Marinchip 9900 Text Editor

User Guide

by John Walker

Revised January 1980

Marinchip Systems 16 St. Jude Road Mill Valley, CA 94941 ■ (415) 383-1545

Marinchip 9900 Text Editor User Guide

Table of contents

Table of contents

Marinchip 9900 Text Editor User Guide

1. Introduction

The Marinchip 9900 Text Editor (EDIT) is a flexible context oriented text editor for the Marinchip 9900 computer. It runs under the control of a Marinchip Operating System, and edits files as defined by the operating system.

EDIT is a line oriented context editor. This places it in the middle between totally character oriented editors such as TECO and line number editors such as those commonly provided by BASIC systems. This type of editor was first developed by MIT's Project MAC for their timesharing systems. It was later extended by Univac for their 1100 series text editor, and further extended and refined by the University of Maryland in their comprehensive text editor. The idea for the Alter command came from the SOS editor for the PDP-10 originally developed at Stanford. Marinchip EDIT has selected features from all of these implementations to provide an editor which is convenient, efficient, and human engineered to eliminate many of the frustrations found in other text editors.

The most unique feature of Marinchip EDIT is its automatic paging of the file being edited to backing storage provided by files. This paging means that the size of the file being edited is in no way limited by the amount of memory available on the computer running EDIT. The paging is done without any user intervention being required, so that EDIT forces on the user none of the burden of segmenting a long file into short portions.

2. Calling the editor

The text editor is invoked from operating system command mode by the command:

EDIT ‹output file›=‹input file›

This command will cause EDIT to read the named ‹input file› into the editing workspace, and upon completion of the editing session, store the updated file in the named ‹output file›. Either or both of the file specifications may be omitted. If EDIT is called:

EDIT ‹output file›=

EDIT will write its output to the named file, but will read in no file at the start. This command is used when creating a new file from text entered on the console. If EDIT is called:

EDIT =‹input file›

EDIT will read in the named file, but will not write the file back at the end of the editing session. This form is used when using EDIT to inspect a file (for example, a listing file created by a compiler) where no intention of updating the file exists.

EDIT is most frequently called:

EDIT ‹file›

This is how a file is updated in place. EDIT will read in the ‹file›, then write the output back in the ‹file› upon completion of editing. Finally, the simple command:

EDIT

Will read or write no files automatically. The user is given complete control of the disposition of files through the use of the editor's file commands.

3. The editing process

Editing text, regardless of whether it is normal prose text or the more poetic forms of expression found in computer programs, normally consists of two major phases:

1. Entering the information (INPUT)
2. Correcting errors, modifying text, making updates. (EDIT)

Since the functions performed in these two modes are quite different, EDIT operates in two distinct modes: Input mode and Edit mode.

3.1. Input mode

Input mode allows the user of the text editor to enter the text to be subsequently edited. In input mode, lines of information typed by the user are simply transcribed to the file being edited. Limited processing of the text being entered occurs. For example, the user may define tab stops which allow the text being entered to be placed in columns. These tabs are expanded as the text is entered.

3.2. Edit mode

Edit mode allows modification of text. When in Edit mode, the user types commands which cause the text editor to perform such functions as locating words in the text, changing the contents of lines, replacing entire lines, deleting lines, or writing the edited text to an output file.

3.3. Switching modes

When invoked initially, EDIT starts in Edit mode. The user may switch modes at any time by entering a null line (simply pressing RETURN). Whenever the editor changes modes, a message will be typed to indicate which mode the editor has assumed, 'Edit' or 'Input'. In addition, when in Edit mode the user will be prompted for each command by an asterisk (*). In Input mode, no such prompt will be given. Hence the user is continually reminded of the current editor mode. Note that the editor distinguishes between a blank line (one containing one or more spaces) and a null line (no characters before the 'RETURN'). A blank line may be entered when in Input mode without causing a switch to Edit mode: only a null line will switch the mode.

4. Edit mode

Edit mode is the mode which allows commands to be entered which perform various functions upon the text in the file being edited. When in Edit mode, the user will be prompted for each command with an asterisk (*). At the completion of each command, the asterisk will be typed to request the next command. Input mode may be entered at any time by simply typing RETURN when the command prompt is issued.

