INTRODUCTION

The XL-Z80 software system was designed by staff members of XL Computer Products and intended for use on the NORTH STAR MICRO DISK SYSTEM. Version 2.1 includes a powerful combination of Editor/monitor, Assembler, and Disassembler. All three programs use the existing user I/O routines for the North Star DOS. The user should place his own DOS in the reserved area on disk (under file name "USERDOS") prior to running the system.

The editor includes such features as free format command entry, automatic memory management, line numbering and resequencing, string search and replacement, and multi-disk interface of file storage.

The Assembler's features include multiple statement per line ability, multiple file processing, free-format ASCII strings, multi-length label storage, and alphabetical symbol table listing. In addition, the Assembler provides a variety of list and storage options that allow development of programs exceeding the size of RAM available in the user's system.

The Disassembler processes binary files directly from disk and creates, if desired, ASCII files on disk in standard format for later processing by the Editor or Assembler. An optional listing is provided that displays the ASCII equivalent of the disassembled product allowing the user to search for string buffer areas. The Disassembler allows the user to relocate programs or retrieve lost source listings.

Locations 2A07 and 2A08 (in all 3 programs) contain values that define the user's console and printer device numbers. They are defined as 0 and 1 respectively and should be changed if they differ from these defined values.

The remainder of this manual describes in detail the operation of these three programs, with appendices further explaining their operation and interface of this package to meet your needs. With the XL-Z80 and North Star combination, you are on your way toward developing a powerful machine language library!

XL-Z80 Editor

The XL-Z80 Editor resides in RAM starting at 2A00 Hex. Its purpose is to create and modify source listings to be later processed by the Assembler. Each line is entered in the program by providing a line number from 0 through 65535. There are sixteen additional commands that allow the user to easily modify the file or RAM (memory) areas. These commands are:

COMS	Lists all of the optional commands provided by the Editor
ENTR	Enters hexadecimal data into memory
DUMP	Prints contents of memory in hex format
JUMP	Jumps to a specified RAM location
FILE	Assigns and creates a new source file in RAM. If already present, shows parameters of the file
LOAD	Loads a source file from disk to file area
SAVE	Stores the current file on diskette
ASEQ	Provides automatic line numbers for file
RSEQ	Renumbers the current file with new values
LIST	Lists portions of the source file. Lists contents of current diskette, if specified
SCAN	Prints each line of a file where a specified string appears. Replaces with new string

DELT	Deletes lines from the source file
EXIT	Returns control to the DOS
MOVE	Transfers 256 bytes of memory to a new location
CONV	Displays the difference of any two values in hex, octal, and decimal
ASSM	Loads and executes Assembler from disk

In addition, a line editor acts upon the last line entered or listed and provides most of the features found in the North Star Basic line editor. This provides rapid text editing capability to the user. If a file line other than the last line entered must be edited, simply LIST the line to be edited. The following commands are available:

CONTROL-A	Copies one character from old line
CONTROL-D	Copies up to a specified character
CONTROL-G	Copies remainder of old line
CONTROL-N or @	Re-edits old line from beginning
CONTROL-Q or RUB	Backs up one character in both lines
CONTROL-Y	Inserts characters until next Control-Y
CONTROL-Z	Erases one character from old line
CONTROL-C	Aborts any operation and returns to 2A04H (the Editor reentry point)
CONTROL-I	TAB function. Advances print head to next 8-character wide column. Useful for formatting.

EXPLANATION OF COMMANDS

A detailed explanation of each command will now follow. To understand the description of each operation, the following information is provided:

*To execute a command or place the entered/edited line into the file, a carriage return   is typed.

*FILENAME refers to any Type 10 disk file up to 8 characters long and consisting of all printable ASCII characters and no embedded blanks. All files must be created using the DOS and the maximum size of any file is 255 disk blocks (65535 bytes).

*Drive # refers to a numbered disk drive (1-3). If Drive # is not specified, the default drive is 1.

*Words contained in parenthesis are optional and are not required to be entered, but if entered will provide some additional feature.

*Command (non-numbered) lines separate all parameters with one comma or any number of spaces. A space following the command is not required.

*All values, addresses, and line numbers entered in command lines are interpreted as positive decimal values, unless followed by "H" (Hexadecimal) or "Q" (Octal).

