Model 210 users enter the Editor from the MDS Monitor, by typing the Command, GA800, in response to a dot prompt.

Chapter 3

THE TEXT EDITOR

The Intellec text editor lets you create and edit files of text. Examples of text files are source programs written in assembly language or PL/M, tables to be used in a computation, or project documentation. You will probably use the text editor most commonly to prepare and correct programs.

Before we delve into some of the details, here is an overview of how the text editor is used. After the equipment has been turned on you initiate execution of the editor. You type in your program, making changes and corrections as you go; when you are finished creating the file you store it on diskette (or punch it onto paper tape in the Model 210). The text file can now be used however you wish: you can assemble or compile it, if it is a program, or you can ask for a listing on the console or line printer if you have one, or you can call on the text editor again and make changes in the file.

THE POINTER

Once you have entered some text, all further operations depend on a *pointer* to the text. Since the text editor is character-oriented, the pointer locates a character that is to be acted upon. The pointer may be positioned before the first character of the text, between two characters, or after the last character. The pointer is never positioned directly *at* a particular character, but always *between* two characters. When text is entered it is placed at a point immediately following the pointer. As each character is entered, the pointer is moved to a position immediately following that character.

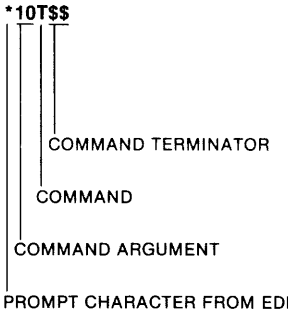
You can move the pointer to any position in your text, using commands that are illustrated below. Pointer movement may be in terms of characters or complete lines. Considering text in terms of lines is convenient because most text is divided into lines.

COMMANDS AND COMMAND STRINGS

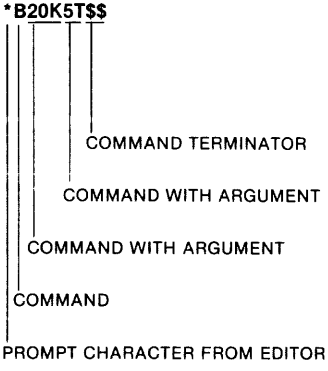
Each editor *command* consists of a single letter. Certain commands take *arguments*. Commands may be entered one at a time or may be combined into *command strings*. The text editor signals its readiness to accept commands by printing a prompting asterisk in the leftmost column of the system console device. Command strings must be terminated with a pair of ALT MODE (alternate mode) or ESC (escape) characters (depending on the type of the console device.)

EXAMPLES OF COMMANDS

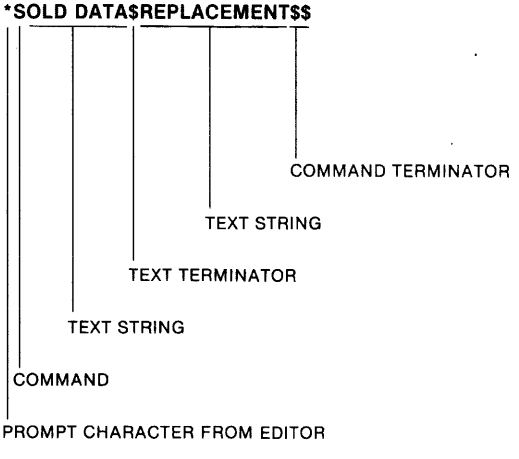
The following three examples show first a typical command, then a command string, and finally a command string combined with a text string. In each case the dollar signs show where the ALT MODE or ESC key was pressed.



This command prints ten lines of text on the system console device.



This command string moves the pointer to the beginning of the text, deletes 20 lines of text, and prints the following five lines on the system console device. The command terminator is placed at the end of the command string; the individual commands do not need terminators.



This command string searches for the string OLD DATA in the text. When found, it is deleted and the text string REPLACEMENT is used as a replacement. The single dollar sign shows where the ESC key was used as the text terminator for OLD DATA; the two dollar signs represent two ESC characters used as the command terminator.

