Z80®-RIO

Text Editor User's Manual

TABLE OF CONTENTS

Other pertinent documentation with which the reader may want to become familiar include:

RIO Operating System User's Manual

1. INTRODUCTION

The RIO Text Editor is a line-oriented editor with string handling capability and automatic interface to the disk. It uses a memory paging technique which allows any size text file to be edited. The Editor automatically determines the work space available, and brings blocks of text into its memory buffer as required by the command issued.

The portion of the user's file which is contained within this memory buffer is called the 'window'. If a command is issued that operates on text which is not in the window, the Editor will write out the memory buffer onto disk and read the next sequential block of text into the window. This action is referred to as 'rolling'. The Editor rolls blocks of text in and out of the window until the required one is found.

The Editor maintains a 'current line pointer' which points to the line last referenced. At the beginning of the editing session this pointer points to a null line at the top of the file. After executing each command the Editor will respond with a prompt '>' to indicate that a new command should be issued. Note that in Input mode no prompt is output. The Editor can handle lines up to a maximum length of 512 characters.

The Editor can be used to create a new text file or to modify an already existing one. If the Editor is given a file name which does not exist in the disk directory it will create a new file with the specified name, and then automatically enter Input mode (see Input command definition). This file will be created with a record length of 128 bytes unless otherwise specified in the 'RL' option.

If the specified file already exists, a backup file is created before the editing session begins. This file is a duplicate of the user file, thus if the user file should get damaged during editing, the user still has a copy of the original file. This backup file is created with the same name as the user file but with an extension of 'OLD'. This extension will be in upper or lower case depending on the first letter of the user file name, i.e., editing a file called 'MYFILE' would cause the creation of a backup file 'MYFILE.OLD', whereas editing 'myfile' would cause the

creation of a backup file 'myfile.old'. One of two options may be used when invoking the editor to either override the default name for the backup file or to suppress its creation entirely. These options are called the OLD option (O) and No backup option (N), respectively.

The Editor is invoked from the RIO executive via the command:

```
EDIT filename [options]
```

where: filename is a standard RIO file name which may be fully or partially qualified;

where: option is one of the following:

O=filename1	specifying a name for the backup file
N	specifying that no backup file should be created
RL=record length (in hex)	specifying that the new file should be created with the given record length

If no filename is given the Editor will respond with:

```
NAME?
```

In this case, the user should name a file to edit or enter a carriage return to return to the RIO executive.

If the specified file is not of type ASCII (20H) the message:

```
INVALID ATTRIBUTES:  filename
```

will be output to the console. This message is also output if a file specified in the GEt command has an invalid type or record length.

If the named file does not exist, it will be created and the Editor will output the message:

```
NEW FILE
INPUT
```

and enter Input mode.

If the file already exists, the Editor will create the backup file and then output the message:

```
EDIT
```

EXAMPLES:

Creating a new file with the Editor:

```
%EDIT MYFILE RL=400     ;MYFILE does not exist and is
                         created with a record length
                         of 400 hex
NEW FILE
INPUT
  :
  :                     ;input text
  :
                        ;null line
EDIT                    ;edit mode
>QUIT                   ;update and close user file
```

Editing an already existing file with a default backup file:

```
%EDIT MYFILE            ;MYFILE exists
EDIT                    ;backup MYFILE.OLD created
>
  :
  :                     ;edit session
  :
>QUIT                   ;update and close user file
```

Editing an already existing file, using the Old option:

```
%EDIT MYFILE O=$YDOS:2/BACKUP   ;MYFILE exists
EDIT                            ;backup file BACKUP
                                 is created under
                                 file system YDOS
                                 on drive 2
>
  :
  :                             ;edit session
  :
>QUIT                           ;update and close
                                 user file
```

Editing an already existing file, using the No backup option:

```
%EDIT MYFILE N          ;MYFILE exists
EDIT                    ;no backup file created
>
  :
  :                     ;edit session
  :
>QUIT                   ;update and close MYFILE
```

If a disk error occurs during an operation on either the user file or backup file, the Editor closes the files and returns to the RIO Executive. At this time a message:

```
I/O ERROR xx ON UNIT yy
```

will be output to the console, where xx is the error return code (see RIO Software User's Manual), and yy is the logical unit number: 04 for the user file, 05 for the backup file.

The editor environment is exited via the QUIT command. This command will cause the user file to be updated and closed and will then return to the RIO Executive.