*To terminate List or Dump operations, a Control-C may be typed. Output will resume following the printing of a "BREAK", provided the carriage return key is pressed. Additionally, typing "R" will return to start of program, or "D" will return to the DOS.

*Control-P is used to place user's printer on (or off) line. All subsequent console output is echoed to the printer device.

COMS ↵

This command prints a listing of the 15 remaining commands:

ENTR	DUMP	JUMP
FILE	LOAD	SAVE
ASEQ	RSEQ	LIST
SCAN	DELT	EXIT
MOVE	CONV	ASSM

ENTR ADDR1↵

Will place values in memory starting at memory address ADDR1. Lines of hexadecimal values are processed only after entire line is entered. A colon ":" prompts each line, indicating that the ENTR mode is in effect. Conversion will continue until a "/" is encountered, or an illegal hex value was encountered. Control-C will also exit this mode. Data will not be allowed to overwrite any protion of the DOS, Editor, or current file.

```
ENTR5000H ↵
: 2A  3B  F2 6  A 77 / ↵ will enter data from
                         5000H to 5005H.

ENTR 1200Q ↵
:2A  3B F20  A 77 ↵      only valid data from 1200Q
I DON'T UNDERSTAND       to 1201Q was accepted.
```

DUMP ADDR1 (ADDR2) ↵

Lists contents of memory in hexadecimal format from memory location ADDR1 to ADDR2. If ADDR2 is not specified, then only contents of ADDR1 is provided. CONTROL-C will suspend listing until Carriage Return is pressed.

```
DUMP 64,71 ↵
0040  01  02  03  45  F7  9A  3B  88
|                     |
└→Hex Address         └→Contents
```

JUMP ADDR1 ↵

This command exits the Editor and resumes execution at ADDR1. The specified address cannot be anywhere within the DOS, Editor, or current file areas.

FILE FILENAME ↵

Creates a file named "FILENAME" and assigns it to a RAM location following the Editor. File parameters are then displayed.

FILE ↵

Displays parameters of current file to include beginning and ending hexadecimal addresses, number of blocks on disk required to store it, and the maximum line number contained in the file.

Example:

```
     FILE ↵
     STARTREK   388C 5B77 (35 Blocks on Disk) MAXLN=2060
     FILE FINANCE ↵
     FINANCE   388C  388C ( 1 Blocks on Disk) MAXLN=0
```

LOAD FILENAME, DRIVE # ↵

Reads in a Type 10 source file from specified disk, examines the loaded file for proper format, and displays the parameters of the file. The file in memory prior to the command is erased, whether the new load was successful or not:

```
     FILE ↵
     FINANCE  388C  388C ( 1 Blocks on Disk) MAXLN=0
     LOAD STARTREK,3 ↵
     STARTREK 388C  5B77 (35 Blocks on Disk) MAXLN=2060
     LOAD FINANCE ↵
     MISSING FILE ERROR    file not saved previously
     FILE ↵
     MISSING FILE ERROR
```

SAVE (FILENAME), DRIVE # ↵

Saves the current file on specified diskette under the current file name or, if specified, the optional FILENAME. The file must have been previously created or a MISSING FILE ERROR will occur:

```
     LOAD STARTREK ↵
     STARTREK  388C  5B77 (35 Blocks on Disk) MAXLN=2060
     10 MESSAGE: "SPOCK, ARM THE PHASERS!"\DEFB CR
     SAVE,2↵          saves as "STARTREK" on drive 2
     READY
     SAVE STARWARS ↵   saves as "STARWARS" on drive 1
     READY
     FILE ↵
     STARTREK 388C  5B99 (35 Blocks on Disk) MAXLN=2060
```

** Note: The Editor will indicate when the disk file addressed is missing or when the current file is too large to store in the disk file. If this occurs, exit the Editor, create your larger file on disk, and jump to 2A04 (the reentry point). If Editor is reentered at 2A00, the file will be erased. Do not use the DOS COmpact command while using the DOS.

