gri-909

source text editor

GRI Computer Corporation

320 NEEDHAM STREET, NEWTON, MASSACHUSETTS 02164

Price $3.75

GRI-909

Source Text Editor

GRI Computer Corporation, 320 Needham Street, Newton, Massachusetts 02164

72-44-001-B
0371

CONTENTS

C H A P T E R O N E

THE GRI-909 SOURCE TEXT EDITOR

The GRI-909 software package includes a comprehensive source text editor which provides the user with a convenient method for

- generating source text on paper tape (the source program input to an assembler is on paper tape)
- correcting and updating source text tapes through keyboard control from the teletype
- listing source text tapes on the teletype

1.1 INTRODUCTION

Source text editors eliminate most of the problems associated with the use of paper tape as an input medium. These problems arise from the limited flexibility of teletypewriters and from the inability to re-arrange information on paper tape. The GRI-909 EDITOR uses core memory as an extension to the teletype and allows the manipulation of source text in distinct and independent one-line units.

The EDITOR reads lines of text (from paper tape or the ASR keyboard) into a buffer in core memory where they are available for examination or manipulation. In particular, the EDITOR supports both line and content-oriented commands to delete, correct, preserve or add ASCII-coded text on source tapes. The lines entered into the buffer may be individually operated upon in order to prepare a correct sequence of text lines <u>before</u> they are punched onto paper tape.

1.2 SYSTEM CONVENTIONS

The GRI-909 EDITOR is designed as a part of a larger software system comprised of the EDITOR and the FAST and BASIC Assembler Programs. These programs all operate on source text and support the same conventions regarding its organization as well as error-recovery mechanisms. These conventions apply throughout the system to text being input either through the ASR keyboard or by a paper tape reader.

Source text is organized into variable length segments called lines and blocks. Certain special characters are reserved to delimit lines and blocks and others permit error recovery within a line.

1.2.1 SOURCE TEXT LINE

A source text line contains up to 80 ASCII characters and is terminated by a *carriage-return*, denoted by). The) may optionally be followed by a *line-feed* character. The line is the basic unit of source text which is operated upon by either the EDITOR or an Assembler.

When the EDITOR inputs a line through the keyboard, the user does not type a *line-feed* after the *carriage-return*. The EDITOR inserts a *line-feed* after the *carriage-return* when punching paper tape so that the tape may be listed off-line. The EDITOR and the Assemblers ignore the *line-feed* when it is encountered on paper tape.

1.2.2 ERROR RECOVERY

When source text is input to one of the system programs, the special characters *rubout* and *back-arrow* (←) are used to facilitate error recovery within a line of text.

When *rubout* is encountered, all previous characters in the current line will be ignored. The current line begins with the next character after the *rubout*.

The *back-arrow*, ←, causes the input routine to ignore (backspace over) the character which immediately precedes it. Two or more consecutive ← characters cause the input routine to ignore the corresponding number of characters preceding the first ←.

Note that the error recovery characters apply only to text contained within the current line.

1.2.3 SOURCE TEXT BLOCK

A source text block contains one or more lines followed by a block terminator. The number of lines in a block is at the discretion of the user - usually a block will consist of a logical sequence of lines such as a separate source program routine or a major segment of a routine. In order to facilitate subsequent re-editing of a source tape, its blocks should be of manageable length; that is, at least short enough so as to be wholly contained in the EDITOR's memory buffer (approximately 3600 characters assuming 80 character lines). The size of the buffer may be expanded for 8K machine users to extend down into the upper memory stack by changing LOC 1623_8 from 170244_8 (4K machines) to 160244_8 (8K machines) giving the user approximately 11,340 characters, assuming 80 character lines.

Lines within a block are implicitly numbered in decimal notation, beginning with line number 1. Line numbers are not punched onto a source tape. When a listing is generated by a GRI-909 Assembler, these implicit line numbers are printed to the left of source text lines. When the block terminator is encountered, the line numbering begins again at 1. Note that when a block is read into the EDITOR's buffer the lines contained therein have the same implicit numbers as appeared on the assembly listing.

The block terminator, which immediately follows the last line in the block, is either the ASCII *alt mode* character or the ASCII *esc* character. On a standard teletype, the block terminator key is located directly above the *ctrl* key on the left side of the keyboard.

Blocks of source text are preceded and followed by at least four inches of blank tape. This blank tape is ignored by system program input routines.

1.3 MODES OF OPERATION

The EDITOR is structured to operate in two distinct modes:

COMMAND Mode

TEXT Mode

In order to select an EDITOR function, the user enters the COMMAND mode. In this mode, the user may issue any of a pre-defined set of commands (described in the next chapter). The EDITOR interprets the command and takes all actions necessary for command execution. Many of the commands cause the EDITOR to enter the TEXT mode. In this mode, the user may input source text directly into the memory buffer.

Depending on the command, text is entered either through the teletype keyboard or from a paper tape reader.

1.3.1 COMMAND MODE

At the completion of the initialization procedure (see Appendix A), the EDITOR outputs an asterisk (*) to the teletype and places itself in COMMAND mode. In this mode, pre-defined user commands are communicated through the teletype keyboard. At the termination of each command execution, the EDITOR re-enters COMMAND mode and causes output of an asterisk to the teletype.

