CHAPTER 2
EDITOR-ASSEMBLER

2.1 OUTLINE OF THE EDITOR-ASSEMBLER

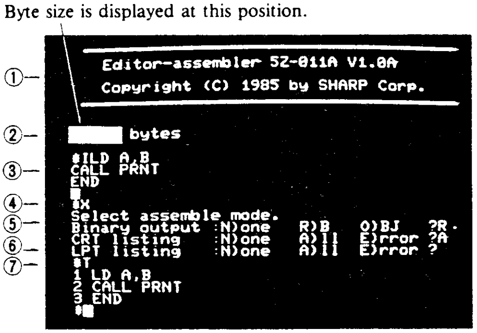
As its name indicates, the editor-assembler is the system program which includes both the text editor and the assembler. This section discusses the editor-assembler in outline; see section 2.2 and 2.3 for details.

Control is transferred between the text editor and the assembler as indicated below.

Text editor $\longrightarrow$ **Assembler:** **"X" command**
Assembler $\longrightarrow$ **Text editor:** $\boxed{\text{SHIFT}}$ + $\boxed{\text{BREAK}}$

The reason for combining the text editor and the assembler in this manner is to eliminate the need to change cassette tapes when control is transferred between the two. That is, combining the text editor and the assembler makes it possible to edit and assemble programs in one setting by allowing the assembly list to be reviewed and errors in the source program to be corrected immediately. For example, it is normal for several errors to be made in keying and symbols during source program preparation; if it were necessary to replace the tape each time an error was corrected, a great amount of time would be consumed. The text editor eliminates this requirement and makes it possible to both edit the source program and check it at the same time.

In the photograph below, the editor-assembler is first loaded by the IPL, then three text lines are prepared using text editor (which is activated first); then the **X command** is executed to shift to assembler; finally, the $\boxed{\text{SHIFT}}$ and $\boxed{\text{BREAK}}$ keys are pressed simultaneously to return to the text editor from the assembler and the **T command** is executed.



```
Editor-assembler SZ-011A V1.0A
Copyright (C) 1985 by SHARP Corp.

       bytes
*ILD A,B
CALL PRNT
END

*X
Select assemble mode.
Binary output :N)one   R)B    O)BJ   ?R
CRT listing   :N)one   A)ll   E)rror ?A
LPT listing   :N)one   A)ll   E)rror ?
*T
1 LD A,B
2 CALL PRNT
3 END
*
```

Example of display by the MZ disk version editor-assembler.

① : Editor-assembler loaded by IPL program.
② : Number of usable edit buffer bytes displayed.
③ : Three lines of text prepared using the text editor "I" command.
④ : "X" command executed to transfer control to the assembler.
⑤ : Instruction entered in response to question from the assembler.
⑥ : Control returned to the text editor with $\boxed{\text{SHIFT}}$ + $\boxed{\text{BREAK}}$ and the command wait state entered.
⑦ : "T" command entered and text lines displayed. The CP remains in the position it was in before control is transferred to the assembler.

2.2 TEXT EDITOR

2.2.1 Outline of the text editor

The text editor is used to prepare source programs for the assembler and files (such as data files) which consist of strings of ASCII characters. It is also used to read in and correct or edit such programs and files and to output edited source files.

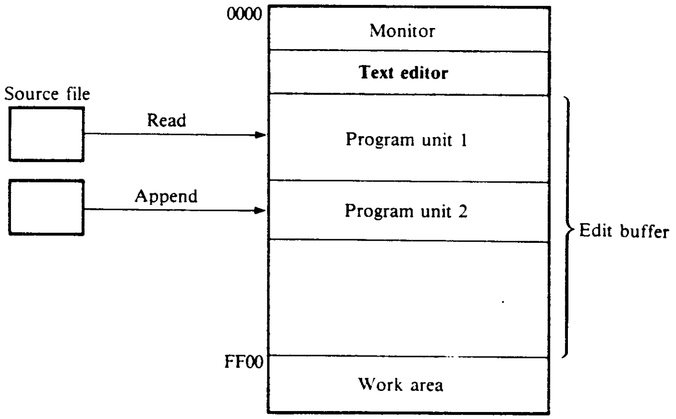
The following functions are provided for making modifications and revisions.

1. **Insertion**
2. **Deletion**
3. **Change**

Data input into the edit buffer is organized two dimensionally in lines and columns. A number which is referred to as the **line number** is assigned to each line in sequence, starting with the first line in the edit buffer.

Locations within the edit buffer which are to be modified are usually specified by means of a pointer (which is referred to as the **character pointer**, hereafter referred to as **CP**). Insertions, deletions, and changes are made by moving the CP to the appropriate line and executing the appropriate command. Revisions and modifications can be made in units of either lines or words. It is also possible to search for or exchange character strings in character string units.

When the text editor is used, the memory is organized as shown in the figure below.



The text editor provides the following commands. These commands are almost compatible with that of the editor of the Data General's NOVA minicomputer.

Command type	Command name	Function
File control commands	\ DEFAULT	Sets the specified external storage device as the default device.
	\ DELETE	Deletes the specified file in the RAM file.
	\ DIR	Displays the contents of the directory.
	\ DIR/P	Prints out the contents of the directory on the printer.
	\ INIT	Initializes the MZ disk.
	\ LOADALL	Loads all files on the a MZ disk into the RAM file.
	\ MODE	Specifies the number of characters to be printed on a line by the colour plotter printer and/or to be displayed on the screen.
	\ RENAME	Changes the name of the specified file.
	\ RUN	Executes machine language programs.
	\ SAVEALL	Saves all files in the RAM files on a MZ disk.
Input command	R	Clears the edit buffer and inputs file indicated by the filename. The CP is positioned at t beginning of the edit buffer after execution of this command. (Read file)
	A	Appends the input file indicated by the filename to the contents of the edit buffer. (Append) The CP position is not changed.
Output command	W	Writes the edit buffer contents to the storage device specified name in ASCII code.
Comparison command	V	Compares the contents of the edit buffer with the contents of the specified file.
Type command	T	Displays the entire contents of the edit buffer. The CP position is not changed.
	nT	Displays n lines starting at the CP position.
CP positioning command	B	Positions the CP at the beginning of the edit buffer.
	nJ	Positions the CP at the beginning of the line indicated by n (line number).
	nL	Moves the CP to the beginning of the line n lines after the current CP position.
	L	Moves the CP to the beginning of the current line. This is the same as when n = 0 in the nL command.
	nM	Changes the CP position by n characters.
	M	Does not move the CP. This is the same as when n = 0 in the nM command.
	Z	Moves the CP to the end of the text in the edit buffer.
	C	Searches for the specified character string and replaces it with another character string; the search starts at the current CP position and proceeds to the end of the edit buffer. The CP is repositioned to the end of the character string replaced.
	Q	Repeats the C command each time the specified character string is found until the end of the buffer is reached. The CP is repositioned to the end of the character string last replaced.
	I	Insert the specified character string at the position of the CP. The CP is repositioned to the end of the character string inserted. Line numbers are updated when a line is inserted with this command.
	nK	Deletes the n lines following the CP. The CP position is not changed.
	K	Deletes all characters preceding the CP until a CR code is detected. The CR code is not deleted.
	nD	Deletes the n characters following the CP.
	D	No operation.
Search command	S	Searches for the specified character command string, starting at the CP position and proceeding to the end of the buffer. The CP is repositioned to the end of the character string when it is found.
	=	Displays the number of characters (including spaces and CRs) stored in the edit buffer.
	.	Displays the number of the line at which the CP is located.
	&	Deletes the entire contents of the edit buffer.
	X	Transfers control to the assembler.
	#	Changes the list mode for listing to the printer.
	!	Passes control to the monitor.

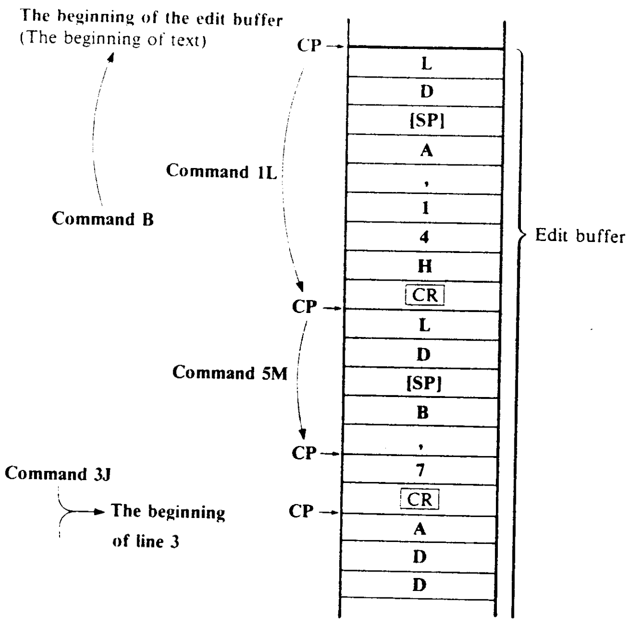
2.2.2. Character pointer and delimiter

The character pointer (CP) is positioned at **the boundary between two adjacent characters** or the beginning or end of the text. It does not point directly at any character.

Movement of the CP is explained below based on the assumption that the following text is stored in the edit buffer.

```
1 LD A,14H
2 LD B,7
3 ADD A,B
4 DAA
```

● Example of text typed in (Line numbers are not stored in the edit buffer as shown in the figure at right.)



The **B** command moves the CP to the beginning of the edit buffer, the **J** command moves it to the top of the specified line and the **L** command to the beginning of the **n**th line from the line in which the **CP** is currently located; the top of the specified line is the boundary following the CR code of a preceding line.

● **The delimiter** is used to separate commands. Enter it by pressing function key F5 . When the delimiter is entered between individual commands, several commands can be entered together and executed in sequence by pressing CR once. Thus, the two sequences shown below perform the same function.

B CR
10L CR
1K CR

$\Longleftrightarrow$ B ▨ 10L ▨ 1K CR

The I (Insert) command must be followed by a delimiter because it uses CR codes as character codes for the source text.

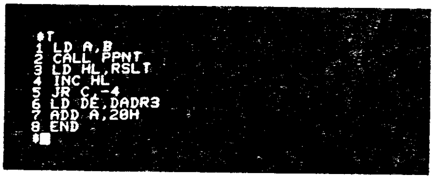
The following example replaces ADD on line 3 in the above program with ADC.

3J ▨ 2M ▨ 1D ▨ IC ▨ CR **or** **B ▨ CADD ▨ ADC** CR

— Screen editing —

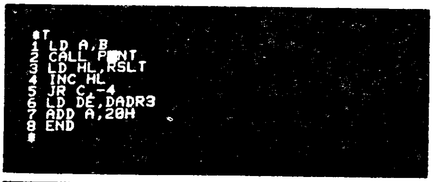
Data can be changed or modified directly on the CRT screen. After the data has been displayed using the T, C, Q, or S commands, the cursor is moved to lines displayed on the screen and the data is rewritten. The line in which the cusor is positioned is changed when [CR] is pressed, and the **CP** is positioned to the end of that line. It is also possible to change multiple lines in succession.