4.1. The current line pointer

Crucial to effective use of the editor is understanding the concept of the current line pointer. At any time, the editor is positioned at some place in the file. This position may be at the top of the file (before the first line), the bottom of the file (after the last line), at a text line in the file, or at an 'empty window' created when a line is deleted. Most commands in Edit mode operate on either the current line, or on a group of lines starting at the current line. If at any time you are unsure where

you are in the file, type "P". This will type out the contents of the current line. If at the top, "*Top*" will be printed; if at the bottom, "*Eof*" (end of file) will be printed. If positioned at an empty window, nothing will be printed.

4.1.1. Initial position

When the editor gives its first command prompt (*), it is positioned at the top of the file.

4.1.2. Moving the current line pointer

When entering text into the editor, the current line pointer will be always be set to the last line entered. Thus, at any time the user can enter Edit mode and work on the last line typed in. Many Edit mode commands move the current line pointer, and the command descriptions below should be examined for the particulars of their actions. Those commands that explicitly move the current line pointer are described below.

4.1.2.1. Go to top of file

The Top command permits the pointer to be set at the top of the file. This allows a search to be made of the entire file, or text to be added before the first line.

4.1.2.2. Relative moving of the pointer

The "+" and "-" commands allow the pointer to be moved relative to the current line. This allows convenient reference to lines in a given region of a file.

4.1.2.3. Absolute positioning

The current line pointer may be set to any line in the file simply by entering its line number instead of a command while in Edit mode.

4.1.2.4. Automatic Top of file

If a command is entered that normally causes motion forward through the file (such as a Print, Find, or Locate) when at the end of file, the editor will automatically go the top of the file and indicate that it did this by typing "*Top*". This is a convenience feature for the user which allows a search to be continued easily when it encounters the end of the file.

4.2. Line numbers

EDIT is not a line number oriented editor such as that provided by BASIC. Nevertheless, there are times when line numbers are useful. EDIT maintains line numbers for all lines in the file being edited. These line numbers change as lines are added and deleted so that all lines currently in the file are numbered from one to the number of lines in the file. Any Edit mode command may be prefixed by a sharp sign (#), which will cause line numbers to be printed before any lines printed by that command. The user can go to any line in the file by number simply by entering its line number instead of a command in Edit mode. An example of use of line numbers might be to search the file for all occurrences of a specific word, listing the lines in which it was found. Those occurrences which the user wished to change could then be reached simply by entering the line numbers typed by the initial search command.

4.3. Edit mode commands

The following paragraphs describe the Edit mode commands. These commands are entered in response to the asterisk (*) prompt in Edit mode. Any number of leading spaces may precede the command mnemonic, and one or more spaces must separate the command mnemonic and any parameters it takes. If the command is prefixed with a sharp sign (#), line numbers will be typed on any lines printed by the command.

4.3.1. A - Alter

A

The Alter command allows interactive editing of the current line. When the "A" command is entered, the editor enters Alter mode and

accepts editing functions entered from the keyboard. The editing functions are single keystrokes or groups of keys that perform specific operations on the line being edited. The commands entered are executed immediately upon being entered, and are not echoed to the display. What is printed on the display depends upon the command entered. The commands normally operate at the "current character pointer", which moves as commands are executed. Initially, the current character pointer is positioned before the first character in the line. The available Alter command characters are:

Space — Advance the column pointer one character. The character spaced over is printed.

D — Delete the next character. The character deleted is printed in corner brackets.

I — Enter insert characters mode. All succeeding characters will be inserted at the current character position. An Escape character will terminate insert mode and return the editor to accepting Alter commands.

K — Kill. The next character is accepted, then all characters between the current column position and the next occurrence of that character are deleted. The characters deleted are printed between corner brackets. If the character entered is not found in the line, no deletion will occur, and the terminal bell will sound to indicate the error.

L — A copy of the line being altered is printed as it currently stands, then the column pointer is returned to before the first character. This command is very useful to see the effects of complex alterations on the line.

O — The editor enters overlay mode. All following characters will overlay successive characters of the original line. Overlay mode is terminated by an Escape character. Note the difference between Insert and Overlay mode: in Insert mode, the characters entered are inserted and the line length is increased to accomodate the added characters. In Overlay mode, the characters replace existing characters in the line and the line length remains unchanged.

P, ^R — The rest of the line is printed, then the line is reprinted up to the current column position. This is a way to see the results of alteration without moving the column pointer. The Control R key is accepted as well as

"P" for compatibility with the normal terminal support of the operating system.

Q — Quit. The results of all alterations made by the command are discarded and the line is left unchanged.

S — Search. The editor waits for the next character to be typed, then searches for the next occurrence of that character in the line. All characters are copied up to that occurrence and the column pointer is left before the occurrence of the character. If the character is not found, the line will be copied to the end.

