R E L O C A T I N G M A C R O A S S E M B L E R

A N D L I N K E R

for

Z 8 0 A N D H D 6 4 1 8 0

by

Patrick O'Connell

Rev. 6/25/85

WARNING

LIMITED WARRANTY

THIS PROGRAM AND INSTRUCTION MANUAL ARE SOLD "AS IS," WITHOUT WARRANTY AS TO THEIR PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THIS PROGRAM IS ASSUMED BY YOU.

HOWEVER, TO THE ORIGINAL PURCHASER ONLY, ECHELON WARRANTS THE MAGNETC DISKETTE ON WHICH THE PROGRAM IS RECORDED TO BE FREE FROM DEFECTS IN MATERIALS AND FAULTY WORKMANSHIP UNDER NORMAL USE FOR A PERIOD OF THIRTY DAYS FROM THE DATE OF SHIPMENT. IF DURING THIS THIRTY-DAY PERIOD THE DISKETTE SHOULD BECOME DEFECTIVE, IT MAY BE RETURNED TO ECHELON FOR A REPLACEMENT WITHOUT CHARGE.

YOUR SOLE AND EXCLUSIVE REMEDY IN THE EVENT OF A DEFECT IS EXPRESSLY LIMITED TO REPLACEMENT OF THE DISKETTE AS PROVIDED ABOVE. IF FAILURE OF A DISKETTE HAS RESULTED FROM ACCIDENT OR ABUSE ECHELON SHALL HAVE NO RESPONSIBILITY TO REPLACE THE DISKETTE UNDER THE TERMS OF THIS LIMITED WARRANTY.

ANY IMPLIED WARRANTIES RELATING TO THE DISKETTE, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED TO A PERIOD OF THIRTY DAYS FROM DATE OF SHIPMENT. ECHELON SHALL NOT BE LIABLE FOR INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OF THIS PRODUCT. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTIAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATIONS MIGHT NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

ZAS (Z80 and HD64180 Relocating Macro Assembler) reads assembly language statements from a disk file and produces either an Intel compatible HEX file or a Microsoft compatible REL file. These files can then be loaded using Echelon supplied MLOAD, or CP/M LOAD, command or any Microsoft object compatible linker. A symbol table file (SYM) is optionally produced that can be used with Echelon DSD or Digital Research SID and ZSID debuggers.

The minimum Z or CP/M system configuration in which to use ZAS is 48k-bytes of RAM with one disk drive.

As soon as you receive ZAS, make backup copies! Then go through the installation process using a copy.

1.2 DISTRIBUTION FILES

You will find the following files on your distribution disk:

File	Function
ZAS.COM	Assembler
ZLINK.COM	Linker
ZLIB.COM	Library Manager
ZCON.COM	8080 to Z80 Code Converter
ZREF.COM	Cross-reference Generator
TEST.Z80	Test Assembly File
INSTZAS.COM	Installation Program

1.3 INSTALLATION

The installation program was designed to set assembler output options. Type in INSTZAS to invoke the installation program. The options described on the next page will appear on the screen.

INSTZAS<cr>

```
ZAS installation options:

1.  Listing to terminal  - off

2.  Listing to disk file - off

3.  Listing to printer   - off

4.  Generate object file - on

5.  Generate symbol file - off

6.  Object file type     - rel

7.  IF trueness based on - least significant bit

99.  Changes complete

Enter option number to change:
```

The preset values for different options is indicated to right of option. To change (toggle) an option value (i.e., **on** to **off**, **rel** to **hex**, or **least significant bit** to **all sixteen bits**), simply enter option number (1 to 7) followed by carriage return <CR>. When desired option changes have been made, type in 99 to end installation program and have ZAS.COM automatically updated.

1.4 SOFTWARE UPDATES

You can assist in refining ZAS by recommending enhancements and reporting any software problems on a copy of the Software Update Form, a sample of which is in Appendix B. Software updates will be provided at regular intervals for a nominal fee. You will be notified by Echelon when software updates are available.

CHAPTER 2
ZAS INVOCATION

2.1 ZAS OPERATION

ZAS is invoked by typing:

```
ZAS filename.filetype
```

where filename is the name of the source file to be assembled. If no filetype is specified, then Z80 is assumed. Typing ^C will cancel ZAS operation.

2.2 ZAS OPTIONS

A variety of options are available to provide control over the execution parameters of ZAS. They are used once at the end of a command line and spaces are not allowed between options:

```
ZAS filename {$}options
```

There are two types of options: non-disk reference options and disk reference options. Using the non-disk reference options reverses the settings supplied by the Install Program and includes the C, H, and L options.

C: CRT Option. Setting the C option will page the output of ZAS, at 23 lines per page. Pressing any key allows you to continue to scroll through the output page by page. However, it should be noted that a ^C will abort the assembly.

H: Hex Option. When this option is set it will generate Intel compatible hex files instead of Microsoft compatible REL files. Note: When using HEX files, you must have an ORG statement of 100H or higher to prevent an inverted address error from MLOAD or LOAD.COM.

L: Listing to Printer Option. Setting the L option sends a formatted assembly listing to Z or CP/M LST: device.

The disk reference options require two characters. The first character is the P, O, or S option characters. The second character indicates the output disk drive for the specified option. The second character must be A-P or Z, where Z (for zero or null) suppresses the output altogether.

O: Object File Generation (filename.REL or filename.HEX). The O option specifies the disk for object file output. Depending on the H option, the object file will be a Microsoft compatible REL file or an Intel compatible HEX file.

P: Listing to a PRN File (filename.PRN). The P option will send a formatted assembly listing to the specified disk.

S: Symbol File Generation (filename.SYM). The S option specifies output disk for Echelon or DRI compatible SYM file.

2.3 ASSEMBLY STATISTICS

At the completion of an assembly, ZAS provides several statistics on the program assembled. The output is as follows:

```
Assembly statistics:

 nnnn lines
 nnnn labels
 nnnn macros read
 nnnn macro expansions
 nnnn errors
 nnnn free bytes
```

where nnnn is a decimal number.

CHAPTER 3
PROGRAM FORMAT

Acceptable program input consists of a sequence of statements in the form:

label operation operand comment

where each field is separated by one or more spaces and/or tabs. All fields are optional and may begin in any column except for the label field which must begin in column one. The statement is terminated by a carriage return and a line feed is allowed but not necessary. You may also insert blank lines into the program.

The statement may be either upper or lower-case except for macro parameters. For macro parameters, the actual and formal parameters must be in the same case for substitution to take place.

3.1 LABEL FIELD

Labels take the form:

label or label:

and are optional except for the SET, EQU, and MACRO assembler directives. The label consists of alphanumeric characters, a ?, an @, or a $ and the first character must not be numeric. If the label exceeds 15 characters then the label is truncated to the right. Labels can be either upper-case or lower-case. The ":" following a label is optional. Examples of labels include the following:

a123	?a123	@a123
aLL:	?ALL:	update_file
All?	INDEX	UPDATE$FILE

3.2 OPERATION FIELD

The operation field contains one of the following three: a mnemonic machine instruction code, a pseudo operation code which directs the assembly process, or a macro. The Z80 mnemonic machine instruction codes are listed in Appendix A and Hitachi HD64180 instruction codes are listed in Appendix D. The assembler pseudo-op codes are discussed in Chapter 5, with a summary of the pseudo-ops listed Appendix C. And the macro instructions are discussed in Chapter 6.

3.3 OPERAND FIELD

The operand field may contain numeric constants, character constants, ASCII strings, relocation counter references, labels, register references, operators, or expressions containing any combination of the previously mentioned items. Expressions are further described in Chapter 4.

3.4 COMMENT FIELD

A comment field is always preceded by a semicolon (;). Comments are ignored by the assembler but are useful for programmer documentation, and later, debugging.

CHAPTER 4
EXPRESSIONS

Before the pseudo operations and macros can be described, it is necessary to discuss expressions because of their complexity. Expressions consist of simple operands combined into properly formed sub-expressions by operators. Blanks and tabs are ignored between operators and operands of the expression. Each expression produces a 16-bit value during the assembly. If only 8 bits are needed, the least significant half of the 16-bit value is used.

4.1 NUMERIC CONSTANTS

A numeric constant is a 16-bit value in one of several number bases. The base, called the radix of the constant, is denoted by a trailing radix indicator. Any numeric constant which does not terminate with a radix indicator uses the default radix which has been initially set to decimal. The radix indicators are:

B	binary constant	base 2
O	octal constant	base 8
Q	octal constant	base 8
D	decimal constant	base 10
H	hexadecimal constant	base 16

A constant is a sequence of digits, followed by an optional radix indicator, where the digits are appropriate for the radix, i.e., binary constants must be composed of 0 and 1 digits etc. For hexadecimal constants, the leading digit must be a decimal digit in order to avoid confusing the hexadecimal constant with an identifier (a leading 0 will work). A numeric constant must produce a binary number which can be contained within a 16-bit value.

4.2 ASCII STRINGS

String constants represent sequences of ASCII characters, and are represented by enclosing the characters within apostrophe symbols ('). All strings must be fully contained within the current physical line. The apostrophe character itself can be included within a string by representing it as a double apostrophe (''), which becomes a single apostrophe when read by the assembler.

4.3 CHARACTER CONSTANTS

Like strings, character constants are composed of 0, 1, or 2 ASCII characters, delimited by an apostrophe (') or quotation (") symbol. One difference between strings and character constants is strings are used only with DB, DC, DEFB, and all macro pseudo-

ops. In all other cases, a character constant is assumed. Another difference is that the value of a character constant is calculated and the result is stored with the low byte in the first address and the high byte in the second address. For example, in the character constant:

```
DW 'AB'
```

the value of A is stored in the second memory location and B is stored in the first memory location. In the string:

```
DB 'AB'
```

the value of A is stored in the first memory location and B is stored in the second memory location.

4.4 LABELS

A label is given a value determined by the type of statement it precedes. If the label precedes a macro definition, the label is given a text value, which is the body of the macro definition. If the label precedes an EQU or SET pseudo operation, then the label is given the value of the operand field. If a label precedes any other type of statement, it is given the value of the current relocation counter.

The value of a label is not allowed to change unless the label precedes a SET pseudo-op. In which case, there is no limit to the number of times the label's value may change.

4.4.1 LABEL CHARACTERISTICS

Labels fall into one of three categories: public, external, or local. Public labels are labels defined in the current program module and can be referenced in other program modules External labels are labels which have been defined as public in some other program module and are being referenced in the module declaring them external. If a label has not been declared external or public then it is local and cannot be referenced by any other program module.

4.4.2 RELOCATION BASES

The symbolic names for independently located memory areas are called relocation bases. These relocation bases may represent ROM, shared COMMON areas, special memory areas such as video refresh, memory mapped I/O, etc. Within each sub-program, each of these memory areas is referenced by a unique name. The actual allocation and mapping of the name to physical addresses is deferred to the link edit and load process. All label references within the assembled program are relative to one of these relocation bases. The four relocation bases and their typical uses are summarized as follows:

Absolute: Absolute assembles non-relocatable code. A programmer selects Absolute mode when a block of program code is to be loaded each time into specific addresses, regardless of what else is located at the same time.