It should be noted that line numbers change when the I,D, and K commands are used; this can make it impossible to change the line desired.



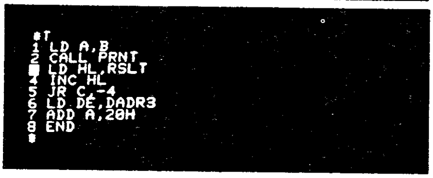
```
*T
1 LD A,B
2 CALL PPNT
3 LD HL,RSLT
4 INC HL
5 JR C,-4
6 LD DE,DADR3
7 ADD A,20H
8 END
*■
```

Display text on the CRT screen with T command. (The 2nd and 6th lines require revision.)



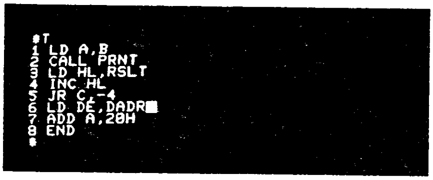
```
*T
1 LD A,B
2 CALL P■NT
3 LD HL,RSLT
4 INC HL
5 JR C,-4
6 LD DE,DADR3
7 ADD A,20H
8 END
*
```

Move the cursor to the point to be modified.



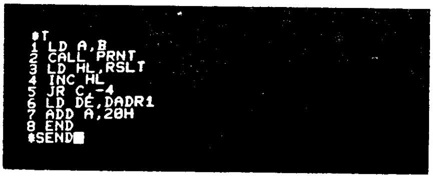
```
*T
1 LD A,B
2 CALL PRNT
■ LD HL,RSLT
4 INC HL
5 JR C,-4
6 LD DE,DADR3
7 ADD A,20H
8 END
*
```

Make the change and press [CR].



```
*T
1 LD A,B
2 CALL PRNT
3 LD HL,RSLT
4 INC HL
5 JR C,-4
6 LD DE,DADR■
7 ADD A,20H
8 END
*
```

Move the cursor to the next line to be modified, make the change, and press [CR].



```
*T
1 LD A,B
2 CALL PRNT
3 LD HL,RSLT
4 INC HL
5 JR C,-4
6 LD DE,DADR1
7 ADD A,20H
8 END
*SEND■
```

Return the text editor to the command wait state by moving the cursor to a blank line and pressing [CR]; or, position the cursor immediately after "*" and enter the next command immediately.

2.2.3 Text editor commands

— File control commands —

\ DEFAULT command

This command sets the specified storage device as the default (current) device to/from which the specified file is written or read when a file name is specified without a device name in the I/O commands. The default device setting is also effective for the DIR command.

*** \ DEFAULT QD** CR	Sets the MZ disk as the default device.
*** \ DEFAULT CMT** CR	Sets the cassette tape as the default device.
*** \ DEFAULT RAM** CR	Sets the RAM file as the default device.

—Type in **/DEFAULT** while the text editor is in the command wait state (*).
—Type in a device name with a space between the command and the device name.
—Press the CR key; the text editor sets the specified device as the default device.
Note: When the editor-assembler is started up, the storage device from which it is loaded is set as the default device.

\ DIR command

This command displays the contentes of the directory of the specified storage device, that is, a list of the names of files stored on the media in the specified storage device. When the MZ disk is specified, it is set as the default device after execution of this command.

*** \ DIR** CR	Displays the contentes of the directory of the current default storage device.
*** \ DIR QD** CR	Displays the contents of the directory of the MZ disk.
*** \ DIR RAM** CR	Displays the contents of directory of the RAM file.

—Type in **\ DIR** while in the command wait state (*).
—Specify a device name with a space between the command and the device name. (The device name may be omitted when the current device is to be specified.)
—Press the CR key; the text editor displays the contents of the directory of the specified or default device.
Note: This command cannot be used with the cassette tape.

\ DIR/P command

This command prints out the contents of the directory on the printer.

*** \ DIR/P** CR	Prints out the contents of the directory of the specified device on the printer.
*** \ DIR/P** CR	Prints out the contentes of the directory of the MZ disk on the printer.
*** \ DIR/P RAM** CR	Prints out the contentes of the directory of the RAM file on the printer.

—Type in **\ DIR/P** while the text editor is in the command wait state (*).
—Specify a device name with a space between the command and the device name.

(The device name may be omitted when the current device is to be specified.)

—Press the CR key; the text editor prints out the contents of the directory of the specified device on the printer.

Note: This command cannot be used with the cassette tape.

\ INIT command

This command initiallizes the MZ disk or RAM file. Refer to the MZ-800 Owner's Manual for the detailed explanation of the other functions of this command.

* \ **INIT** CR	Initializes the MZ disk.
* \ **INIT QD** CR	Initializes the MZ disk.
* \ **INIT "RAM:$FFFF"** CR	Initializes the RAM file.
* \ **INIT "LPT:S2"** CR	Sets the listing device to a CENTRONICS standard printer.

—Type in \ **INIT** while the text editor is in the command wait sate (*).

—Type in QD or RAM:$FFFF with a space between the command and the device name.

—Press the CR key; confirmation message "OK? [Y/N]" appears on the screen.

—Press Y to execute initialization and N to cancel it in response to the message. When you press N, the text editor returns to the command wait state again.

\ MODE command

This command sets the number of characters printed on a line by the colour plotter printer in its text mode and that displayed on the CRT screen. Both the printer and the CRT are set to 40 characters when the power is turned on.

* \ **MODE TN** CR	Sets the line length for printing to 40 characters per line.
* \ **MODE TL** CR	Sets the line length for printing to 26 characters per line.
* \ **MODE TS** CR	Sets the line length for printing to 80 characters per line.
* \ **MODE DL** CR	Sets the line length for display to 40 characters per line.
* \ **MODE DS** CR	Sets the line length for display to 80 characters per line.

—Key in \ **MODE** while the text editor is in the command wait state (*).

—Specify TN, TL, TS, DL or DS.

—Press the CR key.

Note: \ MODE DS and \ MODE DL cannot be used in the MZ-700 mode.

\ RUN command

This command executes the specified machine language program.

* \ **RUN"TRANS"** CR	Executes machine language program TRANS on the current storage device.
* \ **RUN"TEST",R** CR	Sets the memory into the same state as when IPL (Initial Program Loading), loads machine language program TEST, then executes the program.

—Type in \ **RUN** while the text editor is in the command wait state (*).

—Specify the file name. When you execute machine language programs created on the MZ-80K series computers, R must be specified following the file name. Type in a comma "," after the file name when specifying R.

—Press the CR key.

When R is not specified, the specified program is loaded without changing the current memory state and executed. When the R option is specified, the memory is set into the same state as when IPL and the specified program is loaded and executed.

Note: 1. This command cannot be used with the cassette tape.

2. When the RUN command is executed, control is transferred to the specified program after that program is loaded. In some cases, control is not returned to the text editor. If the specified machine language program is to be executed in a memory area overlapping the area in which the editor-assembler is stored, it is loaded over the editor-assembler program and the editor-assembler will be destroyed.

\ DELETE command

* \ **DELETE"RAM:SAMPLE"** CR	Deletes file "SAMPLE" in the RAM file.

—Key in \ **DELETE** while in the command wait state (*).

— Type in the device name, then file name after the commad name.

—Press CR ; the specified file is deleted.

Note: This command cannot be used in the MZ-700 mode.

\ RENAME command

* \ **RENAME"RAM:OLDPROG","NEWPROG"** CR	Changes file name OLDPROG of the file in the RAM file to NEWPROG.

—Type in \ **RENAME** while in the command wait state (*).

—Specify the current file name to be changed and a new file name.

—Press CR ; the current file name is changed to the new file name.

Note: 1. When a file has already been saved under the same file name as the specified new file name in the RAM file, execution of this command results in an error.

2. This command cannot be used in the MZ-700 mode.

\ LOADALL command

*** \ LOADALL** CR	Reads the entire contents of the MZ disk into the RAM file.

—Type in **\ LOADALL** while in the command wait state (*).

—Press CR ; the entire contents of the MZ disk is read into the RAM file.

Note: 1. Optional RAM file MZ-IR18 must be installed and initiallized in advance to execute this command.
See the \ INIT command on page 16 for the method of initializing the RAM file. When the size of the RAM file is smaller than that required, this command cannot be executed even if the RAM file has been initialized. In this case, expand the RAM file size with the \ INIT command (maximum 63KB).

2. This command cannot be used in the MZ-700 mode.

\ SAVEALL command

*** \ SAVEALL** CR	Saves the entire contents of the RAM file on the MZ disk.

—Key in **\ SAVEALL** while in the command wait state (*).

—Press CR ; the entire contents of the RAM file is saved on the MZ disk.

Note: 1. This command cannot be used in the MZ-700 mode.

2. Optional RAM file MZ-IR18 must be installed to execute this command. When two or more files are stored in the RAM file, all files in the RAM file cannot always be saved on the MZ disk even if the total file size does not exceed the capacity of the MZ disk, because those files are recorded on the MZ disk with a blank space for separation between adjacent files.

— Input commands —

R (Read file) command

This command clears the edit buffer area, then loads it with the source file (ASCII file) specified by the filename in it; loading starts at the beginning of the edit buffer. The CP is positioned at the beginning of the edit buffer after execution of this command.

***RFORMULA # 1** CR	Reads source file **FORMULA # 1** into the edit buffer.
***R"CMT:FORMULA # 2"** CR	Reads source file **FORMULA # 1** from the cassette tape into the edit buffer.

—Key in **R** while in the command wait state ("*").

—Specify the filename immediately following R. (When the file to be read is the first file on the cassette tape, the file name can be omitted.)

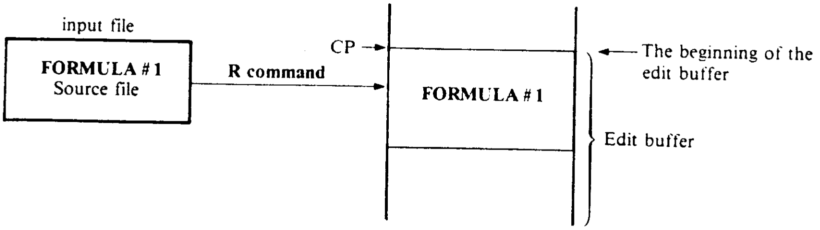
—The text editor locates the specified file and reads it when CR is pressed.

—The file read is stored in the edit buffer, starting at the edit buffer's beginning. (See the figure below.)

—"OK" is displayed after the file has been read; the CP is positioned to the beginning of the edit buffer.

— SHIFT + BREAK terminates the R command.

— The message "Full buffer" is displayed when the buffer becomes full. In this case, the entire file has not been read.



A (Append file) command

This command appends the file specified by the filename to the contents of the edit buffer. The CP position is not changed.

