Z80 DEVELOPMENT SYSTEM

Distributed by:

Lifeboat Associates
1651 Third Avenue
New York
N.Y. 10028
(212) 860-0300

Lifeboat Associates Z80 Development System

CONTENTS

SD SYSTEMS TEXT EDITOR V3.0
OPERATIONS MANUAL

1.0 INTRODUCTION

The SD SYSTEMS Text Editor assists the user in origination and modification of assembly language source programs and English text documentation. The Editor resides on a 32K system diskette. It permits random access editing of ASCII diskette files. The Editor is designed for usage with any CP/M(*) compatible disk operating system (DOS) using a Z80 microprocessor.

The Text Editor permits random access editing of ASCII diskette files on a line basis or character basis. Whole lines and character strings embedded within lines can be easily accessed, changed, deleted, or added to an existing or new diskette file. The size of the file to be edited is limited only by diskette capacity. All I/O operations to the diskette are transparent to the user.

The Editor is resident on diskette. When loaded, it starts at RAM address 100H. Editor buffers and variables are placed in RAM between the top of the Editor and the bottom of the Operating System. All I/O is done with the console device and the disk.

2.0 DEFINITIONS

SOURCE - ASCII characters comprising a Z80 assembly language program or some other text.

FILE - a diskette file which contains the SOURCE.

LINE - a single source statement which ends with a carriage return.

LINE POINTER - the position in the source where the next action of the Editor will be initiated.

----
*) CP/M is a registered trademark of Digital Research of Pacific Grove, California.

CURRENT LINE - the line in the source pointed to by the LINE POINTER.

LINE NUMBER - the decimal number of a line, beginning at one (0001) for the first line in a file and increasing sequentially for each line. The maximum line number allowed is 9999 (decimal). Line numbers are assigned dynamically as editing of the file progresses. This means that when lines are added to or deleted from a file, all lines are automatically renumbered.

INSERT - installation of one or more lines in a file immediately following the current line. Inserted lines are assigned sequentially increasing line numbers.

DELETE - removal of one or more lines from a file.

3.0 USING THE EDITOR - CONSOLE INTERACTION

All user interaction with the Editor is via the user console. The Editor issues prompts and messages to direct the user. The user responds by entering commands or data via the console keyboard. Each command or data line is terminated by a carriage return.

The following conventions are used in this manual:

> (CR) stands for carriage return.
> All user input is underlined.
> User input which must be entered exactly as shown
> is in upper case letters.
> User input which is variable is shown in lower case.

4.0 USING THE EDITOR - ENTERING COMMANDS

The Editor prompts for a command with an asterisk (*). The user may then enter commands via the console keyboard. Modification of the input, such as rubout, backspace, and line delete. is supported by the operating system. entered in lower case as well as upper case. Several commands may be entered on one line. Blanks and commas are ignored on input. A command line is terminated by a carriage return. A command line may have up to 80 characters in it, including the carriage return.

All commands consist of one character followed by an optional operand. The operand may be separated from the command by zero or more blanks or commas. The operand may be a decimal number in the range 0-9999. This specifies the number of lines upon which the command is to operate. Alternatively, the operand may be two decimal numbers separated by a minus sign (-). In this case, the command takes effect on lines numbered from the first number in the operand through and including the second number. If the operand is not entered, it assumes a value of one (except for the 'F' command).

EXAMPLE

```
        V
                -VIEW command with no operand.  The operand
                value assumes the value of one.
        V5  or   V 5    or    V,5
                -one operand shown which acts on the next 5
                lines in the source file.
        V42-45   or  V 42-45    or    V,42-45
                -two operands entered.  The VIEW command acts
                on lines numbered 42 through 45.
```

5.0 USING THE EDITOR - FIRST STEPS

After booting up DOS, the Text Editor may be executed by the following command:

```
        A>EDIT filename(CR)
          -----------------
                -where filename is the name of the diskette file
                to be edited on the currently selected disk.
                The file may not have an extension of COM, BAK,
                or $$$.
```

The Editor responds with the following message:

```
        SD SYSTEMS EDITOR
```

EXAMPLE

```
        A>EDIT MYFILE(CR)
          ---------------
                -user selects to run the Editor to edit the file
                named 'MYFILE' on the currently selected disk.
        SD SYSTEMS EDITOR
```

If the file does NOT exist on the diskette, then the Editor outputs the following message on the console:

```
        ***NEW FILE
                -Editor indicates that a new file is being created.
```

The Editor then enters the 'DATA MODE' and waits for lines of data to be entered by the user:

```
        ***DATA MODE
        0001
                -Editor prompts for data lines starting with
                line number 0001 (see I-INSERT command).
```

At the end of editing, the new file will automatically be created.

If the file does exist on the disk, then editing of that file will be done, and the Editor prompts for a command:

```
        *
                -Editor prompts for a command (see below).
```

At the end of editing, the original file will be renamed with an extension of 'BAK'. The file which was edited will have all the changes in it.

The following pages describe each of the Editor commands in detail.

6.0 EDITOR COMMANDS

6.1 An - ADVANCE

```
Format:
        An
or
        an
                -where n is a decimal number.
```

This command is used to advance the line pointer (toward the end of the file) a specified number of lines. If the operand n is not entered or it is zero, then the pointer will be position to the next line in the file. The line which is accessed is printed on the console after this command.

```
EXAMPLE
        *A(CR)
         -----
                -user advances to next line.
        0015 ANY STATEMENT.
                -the next line with its line number is printed on
                the console.
        *A5(CR)
         ------
                -user advances 5 lines from current pointer.
        0020 SOME STATEMENT
                -Editor prints line number and the line.
        *
                -Editor prompts for a command.
```

If the user attempts to advance the line pointer beyond the end of the file, then an end-of-file indicator message wi'l be printed on the user console. The line pointer will be on the last line of the file.

EXAMPLE

```
*A9999(CR)
 ---------
          -user advances over a large number of lines.
0438 LAST LINE OF FILE
***EOF
          -Editor prints last line of file and end-of-file
          indicator.
*
          -Editor prompts for a command.
```

6.2 Bn - BACKUP

Format:

```
          Bn
or
          bn
                    -where n is a decimal number.
```

This command is used to backup the line pointer (toward the beginning of the file) a specified number of lines. If the operand n is zero or it is not entered, then the pointer is positioned to the previous line in the file. The line which is accessed is printed on the console.