ASEQ VALUE ↵

Provides automatic line numbers for each file line to be entered in increments of 1, starting at the first line number VALUE until an illegal line is entered or a Control-C is pressed. The line number is printed and print head stops at column one of the line, reserved for labels.

```
ASEQ 121 ↵
121 LABEL2: LD HL, MESSAGE ↵
122         CALL PRINT \ INC HL \ CALL PRINT ↵
123         SUB E \ CP D \ JR NZ,LABEL2 ↵
124 (Control-C)
READY
```

RSEQ VALUE1, (VALUE2) ↵

Renumbers the entire current file from VALUE1 in increments of 10 or VALUE2, if specified. If the maximum line value of 65535 is exceeded, then an error message is printed and the file is renumbered starting at line value 1 in increments of 1.

```
FILE ↵
STARTREK  388C  5B99 (35 Blocks on Disk) MAXLN=2060
RSEQ 65000,10000 ↵
I DON'T UNDERSTAND
FILE ↵
STARTREK 388C  5B99 (35 Blocks on Disk) MAXLN=207
RSEQ 10 ↵
FILE ↵
STARTREK 388C 5B99 (35 Blocks on Disk) MAXLN=2070
```

LIST (LINE1) (LINE2) ↵

Lists lines from the current file from LINE1 to LINE2 (if specified). If LINE1 is not specified, then the entire file is listed. Listing is suspended by typing CONTROL-C and continued with CARRIAGE RETURN. The line-editor operates on the last line listed when listing is terminated.

```
LIST10 ↵
10 MESSAGE: "SPOCK, ARM THE PHASERS!"\DEFB CR
```

LIST,DRIVE# ↵

Lists the directory of the specified disk drive.

SCAN "STRING 1"("STRING 2") ↵

This command scan through every line of the file and prints each line where the first character string is found. If the optional second string is entered, it will replace every occurence of the string in the program. The resulting line must be equal to or less than 80 characters in length or the operation will not be allowed.

```
SCAN "MESSAGE" ↵
1020  CALL MESSAGE
2030  CALL OUTPUT\LD HL, MESSAGE\CALL PRINT\ RET
3650  MESSAGE: LD A,(DE)\CP CR\RET Z\ LD B,A
3660           CALL OUTPUT\INC DE\ JP MESSAGE
SCAN "MESSAGE" "PRINT" ↵
1020  CALL PRINT
2030  CALL OUTPUT\ LD HL,PRINT\CALL PRINT\ RET
3650  PRINT:  LD A,(DE)\ CP CR\RET Z\LD B,A
3660          CALL OUTPUT\INC DE\JP PRINT
```

** Note: This powerful command is especially useful in renaming labels produced by the system Disassembler. Once a routine has been identified, the assigned label may be replaced with a logical name to help the programmer follow the program.

** CAUTION: Be sure to <u>completely</u> identify the string you wish to replace and remember that <u>every</u> occurrence of the string will be replaced.

DELT LINE1 (LINE2) ↵

Deletes lines from the current file from LINE1 to an optional LINE2 and compacts the file so that no wasted memory exists in the file.

```
FILE ↵
STARTREK 388C 5B99 (35 Blocks on Disk) MAXLN=2070
LIST 2070 ↵
2070  LNK STRTRK2  END OF PART ONE
DELT 2070 ↵
FILE ↵
STARTREK 388C 5B8Z (35 Blocks on Disk) MAXLN=2060
```

EXIT ↵

Returns program control to DOS for creating files on diskette. Return to Editor is made to 2A00H or to 2A04 (to retain the file.)

MOVE ADDR1 ADDR2 ↵

Transfers 256 bytes of data in memory from address ADDR1 to ADDR2. Command will not allow data to over-write any portion of the DOS, Editor, or current file.

```
MOVE 2000H,65000 ↵
MOVE 2000,2A00H ↵
PROGRAM OVERWRITE ERROR
```

CONV VALUE1 (VALUE2) ↵

Displays the difference between VALUE1 and VALUE2 by subtracting the latter from VALUE1. If VALUE2 is not entered, then only the converted value of VALUE1 is computed. The result is printed in hexadecimal, octal, and decimal:

```
CONV 100000Q ↵
8000 HEX   100000 OCTAL  32768 DECIMAL
CONV 8000H,33000 ↵
FF18 HEX  177430 OCTAL  -232 DECIMAL
```

