text.
- BASIC's APPEND command cannot input lines that are more than 72 characters long. Attempting to append a line that has more than 72 characters causes a NO PROGRAM FOUND or STATEMENT TOO LONG error message to appear on the display. However, the EDITOR's APPEND command can input lines of any length (up to the current size of the text buffer).

## Error Messages

If no file is open on the specified device, attempting to execute an APPEND command causes a MT File error message to appear on the display.

Attempting to execute an APPEND command causes a Device Access error if the file is open to Output operations only — that is, if a LIST, PRINT, SEARCH, or WRITE command has been executed since the last time the file was opened.

Attempting to execute an APPEND command while the tape is positioned to the beginning of a file marked NEW, LAST, or SECRET causes a MT File error message to appear on the display.

If the APPEND command attempts to input a line that exceeds the current length of the text buffer, execution is terminated and a Text Buf. Overflow error message is printed on the display. After the message appears, the text buffer contains inputted lines up to, but not including, the line that overflowed the text buffer.

## An Editing Example

The following example shows how the APPEND command adds previously stored lines to the current text.

**I/O Commands**  
**APPEND**

**Example 1**

```
LIST
  50:FOR K=1 TO N-1
  60:FOR I=K+1 TO N
  70:M=A(I,K)/A(K,K)
  80:A(I,K)=M
  90:FOR J=K+1 TO N+1
 100:A(I,J)=A(I,J)-M*A(K,J)
 110:NEXT J
 120:NEXT I
 130:NEXT K

FIN1

A 50

LIST
:PRINT "ENTER SIZE OF MATRIX- ROWS, COLUMNS:";
:INPUT N,K
:DELETE A
:DIM A(N,K)
:FOR I=1 TO N
:FOR J=1 TO K
:PRINT "A(";I;",";J;")=";
:INPUT A(I,J)
:NEXT J
:NEXT I
 50:FOR K=1 TO N-1
 60:FOR I=K+1 TO N
 70:M=A(I,K)/A(K,K)
 80:A(I,K)=M
 90:FOR J=K+1 TO N+1
100:A(I,J)=A(I,J)-M*A(K,J)
110:NEXT J
120:NEXT I
130:NEXT K
```

The initial listing in Example 1 shows that the text buffer contains nine lines of text. The lines are BASIC program statements entered without program line numbers.

The command `FIN1` is executed to open file 1 on the internal magnetic tape and position the tape head to the beginning of the file. Then the command `A 50` is executed.

When the `APPEND` command is executed, the contents of file 1 are brought into the text buffer and placed immediately before line 50 of the current text. A new listing shows that ten lines have been added to the text buffer. The appended lines are the unnumbered lines in the new listing.

None of the BASIC statements in Example 1 have BASIC program line numbers. The program line numbers were omitted from the statements in the text buffer and on file 1 to avoid the problem of keeping line numbers in sequence (the EDITOR command APPEND does not resequence BASIC program line numbers). After the APPEND command is executed and all editing is complete, you may create BASIC program line numbers by executing an appropriate RENUMBER command, then the SWN (Save With Number) command. For more information on how to create BASIC program line numbers in this way, refer to the explanation of the SWN command.

## NOTES

## The FIND Command

### Syntax Form:

F [ I/O address ] [ numeric constant ]

### Descriptive Form:

FIND [ I/O address ] [ file number ]

## PURPOSE

The FIND command positions the magnetic tape to the beginning of a specified file on a peripheral device, and opens the file for access by Input/Output operations.

## EXAMPLES

FIN

FIN5

FIN@29:5

FIN@29,27:5

## EXPLANATION

The FIND command allows an I/O address and a file number to be specified.<sup>1</sup> The FIND command positions the tape so that the READ/WRITE (magnetic tape) head is at the beginning of the desired file on the peripheral device, and "opens" the file for access by Input/Output operations. For example, the command FIN@29:5 listed above positions the tape to the beginning of the storage area on file 5 of device 29, and opens the file for access by Input/Output operations.

<sup>1</sup>A file name cannot be specified. The EDITOR is not intended for use with "file management" devices, that is, devices that require file names and passwords.



## I/O Commands

### FIND

#### Default Values

If the I/O address is omitted in a FIND command, the internal magnetic tape is selected as the peripheral device by default. If the file number is omitted, the EDITOR supplies the value 0 by default, and performs a REWIND function on the specified device.

For example, the command `FIN5` positions the internal magnetic tape to the beginning of file 5 and opens the file for access by Input/Output operations. The command `FIN@29:` rewinds the magnetic tape on device 29, and the command `FIN` rewinds the internal magnetic tape.

#### Notes on the Command Syntax

When entering a FIND command from the keyboard, do not enter any blank spaces between the I/O address and the file number. For example, the command `FIN@29: 5` causes a syntax error.

#### Error Messages

If a non-existent internal magnetic tape file is specified in a FIND command, the EDITOR returns a `MAG TAPE FILE NOT FOUND` error message.

#### Finding the LAST File

As in BASIC, the magnetic tape head may be positioned to the LAST (dummy) file by specifying the appropriate file number in a FIND command. The tape is automatically positioned to the beginning of the file header in preparation for creating new files with the MARK command. When a MARK command is executed after finding the LAST file, the dummy file is overwritten with new files as they are created, and a new dummy file is created on tape as the LAST file.

#### Accessing the Tape File Header

As in BASIC, if a magnetic tape status parameter is changed by a special PRINT command, the FIND command positions the tape to the beginning of the specified file header instead of to the beginning of the storage area. This allows direct access to the file header. Information in the header can be changed or deleted, or new information can be added.<sup>2</sup> For example, when the following EDITOR commands are executed, all of file 1 beginning with the file header is loaded into the text buffer and printed on the display:

```
PRI@33,0:0,0,1
```

```
FIN1
```

```
OLD
```

```
LIS
```

<sup>2</sup>A tape file header can be changed as long as the header meets the minimum format requirements described in the Graphic System Reference Manual under the topic "Changing a Tape File Header" in the explanation of the FIND command.

The first command shown above changes the internal magnetic status parameter to "no header" format. This causes the EDITOR to position the tape to the beginning of the header on file number 1 when the command `FIN1` is executed. (The EDITOR assumes that the file header is the first logical record in the storage area because a "no header" format has been specified.)

Executing the command `OLD` brings the file header into the text buffer along with the contents of file 1. The file header is now available for editing using the keyboard and LINE EDITOR keys.

After a file header is changed, the new information can be stored on the tape file using an EDITOR `SAVE`, `SWN`, or `WRITE` command. Executing the EDITOR command `PRI@33,0:0,0,0` returns the internal magnetic tape status parameter to its normal value.

### Opening a Magnetic Tape File

Opening a tape file by specifying the file number in a FIND command makes the contents of the file available to EDITOR Input/Output commands `APPEND`, `OLD`, `INPUT`, `MARK`, `PRINT`, `SAVE`, `SKIP`, `SWN`, or `WRITE`. For example, after file 5 on device 29 is opened by executing the command `FIN@29:5`, file 5 is prepared for access by Input/Output commands that specify peripheral device 29.

Once a file is open, it remains open until a subsequent EDITOR command closes the file.

### Closing a Magnetic Tape File

Certain EDITOR Input/Output commands automatically "close" the file that is currently open on the specified peripheral device. After a file is closed, it is unavailable to Input/Output operations until it is reopened by a FIND command.

Closing a file places an end-of-file mark after the last record in the file. If the command that closes the file is an Output command, closing the file also forces any information remaining in the magnetic tape buffer onto the tape file.<sup>3</sup>

The following EDITOR commands close the current file after the Input or Output operation is completed:

`APPEND`  
`FIND4`  
`OLD`  
`MARK`  
`SAVE`  
`SWN`

<sup>3</sup>The magnetic tape buffer is a portion of memory used for intermediary storage of text during Input/Output operations.

<sup>4</sup>When the FIND command is executed, the EDITOR closes the file that is currently open on the device, and opens the file specified by the command.

## **I/O Commands**

### **FIND**

After any of the above commands are executed, the open file on the specified device is closed and unavailable for access. Before another I/O operation is performed on the file, the file must be reopened by executing the FIND command again.

If one of these commands specifies a device other than the internal magnetic tape, the command closes any open file on the internal magnetic tape as well as on the specified external device. Pressing user-definable key 5 (the RETURN TO BASIC overlay key) closes all open files in the system.

The INPUT and SKIP commands also close the current file if an attempt is made to read beyond the physical or logical end of the file (refer to the explanations of the INPUT and SKIP commands).

### **EDITOR Input/Output Commands that Do Not Close the Current File**

The following commands do not close the open file on the specified peripheral device:

INPUT  
LIST  
PRINT  
SEARCH  
SKIP  
WRITE

After any of these commands are executed, the file involved in the I/O operation is still open and available for another Input or Output operation. For example, after the WRITE command outputs text to a tape file, the file remains open for Output operations. Thus you may execute WRITE commands repeatedly without executing the FIND command again, or execute a SAVE after a WRITE command without having to reopen the file.

The commands LIST, PRINT, and WRITE place a logical end-of-file mark after the last record stored on the tape, but do not close the file. If another Output command is executed, lines sent to the file overwrite that end-of-file mark, and a new end-of-file mark is placed at the end of the newly stored lines.

### **Special Uses for the FIND Command**

To close an open file on a particular device, you need only execute a FIND command that specifies a different file on the device. The reason for this is that the FIND command closes any open file on the device before opening the file specified in the command.

For instance, after the WRITE command is used to output text to file 5 on device 29, you may close the file by executing a command such as `FIN@29:1`. When the command `FIN@29:1` is executed, file 5 is closed and file 1 is opened.

It can be important to close a file in this manner, since closing a file forces the last information in the magnetic tape buffer onto the tape file. If the system power is turned off or the internal tape unit removed while a file is still open, text remaining in the magnetic tape buffer does not reach the file and is lost. Closing the file ensures that all transmitted text reaches the tape.

## NOTES

## The INPUT Command

**Syntax Form:**

INP [ I/O address ]

**Descriptive Form:**

INPUT [ I/O address ]

### PURPOSE

The INPUT command displays one logical record from the file currently open on a specified peripheral device, and advances the magnetic tape to the next record on the file.

### EXAMPLES

INP

INP@29:

INP@29,13:

INP@29,30:

### EXPLANATION

When the INPUT command is executed, the EDITOR brings one logical record (stored line) from a magnetic tape file into a temporary buffer, then displays the line on the screen. At the same time, the magnetic tape is positioned to the next record on the file.

The current text is not affected by the INPUT command, since the displayed line is held in a temporary buffer and is not sent to the text buffer.

Before the INPUT command is executed, a file must be opened on the desired peripheral device by executing a FIND command. When the INPUT command is executed immediately after the FIND command, the first logical record of the open file appears on the display, and the tape advances one logical record on the file.

## **I/O Commands**

### **INPUT**

After the INPUT command is executed, the file remains open for access by EDITOR Input/Output commands. For example, another INPUT command can be executed without executing the FIND command again. This time, the second logical record of the open file appears on the display, and the tape moves forward one more record.

You may execute the INPUT command repeatedly, to display successive logical records and advance the tape one logical record at a time. Or, you may use the FIND and SKIP commands to position the tape to a particular logical record, then execute an INPUT command to display the record on the screen.

Whether the tape is positioned to the beginning of a file using the FIND command, or to a particular logical record using FIND and then INPUT or SKIP, the INPUT command displays the logical record that is currently positioned at the tape head.

### **Default Values**

When no I/O address is specified in the INPUT command, the peripheral device is the internal magnetic tape by default. That is, the command INP displays one logical record from the file currently open on the internal magnetic tape, and positions the tape to the beginning of the next record on the file.

### **The Difference Between SKIP and INPUT**

Both the SKIP and INPUT commands reposition the magnetic tape on a peripheral device. The INPUT command displays one logical record, and moves the tape to the beginning of the next record on the file. However, the SKIP command is used to move the tape forward a specified number of logical records, and does not display records on the screen. (Refer to the explanation of the SKIP command.)

### **The Difference Between BASIC INPUT and EDITOR INPUT**

Like the EDITOR INPUT command, the BASIC INPUT command inputs one logical record from a specified device, and positions the tape head to the beginning of the next record. However, instead of being held in a temporary buffer, the data is brought into memory and assigned to a variable specified in the INPUT command.

Executing the command INP@29:A\$ and then PRI A\$ in BASIC is analagous to executing the command INP@29: under EDITOR control. The only difference is that the EDITOR command INP@29: displays a line on the screen, but does not store it for later use.

## Error Messages

If the INPUT command attempts to input a logical record from an empty internal magnetic tape file or an internal magnetic tape file that has a header marked NEW, SECRET, or LAST, the EDITOR returns a MT File error message.

If no file is open on the peripheral device, executing an INPUT command causes a MT File error.

Attempting to execute an INPUT command causes a Buffer Access error if the file is open to Output operations only—that is, if a LIST, PRINT, SEARCH, or WRITE command has been executed since the last time the file was opened.

If the INPUT command attempts to move the tape beyond the physical or logical end of the file, a Device at EOF error message appears on the display, and the file is closed to Input/Output operations.

## Special Uses for the INPUT Command

The INPUT command may be used to check the position of the tape after SKIP or INPUT commands have repositioned the tape.

The INPUT command allows you to view the first line of a file without bringing the file into the text buffer.

The INPUT command may be used to position the magnetic tape to a particular logical record within a file before performing an Input/Output operation. The INPUT command may be used before any of the following EDITOR Input/Output commands: APPEND, INPUT, LIST, OLD, PRINT, SEARCH, SKIP, SAVE, SWN, and WRITE.

### NOTE

*INPUT is not intended for use before the MARK command. Attempting to execute a MARK command can destroy the contents of the magnetic tape if the tape is not positioned to the beginning of a file.*

The INPUT command can be used to read the status of an external peripheral device or to clear an error condition that has occurred on the device. For example, if device 29 on the GPIB is the TEKTRONIX 4924 Digital Cartridge Tape Drive, you may execute the command `INP@29,30:` to obtain an error message number and clear the error.



## I/O Commands

### INPUT

#### An Editing Example

The following example shows the INPUT command being used to move the tape and display lines on the screen.

#### Example 1

```
FIN2
OLD
LIST
      :100 REM ** SUBROUTINE
      :110 PRINT "ENTER CONSTANTS:"
      :120 FOR I=1 TO N
      :130 PRINT "B(";I;")=";
      :140 INPUT B(I)
      :150 NEXT I
      :160 PRINT "END OF INPUT"
      :170 RETURN
```

```
FIN2

INP
100 REM ** SUBROUTINE

INP
110 PRINT "ENTER CONSTANTS:"

INP
120 FOR I = 1 TO N

INP
130 PRINT "B(";I;")=";

INP
140 INPUT B(I)

INP
150 NEXT I

INP
160 PRINT "END OF INPUT"

INP
170 RETURN
```

In Example 1 the commands FIN2 , OLD , and LIST are executed to open file 2 on the magnetic tape, bring the file into the text buffer, and list the lines on the display. Then the command FIN2 is executed to reopen file 2, and the INPUT command is executed eight times. Each time the INPUT command is executed, the EDITOR displays a logical record from file 2, and positions the tape head to the beginning of the next record. Lines displayed on the screen by the INPUT command are not preceded by an EDITOR colon (:)

## The MARK Command

### Syntax Form:

MA [ numeric constant ] [ , [ numeric constant ] ]

### Descriptive Form:

MARK [ number of files ] [ , [ number of bytes per file ] ]

## PURPOSE

The MARK command reserves space on the internal magnetic tape for a specified number of files.

## EXAMPLES

```
MA
MA 10,
MA ,5120
MA 2,5120
```

## EXPLANATION

The MARK command is almost identical to its BASIC counterpart. Just as in BASIC, files are formatted in 256-byte physical records unless a special PRINT command instructs the EDITOR to mark files in 128-byte physical records. The first physical record of each file serves as the file header, and the file storage area begins with the second physical record.

When the MARK command is executed, the specified number of files are created on the internal magnetic tape, starting at the current position of the magnetic tape head. The file header for each newly created file is marked NEW, and the final "dummy" file has a file header marked "LAST."