V — Convert to lower case. This command is identical to the "S" (Search) command, except that all upper case letters encountered while searching for the character are translated to lower case.

^ — Convert to upper case. This command is identical to the "S" (Search) command, except that all lower case letters encountered while searching for the character are translated to upper case.

CR — The Carriage Return key causes the rest of the line to be copied, and the line as altered to replace the original line in the file. This is the normal way to terminate an Alter command.

LF — The Line Feed key truncates the line at the current column position, replaces the original line with the line as altered, and ends the Alter command. This is used to terminate an Alter command when the rest of the line is to be deleted.

BS, DEL — The Backspace or DEL (Rubout) keys cause the column pointer to be backed up one position. The character backed up over is printed between back slashes (\).

The Alter command is a very effective way of editing information, which allows more complicated editing with fewer keystrokes than are required for the Change command (see below). Effective use of the Alter command comes only with experience with it, so the reader is urged to experiment with the command guided by this manual.

4.3.2. AB - Abort

AB

The Abort command immediately terminates the editor and returns the user to operating system command level. If the editor was called with an output file on the EDIT command, the file will not be written back. This command may be used when a bad command has so messed up a file that it is easier to start over than to undo the catastrophe. If the editor was called without an output file, the Abort command acts the same as the End command.

4.3.3. C - Change

C,⟨lines⟩ /⟨old string⟩/⟨new string⟩/⟨per line⟩

The Change command allows the selective modification of strings of characters occurring within text lines. In its simplest form, Change modifies the current line. If the current line contains the text:

Now is the tome for all good men to give up.

and the command:

C /tome/time/

is entered, the first string, "tome", will be located in the line and replaced with the second string, "time". The line will then be typed out as modified:

Now is the time for all good men to give up.

The ⟨per line⟩ specification controls the action of the command when more than one occurrence of the first string occurs on one line. If this specification is omitted, only the first occurrence will be changed. If a number is specified for ⟨per line⟩, that many occurrences will be changed, scanning from the left. As with all other numbers accepted by the editor, an asterisk stands for an infinite number, so an asterisk for ⟨per line⟩ will mean to change all occurrences on the line. For example, suppose the current line contained the text:

That that is is simply that that is.

the command:

```
C /is/was/
```

would result in:

```
That that was is simply that that is.
```

while the command:

```
C /is/was/*
```

would result in:

```
That that was was simply that that was.
```

If the command:

```
C /is/was/2
```

had been entered, the result would have been:

```
That that was was simply that that is.
```

Now suppose that you wanted to change the last "is" in the line without affecting either of the first two. You could not use the ⟨per line⟩ specification for this, because it only specifies the number to change starting at the left. What you would have to do is to decide what makes the last one different, and include that in the ⟨old string⟩ specification. Inspection of the line immediately reveals that only the last "is" is followed by a period. Hence, the command:

```
C /is./was./
```

would result in:

```
That that is is simply that that was.
```

So far in our discussion of the Change command, we have always used a slash to delimit the ⟨old string⟩ and ⟨new string⟩ specifications. In fact, any nonblank character that does not appear in either ⟨old string⟩ or ⟨new string⟩ may be used. For example, if the current line contained:

```
Everybody knows that 10/2 is 12.
```

the command:

```
C :/:+:
```

would correct the line to read:

Everybody knows that 10+2 is 12.

Suppose now that you've just typed in an entire book, and discovered that you've consistently misspelled the word "receive" as "recieve" throughout the manuscript. Certainly you could go through the text looking for the errors and change them one by one with the Change command, but this would be very laborious and error-prone. Fortunately, the Change command was implemented by people who make these kinds of large-scale boo-boos, so the ⟨lines⟩ specification may be brought to the rescue. If specified, Change will start with the current line and search for lines which contain the ⟨old string⟩. For each such line found, the Change command will be applied. This action will continue until the number of lines specified by ⟨lines⟩ have been changed. Note that this is a count of lines CHANGED, not lines EXAMINED. If ⟨lines⟩ is an asterisk, denoting an infinite number, all lines from the current line to the end of file will be examined. Note that the command:

```
C,1 /bad/good/
```

is not the same as:

```
C /bad/good/
```

since the "C,1" would search through the file looking for a line containing "bad" to change, while if the current line did not contain "bad" the second command would type the error message:

```
No match.
```

and perform no change. Getting back to our poor author's book, the command:

```
C,* /recieve/receive/*
```

will make things much more presentable. (Note that we had to specify the asterisk for the ⟨per line⟩ specification. If it were omitted, only the first "recieve" on a line would have been fixed, so a line with more than one of the errors would have been left with an uncorrected word.)