1.3.2 TEXT MODE

Certain commands such as appending, inserting, or changing the lines in the memory buffer cause the EDITOR to enter the TEXT mode. The EDITOR, when the TEXT mode is entered, types a 3-digit line number on the teletype (except when the text is appended from paper tape). This line number corresponds to the position of a line location pointer within the memory buffer. In the TEXT mode, the user may type consecutive lines of source text. At the beginning of each line of source text, the EDITOR updates the line location pointer and prints the corresponding line number. At any time, the user may exit from the TEXT mode by typing the *alt mode* or *esc* character *(alt/esc)* on the teletype keyboard. This causes the EDITOR to return to the COMMAND mode.

1.4 COMMAND STRUCTURE

EDITOR commands, which are entered from the keyboard, are of the following general form:

```
[COMMAND CHARACTER]     [OPTIONAL COMMAND PARAMETERS]     ⮐
```

Spaces are not allowed in a command and optional command parameters are separated by *commas* or, in the case of the MOVE and EXCHANGE commands, by a *colon*.

The command character is selected from the set of valid commands which are described in the following chapter. Parameters each denote line numbers or counts and may be composed of:

a. a string of ASCII characters enclosed in *ctrl-L* characters and denoting the first line number in which the string is contained. The *ctrl-L* character is echoed by the editor as the character \ (*backslash*) to avoid output confusion with the ***form-feed*** code which is also defined as *ctrl-L* The character *backslash* which is generated by a *shift-L* is also echoed as \ but does not serve to delimit a character string as does *ctrl-L*

b. a line number (not greater than 999) denoting itself

c. the special character, *period* (.) which denotes the line number corresponding to the current position of the line location pointer, i.e. to the line last operated on.

d. the special character, *slash* (/) which denotes the line number corresponding to the last line in the source buffer

e. a decimal number count (not greater than 32,767) denoting itself

Parameters may also be composed of meaningful combinations of the above separated by the arithmetic operators *plus* (+) or *minus* (-).

Examples of valid parameter forms are:

```
10
\AB\                      ( \ = ctrl-L)
.+2
/-5
3,7
3,7:15
\HLT\ +5, /-2             (/ = last line number)
```

1.5 OFF-LINE OPERATIONS

It is sometimes necessary to list a source tape off-line. Paper tapes prepared using the GRI-909 EDITOR may be listed off-line using any ASR Teletypewriter. The off-line listing will look exactly the same as if paper tape were listed on-line with the EDITOR using the LIST command, except line numbers will not appear before each line of printout.

Source tapes for input to one of the system programs may be prepared off-line. The user types source lines on an ASR keyboard according to the conventions outlined in section 1.2. The first block on the source tape must be preceded by a few inches of blank tape.

Note - source tapes may be prepared on any teletype regardless of whether it generates 8-bit or even-parity ASCII code. Although GRI-909 Software recognizes only 8-bit ASCII code internally, text input routines logically OR in the high-order bit before processing.

C H A P T E R   T W O

OPERATING COMMANDS

Commands to the EDITOR are grouped under two general headings

Editing Commands

Paper Tape Commands

An explanation and examples of the commands in the EDITOR command repertoire are given in the following sections.

When the EDITOR is in COMMAND mode and the user types a command or command parameter incorrectly on the keyboard (see 1.4), the EDITOR will respond with an error message consisting of a *question mark* (?) and the command will be ignored.

Similarly, if the user desires to abort a command before execution of the command is initiated (i.e., before the *carriage-return* is typed), the user need only end the command statement with an invalid character. An exception to this rule occurs when the user is typing a source string enclosed in *ctrl-L* 's. The *alt/esc,* in this case, is the only character which will cause an abort.

2.1 EDITING COMMANDS

The following commands permit expanding, altering, deleting, and listing text in the memory buffer.

Each section, containing one command, includes a detailed command description and, in most cases, an example of command operation.

2.1.1 APPEND

Function: ADD INCOMING TEXT FROM ASR KEYBOARD TO CONTENTS OF MEMORY BUFFER

Command: A↲

The EDITOR enters TEXT mode, the line location pointer is moved to the first unused position in the memory buffer (line 1 if the buffer is empty), and the corresponding line number is output to the teletype. User input from the keyboard is appended to the memory buffer until the *alt/esc* is struck or until the buffer becomes full.

EXAMPLE: Assume the source buffer to be empty prior to entering the following command to the teletype. The information printed to the right is comment information for example clarification.

```
*                                APPEND command enters TEXT mode.
A↲                               Line numbers are printed by the
001 AAAAA↲                       EDITOR.  Source text is entered by
002 BBBBB↲                       the user.
003 CCCCC↲
004 DDDDD↲
005 EEEEEE←↲                     Back-arrow deletes preceding character.
006 FFFFF↲
007 HHHHHrubout                  Rubout deletes entire line.
007 GGGGG↲                       Note line no. is repeated.
008 HHHHH↲
009 IIIII↲
010 JJJJJ↲
011 alt/esc                      * indicates return to COMMAND mode.
*
```

The *alt/esc* places the EDITOR back in COMMAND mode. Upon termination of the APPEND command, the line location pointer, denoted by *period* (.) (see section 1.4), is set to the last line in the memory buffer, i.e. line number 10 in the example above.