Files are marked to have a minimum of 768 bytes. The exact length of the new files is determined by the number of bytes specified as the second parameter in the MARK command. If the specified number of bytes is not a multiple of 256, the EDITOR takes the new file size to be

## **I/O Commands**

### **MARK**

the next highest multiple of 256. for example, when the command `MA 2,2000` is executed, the EDITOR marks two files having 2048 bytes each, because 2048 is the next highest multiple of 256.

After the MARK command is executed, the tape is positioned to the beginning of the LAST file just created. If another MARK command is immediately executed, additional files are marked on the tape.

You may execute a MARK command while the tape is positioned to the beginning of any file on the internal magnetic tape. As in BASIC, however, any old information stored underneath and beyond the newly marked files, is lost.

#### **NOTE**

*Execute the MARK command only when the tape is positioned to the beginning of a file. Executing a MARK command after a SKIP or INPUT command can destroy the contents of the tape. If this happens, the tape must be re-marked.*

### **Changing the Tape Format**

Internal status parameters can be changed to give a different file marking format. If the special PRINT command `PRI@33,0:1,1,1` is executed, files are created without a file header; are marked in 128-byte physical records; and are marked without using the "Checksum" error checking technique. This allows the EDITOR to make digital tape recordings in a format compatible with other recording devices—a TEKTRONIX 4923 Digital Tape Recorder, for example. (Refer to the PRINT command for more information about changing the tape format.)

### **Default Values**

When the number of bytes is not specified in a MARK command, the EDITOR marks files to be 768 bytes long. If the number of files is not specified, only a LAST file is marked, and no new files are created.

For instance, the command `MA` creates a LAST file that is 768 bytes long. The command `MA 10,` marks ten files of 768 bytes each, and the command `MA ,5120` marks one file of 5120 bytes.

### **Notes on the Command Syntax**

When only the first parameter is entered, the delimiter may be omitted without altering the meaning of the command. For example, the command `MA 5,` is the same as `MA 5 .` Both commands tell the EDITOR to mark five new files of 768 bytes each.

### **Differences Between BASIC MARK and EDITOR MARK**

- BASIC requires both the number of files and the number of bytes per file to be specified in a MARK command. The EDITOR allows these parameters to be omitted.
- The EDITOR command MARK does not allow an I/O address to be specified. Files are marked on an external device by executing a PRINT command (see "Marking Files on an External Device").

### **Marking Files on an External Device**

To mark new files on a peripheral device other than the internal magnetic tape, execute a PRINT command that specifies 28 as the secondary address. For instance, the command `PRI@3,28:5,2048` creates five new files of 2048 bytes each on device 3 on the GPIB (General Purpose Interface Bus). This enables you to mark files on any device (a 4924 Digital Cartridge Tape Drive, for example) while the system is under EDITOR control.

## NOTES

## The OLD Command

**Syntax Form:**

O [ I/O address ]

**Descriptive Form:**

OLD [ I/O address ]

### PURPOSE

The OLD command clears the text buffer, then brings logical records into the text buffer from the file currently open on a peripheral device. If no peripheral device is specified, the OLD command inputs logical records (stored lines) from a file on the internal magnetic tape.

### EXAMPLES

OLD

OLD@29:

OLD@29,4:

### EXPLANATION

The OLD command allows an I/O address to be specified. When the OLD command is executed the EDITOR clears (erases) the text buffer, then loads logical records into the text buffer from the specified peripheral device. For example, the command OLD@29: listed above clears the text buffer, then inputs stored lines from the file currently open on device 29.

Before the OLD command is executed, a file on the chosen peripheral device must be "opened" by executing a FIND command. The FIND command positions the READ/WRITE (magnetic tape) head to the beginning of the file, and opens the file for access by Input/Output commands. Unless a FIND command is executed to position the tape and open a file, attempting to execute the OLD command erases the text buffer and causes a MT File error message to appear on the display.<sup>5</sup>

<sup>5</sup>For this reason, you must be careful not to execute an OLD command unintentionally by entering O and pressing RETURN. If a FIND command has not been executed, an error message appears and the entire contents of the text buffer are lost.

### OLD

After executing a FIND command, you may position the tape to a particular logical record within the file by executing a SKIP command. Using the SKIP command to position the tape to a particular logical record, allows you to input part of a magnetic tape file instead of the entire file. (Refer to the FIND and SKIP commands for more detailed explanations.)

Whether the tape is positioned to the beginning of the file using the FIND command, or to a particular logical record using FIND and then SKIP, the OLD command inputs all logical records from the current position of the tape head to the end of the file. After the OLD command is executed, the file is closed and no longer available for access by Input/Output commands.

### Default Values

When no I/O address is specified in the OLD command, the peripheral device is the internal magnetic tape by default. That is, the command O clears the text buffer, then brings logical records into the text buffer from the file currently open on the internal magnetic tape.

### The Difference Between OLD and APPEND

The OLD command deletes the current contents of the text buffer before inputting lines, but the APPEND command does not. That is, APPEND is used to add lines to the current text, but OLD is used to replace the current text with previously stored lines.

### The Differences Between BASIC OLD and EDITOR OLD

The differences between the BASIC command OLD and the EDITOR command OLD are as follows:

- Partial files may be brought in under EDITOR control, by using SKIP or INPUT commands to position the tape to a particular logical record within a file. In BASIC this can only be done using INPUT commands.
- BASIC's OLD command examines program statements for syntax errors, and rearranges statements according to their BASIC program line numbers. Also, if two statements have the same program line number, BASIC's OLD command inputs only the second of the statements.

The EDITOR's OLD command does not perform these functions. The EDITOR sees BASIC program line numbers and statements as free text, and does not recognize BASIC syntax errors.

- Attempting to execute an OLD command when no file is open on the specified device, causes an error both in BASIC and under EDITOR control. In BASIC, the error does not affect the program currently stored in memory, but under EDITOR control, the entire text is deleted.
- BASIC's OLD command cannot input lines that are more than 72 characters long. Attempting to input a line that has more than 72 characters causes a NO PROGRAM FOUND or STATEMENT TOO LONG error message to appear on the display. However, the EDITOR's OLD command can input lines of any length (up to the current size of the text buffer).

### **Error Messages**

If no file is open on the specified device, attempting to execute an OLD command causes a `MT FILE` error message to appear on the display.

Attempting to execute an OLD command causes a `Device Access` error if the file is open to Output operations only—that is, if a `LIST`, `PRINT`, `SEARCH`, or `WRITE` command has been executed since the last time the file was opened.

If an OLD command attempts to input logical records from an internal magnetic tape file marked `NEW`, `SECRET`, or `LAST`, a `MT File` error message appears on the display.

If the OLD command attempts to input a line that exceeds the current length of the text buffer, execution is terminated and a `Text Buf. Overflow` error message is printed on the display. After the message appears, the text buffer contains inputted lines up to, but not including, the line that overflowed the text buffer.

### **An Editing Example**

The following example shows how the OLD command clears the text buffer, then brings previously stored lines into the text buffer.



Example 1

```
LIST
:FOR K=1 TO N-1
:FOR I=K+1 TO N
:M=A(I,K)/A(K,K)
:A(I,K)=M
:FOR J=K+1 TO N+1
:A(I,J)=A(I,J)-M*A(K,J)
:NEXT J
:NEXT I
:NEXT K
```

FIN4

OLD

```
LIST
:
:100  K=0
:      FORMAT (F6.1)
:      READ (5,100) X0
:10   X=X0-F(X0)/DF(X0)
:      IF (ABS(F(X)).LT.(0.5E-3)) GO TO 30
:      K=K+1
:      IF (K.GT.500) GO TO 20
:      X0=X
:      GO TO 10
:20   WRITE (6,200)
:200  FORMAT (1X,4HPRG,4HRA,4HAB,4HTE.)
:      STOP
:30   WRITE (6,300) X
:300  FORMAT (1X,4HROOT,4H IS ,F9.4)
:      STOP
:      END
:      FUNCTION F(X)
:      F=X**3-1.473*X**2-5.738*X+6.763
:      RETURN
:      END
:      FUNCTION DF(X)
:      DF = 3*X**2-2.946*X-5.738
:      RETURN
:      END
```

The initial listing in Example 1 shows that the text buffer contains nine lines of text. Then the command FIN4 is executed to open file 4 on the internal magnetic tape and position the tape to the beginning of the file.

Next the command OLD is executed. The text buffer is cleared of its previous contents, and file 4 of the internal magnetic tape is loaded into the text buffer.

A new listing shows that the lines of text that appeared in the first listing have been deleted, and the text now consists of the lines stored in file 4.

## The PRINT Command

**Syntax Form:**

```
P [ I/O address ] [ string ]
```

**Descriptive Form:**

```
PRINT [ I/O address ] [ ASCII character string ]
```

### PURPOSE

The PRINT command outputs an ASCII character string to a specified peripheral device. If no peripheral device is specified, the string is printed on the display by default.

### EXAMPLES

```
PRI CHAPTER 2  
PRI@29:CHAPTER 2  
PRI@3,28:1,5120  
PRI@37,0:10,4,3  
PRI@37,26:1  
PRI@33,0:1,1,1
```

### EXPLANATION

The PRINT command allows an I/O address and an ASCII string to be specified. When the PRINT command is executed, the parameter string is sent to the specified peripheral device. For example, the command `PRI@29:CHAPTER 2` transmits the string `CHAPTER 2` to the open file on device 29.

Executing a PRINT command does not change the current contents of the text buffer. When the specified string is transmitted to a magnetic tape device, the EDITOR records the string on the tape and places an end-of-record character and an end-of-file mark after the string. If the peripheral device is the system display, the string is displayed on the screen, but does not enter the text buffer.

## **I/O Commands**

### **PRINT**

Executing the PRINT command when the tape is positioned to the beginning of the file changes the file header name to ASCII TEXT (if this has not been previously done).

Before executing the PRINT command, you must open a file on the chosen peripheral device by executing a FIND command. The FIND command positions the magnetic tape to the beginning of the file, and opens the file for access. Unless a file is open on the peripheral device, attempting to execute a PRINT statement causes a MT File error message to appear on the display.

Once the file is open, you may position the tape to a particular logical record within the file by executing a SKIP or INPUT command. Using SKIP or INPUT commands to position the tape to a particular logical record allows you to output the ASCII string to a particular location in the file. (Refer to the SKIP and INPUT commands for more detailed explanations.)

Whether the tape is positioned to the beginning of the file or to a particular logical record within the file, the PRINT command stores the specified string on the tape beginning at the current position of the tape head. Any previously recorded information that lies beyond the tape head is lost.

After the PRINT command is executed, the file remains open and available for access to Output operations. Subsequent Output commands overwrite the end-of-file mark left by the PRINT command, and insert a new end-of-file mark when the operation is finished.

For example, a SAVE command can be executed after a PRINT command, without reopening the file. The SAVE command overwrites the previous end-of-file mark, and places a new end-of-file mark at the end of the saved lines.

The PRINT command can be executed repeatedly to output strings to a device. Once a PRINT command has been executed, each subsequent PRINT command overwrites the end-of-file mark left by the previous PRINT command, and begins storing the parameter string at the current position of the tape head.

After executing a PRINT command and before turning the system power off or removing the internal magnetic tape unit, close the file by executing a FIND command or pressing the RETURN TO BASIC overlay key. Closing the file forces any information remaining in the magnetic tape buffer onto the tape, so that no part of the string is lost.

### **Default Values**

When no I/O address is specified in a PRINT command, the peripheral device is selected to be the system display by default. For example, the command PRI@33:CHAPTER 2 records the string CHAPTER 2 on the file currently open on the internal magnetic tape.

When no string is specified in a PRINT command, a blank line is stored on the peripheral device. For example, the command `PRI@33:` saves a blank line on the file currently open on the internal magnetic tape.

### Notes on the Command Syntax

The EDITOR PRINT command does not require the parameter string to be enclosed in quotation marks as the BASIC PRINT command does. That is, the EDITOR command `PRI@33:CHAPTER 2` is equivalent to the BASIC command `PRI@33:"CHAPTER 2"`. Both commands print the string `CHAPTER 2` on the internal magnetic tape.

When entering a PRINT command from the keyboard, do not enter any blank spaces between the I/O address and the parameter string. For example, a command such as `PRI@33: CHAPTER 2` causes a syntax error.

### Error Messages

If no file is open on the internal magnetic tape, attempting to output a string to the tape by executing a PRINT command causes a `MT File` error message to appear on the display.

Executing the PRINT command causes a `Device at EOF` error message to appear if the file is not large enough to hold all of the specified string. When this happens, string characters are written on the tape file until the last byte before the physical end of the file is reached. The EDITOR places a logical end-of-file mark in the last byte of the file, and returns the `Device at EOF` error message. The remainder of the string transmitted by the PRINT command is not stored on the tape.

After a PRINT command is executed, the file remains open to Output operations only. Attempting to execute an APPEND, OLD, INPUT, or SKIP command without reopening the file causes an error. A `Device Access` or `Buffer Access` error message appears on the display. Both messages mean that the specified device is not available for access by Input operations, and that the magnetic tape buffer cannot receive information from the device.

**I/O Commands**  
**PRINT**

**An Editing Example**

The following examples show the PRINT command being used to output a specified string.

**Example 1**

```
PRI SECTION C  
SECTION C
```

**Example 2**

```
FIN15  
PRI@33:SECTION C  
FIN15  
OLD  
LIST  
      :SECTION C
```

In Example 1 the command `PRI SECTION C` tells the EDITOR to display the string `SECTION C` on the screen. In Example 2 the command `PRI@33:SECTION C` tells the EDITOR to write the same string on the internal magnetic tape. First the command `FIN15` opens file 15 on the internal magnetic tape, then the PRINT command is executed. To display the result, the commands `FIN15`, `OLD`, and `LIST` are executed. The new listing shows that file 15 now contains the logical record `SECTION C`.

## SPECIAL USES FOR THE PRINT COMMAND

### Duplicating Other Editor Commands

As in BASIC, the PRINT command can be used to duplicate commands such as FIND, MARK, and LIST by specifying the appropriate secondary address.<sup>6</sup> For example, the EDITOR command `PRI@20,27:5` is equivalent to the command `FIN@20:5` because secondary address 27 in the PRINT command is secondary address for the FIND operation.

### Marking Files on an External Device

The MARK command can also be duplicated by a PRINT command that specifies 28 as the secondary address. For example, the command `PRI@3,28:5,2048` creates five new files on device 3.

Because the EDITOR MARK command does not allow an I/O address to be specified, executing a PRINT command with secondary address 28 is the only way to mark files on an external device.

## SPECIAL PRINT COMMANDS

The EDITOR allows three special PRINT commands to be executed in order to change internal status parameters. The first special PRINT command changes the internal magnetic tape status, and affects how files are marked and read. The other special PRINT commands change the status of the microprocessor, and affect the results of Input/Output operations.

### Internal Magnetic Tape Status

The status byte for the internal magnetic tape unit is changed by a special PRINT command that specifies 33 as the primary address, and 0 as the secondary address. The command sends information to the EDITOR for later use in marking and reading files. Changing the status byte allows the EDITOR to read tapes recorded on external recording equipment. For example, the command `PRI@33,0:1,1,1` allows the EDITOR to read and write in a format compatible with that of the TEKTRONIX 4923 Digital Cartridge Tape Recorder.

As in BASIC, the special PRINT command consists of the keyword PRINT and the I/O address `@33,0:` followed by three parameters. Each parameter must be assigned a value of 0 or 1. The value assigned to each parameter determines how files are marked and read until the system power is turned off or another PRINT command is executed with different values. The first parameter in the PRINT command represents the physical record length; the second parameter specifies whether or not the magnetic tape unit is to use the checksum error checking technique; and the third parameter selects file header format or non-header format.

<sup>6</sup> Depending on the device involved in the operation, other EDITOR Output commands may be duplicated by a PRINT command.

### Physical Record Length

When the MARK command creates files on the internal magnetic tape, the file is divided into physical records. The number of bytes in a physical record is determined by the first parameter in the special PRINT command. If the value assigned to the first parameter is 1, a physical record consists of 256 bytes; if the value assigned is 0, a physical record consists of 128 bytes. If no value is specified, the parameter is set to 0 by default.