EXAMPLE

```
*B(CR)
 -----
          -user backs up over one line in the file.
0019 A LINE OF INFORMATION
          -Editor prints the line number and the line.
*B4(CR)
 ------
          -user backs up 4 lines from current position.
0015 SOME LINE
          -Editor prints the line number and the line.
*
          -Editor prompts for a command.
```

If the user attempts to back up the line pointer past the start of the file, then a top-of-file indicator will be printed on the user console. The line pointer will be on line number 0001.

EXAMPLE

```
*B9999(CR)
 ---------
          -user backs up over a large number of lines.
***TOF
0001 FIRST LINE OF FILE
          -Editor prints top-of-file indicator and first
          line of the file.
*
          -Editor prompts for a command.
```

6.3 Cn /string1/string2/ - CHANGE STRING

Format:

```
        Cn/string1/string2/
or
        cn/string1/string2/
```

-where n indicates the number of occurrences to change, string1 represents the characters to be changed, string2 represents the substitute or new characters, and / represents a delimiter character which does not appear in either string.

This command changes the next n occurrences of character string1 to character string2 starting with the current line. Any character which does not appear in either string1 or string2 may be used as a delimiter. All three delimiters must be identical, with the exception that the last delimiter may be a carriage return. If the operand is zero or if it is not entered, then only one occurrence of string1 will be changed. In this case, only the current line will be searched in order to locate string1. If string1 is not found in the current line, then the Editor issues a warning prompt ('?') and a new command prompt (*). The line pointer will stay on the same line.

If the operand n is greater than 1, then the search for occurrences of string1 occurs in a sequential manner starting with the current line. Each line which is changed is printed on the user console. After all changes are done, the line pointer will be on the last line that was changed. If the nth occurrence of string1 is not found before the end of the file is encountered, then the last line of the file is printed by the Editor, as well as an end of file indicator (***EOF). The line pointer will be on the last line of the file.

If string2 has no characters in it, then character string1 will be deleted each time it is encountered by the change command.

EXAMPLES

```
        *V(CR)
         -----
                -user views current line.
        0009 THIS IS A RECORD
                -Editor prints line number and line.
        *C /THIS/THAT/(CR)
         ----------------
                -user enters change command.
        0009 THAT IS A RECORD
                -Editor prints it.
        *C/IS/WAS(CR)
         -----------
                -note that a carriage return for the last delimiter
                is allowed.
```

```
0009 THAT WAS A RECORD
*C /WAS //(CR)
 ------------
```

-this is the method used to delete characters.

```
0009 THAT A RECORD
*C 2 /T/V/(CR)
 -------------
```

-note that blanks can be inserted between the command and operand and string definition to make the command more readable.

```
0009 VHAV A RECORD
*C4/VHAV/THAT/(CR)
 -----------------
```

-this is a multiple change request which will search forward in the file starting with the current line.

```
0009 THAT A RECORD
0024 SOME TIMES THAT IS
```

-Editor prints out each line that is changed.

```
0043 LAST LINE OF FILE
***EOF
```

-Editor reached the end of the file before any more changes could be done. An end-of-file indicator message is printed. The line pointer is on the last line in the file.

```
*
```

-Editor prompts for a command.

6.4 Dn - DELETE

Format:

```
          Dn
or
          dn
or
          Dn-m
or
          dn-m
```

-where n and m are decimal numbers.

This command is used to delete, or remove, the specified lines from the file. If the operand is not entered or is zero, then only the current line is deleted. Note that line numbers are assigned dynamically as editing progresses. This means that lines in a file essentially get renumbered each time one or more lines are deleted from the file.

EXAMPLE

```
*D(CR)
 -----
```

-user deletes the current line from the file. The line pointer is on the next line in the file.

```
*D4(CR)
 ------
            -user selects to delete 4 lines starting with
            the current line from the file.
*D4-15(CR)
 ---------
            -user deletes lines numbered 0004 through and including
            0015 from the current file.
*
            -Editor prompts for a command.
```

6.5 En - EXCHANGE

Format:

```
        En
or
        en
or
        En-m
or
        en-m
                -where n and m are decimal numbers.
```

This command exchanges the specified lines with new lines to be inserted via the DATA MODE. It is exactly equivalent to the command sequence:

```
        Dn                  -delete lines
        B                   -backup one line
        I                   -go to DATA MODE
```

6.6 Fn - PRINT FLAG

Format:

```
        Fn
or
        fn
                -where n=0 will inhibit printing after all but
                the V-VIEW command, and n not = 0 will allow
                printing after all change or access commands.
```

The Editor normally prints on the console device any lines which are accessed or changed. Thus, the following commands print out a line: An, Bn, Cn, Ln, Sn, Vn. In order to reduce the print out time on a slower device (such as a teletype), this command can be used to inhibit print out on all of the commands except V-VIEW.

6.7 G file - GET FILE COMMAND

```
Format:
        G filename
or
        g filename
                -where filename is the name of a file on the
                selected disk.
```

This command is used to obtain lines from a given file on disk and insert them in sequence following the current line. All lines in the file requested are read. The file which is read is not altered in any way. This command can be used with the P - PUT command to move blocks of text around within a file being edited (See P - PUT command for example).

6.8 I - INSERT COMMAND

```
Format:
        I
or
        i
```

This command is used to insert data lines into the file being edited or to build new files. The inserted lines always FOLLOW the current line. After the command is entered, the Editor responds with the message:

```
        ***DATA MODE
```

The user then enters data lines ending with carriage returns. The Editor prompts with the line number for each line to be inserted. To terminate the insertions, the user enters a single carriage return. Note that blank lines must be entered as 'space carriage return' because a single carriage return terminates the DATA MODE. After the user terminates the DATA MODE, the Editor prompts for a new command (*). Lines can be inserted before the first line of a file by using the T - INSERT AT TOP command. Note that line numbers are assigned dynamically while editing progresses. This means that the lines of a file essentially get renumbered whenever new lines are inserted.

```
EXAMPLE
        *I(CR)
         -----
                -user selects data mode to insert lines into the file
                being edited.
        ***DATA MODE
                -Editor responds with message
        0004 THIS IS AN INSERTED LINE.(CR)
             -----------------------------
                -the line number being entered is printed by the
```

```
                    Editor.  The user then enters the line of data.
          0005 (CR)
               ----
                    -user terminates DATA MODE with a carriage return.
          *
                    -Editor prompts for another command.
```

Note that modification of entered data lines can be done while they are being typed just as in the DOS system. Inserted lines can be up to 128 characters long.