EDITOR COMMANDS

Editor commands are provided to perform four groups of operations: text input/output, pointer manipulation, text modification, and string search and substitution.

B — BEGINNING OF TEXT

The B command moves the pointer to the beginning of the text. It is useful in several ways, such as:

- Defining a starting point when the entire text is to be typed out;
- Moving the pointer to the start of the text prior to starting a search for a selected text string;
- Inserting text at the beginning of the text, ahead of text already there.

Z — END OF TEXT

The Z command positions the pointer immediately following the last character in the text. This command is used mainly to position the pointer so that new text can be inserted after the end of old text.

I — INSERT TEXT

The I command is used to enter text from the system console device, beginning where the pointer is positioned.

Entering a carriage return character causes a line feed character to be generated by the text editor and appended to the carriage return character. Thus, the entry of a carriage return character causes a *pair* of characters to be stored.

After recognizing the letter I as a command, the text editor accepts all subsequent input as text (including carriage returns and appended line feed characters) until ALT MODE, ESC, or Control and C keys are pressed. The ALT MODE or ESC character specifies the termination of the text string; the Control C character cancels the command (and inserts no text).

T — TYPE OUT TEXT

The T command types as many lines of text as the value of the argument written in front of it, as follows:

*nnnnnT$$	nnnnn represents any decimal number from – 65,535 to + 65,534

If the argument is positive, typing starts at the pointer; the argument value specifies the number of lines to be typed. If the argument is negative, typing begins at the pointer minus the number of lines specified by the argument value; typing continues until the pointer is reached. If the argument value is zero, typing starts at the beginning of the current line; all characters up to the pointer are typed. If no argument value is specified, a default value of 1 is assumed.

EXAMPLES OF EDITING USING B, Z, I, AND T COMMANDS

Suppose you have entered text using the I command, and now wish to type out the entire text. The following command string may be used.

*B500T$$	The B command moves the pointer to the beginning of the text. The 500T command types out 500 lines of text. The argument 500 is assumed to be larger than the number of lines of text; this being the case, the T command is terminated when the end of the text is reached, even though the full count has not been reached. The dollar signs stand for the ALT MODE or ESC character.

Suppose you are entering a source program, using the I command, and have already entered a large number of text lines. For some reason the I command is terminated and the pointer is moved to some other location. When you wish

to resume entering the source file, you simply move the pointer to the end of the text and use the I command again. A typical command string will be as follows:

*ZITEXT STRING----------$$	The new text will be inserted following the old text.

If you are entering text and wish to see the previous five lines, without moving the pointer, the following command may be used:

*-5T$$	The five lines before the current line (the one within which the pointer is located) are printed on the system console device. The current line is printed up to the position of the pointer.

The following command may be used to print the current line of text, without moving the pointer:

*0TT$$	The 0T part of the command prints from the beginning of the line up to the pointer. The following T command prints from the pointer to the end of the line.

L — LINE

The L command moves the pointer as many lines as the value of the argument written in front of it, as follows:

*nnnnnL$$	nnnnn represents any decimal number from − 65,535 to + 65,534

The line feed character serves as the delimiter between lines. A line of text is defined as any text string having a line feed character as its last character. (Recall that a line feed is automatically generated by the text editor when we press carriage return.)

When the argument value is 1 or no argument is used (default value of 1 assumed), the pointer is advanced to the start of the next line. A positive argument value advances the pointer to the beginning of the nth line following the current line. A negative argument value moves the pointer back to the beginning of the nth line preceding the current line. When the argument value is − 1, or just − , the pointer is moved back to the beginning of the line preceding the current line. Finally, if the argument is 0, the pointer is moved to the beginning of the current line.

The command string 0LT finds frequent use. It moves the pointer to the beginning of the current line, and then types the entire line.