ASSM (FILENAME),DRIVE# ↵

This command loads in the Assembler, passes on the name of the current source file (or optional FILENAME) to the Assembler, and executes assembly operation. The Assembler must be stored on disk as ASSMZ80. If no file is current in the Editor, and no FILENAME is entered, the Assembler is loaded and started from the beginning. The specified drive contains the first source listing to be processed:

```
FILE ↵
STARWARS 388C 5B77 (35 Blocks on Disk) MAXLN=2060
SAVE STARTREK ↵
READY
ASSM STARTREK,2 ↵                  **needed to specify
ENTER YOUR OUTPUT LIST OPTION:     STARTREK as file to be
1.  PRINT ONLY ERRORS              assembled since STARWARS
2.  OUTPUT A COMPLETE LISTING      is current file.
    .
    .
    .
```

XL-Z80 ASSEMBLER

After the XL-Z80 Assembler is given control by the Editor, or loaded in separately and run, it will translate Type 10 source listings into Z80 machine language code. The Assembler processes source code written in Zilog format and has such outstanding features as:

1. Multiple statement lines allow more than one instruction per line to be processed, reducing file storage requirements.

2. Symbolic and relative addressing allows complex addressing relative to the program location counter or to any pre-defined symbol name.

3. Unlimited number and variable length symbols allows efficient storage of symbols and labels in memory. Holds as many labels as user's memory can store.

4. Predefined symbols include all Z80 registers.

5. ASCII String generation provides fast, simple, conversion of ASCII strings and messages.

6. Automatic Binary Code storage optionally stores generated code in memory or on diskette, tracks the size and disk space needed, and number of assembler errors detected.

7. Multiple file processing allows large programs to be "daisy chained" together, thus processing programs normally too large to be stored in memory at one time.

8. Multi-Disk capability allows user to process source and binary files on different disk drives.

THEORY OF OPERATION

After determining the listing and storage options, the Assembler performs two passes on the source file stored on diskette. During Pass 1, the values of all symbols and labels found in the program are evaluated and stored in a symbol table located in memory immediately following the Assembler. The program generates object code (if desired) during Pass 2 and places the object code directly in memory following the symbol table. For storing large programs, the Assembler will store the code on diskette as it is processed. The largest single file capable of being read from or written to is 255 blocks (65280 bytes). Of course, several source listings that large may be linked together.

After completion of Pass 2, the Assembler will compute and print the size of the program generated (exclusive of any buffers found at the end of the program), the number of blocks on disk required for its storage, and number of errors found. An optional symbol table listing is then offered that will print in alphabetical order the entire symbol table. Return is then made to the DOS.

USE OF THE XL-Z80 ASSEMBLER

Each instruction processed is divided into four possible "fields" as follows:

> The optional label field identifies in the symbol table the value of the program counter at that given line. If present, it must begin in the second character position following the line number. The first character position is always occupied by a space. The label may consist of up to ten alpha-numeric characters, followed by a blank or optional colon (:). The entire label is stored in the symbol table with its associated value. For processing large programs with limited memory available, the user should minimize the number and length of labels used. The first character of the label must be a value from A through Z.

The opcode field may contain any legal Z80 opcode in Zilog format or pseudo operation instruction, including ASCII strings of any length.

The operand field contains the values associated with the opcode. If two registers are used in the operand field, then a comma must follow the first argument.

All characters following the operand field define the comment field. If the entire line is to be used as a comment, an asterisk (*) or semi colon (;) must be placed in the label field.

Where more than one opcode is to be entered on one line of source file listing, the comment field is replaced or followed by a backslash (\) that indicates a multiple line is being read. The Assembler treats each multiple instruction as a separate line and echoes the orginal line number on the Assembler listing.

Assume the following subroutine was created using the Editor:

```
005 OUTPUT EQU 200DH
100 MESSAGE: LD A,(DE)\ CP 1 \RET Z RETURN IF DONE
105   LD B,A \CALL OUTPUT \INC DE\JP MESSAGE
110 *********************************************
111 *  THIS ROUTINE MULTIPLIES (HL) BY 5          *
112 *********************************************
115 MULT5  PUSH HL \POP DE  COPY HL TO DE
120  ADD HL,HL \ADD HL,HL  MULTIPLY BY 4
125  ADD HL,DE  4+1=5
130  RET
```