The Change command is frequently used along with the other editing commands to perform repetitive changes beyond the scope of the ⟨lines⟩ specification. To ease such use, Change "remembers" the ⟨old string⟩ and ⟨new string⟩ used on the last application of the command. If a subsequent Change command is entered with no specifications other than possibly ⟨lines⟩, these saved strings will be used. Thus, if the command:

```
C /widget/sophisticated technological achievement/
```

had been entered previously, the command:

```
C
```

would apply the same change to a subsequent line.

The line-scanning operation performed by the Change command when searching for the specified target string may be modified through use of the Column limits (CL) command. Normally, the entire line is subject to scrutiny, but the CL command may be used to set left and right column limits for such scrutiny. See the description of the Column limits command (below) for further information.

4.3.4. CL - Column limits

```
CL <num1>[,<num2>]
```

The Column limits (CL) command may be used to set left and right column limits for the Change, Find, and Locate commands. These commands will operate only on the portion of the line defined by these column limits. If two numbers are specified, the first is taken as the left column limit, and the second as the right column limit. If just one number is specified, it is taken as the right column limit, and the left column limit is assumed to be one (1).

When the editor is invoked, the CL settings are 1,132, and they remain so set until changed by the user. The current CL column settings are displayed by the Status (ST) command.

Details of the effects of the CL column settings on the Change, Find, and Locate commands are discussed below.

4.3.4.1. Effect on the Change command

The CL column limits affect the Change command in several ways:

A. Only that portion of a line which lies between the CL column limits (inclusive) will be examined and compared to the target string. For example:

```
I Testing, Testing, Testing.
CL 10,30
C /Test/Work/*
```

would result in:

```
Testing, Working, Working.
```

since only the last two occurrences of the target string fell within the CL column limits.

B. Only that portion of a line which lies between the CL column limits will be modified if the target string is found. Note in the preceding example that the portion of the line which was outside the CL limits was unchanged.

C. If the replacement string is longer than the target string, no column following the right column limit will be altered. Any lengthened image will be truncated without warning at the right column limit to prevent changing the columns after it. For example:

```
I ASMTAG    LI        R0,TAG2              get address
CL 40
C /TAG2/SOMETAG/
```

would result in:

```
ASMTAG    LI        R0,SOMETAG           get address
```

Since the replacement string is three characters longer than the target string, this would normally have resulted in the comment being shifted three characters to the right. However, since the comment begins in column 41 and the right column limit set by the CL command was 40, the position of the comment was not affected by the change. Furthermore, if there had been an occurrence of the string "TAG2" within the comment, it would not have been changed.

D. If the replacement string is shorter than the target string, and the original line extends beyond the right CL column limit, the portion of the line within the limits will be blank-filled up to the right column limit. For example:

```
I POINT2    LI        R0,TABLE+LENGTH+150 point to end
CL 40
C /+LENGTH//
```

would result in:

```
POINT2    LI        R0,TABLE+150         point to end
```

Notice again that the position of the comment was not affected, and that the image was blank-filled up to column

40.

4.3.4.2. Effect on the Locate command

The Locate command will search only those characters which lie between the CL column limits (inclusive). For example:

```
CL 21,40
L,* TABLE
```

might be used to locate all occurrences of the string "TABLE" used as an operand in an assembly language program (anyhwere between columns 21 and 40).

4.3.4.3. Effect on the Find command

The CL column limits' effect on the Find command is similar to their effect on the Locate command. The Find command, however, is column-sensitive by design, and simply uses the left column limit as its "first" column for searching purposes. For example:

```
CL 11,20
F,* MOV
```

might be used to find all the MOV instructions in an assembly language program, without unnecessarily matching many possible occurrences of the word "MOVE" which may have started in columns other than 11.

4.3.5. D - Delete

```
D ‹number›
```

The Delete command deletes text starting with the current line and continuing until ‹number› lines have been deleted. If ‹number› is omitted, only the current line will be deleted. At the end of the deletion, the current line pointer will be positioned at an "empty window" between the line preceding the lines deleted and the first line following them. If text is inserted at this point, the empty window situation will go away and the current line pointer will point to the last line inserted as usual. If a Change command, or any other command that modifies the text of the current line is entered when positioned at an empty window, the error message:

```
No current line.
```

will appear, and no action will occur. The empty window mechanism was implemented to remove the ambiguity of position after a deletion common to most editors, and to detect the common error of "change after delete". If the user wishes to advance through a file, deleting as he goes, the "DN" command (see below) should be used.