6.9 Ln - GO TO LINE NUMBER n

Format:

```
        Ln
or
        ln
```

This command positions the line pointer to line number n. If the operand is zero or it is not entered, then line number 0001 is accessed. Any line number can be accessed from any position in the file. The line which is accessed is printed on the console. If the line number cannot be found because it is larger than the last line number in the file, then the pointer will be positioned at the last line in the file and an end-of-file indicator message will be printed.

EXAMPLE

```
        *L10(CR)
         -------
                -user accesses line number 0010.
        0010 THIS IS A LINE OF DATA
                -line number 0010 is printed with its line number.
        *L2001(CR)
         ---------
                -user selects line number 2001.
        0943 LAST LINE OF FILE
                -Editor prints last line of file.
        ***EOF
                -Editor prints an end-of-file indication.
        *L1(CR)
         ------
                -user selects line number 1.
        ***TOF
        0001 FIRST LINE OF FILE
                -Editor responds with top of file indicator
                and first line of file and its line number.
        *
                -Editor prompts for a command.
```

6.10 Pn file - PUT

Format:

```
        Pn filename
or
        pn filename
or
        Pn-m filename
or
        pn-m filename
                -where n and m are decimal numbers and filename
                is the name of a file on the selected disk.
```

This command is used to output one or more lines to a file on disk. This can be used to break up a given source module. It can also be used with the G - GET command to move blocks of text around in a file being edited. If the operand is not entered or it is zero, then only one line will be output. Lines of text which are output by the PUT command are not deleted. They may be deleted via the D - DELETE command after the PUT command is used. The filename specified must not be the same as the current file being edited. If the file already exists on disk, it is erased before any lines are output to it. After the PUT command is used, the file output to the disk remains on the disk.

EXAMPLE

```
        *P25-30 TEMP(CR)
         ---------------
                -user outputs lines 25 through 30 to a file
                called TEMP.  The space between the number and
                the filename is not required.
        *D25-30(CR)
         ----------
                -user deletes lines 25 through 30 from file
                being edited.
        *L1(CR)
         ------
                -user accesses line number one.
        *G TEMP(CR)
         ----------
                -user reads lines from file TEMP and places them
                after line number one.  This effectively moves
                lines 25-30 to just after line 1.  The space
                between the command and the filename is not
                required.
        *
                -Editor prompts for a command
```

6.11 Q - QUIT

Format:

```
        Q
```

or

```
        q
```

This command returns control to the Operating System. The original file will be backed up on the same primary filename with a secondary filename of BAK. All of the editing will be saved in the file under the original file name. QUIT can be done at any time during the course of editing.

6.12 Sn /string/ - SEARCH FOR STRING

Format:

```
        Sn /string/
or
        sn /string/
                -where n is the number of occurrences to be found.
                string represents any set of characters which is to
                be searched for, and / represents a delimiter character
                which does not appear in the string.
```

This command searches the file, starting with the NEXT line, for n occurrences of the character string between the delimiters. Each line which contains the string will be printed on the console. The pointer is positioned on the line of the nth occurrence of the string. If the nth occurrence of the string cannot be found before the end of the file being edited, then the Editor issues an end-of-file indicator (***EOF). This command always searches forward (toward increasing line numbers) in the file.

Any character which does not exist in the string to be searched for may be used as a delimiter. The second delimiter may be a carriage return. If the operand n is zero or it is not entered, then only the first occurrence of the string will be sought.

EXAMPLE

```
        *S /ORD/(CR)
         -----------
                -user selects to search forward in the file, beginning
                with the next line, for the string 'ORD'. Only the
                first occurrence of the string is sought. The blank
                between the command and the string is not required.
        0021 SOME ORDERLY DATA
                -Editor prints the line number and line when the
                string is found. The line pointer is on line 0021.
        *S10/9AH/(CR)
         ------------
                -user selects to search for and view the next
                10 occurrences of the string '9AH'.
        ***EOF
                -Editor encountered the end of the file and found
                no occurrences of the string. The end-of-file
```

```
                    indicator is printed.
        *
                    -Editor prompts for a command.
```

6.13 T - INSERT AT TOP

```
Format:
        T
or
        t
```

This command inserts data lines at the top (start) of the file BEFORE the first line in the file. See the I-INSERT command for proper usage.

6.14 Vn - VIEW

```
Format:
        Vn
or
        vn
or
        Vn-m
or
        vn-m
                -where n and m are decimal numbers.
```

This command prints the specified lines on the console device. The line pointer is updated to the last line printed. If the operand n is zero or is not entered, then only the current line is printed.

EXAMPLE

```
        *V(CR)
         -----
                -user views current line on the console.
        0009 THIS IS A LINE
                -Editor prints line number and line.
        *V3(CR)
         ------
                -user views current line plus two more.
        0009 THIS IS A LINE
        0010 THIS IS NEXT LINE
        0011 THIS IS ANOTHER LINE
                -Editor prints 3 lines on the console.  The
                line pointer now points to line 0011.
        *V3-4(CR)
         --------
                -user selects to view lines 3 through 4.
        0003 SOME LINE OF DATA
        0004 NEXT LINE OF DATA
                -Editor prints lines 3 through 4.
        *
                -Editor prompts for a command.
```

7.0 EDITING LARGE FILES

Editing of large files is no different than editing small files. All commands are fully functional. However, diskette access may be required for certain operations and a slight delay may be apparent before the Editor responds.

8.0 EDITOR MESSAGES

If the user enters an unrecognizable file name, a syntax error will be indicated and the Editor will return to DOS. If the user enters an unrecognizable command, then the Editor will print a question mark and another command prompt:

```
EXAMPLE
          *R20(CR)
           -------
          ?*
```

All I/O errors to and from disk result in appropriate error messages. The original file should be backed up on another disk before using the Editor.