Data Relative: Data Relative assembles code for a section of a program that may change and therefore must be loaded into RAM. This applies especially to program data areas. Symbols in Data Relative are relocatable.

Code Relative: Code (program) Relative assembles code for sections of programs that will not be changed and therefore can be loaded into ROM/PROM. Symbols in Code Relative are relocatable.

COMMON: COMMON assembles code that is loaded into a defined common data area. This allows program modules to share a block of memory and common values.

To change the relocation base, use one of the following pseudo-ops in a statement line:

ASEG	Absolute
DSEG	Data Relative
CSEG	Code Relative--default
COMMON	COMMON

4.5 RELOCATION COUNTER REFERENCE

The current relocation counter may be referenced as a 16-bit value by use of the symbol $. The value represented by $ is always the relocation counter value at the start of the current statement. For example,

```
JP $
```

will endlessly jump to itself.

4.6 REGISTERS

When ZAS encounters a one or two character symbol, it will look up the symbol in the corresponding 8 or 16-bit register table (see the next page). If the symbol is found, then the operand is assumed to be a register reference. Because these single and double character symbols are reserved words, do not use them as labels.

8-Bit Registers (Reserved Words)	16-Bit Registers (Reserved Words)
A	BC
B	DE
C	HL
D	IX
E	IY
H	SP
L	AF
M	
I	
R	

4.7 OPERATORS

The operands previously described can be combined in normal algebraic expression using any combination of properly formed operands, operators, and parenthesized expressions. All arithmetic operators (+,-,*,/,MOD,SHL, and SHR) produce a 16-bit unsigned arithmetic result. The relational operators (EQ, LT, LE, GT, GE, and NE) produce a true (0FFFFH) or false (0000H) 16-bit result. And the logical operators (NOT, AND, OR, and XOR) operate bit-by-bit on their operand(s) producing a 16-bit result of 16 individual bit operations. The HIGH and LOW operators always produce a 16-bit result with a high order byte which is zero. The NUL operator produces a true or false result.

The operators for the operand field are given below. In general, the letters x and y represent operands which are treated as 16-bit unsigned quantities in the range 0-65535.

Arithmetic Operators	Result
x+y	arithmetic sum of x and y
x-y	arithmetic difference between x and y
x * y	unsigned multiplication of x by y
x / y	unsigned division of x by y
x MOD y	remainder after division of x by y
x SHL y	shift left by y, with zero right fill
x SHR y	shift right by y, with zero left fill

Relational Operators	Result
x EQ y, x=y	true if x equals y, false otherwise
x LT y, x<y	true if x is less than y, false otherwise
x LE y, x<=y	true if x is less or equal to y, else false
x GT y, x>y	true if x is greater than y, false otherwise
x GE y, x>=y	true if x is greater or equal to y, else false
x NE y, x<>y	true if x is not equal to y, false otherwise

Logical Operators	Result
NOT y	bit-by-bit logical inverse of y
x AND y	bitwise logical AND of x and y
x OR y, x!y	bitwise logical OR of x and y
x XOR y	logical exclusive OR of x and y

Special Operators	Result
HIGH y	identical to y SHR 8 (high order byte of y)
LOW y	identical to y AND OFFH (low order byte of y)
NUL line	true if the remainder of the current line is null or contains only space and/or tab characters. Because the NUL operator uses the rest of the current source line as an operand, it must be the last operator on a line.

4.8 PRECEDENCE OF OPERATORS

Without parentheses or brackets operators have an order of application as if they were parenthesized or bracketed. As described below, the operators listed first have highest precedence, and the operators listed last have lowest precedence. Operators listed on the same line have equal priority and are applied from left to right in the expression

highest precedence	* / MOD SHL SHR
	+ -
	EQ LT LE GT GE NE
	NOT
	AND
	OR XOR
	HIGH LOW
lowest precedence	NUL

The expressions shown below are equivalent:

```
x + y * z  =  x + [y * z]
x OR y * a SHR b  =  x OR [y * [a SHR b]]
```

Balanced parenthesized or bracketed sub-expressions can always be used to override the order of precedence described above. The last expression could be rewritten to force application of operators in a different order:

```
[x OR y] * [a SHR b]
```

4.9 PARENTHESES VERSUS BRACKETS

Parentheses and brackets are not interchangeable. They serve different purposes. Parentheses are used in expressions that have indirect addressing modes. For example,

```
LD HL, (5+1)
```

will load the register pair HL from the contents of memory location six (5+1) and seven.

Brackets are used for all other expressions where the addressing mode is not indirect. Using the above example with brackets,

```
LD HL, [5+1]
```

will load the register pair HL with the immediate value six.

4.10 EXPRESSION RESTRICTIONS

The operand field of a statement may consist of a complex arithmetic expression with the following restrictions:

(1) An external may only have an absolute quantity added or subtracted from it. The result will be external.

(2) A relocatable value may have an absolute or another relocatable value (in the same relocation base) added to or subtracted from it. The result will relocatable.

(3) If two relocatable values are subtracted then the result will be absolute.

(4) In all other arithmetic and logical operations, both operands must be absolute. The result will be absolute.

An expression error will be generated if an expression does not follow the above restrictions.

CHAPTER 5
PSEUDO-OPS

5.1 GENERAL PSEUDO-OPS

DB: The Define Byte pseudo-op is used to enter one or more one-byte data values into the program. The statement form is:

```
DB n {,n...}
```

where n is any expression with a valid 8-bit value. More than one byte can be defined at a time by separating it from the preceding value with a comma. All of the bytes defined in a single DB statement are assigned consecutive memory locations. The Zilog mnemonic DEFB can be used instead of DB.

DC: The Define Character pseudo-op stores the characters in a string in successive memory locations beginning with the current relocation counter. The most significant bit of the last character will be set to one. The form for the DC pseudo-op is:

```
DC 'string'
```

DS: The Define Space pseudo-op reserves an area of memory. The form is:

```
DS  expression {,expression}
```

where the value of the first expression gives the number of bytes to be reserved. The Zilog mnemonic DEFS can be used instead of DS.

To initialize the reserved space, set the optional second expression to the value desired. If the second expression is omitted, the reserved space is left as is (uninitialized). The reserved block of memory is not automatically initialized to zeros. To initialize to zeros give the second expression the value 0.

All names used in the first expression must be previously defined on pass 1. Otherwise, a U error (undefined symbol) is generated during pass 1, and a P error (phase error) will probably be generated during pass 2 because the DS pseudo-op generated no code on pass 1.

DW: The Define Word directive is used to enter a 16-bit value into the program. This directive takes the form:

```
DW nn {,nn...}
```

Where nn is any expression with a valid 16-bit value. Multiple 16-bit values may be defined with one DW statement by separating the values with a comma. All 16-bit values defined by the DW pseudo-op are stored in standard Z80 word format with the least significant byte first. The Zilog mnemonic DEFW can be used instead of DW.

END: The END statement is optional. All statements following the END are ignored. The form is:

```
END    {expression}
```

The optional expression is the program starting address. If an Intel compatible hex file is being generated, then this starting address will be included in the last record of the hex file. If a REL file is being generated, then ZLINK will place a JUMP instruction at 100H to the specified starting address.

EQU: The EQUate statement is used to name synonyms for particular numeric values. The form is:

```
label     EQU     expression
```

The label must be present and cannot label any other statement. The assembler evaluates the expression and assigns this value to the label. The label is usually a name which describes the value of the expression. Also, this name can be used throughout the program as a parameter or operand.

.IN: The INsert (or MACLIB) pseudo-op allows the programmer to use the same section of assembler source code in a number of different assemblies. The format is:

```
.IN {d:}filename  or  MACLIB {d:}filename
```

where d is the optional Z or CP/M disk specifier (defaulting to the logged disk) and filename is the file on disk with the assumed filetype LIB.

This directive causes the specified file to be copied into the assembly in its entirety, and to be treated exactly as if it were part of the original source file. All inserted source lines are flagged with a "+" on the listing. Only one level of insert is allowed, they cannot be nested.

.LIST: This pseudo-op resumes a listing which has been suppressed by the .XLIST directive. See the next page.

PAGE: The page pseudo-op gives control over the output formatting which is sent to the PRN file and/or directly to Z or CP/M LST: device. The form for the PAGE statement is:

```
PAGE   {expression}
```

If the PAGE statement is used without the optional expression then a form feed is sent to the output file and/or Z or CP/M LST: device. The form feed is sent before the statement with PAGE has been printed. Consequently, the PAGE command is often issued directly ahead of major sections of an assembly language program, such as a group of subroutines, to cause the next statement to appear at the top of the following printer page.

The second form of the PAGE command is used to specify the output

page size. In this case, the expression which follows the PAGE pseudo-op determines the number of output lines to be printed on each page. If the expression equates to a value between 40 and 90, then the page size is set to the value of the expression. When this value is reached for each page, a form feed is issued to cause a page eject. The assembler initially assumes a 56 line page size and produces a page eject at the beginning of the

listing. Usually, no more than one PAGE statement with the expression option is included in a particular program.

.RADIX: The statement form is:

```
.RADIX n
```

where n is 2, 8, 10, or 16. This pseudo-op sets the radix to n for all numbers which follow, unless another .RADIX statement is encountered, or the radix is overridden by a suffix radix modifier. Initially, the default radix is set to 10 (decimal).

SET: The SET statement is used to name synonyms for particular numeric values. The form is:

```
label     SET     expression
```

The label must be present and cannot label any other statement, except for another SET. The assembler evaluates the expression and assigns this value to the label. The label is usually a name which describes the value of the expression. Also, this name can be used throughout the program as a parameter or operand. The Zilog mnemonic DEFL can be used instead of SET.

.TITLE and .SBTTL: The title and subtitle pseudo-ops take the form:

```
.TITLE 'string-constant 1'
.SBTTL 'string-constant 2'
```

where the string-constants are an ASCII string, enclosed in apostrophes, which do not exceed 64 characters. If a .TITLE and/or .SBTTL is encountered during the assembly, then each page of the listing is prefixed with the title and/or subtitle string-constant. The title line will be preceded by a standard ZAS header as follows:

```
MITEK Relocating Macro Assembler vers n.n        page nnn
string-constant 1
string-constant 2
```

where n.n is the ZAS version number, nnn is the current page number and string-constant 1 and/or 2 is the string given in the corresponding pseudo-op. ZAS initially assumes that these pseudo-ops are not in effect. When specified, the title line, along with the subtitle line are not included in the line count for the page. Usually, no more than one .TITLE statement is included in a particular program.

5.2 LISTING CONTROL PSEUDO-OPS

.LALL: List ALL macro lines, including lines that do not generate code.

.LIST: This pseudo-op resumes a listing which has been suppressed by the .XLIST directive.

.LFCOND: The List False CONDitionals pseudo-op assures the listing of conditional expressions that evaluate false.