If the first parameter is set to 1, the EDITOR can only read magnetic tapes marked in 128-byte physical records. If the parameter is set to 0, the EDITOR can only read tapes marked in 256-byte physical records. If the parameter is not set to the appropriate value, attempting to read a tape causes a `MAG TAPE READ ERROR IN IMMEDIATE LINE` error message to appear on the display.

### Checksum

As in BASIC, "checksum" is a technique used to check for errors when tapes are read or recorded.<sup>7</sup> If the second parameter in the special PRINT command is assigned the value 0, the EDITOR uses the checksum technique during subsequent Input/Output operations. However, if the parameter is assigned the value 1, the EDITOR does not use the checksum technique. If no value is specified, the parameter is set to 0 by default.

Once the second parameter is set to 1, the EDITOR can only read tapes recorded without using the checksum technique; and after the parameter is reset to 0, the EDITOR can only read tapes recorded using the checksum technique. Attempting to read a tape causes a `MAG TAPE READ ERROR` if the parameter is not set to the appropriate value.

### Header Format

If the third parameter of the PRINT command is set to 1, the EDITOR marks files in "no header" format—that is, the first logical record on the file is treated as a file header, and the second logical record on the file is considered to be the beginning of the storage area.

If an ASCII file has been marked in header format, the tape file header may be accessed by setting the third parameter to 1, then executing an OLD command to bring the file into the buffer. The first logical record brought into the text buffer is the file header. (Refer to the FIND command for more information about accessing a file header.)

The value 0 or 1 assigned to each parameter in the PRINT command determines how files are marked and read until the system power is turned off or another PRINT command is executed to assign new values. Turning the system power off resets the parameters to 0, and has the same effect as executing the command `PRI@33,0:0,0,0` or `PRI@33: .` Pressing the RETURN TO BASIC overlay key does not alter the values assigned to the magnetic tape status parameters.

<sup>7</sup>The checksum technique is described in the section on Magnetic Tape Status in the Graphic System Reference Manual.

## Examples

Some examples of the special PRINT command are given below, with a summary of their effect on subsequent magnetic tape operations.

PRI@0,0:1,1,1	— 128-byte physical record — no checksum — non-header format
PRI@33,0:0,0,0	— 256-byte physical record — checksum — header format
PRI@33,0:1,0,1	— 128-byte physical record — checksum — non-header format

## Microprocessor Status Parameters

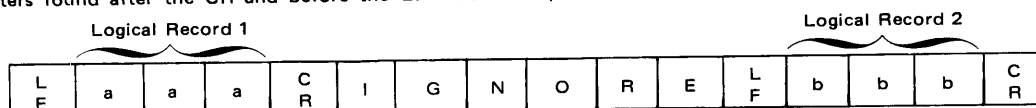
Input and output delimiters are controlled by assigning values to microprocessor status parameters in two special PRINT commands. The first special PRINT command specifies the I/O address @37,26: and controls the input and output delimiters (end-of-record characters) used during APPEND, OLD, LIST, and SEARCH commands. The second special PRINT command specifies the I/O address @37,0: and determines the delimiters that are used when a percent sign (%) appears instead of the symbol @ in subsequent Input/Output commands.

### Changing the ASCII Input/Output Delimiter from CR to CR/LF

The ASCII Input/Output delimiter is controlled by a special PRINT command. As in BASIC, the PRINT command consists of the keyword PRINT and the I/O address @37,26: followed by one parameter. The parameter must be assigned a value of 0 or 1. The value 0 or 1 determines the Input/Output delimiter to be used until the system power is turned off or the value of the parameter is changed by executing another PRINT command.

Assigning the value 1 to the parameter by executing the command PRI@37,26:1 tells the EDITOR to insert a LF (LINE FEED) character after each CR (CARRIAGE RETURN) in the text when subsequent LIST or SEARCH command output text to a device other than the system display. The command also tells the EDITOR to delimit incoming logical records on a LF (LINE FEED) character instead of a CR character during subsequent APPEND or OLD operations that specify a device other than the internal magnetic tape.<sup>8</sup>

<sup>8</sup> Actually each incoming record begins with a LF character and ends with a CR. The EDITOR ignores any characters found after the CR and before the LF. For example:



When an APPEND or OLD command is executed, the EDITOR inputs two logical records (the strings aaa and bbb) but does not input the string IGNORE.



Assigning the value 0 to the parameter by executing the command `PRI@37,26:0` tells the EDITOR to use a CR (CARRIAGE RETURN) character as the end-of-record character during all Input/Output operations.

Tapes recorded while the parameter is set to 0 cannot be read while the parameter is set to 1.

The value 0 or 1 assigned to the parameter in the PRINT command determines the Input/Output delimiters until the system power is turned off or another PRINT command is executed to assign a different value. Turning the system power off resets the parameter to 0, and has the same effect as executing the command `PRI@37,26:0`. Pressing the RETURN TO BASIC overlay key does not alter the value assigned to the status parameter.

### Selecting an Alternate Input/Output Format

When the symbol `@` appears in an Input/Output command, the EDITOR uses a CR (CARRIAGE RETURN) character for the end-of-record character; 255 (hexadecimal FF) for the end-of-file mark; and 255 for a character to be deleted upon input or output (255 does not represent an ASCII character, so no ASCII character is deleted).

When the symbol `%` appears in the I/O address instead of `@`, the EDITOR uses an alternate I/O format that has been established in a special PRINT command. The special PRINT command consists of the I/O address `@37,0:` followed by three parameters.

The parameters represent the desired end-of-record character, end-of-file character, and character to be ignored (removed from the text upon input or output). The values assigned to the first two parameters are ASCII code numbers, and must be in the range 0-255. The value assigned to the third parameter may be an ASCII code number, or may be in the range 128-255. If the number falls in the range 128-255, no ASCII character is deleted upon input or output.

For example, executing the command `PRI@37,0:10,4,13` prepares the EDITOR for requests for an alternate Input/Output format. If subsequent commands specify the symbol `%` instead of `@` in the I/O address, the EDITOR uses a LF (ASCII code 10) for the end-of-record character; EOT (ASCII code 4) for the end-of-file mark; and CR (ASCII code 13) for the character to be removed from incoming or outgoing text.

**Differences Between BASIC and the EDITOR.** BASIC only permits alternate Input/Output formats to be used during Input operations, by specifying a % sign in APPEND, OLD, or INPUT commands. The EDITOR, however, allows alternate Input/Output formats to be used during Output operations PRINT, SAVE, WRITE, SWN, as well as Input Operations APPEND, OLD, SKIP, and INPUT. Only the EDITOR commands LIST and SEARCH are not affected by specifying an alternate format and entering a % sign in the I/O address.

Another difference between BASIC and the EDITOR is the treatment of CR (CARRIAGE RETURN) characters upon input. BASIC cannot store CR characters in memory. When an alternate format is used and an incoming string contains a CR character, BASIC inputs characters up to the CR; but the CR character and the remainder of that logical record are lost.

However, when an alternate Input/Output format is used and the end-of-record character is a character other than CR, the EDITOR treats incoming CR characters as text characters. CR characters are stored in the text buffer, but are not treated as end-of-record characters and are not represented by the symbol specified in the ]= command.

**Resetting the Status Parameters.** Once the three parameters have been assigned values in a special PRINT command, the alternate Input/Output format is determined until the system power is turned off or another PRINT command is executed to change the values. Pressing the RETURN TO BASIC overlay key does not affect the values assigned to the three parameters.

## NOTES

## The SAVE Command

**Syntax Form:**

```
SA [ I/O address ] [ edit line number ] [ , [ edit line number ] ]
```

**Descriptive Form:**

```
SAVE [ I/O address ] [ starting line number ] [ , [ ending line number ] ]
```

### PURPOSE

The SAVE command outputs a copy of some or all of the current text to a specified peripheral device. If no peripheral device is specified, the command saves text on the file that is currently open on the internal magnetic tape.

### EXAMPLES

```
SAV  
SAV 100,  
SAV ,500  
SAV 100,500  
SAV@29:  
SAV@29:100,  
SAV@29: ,500  
SAV@29:100,500
```

### EXPLANATION

The SAVE command allows an I/O address, a starting line number, and an ending line number to be specified. When the SAVE command is executed, a copy of the current text between and including the starting and ending lines is sent to the specified peripheral device. For example, the command SAV@29:100,500 transmits lines 100 through 500 of the current text to the open file on device 29.

## I/O Commands

### SAVE

Executing a SAVE command does not change the current contents of the text buffer. When a copy of the text is transmitted to an output device, the EDITOR removes the edit line number from each line of text, and inserts an end-of-record character. The EDITOR also places an end-of-file mark after the last line recorded on the tape.

Executing the SAVE command when the tape head is positioned to the beginning of the file changes the file header name to ASCII TEXT (if this has not been previously done).

Before the SAVE command is executed, a file on the chosen peripheral device must be opened by executing a FIND command. The FIND command positions the magnetic tape to the beginning of the file, and opens the file for access. Unless a file is open on the specified device, attempting to execute the SAVE command causes a MT File error message to appear on the display.

Once the file is open, you may position the tape to a particular logical record within the file by executing a SKIP or INPUT command. Using SKIP or INPUT commands to position the tape to a particular logical record, allows you to output text to a particular location in the file. (Refer to the SKIP and INPUT commands for more detailed explanations.)

Whether the tape is positioned to the beginning of the file using the FIND command or to a particular logical record using FIND and then SKIP or INPUT, the SAVE command stores text on the tape beginning at the current position of the tape head. Any previously recorded information that lies beyond the tape head, is lost.

After the SAVE command is executed, the file is closed and no longer available for access by Input/Output commands.<sup>9</sup>

### Default Values

When no I/O address is specified in a SAVE command, the peripheral device is selected to be the internal magnetic tape by default. If the starting line number is not specified, the first line sent to the tape is the first line in the current text. If the ending line number is omitted, the last line transmitted is the last line in the current text.

For example, the command SAV outputs a copy of the entire text to the file currently open on the internal magnetic tape. SAV 100, outputs all text from line 100 on, and SAV ,500 saves text up to and including line 500.

<sup>9</sup> If the device specified in the SAVE command is the TEKTRONIX 4924 tape unit, the file is not closed. You should close the file by pressing the RETURN TO BASIC key or executing a FIND command that specifies the 4924 tape unit as the peripheral device.

## Notes on the Command Syntax

The edit delimiter (,) may be omitted if the ending line number is not specified. That is, the commands `SAV 100,` and `SAV 100` are the same. Both commands tell the EDITOR to output the current text from line 100 on, and store the text on the internal magnetic tape. To store only line 100 on the tape, you must specify 100 as both the starting and ending line number, and enter the command `SAV 100,100`.

When specifying a peripheral device in a SAVE command, do not enter any blank spaces between the I/O address and the starting line number. For example, a command such as `SAV@29: ,500` causes a syntax error.

## The Difference Between BASIC SAVE and EDITOR SAVE

Under EDITOR control you may send lines to a particular location on a tape file by using SKIP or INPUT commands to position the tape, then executing a SAVE command. In BASIC this can only be done using INPUT commands.

## Error Messages

If no file is open on the specified peripheral device, attempting to execute a SAVE command causes a `MT File` error message to appear on the display.

Executing the SAVE command causes a `Device at EOF` error message to appear if the file is not large enough to hold all of the text. When this happens, text is written on the tape file until the last byte before the physical end of the file is reached. The EDITOR places a logical end-of-file mark in the last byte of the file, and returns the `Device at EOF` error message. The remainder of the text transmitted by the SAVE command is not stored on the tape.

## The Differences Between LIST and SAVE

- As in BASIC, the LIST and SAVE commands differ in their treatment of control characters. The LIST command converts the control character to a printable symbol before sending it to the specified peripheral device; but the SAVE command does not.
- Executing the special PRINT command `PRI@37,26:1` affects the operation of the LIST command, but not the SAVE command (refer to the PRINT command for more information).
- When no I/O address is specified in a LIST command, the display is chosen to be the peripheral device by default. But when the peripheral device is omitted in a SAVE command, the internal magnetic tape is chosen by default.

## I/O Commands

### SAVE

- When the SAVE command is executed, the EDITOR removes all edit line numbers, and transmits only text characters to the specified device. However when the LIST command is executed, the EDITOR transmits both edit line numbers and text, and inserts a colon (:) immediately before the first text character in each line.

### An Editing Example

The following example shows the SAVE command being used to output specified lines of text.

#### Example 1

```
LIST
  1:C  DEFINE FORTRAN FUNCTIONS F(X) AND DF(X)
  2:    FUNCTION F(X)
  3:    F=X**3-1.473*X**2-5.738*X+6.763
  4:    RETURN
  5:    END
  6:    FUNCTION DF(X)
  7:    DF=3*X**2-2.946*X-5.738
  8:    RETURN
  9:    END
 10:C  END OF FORTRAN FUNCTION DEFINITIONS

FIN5

SAV 2,9

FIN5

OLD

LIST
  :    FUNCTION F(X)
  :    F=X**3-1.473*X**2-5.738*X+6.763
  :    RETURN
  :    END
  :    FUNCTION DF(X)
  :    DF=3*X**2-2.946*X-5.738
  :    RETURN
  :    END
```

The initial listing in Example 1 shows that the text buffer contains ten lines of FORTRAN code. The command `FIN5` is executed to open file 5 on the internal magnetic tape and position the tape to the beginning of the file. Then the command `SAV 2,9` instructs the EDITOR to store lines 2 through 9 on file 5 of the internal magnetic tape.

The next three commands input and display the contents of file 5. Because the SAVE command closed the file, another `FIND` command is executed, then the `OLD` command. A new listing shows that lines 2 through 9 of the text have been saved on file 5 as requested.

## The SKIP Command

### Syntax Form:

SK [ I/O address ] [ numeric constant ]

### Descriptive Form:

SKIP [ I/O address ] [ number of logical records to advance the READ/WRITE heads ]

## PURPOSE

The SKIP command moves the magnetic tape forward a specified number of logical records on a peripheral device. If no device is specified, the internal magnetic tape is chosen by default.

## EXAMPLES

SK

SK 10

SK@29:10

SK@29,13:10

## EXPLANATION

The SKIP command is used to position the magnetic tape to a particular logical record within a file before performing an Input/Output operation. Executing the SKIP command before an Input command allows a portion of a file to be brought into the text buffer; executing the SKIP command before an Output command allows text to be sent to a particular location in a file.

The SKIP command may be executed before any of the following EDITOR Input/Output commands: APPEND, INPUT, LIST, OLD, PRINT, SEARCH, SKIP, SAVE, SWN, and WRITE.

### NOTE

*SKIP is not intended for use before the MARK command. Attempting to execute a MARK command can destroy the contents of the magnetic tape if the tape is not positioned to the beginning of a file.*



## I/O Commands

### SKIP

The SKIP command allows an I/O address and a positive integer to be specified. When the SKIP command is executed, the EDITOR moves the tape forward the specified number of logical records on the peripheral device. For example, the command `SK@29:10` moves the tape forward 10 logical records on the file currently open on device 29.

The contents of the tape file are not altered by a SKIP command: logical records are "skipped over" by the command, but not lost. The skipped records may be accessed again by repositioning the tape using FIND and SKIP or INPUT commands.

### Default Values

If no peripheral device is specified in the SKIP command, the EDITOR moves the tape on the internal magnetic tape. For example, the command `SK 10` advances the tape 10 logical records on the internal magnetic tape.

If the number of logical records is omitted in a SKIP command, the EDITOR moves the tape forward 65536 records by default.

### Notes on the Command Syntax

When entering a SKIP command from the keyboard, do not enter any blank spaces between the I/O address and number of logical records. For example, a command such as `SK@29: 10` causes a syntax error.

### Error Messages

If a SKIP command attempts to move the tape beyond the physical or logical end of the file, a `Device at EOF` error message appears on the display, and the file is closed to further Input/Output operations. This means that the number of logical records specified in a SKIP command must not exceed the number of records that remain on the file beyond the current position of the tape head. For example, if there are three logical records between the current position of the tape head and the end-of-file mark, executing the command `SK 4` causes a `Device at EOF` error and closes the file.

