4.0 ZASM

Z80 assembler - Zilog style

Table of Contents

4.1 INTRODUCTION

The Dynabyte Inc. Z-80 Zilog assembler, ZASM.COM, reads Zilog assembly language source files previously created with the systems text editor and produces Z-80 machine object code. The object output of the assembler is in INTEL standard hex format. The hex output can be converted to absolute machine code with the utility LOAD.COM.

4.2 Assembler Execution

The Assembler is called from disk simply by typing "ZASM" followed by the file name of the source code to be assembled. This source file MUST have the extension '.Z80' to be found by the Assembler, regardless of whether or not it consists entirely of Z80 code.

When calling ZASM, the user may specify an optional 3-letter drive-request for the file name that has NO relation to the 3-letter extension of the file name on disk. Note that if this 3-letter drive-instruction is omitted, ZASM will default to the CURRENT drive for all operations.

This drive-request instruction is of the form SXP, where:

S indicates where the SOURCE file is;
X indicates where the HEX object file is to be placed;
P indicates where the PRINT file is to be placed.

The letters (@, A through D) indicate the disk drive to place or find the file, where @ - current and otherwise a specific drive. For the two output files (print, and object), X, Y, or Z is allowed, which means:

X - Console
Y - Printer
Z - Dummy (no output)

The object file will be created on the disk with the extension, .HEX, and the print-listing will be created with the .PRN extension.

For example:

Suppose the file to be assembled resides on disk drive A under the file name SAMPLE.Z80. If it is desired not to have the .HEX and .PRN files sent to drive A (for lack of room on disk A, for example), the Assembler might be called by the command line:

```
ZASM SAMPLE.ABY (return)
```

This will assemble the source file on drive A, create an object file on drive B, and send the print-listing to the printer.

One option may be specified at assembly time if desired. It instructs the assembler to construct a cross-reference listing as part of the print (.PRN) file. This option is specified simply by typing it as part of the command line when calling ZASM. The option is designated by a single letter as follows:

X - generate a cross reference

It should be noted that the 'X' option requires additional memory space and on very lengthy programs an overflow error message may be given. Consider the following example:

```
ZASM SAMPLE X (return)
```

This will generate a X-reference as part of the file for this assembly. Notice that the options must be separated from the file name by at least one space.

4.3 SOURCE FORMAT

The Assembler recognizes four fields or different types of expressions. These are:

labels,
opcode mnemonics,
operands,
remarks.

The conventions which apply in the use of these four fields are given below.

Any two of the four fields must be separated from each other by at least one delimiter; these are: a tab, a space, a colon (after labels only) a semi-colon (before remarks only), or a CR-LF (to terminate lines). Multiple delimiters may be used to improve readability.

LABELS

May be as long as desired (if all on one line); however, only up to the first 6 characters are used by the assembler. Thus, the first six characters of a label may not be duplicated in another label.

The first character of a label must be an alphabetic character, the remaining characters may be any alphanumeric (A-Z, 0-9). The delimiter for a label is generally a colon space, colon-tab.

The label must be followed by a colon. The colon may be followed immediately by the operation or one or more blanks. Labels need not start in column one. A label can not be a register name.

Correct Labels:

T12345

A1

T123456 (last character is ignored)

Incorrect Labels:

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

4A5B (Starts with a numeric character.)

OPCODES

May be preceded by a label. A space is not required between the label and the op-code. The op-code must be followed by at least one space.

The operands must be separated by commas. The length is governed by the type of reference. A reference to a register pair is typically two characters. A label as an operand is up to six alphanumeric characters, and a numeric literal may not exceed OFFFF hexadecimal. The op-code of an unlabeled code line may start in column 1.

The ZASM Assembler recognizes all standard Z-80 mnemonics. For those who do not have familiarity with these, they are well-documented in the Z-80 CPU Technical Manual published by both Zilog and Mostek. The following mnemonics are recognized by ASMZ in place of those published by ZILOG:

```
ADC s;          ADD n;          ADD r;
ADD (HL):       ADD (1x+d);     ADD (1Y+d);
SBC s;          IN A,n;         OUT n,A.
```

which were published by Zilog as:

```
ADC A,s;        ADD A,n;        ADD A,r;
ADD A,(HL);     ADD A,(1x+d);   ADD A,(1Y+d);
SBC A,s;        IN A,(n);       OUT (n),A.
```

PSEUDO-OPCODES

Pseudo-Opcodes are a special form recognized only by the Assembler and for which no object code is generated. The conventions of ZASM for pseudo-ops are described in another section. These are ORG, EQU, DEFB, DEFW, DEFS, and END.

OPERANDS

May consist of register names, constants, label names, or expressions. Register names include all standard Z-80 registers. These are documented in the Z-80 CPU Technical Manual published by Zilog and Mostek for the reader who is not familiar with their names or purposes. Constants consist of one of the types outlined below.

Constants - allowed; hexadecimal, decimal, and ASCII constants according to the following conventions:

Hex - Numbers formed from hexadecimal digits (0-9 and A-f) and terminated by the character 'H'. A hex number beginning with a letter MUST be preceded by a '0' to distinguish it from a label or register name.
Range: -0FFFFH .... 0FFFFH.