K — KILL

The K command deletes as many lines of text as the value of an argument that is placed in front of it, as follows:

*nnnnnK$$	nnnnn represents any decimal number from − 65,535 to + 65,534

A negative argument deletes lines prior to the line containing the pointer. A positive argument deletes lines following the line containing the pointer. If the argument is zero, the characters from the start of the current line up to the buffer pointer are deleted. If the argument is 1 (actual or default), the characters from the pointer, up to and including the line feed character which is used to terminate the line, are deleted.

A — APPEND

The A command reads text from diskette or paper tape into the area in Intellec memory where it is processed. It reads at most 50 lines of text, so it must be issued several times for large files.

F — FIND TEXT STRING

The F command searches for a text string of up to 16 characters. Its format is:

*FXXXXXXXX$ where XXXXXXXX is any text string of up to 16 characters.

The search begins at the pointer, and terminates either upon finding the first occurrence of the string or upon reaching the end of the text without finding a match. If a match is found, the pointer is positioned after the end of the matching string. If a match is not found, the following message is printed:

CANNOT FIND "XXXXXXXX"

BREAK

The string searched for must be terminated in the command with an ALT MODE or ESC character. Other commands may follow in the same string.

S — SUBSTITUTE TEXT STRING

The heavily used S command combines the actions of the Find command with a substitution if a match is found. The format:

*SOLD STRING$REPLACEMENT STRING$

If a match with OLD STRING is found, it is replaced with REPLACEMENT STRING, and the pointer is positioned after the end of the replacement. Each of the strings must be terminated by an ALT MODE or ESC. If no replacement string is included, the old string is simply deleted.

E — EXIT AFTER WRITING TEXT TO OUTPUT FILE

The E command writes text from Intellec memory to diskette or punches it into paper tape for later use. It is ordinarily used at the end of all text editing sessions.

Examples of these commands are found in the sample text editing session that follows.

A SAMPLE TEXT EDITING SESSION

To demonstrate some of the features of the text editor in operation, let us enter a small program and then make some changes to it. The dialog of the session is shown on the facing page with explanatory comments keyed to bold face numbers in the margin.

1. I initialize the system by turning on power and pressing the Reset key on the Intellec front panel. The system responds with the ISIS identification and the ISIS hyphen prompt.

2. I enter the ISIS EDIT command, specifying a file on diskette unit 1 named ONESEC and having an extension of SRC (for source). (Model 210 users enter the editor from the monitor, by typing the command GA800 in response to a dot prompt.)

3. The text editor identifies itself and notes that since there is no file with this name on diskette 1 this is a new file.

4. The editor prompts with an asterisk. Using the I (Insert) command, I enter a simple assembly language program (which will be taken up again in the section on the assembler). The I command continues until I hit ESC twice, as shown by the two dollar signs after the END, which is identifed as step 7.

   In entering the program text I make free use of the tab feature: any time I simultaneously press Control and the letter I, the system responds as a typewriter would to the tab key, with automatic tab stops every eight positions.

5. I type DEALY where I meant DELAY. Noticing the mistake before going on, I press the rubout key three times; the three wrong characters are echoed back as they are erased. I then type the correct characters. To be sure I have the made the correction properly, before hitting carriage return to enter the line I press the Control key and R together, which repeats the line as corrected. Since it appears to be correct, I press carriage return and continue entering the program.

6. I get a line so badly messed up that I decide to start over. Pressing Control and X causes the entire line to be erased; the crosshatch (#) indicates that this was done.

7. I press ESC twice, once to terminate the input string and a second time to terminate the I command.

8. Now there are errors to correct. Using the B command I move the pointer to the beginning of the text, then use the S (substitute) command to correct the spelling of ASSEMBLY. The 0LT combination prints the modified line, which is now correct.

9. I use the F (find) command to locate the label L2, which I entered without the colon. The T (type) command types from where the pointer is positioned after the F; this is to assure myself that I am where I want to be.

10. I insert the colon, then type the entire line.

11. I notice that I have entered an instruction twice, so I use the F to find it.

12. The K (kill) command removes the entire line; to be doubly sure that I removed what I wanted to, I type the three lines before and the three lines after the current position of the pointer.