4.3.6. DH - Delete and Hold

```
DH [<image number>]
```

The Delete and Hold (DH) command copies the current image to an internal editor save area, and deletes it from the file. The held image may be retrieved later using the Duplicate command. There are 9 internal save areas which may be used to hold different lines. The <image number> specification indicates in which save area the line is to be saved. If no specification is given, area 1 will be used.

4.3.7. DN - Delete and Next

```
DN <number>
```

The Delete and Next (DN) command acts the same as the Delete command described above in that it deletes <number> lines starting at the current line and that it deletes only the current line if <number> is omitted. After performing the deletion, however, Delete and Next advances to the next line in the file, makes it the current line, and prints it. This makes DN suitable for deleting a block of text where the user is unsure how many lines are to be deleted. The user can simply type "DN", deleting a line each time, until the first line not to be deleted appears.

4.3.8. DU - Duplicate

```
DU [<number>][,<image number>]
```

The Duplicate (DU) command retrieves the image held by the last Hold (or Delete and Hold) command, and inserts it at the current position in the file. If a <number> is specified, the image will be inserted <number> times. If <image number> is specified, the image from that specific save area will be the image duplicated

(save area 1 will be assumed if no ‹image number› is specified).

If the Duplicate command is used before an image has been held via a Hold (HO) or Delete and Hold (DH) command, the message:

No held image

will be displayed, and no change will occur.

4.3.9. END - End editing

END

The End command terminates the editing session and returns control to the operating system. If the editor was called with an output file, the updated file will be written back before the editor terminates. If the editor was called without an output file, the editor will simply exit. If you called the editor without an output file and want any changes made in the editing session to be saved, you must write the file out with the "WF" command before entering the "END" command.

4.3.10. F - Find

F,‹lines› ‹text›

The Find command searches for a line beginning with a given string. Starting with the line following the current line, lines are examined until a line is found that begins with the specified text. Note that if the line to be found has leading spaces, the leading spaces must be included on the Find command. The case of the characters being compared is ignored by the Find command: a lower case character will match an upper case character, and vice versa. If the ‹lines› specification is supplied, the Find command will continue locating lines starting with ‹text› until the specified number of lines have been found. Hence, "F,1" is equivalent to "F". At the completion of the Find command, the current line pointer will be set to the last line found. If the end of the file is reached before the requested number of occurrences of ‹text› have been found, the message "*Eof*" will appear and the command will stop. If the ‹text› is omitted from the Find command, the last string used will be assumed. This is very useful when the end of file is reached and the user wishes to continue the Find command from the top.

The line-scanning operation performed by the Find command when

searching for the specified target string may be modified through use of the Column limits (CL) command. Normally, the entire line is subject to scrutiny, but the CL command may be used to set left and right column limits for such scrutiny. See the description of the Column limits command for further information.

4.3.11. HO - Hold

```
HO [<image number>]
```

The Hold (HO) command copies the current image to an internal editor save area, without deleting it from the file. The held image may be retrieved later using the Duplicate command. There are 9 internal save areas which may be used to hold different lines. The <image number> specification indicates in which save area the line is to be saved. If no specification is given, area 1 will be used.

4.3.12. HW - Horizontal window

```
HW <num1>[,<num2>]
```

The Horizontal window (HW) command sets column limits for the printing of text lines by the editor. If two numbers are specified, the first number is taken as the left limit, and the second as the right limit. If just one number is specified, it is taken as the right limit, and the left limit is assumed to be one (1).

Only the columns between (and including) the left and right limits will be displayed as the result of commands which, by their natrue, display text lines. The HW command affects only the printing of lines: the actual text is not affected. For example, if the following commands were performed:

```
I This is an example of the HW command.
HW 9,18
P
```

the editor would display:

```
an example
```

The HW command is most useful when long text lines (such as in an assembly print file) are being examined, and the console device cannot properly display such long images. In such a situation,

the HW command may be used to select any "x" consecutive columns to be displayed, where "x" is a character count up to which the console device will display reasonable information.

When the editor is invoked, the HW settings are 1,132, and they remain so set until changed by the user. The current HW column settings are displayed by the Status (ST) command.