2.1.2 LIST MEMORY BUFFER

Function: LIST THE CONTENTS OF ENTIRE MEMORY BUFFER

Command: L↲

The contents of the entire memory buffer are listed on the teletype. Line numbers are inserted at the beginning of each line of listing. Upon termination of the LIST command, the line location pointer is set to the beginning of the last line in the memory buffer.

<u>EXAMPLE:</u> Assume the memory buffer to contain the text from the example in 2.1.1.

```
*
L↲
001 AAAAA                                The EDITOR lists the contents of
002 BBBBB                                the memory buffer on the teletype-
003 CCCCC                                writer.
004 DDDDD
005 EEEEE
006 FFFFF
007 GGGGG
008 HHHHH
009 IIIII
010 JJJJJ
*
```

The line location pointer is set to line number 10.

Note that the ← and *rubout* characters corrected the text during input in the previous example.

2.1.3 LIST LINE

Function: LIST LINE (denoted by m)

Command: Lm↲

The contents of the line denoted by parameter m are listed on the teletype. A line number is inserted at the beginning of the line of listing. Upon termination, the line location pointer is set to the beginning of the line listed.

<u>EXAMPLE:</u> Assume the memory buffer to contain the text from the example in 2.1.2.

```
*
L5↲                                      List line 5.
005 EEEEE
*
```

or

```
*
L\EEE\↲        (\ = ctrl-L)              List the first line in the memory buffer
005 EEEEE                                which contains the string "EEE".
*
```

2.1.4 LIST LINES

Function: LIST LINES (denoted by m through n, inclusive)

Command: Lm,n↲

The contents of the line denoted by parameter m through the contents of the line denoted by parameter n are listed on the teletype. A line number is printed at the beginning of each line of listing. Upon termination of the LIST lines command, the line location pointer is set to the line number of the last line listed.

<u>EXAMPLE:</u> Assume the source buffer again to contain the text from the example in 2.1.2.

```
*
L\AA\,\CCC\↲                     List the first line in the memory
001 AAAAA                        buffer containing the string "AA"
002 BBBBB                        through the line containing the
003 CCCCC                        string "CCC".
*
```

or

```
*                                List lines 1 through 3.
L1,3↲
001 AAAAA
002 BBBBB
003 CCCCC
*
```

The line location pointer is set to line 3 of the text in the memory buffer.

2.1.5 GET LINE(S)

Function: GET LINE(S) (denoted by character string)

Command: G*character string*\\↲ (\\ = *ctrl-L*)

The parameter associated with the GET is always a string of characters enclosed in *ctrl-L* 's. All lines in the memory buffer containing this string are listed on the teletype. Upon termination, the line location pointer is set to the beginning of the last line listed.

<u>EXAMPLE:</u> Assume the source buffer to be empty prior to entering the following commands.

```
*
A↲                                  Enter 3 lines.
001 ABB↲
002 BABBB↲
003 BAAB↲
004 alt/esc
*
G\A\↲                               Get all lines containing "A".
001 ABB
002 BABBB
003 BAAB
*
G\BB\↲                              Get all lines containing "BB".
001 ABB
002 BABBB
*
G\BBB\↲                             Get all lines containing "BBB".
002 BABBB
*
```

After the last GET the line location pointer is set to line 2.

2.1.6 DELETE LINE

Function: DELETE LINE (denoted by m)

Command: Dm↲

The contents of the line denoted by parameter m are deleted from the memory buffer. All lines in the buffer following this one (if any) are moved up to fill the vacated space. Upon termination of the DELETE line command, the line location pointer is set to the beginning of the line <u>preceding</u> the one deleted. All lines moved up are re-numbered according to their new relative position in the buffer.

2.1.7 DELETE LINES

Function: DELETE LINES (denoted by m through n, inclusive)

Command: Dm,n↲

The contents of the line denoted by parameter m through the contents of the line denoted by parameter n are deleted from the memory buffer. All lines in the buffer following the last one deleted (if any) are moved up to fill the vacated space. Upon termination of the DELETE lines command, the line location pointer is set to the line preceding the first one deleted.

EXAMPLE: Assume the contents of the memory buffer to be the ten lines of the example in 2.1.2.

```
*
D/-4,9↵                    Delete lines 6 through 9, (/ denotes
*                          the last line, number 10)
L↵                         List to show buffer contents.
001 AAAAA
002 BBBBB
003 CCCCC
004 DDDDD
005 EEEEE
006 JJJJJ
*
```

The original contents of lines 6 through 9 have been deleted from the memory buffer. The characters formerly comprising line 10,'JJJJJ', have been moved to line 6.

Note - Since lines moved up are re-numbered, the successive deletion of non-sequential lines is facilitated by making the deletions in reverse numeric order (i.e., from bottom up).

2.1.8 KILL MEMORY BUFFER

Function: KILL THE CONTENTS OF THE ENTIRE MEMORY BUFFER

Command: K↵

The entire contents of the memory buffer are deleted and the position of the line location pointer is set to the beginning of the buffer (line 1).

EXAMPLE: Assume the source buffer is empty prior to entering the following commands.

```
*
A↵                         Build a four-line memory buffer.
001 AAAAA↵
002 BBBBB↵
003 CCCCC↵
004 DDDDD↵
005 alt/esc
*
K↵                         Kill the buffer contents.
*
L?↵                        List the entire buffer contents.
*
```

The EDITOR replied to the LIST command with a *question mark* (?) since the memory buffer is empty and cannot be listed to the teletype.