If the user file was damaged during the editing session (via disk errors or misuse of editing commands), the user may wish to QUIT and begin again. This can be done in several ways. Following is a description of three methods with an example for each.

One method is to QUIT from the Editor environment, DELETE the user file, and then RENAME the backup file:

```
%EDIT MYFILE                ;invoke Editor
EDIT
>
  :
  :                         ;edit session
  :
>QUIT                       ;quit edit environment
%DELETE MYFILE              ;delete the user file
DELETE 2/MYFILE (Y/N/A/Q)?Y
%RENAME MYFILE.OLD MYFILE   ;rename backup file
MYFILE.OLD ----> MYFILE
%
```

Another method is to remain in the Editor environment, DElete all of the lines in the user file, and then GEt the backup file:

```
%EDIT MYFILE                    ;invoke Editor
EDIT
>
  :
  :                             ;edit session
  :
>T                              ;go to the top of the
                                 file
T>DE *                          ;delete all of the lines
                                 in the file
>GE MYFILE.OLD                  ;insert contents of the
                                 backup file
last line
>
  :
  :                             ;continue edit session
  :
>QUIT                           ;close and update user
                                 file
%
```

If a disk error occurs while operating on the user file, the Editor will automatically return to the RIO Executive. In this case the user may wish to DELETE the damaged user file, create a new file with the Editor, and then GEt the backup file:

```
%EDIT MYFILE                    ;invoke Editor
EDIT
>
  :
  :                             ;edit session
  :
I/O ERROR C4 ON UNIT 04         ;disk error C4 on user
                                 file
%DELETE MYFILE
DELETE 2/MYFILE (Y/N/A/Q)Y      ;delete user file
%EDIT MYFILE                    ;create new file with
                                 Editor
NEW FILE
INPUT
                                ;null line
EDIT
>GE MYFILE.OLD                  ;insert contents of
                                 backup file
last line
>
  :
  :                             ;edit session
  :
>QUIT                           ;update and close user
                                 file
%
```

2. EDITING COMMANDS

The Editor currently offers 24 commands which are executed via a one or more letter code. The commands may be issued in either upper case or lower case characters. In the following summary, the capital letter (or letters) indicates the minimum call for each command:

Again	Find	LIneno	PUt	Up
Bottom	GEt	Locate	PUTD	Verify
BRief	Goto	Macro	QUIT	Window
Change	Input	Next	Replace	Xecute
DElete	Join	Print	Top	

There are two general forms of command modifiers which are used with many of the edit commands: a number n, or a string.

The first form, a decimal number n, indicates the number of times the command operation is to be repeated. For example, Print 15 would output 15 lines to the console. The symbol '*' can be used to indicate that the operation should be repeated over the entire range of the file beginning with the current line.

The second form is a string of characters between delimiters. This form indicates that the command should repeat until the string is found. For example, Print /LD A,B/ would print the current line and each following line until it printed the line containing the first occurrence of the string 'LD A,B'. A delimiter is defined as the first nonblank, nonnumeric character in the modifier. The string may contain any character except for the delimiter. The second delimiter is optional. In the following discussion, the character / is used to represent any valid delimiter.

If the string is not found in the current window, the message:

```
STRING NOT IN BLOCK
PROCEED?
```

will be output to the console. If a 'Y' is entered the command will execute on the current window and the next block will be brought in from the disk. Otherwise, the command will terminate, leaving the pointer pointing to the current line. When using the string modifier with the Up command, only the first message will be output and the option to proceed will not be given. The pointer will be left on the current line.

These modifiers are used with the following commands:

Delete, Next, Print, PUt, PUTD, Up

In general, commands and modifiers must be separated by at least one or more blanks.

The Editor has two modes - Brief and Verbose - which affect the Bottom, Change, Find, Goto, Get, Locate, Next, and Up commands. When in Verbose mode (the default mode) a line of text is output following each of these commands. When in Brief mode this printing is suppressed.

Brief mode can be entered in two ways. The first way is via the BRief command and will affect all commands until Brief mode is exited via the Verbose command.

Brief suppression can also be obtained for a single command by issuing a dot '.' immediately after the command (i.e., F. /LOOP1/).

When a '?' is entered during the execution of the Change or Print commands, execution will stop until another '?' is entered. If an ESC is entered, the command will abort and the editor will return to Edit mode. The commands check for a '?' or ESC after processing each line. After an ESC is recognized, the pointer is left at the last line processed. Note: this feature does not apply to the ZDS system.