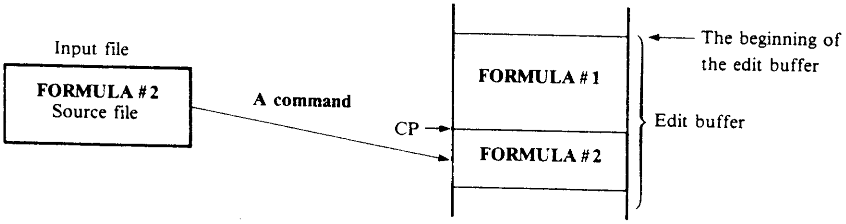
***AFORMULA #2** CR	Appends source file **FORMULA #2** to the contents of the edit buffer starting at the CP position.
***A"CMT:FORMULAR #3"** CR	Appends file **FORMULAR #3** on the cassette tape to the contents of the edit buffer starting at the CP position.

— Key in **A** while in the command wait state ("*").

— Specify the filename immediately following A. (The filename may be omitted when the file to be append is the first file on the cassette tape.)

— The text editor locates the specified file and reads it when CR is pressed.

— The file read is stored in the edit buffer, starting at the position of the CP. **Use Z in order to position the CP to the end of the text when an addition is to be made to its end.** The figure below shows addition of input file "FORMULA #2" to the end of text "FORMULA #1".)



— The CP is positioned to the beginning of the data added.

— Press SHIFT + BREAK to terminate the A command.

— The message "Full buffer" is displayed when the buffer becomes full. In this case, the entire file has not been read in.

— Output command —

W (Write) command

This command outputs the entire contents of the edit buffer to the output file specified by the filename regardless of the CP position.

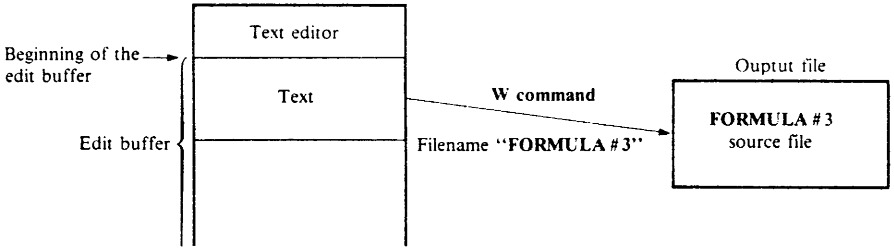
***WFORMULA #3** CR	Assigns file name "**FORMULA #3**" to the file in the edit buffer and outputs the file to the current storage device.
***W"QD:FORMULA #4"** CR	Outputs the text created in the edit buffer to the MZ disk under file name **FORMULA #4**.

—Key in **W** while in the command wait state ("*").

—Specify the file name. (When the source file is output to the cassette tape, the file name can be omitted.)

—The text editor begins output of the text to the specified device when CR is pressed.

—After output of the file has been completed, the text editor enters the command wait state. The file output is a source file.



—The CP position is not affected by execution of the W command.

—Press SHIFT + BREAK to terminate the W command.

— Verify command —

V (Verify) command

This command verifies the contents of the edit buffer with the contents of the file whose file name is specified.

***VFORMULA #3** $\boxed{\text{CR}}$	Verifies the contents of the edit buffer with the contents of file **FORMULA #**3.
***V"CMT:FORMULA #4"** $\boxed{\text{CR}}$	Verifies the contents of the edit buffer with the contents of file **FORMULA #4** on the cassette tape.

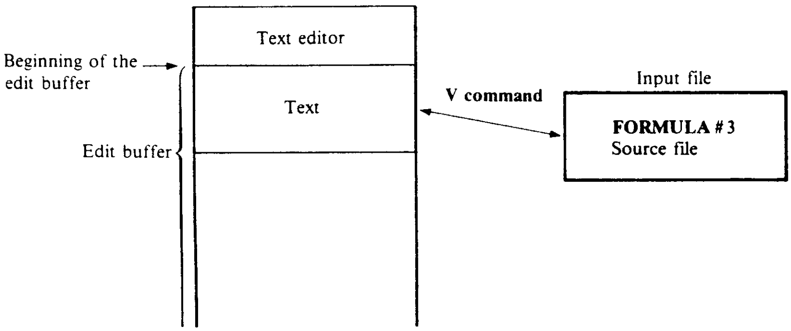
—Key in **V** while the text editor is in the command wait state.

—Key in the name of the file whose contents are to be verified. (When the file is the first file on the cassette tape, the file name may be omitted.)

—Press $\boxed{\text{CR}}$; the system then searches for the specified file and starts verification.

—When the contents of the file is the same as that of the edit buffer, the system returns to the command wait state. Otherwise, "Not same" is displayed.

—The CP position is not affected by execution of the W command.



— Type command —

T (Type) command

This command displays all or a part of the contents of the edit buffer with line numbers attached. The CP position is not changed.

∗T CR	Displays all of the contents of the edit buffer with line numbers attached.
∗nT CR	Assigns line numbers to lines, starting at the CP position and continuing to the line specified by n, then displays them. (Same as above when n = 0).

—Key in **the number of lines, n** followed by **T** (Type) while in the command wait state.

—Press CR ; the contents of the edit buffer is displayed.

—The following are special cases of nT.

n = 0:	The same as T
0<:	Error message "???" is displayed.
n ≧ m	(Where m is the number of lines from the one at which the CP is located to the end of the buffer contents): only m lines are displayed.

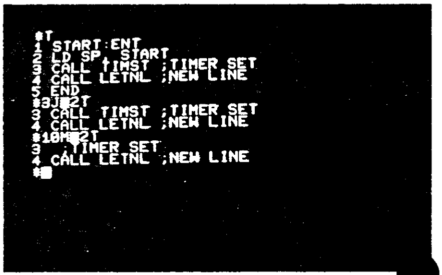
—The current CP position can be determined with the nT command, since display starts with the character following the CP.

—Press SHIFT + BREAK to terminate the T command. Press SPACE to suspend T command execution, and press it again to resume it.

—The photograph at right shows the relationship between the type command and the CP for the following text.

```
1  START:ENT
2  LD SP, START
3  CALL TIMST ;TIMER SET
4  CALL LETNL ;NEW LINE
5  END
```

—Error message "Large" is displayed when n exceeds 65535.



```
*T
1 START:ENT
2 LD SP, START
3 CALL TIMST ;TIMER SET
4 CALL LETNL ;NEW LINE
5 END
*3J█2T
3 CALL TIMST ;TIMER SET
4 CALL LETNL ;NEW LINE
*10M█2T
3   TIMER SET
4 CALL LETNL ;NEW LINE
*█
```

— CP positioning commands —

B (Begin) command

***B** CR	Positions the **CP** to the beginning of the edit buffer.

—Key in **B** while in the command wait state (*).
—Press CR .
—The B command is executed to position the CP to the beginning of the edit buffer.
—nB performs the same function.

Z command

***Z** CR	Moves the **CP** to the end of text in the edit buffer.

—Key in **Z** while in the command wait state (*).
—Press CR .
—When the Z command is executed, the CP is positoned to the end of the text in the edit buffer.
—nZ performs the same function.

J (Jump) command

***nJ** CR	Positions the **CP** to the beginning of line n.

—Key in **line number n** and **J** while in the command wait state (*).
—Press CR .
—The nJ command is executed to position the CP to the beginning of line n.
—The following are special cases.

n=0 or 1 or n is omitted:	
	The command performs the same function as the B command.
n<0:	Error message "???" is displayed.
n≧m	(Where m is the number of lines of the edit buffer contents): This command performs the same function as the Z command.

L (Line) command

This command moves the CP forward or backward by the specified number of lines. The CP is positioned at the beginning of the specified line after execution.

∗nL $\boxed{\text{CR}}$	Moves the **CP** to the beginning of the **n**th line from the line at which it is currently located.
∗L $\boxed{\text{CR}}$	Moves the **CP** to the beginning of the line at which it is currently located.

—Key in **the number of lines, n** and **L** while in the command wait state (∗).
—Press $\boxed{\text{CR}}$.
—The CP is positioned at the beginning of the specified line when the nL command is executed.
—The following are special cases:

$n = 0$:	The command functions in the same manner as the L command.
$n \geqq m$	(where m is the number of lines from the line at which the CP is located to the end of the edit buffer contents): The command functions in the same manner as the Z command.
$n < 0$:	The CP is moved n lines toward the beginning of the edit buffer.
$\lvert n \rvert \geqq \ell - 1$	(where ℓ is the number of the line at which the CP is currently located): The command functions in the same manner as the B command.

M (Move) command

This command moves the CP forward or backward by the specified number of characters. Spaces and carriage returns are counted as characters, but line numbers are not.

∗nM $\boxed{\text{CR}}$	Moves the **CP** to the position which is **n** characters from its current position.

—Key in **the number of characters, n** and **M** while in the command wait state (∗).
—Press $\boxed{\text{CR}}$; the nM command is executed to move the CP to the specified boundary between characters.
—When $n < 0$, the CP is moved backward by $\lvert n \rvert$ characters.
—The CP position is not changed when $n = 0$ or if it is omitted.

— Correction commands —

C (Change) command

This command replaces a string in the edit buffer with another string. The search for the specified string starts at the current CP position and proceeds toward the end of the edit buffer; the string is replaced when it is found and the CP is positioned at the end of the string replaced.

***Cstring 1 ▩ string 2 [CR]**	**Searches for the character string specified with string 1, starting at the current CP position and proceeding toward the end of the edit buffer; replaces the string with the one specified by string 2 when it is found.**
***Cstring 1 [CR]**	**Deletes the character string specified by string 1.**

—Key in **C** while in the command wait state (*).
—Key in the string to be located followed by a delimiter.
—Key in the string which is to replace the one located.
—Press [CR] and a search is made for the first string. Only the first occurrence of the string is replaced. The line including the string replaced is displayed and the CP is positioned at the end of that string.
—The message "Not found" is displayed if the specified string is not found.
—Strings 1 and 2 need not be of the same length.

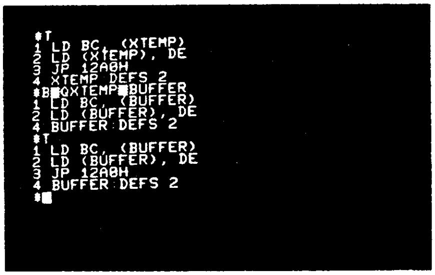
Q (Queue) command

This command repeats the function of the C command each time the specified character string is found until the end of the edit buffer is reached. The CP is repositioned to the end of the string last replaced.