2.1.9 INSERT LINE(S)

Function: INSERT LINE(S) BEFORE LINE (denoted by m)

Command: Im↵

The EDITOR enters the TEXT mode and types out the line number corresponding to the parameter m. The user inputs one or more lines (through the keyboard) which are inserted into the memory buffer at this point. The insertion causes lines following those typed in to be moved down and renumbered. The command terminates and COMMAND mode is entered if *alt/esc* is struck or if the memory buffer becomes full. Upon termination, the line location pointer is set to the line number of the last line inserted.

<u>EXAMPLE:</u> Assume the memory buffer is empty prior to entering the following commands.

```
*
A↵                                  Build a three-line memory buffer.
001 AAAAA↵
002 BBBBB↵
003 EEEEE↵
004 alt/esc
*
I\EE\↵                              Insert new lines before the first
003 CCCCC↵                          line containing the character
004 DDDDD↵                          string "EE".
005 alt/esc
*
L↵                                  List the buffer as modified.
001 AAAAA
002 BBBBB
003 CCCCC
004 DDDDD
005 EEEEE
*
```

The INSERT command given above is equivalent to I3↵.

2.1.10 CHANGE LINE

Function: CHANGE LINE (denoted by m)

Command: Cm↲

The contents of the line denoted by m are deleted from the buffer, the EDITOR enters the TEXT mode, and the line number corresponding to m is output to the teletype. The user inputs one or more lines (through the keyboard) which are inserted into the buffer at this point. The command terminates and COMMAND mode is entered if *alt/esc* is struck or if the buffer becomes full. Upon termination, the line location pointer is set to the line number of the last line inserted.

A CHANGE command is equivalent to a DELETE followed by either an INSERT or an APPEND.

<u>EXAMPLE:</u> Assume the buffer to contain the text from the example in 2.1.2.

```
*
C6↲                                   Change line 6.
006 11111↲
007 22222↲
008 alt/esc
*
L↲                                    List new contents.
001 AAAAA
002 BBBBB
003 CCCCC
004 DDDDD
005 EEEEE
006 11111
007 22222
008 GGGGG
009 HHHHH
010 IIIII
011 JJJJJ
*
```

2.1.11 CHANGE LINES

Function: CHANGE LINES (denoted by m through n, inclusive)

Command: Cm,n↲

The contents of the lines denoted by m through n are deleted from the buffer, the EDITOR enters the TEXT mode, and the line number corresponding to m is output to the teletype. The user inputs one or more lines which are inserted at this point. Command termination is as above (2.1.10).

EXAMPLE: Assume the buffer to be as at the end of the previous example.

```
*
C/-3,/-1↵
008 88↵
009 99↵
010 alt/esc
*
C\B\,6↵
002 BBB↵
003 alt/esc
*
L↵                                         List new contents.
001 AAAAA
002 BBB
003 22222
004 88
005 99
006 JJJJJ
*
```

2.1.12 MOVE LINE

Function: MOVE LINE (denoted by m) BY INSERTING BEFORE LINE (denoted by p)

Command: Mm:p↵

The contents of the line denoted by parameter m are inserted at a position in the memory buffer denoted by parameter p. The line is deleted from its original position.

EXAMPLE: Assume the memory buffer is empty prior to entering the following commands.

```
*
A↵                                         Build a six-line memory buffer.
001 AAAAA↵
002 BBBBB↵
003 CCCCC↵
004 EEEEE↵
005 FFFFF↵
006 DDDDD↵
007 alt/esc
*
M\DD\:4↵                                   MOVE the first line containing the
                                           character string "DD" to before line
                                           number 4.
```

```
*
L↲                                          List the buffer contents.
001 AAAAA
002 BBBBB
003 CCCCC
004 DDDDD
005 EEEEE
006 FFFFF
*
```

2.1.13 MOVE LINES

Function: MOVE LINES (denoted by m through n, inclusive) BY INSERTING
BEFORE LINE (denoted by p)

Command: Mm,n:p↲

The contents of the line denoted by parameter m through the contents of the line denoted by parameter n are inserted at a position in the memory buffer denoted by parameter p. The lines are deleted from their original positions.

EXAMPLE: Assume the buffer contents to be as at the end of the previous example (2.1.12).

```
*
M1,.+2:/↲                                   Move 3 lines to before last line.
*
L↲                                          List the buffer contents.
001 DDDDD
002 EEEEE
003 AAAAA
004 BBBBB
005 CCCCC
006 FFFFF
*
```

Note that when *period* (.) is encountered in the second parameter, it takes on the value denoted by the first parameter.

EXAMPLE: Assume the buffer contents as above. To move lines 1 and 2 to the end of the buffer

```
*
M3,/:1↲
*
L↲
001 AAAAA
002 BBBBB
003 CCCCC
004 FFFFF
005 DDDDD
006 EEEEE
```

Note this was accomplished by moving the last half of the buffer before the beginning.

2.1.14 EXCHANGE

FUNCTION: EXCHANGE, in the lines m through n inclusive, the new character string, a, for the old character string starting at b and ending at c.