The following notation is used in the command definitions in section 3:

> Portions of a modifier that are optional are enclosed in brackets [].
>
> The symbol for logical or, ¦, is used if either option can be used, i.e., DE [n/string[/]] can be expanded as DE n or DE /string/.
>
> Parameters which can be repeated more than once are followed by an asterisk * - i.e., J &command&[command&]*

2.1 RIO EDITOR COMMAND SUMMARY

COMMAND ABBREVIATION	COMMAND NAME	COMMAND PARAMETERS
A	Again	[# of times]
B	Bottom	
BR	Brief	
C	Change	/old string/new string/[# of lines[# of times per line]]
DE	Delete	[# of lines ¦ /string/]
F	Find	/string/
GE	Get	[filename]
G	Goto	line #
I	Input	[text line]
J	Join	&command&command&...
LI	Lineno	
L	Locate	/string/
M	Macro	&command&command&...
N	Next	[line # ¦ /string/]
P	Print	[# of lines ¦ /string/]
PU	Put	[# of lines ¦ /string/[filename [record length]]]
PUTD	Put and Delete	[# of lines ¦ /string[filename [record length]]]
Q	Quit	
R	Replace	[text line]
T	Top	
U	Up	[# of lines ¦ /string/]
V	Verify	
W	Window	
X	Xecute	

3. COMMAND DEFINITIONS

3.1 Again [n]

Function:

Repeats the previous command n times. If n is not specified the previous command is repeated once. When the Again command is issued after a Join or Xecute command, only the last specifed single command will be repeated.

Examples:

```
>P 3                ;print 3 lines
line 1
line 2
line 3
>A                  ;print another 3 lines
line 3
line 4
line 5
>J &U 4&P 4&        ;join command (up 4, print 4)
line 1              ;up 4 lines
line 1              ;print 4 lines
line 2
line 3
line 4
>A                  ;repeat last command (print 4)
line 4
line 5
line 6
line 7
>
```

3.2 Bottom

Function:

Moves the current line pointer to the last line of the file and prints the line on the console. If Brief mode has been set the printing of the last line will be suppressed.

Examples:

```
>B              ;move pointer to bottom of file
last line
>N              ;next line
EOF
>
```

3.3 Brief

Function:

Causes the editor to enter Brief mode in which the normal printing of the line of text following the Bottom, Change Find, Goto, Get, Locate, Next, and Up command is suppressed.

Examples:

```
>P                    ;print 1 line
This is line 10
>C /10/11             ;change 10 to 11
This is line 11       ;in Verbose mode - line is
                       printed after change
>BR                   ;enter Brief mode
>C /11/10/            ;change 11 to 10, in Brief mode,
                       change does not print line
>P                    ;print 1 line
This is line 10
>
```

3.4 Change /old string/new string[/[n1[n2]]]

Function:

Locates 'old string' within the range specified by n1 and n2, and replaces it with 'new string'. n1 specifies the number of lines in which 'old string' should be looked for and changed. n2 specifies the number of occurrences per line to be changed. The Change command begins its search in the current line (unlike the Find and Locate commands which begin the search in the next line). If n1 and/or n2 is not specified, the default is one. A '*' can be used for n1, specifying that all lines from the current line on be changed, or for n2 specifying that all occurrences in the specified lines be changed. When in Verbose mode the line will be printed after the change has been made. This is suppressed in Brief mode. If 'new string' is not given (i.e., C /old string//), 'old string' will be deleted. If 'old string' is not found the message:

NO CHANGE

will be output to the console and the command will terminate. Upon termination of the Change command, the current line pointer is left on the last line in which 'old string' was looked for. The execution of the Change command may be temporarily or permanently halted (on MCZ systems) via the ? or ESC mechanism described in section 2.

Examples:

```
>P                ;print 1 line
   ADD A,B
>C /A,B/HL,DE     ;change string 'A,B' to 'HL,DE'
   ADD HL,DE
>P 4              ;print 4 lines

line 110
line 111
line 112
line 113
>U 3              ;up 3 lines
line 110
>C /1/2/3 2       ;change first 2 occurrences of '1' to '2'
                   in next 3 lines
line 220
line 221          ;note only 2 occurrences changed
line 222

>T                ;go to top of file
T>C /IX/IY/* *    ;change every occurrence of IX to IY
```

3.5 DElete [n|/string[/]]

Function:

Deletes lines from the file beginning with the current line. If n is specified, n lines will be deleted. If a string is specified, all lines up to but not including the line containing the specified string will be deleted. After deleting, the current line pointer is left on the line after the last line deleted.