.PRINT: The print on console pseudo-op takes the form:

```
.PRINT    pass,text
```

This pseudo-op will output text to the console during the specified pass. The pass can be one of three values:

0 - print text during both passes

1 - print text during pass one

2 - print text during pass two

.SALL: Suppress ALL of the macro listing, including all text and object code produced by macros.

.SFCOND: The Suppress False CONDitionals pseudo-op suppresses the portion of the listing that contains conditional expressions that evaluate false.

.XALL: The EXclude ALL non-code macro lines pseudo-op will list source and object code produced by a macro, but source lines which do not generate code are not listed.

.XLIST: This pseudo-op suppresses all list output until a .LIST pseudo-op is encountered.

5.3 CONDITIONAL ASSEMBLY PSEUDO-OPS

The next two sections describe the ZAS conditional assembly facility.

5.3.1 IF PSEUDO-OP EVALUATION

ZAS has two different methods for evaluating the trueness of an IF expression. One method bases the trueness on the least significant bit of the IF expression, which is compatible with Digital Research's ASM, MAC, and RMAC assemblers. The second method bases the trueness of the expression on the full 16-bit expression value. This method is compatible with the Microsoft M80 assembler.

The default evaluation is set by the installation program (section 1.3). The evaluation method may also be explicitly set by the following two pseudo-ops:

.IF1 - will cause IF expressions to evaluate to true if the least significant bit of the IF expression evaluates to 1.

OR

.IF16 - will cause IF expressions to evaluate to true when the IF expression evaluates to non-zero.

5.3.2 CONDITIONAL ASSEMBLY FORMS

The IF, ELSE, and ENDIF pseudo-ops define a range of assembly language statements which are to be included or excluded during the assembly process. The IF and ENDIF statements alone can be used to bound a group of statements to be conditionally assembled thus:

```
IF      expression
statement #1
statement #2
      .
      .
      .
statement #n
ENDIF
```

Upon encountering the IF statement, the assembler evaluates the expression following the IF (all operands in the expression must be defined ahead of the IF statement). Depending on the conditional assembly option in effect, if the expression evaluates to a non-zero value or the least significant bit evaluates to a 1, then statement #1 through statement #n are assembled. If the expression evaluates to a zero, then the statements are listed but not assembled.

The ELSE statement can be used as an alternative to an IF statement, and must occur between the IF and ENDIF statements. The form is:

```
IF      expression
statement #1
statement #2
      .
      .
      .
statement #n
ELSE
statement #n+1
statement #n+2
      .
      .
      .
statement #m
ENDIF
```

If the expression produces a non-zero (true) value, then statements 1 through n are assembled. However, statements n+1 through m are skipped in the assembly process. When the expression produces a zero value (false), statements 1 through n are skipped, while statements n+1 through m are assembled. As an example, the conditional assembly shown in Listing A could be rewritten as shown in Listing B.

Listing A

```
TTY     EQU     1
CRT     EQU     2
DEVICE  EQU     TTY
TTYOUT  EQU     0F003H
CRTOUT  EQU     0F100H
        IF      DEVICE  EQ      TTY
        CALL    TTYOUT
        ENDIF
        IF      DEVICE  EQ      CRT
        CALL    CRTOUT
        ENDIF
```

Listing B

```
TTY     EQU     1
CRT     EQU     2
DEVICE  EQU     TTY
TTYOUT  EQU     0F003H
CRTOUT  EQU     0F100H
        IF      DEVICE  EQ      TTY
        CALL    TTYOUT
        ELSE
        CALL    CRTOUT
        ENDIF
```

Properly balanced IF's, ELSE's, and ENDIF's can be completely contained within the boundaries of outer encompassing conditional assembly groups. The structure outlined below shows properly nested IF, ELSE, and ENDIF statements:

```
IF      exp#1
group #1
IF      exp#2
group#2
ELSE
group#3
ENDIF
group#4
ELSE
group#5
IF     exp#3
group#6
ENDIF
group#7
ENDIF
```

where group 1 through 7 are sequences of statements to be conditionally assembled, and exp#1 through exp#3 are expressions which control the conditional assembly. If exp#1 is true, then group#1 and group#4 are always assembled, and group 5,6, and 7 will be skipped. Further, if exp#1 and exp#2 are both true, then group#2 will also be included in the assembly, otherwise group#3 will be included. If exp#1 produced a false value, groups 1, 2, 3, and 4 will be skipped, and group 5 and 7 will always be assembled. If under these circumstances, exp#3 is true then group#6 will also be included with 5 and 7, otherwise it will be skipped in the assembly.

Conditional assembly of this sort can be nested up to eight levels (i.e., there can be up to eight pending IFs or ELSEs with unresolved ENDIFs at any point in the assembly), but usually becomes unreadable after two or three levels of nesting. The nesting level restriction also holds for pending IFs and ELSEs during macro evaluation. Nesting level overflow will produce an error during assembly.

5.4 LINKAGE PSEUDO-OPS

EXTRN: The EXTeRNal pseudo-op identifies symbols which are defined in some other program but are used in the current program. The form is:

EXTRN symbol {,symbol...}

where symbol is the symbol being declared as external. Multiple symbols may be declared in the same statement by separating them with commas. Also, if a symbol in an expression is suffixed with one or two # signs, then the symbol is treated as an external. EXT is a synonym for EXTRN.

NAME: The NAME pseudo-op takes the form:

NAME symbol

where symbol is the relocatable module name. This name is used by the linking loader and library manager to identify the module for selective loading or manipulation. Only the first six characters are significant in the module name. In the absence of the NAME pseudo-op, up to the first six characters of the program name are used.

PUBLIC: The PUBLIC pseudo-op identifies those symbols within the current program which are to be made accessible to other programs as external symbols. This directive has no effect on the assembly process for the current program, but merely records the name and value of the identified symbols on the object file for later use by the linking loader. A public symbol must be defined within the current program as a label.

.REQUEST: Request a library search. The form is:

.REQUEST filename {,filename...}

This pseudo-op sends a request to ZLINK or any Microsoft compatible loader to search the filenames in the list for undefined external symbols. The filename in the list should not include filetypes or device designation. ZLINK assumes the default extension .REL and the currently logged disk drive.

5.5 RELOCATION BASE PSEUDO-OPS

ASEG: The Absolute SEGment pseudo-op never has operands. ASEG generates non-relocatable code.

ASEG sets the location counter to an absolute segment (actual address) of memory. The ASEG will default to 0, which could cause the module to write over part of the operating system. It is recommended that each ASEG be followed with an ORG statement set at 100H or higher.

COMMON: COMMON statements are non-executable, storage allocating statements. COMMON assigns variables, arrays, and data to a storage area called COMMON storage. This allows various program modules to share the same storage area. The length of a COMMON area is the number of bytes required to contain the variables, arrays, and data declared in the COMMON block, which ends when another relocation base pseudo-op is encountered.

CSEG: The Code SEGment directive never has an operand. Code assembled in Code Relative mode can be loaded into ROM/PROM.

CSEG resets the location counter to the code relative segment of memory. The location will be that of the last CSEG (default to 0), unless an ORG is done after the CSEG to change the location.

However the ORG statement does not set a hard absolute address under CSEG mode. An ORG statement under CSEG causes the assembler to add the number of bytes specified by the expression argument in the ORG statement to the last CSEG address loaded. For example, if ORG 25 is given, 25 bytes will be added to the current CSEG location. Then CSEG will be loaded. The clearing effect of the ORG statement following CSEG (and DSEG) can be used to give the module an offset. Rationale for not allowing ORG to set an absolute address for CSEG is to keep the CSEG relocatable.

CSEG is the default mode of the assembler. Assembly begins with a CSEG automatically executed, and the location counter in the Code Relative mode, pointing to location 0 in the Code Relative segment of memory. All subsequent instructions will be assembled into the Code Relative segment of memory until ASEG, DSEG, or COMMON is executed. CSEG is then entered to return the assembler to Code Relative mode, at which point the location counter returns to the next free location in the Code Relative segment.

DSEG: The Data SEGment pseudo-op never has operands. DSEG specifies segments of assembled relocatable code that will later be loaded into RAM only.

DSEG sets the location counter to the Data Relative segment of memory. The location of the data relative counter will be that of the last DSEG (default is 0), unless an ORG is done after the DSEG to change the location. However, the ORG statement does not set a hard absolute address under DSEG mode. An ORG statement under DSEG causes the assembler to add the number of bytes specified by the expression in the ORG statement to the last DSEG address loaded. For example, if ORG 25 is given, 25 bytes will be added to the last DSEG address loaded. Then the DSEG will be loaded. The clearing effect of the ORG statement following DSEG (and CSEG) can be used to give the module an offset. Rational for not allowing ORG to set an absolute address for DSEG is to keep the DSEG relocatable.

ORG: The Set ORGin pseudo-op allows the value of a location counter to be changed at any time. The form is:

```
ORG expression
```

Under the ASEG program counter mode, the relocation counter is set to the value of the expression, and the assembler assigns generated code starting with that value. Under CSEG, DSEG, and COMMON relocation bases, the location counter for that base is incremented by the value of the expression. All names used in the expression must be known on pass 1, and the value must either be absolute or in the same relocation base as the current location counter.

.PHASE/.DPHASE: The form is:

```
.PHASE    expression
   .
   .
   .
.DPHASE
```

where expression is an absolute value. .PHASE allows code to be located in one area, but executed at a different area with a start address specified by expression. .DPHASE is used to indicate the end of the relocated block of code.

The relocation base within a .PHASE block is absolute, the same as the mode of the expression in the .PHASE statement. The code, however, is loaded in the area in effect when the .PHASE statement is encountered. The code within the block is later moved to the address specified by expression for execution.

This example,

```
                        .PHASE      300H
                        CALL        DUMMY
                        JP          ENTRY
                DUMMY:  RET
                        .DPHASE
                ENTRY:  JP          0
```

assembles to:

```
0300                    .PHASE      300H
0300    CD0630          CALL        DUMMY
0303    C30700          JP          ENTRY
0306    C9      DUMMY:  RET
0007                    .DPHASE
0007    C30000  ENTRY:  JP          0
```

5.6 MACRO PSEUDO-OPS

Provided here is only a brief description of the macro pseudo-ops. For a more complete description, see the next chapter.

Pseudo-op	Description
ENDM	End Macro
EXITM	Exit Macro
IRP	Indefinite Repeat
IRPC	Indefinite Repeat Character
LOCAL	Local Symbol Generation
REPT	Repeat
MACRO	Macro Definition

5.7 SPECIAL FUNCTION PSEUDO-OPS

.HD64: This pseudo-op enables ZAS to assemble the ten extended instructions of the Hitachi HD64180 microprocessor, upward Z80 compatible. The ten instructions and their forms are listed in Appendix D.