Then the Assembler-generated listing would be:

```
0000 1A             100 MESSAGE: LD A, (DE)
0001 FE 01          100     CP 1
0003 C8             100     RET Z RETURN IF DONE
0004 47             105     LD B,A
0005 CD 0D 20       105     CALL OUTPUT
0008 13             105     INC DE
0009 C3 00 00       105     JP MESSAGE
000C                110 ***************************
000C                111 *  THIS ROUTINE MULTIPLIES (
000C                112 ***************************
000C E5             115 MULT5  PUSH HL
000D D1             115     POP DE  COPY HL TO DE
000E 29             120     ADD HL,HL
000F 29             120     ADD HL,HL  MULTIPLY BY 4
0010 19             125     ADD HL,DE  4+1=5
0011 C9             130     RET
```

After Pass 1 is completed, the symbol table will contain the following symbols and their associated values:

MESSAGE=0000   MULT5=000C   OUTPUT=200D

Note that MESSAGE and MULT5 were evaluated by the Assembler, while OUTPUT had to be declared by the user since the routine occurs outside the program. There are several variable names already defined by the Assembler and these may not be reassigned values by the user. They are:

A, B, C, D, E, H, L, I, R, IX, IY, HL, BC, DE, SP, AF

<u>Relative Addressing</u>

In larger programs where the user has insufficient memory to store all symbols or labels that would normally be used, relative addressing will significantly reduce the storage area required. A relative jump from any label, numeric value, or present position may be assigned by using the + or - numeric operators. The dollar sign ($) is used to denote the program location counter address immediately following the current instruction.

<u>JP C,OUTPUT-6</u> will cause program to jump to a location six bytes before address OUTPUT and resume operation.

<u>CALL 2010H+3</u> will call subroutine located at 2013H.

<u>XOR A\DEC A\JR NZ,$-3</u> will set register A to zero, then loop 256 times before continuing.

<u>INFLOOP: JR Z,INFLOOP</u> will cause an infinite loop if the status flag Z was set. In the original Zilog format, the above instruction was written as "JR Z,INFLOOP-$". XLZ80, however, does not require the "-$" to be typed.

<u>Constants & Numeric Expressions</u>

The Assembler will process positive or negative decimal, octal, or hexadecimal values. All values are assumed to be decimal unless followed by "Q" or "H" for octal or hexadecimal values. Constants are evaluated as 16-bit values and will cause an assembler error if exceeded. To distinguish between symbols and numeric values, all numeric values must be preceded by a numeric value.

Example:

```
LEGAL CONSTANTS: 256H,256Q,256
                 OPCODE+A-LOCATION-10000Q
                 -0D7H, 176213Q, -VALUE

ILLEGAL CONSTANTS: 67000, B7H, 239Q, 3*4
                   7F62,-0FFQ, VALUE/56
```

Assembler Directives (Pseudo-Ops)

The Assembler contains several directives which will cause certain operations to be performed. These "pseudo-ops" are placed in the normal opcode field:

ORG 2A00H	Sets program pointer to specified value. All instructions processed following this command will be computed from origin 2A00H. If ORG is not specified, assembly will begin at location 0000 by default.
LABEL EQU 3 THREE EQU LABEL	Associates value of 3 with symbols "LABEL" and "THREE" in the symbol table. A symbol may be equated or evaluated only once, or else a duplicate error will occur.
M1: "SPOCK, YOU FOOL!" LD HL, "12" LD A,"?"	Strings contained in double quotes are processed and placed directly in memory if they appear in the opcode field. Two-byte operands are evaluated with the first character in the high-order register, and second in the low-order register.
BUFFER DEFS 1024	Defines a storage area in memory and causes Assembler to skip past the specified number of bytes when assigning code to memory.
DEFB 0DH	Fills one byte (8-bit word) of memory with the associated value.

DEFW 30760Q	Fills two bytes (16-bits) with the associated value in the operand
LNK PART2	Locates next source listing "PART2" on disk and continues assembly with all previously stored symbols

USING THE XL-Z80 ASSEMBLER

The XL-Z80 Assembler was designed to make the assembly process as simple and painless as possible for the user. The Assembler will direct a series of questions and commands to the user before and after the assembly to provide him with his desired output.

First, the name of the source file is requested. Enter in the name of the Type 10 source file on disk that you wish to assemble. If a multiple source file is to be processed, enter the name of the first file in the series. If the source file is on other than drive one, specify the drive after the title (separated by a comma). A CONTROL-C will return control to the DOS.

Next, select the desired list option. Only keyboard inputs 1 or 2 will be accepted. An error-only listing will cause the Assembler to print only lines with assembler detected errors. Option #2 will produce a complete listing of the assembly including errors.