COMMAND: `Xm,n: \a \,b,c↲`

m and n are parameters that define the lines to be operated on. The first field, a (delimited by *ctrl-L* 's), is always a character string. A null string (no characters) may be denoted by two consecutive *ctrl-L* 's with no ASC characters in between, e.g.\\ denotes a null string. b and c may be either character strings (delimited by *ctrl-L*'s) or integers and describe the old string in the line(s) to be replaced by the character string, a, according to the following conventions:

a) If b is a character string, the old string starts at the beginning of b and ends (if no c field) at the end of b.

b) If b is an integer, the old string starts and ends (if no c field) at the bth character in the line(s). ∅ means in front of the first character in the line.

c) If c is a character string, the old string end is further extended to the end of c. The search for the character string c begins with the character following the end of b.

d) If c is an integer, it defines the length of the old string by the number of consecutive characters specified beginning at the beginning of b.

Note: The command allows any character to be used to separate the fields a, b, and c which are separated by commas in the command format above.

b and c can each be a maximum of 15 characters, not including delimiters. Short forms of the command can be used by leaving out unnecessary parameters as shown by the following format variations; e.g.

```
Xm,n:\a\,b↲
```

would in lines m and n inclusive, exchange the new string a for the old string b. b may be a character string delimited by *ctrl-L* 's or an integer (i.e. insert the new string a in place of the b^{th} character). Or,

```
Xm:\a\,b↲
```

would in line m, exchange string a for the old string defined by parameter b.

EXAMPLES:

Suppose the text contains:

```
009 MSI 0,P1↲
```

then,

`X9:\COUNT:\,0↲` changes line 9 to:

```
009 COUNT:MSI 0,P1↲
```

The carriage return may be used as a character inside the delimited string. NOTE: It is not possible to remove this character from the text buffer. The assemblers require this character as a line delimiter and so %STE will detect an error if the X command attempts to delete it. Hence, if a carriage return appears in field b or c, it must also be part of the character string a.

EXAMPLE: Suppose line 5 of the text buffer contains

```
005 MSI 0,P1↵
```

then

```
X5:\;INCR CTR↵
\,\↵
\  changes line 5 to
```

```
005 MSI 0,P1;INCR CTR↵
```

EXAMPLE: Suppose line 10 of the text buffer contains

```
010 ABCDEFGABCXYZ↵
```

then

```
X10:\**\,\D\,\AB\↵
010 ABC**CXYZ↵
```

EXAMPLE: Suppose line 21 of the text buffer contains

```
021 C AX P1;---VALUE↵
```

then

```
X21:\\,9,2↵    changes line 21 to
021 C AX P1;-VALUE↵
```

EXAMPLE: Suppose line 3 of the text buffer contains

```
003 ABCDEFGABCXYZ↵
```

then

```
X3:\**\,\CX\,3↵
003 ABCDEFGAB**Z↵
```

EXAMPLE: Suppose line 7 of the text buffer contains

```
007 LOOP:AY TO AX↵
```

then

```
X7:\ P1 \,8↵  changes line 7 to
007 LOOP:AY P1 TO AX↵
```

The most often useful form of the command is when field b is a character string enclosed in *ctrl-L*'s and there is no field c. In this case, the first occurrance of the character string b is replaced by the character string a in the line(s) specified, e.g. the last example above could have been done as

```
X7:\ P1 \,\ \↵  also changes line 7 to
007 LOOP:AY P1 TO AX↵
```

Note the space on either side of the P1 enclosed in *ctrl-L*'s in the last two examples. Both b fields specify deleting a space which is necessary to replace.

One can also use this command to change all references to a label in a buffer, e.g. X1,/: \LABL\,\LABEL\↵ would replace the first (if any) occurrance of LABEL in each line of the text buffer with LABL. (It is a good idea to issue a G LABEL\ first to see if this is precisely what you want to do.)

A line can be truncated by replacing the character string b by a carriage return, e.g. for the last line 7 above:

```
X7:\↵
\,\ TO\↵
```

changes line 7 to 007 LOOP:AY P1↵

2.2 PAPER TAPE COMMANDS

All EDITOR commands so far have been concerned with filling and manipulating the contents of a memory buffer using only the teletype keyboard as the input device. The following sections explain how the contents of the memory buffer are read from or punched onto paper tape.

In general, the procedure for punching out a source tape depends largely on the user's particular requirements. The EDITOR provides the user with commands which help to satisfy these requirements. These commands enable the user to copy entire tapes or portions of tapes, to punch lines to tape from the memory buffer, to punch the entire contents of the memory buffer to tape, to punch blank tape (leader and trailer), and to punch a paper tape into distinct and manageable blocks.

In this last regard, it is often convenient to punch a paper tape into blocks or sections. In other words, the user may wish to separate certain portions of a paper tape (subroutines, etc.) with a special block mark character which the EDITOR provides (the *alt/esc* character). On input, when either the FAST or BASIC Assembler Program encounters the block mark, an internal line location pointer is reinitialized and the next sequential line of assembler listing printout will contain 001 as its line number. This insures consistency between EDITOR and assembler listings.

2.2.1 PUNCH MEMORY BUFFER

Function: PUNCH MEMORY BUFFER TO PAPER TAPE

Command: P↵

The entire contents of the memory buffer are punched onto paper tape through the appropriate output device. The contents of the memory buffer are not deleted and remain available. The appropriate output device is selected by setting the Switch Register on the user console; bit 14 in the UP position indicates output on the high-speed punch, the DOWN position indicates teletype output.