### Special Uses for the SKIP Command

**Editing a Very Large File.** The SKIP command can ease the process of editing a very large file. Instead of bringing the entire file into the text buffer, input part of the file. Start by executing a FIND command, then position the tape to some record within the file using a SKIP command. Next execute an OLD command to input the portion of the file that lies beyond the current position of the tape head.

Inputting a portion of a file instead of the entire file leaves more free space in the text buffer, and thus more room for text to be inserted during editing.

When you finish editing a portion of a file, return the corrected text to its original location on the tape by reopening the file and executing the SKIP command again, followed by a SAVE, SWN, or WRITE command. You should use the same SKIP command you used before bringing the lines into the text buffer. For example, if you executed FIN3 , SK 50 , and OLD to input all but the first 50 lines of the file, the commands FIN3 , SK 50 , and SAVE output the corrected lines to their original location on the tape file.

**Editing a File that is Too Large for the Text Buffer.** The SKIP command enables you to edit a file that is too large to fit into the text buffer. If a file exceeds the current size of the text buffer, executing an OLD command causes a Text Buf. Overflow error message to appear on the display. After the message appears, the text buffer contains inputted lines up to, but not including, the line that overflowed the text buffer.

To access the entire file, proceed as follows:

1. Execute the OLD command and obtain the Text Buf. Overflow error message. At this point you may edit the portion of the file that has been successfully brought into the text buffer. If you need extra space in the text buffer for inserting new text, delete as many lines as needed from the end of the text buffer.
2. Reopen the file and execute a SKIP command that "skips over" the records already edited.
3. Execute the OLD command again. If no error message appears, the remainder of the file has been successfully brought into the text buffer and may be edited.

However if a Text Buf. Overflow message appears when the OLD command is executed, edit the current contents of the text buffer, and return to step 2 of the above procedure. Continue in this manner until the entire file has been accessed.

**Saving the Edited Text.** Each time a portion of the file is accessed and edited using the procedure outlined above, the corrected text should be saved on a second file that is at least as large as the first one. When storing text on the second file, you must use the SKIP command to ensure that each transmitted portion reaches the appropriate location on the file. After all text has been edited and stored, the second file is a complete and corrected version of the first file.

### An Editing Example

The following examples show the SKIP command being used to position the magnetic tape before Input/Output operations.

#### Example 1

```
FIN1
OLD
LIST
:
:      DO 20 I=1,NCHAR,4
:      J=4
:      IF(NCHAR-I.LT.3) J=NCHAR-I+1
:      K=(I+3)/4
:      JEN=I+J
:      ENCODE(4.10,KA4(K) (KA1(J),J=1,JEN)
:10    FORMAT (A4)
:20    CONTINUE
:      RETURN
:      END
```

```
FIN1
SK 3
OLD
LIST
:
:      K=(I+3)/4
:      JEN=I+J
:      ENCODE(4.10,KA4(K) (KA1(J),J=1,JEN)
:10    FORMAT (A4)
:20    CONTINUE
:      RETURN
:      END
```

#### Example 2

```
FIN1
SK 6
INP
10    FORMAT (A4)
```

## Example 3

```

FIN2
OLD
LIST
      :      J=J+1
      :      IF (J.GT.I) RETURN
      :      N=N+1

DEL 0,0+3

I
      :      IF (BUFF(J).NE.74B) GO TO 40
      :      IN(N)=TABL63(BUFF(J+1)+1,2)
      :      J=J+1
      :      GO TO 30

FIN2
SK 3
SAV

FIN2
OLD
LIST
      :      J=J+1
      :      IF (J.GT.I) RETURN
      :      N=N+1
      :      IF (BUFF(J).NE.74B) GO TO 40
      :      IN(N)=TABL63(BUFF(J+1)+1,2)
      :      J=J+1
      :      GO TO 30

```

Example 1 shows the SKIP command being used before the OLD command. First the commands FIN1 , OLD , and LIST are executed to input and list on the display the contents of the first file from the internal magnetic tape. The listing shows that the file consists of ten lines of FORTRAN code.

The command FIN1 reopens the file and positions the tape to the beginning of the file. Next the command SK 3 advances the tape three logical records on file 1. Then the command OLD is executed.

Because the tape is positioned to the beginning of the fourth record on the file, the first three logical records of file 1 are "skipped" when the OLD command is executed. A listing shows that only the last seven lines from file 1 have been brought into the text buffer for editing.

In Example 2 the SKIP command is executed before the INPUT command. First the command FIN1 reopens file 1 and positions the tape to the beginning of the file. Next the command SK 6 advances the tape six logical records on file 1. Then the command INP is executed.

## I/O Commands

### SKIP

Because the tape is positioned to the beginning of the seventh record on the file, executing the INPUT command causes the seventh logical record of file 1 to appear on the display.

Example 3 shows the SKIP command being used before the Output command SAVE. First the command FIN2 , OLD , and LIST are executed to input and list on the display the contents of file 2 of the internal magnetic tape. The listing shows that the file consists of three lines of FORTRAN code. Then the text buffer is cleared by the command DEL 0,0+3 and the INSERT command is used to create four new lines of code.

The command FIN2 reopens the file and positions the tape to the beginning of the file; then SK 3 advances the tape three records on file 2; and the command SAV is executed. Because the tape is positioned to the beginning of the fourth record on the file, the first three logical records are "skipped" when the SAVE command is executed. After the SAVE command is executed, a new listing shows that the newly created lines have been stored immediately after the third logical record on file 2.

## The SWN (Save With Number) Command

### Syntax Form:

```
SWN [ I/O address ] [ edit line number ] [ , [ edit line number ] ]
```

### Descriptive Form:

```
SWN [ I/O address ] [ starting line number ] [ , [ ending line number ] ]
```

## PURPOSE

The SWN command sends a copy of some or all of the current text to a specified peripheral device. Edit line numbers are saved as part of the text. If no peripheral device is specified, the SWN command sends lines of text and their edit line numbers to the file that is currently open on the internal magnetic tape.

## EXAMPLES

```
SWN
```

```
SWN 100,
```

```
SWN ,500
```

```
SWN 100,500
```

```
SWN@29:
```

```
SWN@29:100,
```

```
SWN@29: ,500
```

```
SWN@29:100,500
```

## EXPLANATION

The SWN (Save With Number) command allows an I/O address, a starting line number, and an ending line number to be specified. When the SWN command is executed, a copy of the current text between and including the starting and ending lines is sent to the specified peripheral device. Edit line numbers are transmitted along with the text. For example, the command `SWN@29:100,500` outputs lines 100 through 500 along with their edit line numbers to the open file on device 29.

## I/O Commands

### SWN (Save With Number)

Edit line numbers are sent to the device, but not the colon (:) the EDITOR uses to separate edit line numbers from text in listings.

Executing a SWN command does not change the current contents of the text buffer. When a copy of the line numbers and text is transmitted to an output device, the EDITOR inserts an end-of-record character for each line of text, and places an end-of-file mark after the last line recorded on the tape.

Executing the SWN command when the tape is positioned to the beginning of a file changes the file header name to ASCII TEXT (if this has not been previously done).

When the SWN command is executed, one blank line is created on the file immediately before the first line stored by the command. If the stored text is a BASIC program, the blank line cannot be detected by the BASIC Interpreter.

Before the SWN command is executed, a file on the chosen peripheral device must be opened by executing a FIND command. The FIND command positions the magnetic tape to the beginning of the file, and opens the file for access. Unless a FIND command is executed to position the tape and open a file, attempting to execute the SWN command causes a MT File error message to appear on the display.

Once the file is open, you may position the tape to a particular logical record within the file by executing a SKIP or INPUT command. Using SKIP or INPUT commands to position the tape to a particular logical record, allows you to output text to a particular location in the file. (Refer to the SKIP and INPUT commands for more detailed explanations.)

Whether the tape is positioned to the beginning of the file using the FIND command or to a particular logical record using FIND and then SKIP or INPUT, the SWN command stores text on the tape beginning at the current position of the tape head. Any previously recorded information that lies beyond the tape head, is lost.

After the SWN command is executed, the file is closed and no longer available for access.

### Format

Edit line numbers saved by the SWN command become part of the text and no longer serve as edit line numbers. When the text is brought back into the text buffer and listed on the display, the format is as follows: the colon that precedes each line of text is followed by a blank space. The next four character positions in the line contain the edit line number saved by the SWN command. One or more of these positions may be blank, depending on the number of digits in the line number, and all four positions are blank if the line was unnumbered. Another blank space follows, then the remainder of the text begins in the seventh character position. For an illustration of the format, see "An Editing Example" in this explanation.

## Default Values

When no I/O address is specified in a SWN command, the peripheral device is selected to be the internal magnetic tape by default. If the starting line number is not specified, the first line sent to the tape is the first line in the current text. If the ending line number is omitted, the last line transmitted is the last line in the current text.

For example, the command `SWN` outputs a copy of the entire text including edit line numbers to the file currently open on the internal magnetic tape. `SWN 100,` outputs all line numbers and text from line 100 on, and `SWN ,500` saves line numbers and text up to and including line 500.

## Notes on the Command Syntax

The edit delimiter (,) may be omitted if the ending line number is not specified. That is, the commands `SWN 100,` and `SWN 100` are the same. Both commands tell the EDITOR to output the current edit line numbers and text from line 100 on. To store only line 100 on the tape you must specify 100 as both the starting and ending line number, and enter the command `SWN 100,100` .

When specifying a peripheral device in a SWN command, do not enter any blank spaces between I/O address and the starting line number. For example, the command `SWN@29: 100,500` causes a syntax error.

## The Difference Between SWN and SAVE

The only difference between the SWN and SAVE commands is that SWN stores edit line numbers along with the text, but the SAVE command does not. If the specified lines of text are unnumbered, the SWN and SAVE commands perform the same function; however, six blank spaces precede each line stored by the SWN command. (These six spaces include the four that would have been used to store the edit line number if the line had been numbered.)

## Differences Between SWN and LIST

One difference between the SWN and LIST commands is that SWN closes the file, but LIST does not. That is, the LIST command can be executed repeatedly without executing another FIND command, but the SWN command cannot.

Another difference is that the LIST command outputs a colon (:) before the first text character in each line, but the SWN command does not.

The LIST and SWN commands also differ in their treatment of control characters. The LIST command converts the control character to a printable symbol before sending it to the specified device; but the SWN command does not.



## **Error Messages**

If no file is open on the internal magnetic tape, executing a SWN command to output text to the tape causes a `MT File` error message to appear on the display.

Executing the SWN command causes a `Device at EOF` error message to appear if the file is not large enough to hold all of the edit line numbers and text. When this happens, text is written on the tape file until the last byte before the physical end of the file is reached. The EDITOR places a logical end-of-file mark in the last byte of the file, and returns the `Device at EOF` error message. The remainder of the line numbers and text transmitted by the SWN command are not stored on the tape.

For information about how large a file must be marked in order to store the current text, refer to the explanation of the LASTLINE command.

## **Special Uses for the SWN Command**

The SWN command can be used to create BASIC program line numbers. When using the EDITOR to write a BASIC program, you may enter the statements without program line numbers, execute an appropriate RENUMBER command, then use the SWN command to convert *edit* line numbers into *program* line numbers. This method is illustrated below in "An Editing Example."

## **An Editing Example**

The following example shows the SWN command being used to output specified lines of text along with their edit line numbers.

### **Example 1**

```
LIST
 460:FOR R=K+1 TO N
 470:IF ABS(A(R,K)) <= ABS(A(L,K)) THEN 490
 480:L=R
 490:NEXT R
 500:IF L=K THEN 610
 510:FOR P=1 TO N+1
 520:T=A(K,P)
 530:A(K,P)=A(L,P)
 540:A(L,P)=T
 550:NEXT P

FIN3

SWN
```

(continued on next page)

```
FIN3
OLD
LIST
:
: 460 FOR R=K+1 TO N
: 470 IF ABS(A(R,K)) <= ABS(A(L,K)) THEN 490
: 480 L=R
: 490 NEXT R
: 500 IF L=K THEN 610
: 510 FOR P=1 TO N+1
: 520 T=A(K,P)
: 530 A(K,P)=A(L,P)
: 540 A(L,P)=T
: 550 NEXT P
```

Example 1 illustrates how you may use the SWN command to create BASIC program line numbers, instead of entering each line number from the keyboard. The initial listing shows that the text consists of ten BASIC program statements entered without program line numbers. Prior to this listing, a RENUMBER command was executed to assign edit line numbers 460, 470, 480, ... to the text.

File 3 on the internal magnetic tape is opened using the command FIN3 , then the SWN command is executed. When the SWN command is executed, each BASIC program statement is stored on file 3, along with the edit line number that preceded the statement in the text buffer.

To display the results, file 3 is reopened and the text brought back into the text buffer using the commands FIN3 and OLD . A new listing shows that the numbers previously assigned as edit line numbers have been incorporated into the text. The numbers now appear as BASIC program line numbers preceding each statement.

## NOTES

## The WRITE Command

**Syntax Form:**

W [ I/O address ] [ edit line number ] [ , [ edit line number ] ]

**Descriptive Form:**

WRITE [ I/O address ] [ starting line number ] [ , [ ending line number ] ]

### PURPOSE

The WRITE command outputs a copy of some or all of the current text to a specified peripheral device. If no peripheral device is specified, the command writes text on the file that is currently open on the internal magnetic tape. After text is written on a file, the file remains open for access by Output operations.

The WRITE and SAVE commands perform similar functions. The only difference is that after storing text on a tape file, the SAVE command closes the file, but the WRITE command leaves the file open to Output operations.

### EXAMPLES

W

W 100,

W ,500

W 100,500

W@29:

W@29:100,

W@29:,500

W@29:100,500

**EXPLANATION**

The WRITE command allows an I/O address, a starting line number, and an ending line number to be specified. When the WRITE command is executed, a copy of the current text between and including the starting and ending lines is sent to the specified peripheral device. For example, the command `W@29:100,500` transmits lines 100 through 500 of the current text to the open file on device 29.

Executing a WRITE command does not change the current contents of the text buffer. When a copy of the text is transmitted to an output device, the EDITOR removes the edit line number from each line of text, and inserts an end-of-record character. The EDITOR also places an end-of-file mark after the last line recorded on the tape.

Executing the WRITE command when the tape head is at the beginning of the file changes the file header name to `ASCII TEXT` (if this has not been previously done).

Before executing the WRITE command, you must open a file on the chosen peripheral device by executing a FIND command. The FIND command positions the magnetic tape to the beginning of the file, and opens the file for access. Unless a FIND command is executed to position the tape and open a file, attempting to execute the WRITE command causes a `MT File` error message to appear on the display.

Once the file is open, you may position the tape to a particular logical record within the file by executing a SKIP or INPUT command. Using SKIP or INPUT commands to position the tape to a particular logical record, allows you to output text to a particular location in the file. (Refer to the SKIP and INPUT commands for more detailed explanations.)

Whether the tape is positioned to the beginning of the file using the FIND command, or to a particular logical record using FIND and then SKIP or INPUT, the WRITE command stores text on the tape beginning at the current position of the tape head. Any previously recorded information that lies beyond the tape head is lost.

After the WRITE command is executed, the file remains open and available to Output operations. Subsequent Output commands overwrite the end-of-file mark left by the WRITE command, and insert a new end-of-file mark when the operation is finished.

For example, a SAVE command can be executed after a WRITE command, without reopening the file. The SAVE command overwrites the previous end-of-file mark, and places a new end-of-file mark at the end of the saved lines.

The WRITE command can be executed repeatedly to output portions of the text. Once a WRITE command has been executed, each subsequent WRITE command overwrites the end-of-file mark left by the previous WRITE command, and begins storing text at the current position of the tape head. Thus you may edit and save text in small portions, by executing a WRITE command each time you finish editing a section of the text.

For instance, the commands `W 1,4` then `W 5,10` then `W 11,20` may be executed to transmit lines through 20 of the current text to the internal magnetic tape. The final result of executing the three commands is the same as if the command `W 1,20` had been executed.