Finally, tell the Assembler where you would like the generated object code to be stored. If the disk storage option is selected, the Type 1 file should have been reserved prior to running the Assembler using the DOS. As a rule, this option is used after most errors have been eliminated and the user knows how large to make his file. If the memory storage option is used, the Assembler places the binary code in memory after the last symbol in the symbol table in order to use RAM as efficiently as possible.

The Assembler will now begin a two pass scan of the source listing(s). To suspend the assembly at any point, the previously mentioned "BREAK" feature may be used, however the Assembler cannot be reentered if exited other than at 2A00H. Pressing Control-P on the keyboard at any time the Assembler is waiting for an input will place the printer on (or off) line.

XL-Z80 DISASSEMBLER

The XL-Z80 Disassembler examines Type 1 object files stored on disk and produces a source listing of the file. Depending upon the option selected, the program will automatically fill Type 10 files on disk with the source listing. In addition, the user may select a complete listing of the disassembled file to be output on the terminal. To detect stored messages within the file, this listing includes a periodic scan of the object code interpreted as printable ASCII characters.

The purpose of this program is to allow the user to retreve lost source listings or to relocate existing programs to different addresses. It is not intended to be used to copy or modify licensed and copyrighted programs or materials. The Disassembler will not, therefore, properly decode XL-Z80 software.

USING THE XL-Z80 DISASSEMBLER

Like the Assembler, the Disassembler provides a simple and direct series of commands that allow the user to select the desired disassembled product.

First, determine how large the object file is and how large the source file will be. For each block (256 bytes) of object code processed, about 8 blocks of ASCII source listing is usually generated. Thus a 10-block Type 1 object file would require approximately 80 disk blocks to store the listing.

Next, determine how large of a file your XL-Z80 Editor will be able to load (since you will probably need to modify the listing) and use the DOS to create the number of files needed. A 16K system will accommodate 40 blocks of Type 10 listing, a 32K up to 104 blocks, etc. The exact value may be found using the CONV command and finding the difference between your end of RAM and the first assigned file area used by the Editor.

Create your Type 1 object file using the DOS. The Disassembler will use the "Go Address" that you specify as the origin for the disassembly.

Now run the Disassembler with the desired options selected. For large programs, a listing is essential for later debugging in order to identify embedded buffer areas.

Once a source file is filled on diskette, the program will ask for the name of the next succeeding source listing file. Enter the name of the next source file previously reserved. The program will insert a "Link" statement and close the previous file after the new name is entered. A new name must be entered to close the file properly. if the next file name is not found, a message will be printed and the user instructed to place the proper diskette into the drive. At this point, a CONTROL-C will terminate the disassembly and jump to the DOS. The disassembly process is not complete until the program has typed:

```
END OF DISASSEMBLY.
*
```

You now have a complete listing(s) of the object code file on diskette. The source file can be assembled to reproduce the object code. To relocate the program, simply load in the first source file using the Editor and change the "ORG" of the program to your desired address before assembling.

** Note: the Disassembler assigns names to labels in the file by compiling a symbol table in Pass 1. It forms a label name by preceding the address with the letter "L":

```
PRINT:  LD A,(DE)        appears as    L2A15: LD A,(DE)
        CP ODH                                CP ODH
        RET Z                                 RET Z
        LD B,A                                LD B,A
        CALL OUTPUT                           CALL L200D
        INC DE                                INC DE
        JP PRINT                              JP L2A15
```

The Editor SCAN command is useful in renaming each occurance of a label throughout the entire file (see Page 8).

An inherent problem with any disassembler is determining which memory locations contain data and which contain machine language instructions. Most well-written software will separate the data and ASCII buffers from the rest of the program. This not only provides a central location for examining stored values when debugging programs, but requires less memory for actual program storage since data buffers generally do not operate correctly, the user should search the program listing to determine where a disassembler error may have occurred. The problem occurs where a data buffer ends and a series of instructions begins. In this case, the last data byte in the buffer may be translated as a 2 or 3-byte opcode and overwrite the first few bytes of actual instructions. The XL-Z80 Assembler will detect this error most often as a missing label error, since labels are only assigned on the M1 processor state by the Disassembler.

Appendix 1: Error Message Explanation

The Editor, Assembler, and Disassembler contain a variety of error messages that aid the programmer debugging source listings. While the Disassembler's messages are self-explanatory, the Editor and Assembler require some additional remarks.