Upon termination of this command, the line location pointer is set to the last line punched from the text buffer.

Note - In the paper tape command examples, items enclosed in quotes are manual operations performed by the user.

<u>EXAMPLE:</u> Assume the source buffer to contain the text from the example in 2.1.2. Assume also that bit 14 of the Switch Register is set in the DOWN position (indicating teletype output) and that the teletype punch is in the OFF condition.

```
*
P↵ "set low speed punch in ON condition"↵
AAAAA
BBBBB
CCCCC
DDDDD
EEEEE
FFFFF
GGGGG
HHHHH
IIIII
JJJJJ "set low speed punch in OFF condition"↵
*
```

The entire contents of the memory buffer are listed on the teletype as they are output to the low speed punch. This is a function of the ASR device. The *carriage-return* (or any character on the teletype keyboard) must be struck immediately after the low speed punch is set in the ON condition and once again after the low speed punch is set OFF. This safety procedure insures that the punch is off while the user is typing commands on the teletype keyboard (if the low speed punch were set ON, the commands as well as the input text would be punched on paper tape).

<u>EXAMPLE:</u> Assume the memory buffer contains the text inserted in section 2.1.2. Assume output is to the high-speed punch (bit 14 of the Switch Register is set in the UP position).

```
*
P↵
*
```

When the high-speed punch is the output device, the contents of the memory buffer are not listed on the teletype and the safety procedure described above does not apply.

2.2.2 PUNCH LINE

Function: PUNCH LINE (denoted by m)

Command: Pm↵

The contents of the line denoted by the parameter m are punched onto paper tape through the appropriate output device. The contents of the line are not deleted from the memory buffer and remain available to the user. The appropriate output device is selected as in the PUNCH buffer command in 2.2.1.

2.2.3 PUNCH LINES

Function: PUNCH LINES (denoted by m through n, inclusive)

Command: Pm,n↵

The contents of the line denoted by the parameter m through the contents of the line denoted by the parameter n are punched onto paper tape through the appropriate output device. The contents of the lines are not deleted from the memory buffer and remain available to the user. The appropriate output device is selected as in the PUNCH buffer command in 2.2.1. Upon termination of this command, the line location pointer is set to the last line punched.

2.2.4 BLOCK

Function: PUNCH BLOCK MARK

Command: B↵

A block mark *alt/esc* plus four inches of blank tape is punched onto paper tape through the selected output device. This mark defines the end of a block of text on tape. The output device is selected as in the PUNCH commands.

<u>EXAMPLE:</u> Assume bit 14 is in the UP position (high-speed punch) and that the buffer is not empty.

```
*
P↵                                    Punch the buffer on the high-speed punch.
*
B↵                                    Punch the block mark and blank tape.
*
```

2.2.5 PUNCH BLANK TAPE

Function: PUNCH BLANK TAPE (length, in inches, is denoted by q)

Command: Tq↵

q inches of blank paper tape are punched through the selected output device. The default value of q is 10 (i.e., when q is unspecified, 10 inches of blank tape are punched). The output device is selected as in the PUNCH commands.

<u>EXAMPLE:</u> Assume bit 14 of the Switch Register is in the DOWN position (low-speed teletype output).

```
*
T5↵ "set low-speed punch in ON condition"↵
    "set low-speed punch in OFF position when tape is punched"↵
*
```

Once again the ↵ (or any character on the keyboard) must be struck immediately after the low-speed punch is set in the ON condition and also after the low-speed punch is set OFF (in order to avoid punching commands along with text).

2.2.6 READ BLOCK FROM PAPER TAPE

Function: READ BLOCK FROM PAPER TAPE

Command: R↵

A block of text is read from the appropriate input device. Text is read and appended to the memory buffer until a block mark is encountered, the memory buffer becomes full, or the end of tape is reached. The input device is selected by setting the Switch Register on the user console prior to the READ command; bit 15 in the UP position indicates high-speed reader input, the DOWN position indicates low-speed teletype input.

Upon termination of the READ BLOCK command, the line location pointer is set to the first line of text in the memory buffer.

<u>EXAMPLE:</u> Assume the memory buffer contains the text from the example in 2.1.2. Assume also that the paper tape in the input device contains three lines of text, followed by a block mark, followed by three more lines of text. Bit 15 of the Switch Register is in the DOWN position indicating low-speed input.

```
* "set the teletype reader in start position"
R↵                                      Read one block from tape.
*
L.↵                                     List the last line read.
013 PAPER TAPE
*
```

```
L⤶                                        List the entire buffer.

001 AAAAA
002 BBBBB
003 CCCCC
004 DDDDD
005 EEEEE
006 FFFFF
007 GGGGG
008 HHHHH
009 IIIII
010 JJJJJ
011 THESE THREE LINES WERE CONTAINED IN ONE BLOCK
012 AND WERE READ FROM
013 PAPER TAPE
*
```

Note that only the three lines of text preceding the block mark character were read from paper tape. A second READ command would now read in and append the second block to the memory buffer.