***Qstring 1 ▩ string 2 [CR]**	**Causes the function of the C command to be executed repeatedly.**
***Qstring 1 [CR]**	**Deletes all occurrences of the character string specified by string 1.**

—Key in **Q** while in the command wait state (*).
—The remainder of the operation is the same as for the C command.
—The photograph at right shows the result of execution of the Q command on the following text.

```
1 LD BC, (XTEMP)
2 LD (XTEMP), DE
3 JP 12A0H
4 XTEMP:DEFS 2
```



```
*T
1 LD BC, (XTEMP)
2 LD (XTEMP), DE
3 JP 12A0H
4 XTEMP DEFS 2
*B█QXTEMP█BUFFER
1 LD BC, (BUFFER)
2 LD (BUFFER), DE
4 BUFFER:DEFS 2
*T
1 LD BC, (BUFFER)
2 LD (BUFFER), DE
3 JP 12A0H
4 BUFFER:DEFS 2
*█
```

I (Insert) command

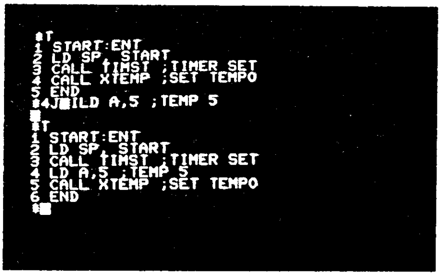
This command inserts the specified string at the CP position. A carriage return is performed on the CRT screen if one is included in the string. Line numbers are updated automatically when a new line is inserted. The CP is repositioned to the end of the string inserted.

***Istring** ▩ [CR]	Inserts the specified string at the **CP** position.
***Istring 1** [CR]	Inserts the lines specified by **string 1**,
string 2 [CR]	**string 2** and **string 3** at the **CP** position.
string 3 [CR]	
▩ [CR]	A CR is treated as a character by the I command. Therefore, a delimiter must be keyed in before [CR] is pressed to separate the CR from the preceding string and terminate the I command.

—Key in **I** while in the command wait state (*).
—Key in the string to be inserted.
—Characters keyed in are inserted starting at the CP position. Therefore, the edit buffer contents following the CP is automatically shifted toward the end of the edit buffer.
—When a [CR] is pressed in, it is inserted as a carriage return code.
—Key in a delimiter after all the strings have been keyed in.
—Press [CR] to execute the I command.
—The photograph at right shows an example of using the I command.

Text:
```
1 START:ENT
2 LD SP, START
3 CALL TIMST ;TIMER SET
4 CALL XTEMP ;SET TEMPO
5 END
```

LD A,5 ;TEMPO 5 is inserted between lines 3 and 4 of the above text.


```
*T
1 START:ENT
2 LD SP, START
3 CALL TIMST ;TIMER SET
4 CALL XTEMP ;SET TEMPO
5 END
*4J▩ILD A,5 ;TEMP 5
▩
*T
1 START:ENT
2 LD SP, START
3 CALL TIMST ;TIMER SET
4 LD A,5 ;TEMP 5
5 CALL XTEMP ;SET TEMPO
6 END
*■
```

When you create a new source file, first enter the I command (I [CR]) and then type in the program.

K (Kill) command

This command deletes the n lines preceding or following the CP from the edit buffer.

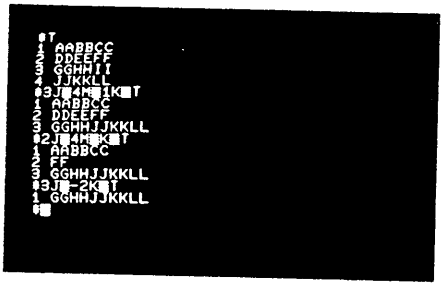
***nK** [CR]	Deletes the **n** lines preceding or following the **CP** from the edit buffer. If the **CP** is located in the middle of a line, the characters preceding the **CP** are not deleted if **n>0** and the characters following the **CP** are not deleted if **n≤0.**
***K** [CR]	Deletes characters preceding the **CP** position until a CR code is detected. The [CR] code is not deleted.

—Key in **the number of lines, n** and **K** while in the command wait state (*).
—Press [CR] to execute the K command.
—Operation differs according to the value of n as follows.

n>0:	Deletes all characters following the CP until n CR codes are detected. CR codes detected are also deleted. Command execution ends after the last code has been deleted.
n<0:	Deletes all charaeters preceding the CP until |n| + 1 CR codes are detected. The (|n| + 1)th CR code is not deleted.
n=0 or not specified	Deletes all characters preceding the CP until a CR code is detected. That is, delelets the part of the line in front of the CP. The CR code detected is not deleted.

—Line numbers are automatically updated after deletion.
—The CP position is not changed.
—The photograph at right shows an example of the result of execution of the K command with the following text. (This text is presented only to illustrate operation of the command; it has no meaning in assembly language.)

```
1 AABBCC
2 DDEEFF
3 GGHHII
4 JJKKLL
```



```
*T
1 AABBCC
2 DDEEFF
3 GGHHII
4 JJKKLL
*3J■4M■1K■T
1 AABBCC
2 DDEEFF
3 GGHHJJKKLL
*2J■4M■K■T
1 AABBCC
2 FF
3 GGHHJJKKLL
*3J■-2K■T
1 GGHHJJKKLL
*■
```

D (Delete) command

This command deletes the specified number of characters from the edit buffer, starting at the **CP** position.

***nD** $\boxed{\text{CR}}$	Deletes the specified number of characters from the edit buffer, starting at the **CP** position. A CR code is counted as a character.
***D** $\boxed{\text{CR}}$	(No operation results.)

—Key in **the number of character n** and **D** while in the command wait state (*).

—Press to execute the command.

—Operation differs according to the value of n as follows.

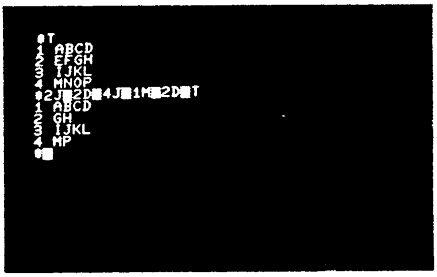
n>0:	Deletes the n characters following the CP from the edit buffer. A CR code is counted as a character.
n<0:	Deletes the n characters preceding the CP from the edit buffer. A CR code is counted as a character.
n=0 or not specified	No operation results.

—Line numbers are automatically updated if necessary.

—The CP position is not changed.

—The photograph at right shows an example of the result of execution of the D command with the following text. (This text is presented only for the purpose of this illustration; it has no meaning in assembly language.)

```
1  ABCD
2  EFGH
3  IJKL
4  MNOP
```



```
*T
1  ABCD
2  EFGH
3  IJKL
4  MNOP
*2J■2D■4J■1M■2D■T
1  ABCD
2  GH
3  IJKL
4  MP
*■
```

— Search command —

S (Search) command

This command searches the edit buffer for the specified character string.

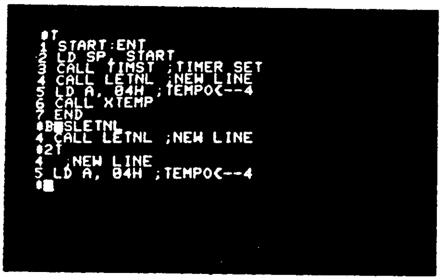
***S string** [CR]	Searches for the specified character **string**, starting at the current **CP** position; the **CP** is repositioned to the end of the character **string** when it is found.

—Key in S.
—Key in the string to be located.
—Press [CR] to execute the S command.

—The search starts at the current CP position and proceeds toward the end of the buffer.
—When the specified string is found, the line containing it is displayed and the CP is positioned to the end of the character string.
—If the specified string cannot be found, the message "Not found" is displayed and the CP is repositioned to the beginning of the edit buffer.

—The photograph at right shows the result of a search for the character string "LETNL" in the following text. The line including "LETNL" is displayed following the S command. The 2T command indicates that the CP is positioned to the end of the string.

```
1 START:ENT
2 LD SP, START
3 CALL TIMST :TIMER SET
4 CALL LETNL :NEW LINE
5 LD A, 04H :TEMPO<--4
6 CALL XTEMP
7 END
```



```
*T
1 START:ENT
2 LD SP, START
3 CALL TIMST ;TIMER SET
4 CALL LETNL ;NEW LINE
5 LD A, 04H ;TEMPO<--4
6 CALL XTEMP
7 END
*B█SLETNL
4 CALL LETNL ;NEW LINE
*2T
4  ;NEW LINE
5 LD A, 04H ;TEMPO<--4
*█
```

— Special commands —

= (equal) command

* = [CR]	Displays the total number of characters (including spaces and CRs) in the edit buffer.

—Key in = (**equal**) while in the command wait state (*).
—Press [CR] ; the total number of characters stored in the edit buffer is displayed.

. (period) command

*. [CR]	Displays the number of the line on which the CP is located.

—Key in . (**period**) while in the command wait state (*).
—Press [CR] ; the the line number on which the CP is located is displayed.

& (ampersand) command

*& [CR]	Clears the edit buffer.

—Key in & (**ampersand**) while in the command wait state (*).
—Press [CR] ; the contents of the edit buffer are then cleared.

X (TRANSfer) command

***X** [CR]	Transfers control to the assembler.

—Key in **X** while in the command wait state (*).
—Press [CR] ; control is then transferred to the assembler and an assembler message is displayed.

(sharp mark) command

* # [CR]	Changes the printer list mode.

—Key in # (**sharp symbol**) while in the command wait state (*).

—Press [CR] ; the printer list mode is then changed.

—The printer list mode is disabled when the text editor is started. It is enabled when the # command is executed once; executing it again disables it, and so on.

—The following shows a listing obtained by executing the T command when the printer list mode is enabled.

```
*
1 ;
2 ;***EDITOR LIST SAMPLE ***
3 ;
4 START:ENT
5 LD SP,START ;INITIAL STACK POINTER
6 CALL LETNL
7 LD A,5
8 CALL XTEMP ;SET TEMPO TO 5
```

! (exclamation mark) command

*! [CR]	Transfers control to the monitor.

—Key in ! (**exclamation mark**) while in the command wait state (*).

—Press [CR] ; the following message is then displayed.

"M)onitor B)oot C)ancel?"

- Pressing the M key transfers control to the monitor.
- Pressing the B key transfers control to the IPL.
- Pressing the C key cancels the ! command and returns the text editor to the command wait state.

—There are three methods of returning control to the text editor from the monitor.

- **Jump to address 5600 (4000): The editor buffer is cleared. (cold start)**
- **Jump to address 5603 (4003): The edit buffer is not cleared. (hot start)**
- **Execute the monitor's R command: Same as the hot start above.**

Note: Addresses within the parentheses must be used in the MZ-700 mode.

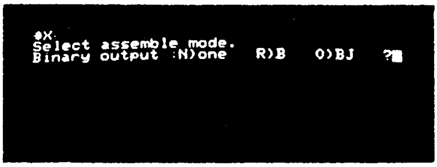
2.3 ASSEMBLER

2.3.1 Outline of the assembler

The assembler is a system program which assembles source files prepared and edited using the text editor and outputs relocatable files (relocatable binary files) or object files. Relocatable files are the stage which is between source files and object files, and are organized in such a manner as to be relocatable and linkable.