4.3.13. I - Insert

I ⟨text⟩

The Insert command allows a single line of text to be inserted after the current line without the bother of going into Input mode, entering the line, and going back to Edit mode. The ⟨text⟩ following the "I" command will be treated exactly as if it were entered in Input mode. The line inserted by the Insert command will become the current line. If the ⟨text⟩ is omitted from the "I" command, the editor will enter Input mode just as if a null line were entered.

4.3.14. IB - Insert Before

IB ⟨text⟩

The Insert Before command inserts the ⟨text⟩ specified on the command before the current line. The current line pointer is unchanged.

4.3.15. L - Locate

L,⟨lines⟩ ⟨text⟩

The Locate command acts exactly like the Find command described above, except that it searches for the appearance of ⟨text⟩ anywhere in a line, not just at the beginning. The ⟨lines⟩ specification will cause the Locate to continue until the specified number of lines have been located. The current line pointer will be left at the last line located. If ⟨text⟩ is omitted, the text last used by a Find or Locate command will be used. The fact that the memory for the last text used for a Find or Locate is shared between the two commands allows the user to Locate text that was originally used on a Find without reentering it. This is so commonly done in Assembly and Fortran program

editing that the common text memory was provided to ease that task.

The line-scanning operation performed by the Locate command when searching for the specified target string may be modified through use of the Column limits (CL) command. Normally, the entire line is subject to scrutiny, but the CL command may be used to set left and right column limits for such scrutiny. See the description of the Column limits command for further information.

4.3.16. LA - Last

LA

The Last command simply moves the current line pointer to the last line in the file and prints it. This is the command normally used when appending information to the end of a file. The pointer is moved to the end with the Last command, then Input mode is entered by typing a null line so that the new text may be added.

4.3.17. MA - Set character mask

MA ‹character mask›

The editor's character masking feature is activated by the MA command. The single non-blank character specified as the ‹character mask› will become the mask character. For example, the command:

MA *

will set "*" as the mask character. The command:

MA

clears any pre-existing mask character and turns the masking feature back off.

Once a mask character has been defined via the MA command, it may be used to indicate a character position which will match any character for the Locate, Find, and Change commands. The mask character may be used any number of times in a Locate or Find command, but due to the unique way masks are handled by the Change command (see below), no more than nine occurrences of the mask character are permitted in the ‹old string› portion of a Change command.

When a mask character is used in the ‹old string› portion of a Change command, the character in that position in the line being changed is saved. When the mask character appears in the ‹new string›, the saved character will replace it in the line being changed. Up to nine characters may be saved in this way. The characters will be output in the same order they were originally saved. As an example of masking in the Change command, if "*" were the mask character, the command "C /A*B/AB*/" would change the string "AXB" to "ABX".

In using the Reverse Change (RC) and Reverse Change Within Range (RR) commands, occurrences of the mask character within the ‹new string› will be replaced by the characters saved by the last Change command rather than the characters masked by the ‹old string› in the RC or RR command.

4.3.18. O - Output

O ‹number›

The Output command types ‹number› lines starting with the next line. If ‹number› is omitted, only the next line will be printed. If ‹number› is an asterisk (*) or a number larger than the number of lines between the current line and the end of file, the remainder of the lines in the file will be printed followed by "*Eof*". As with all commands, prefixing the command with a sharp sign (#) will cause line numbers to be printed before the lines typed. The current line pointer will be left at the last line printed.

4.3.19. P - Print

P ‹number›

The Print command types ‹number› lines starting with the current line. In all other regards it is identical to the Output ("O") command described above. Since it starts with the current line, it may be used to simply to list the current line by typing just "P". Since this causes one line to be listed starting with the current line, the current line pointer will be left at the line printed.

4.3.20. R - Replace

R,⟨column⟩ ⟨text⟩

The Replace command allows either complete or partial replacement of the current line. If no ",⟨column⟩" is specified, the current line will be deleted and ⟨text⟩ will replace it. The action of the command will be identical to deleting the current line with a Delete command and then inserting ⟨text⟩ with an Insert command. If a ⟨column⟩ is specified, the text supplied on the command will overlay the contents of the current line starting at the specified column and continuing until the last character of ⟨text⟩. If a tab character (see below) appears in ⟨text⟩, it will cause the image being overlayed to be space filled until the next tab stop, where overlaying will resume in the next column.