The Editor prompts with several other messages as editing progresses:

***NEW FILE - indicates that a new file is being created rather than editing of an already existing file.

***DATA MODE - indicates that lines of data are to be entered rather than Editor commands.

***TOF - indicates that the top of file (beginning of file) has been encountered.

***EOF - indicates that the end of file has been encountered.

***END OF EDITING - indicates that the Editor has successfully completed. Control is then returned to the DOS Operating System.

***END OF WINDOW. USE 'ADVANCE' TO SEE NEXT LINE - occurs only with the VIEW command. Follow the directions.

9.0 SAMPLE EDITING SESSION

The user is urged to follow the steps given here to become acquainted with the Editor.

```
>EDIT NEWFILE(CR)
 ---------------
                - user selects to run the Editor to create a new file.
SD SYSTEMS EDITOR V1.0
***NEW FILE
                -Editor indicates that a new file is being created.
***DATA MODE
                - Editor prompts for data lines to be input from
                  the console device.  User begins keying in a
                  program.
0001 ; A SIMPLE SAMPLE PROGRAM(CR)
     -----------------------------
0002  LD A,(LAB1)(CR)
     ----------------
0003  LD E,0(CR)
     -----------
0004  CALL SUB1 ;SOME COMMENT(CR)
     ---------------------------
0005 LOOP LD (HL),0 ;STUFF ZEROS(CR)
     -------------------------------
0006  INC HL(CR)
     -----------
0007  DNZ LOOP-$ ;LOOP FOR ALL(CR)
     -----------------------------
0008  END(CR)
     --------
0009 (CR)
     ----
                -user terminates DATA MODE.
*B99V20(CR)
 ----------
                -user backs up to beginning of file and
                 views all lines.
.
.
.
***EOF
                -Editor indicates end of file encountered.
*L7(CR)
 ------
0007  DNZ LOOP-$ ;LOOP FOR ALL
                -user views line 7 and observes an error.
*C /DN/DJN/(CR)
 --------------
                -user modifies the line.
0007  DJNZ LOOP-$ ;LOOP FOR ALL
```

```
                  -Editor prints the changed line
        *Q(CR)
         -----
                  -user terminates the editing session.
```

10.0 EDITOR COMMAND SUMMARY

An	advance n lines
Bn	backup n lines
Cn/sl/s2/	change n occurrences of sl to s2
Dn	delete n lines
En	exchange n lines
Fn	turn on or turn off print flag
G file	get file and insert into current file
I	insert lines of data
Ln	go to line number n
Pn file	put n lines out to file
Q	quit, save all editing and return to DOS
Sn/sl/	search for n occurrences of sl
T	insert lines at top of file before first line
Vr	view n lines on the console

SD SYSTEMS RELOCATING Z80 ASSEMBLER VERSION 3.2
OPERATIONS MANUAL

1.0 INTRODUCTION

The SD SYSTEMS Z80 ASSEMBLER is provided on a standard CP/M compatible diskette. It provides the means for assembling Z80 programs. The Assembler (ZASM) reads standard Z80 source language (Mostek and Zilog definition) and outputs an assembly listing and object code on disk. The object code is in industry standard hexadecimal format extended for relocatable and linkable programs. The Assembler supports conditional assembly, a printed symbol table, and a printed cross reference table. The Assembler can assemble any length program limited only by the symbol table size which is based on available memory and available disk space. Typically over 300 symbols are allowed in one assembly.

Any Z80 based system which is running 32K CP/M compatible disk operating system (DOS) can use the Z80 Assembler.

2.0 COMMAND SUMMARY

```
ZASM file.ext(CR)
        - executes assembler to assemble a file
        - object output is on file.OBJ
        - listing output is on file.PRN
```

OPTIONS

```
C - print cross reference table
K - no listing output
L - direct assembly listing out to listing device
N - no object output
P - pass 2 only
R - reset symbol table for pass 2 only operation
S - print symbol table
T - direct assembly listing out to console device
```

3.0 DEFINITIONS

In this manual, the following symbols are used:

- (CR) means carriage return.
- all user input is underlined.
- use: input which is all upper case must be entered exactly as shown.
- user input which is lower case is variable.

SOURCE MODULE - the user's source program. Each source module is assembled into one object module by the Assembler. The end of a source module is defined by an 'END' statement or CP/M end of file code (1AH) on input.

OBJECT MODULE - the object output of the Assembler for one source module. The object module contains linking information, address and relocating information, machine code, and checksum information for use by the SD SYSTEMS Linker. The object module is in ASCII. The object module is output to a disk file with extension OBJ. The SD SYSTEMS Linker must then be used to link and relocate one or more object modules into a module loadable by the DOS. See the SD SYSTEMS Linkder Operations Manual for more details.

LOAD MODULE - the absolute machine code of one complete program. The load module is defined on disk as an absolute object file with extension HEX. The file may be loaded by the DOS loader. It is created by the SD SYSTEMS Linker from one or more relocatable object modules (secondary file name OBJ) which were created by the Z80 Assembler.

LOCAL SYMBOL - a symbol in a source module which appears in the label field of a source statement.

INTERNAL SYMBOL - a symbol in a source (and object) module which is to be made known to all other modules which are linked with it by the Linker. An internal symbol is also called global, defined, public, or common. Internal symbols are defined by the GLOBAL pseudo-op. An internal symbol must appear in the label field of the same source module. Internal symbols are assumed to be addresses, not constants, and they will be relocated when linked by the Linker.

EXTERNAL SYMBOL - a symbol which is used in a source (and object) module but which is not a local symbol (does not appear in the label field of a statement). External symbols are defined by the GLOBAL pseudo-op. External symbols may not appear in an expression which uses operators. An external symbol is a reference to a symbol that exists and is defined as internal in another program module.

GLOBAL DEFINITION - both internal and external symbols are defined as GLOBAL in a source module. The Assembler determines which are internal and which are external.

POSITION INDEPENDENT - a program which can be placed anywhere in memory. It does not require relocating information in the object module.