Source files are written in assembly language (label symbols, mnemonic symbols of instruction codes and directive statements) in accordance with the assembler rules. Source programs edited with the text editor are output in the ASCII code format as they are. The assembler interprets the syntax of such source programs and produces relocatable or object files. Information concerning the status of symbolic address (data) definition and syntax errors is also prepared at this time.

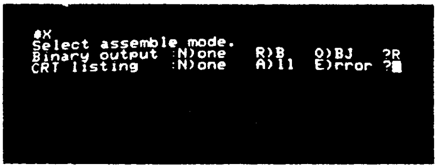
— Starting the assembler —



```
*X
Select assemble mode.
Binary output :N)one   R)B   O)BJ   ?
```

Control is transferred from the text editor to the assembler by entering the X command.
First, select the type of an output file to be generated. When no output file is needed, select None.

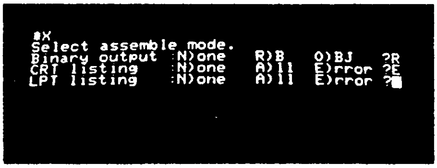
No output file	→None
Relacatable file	→RB
Object file	→OBJ



```
*X
Select assemble mode.
Binary output :N)one   R)B   O)BJ   ?R
CRT listing   :N)one   A)ll  E)rror ?
```

Ne , select what is to be displayed on the CRT screen.

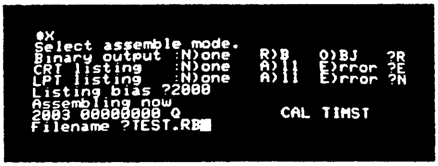
Nothing	→None
Everything	→All
Error information only	→Error



```
*X
Select assemble mode.
Binary output :N)one   R)B   O)BJ   ?R
CRT listing   :N)one   A)ll  E)rror ?E
LPT listing   :N)one   A)ll  E)rror ?
```

Finally, select what is to be printed on the printer.

Nothing	→None
Everything	→All
Error information only	→Error



```
*X
Select assemble mode.
Binary output :N)one   R)B   O)BJ   ?R
CRT listing   :N)one   A)ll  E)rror ?E
LPT listing   :N)one   A)ll  E)rror ?N
Listing bias ?2000
Assembling now
2003 00000000 Q          CAL TIMST
Filename ?TEST.RB
```

Then, enter the listing bias (4-digit hexadecimal number) (to be discussed later).
The S option can be specified after the listing bias. When the S option is specified, the following parts of an assembly list are not output.

- Macro expansions (instructions invoked by macro calls)
- Instructions which were not assembled because the condition in the preceding IF directive was not met.
- ASCII codes generated by DEFM instructions

Finally, enter a file name to be assinged to the output file when a relocatable or object file is to be generated.

— Listing bias and ORG dirdctive —

In the sample listing below, the relative address starts at 2000 as specified in the **ORG** directive at the top of the program. (The detailed explanation of the ORG directive is given later.)

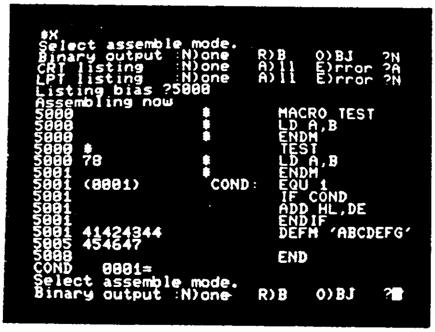
The assembly listing can be started at an appropriate address in the same manner to make it easier to read. This is the idea of the "**listing bias**" which was mentioned earlier. For example the same listing as the one shown below can be obtained without the ORG directive if a listing bias of 2000 is specified. Unlike the ORG directive, however, the listing bias is temporarily effective only on listing and no effect on relocatable or object files produced. Further, when a listing bias is specified for a program whose starting address is specified with an ORG directive, the ORG directive has priority over the listing bias.

When the S option is specified, the following parts of an assembly list are not output as shown in the photographs below.

- Macro expansions (instructions invoked by macro calls)
- Instructions which were not assembled because the condition in the preceding IF directive was not met.
- ASCII codes generated by DEFM instructions

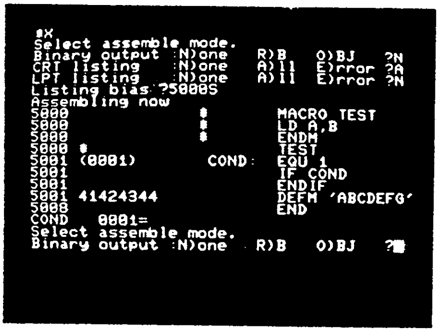
To set the listing bias to zero, press the CR key, or key in S and press the CR key (when the S option is specified).

The simple program shown below is provided for the purpose of helping to explain the function of the assembler; it has no meaning in execution.



```
*X
Select assemble mode.
Binary output  :N)one   R)B   O)BJ    ?N
CRT listing    :N)one   A)ll  E)rror ?A
LPT listing    :N)one   A)ll  E)rror ?N
Listing bias ?5000
Assembling now
5000                 *        MACRO TEST
5000                 *        LD A,B
5000                 *        ENDM
5000 *                        TEST
5000 78              *        LD A,B
5001                 *        ENDM
5001 (0001)           COND:   EQU 1
5001                           IF COND
5001                          ADD HL,DE
5001                          ENDIF
5001 41424344                 DEFM 'ABCDEFG'
5005 454647
5008                          END
COND   0001=
Select assemble mode.
Binary output  :N)one   R)B   O)BJ    ?
```

When the S option is not specified



```
*X
Select assemble mode.
Binary output  :N)one   R)B   O)BJ    ?N
CRT listing    :N)one   A)ll  E)rror ?A
LPT listing    :N)one   A)ll  E)rror ?N
Listing bias ?5000S
Assembling now
5000                 *        MACRO TEST
5000                 *        LD A,B
5000                 *        ENDM
5000 *                        TEST
5001 (0001)          COND:    EQU 1
5001                           IF COND
5001                          ENDIF
5001 41424344                 DEFM 'ABCDEFG'
5008                          END
COND   0001=
Select assemble mode.
Binary output  :N)one   R)B   O)BJ    ?
```

When the S option is specified

```
0001: 0000                    ;
0002: 0000                    ; SAMPLE LIST
0003: 0000                    ;
0004: 2000                            ORG     2000H
0005: 2000 3E33                       LD      A,'3'
0006: 2002 FE43                       CP      43H
0007: 2004 FE43                       CP      'C'
0008: 2006 22                         DEFB    '"'
0009: 2007 27                         DEFB    "'"
0010: 2008 3A3B2C22                   DEFM    ':;,"'
0011: 200C 7E                         LD      A,(HL)
0012: 200D 7E                         LD      A,M          — M may be used instead of
0013: 200E                    ;                               (HL).
0014: 200E (000A)             XYZ:    EQU     10
0015: 200E C31B20                     JP      ABC+XYZ      — Address label symbol +
0016: 2011 C30A00             ABC:    JP      XYZ            EQU defined symbol
0017: 2014 C30E20                     JP      ABC-3
0018: 2017 C30A00                     JP      10           — Absolute address 10
0019: 201A 210000        E            LD      HL,D000      — D000 interpreted as symbol.
0020: 201D 210000        E            LD      HL,100D      — 100D is interpreted as a
0021: 2020 213930                     LD      HL,12345       symbol.
0022: 2023 211B20                     LD      HL,ABC+XYZ   — EQU defined symbol +
0023: 2026 3E0D                       LD      A,XYZ+3        numerical data
0024: 2028 3EFF                       LD      A,-1
0025: 202A 21FFFF                     LD      HL,-1
0026: 202D C32C20                     JP      -1
0027: 2030                    ;
0028: 2030 CD4820                     CALL    ZZZ+10
0029: 2033 CD3F20                     CALL    ZZZ+XXX
0030: 2036 21FFFF                     LD      HL,-XXX
0031: 2039 21FEFF                     LD      HL,-XXX-XXX
0032: 203C 3D20                       DEFW    ZZZ-XXX
0033: 203E 00                 ZZZ:    NOP
0034: 203F (0001)             XXX:    EQU     1
0035: 203F                            END

100D    201DU   ABC     2011    D000    201AU   XXX     0001=   XYZ     000A=
ZZZ     203E
```

— Indicates the contents of the symbol table

2.3.2. Assembly language rules

The source program must be written in accordance with the assembly laguage rules. This subsection describes the structure of the source program and the assembly language rules.

A assembly source program consists of the following.

Z-80 instruction mnemonics
Label symbols
Comments

Assembler directives (Pseudo instructions)	**Definition directives**
	Entry directive
	Skip directive
	End directive

Comments may be used as needed by the programmer; they have no effect on execution of the program and are not included in output files.

All assembly source programs must be ended with a assembler directive END.

Z80 instruction mnemonic cods form the body of the assembly source program. These are explained in a separate volume.

A mnemonic code consists of an op-code of up to 4 characters (CALL, JP, etc.), separators (space, comma, etc.) and operands.

A label symbol represents an address or data. It is placed in the label field separated from the following instruction with a colon (:), and is referenced by using it as an operand. The first 6 characters of a label symbol are significant to discriminate one from the other. The 7th and following characters are ignored if they are used.

Therefore, **ABCDEFG** and **ABCDEFH** are treated as the same label symbol.

Alphanumerics are generally used for label symbols, but any characters other than those used for separators and other special purposes may be used.

Commnets are messages used to help to understand the operation; it must be preceded by a semi colon (;) and ended with a CR code.

Assembler directives (also called pseudo-instruction or pseudo-operation) are a number of commands to the assembler. Thay do not generate instruction codes. Instead, they inform the assembler of certain actions to be taken, or they create data values. Assembler directives are written in the same field as Z80 instruction mnemonics.

Definition directives, entry directive, skip directive and so forth are included in the assembler directives.

The END directive is one of the assembler directives. It marks the end of the assembly source file. All assembly source file must be ended with an END directive.

— Characters —

Characters which are used to write an assembly source program are alphanumeric characters, special symbols and so on. Special symbols have functional meaning to the assembler (separators (" : ", " , " and " ; "), CR code, SPACE code, etc.)

1) Alphabetic characters: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

   These characters are used to represent symbols and instruction mnemonic codes. Six letters A - F are also used as numerics of hexadecimal system which represent 10 to 15 in decimal. H is used to indicate hexadecimal.

2) Numerics: 0 1 2 3 4 5 6 7 8 9

   These are used to represent numeric constants and symbols. Whether a constant is handled as a decimal number or a hexadecimal number is determined according to the rules of numeric representation.

3) Space

   When an assembly source program is listed with the T command of the text editor, spaces are displayed (printed) as they were typed in. However, when an assembly list is output, spaces are treated as separators except when they are used in comments and cause tabulation. Spaces placed between op-code and operand or between operand and comment perform the tabulation function as shown below.