As an added convenience, if the Replace command is of the form:

R,* ⟨text⟩

the "*" is taken to mean the last column of the current image. Thus, if the current line contains:

110 PRINT "This is an example"

Then R,* of the Replace command"

would result in:

110 PRINT "This is an example of the Replace command"

Note the use of one extra space after the "*" -- the first space is required to separate the text from the command, and the second space is the first character of the overlay text.

4.3.21. RC - Reverse change

RC,⟨lines⟩ /⟨new string⟩/⟨old string⟩/⟨per line⟩

The Reverse change command has the same effect as the Change command, except that the meaning of the two strings is reversed. This command is normally used without any specifications, or with just a ",⟨lines⟩" specification, to counteract the effects of an unwanted Change command. Caution should be exercised in its use, however, since it may have undesired effects. For example, if the current line contains:

```
This was an example
```

and the command:

```
C /was/is/
```

was performed, the line would then read:

```
This is an example
```

If the command:

```
RC
```

was now entered, the result would be:

```
Thwas is an example
```

4.3.22. RF - Read file

```
RF <file name>
```

The contents of the named text file will be read in and inserted following the current line. At the completion of the command, the current line pointer will be at the last line of the file read in. This command may be used to add the contents of one file to another, or initially, starting with an empty editor, to read in the file to be edited.

4.3.23. SC - Scale

```
SC [<num1>[,<num2>]]
```

The Scale (SC) command is a convenience feature to assist in determining the proper column numbers to use in CL, HW, and R,<column> commands. It simply prints a column number scale, starting in the column specified by <num1>, and ending in the column specified by <num2>. If only one column number is specified, the scale will begin in column one, and end in the specified column. If no specification at all is given, the scale will begin in column one and will extend to the maximum column number. For example:

```
SC 11,25
```

would result in:

being displayed.

4.3.24. ST - Status

ST

The Status (ST) command requires no parameters, and results in a display of the current settings of several editor variables. The display has the format:

TAB: tc t1,t2,...tn CL: cl1,cl2 HW: hw1,hw2

where tc is the current tab character. If no tab character is currently defined, "none" will appear in this field.

t1-tn are the current tab stop column settings. If no tab stops have been defined, this field will be absent.

cl1,cl2 are the current limits set by the CL command, affecting the Find (F), Locate (L), and Change (C) commands.

hw1,hw2 are the current limits set by the HW command, affecting the printing of lines by the editor.

4.3.25. T - Top

T

The current line pointer will be set to the top of the file being edited. If insertions are made at this point, the lines will be added before the first line of the file.

4.3.26. TA - Tab: set character and stops

TA ‹char› ‹stop›,‹stop›,...

The Tab command allows the user to define a tab character to be used to align text entered into columns. The ‹char› specification

is the character to be used as a tab, and the ⟨stop⟩ specifications are the columns in which the tabs are to be set. Upon encountering the tab character in text being added to the file in Input mode, or on an Insert ("I") or Replace ("R") command, the text editor will look for the tab stop with the lowest column number greater than the column the tab character would have been stored in if it were not a tab character. Spaces will be inserted in the line so that the character following the tab stop will be stored in that column. If no tab stop exists greater than or equal to the current column number, a single space will be inserted. This sounds very confusing, but is much easier to use than to explain. For example, suppose we wish to enter some assembly language text and have it tabbed to columns 11, 21, and 41. We know that we won't be using any semicolons in the text to be entered, so we choose a semicolon as the tab character. To activate tabbing of text, we use the command:

```
TA ; 11,21,41
```

Then, if we go into Input mode and enter:

```
TAG;LI;R1,1;Load a one
;MOV;R1,*R4+;Stuff it in the table
```

and then list the lines we entered, we will find:

```
TAG         LI          R1,1                Load a one
            MOV         R1,*R4+             Stuff it in the table
```

Note that the expansion of tab stops occurs even on a partial line Replace command. For example, if we were positioned at the last of the two lines entered above and typed (in Edit mode):

```
R,21 R1,2(R4);Place
```

the line would be changed to:

```
            MOV         R1,2(R4)            Place it in the table
```

If only the ⟨char⟩ specification is given on the Tab command, the tab character will be changed without affecting the current stops. If all specifications are absent, tabbing will be turned off entirely.

The current tab character and tab stops may be displayed through use of the Status (ST) command.

4.3.27. WF - Write file

WF ⟨file name⟩

The file being edited will be written out to the named file. At the completion of the Write File command, the current line pointer will be set to the top of the file being edited. If the ⟨file name⟩ is omitted, the file will be written back to the output file specified when EDIT was called (see "Calling the editor" above). If no output file was specified on the EDIT command, the file name may not be omitted on the Write File command. The Write File command may be used to save the contents of the editing session at convenient points through it so that loss of power does not cause extensive loss of editing.