ABSOLUTE - a program which has no relocating information in the object module. An absolute program which is not position independent can be loaded only in one place in memory in order to work properly.

RELOCATABLE - a program which has extra information in the object module which allows the Linker to place the program anywhere in memory.

LINKABLE - a program which has extra information in the object module which defines internal and external symbols. The Linker uses the information to connect, resolve, or link, external references to internal symbols.

4.0 USING THE ASSEMBLER

The SD SYSTEMS Z80 ASSEMBLER is resident on a CP/M compatible system diskette. The user first prepares his source module using the SD SYSTEMS Editor. To use the Z80 Assembler, enter the following command:

```
        A>ZASM file.ext /xyz(CR)
          ----------------------
```

where 'file' is the primary file name and 'ext' is the secondary file name of the file to be assembled and x,y and z are options described in paragraph 4.1. If the slash (/) is included in the command, with or without options, the option prompt will be skipped by the assembler. This is a useful feature when using the system under a SUBMIT batch skript.

The object output of the Assembler is sent to the disk on file.OBJ, and the listing output is sent to the disk on file.PRN. One or more object files from the Assembler may be linked and relocated by using the SD SYSTEMS Linker, which produces an absolute object file with extension HEX. The absolute object file may then be loaded via the DOS loader, and the listing file may be printed using PIP. If no options are selected in the initiating command line, the assembler will request them for the console with the following prompt:

```
        SD SYSTEMS Z80 ASSEMBLER V3.2. OPTIONS?
```

If no options are to be entered, the user enters 'carriage return'. The Assembler makes two passes over the source file. At the end of the first pass the following message is printed on the user console:

```
        PASS 1 DONE
```

At the end of the assembly, the Assembler prints the total number of errors (in decimal) found:

```
        ERRORS=nnnn
```

Control is then returned to the DOS console processor (A>).

4.1 ASSEMBLER OPTIONS

When the Assembler outputs the message:

```
OPTIONS?
```

the user may enter any of the following codes, terminated with a carriage return:

- C - cross reference table - prints a cross reference table of all the symbols at the end of the assembly listing.
- K - no listing - this suppresses the assembly listing. All errors are output to the user console for this option.
- L - list to listing device - this option directs the assembly listing out to the listing device rather than to a disk file.
- N - no object output - this suppresses object output from the Assembler.
- P - pass 2 only - this option selects and runs only pass 2 of the Assembler. The symbol table is left intact from a previous run of the Assembler.
- R - reset the symbol table - clears the symbol table of all previous symbol references. This operation is automatically done for pass 1. It is used primariy for single pass operation (see paragraph 5.1).
- S - symbol table - prints a symbol table at the end of the assembly listing.
- T - list to console device - this option directs the assembly listing out to the console device rather than to a disk file.

4.2 ERROR MESSAGES

Any error which is found is denoted in the assembly listing. A message is printed immediately after the statement in error. All messages are self-explanatory.

EXAMPLE

```
H2:  LC  A,B
***** ERROR ***** BAD OPCODE
```

Certain errors abort the Assembler when they are encountered. Abort error messages are output only to the user console. Control is immediately returned to the DOS console processor (A>). Abort errors may occur during pass 1 or pass 2.

4.3 OBJECT OUTPUT

The object output from the Assembler is put on diskette to the same primary file name as the source input file, with a secondary file name of 'OBJ'. One or more object modules may be linked and relocated by the SD SYSTEMS Linker to produce an absolute object file with a secondary file name of 'HEX'. This file may then be loaded by the DOS loader.

4.4 ASSEMBLY LISTING OUTPUT

The assembly listing is put on diskette to the same primary file name as the source input file, with a secondary file name of 'PRN'. The user may insert tab characters in the source to obtain columns in the assembly listing. The value of each equated symbol will be printed with a pointer (>) next to it. The statement number and page number are printed in decimal. Assembler directives (see paragraph 6.4.1.) do not appear in the assembly listing, but they are assigned statement numbers. If the no listing option is selected, errors will be output to the user console. Any addresses which are relocatable will have a prime (') printed next to them.

5.0 ADVANCED OPERATIONS

5.1 PASS 2 OPERATION (SINGLE PASS OPERATION)

The Z80 Assembler can be used as a single pass assembler under the following restrictions:

1. No forward symbol references are allowed.
2. The NAME pseudo-op is not allowed.
3. A cross reference table is not selected.

The Assembler will correctly assemble Z80 programs under the above restrictions using the pass 2 only option ('P'). This is useful for assembling data tables and certain types of programs. The Assembler symbol table should be reset to assure proper operation in this mode by using the 'R' option.

5.2 ASSEMBLING SEVERAL SOURCE MODULES TOGETHER

Several source modules may be assembled together to form one object module. The 'INCLUDE' pseudo-op may be used any number of times in one module to properly sequence a set of source modules.

```
EXAMPLE         NAME    MYFILE  ;name of final object module
                INCLUDE FILE1
                INCLUDE FILE2
                INCLUDE FILE3
                END
                        - the object module named 'MYFILE' will be
                          built by the Assembler from FILE1 + FILE2
                          + FILE3.
```

6.0 Z80 ASSEMBLY LANGUAGE

An assembly language program (source module) consists of labels, opcodes, pseudo-ops, and comments in a sequence which defines the user's program. The assembly language conventions are described in the following paragraphs.

6.1 DELIMITERS

Labels, opcodes, operands, and pseudo-ops must be separated from each other by one or more commas, spaces, or tab characters (ASCII 09H). The label may be separated from the opcode by a colon, only, if desired.

6.2 LABELS

A label is composed of one or more characters. If more than 6 characters are used for the label, only the first 6 are recognized by the Assembler. The characters in the label cannot include ' ( ) * + - / , = < > . : ; or space. In addition, the first character cannot be a number (0-9). A label can start in any column if immediately followed by a colon (:). It does not require a colon if started in column one.

EXAMPLE	allowed	not allowed	
	LAB	9LAB	;STARTS WITH ILLEGAL CHARACTER
	L923	L)AB	;CONTAINS ILLEGAL CHARACTER
	$25	L:ABC	;CONTAINS ILLEGAL CHARACTER

6.3 OPCODES

The full set of Z80 opcodes is documented in the 'Z80 PROGRAMMING MANUAL' (which is available from SD SYSTEMS).