Examples:

```
>P 5            ;print 5 lines
line 1
line 2
line 3
line 4
line 5
>U 4            ;up 4 lines
line 1
>DE 2           ;delete 2 lines
>P              ;print 1 line
line 3
>


>P 5            ;print 5 lines
line 1
line 2
line 3
line 4
line 5
>U 4            ;up 4 lines
line 1
>DE /2/         ;deletes up to first line containing '2'
>P              ;print 1 line
line 2
>
```

3.6 Find /string[/]

Function:

Moves the current line pointer to the first line following the current line which contains the specified string beginning in column one. This command is a special case of the Locate command and is useful in locating labels in assembly language source.

Examples:

```
>P 5                    ;print 5 lines
   LD A,B
   ADD A,C
   JR Z,LOOP1
   INC A
LOOP1: LD D,A
>U 4                    ;up 4 lines
   LD A,B
>F /LOOP1/              ;find LOOP1 beginning in column 1
LOOP1: LD D,A
>U 4                    ;up 4 lines
   LD A,B
L /LOOP1                ;locate LOOP1 in any column
   JR Z,LOOP1
>
```

3.7 GEt [filename]

Function:

Reads a disk file and inserts its contents into the user file after the current line. If no filename is specified, the temporary PUT/GET file created by a PUT or PUTD command is inserted. If a filename is specified, the entire contents of this file will be inserted. The current line pointer is moved to the last line of the inserted file. Note that it is not possible to insert just a part of a file with the GEt command. This can be done, however, by 'PUT'ting the desired portion onto a separate file and then 'Get'ting this new file. If a file name is specified, the file must be of type ASCII with a record length of less than or equal to 512. If it is not, the message:

INVALID ATTRIBUTES: filename

will be output to the console.

Examples:

```
>P 5                    ;print 5 lines
line 1
line 2
line 3
line 4
line 5
>U 3                    ;go up 3 lines
line 2
>PUT 2                  ;put 2 lines into temporary file
>P                      ;print 1 line
line 4
>B                      ;go to the bottom of the file
line 5
>GET                    ;insert contents of the temporary file
line 3
>T                      ;go to the top of the file
T>P *                   ;print all of the file
line 1
line 2
line 3
line 4
line 5
line 2
line 3
EOF
>
```

```
>P 3                    ;print 3 lines
line 1
line 2
line 3
>GET EXTFILE.TEXT      ;insert the contents of file
external line 5
>T                     ;go to the top of the file
T>P *                  ;print all of the file
line 1
line 2
line 3
external line 1
external line 2
external line 3
external line 4
external line 5
EOF
>
```

3.8 Goto n

Function:

Moves the current line pointer to point to the line with the specified decimal line number n

Examples:

```
>P 5            ;print 5 lines
line 1
line 2
line 3
line 4
line 5
>T              ;go to the top of the file
>G 3            ;go to the third line in the file
line 3
>
```

3.9 Input [text line]

Function:

Inputs text into the file after the current line. If a line of text is given, it will be inserted. Note that the text line must be separated from the command 'I' by one blank. Any additional blanks will be treated as part of the text line, i.e., I line 1A will cause 'line 1A' to be inserted, whereas I line 1A will cause ' line 1A' to be inserted. If no line is specified, the Editor will enter Input mode. In this mode, all text that is entered from the console is inserted after the current line. Input mode is terminated when a null line is entered (by typing just a carriage return). Upon termination of the Input command the current line pointer points to the last line input.

Examples:

```
>p              ;print 1 line
line 1
>I line 1A      ;insert 'line 1A' into text
>P              ;print 1 line
line 1A
>I              ;enter Input mode
INPUT
line 1B
line 1C
line 1D
                ;null line
EDIT
>T              ;go to top of file
T>P *           ;print all of file
line 1
line 1A
line 1B
line 1C
line 1D
EOF
>
```

3.10 Join &command&[command&]*

Function:

Causes the specified sequence of commands to be executed as soon as the carriage return is received. Join is similar to the Macro command immediately followed by an Xecute command. Any number of commands can be concatenated as long as they fit on a single line (512 characters). Spaces are not allowed between the commands and delimiters. Any delimiter may be used, however, it must not occur in any of the commands.