Example

```
        OR [SP] FOH [SP] ; -X0                     |
        XYZ: PUSH [SP] AF                          | Editor list
        ADD [SP] HL, BC [SP] ; BC=COUNT            |

        OR          F0H        ;  A<-X0            |
  XYZ:  PUSH        AF                             | Assembly listing
        ADD         HL, BC     ;  BC=COUNT         |
        ↑           ↑             ↑
        Tab position Tab position Tab position
```

4) Colon " : "

   A colon behaves as a separator when it is placed between a label symbol and instruction (or assembler directive). It performes the tabulation function on the assembly listing.

Example:

```
   START:  LD          SP, START
   MAIN:   ENT
           ↑
           Tab position
```

A label symbol can also be defined by writing only a label symbol and colon on a line. (See the paragraph describing label symbols.)

Example:

```
   ENTRY:                      — ENTRY is defined as the same addres as
                                 TOP0.

   TOP0:   PUSH HL
```

5) Semicolon " ; "

A semicolon indicates the beginning of a comment. The part of a line from a colon to the line end (CR) has no influence on program execution. A semicolon may be placed at the top of a line or at the beginning of the comment column.

Example:

```
;                                              The entire lines are used as
;  SAMPLE PROGRAM                              comment lines.
;
CMMNT:  ENT            ;  COMMENT
                       (Comment column)
```

6) Carriage return (CR code)

A carriage return code marks the end of a line.

7) Other special symbols [+ – ' ( ) ,]

These symbols are used in instructions.

8) Other characters

The other characters such as graphic characters are not generally used although the assembler allows the use of them for label symbols and comments.

— Line —

A typical line of a source program is made up of a label symbol, Z80 instruction or assembler directive, and comment. Components on each line are arranged according to the tab setting when it is listed. (See the assembly list on page 40.)

— Label symbols —

All characters other than special symbols may be used for label symbols, but generally alphanumerics are used. Each label symbol can consist of up to 6 characters; the 7th and following characters, if used, are ignored by the assembler.

Example:	Correct	ABC	START	BUFFER	STEP 50	
	Incorrect	(ABC),	HL	IY + 3	XYZ + 3	— Special characters are used.
	More than 6 characters	COMPARE0 COMPARE1	These are treated as the same label symbol, COMPAR.			

A label symbol can be defined as data and equated to a numeric constant (1 or 2 bytes) using assembler directive EQU.

```
Example:  ABC:      EQU   3
          CR:       EQU   0DH
          VRAMO:    EQU   D000H
```

A label symbol preceding the instruction field and followed by a colon (:) is defined as an address. It can be defined as a global symbol with assembler directive ENT.

```
Example:  RLDR:     ENT
          RLDR0:    PUSH  HL
```

When a label symbol is referenced (that is, when it is used as an operand), it must be defined in the assembly source program unit in which it is referenced, or must be declared as a global symbol in other program unit with an ENT directive.

A label symbol which has once been defined is not defined again in the same program unit.

Two or more diferrent symbols can be defined as the same relocatable instruction address as shown below.

```
Example:  ABCD:     ENT
          EFGH:     ENT
          IJK:      LD    A, B
```

Label symbols ABCD, EFGH and IJK are all defined as relocatable address of op-code LD of instruction LD A, B. ABCD and EFGH are also defined as global symbols.

```
          ABCD:
          EFGH:
          IJK:      LD    A,B
```

Same as the above, except that ABCD and EFGH ar not global symbols.

— Constants —

There are two types of constants: decimal and hexadecimal. Plus (+) and minus (-) signs can be attached to them. An alphanumeric character string which is defined as a label symbol is assumed as a label symbol even if it satisfies the requirements for a constant.

The assembler handles a constant as a decimal number when it consists of numerics only.

Example 23 999 +3 −62

The assembler handles a constnt as a hexadecimal number when it consists of 0-9, A, B, C, D, E or F and followed by H.

Example 2AH CDH +0lH −BH 0010H 00ADH 00H

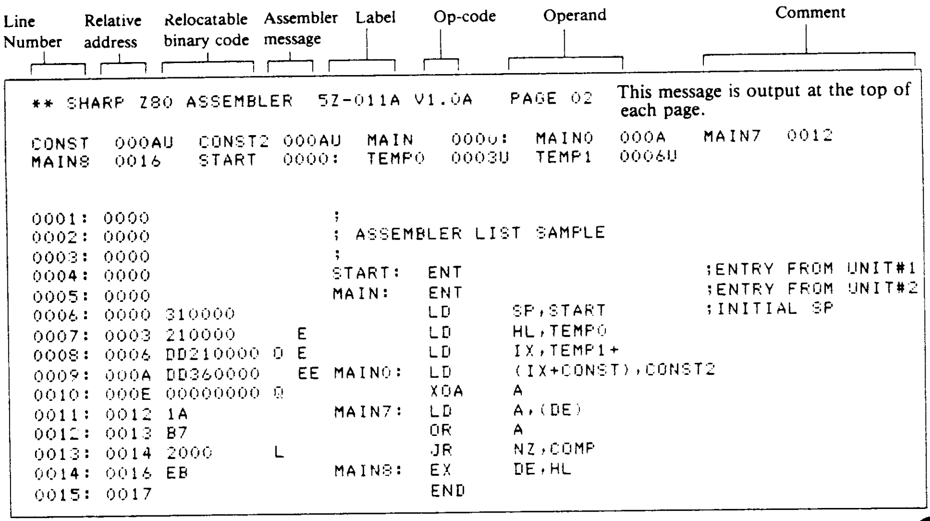
A constant used in the operands of a JP, JR, DJNZ or CALL instruction represents an absolute address when it has no sign and a location relative to the current address when it has a sign. Constants used in the operands of the other instructions represent numeric data. Negative constants are converted into two's complement.

2.3.3 Assembly listing and assembler messages

When you select "All" among the itmes displayed following "CRT listing" or "LPT listing" after the assembler is started up, an assembly list is output on the CRT screen and/or printer according to your selection. Examining this assembly listing is one of the most important procedure in programming in the assembly language, since whether there are errors in the source program, whether the desired machine codes have been obtained and so on must be checked by this examination.

The assembler assembles a source program and outputs the assembly list, which includes line numbers, relative addresses, relocative binary codes, assembler messages and source program list (including label symbols, Z80 instruction mnemonic codes and comments). The assembly listing is paged every 60 lines. Line numbers and comments are not displayed on the CRT screen.

The assembly listing format is shown below. The listing shows that tabs are set at the beginnings of the label symbol, op-code, operand and comment columns.



```
** SHARP Z80 ASSEMBLER  5Z-011A V1.0A    PAGE 02

CONST   000AU   CONST2 000AU   MAIN    0000:   MAIN0   000A    MAIN7   0012
MAIN8   0016    START  0000:   TEMP0   0003U   TEMP1   0006U


0001: 0000                        ;
0002: 0000                        ; ASSEMBLER LIST SAMPLE
0003: 0000                        ;
0004: 0000                        START:   ENT                           ;ENTRY FROM UNIT#1
0005: 0000                        MAIN:    ENT                           ;ENTRY FROM UNIT#2
0006: 0000 310000                          LD      SP,START              ;INITIAL SP
0007: 0003 210000       E                  LD      HL,TEMP0
0008: 0006 DD210000 0 E                    LD      IX,TEMP1+
0009: 000A DD360000   EE  MAIN0:   LD      (IX+CONST),CONST2
0010: 000E 00000000 Q                      XOA     A
0011: 0012 1A                     MAIN7:   LD      A,(DE)
0012: 0013 B7                              OR      A
0013: 0014 2000     L                      JR      NZ,COMP
0014: 0016 EB                     MAIN8:   EX      DE,HL
0015: 0017                                 END
```

The messages printed in the message column of the assembly listing are divided into two types: definition status messages and error messages.

— Definition status messages —

E (External)

This message indicates that **external symbol reference** is made; i.e., the label symbol which is referenced in the operands of the instruction is not defined inside the current program unit. Therefore, label symbols for which the E messages are printed in the message column must be defined as global symbols in other source program units (see assembler directive ENT on page 43). Reference to external label symbols is accomplished after the current program unit and those in which the referenced symbols are defined as global symbols are linked with the symbolic debugger.

When an external symbol referenced is not defined as a global symbol in any other program units, that symbol is assumed as an undefined symbol. If a program unit including such undefined symbols is assembled into an object file, undefined symbols treated as 1-byte data are converted into 00 and those treated as 2-byte or longer data (address) are not certain.

Example

```
E   LD          B, CONST0
↑
└─Indicates that 1-byte data CONST0 is an extertnal symbol.

E   CALL    SORT
↑
└─Indicates that address SORT (1-byte data) is an external symbol.

EE BIT TOP, (IY+FLAG)
↑↑
│└─Indicates that 1-byte data FLAG is an external symbol.
│
Indicates that 1-byte data TOP is an external symbol.
```

— Error messages —

C (illegal Character error)

This message indicates that illegal characters are used in operands.

F (Format error)

This message indicates that the instruction format is incorret.

N (Non label error)

This message indicates that assembler directive ENT or EQU has no label symbol.

Example

```
N                   EQU 0012H
↑
└─Indicates that a label symbol is missing.
```

L (erroneous Label error)

This message indicates that an illegal reference is made.

Example:
```
L    JR XYZ
↑
└──XYZ is not defined in the current source program. (External symbols cannot be
   referenced in the JR and DJNZ instructions.)
```

M (Multiple label error)

This message indicates that a label symbol is defined two or more times.

Example:
```
M    ABC: LD   DE, BUFFER
      |
M    ABC: ENT
↑
└──Indicates that ABC is defined more than once.
```

O (erroneous Operand)

This message indicates that an illegal operand has been specified.

Example:
```
O    JP +100-ABC
```

Q (Questionable mnemonic)

This message indicates that a mnemonic code is incorrect.

Example:

Q $\underbrace{\text{CAL XYZ}}$

CALL XYZ is correct.

Q $\underbrace{\text{PSH B}}$

PUSH BC is correct.

S (String error)

This message indicates that single or double quotation mark(s) is omitted.

Example:

S $\underbrace{\text{DEFM GAME OVER}}$

DEFM 'GAME OVER' is correct.

U (Undefined parameter)

This message indicates that a parameter is not defined when a macro instruction is called.

Example:
```
U    JP Z, @3
```

V (Value over)

This message indicates that a numeric specified for an operand exceeds the range allowed.

Example:
```
V    LD      A, FF8H
V    SET     8, A
V    JR      -130
```

2.3.4. Assembler directives

Assembler directives (also referred to as pseudo instructions) are commands to the assembler and are not converted into machine codes themselves, instead they inform the assembler of actions to be taken.

Among them, the DEFB, DEFW and DEFM directives generate machine codes corresponding thei operands. The other assembler directives are provided to allow effective use of label symbols, to allow to write programs effectively or to determine the assembly listing format.

— ENT (ENTry) —

This assembler directive makes an entry declaration; that is, it declares that the specified label symbol is a global symbol. Label symbols referenced in other program units must be declared as global symbols. A label symbol declared as a global symbol not only makes it possible for the symbolic debugger to link the related program units, but also allows the symbolic addressing from the other program units.