6.4 PSEUDO-OPS

Pseudo-ops are used to define assembly time parameters. Pseudo-ops appear like Z80 opcodes in the source module. Several pseudo-ops require a label. The following pseudo-ops are recognized by the Assembler:

DEFB n,n,n... - define byte - defines the contents of successive bytes to be the expressions n.

label DEFL nn - define label - sets the value of the label to the expression nn; may be repeated in the program with different values for the same label. At any point in the program, the label assumes the last previously defined value.

DEFM 'aa' - define message - defines the contents of successive bytes of memory to be the ASCII equivalent code of characters within quotes. Up to 63 characters may be in one message. Quote characters in the message may be defined by two successive quote characters ('').

DEFS nn - define storage - reserves nn bytes of memory starting at the current program counter, where nn is an expression. When loaded, these bytes will contain what was previously in memory. This pseudo-op cannot be used at the start or at the end of a program to reserve storage.

DEFW nn,nn,nn... - define word - defines the contents of successive two-byte words to be the value of expressions nn. The least significant byte is located at the current program counter address, and the most significant byte follows it.

END - end statement - defines the last line of the program. The 'END' statement is not required.

ENDIF - end of conditional assembly - re-enables assembly of subsequent statements after an IF pseudo-op.

label EQU nn - equate - sets the value of a label to the expression nn; can occur only once for any label.

GLOBAL symbol - define global symbol - any symbol which is to be made known among several separately assembled modules must appear in this type of statement. The Assembler determines if the symbol is internal (defined as a label in the program), or external (used in the program but not defined as a label).

IF nn - conditional assembly - if the expression nn is true (non-zero), the IF pseudo-op is ignored. If the expression is false (zero), the assembly of subsequent statements is disabled until an ENDIF pseudo-op. IF statements cannot be nested.

INCLUDE file.ext - include source statements from another file - allows source statements from another input file to be included within the body of the given program. If the file cannot be opened properly, then assembly is aborted. The source module to be included must not end with an END pseudo-op (otherwise, assembly would be terminated). The INCLUDE pseudo-op cannot be nested.

NAME symbol - module name - this pseudo-op defines the name of the program (source and object). The name is placed in the heading of the assembly listing and in the first record of the object output. The module name defaults to 6 blanks.

PSECT op - program section - may appear only once at the start of a source module. This pseudo-op defines the program module attributes for the following operands:

REL - relocatable program (default)
ABS - absolute program. No relocating information is generated in the object module. The module will be linked where it is origined.

ORG nn - origin - sets the program counter to the value of the expression nn. If more than one ORG statement is used in a source module, then the expression nn is a given ORG statement must be greater than a previous ORG statement.

6.4.1 ASSEMBLER DIRECTIVES

Assembler directives are pseudo-ops which are designed to format the assembly listing.

EJECT	- eject a page of assembly listing.
LIST	- turn assembly listing on (default).
NLIST	- turn assembly listing off.
TITLE s	- place title of characters 's' at top of each page of assembly listing. s can be up to 32 characters long.

6.5 OPERANDS

There may be zero, one, or more operands in a statement depending on the opcode or pseudo-op used. Operands in the Assembler may take the following forms:

A GENERIC OPERAND, such as the letter 'A', which stands for the accumulator. The following are Z80 generic operands:

A - Accumulator
B - B register
C - C register
D - D register
E - E register
F - F register
H - H register
L - L register

AF - AF register pair
AF' - AF' register pair
BC - BC register pair

DE - DE register pair
HL - HL register pair

SP - stack pointer register
$ - program counter

I - I register
R - refresh register

IX - IX index register
IY - IY index register

NZ - not zero
Z - zero
NC - not carry
C - carry
PO - parity odd/not overflow
PE - parity even/overflow
P - sign positive
M - sign negative

A CONSTANT. The constant must be in the range 0 thru 0FFFFH. It can be in the following forms:

DECIMAL	- (default); any number can be denoted as decimal by following it with the letter D. Eg: 35, 249D
HEXADECIMAL	- must begin with a number (0-9) and end with the letter H. Eg: 0AF1H
OCTAL	- must end with the letter Q or O. Eg: 377Q, 277O
BINARY	- must end with the letter B. Eg: 0110110B
ASCII	- letters enclose in quote marks will be converted to their ASCII equivalent value. Eg: 'A' = 41H

A LABEL which appears elsewhere in the program. Note that labels cannot be defined by labels which have not yet appeared in the program:

```
EXAMPLE allowed                  not allowed
        -------                  -----------
        I EQU 7                  L EQU H
        H EQU I                  H EQU I
        L EQU H                  I EQU 7
```

AN EXPRESSION. The Assembler accepts a wide range of expressions in the operand field of a statement. All expressions are evaluated left to right constrained by the hierarchies shown below. Parentheses may be used to ensure cc:rect expression evaluation.

operation	operator	hierarchy
equal to	= or .EQ.	0
signed less than	<	0
signed greater than	>	0
signed less than or equal to	<= or =<	0
signed greater than or equal to	>= or =>	0
not equal	>< or <> or .NE.	0
unsigned less than	.LT.	0
unsigned greater than	.GT.	0
unsigned less than or equal to	.LE.	0
unsigned greater than or equal	.GE.	0
reset overflow	.RES.	0
unary plus	+	1
unary minus	-	1
logical NOT (one's complement)	.NOT.	1
multiplication	*	2
division	/	2
addition	+	3
subtraction	-	3
logical AND	.AND.	4
logical OR	.OR.	4
logical XOR	.XOR.	4
logical shift right	.SHR.	4
logical shift left	.SHL.	4

All operands and expressions are converted to 16-bit values. The only exception to this is when expressions take the form:

'character string 1'='character string 2'

In this case, character string 1 and character string 2 are compared character by character for a match. If they do not match, then the value of the expression is false. If they have the same length and match, then the value of the expression is true (0FFFFH).

The reset operator (.RES.) unconditionally resets any overflow error in an operand expression. The shift operators shift their first argument right or left by the number of bit positions given in their second argument. Zeros are shifted into the vacated bit positions. The negative (2's complement) of an expression may be formed by preceding it with a minus sign. The one's complement of an expression may be formed by preceding it with the .NOT. operator.