CHAPTER 6
MACRO FACILITY

A common characteristic of assembly language programs is that many coding sequences are repeated over and over with only one or two of the operands changing. Macros provide a mechanism for generating the repeated sequences with a single statement. The repeated sequences are written with dummy values for the changing operands. A single statement, referring to the macro by name and providing values for the dummy operands, can then generate the repeated sequence.

The coding sequence begins with either the macro definition pseudo-op or one of the repeat pseudo-ops and ends with the ENDM pseudo-op. All of the macro pseudo-ops may be used inside a macro sequence. The one exception is a stored macro which, cannot be defined inside a repeat type macro. Macro nesting is allowed up to 15 levels deep.

The macro facility includes pseudo-ops for:

```
macro definition:
    MACRO (macro definition)

repetitions
    REPT (repeat)
    IRP (indefinite repeat)
    IRPC (indefinite repeat character)

terminations:
    ENDM (end macro)
    EXITM (exit macro)

unique symbols within macro sequences:
    LOCAL

operators:
    &
    ;;
    ^
    %
    <>
```

6.1 REPEAT (OR INLINE) MACROS

The simplest macro facilities involve the REPT, IRPC, and IRP macro groups. All these forms cause the assembler to repetitively re-read portions of the source program under control of a counter or list of textual substitutions. These groups are listed in increasing order of complexity.

REPT-ENDM GROUP: The REPT-ENDM group is written as a sequence of assembly language statements starting with the REPT pseudo-op and terminated by an ENDM pseudo-op. The form is:

```
label: REPT expression
    .
    .
    .
label: ENDM
```

where the labels are optional, and the expression indicates the number of times the sequence of statements between REPT and ENDM will be repeated. The expression is evaluated as a 16-bit unsigned number. If the expression contains an external symbol or undefined operands, an error is generated.

In general, if a label appears on the REPT statement, its value is the first machine code address which follows. This REPT label is not re-read on each repetition of the loop. The optional label on the ENDM is re-read on each iteration and thus constant labels (not generated through concatenation or with LOCAL pseudo-ops) will generate phase errors if the repetition count is greater than 1.

IRPC-ENDM GROUP: Similar to the REPT group, the IRPC-ENDM group causes the assembler to re-read a bounded set of statements. The form is:

```
label: IRPC identifier,string
    .
    .
    .
label: ENDM
```

where the optional labels follow the same conventions as in the REPT-ENDM group. The identifier is any valid symbol and string denotes a string of characters, terminated by a delimiter (space, tab, end-of-line, or comment).

The sequence of statements between IRPC and ENDM are repeated once for each character in the string. Each repetition substitutes the next character in the string for every occurrence of identifier in the sequence.

IRP-ENDM GROUP: The IRP is similar in function to the IRPC, except that the controlling identifier can take on a multiple string value. The form is:

```
label: IRP identifier, string {,string...}
    .
    .
    .
label: ENDM
```

where the optional labels follow the conventions of the REPT and IRPC groups. The sequence of statements between IRP and ENDM is repeated for each string. On the first iteration, the string is

substituted for the identifier wherever the identifier occurs in the sequence of statements. On the second iteration, the second string becomes the value of the controlling identifier and so on until the last string is encountered and processed.

6.2 STORED MACROS

MACRO DEFINITION: The form for the macro definition is:

```
macname MACRO dummy{,dummy...}
          .
          .
          .
        ENDM
```

The sequence of statements from the MACRO statement line to the ENDM statement line comprises the body of the macro, or the macro's definition. The macname is any non-conflicting assembly language label. Dummy parameter is a place holder that is replaced by an actual parameter in a one for one text substitution when the MACRO sequence is used.

The prototype statements are read and stored in the assembler's internal tables under the name given by "macname", but are not processed until the macro is expanded.

A comment preceded by two semicolons is not saved as part of the macro definition. But a comment preceded by only one semicolon is preserved and will appear in the expansion.

6.3 EXITING MACROS

The EXITM pseudo-op is used inside a MACRO or Repeat block to terminate an expansion when some condition makes the remaining expansion unnecessary or undesirable. Usually, EXITM is used in conjunction with a conditional pseudo-op.

The expansion is exited immediately when an EXITM is assembled. Any remaining expansion or repetition is not generated. If the block containing the EXITM is nested within another block, the outer level continues to be expanded.

6.4 LOCAL SYMBOLS

The LOCAL pseudo-op is allowed only inside a MACRO definition. The form for the LOCAL directive is:

```
LOCAL identifier {,identifier...}
```

When LOCAL is executed, ZAS creates a unique symbol for each identifier and substitutes that symbol for each occurrence of the identifier in the expansion. These unique symbols are usually used to define a label within a macro. This eliminates multiple-

defined labels on successive expansions of the macro. The symbols created by ZAS range from ??0001 to ??9999. Users should avoid the form ??nnnn for their own symbols. A LOCAL statement must precede all other types of statements in the macro definition.

6.5 MACRO INVOCATION

The form for the macro invocation is:

macname parameter{,parameter...}

Upon recognition of the macname, ZAS "pairs-off" each dummy parameter in the MACRO definition with the actual parameter text, i.e., the first dummy parameter is associated with the first actual parameter, the second dummy is associated with the second actual, and so on until the list is completed. If more actuals are provided than dummy parameters then the extras are ignored. If fewer actuals are provided, then the extra dummy parameter are associated with the empty string, i.e., a text string of zero length. It is important to realize at this point that the value of dummy parameter is not a numeric value, but is instead a textual value consisting of a sequence of zero or more ASCII characters.

6.6 PARAMETER EVALUATION

There are several options available in the construction of actual parameters, as well as in the specification of character lists for the IRP group. Although an actual parameter is simply a sequence of characters placed between parameter delimiters, these options allow overrides where delimiter characters themselves become a part of the text. In general, a parameter x occurs in the context:

label: macname ...,x,...

where the label is optional and the macname is the name of a previously defined macro. The ellipses (...) represent optional surrounding actual parameters in the invocation of macname. In the case of an IRP group, the occurrence of a character list x would be:

label: IRP id, ...,x,...

where the label is optional, and the ellipses represent optional surrounding character lists for substitution within the IRP group where the controlling identifier "id" is found. In either case, the statements could be contained within the scope of a surrounding macro expansion. Therefore, dummy parameter substitution could take place for the encompassing macro while the actual parameter is being scanned.

ZAS follows these steps in forming an actual parameter or character list:

(1) Leading blanks and tabs are removed when they occur in front of x.

(2) The leading character of x is examined to determine the type of scan operation which is to take place.

(3) If the leading character is a string quote, then x becomes the text up through and including the balancing string quote, using the normal string scanning rules: double apostrophes within the string are reduced to a single apostrophe, and upper case dummy parameters adjacent to the ampersand symbol are substituted by their actual parameter values. Note that the string quotes on either end of the string are included in the actual parameter text.

(4) If instead the first character is the left caret (<) then the bracket is removed, and the value of x becomes the sequence of characters up to , but not including, the balancing right caret (>) which does not become part of x. In this case, left and right carets may be nested to any level within x, and only the outer carets are removed in the evaluation. Quoted strings within the carets are allowed, and substitution within these strings follows the rules stated in (3) above. Note that left and right carets within quoted strings become a part of the string, and are not counted in the caret nesting within x. Further, the delimiter characters comma, blank, semicolon, and tab, become a part of x when they occur within the caret nesting.

(5) If the leading character is a %, then the sequence of characters which follows is taken as an expression which is evaluated immediately as a 16-bit value. The resulting value is converted to a decimal number and treated as an ASCII sequence of digits, with left zero suppression (0-65535).

(6) If the leading character is none of the above (quote, left bracket, or percent), the sequence of characters which follow, up to the next comma, blank, tab, or semicolon, becomes the value of x.

There is one important exception to the above rule: the single character escape, denoted by an up-arrow, causes ZAS to read the character immediately following as a part of x without treating the character as significant. However, the character which follows the up-arrow, must be a blank, tab, or visible ASCII character. The up-arrow itself can be represented by two up-arrows in succession. If the up-arrow directly precedes a dummy parameter, then the up-arrow is removed and the dummy parameter is not replaced by its actual parameter value. Thus, the up-arrow can be used to prevent evaluation of dummy parameters within the macro body. Note that the up-arrow has no special significance within string quotes, and is simply included as a part of the string.

Evaluation of dummy parameters in macro expansions must also be considered, although this topic has been presented throughout the

previous sections. Generally the macro assembler evaluated dummy parameters as follows:

(1) If a dummy parameter is either preceded or followed by the concatenation operator (&), then the preceding and/or following "&" operator is removed, the actual parameter is substituted for the dummy parameter, and the implied delimiter is removed at the position(s) the ampersand occurs.

(2) Dummy parameters are replaced only once at each occurrence as the encompassing macro expands. This prevents the "infinite substitution" which would occur if a dummy parameter evaluated itself.

In summary, parameter evaluation follows these rules:

- leading and trailing tabs and blanks are removed
- quoted strings are passed with their string quotes intact
- nested carets enclose arbitrary characters with delimiters
- a leading % causes immediate numeric evaluation
- an up-arrow passes a special character as a literal value
- an up-arrow prevents evaluation of a dummy parameter
- the "&" operator is removed next to a dummy parameter
- dummy parameters are replaced only once at each occurrence

CHAPTER 7
ZAS ERROR MESSAGES

There are two types of error messages: Non-fatal errors and fatal errors. Non-fatal errors are indicated by a single letter code to the left of the statement line with the error. Fatal errors kill the assembly and give messages as to why the error may have occurred. Statement lines with errors will not generate object code.

7.1 NON-FATAL ERRORS

Error Code	Explanation
A	**Argument error.** One of the arguments for the op-code is invalid.
B	**Balance error.** An ELSE or an ENDIF pseudo-op does not have a preceding IF statement. Or an END macro statement has no preceding macro call and/or macro definition.
C	**Character is invalid.** ZAS has found an invalid character and it is probably a control character. The invalid character will be replaced by a "^".
D	**Duplicate error.** A label has been defined more than once.
E	**Expression error.** The expression is ill-formed and cannot be computed.
I	**Insert error.** The specified insert file cannot be found or an insert is already in progress.
M	**Mode error.** The statement contains an addressing mode error.
O	**Opcode error.** The statement contains an illegal opcode.
P	**Phase error.** A label has a different value on Pass 2 than it did on Pass 1.
S	**Syntax error.** The assembly statement is ill-formed and cannot be processed. This error may be due to invalid characters or delimiters which are out of place.
U	**Undefined symbol.** A label argument has not been defined in the program.
V	**Value error.** The operand (argument) is out of its allowable range.

7.2 FATAL ERRORS

Fatal error messages have been classified into two categories: errors caused by macros and general errors (or errors not caused by macros).

7.2.1 GENERAL FATAL ERROR MESSAGES

(1) **"Filename.filetype not found."**
The specified source file cannot be found on the disk.

(2) **"Invalid option specification."**
One or more of the assembler options specified in the command line is invalid.

(3) **"More than eight IF levels are pending at line nnnn"**
Where line nnnn is the line with the ninth IF. A maximum of eight IF levels can be nested.