Label symbols not declared as global symbols can be referenced only inside the current program unit.

The example below shows mutual external symbol reference between program units GAUSS-MAIN and GAUSS-SR. The E messages to the left of CALL CMPLX and JP MAIN indicate that the label symbols MAIN and CMPLX are external.

Program unit 1
"GAUSS-MAIN"

```
                           ; GAUSS-MAIN
                           ;
                            MAIN:  ENT             ← Entry definition of label symbol
 Address undefined                  :                MAIN
   CD0000  E                       CALL  CMPLX
       E message                    :
                                   CALL  CMPLX+2   ← No offset can be added to a label symbol
                                    :                which is defined externally.
                                   END             ← END is always required at the end of a
                                                     program unit.
```

Program unit 2
"GAUSS-SR"

```
                           ; GAUSS-SR
                           ;
                           CMPLX:  ENT             ← Entry definition of label symbol
                                    :                CMPLX
                                   RET
 Address undefined                  :
   C30000  E                       JP     MAIN
       E message                    :
                                   END
```

— EQU (EQUate) —

This assembler directive equates a label symbol to a numeric value (or address). The numeric value must be a decimal or hexadecimal constant. Once a label symbol has been defined as a numeric constant, numerics can be added or subtracted to/from it; this allows new symbols to be defined using it.

Label symbols used in operands are handled as relative addresses and set to various values according to the starting address specified when the program is assembled. However, when a label symbol is equated to a numeric value with the EQU directive, that label symbol is set to the value regardless of the starting address.

The EQU directive also defines a label symbol as a global symbol. Therefore, a label symbol defined by the EQU directive can be referenced from other program units.
However, program units including EQU directives must be loaded before other program units to be linked.

For the above functions, the EQU directive is useful to assign easily remenbered names to entry address of monitor subroutines, I/O device port numbers and so forth.

```
0001: 0000                    ;
0002: 0000                    ; MONITOR LINK
0003: 0000                    ;
0004: 0000  (001E)            BRKEY:   EQU     001EH
0005: 0000  (0033)            TIMST:   EQU     0033H
0006: 0000                             SKP     2


0007: 0000                    ;
0008: 0000                    ; SET PORT#:PRINTER
0009: 0000                    ;
0010: 0000  (00FE)            POTFE:   EQU     FEH
0011: 0000  (00FF)            POTFF:   EQU     POTFE+1
0012: 0000                    ;
0013: 0000  (0001)            CON1:    EQU     1
0014: 0000  (0002)            CON2:    EQU     2
0015: 0000  (0003)            CON3:    EQU     CON1+CON2
0016: 0000                    ;
0017: 0000                    ;
0018: 0000                             END

** SHARP Z80 ASSEMBLER   SZ-011A V1.0A    PAGE 02

BRKEY   001E=  CON1    0001=  CON2    0002=  CON3    0003=  POTFE   00FE=
POTFF   00FF=  TIMST   0033=
```

– POTFF is equated to FF (hexadecimal).

– CON 3 is equated to 3 (decimal). In this case, CON 1 and CON 2 must be defined in advance as in this example.

The equal signs (=) in the symbol table output following the assembly list indicate that the corresponding label symbols are defined with the EQU directive.

— ORG (ORiGin) —

This assembler directive determines the object program loading address. For example, when
 ORG 2000H
is placed at the beginning of the program to be assembled, the assembler assembles the program with a load address of 2000H.

When a relocatable binary file generated with the loading address specified with the ORG directive is linked with other programs by the symbolic debugger, the loading address specified with the ORG directive is effective and that specified by the symbolic debugger is ignored.

When relocatable files with loading addresses specified with ORG directives are linked or when more than one ORG directives are used in a program, the loading addresses must be specified so that programs loaded do not overlap each other and must appear in the sequencial order.

```
0001: 0000                   ;
0002: 0000                   ;
0003: 0000                   ; BLOCK  TRANSFER
0004: 0000                   ;
0005: 2000                            ORG    2000H
0006: 2000 110000      E     B.XFER:  LD     DE,DEST
0007: 2003 210000      E              LD     HL,SOURCE
0008: 2006 010001                     LD     BC,BLOCK#
0009: 2009 EDB0                       LDIR
0010: 200B C9                         RET
0011: 200C                   ;
0012: 200C (0100)            BLOCK#:  EQU    256
0013: 200C                            END

B.XFER 2000   BLOCK# 0100=   DEST    2000U   SOURCE 2003U
```

In the symbol table output following the assembly list, symbols marked with a equal sign (=) are defined with the EQU directive, those marked with a U are undifined, and those with no messsge are correctly used inside the program.

— MACRO/ENDM —

These assembler directive define instructions between them as a macro instruction. The MACRO directive defines its operand (symbol) as the name of the macro and the ENDM directive ends the definition.

Parameters (arguments) can be used in a macro and they must be represented by serial numbers preceded by @, i.e., @1 and @2. The maximum number of parameters is 7. Use of parameters gives a macro a higher flexibility to fit the generalized case.

Macros once defined can be used to define a new macro and nesting is allowed up to three levels.

To call a macro, use its label symbol as a mnemonic code and specify real values (symbols or numerics) to replace the parameters in the macro as its operands in the ascending order of the parameter numbers.

Whenever the assembler encounters a macro name, it generates a macro expansion for the instructions defined as the macro and invokes those instructions.

The assembler outputs the assembly list with all macros expanded and their parameters replaced with real values. Macros can be defined anywhere in a program before they are called.

A macro is similar to a subroutine on the point that it is called where its function is needed in a program. However, unlike a subroutine, macro call does not transfer control to the macro. Instead, instructions defined as the macro are inserted there in the program when the program is assembled.

The listing below shows an example of the macro use. The MACRO directive, ENDM directive, instructions defined as a macro and macro call instruction (on line 11) are marked with an asterisc "*".

```
0001: 0000                *          MACRO   INT         A macro whose label
0002: 0000                * @3:      LD      A,(@1)      name is INT is defined.
0003: 0000                *          SUB     2           Four parameters are us-
0004: 0000                *          JR      NC,@3       ed.
0005: 0000                *          LD      (@4),A
0006: 0000                *          ENDM                A macro whose label na-
0007: 0000                *          MACRO   STRING      me is STRING is defined.
0008: 0000                *          DEFM    '@1@2'
0009: 0000                *          ENDM
0010: 0000                  ;
0011: 0000 *                         INT     DIV,4,START,ANS
      0000 3A1500         * START:   LD      A,(DIV)           Macro INT is expaned
      0003 D602           *          SUB     2                 and assembled with its
      0005 30F9           *          JR      NC,START          parameters replaced with
      0007 321600         *          LD      (ANS),A           the real values.
      000A                *          ENDM
0012: 000A *                         INT     DE,M,LOOP
      000A 1A             * LOOP:    LD      A,(DE)      Macro INT is expanded.
      000B D602           *          SUB     2           Error message U is
      000D 30FB           *          JR      NC,LOOP     printed in the message
      000F 320000     U E*           LD      (@4),A      column due to lack of a
      0012                *          ENDM                parameter value.
0013: 0012                  ;
0014: 0012 *                         STRING  A,BC
      0012 414243         *          DEFM    'ABC'       Macro STRING is expan-
      0015                *          ENDM                ed.
0015: 0015 60               DIV:     DEFB    60H
0016: 0016 01               ANS:     DEFB    1
0017: 0017                           END
```

— IF, IFF, IFT, IFD, IFU /ENDIF —

The IF directives instruct the assembler whether or not the text following them is to be assembled. If the condition is met, the instructions between the IF directive and the END directive are assembled. Otherwise, they are ignored.

Following five IF directives are provided for setting different conditions.

IF operand	If the operand is zero, instructions following the IF directive are assembled. Otherwise, they are ignored.
IFF operand	The function is the same as IF. (IF False)
IFT operand	If the operand is not zero, instructions following the IFT directive are assembled. Otherwise, they are ignored. (IF True)
IFD operand	If the operand is defined, instructions following the IFD directive are assembled. Otherwise, they are ignored. (IF Defined)
IFU operand	If the operand is not defined, instructions following the IFU directive are assembled. Otherwise, they are ignored. (IF undefined) This directive is used in a macro.

When a label symbol is used as the operand of the IF, IFF, or IFT directive, it must be defined before the IF directive. That is, the value of the label symbol must be determined before the assembler reads the IF directive. It is also possible to use an expression in which a numeric is added or subtracted to/from a label symbol as the operand of the IF, IFF and IFT directives.

```
0001: 0000                ; SAMPLE#1
0002: 0000                ;
0003: 0000 (0000)         COND:  EQU    0
0004: 0000                       IF     COND
0005: 0000 86                    ADD    A,M       Assembled as COND = 0.
0006: 0001 12                    LD     (DE),A
0007: 0002                       ENDIF
0008: 0002                       IF     COND+1
0009: 0002                       SUB    M         Not assembled as COND + 1 ≠ 0.
0010: 0002                       LD     (DE),A
0011: 0002                       ENDIF
0012: 0002                       IFF    COND
0013: 0002 AE                    XOR    M         Assembled as COND = 0.
0014: 0003 12                    LD     (DE),A
0015: 0004                       ENDIF
0016: 0004                       IFT    COND
0017: 0004                       OR     M         Not assembled as COND = 0.
0018: 0004                       LD     (DE),A
0019: 0004                       ENDIF
0020: 0004                       END
```

```
0001: 0000                 ; SAMPLE#2
0002: 0000                 ;
0003: 0000               *          MACRO   PUSHN
0004: 0000               *          PUSH    @1
0005: 0000               *          PUSH    @2
0006: 0000               *          IFD     @3
0007: 0000               *          PUSH    @3
0008: 0000               *          ENDIF
0009: 0000               *          IFD     @4
0010: 0000               *          PUSH    @4
0011: 0000               *          ENDIF
0012: 0000               *          IFU     @4
0013: 0000               *          PUSH    IX
0014: 0000               *          PUSH    IY
0015: 0000               *          ENDIF
0016: 0000               *          ENDM
0017: 0000                 ;
0018: 0000 *                        PUSHN   BC,DE,HL
      0000 C5            *          PUSH    BC
      0001 D5            *          PUSH    DE
      0002               *          IFD     HL ┐
      0002 E5            *          PUSH    HL │ Assembled as parameter
      0003               *          ENDIF      ┘ 3 is defined.
      0003               *          IFD     @4 ┐ Not assembled as para-
      0003               *          ENDIF      ┘ meter 4 is undefined.
      0003               *          IFU     @4 ┐
      0003 DDE5          *          PUSH    IX │ Assembled as parameter
      0005 FDE5          *          PUSH    IY ┘ 4 is undefined.
      0007               *          ENDIF
      0007               *          ENDM
0019: 0007                 ;
0020: 0007                 ;
0021: 0007                          END
```

— DEFB n (DEFine Byte) —

This directive sets constant n (1-byte numeric) in the address of the line on which this dirctive is written. A label symbol equated to a 1-byte constant with the EQU directive may be used in place of n.