After executing a `WRITE` command and before turning the system power off or removing the internal magnetic tape unit, close the file by executing a `FIND` command or pressing the `RETURN TO BASIC` overlay key. Closing the file forces any information remaining in the magnetic tape buffer onto the tape, so that no transmitted text is lost.

### **Default Values**

When no I/O address is specified in a `WRITE` command, the peripheral device is selected to be the internal magnetic tape by default. If the starting line number is not specified, the first line sent to the tape is the first line in the current text. If the ending line number is omitted, the last line transmitted is the last line in the current text.

For example, the command `W` outputs a copy of the entire text to the file currently open on the internal magnetic tape. `W 100,` outputs all text from line 100 on, and `W ,500` saves text up to and including line 500.

### **Notes on the Command Syntax**

The edit delimiter (,) may be omitted if the ending line number is not specified. That is, the commands `W 100,` and `W 100` are the same. Both commands tell the `EDITOR` to output the current text from line 100 on, and store the text on the internal magnetic tape. To store only line 100, you must specify 100 as both the starting and ending line number, and enter the command `W 100,100` .

When specifying a peripheral device in a `WRITE` command, do not enter any blank spaces between the I/O address and the starting line number. For example, the command `W@29: 100,500` causes a syntax error.

### **The Difference Between BASIC WRITE and EDITOR WRITE**

The `EDITOR`'s `WRITE` command is completely different from `BASIC`'s `WRITE` command. In `BASIC`, the `WRITE` command sends specified data items to a peripheral device in machine dependent binary code. However, the `EDITOR` command `WRITE` outputs text to a peripheral device in `ASCII` code.

### WRITE

#### Error Messages

If no file is open on the specified peripheral device, attempting to execute a WRITE command causes a `MT File` error message to appear on the display.

After a WRITE command is executed, the file remains open to Output operations only. Attempting to execute an APPEND, OLD, INPUT, or SKIP command without reopening the file causes an error. A `Device Access` or `Buffer Access` error message appears on the display. Both messages mean that the specified device is not available for access by Input operations, and that the magnetic tape buffer cannot receive information from the device.

Executing the WRITE command to the internal magnetic tape causes a `Device at EOF` error message to appear if the file is not large enough to hold all of the text. When this happens, text is written on the tape file until the last byte before the physical end of the file is reached. The EDITOR places a logical end-of-file mark in the last byte of the file, and returns the `Device at EOF` error message. The remainder of the text transmitted by the WRITE command is not stored on the tape.

For more information about how large a file must be marked in order to store the current text, refer to the explanation of the LASTLINE command.

#### The Difference Between WRITE and SAVE

The only difference between the WRITE and SAVE commands is that the SAVE command closes the file, but the WRITE command does not. That is, the WRITE command can be executed repeatedly without executing another FIND command, but the SAVE command cannot.

## An Editing Example

The following example shows the WRITE command being used to output specified lines of text.

### Example 1

```
LIST
550:      READ(5,1)BUFF
560:1     FORMAT(500R1)
570:C     ROUTINE TO INPUT STRINGS OF LESS THAN
580:C     500 CHARS
590:      I=501
600:10    CONTINUE
610:      I=I-1
620:      IF (I.EQ.1) GO TO 20
630:      IF (BUFF(I).EQ.55B) GO TO 10
640:20    CONTINUE
650:C     DECODE LOOP
660:      N=0
670:      J=0
680:30    CONTINUE
```

FIN3

W ,560

W 590,640

W 660,

FIN3

OLD

```
LIST
:        READ(5,1)BUFF
:1       FORMAT(500R1)
:        I=501
:10      CONTINUE
:        I=I-1
:        IF (I.EQ.1) GO TO 20
:        IF (BUFF(I).EQ.55B) GO TO 10
:20      CONTINUE
:        N=0
:        J=0
:30      CONTINUE
```

Three WRITE commands are executed in Example 1. The initial listing shows that the text consists of fourteen lines of FORTRAN code. The command FIN3 is executed to open file 3 on the internal magnetic tape and position the tape to the beginning of the file.



## I/O Commands

### WRITE

Next the three WRITE commands are executed. The command `W ,560` tells the EDITOR to output the text up to and including line 560 to file 3 on the internal magnetic tape. The command `W 590,640` tells the EDITOR to output lines 590 through 640, and `W 660,` outputs the text from line 660 on.

To display the results, the contents of file 3 are brought into the text buffer using the command `FIN3` and `OLD` . A new listing shows that the specified portions of the text have been saved in order on the tape file.

# Appendix A

## ERROR MESSAGES

<b>Error Number</b>	<b>Meaning</b>
<b>129</b>	A peripheral device on the GPIB (General Purpose Interface Bus) is requesting service.
<b>132</b>	A RENUMBER command has attempted to assign an edit line number larger than 9999 to one or more lines of text. Execute a new RENUMBER command that specifies either a smaller increment, or a smaller first new edit line number.
<b>133</b>	Text inserted from the keyboard using the INSERT command, or brought from a peripheral device using OLD or APPEND, has exceeded the current length of the text buffer. The text buffer contains inserted or inputted text up to, but not including, the line that overflowed the buffer. To increase the size of the text buffer, you may 1) delete several lines of text; or 2) return to BASIC, delete variables and programs currently stored in memory, then call the EDITOR again.
<b>134</b>	Characters have been entered too rapidly from the keyboard, overflowing the "queue" buffer that acts as an intermediary between the keyboard and the line buffer. The error is not fatal. However, the characters that overflowed the queue are lost.
<b>138</b>	An incorrect command has been entered from the keyboard.
<b>139</b>	An incorrect or meaningless command has been entered from the keyboard.
<b>141</b>	1) An attempt has been made to input text from a magnetic tape file that is currently open to Output operations only. An example is executing the command W 100,500 and then OLD .  2) A peripheral device has attempted to use the magnetic tape buffer while information from a different device is still in the magnetic tape buffer. An example is executing the command W@3:100,500 and then INP .

Error Number	Meaning
143-144-145	1) An attempt has been made to execute an Input/Output command while no file is open on the specified peripheral device. A file must be opened for access by executing a FIND command.  2) An Input command (APPEND, INPUT, SKIP, or OLD) has attempted to read from a file marked NEW or LAST .
146	1) An Input command has attempted to read beyond the logical or physical end of the file. An example is executing the command SKIP 20 when the open file on the internal magnetic tape consists of fewer than 20 logical records.  2) An Output command has attempted to write beyond the physical end of the file.
147	Same as error number 141. An example is executing the command W 100,500 and then INP .
148	An incorrect command has been entered from the keyboard. (Same as error number 138.)

In addition to those described on the preceding pages, EDITOR ERROR messages may appear that are the same as BASIC error messages. Error messages that are "borrowed" from BASIC are preceded by the line EDITOR ERROR , immediately followed by a blank line, and then a message that gives an error number smaller than 129. The most common of these errors are described below. If you encounter others, refer to Appendix A in the Graphic System Reference Manual.

Error Number	Meaning
52	A non-existent tape file has been specified in a FIND command.
53	The EDITOR is unable to read a portion of the current internal magnetic tape file. An example is attempting to input a file under non-header format that was created under header format.
54	The end of the magnetic tape has been detected. An example is executing a MARK command that attempts to mark files beyond the end of the magnetic tape.
56	An attempt has been made to send information to a file marked SECRET .
57	An attempt has been made to read or write to a non-existent tape cartridge. Insert a tape cartridge into the tape slot and execute the operation again.
58	An attempt has been made to read text that is stored in an invalid magnetic tape format. Executing a MARK command while the tape is not positioned to the beginning of a file, can create an invalid magnetic tape format. When this happens, the tape must be re-marked.
67	An attempt has been made to execute an illegal Input/Output operation on an internal peripheral device. An example is executing the command A@32:200 .
69	<p>1) An input error has occurred on the General Purpose Interface Bus. This usually means that there are no devices connected to the GPIB.</p> <p>2) The primary address specified in an Input/Output command is not within the range 0 through 255.</p>

# Appendix B

## TABLES

ASCII CODE CHART

NUL <sup>0</sup>	DLE <sup>16</sup>	SP <sup>32</sup>	Ø <sup>48</sup>	@ <sup>64</sup>	P <sup>80</sup>	\ <sup>96</sup>	p <sup>112</sup>
SOH <sup>1</sup>	DC1 <sup>17</sup>	! <sup>33</sup>	1 <sup>49</sup>	A <sup>65</sup>	Q <sup>81</sup>	a <sup>97</sup>	q <sup>113</sup>
STX <sup>2</sup>	DC2 <sup>18</sup>	" <sup>34</sup>	2 <sup>50</sup>	B <sup>66</sup>	R <sup>82</sup>	b <sup>98</sup>	r <sup>114</sup>
ETX <sup>3</sup>	DC3 <sup>19</sup>	# <sup>35</sup>	3 <sup>51</sup>	C <sup>67</sup>	S <sup>83</sup>	c <sup>99</sup>	s <sup>115</sup>
EOT <sup>4</sup>	DC4 <sup>20</sup>	\$ <sup>36</sup>	4 <sup>52</sup>	D <sup>68</sup>	T <sup>84</sup>	d <sup>100</sup>	t <sup>116</sup>
ENQ <sup>5</sup>	NAK <sup>21</sup>	% <sup>37</sup>	5 <sup>53</sup>	E <sup>69</sup>	U <sup>85</sup>	e <sup>101</sup>	u <sup>117</sup>
ACK <sup>6</sup>	SYN <sup>22</sup>	& <sup>38</sup>	6 <sup>54</sup>	F <sup>70</sup>	V <sup>86</sup>	f <sup>102</sup>	v <sup>118</sup>
BEL <sup>7</sup> BELL	ETB <sup>23</sup>	/ <sup>39</sup>	7 <sup>55</sup>	G <sup>71</sup>	W <sup>87</sup>	g <sup>103</sup>	w <sup>119</sup>
BS <sup>8</sup> BACK SPACE	CAN <sup>24</sup>	( <sup>40</sup>	8 <sup>56</sup>	H <sup>72</sup>	X <sup>88</sup>	h <sup>104</sup>	x <sup>120</sup>
HT <sup>9</sup>	EM <sup>25</sup>	) <sup>41</sup>	9 <sup>57</sup>	I <sup>73</sup>	Y <sup>89</sup>	i <sup>105</sup>	y <sup>121</sup>
LF <sup>10</sup>	SUB <sup>26</sup>	* <sup>42</sup>	: <sup>58</sup>	J <sup>74</sup>	Z <sup>90</sup>	j <sup>106</sup>	z <sup>122</sup>
VT <sup>11</sup>	ESC <sup>27</sup>	+ <sup>43</sup>	; <sup>59</sup>	K <sup>75</sup>	[ <sup>91</sup>	k <sup>107</sup>	{ <sup>123</sup>
FF <sup>12</sup>	FS <sup>28</sup>	, <sup>44</sup>	< <sup>60</sup>	L <sup>76</sup>	\ <sup>92</sup>	l <sup>108</sup>	: <sup>124</sup>
CR <sup>13</sup> RETURN	GS <sup>29</sup>	- <sup>45</sup>	= <sup>61</sup>	M <sup>77</sup>	] <sup>93</sup>	m <sup>109</sup>	} <sup>125</sup>
SO <sup>14</sup>	RS <sup>30</sup>	· <sup>46</sup>	> <sup>62</sup>	N <sup>78</sup>	^ <sup>94</sup>	n <sup>110</sup>	~ <sup>126</sup>
SI <sup>15</sup>	US <sup>31</sup>	/ <sup>47</sup>	? <sup>63</sup>	O <sup>79</sup>	_ <sup>95</sup>	o <sup>111</sup>	RUBOUT (DEL) <sup>127</sup>

**4051R06 EDITOR  
COMMAND SUMMARY**

Command	Example	Action Taken
<b>EDITING COMMANDS</b>		
<b>CARD</b>	<b>CA 80,42</b>	Formats the text buffer into lines that are 80 characters long, splitting lines of text having more than 80 characters into 2 or more lines, and using the character * (decimal equivalent 42) to fill out any lines of text that have less than 80 characters.
<b>CASE</b>	<b>CAS 1,100</b>	In lines 1 through 100 of the text buffer, lower case characters a-z are replaced by their uppercase equivalents A-Z, if the UPPERCASE flag has been set. If the LOWERCASE flag has been set, uppercase characters A-Z are replaced by their lowercase equivalents a-z.
<b>COPY</b>	<b>C 1,3,10</b>	Duplicates lines 1 through 3 of the text buffer, placing the copied text immediately before line 10.*
<b>DELETE</b>	<b>D 1,10</b>	Deletes lines 1 through 10 from the text buffer.
<b>INSERT</b>	<b>I 100</b>	Responds with five spaces and a colon (:) to prompt the entry of new text from the keyboard. Lines of text entered after the colon are placed in the text buffer immediately before the line 100.
<b>LIST</b>	<b>L @29:200,300</b>	Lists lines 200 through 300 of the text buffer on peripheral device 29 on the General Purpose Interface Bus.
<b>MOVE</b>	<b>M 1,3,10</b>	Moves lines 1 through 3 of the text buffer, placing them immediately before line 10.**
<b>NLSEARCH and Delete Line</b>	<b>NL 0,1000 "REM"*</b>	Searches lines 0 through 1000 of the text buffer, and deletes lines found to contain the string REM .
<b>NLSEARCH and Replace String</b>	<b>NL 1,100 "ON ERROR","ON SIZE"</b>	Searches lines 1 through 100 of the text buffer, and replaces occurrences of the string ON ERROR with the string ON SIZE .
<b>SEARCH and List Line</b>	<b>S @29:0,2000 "638-"</b>	Searches lines 0 through 2000 of the text buffer, and lists on device 29 all lines found to contain the string 638- .
<b>SEARCH and Edit Line</b>	<b>S 1,100 "Mr. ",</b>	Searches lines 1 through 100 of the text buffer for the string Mr. . One by one, lines found to contain the string are recalled to the line buffer, and wait to be edited.
<b>SEARCH and Delete Line</b>	<b>S @29:0,3000 "pd."*</b>	Searches lines 0 through 3000 of the text buffer for the string pd. . Lines found to contain the string are listed on device 29, and deleted from the text buffer.
<b>SEARCH and Replace String</b>	<b>S @29:1,1000 "PRI","PRI @3:"</b>	Searches lines 1 through 1000 of the text buffer and replaces occurrences of the string PRI with the string PRI @3: . Changed lines are listed on device 29.

\*See end of table

\*\*See end of table

**4051R06 EDITOR  
COMMAND SUMMARY (cont.)**

<b>Command</b>	<b>Example</b>	<b>Action Taken</b>
<b>SORT</b>	<b>SO 1,10:1,2,3</b>	Rearranges lines 1 through 10 in the text buffer, sorting "alphabetically" according to the ASCII values of characters found in the first three character positions within each line.
<b>REVSORT</b>	<b>REV 1,10:1,2,3</b>	Rearranges lines 1 through 10 in the text buffer, sorting "alphabetically in reverse" according to the ASCII values of characters found in the first three character positions within each line.
<b>ENVIRONMENTAL COMMANDS</b>		
<b>LASTLINE</b>	<b>LA</b>	Returns the following information about the current status of the text buffer: <ul style="list-style-type: none"> <li>— The last edit line number in the text buffer (including offset if the line has no number).</li> <li>— The total number of lines in the text buffer.</li> <li>— The number of bytes needed to save the current contents of the buffer on a storage device.</li> <li>— The number of unused bytes remaining in the text buffer.</li> </ul>
<b>LOWERCASE</b>	<b>LO</b>	Enables the EDITOR to distinguish between lowercase characters a-z and their uppercase equivalents A-Z during searching and sorting operations. Prepares the EDITOR to change uppercase characters into lowercase characters if the CASE command is executed.
<b>UPPERCASE</b>	<b>U</b>	Causes the EDITOR to treat lowercase characters a-z in the text buffer as uppercase characters A-Z during searching and sorting operations. Prepares the EDITOR to change lowercase characters into uppercase characters if the CASE command is executed.
<b>RENUMBER</b>	<b>R 100,10,3</b>	Renumsbers all lines in the text buffer starting with the line currently numbered 3. The new edit line numbers start at 100 and increase with an increment of 10.
<b>] =</b>	<b>] = /</b>	Makes the character / stand for "END-OF-RECORD." The character / may be used during line editing to insert end-of-record characters into the text, and may appear in target and replacement strings during searching operations.
<b># =</b>	<b># = *</b>	Makes the character * stand for "any digit 0 to 9," so that any of the digits 0 through 9 satisfy a search for the character*.
<b>~ =</b>	<b>~ = ?</b>	Makes the character ? stand for "any character." Any ASCII character satisfies a search for the character ? When used in a replacement string, ? indicates that the ASCII character found in that position should remain unchanged by the replacement procedure.
<b>_ =</b>	<b>_ = +</b>	Allows the character + to be used as a prefix meaning "all but," so that all characters satisfy a search except the one immediately following +. For instance, the command S "+A" searches the text buffer for all ASCII characters except A.