(4) **"Unterminated IF!"**
The end of file has been reached with no terminating ENDIF.

(5) **"Memory full at line nnnn"**
The assembler's internal tables have run out of memory.

7.2.2 MACRO FATAL ERROR MESSAGES

(1) **"Unterminated macro starting at line nnnn"**
Where line nnnn is the line with the error. This error is caused by a macro definition that has no terminating END macro statement.

(2) **"Local label limit exceeded!"**
The maximum of 9,999 local symbols has been exceeded.

(3) **"Macro nested past 16 levels at line nnnn"**
A maximum of 16 levels of nested macros are allowed.

(4) **"Local table exceeds 127 bytes at line nnnn"**
The total length of all local symbols cannot exceed 127 bytes for a particular macro definition.

(5) **"Macro definition inside an inline macro at line nnnn"**
This message indicates that a macro definition has been placed inside a repeat type macro and that is not allowed.

CHAPTER 8
CROSS-REFERENCE GENERATION

8.1 OVERVIEW

The cross-reference generator (ZREF) is used to provide a summary of symbol usage throughout a program. ZREF reads the file specified line by line, attaches a line number prefix to each line, and writes each prefixed line to the file filename.XRF. After completing this operation, ZREF appends to the file filename.XRF, a cross-reference report that lists all the line numbers where each symbol in the file appears. It also flags with an *, each line number where the referenced symbol is defined.

8.2 ZREF OPERATION

ZREF is invoked by typing

ZREF filename.filetype {$}option

where filename.filetype is the name of the file to be cross-referenced with the assumed filetype .Z80, and option is the letter L, if the output is to the list device instead of a file.

8.3 RESERVED SYMBOLS

The following symbols will not be part of the cross reference:

A	HI	NUL
AF	HL	NOT
AND	I	NZ
B	IX	OR
BC	IY	P
C	L	PE
D	LE	PO
DE	LOW	R
E	LT	SHL
EQ	M	SHR
GE	MOD	SP
GT	NC	XOR
H	NE	Z

CHAPTER 9
CODE CONVERTER

9.1 CODE CONVERTER OPERATION

The code converter (ZCON) converts 8080 source statements, all of the TDL machine instruction statements, and most of the common TDL pseudo-ops to Z80 source statements (see the next section for a listing of the convertible TDL pseudo-ops). In addition, except for character-constants, ASCII strings, and comments, parentheses are converted to brackets. Also, parity bit (bit 7) is zeroed.

To invoke the code converter, type:

ZCON filename.filetype {$}u

where filename is the name of the source file to be converted. If no filetype is specified, then ASM is assumed. When the "u" option is specified, only upper-case conversion is done. This is useful if you already have a Z80 source file in lower case. When the conversion is completed, the output will be in a file called filename.Z80 and one of two messages will be displayed.

Message 1:
"nnnn lines converted, with no errors detected."
Where nnnn is the number of lines converted.

OR

Message 2:
"nnnn lines converted, with eee errors logged in filename.ERR"
Where nnnn is the number of lines converted and eee is the number of errors detected.

9.2 CONVERTIBLE TDL PSEUDO-OPS

The code converter will convert the most common TDL pseudo-ops. They include the following:

.ASCII	.IDENT
.BLKB	.INTERN
.BLKW	.LIST
.BYTE	.WORD
.EXTERN	.XLIST

9.3 ERROR MESSAGES

If the code converter detects an error in a statement line, it leaves the line unchanged. There are two types of error messages.

(1) **"*** Syntax error at line nnn, line follows ***"**
error line

Where nnn is the statement line number, and error line is the statement line with the syntax error. Normally, this error should not occur because it indicates that the operand for this particular op-code is syntactically incorrect.

(2) **"*** IF/ENDIF unbalanced ***"**

This error message appears if the IFs and ENDIFs are not paired. For every IF, there should be an ENDIF, and vice versa.

CHAPTER 10
LINKER

10.1 OVERVIEW

The Z80 Linker (ZLINK) is used to combine Microsoft relocatable object modules into an absolute file ready for execution under Z or CP/M. When completed, ZLINK lists the sorted symbol table, any unresolved or duplicate symbols, and a load map which shows the number of free bytes left and the size and locations of the different segments:

```
LOAD MAP FOR FILENAME.COM

SEGMENT    SIZE    START    STOP
ABSOLUTE
CODE
DATA
COMMON
FREE
```

ZLINK writes the sorted symbol table to a .SYM file suitable for use with Echelon Dynamic Screen Debugger (DSD) and Digital Research Symbolic Instruction Debuggers (SID and ZSID) as described in the S option (see next page). ZLINK also creates a COM file for direct execution under Z or CP/M. If errors are detected, the P option (see next page) will be set automatically.

10.2 ZLINK OPERATION

ZLINK is invoked by typing

```
ZLINK filename1{,filename2,...,filenameN}
```

where filename is the name of the object module(s) to be linked. If no filetype is specified, then REL is assumed. If some other filename is desired for the COM and SYM files, it may be specified as follows:

```
ZLINK newfilename=filename1 {,filename2,...filenameN}
```

If ZLINK encounters a starting address which is caused by supplying an optional program starting address to the assembler END pseudo-op then ZLINK will place a JUMP instruction at 100H to the program starting address.

10.3 ZLINK OPTIONS

A variety of options are available to provide control over the execution parameters of ZLINK. Except for the / option (library

search option) all of the options are link control options. They are used once at the end of a command line:

filename1{,filename2,...filenameN} $Cnnnn,Dnnnn,P,Rnnnn

Where nnnn is a hexadecimal number.

ZLINK options include:

C: Code Segment Origin Option. The C option is used to specify the load address of the code segment. If it is not used, then ZLINK will put the code segment at the address (100H). Unless the R option indicates otherwise, the relocation value of the code segment will be set to its load address. The syntax for the C option is Cnnnn, where nnnn if the desired code origin in hex.

D: Data Origin Option. The D option indicates the load address of the data and common segments. If the D option is used, the address specified must be higher than the load address for the code segment. If it is not used, ZLINK will put the data and common segments immediately after the program segment. The syntax for the D option is Dnnnn, where nnnn is the desired data origin in hex.

P: Paging Option. The P option will page the output of ZLINK, at 23 lines per page to the terminal. Pressing any key allows you to continue to output one page at a time.

R: Relocate Origin Option. The R option specifies the relocation value for the code segment. If not used, then ZLINK will set the relocation value of the code segment to its load address.

S: .SYM File Option. If this option is set, ZLINK will write the sorted symbol table to a .SYM file suitable for use with the Echelon DSD or Digital Research SID and ZSID debuggers.

/: Search Option. This option is used to indicate that the preceding file should be treated as a library. ZLINK will search the file and include only those modules containing symbols which are referenced but not defined in the modules already linked. Unlike the link control options which can be used once at the end of a command line, the / option must be used after each filename to be searched:

filename1/,filename2/,...filenameN/

10.4 DEFINE NEXT FREE MEMORY LOCATION

If the public symbol $MEMRY is encountered during the link process, then the two bytes addressed by the value $MEMRY and $MEMRY + 1 are filled in with the address of the next free memory location. The statement labeled $MEMRY must be a DS statement.

For example:

```
                PUBLIC    FREBEG,$MEMRY
        FREBEG: LD        HL,($MEMRY)    ;This routine returns
                RET                      ;the first free byte
        $MEMRY: DS        2              ;of memory
```

10.5 ZLINK ERROR MESSAGES

(1) **"Can't find filename.filetype"**

Specified file cannot be found on the disk.

(2) **"Filename.filetype is an invalid REL file!"**

One of the files specified is not a Microsoft compatible REL file.

(3) **"Invalid option specification!"**

One of the options specified is invalid.

(4) **"Memory full!"**

There is insufficient memory to complete the linking process.

(5) **"Undefined symbols:"**

The symbol name(s) following this heading are referenced but not defined in any of the modules being linked.

(6) **"Duplicate symbols:"**

The symbol name(s) following this heading are defined as a PUBLIC symbol in more than one of the modules being linked.

(7) **"***Overlapping segments***"**

ZLINK attempted to write a segment into memory already used by another segment. This error is probably caused by incorrect use of the C and/or D options.

(8) **"Read error!"**

A file cannot be read properly.

(9) **"Syntax error in command line!"**

The command line is ill formed.

(10) **"Multiple main modules!"**
Two or more modules contain a program starting address.

(11) **"Library search limit exceeded!"**
A maximum of ten libraries can be specified from assembler .REQUEST statements.

CHAPTER 11
LIBRARY MANAGER

11.1 OVERVIEW

The Library Manager (ZLIB) is used to combine Microsoft relocatable object modules into a library. Libraries are files consisting of any number of relocatable object modules. ZLIB can delete modules from a library, concatenate REL files into a library, re-place modules in a library, and print module names and public symbols from a library.

11.2 ZLIB OPERATION

ZLIB is invoked by typing:

```
ZLIB libname=filename{,filename,...} $option
```

where libname is the name of the library with filetype REL and filename is the name of the object module(s). If no filetype is specified, then REL is assumed.

An alternate form of invoking ZLIB when using the M or P option (as described below) is:

```
ZLIB libname $listoption
```

where listoption is the M or P option.

11.3 ZLIB OPTIONS

If no option is specified, then the specified modules will be appended to the library. The options include:

D: Delete the specified modules.

M: Print the module names in the library.

P: Print the module names and public symbols in the library.

R: Replace the specified modules.

11.4 ZLIB MESSAGES

Under the following circumstances ZLIB will produce messages.

(1) When a module is being appended to the library:

"Appending filename.filetype"

(2) If the specified library does not exist on disk and the specified option is append:

"Creating library"

(3) If a module is being deleted:

"Deleting modulename"

(4) If a module is being replaced:

"Deleting modulename
Appending filename.filetype"

11.5 ZLIB ERROR MESSAGES

(1) **"Can't find filename.filetype"**

Specified file cannot be found on the disk.

(2) **"Filename.filetype is an invalid REL file!"**

One of the files specified is not a Microsoft compatible REL file.

(3) **"Invalid option specification!"**

The option specified is invalid.

(4) **"Syntax error in command line!"**
The command line is ill formed.