13. I notice that there is an instruction missing in the subroutine; the missing line should be immediately after the one having the label L3, which means that the pointer must be positioned at the start of the following line, so I use the F command to find the operation code of the following instruction. But there was another JNZ before the one I wanted.

14. I use the L (line) command to move the pointer past the JNZ that I don't want, and use F again. This time the desired instruction is found.

```
1  ISIS-II, V2.2
2  -EDIT :F1:ONESEC.SRC

3  ISIS-II TEXT EDITOR, V1.6
   NEW FILE
4  *I      ; AN ASSEMLBY LANGUAGE PROGRAM TO SEND THE LETTER 'X'
           ; TO THE CONSOLE OUTPUT DEVICE, ONCE EACH SECOND
           ;
   CO      EQU     0F809H  ; PROVIDE ADDRESS OF CO ROUTINE IN MONITOR
           CSEG            ; MAKE THE SEGMENT RELOCATABLE
           STKLN   2       ; SET STACK LENGTH
   START:  LXI     SP,STACK ; INITIALIZE STACK POINTER
   L1:     MVI     D,50    ; WILL CALL DELAY SUBROUTINE 50 TIMES
   L2      MVI     A,200   ; SUBROUTINE PARAMETER
5          CALL    DEALYYLALAY     ; INVOKE SUBROUTINE
           CALL    DELAY   ; INVOKE SUBROUTINE
           DCR     D
           DCR     D
           JNZ     L2      ; 50 TIMES AROUND THIS LOOP = 1 SEC
           ;
           MVI     C,'X'   ; SEND 'X' TO CONSOLE OUTPUT DEVICE
6  CALL    CO      ; MONTIRO CONSOLE#
           CALL    CO      ; MONITOR CONSOLE OUTPUT ROUTINE
           JMP     L1      ; AROIUND THE LOOP INDEFINITELY
           ;
           ; THE DELAY SUBROUTINE
           ;
   DELAY:  MVI     B,12
   L3:     MOV     C,B
           JNZ     L4
           DCR     A
           JNZ     L3
           RET
           ;
           END     START   ; PROGRAM EXECUTION BEGINS WITH SYMBOL 'START'
7  $$
8  *BSLBY$BLY$0LT$$
           ; AN ASSEMBLY LANGUAGE PROGRAM TO SEND THE LETTER 'X'
9  *FL2$T$$
           MVI     A,200   ; SUBROUTINE PARAMETER
10 *I:$0LT$$
   L2:     MVI     A,200   ; SUBROUTINE PARAMETER
11 *FDCR$0LT$$
           DCR     D
12 *K-3T3T$$
   L1:     MVI     D,50    ; WILL CALL DELAY SUBROUTINE 50 TIMES
   L2:     MVI     A,200   ; SUBROUTINE PARAMETER
           CALL    DELAY   ; INVOKE SUBROUTINE
           DCR     D
           JNZ     L2      ; 50 TIMES AROUND THIS LOOP = 1 SEC
           ;
13 *FJNZ$0LT$$
           JNZ     L2      ; 50 TIMES AROUND THIS LOOP = 1 SEC
14 *LFJNZ$0LT$$
           JNZ     L4
```

15. I insert the entire line, including a carriage return, then hit ESC twice.

16. Now I notice an error earlier in the program. I could use L with a negative argument to back up, but the program is short enough that there is no time penalty in simply going back to the beginning and then using an S. (I am reasonably sure that the combination AROIU does not occur elsewhere in the program.)

17. Now I move the pointer to the beginning again and ask for 50 lines to be typed. I don't really know how many lines there are, but certainly less than 50, so I get the entire program.

18. All seems to be in order, so I use the E (exit) command to store the program on diskette (under the name used with the ISIS EDIT command at the beginning), and return to ISIS. (On the Model 210, a paper tape is punched by the E command.)

Why not try it yourself?