**4051R06 EDITOR  
COMMAND SUMMARY (cont.)**

Command	Example	Action Taken
<b>Special PRINT Commands</b>		
<b>Processor Status</b>	<b>PRI @37,0:10,4,13</b>	Prepares the microprocessor for requests for an alternate Input/Output format. When a % sign is used in an I/O command instead of @, the special format is as follows: end-of-record = LF (ASCII 10) end-of-file = EOT (ASCII 4) character to ignore = CR (ASCII 13)
	<b>PRI @37,26:1</b>	Tells the microprocessor to send a line feed (LF) character after each CR in the text when listing text on external devices (LIST and SEARCH commands).
	<b>PRI @37,26:0</b>	Tells the microprocessor to send a CR (instead of CR and LF) after each line when listing text on external devices.
<b>Magnetic Tape Status</b>	<b>PRI @33,0:1,1,1</b>	Sends the following status information to the microprocessor: — Format the tape into 128-byte physical records. — Do not use the checksum error checking technique. — Do not use file header format.
	<b>PRI @33,0:0,0,0</b>	Sends the following status information to the microprocessor: — Format the tape into 256-byte physical records. — Use the checksum error checking technique. — Use file header format.
<b>I/O COMMANDS</b>		
<b>APPEND</b>	<b>A @29:50</b>	Adds logical records to the text buffer from the file currently open on device 29. The incoming text is added immediately before line 50 of the text buffer.
<b>FIND</b>	<b>F @29:4</b>	Positions the READ/WRITE heads to the beginning of file 4 on device 29, and opens the file for access by Input/Output operations.
<b>INPUT</b>	<b>INP @29:</b>	Displays one logical record from the file currently open on device 29, and positions the READ/WRITE heads over the next record on file.
<b>MARK</b>	<b>MA 2,5120</b>	Reserves space on the internal magnetic tape for 2 new files, starting at the current position of the tape heads. 5120 bytes of storage are reserved for each file.
<b>OLD</b>	<b>O @29:</b>	Clears the text buffer, then brings logical records into the text buffer from the current file on device 29.
<b>PRINT</b>	<b>P @29: List of File 3</b>	Prints List of File 3 on device 29.
<b>SAVE</b>	<b>SA @29:100,500</b>	Stores an unnumbered copy of text buffer lines 100 through 500 on the file currently open on device 29. Once the lines are saved, the file is closed to access by Input/Output operations.
<b>SWN</b>	<b>SWN @29:100,500</b>	Stores a numbered copy of text buffer lines 100 through 500 on the file currently open on device 29. Once the lines and edit line numbers are saved, the file is closed to access by Input/Output operations.

**4051R06 EDITOR  
COMMAND SUMMARY (cont.)**

<b>Command</b>	<b>Example</b>	<b>Action Taken</b>
<b>SKIP</b>	<b>SK @29:10</b>	Moves the READ/WRITE heads 10 logical records forward on the current file on device 29. The portion of the file beyond the new position of the READ/WRITE heads remains open for access by Input/Output operations.
<b>WRITE</b>	<b>W @29:100,500</b>	Stores an unnumbered copy of text buffer lines 100 through 500 on the file currently open on device 29. Once the lines are stored, the file remains open for access by Output operations.

\*If a non-existent line number is specified in an EDITOR command, the EDITOR automatically uses the line having the next largest edit line number.

\*\*A portion of the text buffer may include lines which do not have edit line numbers. EDITOR commands act upon all lines of the text buffer (numbered or not) which fall between the specified starting and ending line numbers.

**TABLE OF EDITOR COMMANDS  
AND DEFAULT PARAMETER VALUES**

Command	Syntax (Descriptive Form)	Default Values
<b>APPEND</b>	APPEND { I/O address } [ destination for appended text ]	I/O address = @33,4: destination = after the last line of text
<b>CARD</b>	CARD [ number of characters ] [ , [ fill character (decimal equivalent) ] ]	number of characters = 80 fill character = space
<b>CASE</b>	CASE [ starting line number ] [ , [ ending line number ] ]	starting line number = first line of text ending line number = last line of text
<b>COPY</b>	COPY [ starting line number ] [ , [ ending line number ] [ , [ destination for copied text ] ] ]	starting line number = first line of text ending line number = last line of text destination = after the last line of text
<b>DELETE</b>	DELETE starting line number [ , [ ending line number ] ]	ending line number = starting line number
<b>FIND</b>	FIND [ I/O address ] [ file number ]	I/O address = @33,27: file number = 0
<b>INPUT</b>	INPUT [ I/O address ]	I/O address = @33,13:
<b>INSERT</b>	INSERT [ destination for inserted lines of text ]	destination = before the first line of text
<b>LASTLINE</b>	LASTLINE	
<b>LIST</b>	LIST [ I/O address ] [ starting line number ] [ , [ ending line number ] ]	I/O address = @32,19: starting line number = first line of text ending line number = last line of text
<b>LOWERCASE</b>	LOWERCASE	LOWERCASE
<b>MARK</b>	MARK [ number of files ] [ , [ number of bytes per file ] ]	number of files = 0 number of bytes per file = 768
<b>MOVE</b>	MOVE [ starting line number ] [ , [ ending line number ] [ , [ destination for moved text ] ] ]	starting line number = first line of text ending line number = last line of text destination = after the last line of text
<b>NLSEARCH</b>	NLSEARCH [ [ starting line number ] , [ ending line number ] , ] space " target string " { * " replacement string " }	starting line number = first line of text ending line number = last line of text

**TABLE OF EDITOR COMMANDS  
AND DEFAULT PARAMETER VALUES  
(cont)**

Command	Syntax (Descriptive Form)	Default Values
<b>OLD</b>	OLD [ I/O address ]	I/O address = @33,4:
<b>PRINT</b>	PRINT [ I/O address ] [ ASCII character string ]	I/O address = @32,12:
<b>RENUMBER</b>	RENUMBER [ new starting line number ] [ . [ increment between new line numbers ] [ . [ line in the current text where renumbering is to begin ] ]	new starting line number = 1 increment between new edit line numbers = 1 where to begin renumbering = first line of text
<b>REVSORT</b>	REVSORT [ starting line number ] , [ ending line number ] , character position [ , character position [ . . . . ] ]	starting line number = first line of text ending line number = last line of text
<b>SAVE</b>	SAVE [ I/O address ] [ starting line number ] [ . [ ending line number ] ]	I/O address = @33,12: starting line number = first line of text ending line number = last line of text
<b>SEARCH</b>	SEARCH [ I/O address ] [ { starting line number } , { ending line number } , ] space " target string " [ { * " replacement string " } ]	I/O address = @32,19: starting line number = first line of text ending line number = last line of text
<b>SKIP</b>	SKIP [ I/O address ] [ number of logical records to advance the READ/WRITE heads ]	number of logical records = 65536
<b>SORT</b>	SORT [ starting line number ] , [ ending line number ] , character position [ , character position [ . . . . ] ]	starting line number = first line of text ending line number = last line of text
<b>SWN (Save With Number)</b>	SWN [ I/O address ] [ starting line number ] [ . [ ending line number ] ]	I/O address = @33,12: starting line number = first line of text ending line number = last line of text
<b>UPPERCASE</b>	UPPERCASE	LOWERCASE
<b>WRITE</b>	WRITE [ I/O address ] [ starting line number ] [ . [ ending line number ] ]	I/O address = @33,12: starting line number = first line of text ending line number = last line of text

**TABLE OF EDITOR COMMANDS  
AND DEFAULT PARAMETER VALUES  
(cont)**

Command	Syntax (Descriptive Form)	Default Values
]=	]= [ ASCII character ]	initial value: ] = ] default: no assignment
# =	# = [ ASCII character ]	initial value: # = # default: no assignment
~ =	~ = [ ASCII character ]	initial value: ~ = ~ default: no assignment
_ =	_ = [ ASCII character ]	initial value: _ = _ default: no assignment

## Syntax and Descriptive Forms

### The APPEND Command

**Syntax Form:**

A [ I/O address ] [ edit line number ]

**Descriptive Form:**

APPEND [ I/O address ] [ destination for appended text ]

### The CARD Command

**Syntax Form:**

CA [ numeric constant ] [ , [ numeric constant ] ]

**Descriptive Form:**

CARD [ number of characters ] [ , [ fill character (decimal equivalent) ] ]

### The CASE Command

**Syntax Form:**

CAS [ edit line number ] [ , [ edit line number ] ]

**Descriptive Form:**

CASE [ starting line number ] [ , [ ending line number ] ]

### The COPY Command

**Syntax Form:**

C [ edit line number ] [ , [ edit line number ] [ , [ edit line number ] ] ]

**Descriptive Form:**

COPY [ starting line number ] [ , [ ending line number ] [ , [ destination for copied text ] ] ]

**The DELETE Command**

**Syntax Form:**

D edit line number [ , [ edit line number ] ]

**Descriptive Form:**

DELETE starting line number [ , [ ending line number ] ]

**The FIND Command**

**Syntax Form:**

F [ I/O address ] [ numeric constant ]

**Descriptive Form:**

FIND [ I/O address ] [ file number ]

**The INPUT Command**

**Syntax Form:**

INP [ I/O address ]

**Descriptive Form:**

INPUT [ I/O address ]

**The INSERT Command**

**Syntax Form:**

I [ edit line number ]

**Descriptive Form:**

INSERT [ destination for inserted lines of text ]

## Appendix B

### The LASTLINE Command

**Syntax Form:**

LA

**Descriptive Form:**

LASTLINE

### The LIST Command

**Syntax Form:**

L [ I/O address ] [ edit line number ] [ , [ edit line number ] ]

**Descriptive Form:**

LIST [ I/O address ] [ starting line number ] [ , [ ending line number ] ]

### The LOWERCASE Command

**Syntax Form:**

LO

**Descriptive Form:**

LOWERCASE

### The MARK Command

**Syntax Form:**

MA [ numeric constant ] [ , [ numeric constant ] ]

**Descriptive Form:**

MARK [ number of files ] [ , [ number of bytes per file ] ]



## The MOVE Command

**Syntax Form:**

```
M [ edit line number ] [ , [ edit line number ] [ , [ edit line number ] ] ]
```

**Descriptive Form:**

```
MOVE [ starting line number ] [ , [ ending line number ] [ , [ destination for moved text ] ] ]
```

## The NLSEARCH Command

**Syntax Forms:**

```
NL [ [ edit line number ] , [ edit line number ] , ] b " string " { * " string " }
```

**Descriptive Forms:**

```
NLSEARCH [ [ starting line number ] , [ ending line number ] , ] space " target string "
{ * " replacement string " }
```

## The NLSEARCH and Delete Line Command

**Syntax Form:**

```
NL [ [ edit line number ] , [ edit line number ] , ] b " string " *
```

**Descriptive Form:**

```
NLSEARCH [ [ starting line number ] , [ ending line number ] , ] space " target string " *
```

## The NLSEARCH and Replace String Command

**Syntax Form:**

```
NL [ [ edit line number ] , [ edit line number ] , ] b " string " , " string "
```

**Descriptive Form:**

```
NLSEARCH [ [ starting line number ] , [ ending line number ] , ] space " target string "
" replacement string "
```

## Appendix B

### The OLD Command

**Syntax Form:**

O [ I/O address ]

**Descriptive Form:**

OLD [ I/O address ]

### The PRINT Command

**Syntax Form:**

P [ I/O address ] [ string ]

**Descriptive Form:**

PRINT [ I/O address ] [ ASCII character string ]

### The RENUMBER Command

**Syntax Form:**

R [ edit line number ] [ , [ numeric constant ] [ , [ edit line number ] ] ]

**Descriptive Form:**

RENUMBER [ new starting line number ] [ , [ increment between new line numbers ] [ ,  
[ line in the current text where renumbering is to begin ] ] ]

### The REVSORT Command

**Syntax Form:**

REV [ edit line number ] , [ edit line number ] , numeric constant [ , numeric constant [ , . . . ] ]

**Descriptive Form:**

REVSORT [ starting line number ] , [ ending line number ] , character position [ , character position  
[ , . . . ] ]

**The SAVE Command**

**Syntax Form:**

SA [ I/O address ] [ edit line number ] [ , [ edit line number ] ]

**Descriptive Form:**

SAVE [ I/O address ] [ starting line number ] [ , [ ending line number ] ]

**The SEARCH Command**

**Syntax Forms:**

S [ I/O address ] [ [ edit line number ] , [ edit line number ] , ] b " string " [ { \* , [ " string " ] } ]

**Descriptive Forms:**

SEARCH [ I/O address ] [ [ starting line number ] , [ ending line number ] , ] space " target string " [ { \* , [ " replacement string " ] } ]

**The SEARCH and List Line Command**

**Syntax Form:**

S [ I/O address ] [ [ edit line number ] , [ edit line number ] , ] b " string "

**Descriptive Form:**

SEARCH [ I/O address ] [ [ starting line number ] , [ ending line number ] , ] space " target string "

**The SEARCH and Delete Line Command**

**Syntax Form:**

S [ I/O address ] [ [ edit line number ] , [ edit line number ] , ] b " string " \*

**Descriptive Form:**

SEARCH [ I/O address ] [ [ starting line number ] , [ ending line number ] , ] space " target string " \*

## Appendix B

### The SEARCH and Edit Line Command

**Syntax Form:**

S [ [ edit line number ] , [ edit line number ] , ] b " string " ,

**Descriptive Form:**

SEARCH [ [ starting line number ] , [ ending line number ] , ] space " target string " ,

### The SEARCH and Replace String Command

**Syntax Form:**

S [ I/O address ] [ [ edit line number ] , [ edit line number ] , ] b " string " , " string "

**Descriptive Form:**

SEARCH [ I/O address ] [ [ starting line number ] , [ ending line number ] , ] space " target string " , " replacement string "

### The SKIP Command

**Syntax Form:**

SK [ I/O address ] [ numeric constant ]

**Descriptive Form:**

SKIP [ I/O address ] [ number of logical records to advance the READ/WRITE heads ]

### The SORT Command

**Syntax Form:**

SO [ edit line number ] , [ edit line number ] , numeric constant [ , numeric constant { , . . . } ]

**Descriptive Form:**

SORT [ starting line number ] , [ ending line number ] , character position [ , character position { , . . . } ]

**The SWN (Save With Number) Command****Syntax Form:**

SWN [ I/O address ] [ edit line number ] [ , [ edit line number ] ]

**Descriptive Form:**

SWN [ I/O address ] [ starting line number ] [ , [ ending line number ] ]

**The UPPERCASE Command****Syntax Form:**

U

**Descriptive Form:**

UPPERCASE

**The WRITE Command****Syntax Form:**

W [ I/O address ] [ edit line number ] [ , [ edit line number ] ]

**Descriptive Form:**

WRITE [ I/O address ] [ starting line number ] [ , [ ending line number ] ]

**The ] = Command (A Character to mean "END-OF-RECORD")****Syntax Form:**

] = [ ASCII character ]

**The # = Command (A Character to mean "DIGIT")****Syntax Form:**

# = [ ASCII character ]

## Appendix B

The ~ = Command (A "WILDCARD" Character)