APPENDIX A
Z80 MNEMONIC MACHINE INSTRUCTION CODES

Object Code	Source Statement		Operation	Notes
8E	ADC	A,(HL)	Add with Carry Oper-	Leading A Oper-
DD8E05	ADC	A,(IX+d)	and to Acc.	and is Optional
FD8E05	ADC	A,(IY+d)		
8F	ADC	A,A		If d is Omitted
88	ADC	A,B		0 is Assumed
89	ADC	A,C		
8A	ADC	A,D		
8B	ADC	A,E		
8C	ADC	A,H		
8D	ADC	A,L		
CE20	ADC	A,n		
ED4A	ADC	HL,BC	Add with Carry Reg.	
ED5A	ADC	HL,DE	Pair to HL	
ED6A	ADC	HL,HL		
ED7A	ADC	HL,SP		
86	ADD	A,(HL)	Add Operand to Acc.	Leading A Oper-
DD8605	ADD	A,(IX+d)		and is Optional
FD8605	ADD	A,(IY+d)		
87	ADD	A,A		If d is Omitted
80	ADD	A,B		0 is Assumed
81	ADD	A,C		
82	ADD	A,D		
83	ADD	A,E		
84	ADD	A,H		
85	ADD	A,L		
C620	ADD	A,n		
09	ADD	HL,BC	Add Reg. Pair to HL	
19	ADD	HL,DE		
29	ADD	HL,HL		
39	ADD	HL,SP		
DD09	ADD	IX,BC	Add Reg. Pair to IX	
DD19	ADD	IX,DE		
DD29	ADD	IX,IX		
DD39	ADD	IX,SP		
FD09	ADD	IY,BC	Add Reg. Pair to IY	
FD19	ADD	IY,DE		
FD29	ADD	IY,IY		
FD39	ADD	IY,SP		
A6	AND	A,(HL)	Logical 'AND' of	Leading A Oper-
DDA605	AND	A,(IX+d)	Operand and Acc.	and is Optional
FDA605	AND	A,(IY+d)		

Object Code	Source Statement		Operation	Notes
A7	AND	A,A	Logial 'AND' of	Leading A Oper-
A0	AND	A,B	Operand and Acc.	and is Optional
A1	AND	A,C		
A2	AND	A,D		If d is Omitted
A3	AND	A,E		0 is Assumed
A4	AND	A,H		
A5	AND	A,L		
E620	AND	A,n		
CB46	BIT	0,(HL)	Test Bit of Location	If d is Omitted
DDCB0546	BIT	0,(IX+d)	or Reg.	0 is Assumed
FDCB0546	BIT	0,(IY+d)		
CB47	BIT	0,A		
CB40	BIT	0,B		
CB41	BIT	0,C		
CB42	BIT	0,D		
CB43	BIT	0,E		
CB44	BIT	0,H		
CB45	BIT	0,L		
CB4E	BIT	1,(HL)		
DDCB054E	BIT	1,(IX+d)		
FDCB054E	BIT	1,(IY+d)		
CB4F	BIT	1,A		
CB48	BIT	1,B		
CB49	BIT	1,C		
CB4A	BIT	1,D		
CB4B	BIT	1,E		
CB4C	BIT	1,H		
CB4D	BIT	1,L		
CB56	BIT	2,(HL)		
DDCB0556	BIT	2,(IX+d)		
FDCB0556	BIT	2,(IX+d)		
CB57	BIT	2,A		
CB50	BIT	2,B		
CB51	BIT	2,C		
CB52	BIT	2,D		
CB53	BIT	2,E		
CB54	BIT	2,H		
CB55	BIT	2,L		
CB5E	BIT	3,(HL)		
DDCB055E	BIT	3,(IX+d)		
DFCB055E	BIT	3,(IY+d)		
CB5F	BIT	3,A		
CB58	BIT	3,B		
CB59	BIT	3,C		
CB5A	BIT	3,D		
CB5B	BIT	3,E		
CB5C	BIT	3,H		
CB5D	BIT	3,L		
CB66	BIT	4,(HL)		
DDCB0566	BIT	4,(IX+d)		
FDCBO566	BIT	4,(IY+d)		

Object Code	Source Statement		Operation	Notes
CB67	BIT	4,A	Test Bit of Location	If d is Omitted
CB60	BIT	4,B	or Reg.	0 is Assumed
CB61	BIT	4,C		
CB62	BIT	4,D		
CB63	BIT	4,E		
CB64	BIT	4,H		
CB65	BIT	4,L		
CB6E	BIT	5,(HL)		
DDCB056E	BIT	5,(IX+d)		
FDCB056E	BIT	5,(IY+d)		
CB6F	BIT	5,A		
CB68	BIT	5,B		
CB69	BIT	5,C		
CB6A	BIT	5,D		
CB6B	BIT	5,E		
CB6C	BIT	5,H		
CB6D	BIT	5,L		
CB76	BIT	6,(HL)		
DDCB0576	BIT	6,(IX+d)		
FDCB0576	BIT	6,(IY+d)		
CB77	BIT	6,A		
CB70	BIT	6,B		
CB71	BIT	6,C		
CB72	BIT	6,D		
CB73	BIT	6,E		
CB74	BIT	6,H		
CB75	BIT	6,L		
CB7E	BIT	7,(HL)		
DDCB057E	BIT	7,(IX+d)		
FDCB057E	BIT	7,(IY+d)		
CB7F	BIT	7,A		
CB78	BIT	7,B		
CB79	BIT	7,C		
CB7A	BIT	7,D		
CB7B	BIT	7,E		
CB7C	BIT	7,H		
CB7D	BIT	7,L		
**********	**********	**********	**********	**********
DC8405	CALL	C,nn	Call Subroutine at	
FC8405	CALL	M,nn	Location nn if Condi-	
D48405	CALL	NC,nn	tion True	
C48405	CALL	NZ,nn		
F48405	CALL	P,nn		
EC8405	CALL	PE,nn		
E48405	CALL	PO,nn		
CC8405	CALL	Z,nn		
**********	**********	**********	**********	**********
CD8405	CALL	nn	Unconditional Call to Subroutine at nn	
**********	**********	**********	**********	**********
3F	CCF		Complement Carry Flag	

Object Code	Source Statement		Operation	Notes
BE	CP	(HL)	Compar Operand	Leading A Oper-
DDBE05	CP	(IX+d)	with Acc.	and is Optional
FDBE05	CP	(IY+d)		
BF	CP	A		If d is Omitted
B8	CP	B		0 is Assumed
B9	CP	C		
BA	CP	D		
BB	CP	E		
BC	CP	H		
BD	CP	L		
FE20	CP	n		
EDA9	CPD		Compare Location (HL) and Acc. Decrement HL and BC	
EDB9	CPDR		Compare Location (HL) and Acc., Decrement HL and BC, Repeat until BC=0	
EDA1	CPI		Compare Location (HL) and Acc., Increment HL and Decrement BC	
EDB1	CPIR		Compare Location (HL) and Acc., Increment HL, Decrement BC, Repeat until BC=0	
2F	CPL		Complement Acc. (1's Complement)	
27	DAA		Decimal Adjust Acc.	
35	DEC	(HL)	Decrement Operand	If d is Omitted
DD3505	DEC	(IX+d)		0 is Assumed
FD3505	DEC	(IY+d)		
3D	DEC	A		
05	DEC	B		
0B	DEC	BC		
0D	DEC	C		
15	DEC	D		
1B	DEC	DE		
1D	DEC	E		
25	DEC	H		
2B	DEC	HL		
DD2B	DEC	IX		
FD2B	DEC	IY		
2D	DEC	L		
3B	DEC	SP		

Object Code	Source Statement		Operation	Notes
F3	DI		Disable Interrupts	
102E	DJNZ	e	Decrement B and Jump Relative if B=0	
FB	EI		Enable Interrupts	
E3	EX	(SP),HL	Exchange Location and (SP)	
DDE3	EX	(SP),IX		
FDE3	EX	(SP),IY		
08	EX	AF,AF'	Exchange the Contents of AF and AF'	
EB		DE,HL	Exchange the Contents of DE and HL	
D9	EXX		Exchange the Contents of BC,DE,HL with Contents of BC',DE',HL' Respectively	
76	HALT		HALT (wait for Interrupt or Reset)	
ED46	IM	0	Set Interrupt Mode	
ED56	IM	1		
ED5E	IM	2		
ED78	IN	A,(C)	Load Reg. with Input from Device (C)	
ED40	IN	B,(C)		
ED48	IN	C,(C)		
ED50	IN	D,(C)		
ED58	IN	E,(C)		
ED60	IN	H,(C)		
ED68	IN	L,(C)		
34	INC	(HL)	Increment Operand	If d is Omitted 0 is Assumed
DD3405	INC	(IX+d)		
FD3405	INC	(IY+d)		
3C	INC	A		
04	INC	B		
03	INC	BC		
0C	INC	C		
14	INC	D		
13	INC	DE		
1C	INC	E		
24	INC	H		
23	INC	HL		
DD23	INC	IX		
FD23	INC	IY		

Object Code	Source Statement		Operation	Notes
2C	INC	L	Increment Operand	
33	INC	SP		
DB20	IN	A,(n)	Load Acc. with Input from Device n	
EDAA	IND		Load Location (HL) with Input from Port (C), Decrement HL and B	
EDBA	INDR		Load Location (HL) with Input from Port (C), Decrement HL and Decrement B, Repeat until B=0	
EDA2	INI		Load Location (HL) with Input from Port (C); Increment HL and Decrement B	
EDB2	INIR		Load Location (HL) with Input from Port (C), Increment HL and Decrement B, Repeat until B=0	
C38405	JP	nn	Unconditional Jump to Location	
E9	JP	(HL)		
DDE9	JP	(IX)		
FDE9	JP	(IY)		
DA8405	JP	C,nn	Jump to Location if Condtion True	
FA8405	JP	M,nn		
D28405	JP	NC,nn		
C28405	JP	NZ,nn		
F28405	JP	P,nn		
EA8405	JP	PE,nn		
E28405	JP	PO,nn		
CA8405	JP	Z,nn		
382E	JR	C,e	Jump Relative to PC+e if Condition True	
302E	JR	NC,e		
202E	JR	NZ,e		
282E	JR	Z,e		
182E	JR	e	Uncondtional Jump Relative to PC+e	
02	LD	(BC),A	Load Source to Destination	
12	LD	(DE),A		

Object Code	Source Statement		Operation	Notes
77	LD	(HL),A	Load Source to	If d is Omitted
70	LD	(HL),B	Destination	0 is Assumed
71	LD	(HL),C		
72	LD	(HL),D		
73	LD	(HL),E		
74	LD	(HL),H		
75	LD	(HL),L		
3620	LD	(HL),n		
DD7705	LD	(IX+d),A		
DD7005	LD	(IX+d),B		
DD7105	LD	(IX+d),C		
DD7205	LD	(IX+d),D		
DD7305	LD	(IX+d),E		
DD7405	LD	(IX+d),H		
DD7505	LD	(IX+d),L		
DD360520	LD	(IX+d),n		
FD7705	LD	(IY+d),A		
FD7005	LD	(IY+d),B		
FD7105	LD	(IY+d),C		
FD7205	LD	(IY+d),D		
FD7305	LD	(IY+d),E		
FD7405	LD	(IY+d),H		
FD7505	LD	(IY+d),L		
FD360520	LD	(IY+d),n		
328405	LD	(nn),A		
ED438405	LD	(nn),BC		
ED538405	LD	(nn),DE		
228405	LD	(nn),HL		
DD228405	LD	(nn),IX		
FD228405	LD	(nn),IY		
ED738405	LD	(nn),SP		
0A	LD	A,(BC)		
1A	LD	A,(DE)		
7E	LD	A,(HL)		
DD7E05	LD	A,(IX+d)		
FD7E05	LD	A,(IY+d)		
3A8405	LD	A,(nn)		
7F	LD	A,A		
78	LD	A,B		
79	LD	A,C		
7A	LD	A,D		
7B	LD	A,E		
7C	LD	A,H		
ED57	LD	A,I		
7D	LD	A,L		
3E20	LD	A,n		
ED5F	LD	A,R		
46	LD	B,(HL)		
DD4605	LD	B,(IX+d)		
FD4605	LD	B,(IY+d)		
47	LD	B,A		
40	LD	B,B		