Editor Error Messages

The following messages will be returned by the Editor when the following conditions have occurred:

FILETYPE ERROR	A disk SAVE or LOAD attempted to access a file other than a Type 10 file.
I DON'T UNDERSTAND	An illegal command was entered
MISSING FILE ERROR	Disk file not found during a LOAD or SAVE command; Operation on a non-existing file attempted
PROGRAM OVERFLOW ERROR	A file line, if entered, would have resulted in overflow of available RAM in system; Program too large to SAVE in disk file
NUMERIC VALUE ERROR	An illegal hex, octal or decimal value was entered.
FORMAT ERROR AT ADDR1	Before a file operation, the file was checked for proper format and an error detected. Usually results from insufficient RAM or a defective RAM location. No further operations allowed.
PROGRAM OVERWRITE ERROR	An ENTR, JUMP, or MOVE command would have overwritten the DOS, Editor, or file areas.
FILE NOT SAVED	Attempted to assemble a file before saving it on disk.
LINE OVERFLOW ERROR	Use of the SCAN command would have generated a line exceeding 80 characters in length.

Assembler Error Messages

The following error symbols are generated during each assembly as they are detected:

A	Argument Error-an illegal operand entered
D	Duplicate Label Error-same symbol defined twice
E	End of RAM-insufficient RAM for symbol table
L	Label Error-improper format found for label
M	Missing Label on EQUate
O	Opcode Error-an illegal opcode detected
R	Range Error-out of range for relative jump
S	Syntax Error
U	Undefined Symbol not found in symbol table
V	Value Error-8 bit operand too large or small

If the code storage on disk option is selected, the following error messages may also be produced:

MEMORY FULL: CODE STORAGE ON DISK ABORTED.
A RAM buffer following the last symbol in the symbol table is needed to store code until it is output to disk. The required 1280 bytes were not available. The number and length of labels in the symbol table should be shortened, or more RAM added.

DISK FILE TOO SMALL: CODE STORAGE ON DISK ABORTED.
The Type 1 file created on disk by the user is not large enough to store all of the code generated by the Assembler, and needs to be made larger.

Appendix 2: Interfacing Source Files

The XL-Z80 will process only source files stored on disk in Type 10 format. To process source files from paper tape, cassette, or other storage media it will be necessary to write a conversion routine that will convert the file to XL format.

Description of Type 10 Format

Each line of the source file contains four additional characters that describe the parameters of the line. An end-of-file marker follows the last line in the file:

n	L	H	X	X	X	X	X	X	X	X	X	X	CR	1

n	Line Size-contains the number of characters in the line, including the four extra symbols
L,H	Line Number-low and high order of line in binary
XXX...	Instruction-the label,opcode,operand,and comments
CR	Carriage Return-contains value ODH
1	End of File Marker-contains value of 1

Example:

Assume the following program has been entered:

```
200 DELAY:  LD HL,1000
300  DEC HL LD A,H\OR A\JP NZ,DELAY+3
400         LD A,L\OR A\JP NZ,DELAY+3
500 END     RET
```

The program is stored in memory and on disk as:

```
16H,C8H,0,DELAY:  LD HL,1000,ODH,25H,2CH,1,
 DEC HL LD A,H\OR A\JP NZ,DELAY+3,ODH,25H,90H,1,
        LD A,L\OR A\JP NZ,DELAY+3,ODH,OFH,F4H,1,
END     RET,ODH,1
```

Converting to Type 10 Format

The following steps should be taken to create a Type 10 source file from an existing source file:

1. Write a short routine that will convert your existing source file into Type 10 format. The Type 10 format requires less RAM than most other formats. Load in the original file, convert it, and store the new file anywhere in RAM other than at 2000H to 29FFH (the DOS area).

2. CReate a Type 10 file in the Directory using the DOS after determining the approximate size of the new file. If unsure of the exact size, make the file on disk as large as your entire RAM. Use the SF command to store the new file on disk.

3. Load in the Editor. Use the LOAD command to load in the new Type 10 file from diskette. If your conversion routine improperly converted the original file, the Editor will give a FORMAT ERROR and the address in RAM where the error occurred. The Editor will attempt to load the file starting at location 36E1H. If the load is successful, the Editor will print the parameters of the file, including the number of disk blocks required to store it.