**Syntax Form:**

~ = [ ASCII character ]

The \_ = Command (An "ALL BUT" Prefix)

**Syntax Form:**

\_ = [ ASCII character ]

**TABLE OF DEFAULT PRIMARY AND SECONDARY ADDRESSES  
FOR EDITOR INPUT/OUTPUT COMMANDS**

<b>I/O Command</b>	<b>Default I/O Address</b>
<b>APPEND</b>	<b>@33,4:</b>
<b>FIND</b>	<b>@33,27:</b>
<b>INPUT</b>	<b>@33,13:</b>
<b>LIST</b>	<b>@32,19:</b>
<b>OLD</b>	<b>@33,4:</b>
<b>PRINT</b>	<b>@32,12:</b>
<b>SAVE</b>	<b>@33,12:</b>
<b>SEARCH</b>	<b>@32,19:</b>
<b>SKIP</b>	<b>@33,13:</b>
<b>SWN (Save with Number)</b>	<b>@33,12:</b>
<b>WRITE</b>	<b>@33,12:</b>

**32 = GS display**  
**33 = internal magnetic tape**

# Appendix C

## INDEX

### A

Abbreviations 2-8  
"All but" prefix 5-41  
Alphabetizing 4-88, 4-91, 4-93, 4-98  
Alphanumeric keys 1-7, 3-3  
Alternate end-of-file mark 6-36  
Alternate end-of-record character 1-11  
Alternate Input/Output format 1-7, 1-11, 6-36, 6-37  
"Any digit" character 5-27  
APPEND command 6-5  
ASCII code chart B-1  
Asterisk 4-41, 4-42, 4-44, 4-73, 4-74  
AUTO LOAD 1-8, 3-1  
AUTO NUMBER 1-8, 3-1

### B

BASIC programs 2-6  
BASIC program line numbers 5-19, 6-9, 6-52  
Blank spaces 2-11, 2-12, 4-30, 4-42, 4-43, 4-50, 4-62, 5-34, 6-3, 6-6, 6-31, 6-41, 6-44, 6-57  
BREAK key 1-9, 3-3, 3-4, 3-8, 3-9, 4-24, 4-25, 4-65, 4-88, 4-96  
BUSY condition 3-4

### C

CALL statement 1-1, 2-6  
CARD command 4-3, 4-99  
Card images 4-7  
CASE command 4-9, 5-10, 5-22  
Checksum 1-7, 6-22, 6-33, 6-34  
CLEAR function 3-8, 4-25  
Closing a tape file 6-13, 6-14, 6-15, 6-31, 6-40, 6-57  
Colon 2-7, 3-7, 3-8, 4-23, 4-27, 4-30, 4-33, 5-4, 6-2, 6-20, 6-42, 6-50  
Command keywords 2-8  
Command summary 2-2, B-2  
COMPRESS function 3-6  
Concatenating lines 1-9  
COMPRESS/EXPAND key 3-4, 4-24  
Control characters 5-28, 6-41, 6-57  
COPY command 4-13, 4-36  
CR/LF Input/Output delimiter 1-7, 6-35, 6-36  
CR (ASCII 13) 1-10, 1-11, 4-7, 5-4, 5-5, 5-28, 5-34, 5-48, 6-36, 6-37  
CTRL key 3-3, 5-28

### D

Default I/O addresses B-18  
Default values 2-27, 2-28, 2-33, B-7, B-18  
DELETE command 2-27, 4-19  
Deleting lines see DELETE command  
Deleting strings 4-51, 4-82  
Destination line 2-24, 2-28  
Digits, locating 5-27  
Display format 2-7

### E

Edit delimiter 2-10, 2-11, 2-12, 2-29, 4-42, 4-50, 4-61, 4-68, 4-75  
Edit line number 1-10, 2-15, 2-16, 2-17, 2-18, 5-3, 5-4, 5-5, 5-15, 6-49, 6-52  
Editing commands 4-1  
End-of-file mark 4-30, 4-33, 5-4, 6-14, 6-29, 6-30, 6-31, 6-36, 6-40, 6-44, 6-50, 6-52, 6-56, 6-58  
End-of-line marker 3-4, 3-5, 3-6  
End-of-record character 1-7, 1-9, 1-11, 4-30, 4-39, 4-56, 5-4, 5-5, 5-47, 5-48, 5-50, 5-54, 5-55, 6-29, 6-30, 6-36, 6-37, 6-40, 6-50, 6-56  
"END-OF-RECORD" character 1-11, 5-47  
Ending line 2-23, 2-27  
Environmental commands 5-1  
Error messages A-1  
EXPAND function 3-4, 3-5

### F

Fill character 4-3  
File header see header  
FIND command 6-11  
Formatted text 4-3

### H

Header, accessing the tape file header 6-12  
Header format 1-7, 6-12, 6-22, 6-33, 6-34  
HOME/PAGE key 3-2, 3-3

### I

INPUT command 6-17  
Input/Output addresses 6-2  
Input/Output commands 6-1  
Input/Output delimiters 6-35, 6-36  
INSERT command 2-8, 4-23  
Insert mode 3-3, 3-7, 3-8, 4-23, 4-27  
Installation 1-4



## Appendix C

### L

LAST file 6-12  
LASTLINE command 2-7, 5-3  
Line buffer 1-9, 3-4, 3-5, 3-6, 3-7, 4-66  
LINE EDITOR keys 1-7, 1-9, 2-9, 3-4, 4-24, 4-66  
Line of text 1-10  
LIST command 4-29  
Logical record 1-10, 6-17, 6-25, 6-35, 6-36, 6-43, 6-44  
LOWERCASE command 4-9, 4-12, 4-90, 4-98, 5-9  
Lowercase flag 4-9, 4-12, 4-90, 4-98, 5-9

### M

Magnetic tape buffer 6-13, 6-15, 6-31, 6-57, 6-58  
Magnetic tape movement 6-4  
Magnetic tape status 1-7, 6-22, 6-33  
MAKE COPY key 3-9  
MARGIN 1 3-2, 3-3  
MARGIN 2 3-2, 3-3  
MARGIN OFF 3-2, 3-3  
MARK command 6-21  
Marking files on an external device 6-23, 6-33  
Memory requirements 1-2  
Microprocess status parameters 6-35  
MOVE command 4-35

### N

NLSEARCH command 4-39, 5-10, 5-22, 5-27, 5-33, 5-41, 5-47  
NLSEARCH and Delete Line command 4-39, 4-40, 4-41, 5-49  
NLSEARCH and Replace String command 4-39, 4-40, 4-47, 5-29  
Non-existent line numbers 2-5, 2-20, 2-21, 2-22, 2-25, 2-26  
Non-header format 1-7, 6-13, 6-22, 6-33, 6-34

### O

Offset 2-10, 2-17, 2-18, 2-19, 5-16  
OLD command 6-25  
Opening a tape file 6-11, 6-13  
Overflowing the text buffer 5-5, 6-7, 6-27, 6-45, A-1  
Overlay keys 3-1

### P

PAGE FULL condition 3-2  
Peripheral control keys 3-9  
Physical record length 1-7, 6-21, 6-33, 6-34  
Positioning the tape see FIND command, INPUT command, SKIP command  
Power requirements 1-3  
Prefix character see "all but" prefix  
Primary address 6-2  
PRINT command 5-29  
Processor status 6-35

### Q

Queue buffer A-1

### R

RECALL LINE function 1-9, 3-8  
RECALL NEXT LINE function 1-9, 3-8, 3-9  
RECALL NEXT LINE/RECALL LINE key 1-9, 3-8, 4-67, 5-48, 5-50  
RECALL PREVIOUS LINE function (STEP PROGRAM key) 1-9, 3-8, 4-67, 5-50  
RENUMBER command 2-8, 5-15  
Replacement 4-48, 4-80  
Replacement string 4-40, 4-48, 4-56, 4-80, 5-29, 5-33, 5-42, 5-47  
REPRINT function 3-7  
REPRINT/CLEAR key 3-7, 4-24, 4-67  
RETURN key 1-9, 3-3, 3-8, 4-24, 4-25, 4-30, 4-66, 4-67, 5-28  
RETURN TO BASIC key 1-1, 2-6, 3-1, 3-3, 5-29, 5-34, 5-43, 5-46, 6-14, 6-34, 6-36, 6-37  
REVSORT command 4-95, 5-10, 5-22  
REWIND key 3-9  
Rewinding the magnetic tape 6-12  
RUBOUT CHARACTER 1-8, 3-2  
RUBOUT/BACKSPACE key 3-4, 4-24  
RUBOUT/SPACE key 3-4, 4-24

### S

SAVE command 6-39  
Save With Number see SWN (Save With Number)  
SEARCH command 4-55, 5-10, 5-22, 5-27, 5-33, 5-41, 5-47  
SEARCH and Delete Line command 4-56, 4-73, 5-49  
SEARCH and Edit Line command 4-56, 4-65, 5-51  
SEARCH and List Line command 4-56, 4-59  
SEARCH and Replace String command 1-9, 4-56, 4-79, 5-29  
Secondary address 6-2  
SKIP command 6-43  
SORT command 2-12, 2-13, 4-87, 5-10, 5-22  
SP (ASCII 32) 1-10, 4-30, 4-88, 4-96, 5-28, 5-34  
Spaces see blank spaces  
Special PRINT commands 6-33  
Specifications 1-2  
Starting line 2-23, 2-27, 2-30, 2-31  
Status parameters 6-33  
STEP PROGRAM key 1-9, 3-9, 4-67, 5-48  
String 1-10  
String delimiter 1-10, 2-13, 2-14, 2-15, 4-61, 4-69  
SWN (Save With Number) command 6-9, 6-49  
Syntax 2-35, 2-36, 2-37, B-9  
Syntax errors 2-9, A-1

### T

Target string 4-39, 4-40, 4-48, 4-55, 4-56, 4-80, 5-28, 5-33, 5-34, 5-41, 5-42, 5-47  
Text 1-8  
Text buffer 1-7, 2-6, 2-7, 3-1, 5-3, 5-4, 5-5

**U**

Unnumbered lines 2-5, 2-17, 2-18, 2-19, 2-23  
UPPERCASE command 4-11, 4-29, 4-90, 4-98, 5-21  
Uppercase flag 4-9, 4-11, 4-90, 4-98, 5-21  
User-definable keys 3-1

**W**

Wildcard character 5-33  
WRITE command 6-55

**Special symbols**

#= command 5-27  
\_= command 5-41  
~= command 5-33  
]= command 5-47

# MANUAL CHANGE INFORMATION

PRODUCT 4051 EDITOR  
070-2170-00

CHANGE REFERENCE C1/677  
DATE 6-6-77

CHANGE:

DESCRIPTION

The following change information applies to a 4051R06 EDITOR that has a serial number of B020290 or higher.

Since the time this manual was printed, extra features have been added to the EDITOR. A summary of the features is provided below. Because certain changes affect the operation of the commands, you should read the following pages carefully before turning to the rest of the manual.

## SUMMARY

- The syntax of the MARK command has been expanded to allow an I/O address to be specified. This means you may now use the MARK command instead of a PRINT command to mark files on an external device such as the TEKTRONIX 4924 Digital Cartridge Tape Drive. The new syntax and descriptive forms are as follows:

### Syntax Form:

MA [ I/O address ] [ numeric constant ] [ , [ numeric constant ] ]

### Descriptive Form:

MARK [ I/O address ] [ number of files ] [ , [ number of bytes per file ] ]

The default I/O address for the MARK command is @33,28: . That is, if no I/O address is specified, files are marked on the internal magnetic tape by default.

- The syntax of the CARD command has been expanded to allow an I/O address to be specified. When the CARD command is executed, lines found to be longer than the desired number of characters are listed on the specified peripheral device. The new syntax forms are as follows:

### Syntax Form:

CA [ I/O address ] [ numeric constant ] [ , [ numeric constant ] ]

### Descriptive Form:

CARD [ I/O address ] [ number of characters ] [ , [ fill character (decimal equivalent) ] ]

CHANGE:	DESCRIPTION
	<p>The default I/O address for the CARD command is @32,19: . That is, if no I/O address is specified, lines found to be too long are listed on the system display by default.</p> <ul style="list-style-type: none"> <li>• The "all but" prefix may immediately precede the "END-OF-RECORD" character in the target string of a SEARCH or NLSEARCH command. For example, if _ is currently the "all but" prefix and ] is the "END-OF-RECORD" character, the command S "A_]" lists lines containing an occurrence of A that is <u>not</u> immediately followed by an end-of-record character. (Refer to the explanation of the _= and ]= commands.)</li> <li>• The "all but" prefix may immediately precede the "any digit" character in the target string of a SEARCH or NLSEARCH command. For example, if _ is currently the "all but" prefix, and # is the "any digit" character, the command S "A_#" lists lines containing an occurrence of the character A that is <u>not</u> immediately followed by one of the digits 0 through 9.</li> <li>◦ The default I/O address for the SAVE command has been changed to @33,1: .</li> <li>◦ The default I/O address for the SWN (Save With Number) command has been changed to @33,1: .</li> <li>• After a SAVE or SWN command stores text on a TEKTRONIX 4924 tape unit file, the file is automatically closed and no longer available for access by Input/Output operations. That is, you need not execute a FIND command or press the RETURN TO BASIC overlay key to close the file.</li> <li>• The INPUT command is used to display one logical record on the screen, and advance the tape to the next logical record on the file. The operation of the INPUT command has not changed. However, the following note should be added about executing the INPUT command when the tape is positioned to a logical record that contains more than 408 characters:</li> </ul> <p>After the INPUT command displays a complete logical record on the screen, the cursor reappears at the beginning of the next line on the display. However, if the logical record contains more than 408 characters, the INPUT command displays only the first 408 characters, and the cursor reappears in the <u>same</u> line as the displayed portion. To view the rest of the record and advance the tape to the next record on the file, execute another INPUT command (while the cursor is in the same line, or after pressing RETURN). The portion of the logical record that begins with the 409th character is displayed on the screen. If the cursor again reappears in the same line as the displayed characters, continue to execute INPUT commands, until the entire record has been displayed and the cursor reappears at the beginning of the next line on the display.</p> <p style="text-align: center;"><b>NOTE</b></p> <p><i>You are not required to continue executing INPUT commands until the entire record is displayed and the tape advances to the next logical record. However, you should be aware that since the tape only advances in groups of 408 characters, you may be positioning the tape to the middle of the logical record. Subsequent I/O commands that are executed while the file is still open, only affect the portion of the record that lies beyond the tape head.</i></p> <ul style="list-style-type: none"> <li>◦ Executing the UPPERCASE command disables the lowercase flag, and sets an "uppercase flag." Once the uppercase flag is set, the EDITOR does not recognize the difference between lowercase and uppercase characters in the text. For example, the character a in the text is considered to be the same as the character A during searching and sorting operations.</li> </ul>

CHANGE:	DESCRIPTION
---------	-------------

This means that when the uppercase flag is set, the commands S "a" and S "A" are equivalent. The command S "a" searches the text for the character a . The character A also satisfies the search, because the EDITOR cannot see the difference between a and A in the text. Likewise, the command S "A" searches the text for the character A . The character a also satisfies the search, because the EDITOR cannot distinguish between a and A in the text.

At the time the manual was printed, setting the uppercase flag caused the EDITOR to see all text characters as uppercase. This meant that the commands S "a" and S "A" were not equivalent. The command S "A" searched for the character A . The character a also satisfied the search, because the EDITOR saw the lower-case character a to be an uppercase A . The command S "a" searched for the character a , but was never able to find one, because the EDITOR saw all text characters as uppercase.

Because the change significantly affects the operation of the command, a new discussion of the UPPERCASE command is provided at the end of this section. You may replace pages 5-21 through 5-26 of the manual with this new explanation.

### CORRECTIONS TO THE MANUAL

The following is a detailed list of corrections to the manual. These corrections make the manual compatible with your version of the EDITOR.