The symbol $ is used to represent the value of the program counter of the current instruction. Version 3.2 of the Assembler accepts both conventions that are used in Z-80 assembly language in regard to the relative jump, JR, and the decrement B, jump relative non zero, DJNZ. In the earlier convention, the argument required that the program counter value be subtracted from a label value. The later convention does not require the assembly source to make this calculation explicit.

```
EXAMPLE         JR LOOP-$
     or
                JR LOOP
                        - will jump relative to label LOOP.
```

Note that enclosing an expression wholly in parentheses indicates a memory address. The contents of the memory address equivalent to the expression value will be used as the operand value.

The allowed range of an expression depends on the context of its use. For example, the limits on a relative jump instruction are -126 and +129 bytes.

6.6 COMMENTS

A comment is defined as any characters following a semicolon (;) in a line. A semicolon in quotes in an operand is treated as an expression rather than a comment starter. Comments are ignored by the Assembler but they are printed in the assembly listing. Comments can begin in any column.

6.7 ABSOLUTE MODULE RULES

The pseudo-op 'PSECT ABS' defines a module to be absolute. The program will be loaded in the exact addresses at which it is assembled. This is useful for defining constants, a common block of global symbols, or a software driver whose position must be known. This method can be used to define a list of global constants as follows:

EXAMPLE

```
                PSECT   ABS     ;ABSOLUTE ASSEMBLY
                GLOBAL  AA
        AA      EQU     0E3H
                GLOBAL  AX
        AX      EQU     0AF3H
                END
```

6.8 RELOCATABLE MODULE RULES

1. Programs default to relocatable if the 'PSECT ABS' statement is not used or if 'PSECT REL' is used.

2. Only those values which are 16-bit address values will be relocated.

16-bit constants will not be relocated.

EXAMPLE
```
        AA      EQU     0A13H   ;ABSOLUTE VALUE
                LD      A,(AA)  ;AA NOT RELOCATED
        AR      EQU     $       ;RELOCATABLE VALUE
                LD      HL,(AR) ;AR WILL BE RELOCATED UPON LINKING
```

3. Relocatable quantities may not be used as 8-bit operands. This restriction exists because only 16-bit operands are relocated by the SD SYSTEMS Linker.

EXAMPLE
```
        LAB     EQU     $       ;RELOCATABLE VALUE
                DEFB    LAB     ;NOT ALLOWED
                LD      A,(IX+LAB)      ;NOT ALLOWED
                LD      A,(LAB) ;ALLOWED
                LD      HL,LAB  ;ALLOWED
```

4. Labels equated to labels which are constants will be treated as constants. Labels equated to labels which are relocatable addresses will be relocated.

EXAMPLE
```
        B8      EQU     20H     ;CONSTANT
        C8      EQU     B8      ;CONSTANT
                LD      A,(C8)  ;C8 WILL NOT BE RELOCATED
        AR      EQU     $       ;RELOCATABLE ADDRESS
        BR      EQU     AR      ;RELOCATABLE
                LD      A,(BR)  ;BR WILL BE RELOCATED
```

5. External symbols in a relocatable program are marked relocatable, except for the first usage. The code for external symbols is actually a backward link list through the object code.

6.9 GLOBAL SYMBOL HANDLING

A global symbol is a symbol which is known by more than on module. A global symbol has its value defined in one module. It can be used by that module and by any other module which is linked with it by the SD SYSTEMS Linker. A global symbol is defined as such by the GLOBAL pseudo-op.

An internal symbol is one which is defined as global and also appears as a label in the same program. The symbol value is thus defined for all programs which use that symbol. An external symbol is one which is defined as global but does NOT appear as a label in the same program.

EXAMPLE
```
                GLOBAL  SYM1    ;DEFINE GLOBAL SYMBOL
                CALL    SYM1
                .
                .
                END
                          - SYM1 is an external symbol
```

EXAMPLE

```
                GLOBAL  SYM1     ;DEFINE GLOBAL SYMBOL
        SYM1    EQU     $
                LD      A,(SYM1)
                .
                .
                .
                END
                          - SYM1 is an internal symbol.  Its value
                          is the address of the LD instruction.
```

If these two programs were assembled and then linked by the SD SYSTEMS Linker, then all global symbol references from the first program would be 'resolved'. This means that each address in which an external symbol was used would be modified to the value of the corresponding internal symbol. The linked programs would be equivalent (using our example) to one program written as follows.

EXAMPLE

```
                CALL    SYM1
                .
                .
                .
        SYM1    EQU     $
                LD      A,(SYM1)
                .
                .
                .
                END
```

Global symbols are used to allow large programs to be broken up into smaller modules. The smaller modules are used to ease programming, facilitate changes, or allow programming by different members of the same team.

6.10 GLOBAL SYMBOL RULES

1. An external symbol cannot appear in an expression which uses operators.

EXAMPLE

```
                GLOBAL  SYM1     ;EXTERNAL SYMBOL
                CALL    SYM1     ;OK
                LD      HL,(SYM1+2)     ;NOT ALLOWED
```

2. An external symbol is always considered to be a 16-bit address.

Therefore, an external symbol cannot appear in an instruction requiring an 8-bit operand.

EXAMPLE

```
                GLOBAL  SYM1     ;EXTERNAL SYMBOL
                CALL    SYM1     ;OK
                LD      A,SYM1   ;NOT ALLOWED
```

3. An external symbol cannot appear in the operand field of an EQU or DEFL statement.

4. An internal symbol is always marked relocatable in a relocatable assembly. This point is important because an internal symbol will always be relocated even though it looks like a constant. To define constant internal symbols, create an absolute assembly via the PSECT ABS pseudo-op.

5. For a set of modules to be linked together, no duplication of internal symbol names is allowed. That is, an internal symbol can be defined only once in a set of modules to be linked together.

7.0 TECHNICAL INFORMATION

The Assembler is resident on a CP/M compatible system diskette and, when loaded, starts at location 100H. Assembler variables are placed in memory at the top of the Assembler. The symbol table is placed in RAM starting at the end of the Assembler and ending at the starting address of DOS. Typically, more than 300 symbols are allowed per program.