4. Renumber the source listing and SAVE the file on disk. Then, using the DOS, the file may be renamed and made as large or small as needed.

**Note: When the Editor checks for proper file format, it only checks to see that each carriage return (0DH) is where it should be and that the end-of-file marker is present. It is not necessary to have the conversion routine insert line numbers provided the Editor RSEQ command is used to insert line numbers after the program is loaded.

Appendix 3: The Z-80 Instruction Set

Following is a list of Z-80 instructions divided into logic groups. The description of each of these operations may be found in the Z80 Technical Manual provided with your Z80 processor board.

Eight-Bit Load Group

LD r,r'	LD r,n	LD r,(HL)	LD r,(IX+d)
LD r,(IY+d)	LD (HL),r	LD (IX+d),r	LD (IY+d),r
LD (HL),n	LD (IX+d),n	LD (IY+d),n	LD A,(BC)
LD A,(DE)	LD A,(nn)	LD A,I	LD A,R
LD I,A	LD R,A	LD (BC),A	LD (DE),A
LD (nn),A			

Sixteen-Bit Load Group

LD dd,nn	LD IX,nn	LD IY,nn	LD HL,(nn)
LD dd,(nn)	LD IX,(nn)	LD IY,(nn)	LD (nn),HL
LD (nn),dd	LD (nn),IX	LD (nn),IY	LD SP,HL
LD SP,IX	LD SP,IY	PUSH qq	PUSH IX
PUSH IY	POP qq	POP IX	POP IY

Exchange Group and Block Transfer & Search Group

EX DE,HL	EX AF,AF'	EXX	EX (SP),IX
EX (SP),IY	LDI	LDIR	LDD
LDDR	CPI	CPIR	CPD
CPDR	EX (SP),HL		

Eight-Bit Arithmetic And Logical Group

ADD r	ADD n	ADD (HL)	ADD (IX+d)
ADD (IY+d)	ADC s	SUB s	SBC s
AND s	OR s	XOR s	CP s
INC r	INC (HL)	INC (IX+d)	INC (IY+d)
DEC r	DEC (HL)	DEC (IX+d)	DEC (IY+d)

General Purpose Arithmetic & CPU Control Groups

DAA	CPL	NEG	CCF
SCF	NOP	HALT	DI
EI	IM 0	IM 1	IM 2

Sixteen-Bit Arithmetic Group

ADD HL, ss	ADC HL,ss	SBC HL,ss	ADD IX,pp
ADD IY,rr	INC ss	INC IX	INC IY
DEC ss	DEC IX	DEC IY	

Rotate and Shift Group

RLCA	RLA	RRCA	RRA
RLC r	RLC (HL)	RLC (IX+d)	RLC (IY+d)
RL s	RRC s	RR s	SLA s
SRA s	SRL s	RLD	RRD

Bit Set, Reset and Test Group

BIT b,r	BIT b,(HL)	BIT b,(IX+d)	BIT b,(IY+d)
SET b,r	SET b,(HL)	SET b,(IX+d)	SET b,(IY+d)
RES b,r	RES b,(HL)	RES b,(IX+d)	RES b,(IY+d)

Jump Group

JP nn	JP cc,nn	JR e	JR C,e
JR NC,e	JR Z,e	JR NZ,e	JP (HL)
JP (IX)	JP (IY)	DJNZ e	

Call and Return Group

CALL nn	CALL cc,nn	RET	RET cc
RETI	RETN	RST p	

Input and Output Group

IN A,(n)	IN r,(C)	INI	INIR
IND	INDR	OUT (n),A	OUT (C),r
OUTI	OTIR	OUTD	OTDR

Abbreviations defined:

r,r'	=	B, C, D, E, H, L, A
p	=	00H,08H,10H,18H,20H,28H,30H,38H
dd,ss	=	BC, DE, HL, SP
pp	=	BC, DE, IX, SP
qq	=	BC, DE, HL, AF
rr	=	BC, DE, IY, SP
n,nn	=	8-bit,16-bit values respectively
b	=	0,1,2,3,4,5,6,7
s	=	r,n, (HL), (IX+d), or (IY+d)
cc	=	Conditions: NZ,Z,NC,C,PO,PE,P,or M
d	=	r, (HL), (IX+d), or (IY+d)
e	=	any address within 128-byte range