Examples:

```
>J #T#L /LOOP1#C /1/2#U 3#P 5
T                   ;Top command
LOOP1:              ;Locate command
LOOP2:              ;Change command
   LD A,B           ;Up command
   LD A,B           ;Print command
   DEC B
   JR Z,LOOP3
LOOP2:
   DEC BC
>A                  ;repeat last command
   DEC BC           ;Print command
   LD A,(HL)
   CP ASCICR
   JR NZ,LOOP2
   JR END
>
```

3.11 LIneno

Function:

Prints the line number of the current line. This command can be used with the Goto command to operate the editor on a line number concept.

Examples:

```
>P                  ;print 1 line
line number 10
>LI                 ;determine line number
10
>
```

3.12 Locate /string[/]

Function:

Moves the current line pointer to the first line, following the current line which contains the specified string.

Examples:

```
>L /200                 ;locate string '200'
   LD HL,200H
>A                      ;locate it again
STRING NOT IN BLOCK
PROCEED?
>Y                      ;look in next block
   LD DE,200H
>A                      ;and again
STRING NOT IN BLOCK
PROCEED?
>N                      ;terminate command
   ADD HL,DE            ;last line of block
>
```

3.13 Macro &command&[command&]*

Function:

Causes the specified sequence of commands to be loaded into the macro buffer to be executed each time the Xecute command is issued. Any number of commands can be concatenated as long as they fit on a single line (512 characters). Note that spaces are not allowed between commands and the delimiting character. Any character may be used as a delimiter, however, it may not occur in any of the commands. When the editor is first initialized, the macro buffer contains the commands: $U. 6$P 12$. Thus issuing the Xecute command before a Macro command will cause these commands to be executed.

Examples:

```
>M &T&L /A,B/&C /A,B/A,C/      ;initialize macro buffer
>P 5                           ;print 5 lines
LOOP:
   LD A,(HL)
   ADD A,B
   INC HL
   DEC B
>U 10                          ;up 10 lines
   JR NZ,LOOP
>X                             ;Xecute contents of macro buffer
T                              ;Top command
   ADD A,B                     ;Locate command
   ADD A,C                     ;Change command
>
```

3.14 Next [n!/string[/]]

Function:

Causes the current line pointer to be moved down n lines or to the first line which contains the specified string.

Examples:

```
>P              ;print 1 line
line 3
>N 5            ;go down 5 lines
line 8
>T              ;go to top of file
T>P 2           ;print 2 lines
line 0
line 1
>N /3/          ;go down to line containing '3'
line 3
>
```

3.15 Print [n|/string[/]]

Function:

Prints, beginning at the current line, the next n lines or until the first occurrence of the specified string. Upon termination of the command the current line pointer is left on the last printed line. The Print command can be temporarily or permanently halted via the ? and ESC mechanism described in section 2.

Examples:

```
>P 3            ;print 3 lines
line 1
line 2
line 3
>P /7/          ;print until the string '7' is found
line 3
line 4
line 5
line 6
line 7
>
```

3.16 PUt [n|/string[/[filename[RL=m]]]]

Function:

Writes onto a disk file, starting with the current line, n lines or lines up to but not including the first occurrence of the specified string. If a file name is specified the PUt will be made to a file of this name. If one already exists it will be erased and replaced by the new file. If the file exists it must have a record length of less than or equal to 512. If it does not exist, it will be created with a record length of 128, unless otherwise specified via the RL option. Note: m is specified in hex. If the RL option is used, m must be less than or equal to 200H. If no name is specified, the PUt will be made to the temporary Put/Get file, thus overwritting any text previously PUt there. Note that there must be a space between the second string delimiter and the file name. Upon termination of the PUt command the current line pointer is left at the line following the last line written.

Examples:

```
>PU 3                    ;puts 3 lines into the temporary
                          PUt/GEt disk file

>PU /A,B/                ;puts all lines until string 'A,B'
                          is located into the temporary
                          Put/Get disk file

>PU 6 PUT.FILE           ;puts 6 lines into a file called
                          PUT.FILE which is created with
                          a record length equal to 128.

>PU 6 PUT.FILE RL=200    ;puts 6 lines into a file called
                          PUT.FILE which is created with
                          a record length equal to 512.
```

3.17 PUTD [n|/string[/[filename]]]

Function:

Writes onto a disk file, starting with the current line, n lines or lines up to the first occurrence of the specified string and deletes those lines from the user file. If a file name is specified, the PUt will be made to a file of this name. If one already exists it will be erased and replaced by the new file. If the file exists it must have a record length of less than or equal to 512. If it does not exist it will be created with a record length of 128, unless otherwise specified via the RL option. Note: m is specified in hex. If the RL option is used, m must be less than or equal to 200 hex. If no name is specified, the PUt will be made to the temporary Put/Get file, thus overwriting any text previously PUt there. Note that there must be a space between the second string delimiter and the file name. Upon termination of the PUt command the current line pointer is left at the line following the last line written.