4.3.28. X - Execute held image

X [⟨image number⟩]

The image saved by the most recent Hold (or Delete and Hold) command for the specified ⟨image number⟩ will be retrieved and executed as if it were a command entered from the keyboard. This is a handy time-saver when a lengthy command will be needed several times. If no ⟨image number⟩ is specified, 1 will be assumed.

4.3.29. + - Move forward in file

+ ⟨number⟩

The Move Forward command will advance the current line pointer ⟨number⟩ lines in the file, and print the new current line. If ⟨number⟩ is omitted, one is assumed. Hence, entering just a plus sign will advance to the next line.

4.3.30. - - Move backward in file

- ⟨number⟩

The Move Backward command will back the current line pointer up ⟨number⟩ lines in the file, and print the new current line. If ⟨number⟩ is omitted, one is assumed, so the pointer may be backed up to the previous line simply by entering a minus sign.

4.3.31. . - Comment

If the first non-blank character of a command line is a period, the command line will be ignored by the editor. This allows comments to be included in files of editor commands.

4.3.32. ⟨number⟩ - Go to line by number

⟨number⟩

If a simple number is entered, the editor will go to that line in the file. The lines currently in the file are numbered starting at one. Note that if lines have been inserted or deleted during an editing session, the line numbers will not correspond to a listing of the file made before the editing session began.

4.4. Line ranges

The line range facility allows the user to define a range of lines within which a special set of commands will execute. This feature is very convenient when performing extensive editing on a section of a larger file. Since all of the range commands will limit their action to the defined portion of the file, there is no danger of having them "run away" and process a line outside the intended range. The following commands are provided to define and use line ranges.

4.4.1. BE - Defing beginning of range

BE

The Begin command marks the current line as the first line in the line range. If another line had been previously so marked, the new line will supercede it as the top of the range. The use of any of the range-related commands described below will mark the current line at the time of their execution as the end of the line range. Use of any command (such as Delete) which would alter the actual line number of either the beginning or the end line of the line range will cause the end of the line range to be marked undefined.

4.4.2. ER - Mark end of line range

ER

The End Range command marks the current line as the last line in the line range. The line range now defined between the line marked by the BE command (the top) and the ER command (the bottom), inclusive, will be the part of the file on which the following range-oriented commands operate. If no end of range has been defined when one of the following commands is entered, it will be set to the current line when the command is entered.

4.4.3. Range-oriented commands

The following commands are range-oriented versions of regular editor commands:

Range	Regular	Function
CR	C	Change
DR	D	Delete
FR	F	Find
LR	L	Locate
OR	O	Output
PR	P	Print
RR	RC	Reverse Change
TR	T	Top

The above commands operate exactly like their non-range analogues, except that any searching or movement of the current line pointer will be restricted to the line range previously defined. The TR command will position the current line pointer to the first line in the range.

4.4.4. CO - Copy range of lines

CO [<file name>]

The Copy command writes the lines within the current range into the named file. If no <file name> is specified, TEMP3$ will be assumed. The lines in the file being edited will be unchanged. This command is useful when splitting up a large file into several smaller files.

4.4.5. DC - Destructive copy range of lines

DC [<file name>]

The Destructive Copy command is identical to the Copy command, except that the lines written to the output file will be deleted from the file being edited after being written out. This command is used primarily when moving a set of lines within a file being edited. The lines to be moved are marked as a range, then written out and deleted with the DC command. The current line pointer is then moved to the line after which the moved text is to be inserted, and the lines are read back in with the RF command.

4.5. Interrupting a command

An in-progress command may be halted by pressing the Control C key on the keyboard. The editor stops processing the current command and leaves the current line pointer at the line being processed when the Control C key was struck. If Control C is struck while in Input mode, the editor will enter Edit mode on the current line.

5. The editor backing files

When the size of the file being edited exceeds the size of the system's free memory, the editor will page the file back and forth between two scratch files. These files should be created in the user's directory when the editor is installed in the system. These files are named TEMP1$ and TEMP2$, and should be each large enough to hold the largest file to be edited plus a 15 percent safety margin. If the editor runs out of memory or exceeds the maximum size of these files, the message:

Buffer impasse.

will be issued and no further additions to the file will be permitted until the problem is corrected.