2.2.7 READ LINES FROM PAPER TAPE

Function: READ LINES (the number of lines is denoted by q) FROM PAPER TAPE

Command: Rq⤶

q lines of text are read from paper tape through the selected input device. Text is read and appended to the memory buffer until q lines are read, a block mark is encountered, the memory buffer becomes full, or end of tape is reached. The input device is selected as in the READ block command. Upon termination of the READ lines command, the line location pointer is set to the first line of text in the memory buffer.

2.2.8 SKIP BLOCK

Function: SKIP A BLOCK ON PAPER TAPE

Command: S⤶

The paper tape is advanced through the input device until a block mark is encountered (i.e., a block is skipped) or end of tape is reached. No text is added to the memory buffer. The input device is selected as in the READ block command.

2.2.9 SKIP LINES

Function: SKIP LINES ON PAPER TAPE

Command: Sq↲

The paper tape is advanced through the input device until q lines are skipped, a block mark is encountered, or end of tape is reached. No text is added to the memory buffer. The input device is selected as in the READ block command.

2.2.10 COPY PAPER TAPE

Function: COPY PAPER TAPE

Command: N↲

The sequence PUNCH, BLOCK, KILL, and READ is performed until the input tape is entirely copied to the output tape. If the text buffer is empty when this command is entered, the first "PUNCH, BLOCK" sequence above will be skipped. Upon termination of this command, the memory buffer is left empty.

2.2.11 COPY BLOCKS FROM PAPER TAPE

Function: COPY BLOCKS (the number of blocks are denoted by q) FROM PAPER TAPE

Command: Nq↲

The sequence PUNCH, BLOCK, KILL, and READ is performed until either q blocks have been copied from the input device to the output device or until the end of tape is reached. q specifies the number of blocks which will be punched.

<u>EXAMPLE:</u> Suppose a user desires to skip over the first two blocks on an input tape, copy the third block, skip over the fourth block, and then copy the remaining input tape. The following sequence will accomplish this. Assume the input device is low-speed teletype and the output device is high-speed punch.

```
* "set the reader in start position"
S↲                              Skip first block.
*
S↲                              Skip second block.
*
N1↲                             Copy third block.  (Also reads in the 4th block)
*
```

```
K↵                                         Kill fourth block.
*
N↵                                         Copy remaining input tape.
*
```

CHAPTER THREE
USAGE NOTES

This chapter presents further operating features of the EDITOR and constraints imposed upon text input.

3.1 CHARACTER SET

GRI-909 systems programs operate on full 8-bit ASCII characters when processing text. The text input routines logically OR in the high order bit when reading characters - the characters read may, therefore, be in 8-bit, 7-bit, even parity or odd parity code. All characters except *back-arrow, rubout,* and *alt/esc* are valid for text input. (C.f. note at end of section 3.3.)

A complete tabulation of teletype codes and the corresponding characters is contained in Appendix F, "GRI-909 System Reference Manual". The character codes which have special meaning to the EDITOR are:

215	*(carriage-return)* - Terminates a source text line (see section 1.2.1).
212	*(line-feed)* - ignored when encountered after a *carriage-return.* Generated on output after a 215 is either printed or punched.
337	*(back-arrow)* - Used for error recovery within a source text line (1.2.2).

377	*(rubout)* - Also used for error recovery within a source text line (1.2.2).
375	*(alt-mode)* - Terminates a source text block (1.2.3). Also causes EDITOR to pass from TEXT mode to COMMAND mode (1.3.2).
233	*(esc)* - Treated identically to the 375 code.

3.2 ERROR MESSAGES

3.2.1 COMMAND ERRORS

If the EDITOR is in the COMMAND mode and the user types a command or a command parameter (1.4) incorrectly, the EDITOR responds with a *question mark (?)* and ignores the command. Examples of errors detected are:

a. the command character itself is invalid,
b. a list, punch or edit command has been issued <u>and</u> the buffer is empty,
c. a parameter contains an invalid character (such as a letter in a numeric field),
d. a line parameter does not address a line in the buffer - the value of the parameter is negative or is larger than the number of the last line in the buffer,
e. in the form m,n (denoting line m through n) the value of n is less than the value of m, or
f. in the form m,n:p for the MOVE command the value of p does not lie outside the group m,n.
g. no match could be found for a search string enclosed in *ctrl-L*'s.

3.2.2 LINE TRUNCATION

A source text line may contain up to 80 characters followed by a *carriage-return* (1.2.1). A longer line read from paper tape is truncated to 80 characters plus a *carriage-return* and "TR" is printed by the EDITOR upon return to the COMMAND mode. The "TR" indicates that at least one line read was so truncated. If an attempt is made to enter a line longer than

80 characters from the keyboard, a *carriage-return* is issued and the EDITOR proceeds as if it had been typed.

3.2.3 FULL BUFFER

When in the TEXT mode and before an input line is accepted (from paper tape or via the keyboard) the currently available buffer space is checked to see if a maximum length line could be contained therein. If not, the EDITOR prints the message "FULL" and returns to the COMMAND mode. Note that this check applies also to any editing command that <u>might</u> increase the number of characters currently in the buffer.

3.3 SEARCH STRINGS

Any parameter m, n or p in the forms

```
        m
        m,n
        m:p
    or  m,n:p
```

may consist of a series of operands separated by operators, + or - (1.4). Such an expression has the conventional algebraic meaning. An operand may be a search string - a sequence of characters enclosed in *ctrl-L* 's (\ ). The value of a search string is determined via a search performed by the EDITOR - it is the number (in the buffer) of the first line encountered during the search that contains the given sequence of characters.