Example: LD DC,2B7AH

Decimal - Numbers formed from decimal digits (0-9) and left unterminated.
Range: -65535 65535

Example: LD BC,11130

ASCII - Numbers represented by the ASCII character(s) itself (themselves) enclosed in single quotes.
Range: ' ' through '^' which amounts to the values 20H through 7EH, including all alphanumerics and punctuation.

Example: LD BC,'+Z'

The "$" character may be used in the operand of any opcode allowing expressions as operands. The "$" is used to represent the current location counter of the Assembler. Note that "$" points to the BEGINNING of the instruction which contains it and not to the end.

Expressions - are allowed as operands. Computations are performed on both numbers and labels. The operations of addition, subtraction, multipication, and division are allowed. The expression is evaluated from left to right. The expression 2+6 * 2 will evaluate to 16.

```
Example:  LD B,2+6*2                    load with 16
```

COMMENTS

The comment field is free-format includ ing any printable ASCII characters as long as the comment is preceded by a ';'. The remark may follow an opcode, operand, or label or may exist on a line by itself. The ';' may be in column one if it is desired to have the remark on a line by itself. Multiple blanks or tabs may be used before or within the remark to improve readability. A CR-LF terminates the remark. Remarks may appear on any line.

PSEUDO-OPS

DEFB or DB (Defined BYTE)

The DB pseudo-op is used to tell the Assembler to reserve a byte or string of bytes as data in the object code. The bytes may be specified using any of the forms of constants described above; or as a series of labels which have been previously defined or equated to a value. Note that if the value or the label or constant exceeds the range 0 to 255 (or its equivalent representation in hexidecimal, octal, or binary), the DB will generate an expression error. Also note that either of the terms DB or DEFB may be used.

DEFW or DW (Defined Word)

The DW pseudo-op is used to tell the Assembler to reserve a word or string of words in the object code. A word is defined to be 2 bytes. Thus, the DW pseudo-op might be used to specify a look-up table or absolute addressess. The words may be specified using any of the forms of constants described in the Constants section above, or a label which has been previously defined or equated to a word. Note that either of the terms DW or DEFW is recognized by ZASM. Also note that the Assembler places the low byte FIRST, treating every word of two bytes as though it were an address.

ORG (Program Origin)

The ORG pseudo-op sets the Assembler location counter and is used when it is desired to start assembly of a block of code at a particular address. This location may be set by the user to be absolute, or it may be left up to the Assembler to determine the value of the ORG. The location counter may be set to a value as often as desired in a source program; that is, multiple ORG statements may be used.

EQU (Equate)

The EQU pseudo-op is used to inform the Assembler that two named quantities are equivalent. It is also used to equate a label to a particular value. Once this label is defined, it is defined for the entire source program.

END (End Assembly Pass)

The END command is a signal to the Assembler that a logical body of code is complete. Therefore, only one END statement should appear in a module. Should the END appear in the middle of a block of code, everything following the statement will be ignored by ZASM. If an expression occurs, it will be used to indicate the execution address.

4.4 Error Messages

The following error conditons will be flagged by the Assembler and will be placed in the print listing ahead of the line number. A maximum of two errors per line will be given.

A	Argument error
D	Double definition
L	Label error
M	Missing Label
O	Op-code error
P	Phase error
R	Range error
S	Syntax error
U	Undefined
V	Value error

4.5 LOADING THE OBJECT

Once a file has been assembled, an Intel Standard HEX file is generated as described in Section VI. This file contains specific address information as to where the object code is to reside in memory. This file may be converted to a binary image by using LOAD or DDT and the SAVE command.

4.6 Object File Format

Record Mark Field: Frame 0

The ASCII code for a colon (:) is used to signal the start of a record.

Record Length Field: Frames 1 and 2

The number of data bytes in the record is represented by two ASCII hexadecimal digits in this field. The high-order digit is in frame 1. The maximum number of data bytes in a record is 255 (FF in hexadecimal). An end-of-file record contains two ASCII zeros in this field.

Load Address Field: Frames 3 to 6

The four ASCII hexadecimal digits in frames 3-6 give the address at which the data is loaded. The high-order digit is in frame 3, the low-order digit in frame 6. The first data byte is stored in the location indicated by the load address; successive bytes are stored in successive memory locations. This field in an end-of-file record contains zeros or the starting address of the program.

Record Type Field: Frames 7 and 8

The two ASCII hexadecimal digits in this field specify the record type. The high-order digit is in frame 7. All data records are type 0; end-of-file records are type 1. Other possible values for this field are reserved for future expansion.

Data Field: Frames 9 to +2* (record length) -1

A data byte is represented by two frames containing the ASCII characters 0-9 or A-F, which represent a hexadecimal value between 0 and FF (0 and 255 decimal). The high-order digit is in the first frame of each pair. If the data is 4-bit, when either the high or low-order digit represents the data and the other digit of the pair may be any ASCII hexadecimal digit. There are no data bytes in an end-of-file record.

Checksum Field: Frames 9+2* (record length) to 9+2* (record length) +1

The checksum field contains the ASCII hexadecimal representation of the twos complement of the 8-bit sum of the 8-bit bytes that result from converting each pair of ASCII hexadecimal digits to one byte of binary, from the record length field to and including the last byte of the data field. Therefore the sum of all the binary equivalent data, including the checksum is zero (0).

```
':' <len.2> <load address.4> <type.2> <data.n> <check.2>
```