Examples:

```
>T              ;go to the top of the file
T>P 5           ;print 5 lines
line 1
line 2
line 3
line 4
line 5
>T              ;go back to the top
T>PUTD 3        ;put and delete 3 lines into temporary file
>B              ;go to bottom of file
line 5
>GE             ;insert temporary file
line 3
>T              ;go to the top of the file
T>P *           ;print all of the file
line 4
line 5
line 1
line 2
line 3
EOF
>
```

3.18 QUIT

Function:

Updates and closes the user file and returns control to the RIO executive.

Examples:

```
%EDIT MYFILE
EDIT
>
  :
  :                    ;edit session
  :
>QUIT                  ;update and close MYFILE
%                      ;return to RIO Executive
```

3.19 Replace [text line]

Function:

Replaces the current line with the specified text. If a line of text is given, it will replace the line. Note that the text line must be separated from the command 'R' by one blank. Any additional blanks will be treated as part of the text line, i.e., R line 4A will cause 'line 4A' to be inserted whereas R line 4A will cause ' line 4A' to be inserted. If no line is specified, the Editor will enter Input mode. In this mode, all text that is entered from the console is inserted, replacing the current line. Input mode is terminated when a null line is entered (by typing just a carriage return). The current line pointer is left pointing to the last line input.

Examples:

```
>P 2            ;print 2 lines
line 4
line 5
>r line 4A      ;replace current line with string 'line 4A'
>U              ;go up 1 line
line 4
>P 2            ;print 2 lines
line 4
line 4A
>R              ;replace the current line with what follows
INPUT
line 4AA
line 4AB
line 4AC
                ;null line
>U 4            ;go up 4 lines
line 3
>P 5            ;print 5 lines
line 3
line 4
line 4AA
line 4AB
line 4AC
>
```

3.20 Top

Function:

Moves the current line pointer to the null line just above the first line of the user file.

Examples:

```
>P 3            ;print 3 lines
line 1
line 2
line 3
>T              ;go to the top of the file
T>I line 0      ;insert string at top
>P 4            ;print 4 lines
line 0
line 1
line 2
line 3
>
```

3.21 Up [n|/string[/]]

Function:

Moves the current line pointer to the line that is up n lines from the current line, or the first line (moving up) which contains the specified string. Note that Up will only work within the current window when using a string parameter. If a line containing that string is not found, the message:

STRING NOT IN BLOCK

will be output to the console, and the command will terminate leaving the pointer at the first line of the current window.

Examples:

```
>P                      ;print 1 line
line 10
>U 4                    ;up 4 lines
line 6
>U /3                   ;up to line containing '3'
line 3
>B                      ;go to bottom of file
line 900
w>U /line 3/            ;up to line containing 'line 3'
STRING NOT IN BLOCK     ;string not found
>P                      ;print 1 line
line 504
>
```

3.22 Verify

Function:

Exits Brief mode and resumes the normal printing of the line of text following the Bottom, Change, Find Get, Locate, Next, and Up commands.

Examples:

```
>BR             ;enter Brief mode
>N 3            ;go down 3 lines
>V              ;exit Brief mode
>N 3            ;go down 3 lines
line 20
>
```

3.23 Window

Function:

Displays on the console the line numbers of the top and bottom lines of the current window. This is useful because it allows the user to complete all editing in this window before editing lines that are not yet in memory, thus reducing the need for unnecessary disk I/O and therefore speeding up the editing function.

Examples:

```
>WI
0001
0500
>
```

3.24 Xecute

Function:

Causes the commands in the macro buffer to be executed.

Examples:

```
>M &U 3&P 6&     ;initialize macro buffer
>P               ;print 1 line
line 30
>X               ;execute the commands in the macro buffer
line 27          ;up 3
line 27          ;print 6
line 28
line 29
line 30
line 31
line 32
>
```