If a search string operand is encountered when evaluating the parameter m or the parameter p, the EDITOR search is begun at the first line in the memory buffer. During evaluation of n, this search is begun at the next line after the one addressed by the value of m. In either case, the search is begun immediately after the rightmost *ctrl-L* of the operand is typed. If the given character sequence is located, the EDITOR rings the teletype bell indicating that further parameter input may continue. If the search was unsuccessful, the EDITOR responds with a *question mark* and returns to the COMMAND mode - the partial command is ignored.

Since the line terminator *(carriage-return)* is stored at the end of each line in the buffer, it is possible to include it as the last character of a search string and thereby address a line that ends with a given sequence of characters. Since the EDITOR echoes a *carriage-return* with itself and a *line-feed,* the rightmost quote of the operand will appear on the next line of printout.

NOTE: Since the character *ctrl-L* (\ ) has special meaning regarding the definition of character strings, it may not be searched for in text.

3.3.1 CORRECTING ERRORS WITHIN SEARCH PARAMETERS

If an erroneous character has been typed in the string following the opening *ctrl-L,* it may be corrected if the closing *ctrl-L* has not yet been typed. *Rubout* will delete all characters in the string following the opening *ctrl-L* and the entire string may be re-typed. The *rubout* is echoed but causes no physical spacing on the teletype, so although the corrected string follows the erroneous string on the typed copy, the internal buffer contains only the characters typed after the rubout. *Back-arrow (←)* may also be used in the same way as when editing text input - successive *back-arrows* deleting characters one by one from right to left. However, the *ctrl-L* itself cannot be deleted to get out of this mode. This can only be accomplished by typing the *alt/esc* key.

APPENDIX A

OPERATING INSTRUCTIONS

I. Load the text editor with the Absolute Loader.

II. Transmit "0" to SC, or, if restarting, transmit "2" to SC to preserve the contents of the buffer.

III. Set console switches as follows:

Bit 15 selects text input device
Bit 14 selects text output device

UP = High-speed
DOWN = Low-speed

IV. Press START

The text editor will respond with an asterisk (*) indicating that it is in the command mode, awaiting a command.

NOTE: A command such as GET, PUNCH or LIST which causes many lines to be output during execution may be aborted before completion by depressing the space bar on the keyboard. The EDITOR returns to the COMMAND mode after the next line is printed or punched. The line location counter will point to the last line processed. In the case of PUNCH, it is a good idea to issue a 'L.' to get the line number of the last line punched so that if it is desired to punch the rest of the buffer at some later time, it can be done by issuing a 'Pn,/' where n is the number of the line following the one listed by the 'L.'.

APPENDIX B

COMMAND SUMMARY

COMMAND[1,2]	FUNCTION	I/O DEVICE[3]	REFERENCE
A	Append text to buffer	K	2.1.1
B	Punch block mark	P	2.2.4
Cm	Change line m	K	2.1.10
Cm,n	Change lines m through n	K	2.1.11
Dm	Delete line m	-	2.1.6
Dm,n	Delete lines m through n	-	2.1.7
G\cc...c\	Get (and list) all lines containing the string cc...c	T	2.1.5
Im	Insert line(s) before line m	K	2.1.9
K	Kill (delete) buffer contents	-	2.1.8
L	List entire buffer contents	T	2.1.2
Lm	List line m	T	2.1.3
Lm,n	List lines m through n	T	2.1.4
Mm:p	Move line m by inserting before line p	-	2.1.12
Mm,n:p	Move lines m through n by inserting before line p	-	2.1.13
N	Perform sequence R,P,B,K until input tape exhausted	R,P	2.2.10
Nq	Perform sequence R,P,B,K q times	R,P	2.2.11
P	Punch entire buffer contents	P	2.2.1
Pm	Punch line m	P	2.2.2
Pm,n	Punch lines m through n	P	2.2.3
R	Read block	R	2.2.6
Rq	Read q lines	R	2.2.7
S	Skip block	R	2.2.8
Sq	Skip q lines	R	2.2.9
T	Punch 10" of blank tape	P	2.2.5
Tq	Punch q" of blank tape	P	2.2.5
Xm,n:\a\,b,c	Exchange characters b thru c for a in lines m thru n		2.2.14

NOTES:

1. Commands are terminated by a *carriage-return*, ⤶.
2. Parameters m,n,p represent line numbers, q represents a count-- (see section 1.4.), a represents a character string, b & c represent integers or character strings enclosed in *ctrl-L* 's.
3. The device concerned with text input, output (or skip in S commands) is identified as follows:
   - K - Keyboard
   - P - Selected punch
   - R - Selected reader
   - T - Teleprinter

Bits 14 and 15 of the console switch register select the punch and reader respectively: UP selects high speed, DOWN selects teletype.

APPENDIX C

EDITOR ERROR MESSAGES

1.	?	search string not found, illegal parameter, illegal command, command requires text and text buffer is empty.
2.	TR	line truncated to 80 characters and a carriage-return.
3.	FULL	text buffer full.

GRI Computer Corporation

320 NEEDHAM STREET, NEWTON, MASSACHUSETTS 02164

TEL: (617) 969-0800