#### Changes Concerning the MARK Command

page 6-21 change the syntax and descriptive forms to

**Syntax Form:**

MA [I/O address] [ numeric constant ] [ , [ numeric constant ] ]

**Descriptive Form:**

MARK [I/O address] [ number of files ] [ , [ number of bytes per file ] ]

page B-12 same change as above

page 6-21 change the first sentence to

The MARK command reserves space for magnetic tape files on the specified device.

page 6-21 paragraph 2 under **EXPLANATION:** change the first sentence to

When the MARK command is executed, the specified number of files are created on the device, starting at the current position at the magnetic tape head.

CHANGE:	DESCRIPTION
---------	-------------

page 6-22 paragraph 1 under **Changing the Tape Format:** change the first part of the second sentence to  
 If the special PRINT command PRI@33,0:1,1,1 is executed, internal magnetic tape files are created without a file header;

page 6-22 paragraph 1 under **DEFAULT VALUES:** insert as the first sentence  
 When the I/O address is omitted in a MARK command, files are marked on the internal magnetic tape by default.

page 6-23 delete the second item under **Differences Between BASIC MARK and EDITOR MARK.**  
 Also change the title to **The Difference Between BASIC MARK and EDITOR MARK.**

page 6-23 the paragraph entitled **Marking Files on an External Device** is still valid, but no longer necessary.

page 2-33 the entry for the MARK command: change the Syntax (Descriptive Form) to include an optional I/O address. Also insert under Default Values for the MARK command

I/O address = @33,28:

page B-7 same change as above

page 6-3 in the **TABLE OF DEFAULT PRIMARY AND SECONDARY ADDRESSES** after the entry for LIST, insert the entry

MARK @33,28:

page B-19 same change as above

page 6-33 delete paragraph 2 under **Marking Files on an External Device**

**Changes Concerning the CARD command**

page 4-3 change the syntax and descriptive forms to

**Syntax Form:**

CA [I/O address] [ numeric constant ] [ , [ numeric constant ] ]

**Descriptive Form:**

CARD [I/O address] [ number of characters ] [ , [ fill character (decimal equivalent) ] ]

CHANGE:	DESCRIPTION
page B-10	same change
page 4-3	paragraph 1 under <b>EXPLANATION</b> : change the second, third, and fourth sentences to read  Two parameters may be entered after the keyword CARD and the I/O address. The first specifies the number of characters each line is to contain. The second is the decimal code number for an ASCII character.
page 4-3	paragraph 2 under <b>EXPLANATION</b> : change the command CA 50,46 to  CA @33:50,46
page 4-4	line 4: change the word "display" to  specified device
page 4-4	line 5: change the word "appears" to  is made
page 4-4	line 6: change the words "printed on the display" to  issued
page 4-4	paragraph 1 under <b>Default Values</b> : replace paragraph 1 with  All three parameters for the CARD command are optional, and may be omitted or entered in any combination. Several examples are listed on the preceding page.
page 4-4	under <b>Default Values</b> : insert a third paragraph consisting of the sentence  When the I/O address is omitted, lines that are too long are listed on the system display by default.
page 2-33	the entry for the CARD command: change the Syntax (Descriptive Form) to include an optional I/O address. Also insert under <b>Default Values</b> for the CARD command  I/O address = @32,19:
page B-7	same change as above
page 6-3	in the <b>TABLE OF DEFAULT PRIMARY AND SECONDARY ADDRESSES</b> : after the entry for APPEND, insert the entry  CARD @32,19:
page B-19	same change as above

CHANGE:	DESCRIPTION
<p><b>Changes Concerning the ]= Command</b></p>	
<p>page 5-49</p>	<p>delete the indented paragraph called NOTE.</p>
<p><b>Changes Concerning the # = Command</b></p>	
<p>page 5-28</p>	<p>delete the indented paragraph called NOTE that appears at the bottom of the page.</p>
<p><b>Changes Concerning the _ = Command</b></p>	
<p>page 5-28</p>	<p>replace the indented NOTE with the following paragraph:</p>
<p style="text-align: center;"><i>NOTE</i></p> <p style="text-align: center;"><i>The "all but" prefix should not immediately precede the wildcard character in a target string. For example, if _ is the "all but" prefix and ~ is the wildcard character, the EDITOR cannot interpret the command S "_~".</i></p>	
<p><b>Changes Concerning the SAVE Command</b></p>	
<p>page 2-34</p>	<p>the entry for the SAVE command: change the Default Value for the I/O address to</p> <p style="text-align: center;">I/O address = @33,1:</p>
<p>page B-8</p>	<p>same change as above</p>
<p>page 6-3</p>	<p>in the <b>TABLE OF DEFAULT PRIMARY AND SECONDARY ADDRESSES:</b> in the entry for SAVE, change the Default I/O Address to</p> <p style="text-align: center;">@33,1:</p>
<p>page B-19</p>	<p>same change as above</p>
<p><b>Changes Concerning the SWN Command</b></p>	
<p>page 2-34</p>	<p>the entry for the SWN (Save With Number) command: change the Default Value for the I/O address to</p> <p style="text-align: center;">I/O address = @33,1:</p>
<p>page B-8</p>	<p>same change as above</p>
<p>page 6-3</p>	<p>in the <b>TABLE OF DEFAULT PRIMARY AND SECONDARY ADDRESSES:</b> in the entry for SWN (Save With Number), change the Default I/O Address to</p> <p style="text-align: center;">@33,1:</p>
<p>page B-19</p>	<p>same change as above</p>



CHANGE:	DESCRIPTION
	<p data-bbox="235 199 665 231"><b>Changes Concerning the Closing of Files</b></p> <p data-bbox="300 262 649 294">page 6-40      delete footnote 9</p>

CHANGE:	DESCRIPTION
	<p><b>Changes Concerning the INPUT Command</b></p> <p>page 6-19     After the last paragraph on the page, insert the following:</p> <p><b>Logical Records Containing More Than 408 Characters</b></p> <p>After the INPUT command displays a complete logical record on the screen, the cursor reappears at the beginning of the next line on the display. However, if the logical record contains more than 408 characters, the INPUT command displays only the first 408 characters, and the cursor reappears in the <u>same</u> line as the displayed portion. To view the rest of the record and advance the tape to the next record on the file, execute another INPUT command (while the cursor is in the same line, or after pressing RETURN). The portion of the logical record that begins with the 409th character is displayed on the screen. If the cursor again reappears in the same line as the displayed characters, continue to execute INPUT commands, until the entire record has been displayed and the cursor reappears at the beginning of the next line on the display.</p> <p style="text-align: center;"><i>NOTE</i></p> <p><i>You are not required to continue executing INPUT commands until the entire record is displayed and the tape advances to the next logical record. However, you should be aware that since the tape only advances in groups of 408 characters, you may be positioning the tape to the middle of the logical record. Subsequent I/O commands that are executed while the file is still open, only affect the portion of the record that lies beyond the tape head.</i></p>

CHANGE:	DESCRIPTION
	<p><b>Changes Concerning the UPPERCASE Command</b></p> <p>page 2-3      Action Taken by the UPPERCASE command example U: change the first phrase to</p> <p>                 Causes the EDITOR to perceive lowercase text characters a-z to be the same as their uppercase equivalents A-Z.</p> <p>page B-4      same change as above</p> <p>page 4-90      paragraph 6: change the second sentence to</p> <p>                 While the uppercase flag is set, the EDITOR does not distinguish between lowercase and uppercase text characters during the sorting process.</p> <p>page 4-98      paragraph 4: change the second sentence to</p> <p>                 While the uppercase flag is set, the EDITOR does not distinguish between lowercase and uppercase text characters during the sorting process.</p> <p>You may replace the discussion of the UPPERCASE command that appears on pages 5-21 through 5-26 with the explanation that begins on the following page.</p>

CHANGE:	DESCRIPTION
---------	-------------

**The UPPERCASE Command**

<p><b>Syntax Form:</b></p> <p>U</p> <p><b>Descriptive Form:</b></p> <p>UPPERCASE</p>
--

**PURPOSE**

The UPPERCASE command causes the EDITOR to perceive lowercase text characters a-z to be the same as their uppercase equivalents A-Z during searching and sorting operations (NLSEARCH, SEARCH, REVSORT, and SORT commands). The UPPERCASE command also prepares the EDITOR to change lowercase text characters into uppercase characters if the CASE command is executed.

**EXAMPLE**

U

**EXPLANATION**

The UPPERCASE command has no parameters. Only the keyword U is entered from the keyboard, as shown in the example above. Executing the UPPERCASE command has no immediate effect on the text buffer. Instead, a system environment parameter is assigned a value that prepares the EDITOR for subsequent commands.

Changing the value of the environmental parameter by executing the UPPERCASE command disables the lowercase flag, and sets an "uppercase flag". Once the uppercase flag is set, the EDITOR does not recognize the difference between uppercase and lowercase characters in the text. For example, when the uppercase flag is set the EDITOR considers a lowercase b found in the text to be the same as an uppercase B. This affects the operation of the NLSEARCH, SEARCH, REVSORT, SORT, and CASE commands.

CHANGE:	DESCRIPTION
	<p><b>The NLSEARCH and SEARCH Commands</b></p> <p>When the uppercase flag is set, a lowercase character in the text buffer "matches" or satisfies a search for the equivalent uppercase character specified in a target string. Conversely, an uppercase character in the text "matches" the equivalent lowercase character in a target string.</p> <p>For example, when the uppercase flag is set the command NL "A", "" * deletes all occurrences of the character A (ASCII equivalent 65) from the text buffer. Occurrences of the character a are also deleted from the text buffer, because the EDITOR does not recognize the difference between a and A . Likewise, when the uppercase flag is set the command NL "a" * deletes all occurrences of the characters A and a .</p> <p><b>The REVSORT and SORT Commands</b></p> <p>When the uppercase flag is set, the EDITOR does not distinguish between lowercase and uppercase text characters during sorting operations. For example, if a REVSORT or SORT command is executed while the uppercase flag is set, a line of text containing the character b in a specified character position is treated the same as a line containing the character B in the same position. This is because when the uppercase flag is set, the characters b and B are treated as having the same ASCII code value, 66.</p> <p>For an illustration of how the REVSORT and SORT commands operate after the UPPER-CASE command is executed, refer to "An Editing Example" on the following pages.</p> <p><b>The CASE Command</b></p> <p>While the uppercase flag is set, executing a CASE command causes lowercase text characters a-z to be replaced by their uppercase equivalents A-Z. This is the inverse of the function performed by the CASE command if the lowercase flag is set.</p> <p><b>Default Value</b></p> <p>Calling the EDITOR automatically sets the lowercase flag. To set the uppercase flag, you must execute the UPPERCASE command.</p> <p><b>An Editing Example</b></p> <p>The following examples show how the SEARCH, CASE, SORT, and REVSORT commands operate while the uppercase flag is set. The examples are analagous to those used to illustrate the effect of setting the lowercase flag. To compare the results shown below with the results when the lowercase flag is set, refer to "An Editing Example" in the explanation of the LOWERCASE command.</p>

CHANGE:	DESCRIPTION
---------	-------------

Example 1

LIST

- 1:Abernathy, Tod
- 2:Brockway, Marius E.
- 3:Ellis, Terry L.
- 4:Foster, Alice
- 5:Hillstrom, A.
- 6:Keller, Suzanne
- 7:Lentz, John F.
- 8:Pollock, Robert
- 9:Siebold, William B.
- 10:Taylor, Owen

U

S "a"

- 1:Abernathy, Tod
- 1:Abernathy, Tod
- 2:Brockway, Marius E.
- 2:Brockway, Marius E.
- 4:Foster, Alice
- 5:Hillstrom, A.
- 6:Keller, Suzanne
- 9:Siebold, William B.
- 10:Taylor, Owen

S "A"

- 1:Abernathy, Tod
- 1:Abernathy, Tod
- 2:Brockway, Marius E.
- 2:Brockway, Marius E.
- 4:Foster, Alice
- 5:Hillstrom, A.
- 6:Keller, Suzanne
- 9:Siebold, William B.
- 10:Taylor, Owen

CASE

LIST

- 1:ABERNATHY, TOD
- 2:BRCKWAY, MARIUS E.
- 3:ELLIS, TERRY L.
- 4:FOSTER, ALICE
- 5:HILLSTROM, A.
- 6:KELLER, SUZANNE
- 7:LENTZ, JOHN F.
- 8:POLLOCK, ROBERT
- 9:SIEBOLD, WILLIAM B.
- 10:TAYLOR, OWEN

CHANGE:	DESCRIPTION
<p><b>Example 2</b></p>	<pre>LIST :a :b :c :d :e :f :g :h :i :j :k :l :m :n :o :p :q :r :s :t :u</pre> <p>U</p> <p>SO,,1</p> <pre>LIST :a :b :c :d :e :f :g :h :i :j :k :l :m :n :o :p :q :r :s :t :u</pre>
<p><b>Example 3</b></p>	<pre>LIST :a :b :c :d :e :f :g :h :i :j :k :l :m :n :o :p :q :r :s :t :u</pre> <p>U</p> <p>REU,,1</p> <p>(continued on next page)</p>

CHANGE:	DESCRIPTION
---------	-------------

```

LIST
:f
:F
:e
:E
:d
:D
:c
:C
:b
:B
:a
:A
    
```

Example 1 illustrates how the SEARCH and CASE commands operate while the uppercase flag is set. An initial listing shows that the text buffer contains a list of ten names. The command U is executed to set the uppercase flag, then the command S "a" tells the EDITOR to search the text and list lines found to contain the character a .

Because the uppercase flag is set, the EDITOR does not recognize the difference between lowercase and uppercase text characters. Both the lowercase character a and the uppercase equivalent A "match" the target string and satisfy the search. After the command is executed, the EDITOR lists seven lines found to contain the characters A or a . Lines 1 and 2 are printed twice, because they contain two occurrences of the target string.

Next the command S "A" tells the EDITOR to search the text and list lines found to contain the character A . Again the EDITOR does not distinguish between lowercase and uppercase characters in the text. As for the preceding command, both the lowercase character a and the uppercase character A "match" the target string and satisfy the search. After the command is executed, lines found to contain the characters a or A are listed on the display. The list is the same as for the preceding command.

Next the command CASE is executed. Because the uppercase flag is set, the EDITOR replaces all lowercase characters in the text with their uppercase equivalents. A new listing shows that the text buffer now contains only uppercase characters.

Example 2 illustrates how the SORT command operates while the uppercase flag is set. As in the previous example, the command U is executed to set the uppercase flag. Then a listing shows uppercase characters A-F and their lowercase equivalents a-f in lines consisting of one character each. The command SO,,1 tells the EDITOR to rearrange the lines according to the ASCII value of the character found in the first position in each line. Lines are to be rearranged so that the ASCII code values are in increasing order.



CHANGE:	DESCRIPTION
	<p>Because the uppercase flag is set, the EDITOR perceives lowercase characters in the text to be the same as uppercase characters. After the SORT command is executed, a new listing shows that lines consisting of lowercase characters are next to those consisting of their uppercase equivalents. This is because lowercase characters are treated as having the same ASCII code value as their uppercase equivalents.</p> <p>Example 3 is similar to Example 2, but executes the command REV,,1 . This time the lines are rearranged in decreasing ASCII code value. Again, no distinction is made between lowercase and uppercase characters: a new listing shows that each line consisting of a lowercase character is next to the line consisting of its uppercase equivalent.</p>

CHANGE:	DESCRIPTION
	<p><b>Changes Concerning the LOWERCASE Command</b></p> <p>page 5-9 paragraph 1 under <b>PURPOSE:</b> at the end of line 1 insert the word text</p> <p>paragraph 1 under <b>PURPOSE:</b> in line 4 replace "uppercase characters" with uppercase text characters</p> <p>paragraph 2 under <b>EXPLANATION:</b> in line 4 replace "the character b" with the character b in the text</p> <p>paragraph 2 under <b>EXPLANATION:</b> in line 5 correct the misspelling NLSERACH to NLSEARCH</p>
	<p>page 5-10 paragraph 1 under <b>The REVSORT and SORT Commands:</b> in line 2 replace "uppercase characters" with uppercase text characters</p> <p>paragraph 1 under <b>the CASE Command:</b> in line 1 replace "uppercase" with uppercase text</p>