This directive as well as DEFW and DEFM described below is often used to generate message, graphic data, code conversion table, data table and so on.

The following example generates message ERROR in ASCII code with the DEFB directives.

```
0013: 1FF3 B7                          OR      A
0014: 1FF4 CA0000       E              JP      Z,READY
0015: 1FF7 110020                      LD      DE,MESGO
0016: 1FFA CD1500                      CALL    MSG
0017: 1FFD C30000       E              JP      MAIN1
0018: 2000 (0015)             MSG:     EQU     0015H
0019: 2000                    ;
0020: 2000                    ; MESSAGE GROUP
0021: 2000                    ;
0022: 2000                    MESGO:   ENT                              ; "ERROR"
0023: 2000 45                          DEFB    45H
0024: 2001 52                          DEFB    52H
0025: 2002 52                          DEFB    52H
0026: 2003 4F                          DEFB    4FH
0027: 2004 52                          DEFB    52H
0028: 2005 0D                          DEFB    0DH
0029: 2006                             END
```

— DEFB 'S', DEFB "S" (DEFine Byte) —

This directive sets the ASCII code corresponding to the character enclosed in single or double quotation marks in the address of the line on which this directive is written.

Since this directive converts characters to corresponding ASCII codes, the above program MESGO can also written as follows with this directive.

```
0021: 2000                    MESGO:   ENT                              ; "ERROR"
0022: 2000 45                          DEFB    'E'
0023: 2001 52                          DEFB    'R'
0024: 2002 52                          DEFB    'R'
0025: 2003 4F                          DEFB    'O'
0026: 2004 52                          DEFB    'R'
0027: 2005 0D                          DEFB    0DH
0028: 2006 27                 MESG1:   DEFB    "'"  ]  Please notice the way in
0029: 2007 22                          DEFB    '"'  ]  which single and double
0030: 2008                             END             quatation marks are us-
                                                       ed.
```

— DEFW nn' (DEFine Word) —

This directive set n' in the address of the line on which this directive is written and n in the following address; in other words, it sets two bytes of data. A label symbol equated to a 2-byte constant with the EQU directive may also be used in place of nn'.

```
0039: 5FF1                CMDT:    ENT                         ; COMMAND TABLE.
0040: 5FF1  41                     DEFB    41H
0041: 5FF2  0053                   DEFW    CMDA
0042: 5FF4  42                     DEFB    42H
0043: 5FF5  1E53                   DEFW    CMDB+3
0044: 5FF7  53                     DEFB    53H
0045: 5FF8  0000       E           DEFW    CMDS
0046: 5FFA  0D                     DEFB    0DH
0047: 5FFB                CONST0:  ENT
0048: 5FFB  0F01                   DEFW    010FH
0049: 5FFD                CONST1:  ENT
0050: 5FFD  660D                   DEFW    0D66H
0051: 5FFF                         END
```

— DEFM'S', DEFM "S" (DEFine Message) —

This directive sets the character string (S) enclosed in single or double quation marks in ASCII code in addresses starting at that of the line on which this directive is specified. The number of characters must be within the range from 1 to 16. On the assembly listing, codes for 4 characters are output on each line.

```
0022: 2000                 MESG0:  ENT                          ; "ERROR"
0023: 2000  4552524F               DEFM    'ERROR'
    : 2004  52
0024: 2005  0D                     DEFB    0DH
0025: 2006  41274227       MESG1:  DEFM    "A'B'C'"
    : 200A  4327
0026: 200C  0D                     DEFB    0DH
0027: 200D                         END
```

— DEFS nn' (DEFine Storage) —

This directive reserves nn' bytes of memory area starting at the address of the line on which this directive is written. That is, this directive adds nn' to the reference counter contents; the contents of addresses skipped are not defined.

```
0001: 4BB8             ;
0002: 4BB8             TEMP0:  ENT                          ; BUFFER A
0003: 4BB8                     DEFS    1
0004: 4BB9             TEMP1:  ENT                          ; BUFFER B
0005: 4BB9                     DEFS    2
0006: 4BBB             TEMP2:  ENT                          ; BUFFER C
0007: 4BBB                     DEFS    2
0008: 4BBD             TEMP3:  ENT                          ; BUFFER D
0009: 4BBD                     DEFS    128
0010: 4C3D             BFFR:   ENT                          ; BUFFER E
0011: 4C3D                     DEFS    0AH
0012: 4C47             BUFFER: ENT                          ; BUFFER F
0013: 4C47                     DEFS    2
0014: 4C49                     END
```

The addresses are increased by amounts corresponding to the values indicated by the respective DEFS statements.

— LIST, UNLIST —

These directives control output of the assembly list.

LIST	Outputs the assembly list following this directive. If neither LIST nor UNLIST is specified, this directive is executed implicitly.
UNLIST	Suppresses output of the assembly list following this drirctive.

The example below shows the text in the edit buffer and its assembly listing to illustrate the functions of the LIST and UNLIST directives.

```
;
;
GETCHR:CALL GET1C
LD B,A
RET
;
UNLIST
GETBUF:LD DE,BUFFER
CALL GET1L
 RET
 ;
 LIST
 PUTBUF:LD DE,BUFFER
 CALL PUT1L
 RET
 END
```

Actual text in the edit buffer

```
0001: 0000                 ;
0002: 0000                 ;
0003: 0000 CD0000     E    GETCHR: CALL    GET1C
0004: 0003 47                      LD      B,A
0005: 0004 C9                      RET
0006: 0005                 ;
0007: 0005                         UNLIST
0013: 000C 110000     E    PUTBUF: LD      DE,BUFFER
0014: 000F CD0000     E            CALL    PUT1L
0015: 0012 C9                      RET
0016: 0013                         END
```

Assembly listing of the text above.
Although the assembly listing of the GETBUF routine following the UNLIST directive are not output, discontinuous change of the address indicates that the GETBUF routine is assembled.

— SKP n (SKiP n lines) —

This directive feeds n lines and leaves a space between the preceding and following parts of the listing to make the listing easy to read.

```
0030: 3BB8                COMMON: ENT                      ;NORMAL RETURN
0031: 3BB8 AF                     XOR     A                ; A --00
0032: 3BB9 320000     E           LD      (TEMPO),A        ; CLEAR CMD BUFFER
0033: 3BBC 110000     E           LD      DE,MESGO         ; "READY"
0034: 3BBF C9                     RET
0035: 3BC0                        SKP     3
                                                 3 line feeds are made.

0036: 3BC0                ;
0037: 3BC0                ; ABNORMAL RETURN
0038: 3BC0                ;
0039: 3BC0                ABNRET: ENT                      ; SET INVALID MOOD
0040: 3BC0                        END
```

— SKP H (SKiP Home) —

This directive starts a new page.

— END (end) —

This disective marks the end of the source program. All source programs must be ended with this directive.Assembly operation is not completed if this directive is missing. The assember outputs

END?

when it reads a source file which does not include an END directive.

2.4 ERROR MESSAGES

The monitor and editor-assembler detect errors. Errors detectd by the text-editor are indicated with sings (–) preceding the error messages. Those detected by the assembler are output in the message column of the assembly list.

2.4.1 Monitor error messages

Error message	Meaning
System in	The type of the system disk is wrong.
File not found	The specified file was not found.
Hardware	An error occurred in the device's hareware.
Already exist	A file with the same name already exists.
Already open	The file is already opened.
Not open	An attempt was made to referece a file not yet opened.
Write protect	The file or device is write-protected.
Not ready	The disk drive is not ready.
Too many files	The number of files exceeds 32.
No file space	The disk space is insufficient to store the file.
Unformat	The disk is not formatted (initalized).
Dev. name	The device name is wrong.
Can't execute	An attempt was made to make the device execute impossible operation.
Illegal filename	The file name is wrong.
Illegal filemode	The file mode is wrong.
LPT: not ready	The printer is not connected.
Check sum	Check sum error (cassette tape read error)

2.4.2 Text editor error messages

Error message	Meaning	Relevant commands
Full buffer	Edit buffer is full.	R, A
? ? ?	A negative number is specified for n of nT, nJ and so on.	T, J
Large	n greater than 65535 is specified.	T, J, L, M, K, D, B, Z
Not found	The string (or string1) specified in String, Cstring1 ▩ string 2, or Qstring1 ▩ string2 was not found following the CP.	S, C, Q
Invalid	An illegal command was entered or an incorrect format was used. Ex. *H CR : There is no H command. *S CR : A string searched for is not specified.	Any cases
Not same	The contents of the edit buffer and that of the specified file are different.	V
Bad command	The format of the file control command entered is incorrect.	\ preceded commands

2.4.3 Assembler error messages

Definition status message	Meaning	Example
E (External)	Indicates that a label symbol is being referenced extrnally; that is, the label is not defined in the current source program unit.	E LD B, CONST0 The data byte CONST0 is undefined. E CALL SORT The address SORT is undefined. EE BIT TOP, (IY+FLAG) The data byte FLAG is undefined. The data byte TOP is undefined.

Error message	Meaning	Example
C (illegal Character errer)	Indicates that illegal characters are used in the operand.	
F (Format error)	Indicates that the instruction format is incorrect.	
N (Non label symbol)	Indicates that no label symbol is specified for ENT or EQU.	N EQU 0012H No label symbol
L (erroneous Label error)	Indicates that an illegal label symbol is specified.	L JR XYZ XYZ is not defined in the current program. No external symbol can be referenced in the JR and DJNZ instructions. If such a label symbol is specified, the L message is displayed.
M (Multiple label error)	Indicates that the label symbol or macro name is defined two or more times.	M ABC: LD DE, BUFFER M ABC: END ABC is defined twice.
O (erroneous Operand)	Indicates that an illegal operand is specified.	O JP +100 −ABC
Q (Questionable mnemonic)	Indicates that the nmemonic code is incorrect.	Q CAL XYZ CALL XYZ is correct.
S (String error)	Indicates that single or double quation marks are missing.	S DEFM GAME OVER DEFM 'GAME OVER' is correct.
V (Value over)	Indicates that the value of the operand is out of the prescribed range.	V LD A, FF8H V SET 8,A V JR −130
U (Undefined parameter)	Indicates that the value required are not specified in the macro call instruction or the operand of the IF directive is not defined.	U JR Z, @3 U IF ABC
END?	Indicates that the END directive is missing from the source program.	
Pass-1 error	Assemble is aborted during the pass-1 processing.	• The operand of ORG, EQU, DEFS, IF, IFT or IFF was not defined. • Two or more macros were defined under the same label name.

The following messages are output in a symbol table.

:	Indicates that the symbol is a label defined with the ENT directive.
=	Indicates that the symbol is a label defined with the EQU directive.
M (Multi-defined)	Indicates that the symbol is defined two or more times.
U (Undifine)	Indicates that the symbol is not defined.