Object Code	Source Statement		Operation	Notes
41	LD	B,C	Load Source to	If d is Omitted
42	LD	B,D	Destination	0 is Assumed
43	LD	B,E		
44	LD	B,H		
45	LD	B,L		
0620	LD	B,n		
ED4B8405	LD	BC,(nn)		
018405	LD	BC,nn		
4E	LD	C,(HL)		
DD4E05	LD	C,(IX+d)		
FD4E05	LD	C,(IY+d)		
4F	LD	C,A		
48	LD	C,B		
49	LD	C,C		
4A	LD	C,D		
4B	LD	C,E		
4C	LD	C,H		
4D	LD	C,L		
0E20	LD	C,n		
56	LD	D,(HL)		
DD5605	LD	D,(IX+d)		
FD5605	LD	D,(IY+d)		
57	LD	D,A		
50	LD	D,B		
51	LD	D,C		
52	LD	D,D		
53	LD	D,E		
54	LD	D,H		
55	LD	D,L		
1620	LD	D,n		
ED5B8405	LD	DE,(nn)		
118405	LD	DE,nn		
5E	LD	E,(HL)		
DD5E05	LD	E,(IX+d)		
FD5E05	LD	E,(IY+d)		
5F	LD	E,A		
58	LD	E,B		
59	LD	E,C		
5A	LD	E,D		
5B	LD	E,E		
5C	LD	E,H		
5D	LD	E,L		
1E20	LD	E,n		
66	LD	H,(HL)		
DD6605	LD	H,(IX+d)		
FD6605	LD	H,(IY+d)		
67	LD	H,A		
60	LD	H,B		
61	LD	H,C		
62	LD	H,D		
63	LD	H,E		
64	LD	H,H		

Object Code	Source Statement		Operation	Notes
65	LD	H,L	Load Source to	If d is Omitted
2620	LD	H,n	Destination	0 is Assumed
2A8405	LD	HL,(nn)		
218405	LD	HL,nn		
ED47	LD	I,A		
DD2A8405	LD	IX,(nn)		
DD218405	LD	IX,nn		
FD2A8405	LD	IY,(nn)		
FD218405	LD	IY,nn		
6E	LD	L,(HL)		
DD6E05	LD	L,(IX+d)		
FD6E05	LD	L,(IY+d)		
6F	LD	L,A		
68	LD	L,B		
69	LD	L,C		
6A	LD	L,D		
6B	LD	L,E		
6C	LD	L,H		
6D	LD	L,L		
2E20	LD	L,n		
ED4F	LD	R,A		
ED7B8405	LD	SP,(nn)		
F9	LD	SP,HL		
DDF9	LD	SP,IX		
FDF9	LD	SP,IY		
318405	LD	SP,nn		
EDA8	LDD		Load Location(DE) with Location(HL), Decrement DE, HL and BC	
EDB8	LDDR		Load Location (DE) with Location (HL). Repeat until BC=0	
EDA0	LDI		Load Location (DE) with Location (HL), Increment DE, HL, Decrement BC	
EDB0	LDIR		Load Location (DE) with Location (HL), Increment DE, HL, Decrement BC and Repeat until BC=0	
ED44	NEG		Negate Acc. (2's Complement)	
00	NOP		No Operation	

Object Code	Source Statement		Operation	Notes
B6	OR	A,(HL)	Logical "OR" of	Leading A Operand is Optional
DDB605	OR	A,(IX+d)	Operand and Acc.	
FDB605	OR	A,(IY+d)		
B7	OR	A,A		If d is Omitted 0 is Assumed
B0	OR	A,B		
B1	OR	A,C		
B2	OR	A,D		
B3	OR	A,E		
B4	OR	A,H		
B5	OR	A,L		
F620	OR	A,n		
ED8B	OTDR		Load Output Port (C) with Location (HL), Decrement HL and B, Repeat until B=0	
EDB3	OTIR		Load Output Port (C) with Location (HL), Increment HL, Decrement B, Repeat until B=0	
ED79	OUT	(C),A	Load Output Port (C) with Reg.	
ED41	OUT	(C),B		
ED49	OUT	(C),C		
ED51	OUT	(C),D		
ED59	OUT	(C),E		
ED61	OUT	(C),H		
ED69	OUT	(C),L		
D320	OUT	(n),A	Load Output Port (n) with Acc.	
EDAB	OUTD		Load Output Port (C) with Location (HL), Decrement HL and B	
EDA3	OUTI		Load Output Port (C) with Location (HL), Increment HL and Decrement B	
F1	POP	AF	Load Destination with Top of Stack	
C1	POP	BC		
D5	POP	DE		
E1	POP	HL		
DDE1	POP	IX		
FDE1	POP	IY		

Object Code	Source Statement		Operation	Notes
F5	PUSH	AF	Load Source to Stack	
C5	PUSH	BC		
D5	PUSH	DE		
E5	PUSH	HL		
DDE5	PUSH	IX		
FDE5	PUSH	IY		
**********				
CB86	RES	0,(HL)	Reset Bit b of	If d is Omitted
DDCB0586	RES	0,(IX+d)	Operand	0 is Assumed
FDCB0586	RES	0,(IY+d)		
CB87	RES	0,A		
CB80	RES	0,B		
CB81	RES	0,C		
CB82	RES	0,D		
CB83	RES	0,E		
CB84	RES	0,H		
CB85	RES	0,L		
CB8E	RES	1,(HL)		
DDCB058E	RES	1,(IX+d)		
FDCB058E	RES	1,(IY+d)		
CB8F	RES	1,A		
CB88	RES	1,B		
CB89	RES	1,C		
CB8A	RES	1,D		
CB8B	RES	1,E		
CB8C	RES	1,H		
CB8D	RES	1,L		
CB96	RES	2,(HL)		
DDCB0596	RES	2,(IX+d)		
FDCB0596	RES	2,(IY+d)		
CB97	RES	2,A		
CB90	RES	2,B		
CB91	RES	2,C		
CB92	RES	2,D		
CB93	RES	2,E		
CB94	RES	2,H		
CB95	RES	2,L		
CB9E	RES	3,(HL)		
DDCB059E	RES	3,(IX+d)		
FDCB059E	RES	3,(IY+d		
CB9F	RES	3,A		
CB98	RES	3,B		
CB9A	RES	3,D		
CB9B	RES	3,E		
CB9C	RES	3,H		
CB9D	RES	3,L		
CBA6	RES	4,(HL)		
DDCB05A6	RES	4,(IX+d)		
FDCB05A6	RES	4,(IY+d)		
CBA7	RES	4,A		
CBA0	RES	4,B		
CBA1	RES	4,C		

Object Code	Source Statement		Operation	Notes
CBA2	RES	4,D	Reset Bit b of	If d is Omitted
CBA3	RES	4,E	Operation	0 is Assumed
CBA4	RES	4,H		
CBA5	RES	4,L		
CBAE	RES	5,(HL)		
DDCB05AE	RES	5,(IX+d)		
FDCB05AE	RES	5,(IY+d)		
CBAF	RES	5,A		
CBA8	RES	5,B		
CBA9	RES	5,C		
CBAA	RES	5,D		
CBAB	RES	5,E		
CBAC	RES	5,L		
CBB6	RES	6,(HL)		
DDCB05B6	RES	6,(IX+d)		
FDCB05B6	RES	6,(IY+d)		
CBB7	RES	6,A		
CBB0	RES	6,B		
CBB1	RES	6,C		
CBB2	RES	6,D		
CBB3	RES	6,E		
CBB4	RES	6,H		
CBB5	RES	6,L		
CBBE	RES	7,(HL)		
DDCB05BE	RES	7,(IX+d)		
FDCB05BE	RES	7,(IY+d)		
CBBF	RES	7,A		
CBB8	RES	7,B		
CBB9	RES	7,C		
CBBA	RES	7,D		
CBBB	RES	7,E		
CBBC	RES	7,H		
CBBD	RES	7,L		
C9	RET		Return from Subroutine	
D8	RET	C	Return from Subroutine if Condition True	
F8	RET	M		
D0	RET	NC		
C0	RET	NZ		
F0	RET	P		
E8	RET	PE		
E0	RET	PO		
C8	RET	Z		
ED4D	RETI		Return from Interrupt	
ED45	RETN		Return from Non-Maskable Interrupt	

Object Code	Source Statement		Operation	Notes
CB16	RL	(HL)	Rotate Left Through	If d is Omitted
DDCB0516	RL	(IX+d)	Carry	0 is Assumed
FDCB0516	RL	(IY+d)		
CB17	RL	A		
CB10	RL	B		
CB11	RL	C		
CB12	RL	D		
CB13	RL	E		
CB14	RL	H		
CB15	RL	L		
17	RLA		Rotate Left Acc. Through Carry	
CB06	RLC	(HL)	Rotate Left Circular	If d is Omitted
DDCB0506	RLC	(IX+d)		0 i Assumed
FDCB0506	RLC	(IY+d)		
CB07	RLC	A		
CB00	RLC	B		
CB01	RLC	C		
CB02	RLC	D		
CB03	RLC	E		
CB04	RLC	H		
CB05	RLC	L		
07	RLCA		Rotate Left Circ. Acc.	
ED6F	RLD		Rotate Digit Left and Right between Acc. and Location (HL)	
CB1E	RR	(HL)	Rotate Right Through	If d is Omitted
DDCB051E	RR	(IX+d)	Carry	0 is Assumed
FDCB051E	RR	(IY+d)		
CB1F	RR	A		
CB18	RR	B		
CB19	RR	C		
CB1A	RR	D		
CB1B	RR	E		
CB1C	RR	H		
CB1D	RR	L		
1F	RRA		Rotate Right Acc. Through Carry	
CB0E	RRC	(HL)	Rotate Right Circular	
DDCB050E	RRC	(IX+d)		
FDCB050E	RRC	(IY+d)		
CB0F	RRC	A		
CB08	RRC	B		
CB09	RRC	C		
CB0A	RRC	D		