SD SYSTEMS LINKER VERSION 2.0
OPERATIONS MANUAL

1.0 INTRODUCTION

The SD SYSTEMS Linker is provided on a standard OP/M compatible diskette. The Linker (LINK) provides the means for linking object modules produced by the Z80 Assembler (ZASM). The Linker concatenates modules together and resolves global symbol references which provide communication between modules. The Linker produces a load module containing "hex" format machine code which may be read by the DOS LOAD commmand. The LOAD command reads a load module (secondary filename = HEX) and produces a memory image file (secondary filename = COM) which can be executed by the disk operating system (DOS).

2.0 COMMAND SUMMARY

In this manual, the following symbols are used:
- (CR) means carriage return.
- all user input is underlined.
- user input which is all upper case must be entered exactly as shown.
- user input which is lower case is variable.

```
A>LINK  filename 1, filename 2,....filename N /xyz  (CR)
  ----------------------------------------------------------
```

- Links object input files (secondary filename=OBJ)
- Produces a LOAD module (secondary filename=HEX)
- As an option creates a cross reference file (secondary filename=CRS)

OPTIONS

C - Produces an output file containing a global cross reference table and a load map.
U - Lists all undefined global symbols
A - Allows the user to enter a starting link address

3.0 DEFINITIONS

SOURCE MODULE - the user's source program. Each source module is assembled into one object module by the Assembler.

OBJECT MODULE - the object output of the Assembler for one source module. The object module contains linking information, address and relocating information, machine code, and checksum information for use by the Linker. The object module is in ASCII. The object module is output to a disk file with extension OBJ.

LOAD MODULE - the absolute machine code of one complete program. The load module is defined on disk as an absolute object file with secondary filename of HEX. A Load module is produced by the Linker.

GLOBAL DEFINITION - both internal and external symbols are defined as GLOBAL in a source module. The Assembler determines which are internal and which are external. (See Z80 Assembler description of internal and external symbols).

ABSOLUTE - a program which has no relocating information in the object module. An absolute program which is not position independent can be loaded only in one place in memory in order to work properly.

RELOCATABLE - a program which has extra information in the object module which allows the Linker to place the program anywhere in memory.

4.0 LINKER OPERATION

During Pass 1 the Linker reads one or more object input files and places the global symbol definitions in the Linker symbol table. In PASS 2 global symbol references are resolved and an output Load file is produced. The Load file has the same primary filename as the first object input file (filename 1) and has a secondary filename of HEX. If the cross reference option is specified a cross reference file is produced. The cross reference file has the same primary filename name the first object input file (filename 1) and has a secondary filename of CRS.

In Pass 2 as each object input module is read its beginning and ending address in memory is printed on the console. The module type is also listed as either absolute or relocatable (ABS/REL). Absolute modules are always positioned at their starting address in memory as defined by the ORG pseudo-op. Relocatable modules are positioned at the next location after the end address of the previous module. If the first input module is relocatable, it is positioned by the starting link address. If the starting link address is not specified by the A option it assumes a value of 0.

It is suggested that the first object input module read by the Linker have a starting address of 100H for operation with the DOS. This starting address should also serve as the entry point for the combined Load module. A starting address of 0100H can be created either with the ORG pseudo-op or the Linker A option. The DOS loads and begins execution of RAM image files at location 0100H.

When absolute modules are being linked together, the files in the LINK command must appear in sequential order according to their starting addresses in memory. If an absolute module is encountered having a starting address lower in memory than a previous module a module sequence error message will be generated. Furthermore, if a source module contains more than one ORG statement, the address used in any given ORG statement must be greater than a previous ORG statement.

5.0 EXAMPLE OF LINK COMMAND

EXAMPLE 1. Link the relocatable object modules MAIN.OBJ, SUB1.OBJ, ,SUB2.OBJ,SUB3.OBJ together starting at 100H producing the LOAD module MAIN.HEX. Also generate a global cross referecne table and a load map in the file MAIN.CRS.

```
A>LINK MAIN,SUB1,SUB2,SUB3    (CR)

OPTIONS? A C (CR)

ENTER STARTING LINK ADDRESS> 100   (CR)

MAIN     .OBJ
SUB1     .OBJ
SUB2     .OBJ
SUB3     .OBJ
UNDEFINED SYMBOLS 00
PASS 2
MAIN     .OBJ     REL     BEG ADDR 0100     END ADDR 0125
SUB1     .OBJ     REL     BEG ADDR 0126     END ADDR 01CD
SUB2     .OBJ     REL     BEG ADDR 01CE     END ADDR 01E8
SUB3     .OBJ     REL     BEG ADDR 01E9     END ADDR 0212

A>
```

EXAMPLE 2. Using the load module MAIN.HEX created in Example 1 and the DOS LOAD command create a memory image file and begin execution of MAIN.

```
A>LOAD MAIN.HEX    (CR)

A>MAIN    (CR)
```

NOTE: Execution of MAIN has been started.

EXAMPLE 3. Using the DOS TYPE command list the global cross reference table and the load map for the modules linked in Example 1.

```
A>TYPE MAIN.CRS    (CR)
```

```
LOAD MAP

MAIN     .OBJ     REL     BEG ADDR 0100     END ADDR 0125
SUB1     .OBJ     REL     BEG ADDR 0126     END ADDR 01CD
SUB2     .OBJ     REL     BEG ADDR 01CE     END ADDR 01E8
SUB3     .OBJ     RBL     BEG ADDR 01E9     END ADDR 0212


 GLOBAL CROSS REFERENCE TABLE

 SYMBOL ADDR   REFERENCES
 CRLF   E59C   020C 01E6
 MAIN   0100
 MODNO  01FB   01D4 01D1 012C 0129 010C 0109
 MSGBEG 013F   0101
 MSGEND 0165   011E
 MSGMAI 018A   010F
 MSGMOD 01C2   0201
 MSGSB2 0195   01D7
 MSGSB3 019B   01F2
 PRINT  01E0   01F5 013C 0132 0121 0112 0104
 PTEST  0138   01F8 01DD
 SUB1   0126   0115
 SUB123 020F
 SUB2   01CE   0118
 SUB3   01E9   011B
```