Here is a checklist of things you will need to do.

If you have a Model 220 or Model 230:

1. Turn on the Intellec components. Insert an ISIS system diskette in drive 0 and a blank diskette in drive 1.

2. Press the Reset key on the Intellec console and release it. After a brief interval the message

   ISIS-II, Vx.y

   will be produced at the console, where x.y will be numbers indicating the Version number of your ISIS system. (New versions of most programs are issued from time to time.) ISIS will then produce a dash, telling you it is ready to accept a command. Only ISIS prompts with a dash, so any time you see a dash prompt you know you are dealing with ISIS, not the monitor, text editor, ICE, or the Library Manager, which use different prompt characters.

3. Type in the command

   – FORMAT MYDISK.DDM

   Actually you may use any combination of six or fewer characters before the dot and any combination of three or fewer after. What comes after the dot might be your initials or the date or anything else you please.

4. Proceed with the operations shown at the beginning of the text editing session.

5. Save your work on diskette, since the program will be used in Chapter 5.

If you have a Model 210:

1. Turn on the Intellec components.

2. After a brief interval a dot prompt will appear on your console device.

3. Enter the command GA800. This gives control of your Intellec system to the text editor, which in the Model 210 is in a ROM chip set.

4. Proceed with the operations shown at the beginning of the text editing session.

5. If you wish to use the program in connection with the console session on the assembler in Chapter 5, do not turn off the power. Your program will be available when you wish to try the later console session.

As you use your Intellec system you will become very familiar with the text editor, including a few commands that we have not discussed here.

```
15 IL4:     DCR     C
   $$
16 *BSAROIU$AROU$0LT$$
            JMP     L1          ; AROUND THE LOOP INDEFINITELY
17 *B50T$$
            ; AN ASSEMBLY LANGUAGE PROGRAM TO SEND THE LETTER 'X'
            ; TO THE CONSOLE OUTPUT DEVICE, ONCE EACH SECOND
            ;
   CO       EQU     0F809H  ; PROVIDE ADDRESS OF CO ROUTINE IN MONITOR
            CSEG            ; MAKE THE SEGMENT RELOCATABLE
            STKLN   2       ; SET STACK LENGTH
   START:   LXI     SP,STACK ; INITIALIZE STACK POINTER
   L1:      MVI     D,50    ; WILL CALL DELAY SUBROUTINE 50 TIMES
   L2:      MVI     A,200   ; SUBROUTINE PARAMETER
            CALL    DELAY   ; INVOKE SUBROUTINE
            DCR     D
            JNZ     L2      ; 50 TIMES AROUND THIS LOOP = 1 SEC
            ;
            MVI     C,'X'   ; SEND 'X' TO CONSOLE OUTPUT DEVICE
            CALL    CO      ; MONITOR CONSOLE OUTPUT ROUTINE
            JMP     L1      ; AROUND THE LOOP INDEFINITELY
            ;
            ; THE DELAY SUBROUTINE
            ;
   DELAY:   MVI     B,12
   L3:      MOV     C,B
   L4:      DCR     C
            JNZ     L4
            DCR     A
            JNZ     L3
            RET
            ;
            END     START   ; PROGRAM EXECUTION BEGINS WITH SYMBOL 'START'
18 *E$$
```

The Model 210 provides you with the minimum system required for the rapid and efficient development of microcomputer software, while allowing you the option of easily upgrading to a diskette-based system as your performance needs and budget allow. The Model 210's new ROM-based Editor/Assembler combination allows the development of small 8080 or 8085 programs completely in RAM memory - minimizing your usage of paper tape. An optional MCS-48™ ROM Assembler/Editor provides the same capability for the Intel MCS-48 family of single-chip microcomputers.

The compact new system has 32K bytes of RAM, 24K bytes of ROM and its own microprocessor. A self-test diagnostic capability is built into the system. The Model 210 interfaces to your own terminal to get you started on your microcomputer development project - with a minimum of inconvenience and an extremely low price!