Object Code	Source Statement		Operation	Notes
CB0B	RRC	E	Rotate Right Circular	
CB0C	RRC	H		
CB0D	RRC	L		
0F	RRCA		Rotate Right Circular Acc.	
ED67	RRD		Rotate Digit Right and Left Between Acc. and Location (HL)	
C7	RST	00H	Restart to Location	
CF	RST	08H		
D7	RST	10H		
DF	RST	18H		
E7	RST	20H		
EF	RST	28H		
F7	RST	30H		
FF	RST	38H		
DE20	SBC	A,n	Subtract Operand from Acc. with Carry	Leading A Operand is Optional
9E	SBC	A,(HL)		
DD9E05	SBC	A,(IX+d)		
FD9E05	SBC	A,(IY+d)		If d is Omitted 0 is Assumed
9F	SBC	A,A		
98	SBC	A,B		
99	SBC	A,C		
9A	SBC	A,D		
9B	SBC	A,E		
9C	SBC	A,H		
9D	SBC	A,L		
ED42	SBC	HL,BC		
ED52	SBC	HL,DE		
ED62	SBC	HL,HL		
ED72	SBC	HL,SP		
37	SCF		Set Carry Flag (C=1)	
CBC6	SET	0,(HL)	Set Bit b of Location	If d is Omitted 0 is Assumed
DDCB05C6	SET	0,(IX+d)		
FDCB05C6	SET	0,(IY+d)		
CBC7	SET	0,A		
CBC0	SET	0,B		
CBC1	SET	0,C		
CBC2	SET	0,D		
CBC3	SET	0,E		
CBC4	SET	0,H		
CBC5	SET	0,L		
CBCE	SET	1,(HL)		
DDCB05CE	SET	1,(IX+d)		
FDCB05CE	SET	1,(IY+d)		
CBCF	SET	1,A		

Object Code	Source Statement		Operation	Notes
CBC8	SET	1,B	Set Bit b of Location	If d is Omitted
CBC9	SET	1,C		0 is Assumed
CBCA	SET	1,D		
CBCB	SET	1,E		
CBCC	SET	1,H		
CBCD	SET	1,L		
CBD6	SET	2,(HL)		
DDCB05D6	SET	2,(IX+d)		
FDCB05D6	SET	2,(IY+d)		
CBD7	SET	2,A		
CBD0	SET	2,B		
CBD1	SET	2,C		
CBD2	SET	2,D		
CBD3	SET	2,E		
CBD4	SET	2,H		
CBD5	SET	2,L		
CBD8	SET	3,B		
CBDE	SET	3,(HL)		
DDCB05DE	SET	3,(IX+d)		
FDCB05DE	SET	3,(IY+d)		
CBDF	SET	3,A		
CBD8	SET	3,B		
CBD9	SET	3,C		
CBDA	SET	3,D		
CBDB	SET	3,E		
CBDC	SET	3,H		
CBDD	SET	3,L		
CBE6	SET	4,(HL)		
DDCB05E6	SET	4,(IX+d)		
FDCB05E6	SET	4,(IY+d)		
CBE7	SET	4,A		
CBE0	SET	4,B		
CBE1	SET	4,C		
CBE2	SET	4,D		
CBE3	SET	4,E		
CBE4	SET	4,H		
CBE5	SET	4,L		
CBEE	SET	5,(HL)		
DDCB05EE	SET	5,(IX+d)		
FDCB05EE	SET	5,(IY+d)		
CBEF	SET	5,A		
CBE8	SET	5,B		
CBE9	SET	5,C		
CBEA	SET	5,D		
CBEB	SET	5,E		
CBEC	SET	5,H		
CBED	SET	5,L		
CBF6	SET	6,(HL)		
DDCB05F6	SET	6,(IX+d)		
FDCB05F6	SET	6,(IY+d)		
CBF7	SET	6,A		
CBF0	SET	6,B		

Object Code	Source Statement		Operation	Notes
CBF1	SET	6,C	Set Bit b of	If d is Omitted
CBF2	SET	6,D	Location	0 is Assumed
CBF3	SET	6,E		
CBF4	SET	6,H		
CBF5	SET	6,L		
DBFE	SET	7,(HL)		
DDCB05FE	SET	7,(IX+d)		
FDCB05FE	SET	7,(IY+d)		
CBFF	SET	7,A		
CBF8	SET	7,B		
CBF9	SET	7,C		
CBFA	SET	7,D		
CBFB	SET	7,E		
CBFC	SET	7,H		
CBFD	SET	7,L		
**********				
CB26	SLA	(HL)	Shift Operand Left	If d is Omitted
DDCB0526	SLA	(IX+d)	Arithmetic	0 is Assumed
FDCB0526	SLA	(IY+d)		
CB27	SLA	A		
CB20	SLA	B		
CB21	SLA	C		
CB22	SLA	D		
CB23	SLA	E		
CB24	SLA	H		
CB25	SLA	L		
**********				
CB2E	SRA	(HL)	Shift Operand Right	If d is Omitted
DDCB052E	SRA	(IX+d)	Arithmetic	0 is Assumed
FDCB052E	SRA	(IY+d)		
CB2F	SRA	A		
CB28	SRA	B		
CB29	SRA	C		
CB2A	SRA	D		
CB2B	SRA	E		
CB2C	SRA	H		
CB2D	SRA	L		
**********				
CB3E	SRL	(HL)	Shift Operand Right	If d is Omitted
DDCB053E	SRL	(IX+d)	Logical	0 is Assumed
FDCB053E	SRL	(IY+d)		
DB3F	SRL	A		
DB38	SRL	B		
CB39	SRL	C		
CB3A	SRL	D		
CB3B	SRL	E		
CB3C	SRL	H		
CB3D	SRL	L		
**********				
96	SUB	(HL)	Subtract Operand	Leading A Oper-
DD9605	SUB	(IX+d)	from Acc.	and is Optional

Object Code	Source Statement		Operation	Notes
FD9605	SUB	(IY+d)	Subtract Operand	If d is Omitted
97	SUB	A	from Acc.	0 is Assumed
90	SUB	B		
91	SUB	C		
92	SUB	D		
93	SUB	E		
94	SUB	H		
95	SUB	L		
D620	SUB	n		
**********				
AE	XOR	A,(HL)	Exclusive "OR"	Leading A Oper-
DDAE05	XOR	A,(IX+d)	Operand and Acc.	and is Optional
FDAE05	XOR	A,(IY+d)		
AF	XOR	A,A		If d is Omitted
A8	XOR	A,B		0 is Assumed
A9	XOR	A,C		
AA	XOR	A,D		
AB	XOR	A,E		
AC	XOR	A,H		
AD	XOR	A,L		
EE20	XOR	A,n		

APPENDIX B

ECHELON SOFTWARE UPDATE FORM

1. **PRODUCT NAME & VERSION** _____

2. **USER NAME** _____ **DATE** _____

3. **USER'S HARDWARE & SOFTWARE SYSTEM:** _____

_____

4. **REPORT TYPE:**

___ Problem/Possible Error

___ Suggested Enhancement

___ Document Suggestion

___ Other _____

_____

5. **PERFORMANCE IMPACT:**

___ Shuts Down System

___ Impairs System Performance

___ Causes Inconvenience

___ Needs Suggested Enhancement

___ Other _____

6. **PROBLEM DESCRIPTION:** Please describe the problem concisely and how it can be reproduced. If possible, provide your diagnosis and your cure. Attach a listing if available.

7. **RETURN FORM TO:** Echelon, Inc.
101 First Street
Los Altos, CA 94022

YOUR INTEREST IN Z-TOOLS IS APPRECIATED!

APPENDIX C

ZAS PSEUDO-OP SUMMARY

	Pseudo-op	Form	Definition
	ASEG		set absolute segment
	COMMON		set common segment
	CSEG		set code segment
	DB(DEFB)	n {,n...}	define byte
	DC	'string'	define character
	.DPHASE		end .phase
	DS(DEFS)	expression {,expression}	define space
	DSEG		set data segment
	DW(DEFW)	nn {,nn...}	define word
	ELSE		conditional assembly
	END	{expression}	specifies program starting address
	ENDIF		end conditional assembly
	ENDM		end macro
LABEL	EQU	expression	equate label to a value
	EXITM		exit macro
	EXTRN(EXT)	symbol {,symbol...)	define external symbols
	.HD64		assemble HD64180 instructions
	IF	expression	conditional assembly
	.IF1		conditional trueness based on lsb
	.IF16		conditional trueness based on 16-bits
	.IN(MACLIB)	{d:}filename	include file
	IRP	identifier, string {,string...}	indefinite repeat macro
	IRPC	identifier, string	indefinite repeat character macro
	.LALL		list all macro lines
	.LFCOND		list all false conditionals
	.LIST		resume listing
	LOCAL	identifier {,identifier...}	define local macro labels
LABEL	MACRO	dummy {,dummy...}	stored macro definition
	NAME	modulename	define module name

ZAS PSEUDO-OP SUMMARY (con't)

	Pseudo-op	Form	Definition
	ORG	expression	change value of relocation counter
	PAGE	{expression}	page definition or eject
	.PHASE	expression	relocate block of code
	.PRINT	pass,text	print text during assembly
	PUBLIC	symbol {,symbol...}	define public symbols
	.RADIX	n	set radix default
	REPT	expression	repeat macro
	.REQUEST	filename {,filename...}	request library search
	.SALL		suppress macro listing
	.SBTTL	'string'	define subtitle
LABEL	SET(DEFL)	expression	set label to a value
	.SFCOND		suppress listing of false conditionals
	.TITLE	'string'	define title
	.XALL		exclude non-code macro lines
	.XLIST		suppress listings

Legend: items in ( )'s are aliases; in { }'s, optional.

APPENDIX D

HITACHI HD64180 MODE

Object Code	Source Statement		Operation
ED3805	IN0	A,(nn)	Load register with input from port (nn).
ED0005	IN0	B,(nn)	
ED0805	IN0	C,(nn)	
ED1005	IN0	D,(nn)	
ED1805	IN0	E,(nn)	
ED2005	IN0	H,(nn)	
ED2805	IN0	L,(nn)	
ED4C	MLT	BC	Unsigned multiplication of each half of the specified register pair with the 16-bit result going to the specified register pair.
ED5C	MLT	DE	
ED6C	MLT	HL	
ED7C	MLT	SP	
ED8B	OTDM		Load output port (C) with location (HL), decrement HL, B, and C.
ED9B	OTDMR		Load output port (C) with location (HL), decrement HL, B, and C. Repeat until B=0.
ED83	OTIM		Load output port (C) with location (HL), increment HL and C. Decrement B.
ED93	OTIMR		Load output port (C) with location (HL), increment HL and C. Decrement B. Repeat until B=0.
ED3905	OUT0	(nn),A	Load output port (nn) from register.
ED0105	OUT0	(nn),B	
ED0905	OUT0	(nn),C	
ED1105	OUT0	(nn),D	
ED1905	OUT0	(nn),E	
ED2105	OUT0	(nn),H	
ED2905	OUT0	(nn),L	
ED76	SLP		Enter sleep mode.
ED3C	TST	A	Non-destructive AND with accumulator and specified operand.
ED04	TST	B	
ED0C	TST	C	
ED14	TST	D	
ED1C	TST	E	
ED24	TST	H	
ED2C	TST	L	
ED6405	TST	nn	
ED34	TST	(HL)	
ED7405	TSTIO	nn	Non-destructive AND of nn and the contents of port (C).