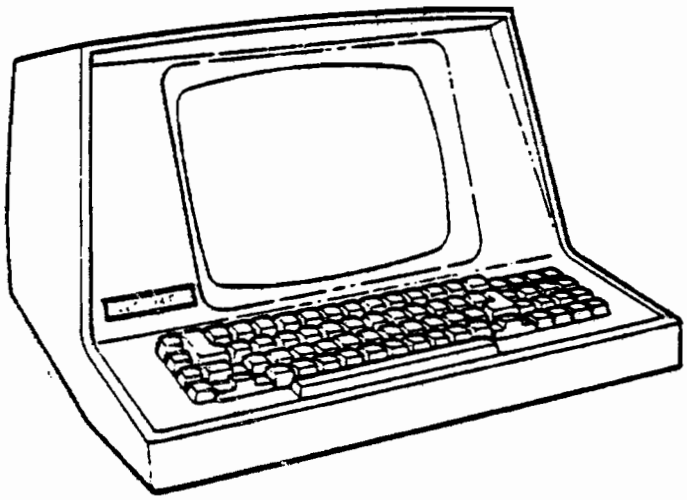
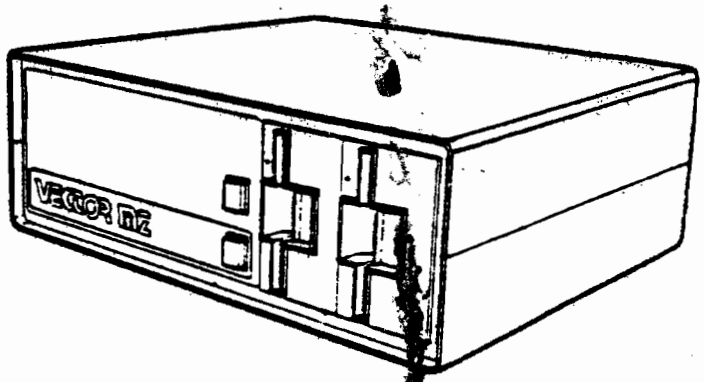
# CP/M 2
# ZSM ASSEMBLER

## USERS MANUAL






VECTOR
VECTOR GRAPHIC, INC

# ZSM ASSEMBLER FOR CP/M

Version 2.5

## USER'S MANUAL

Revision A

February 10, 1980



*CP/M is a registerd trademark of Digital Research.

## Disclaimer

Vector Graphic makes no representations or warranties with respect to the contents of this manual itself, whether or not the product it describes is covered by a warranty or repair agreement. Further, Vector Graphic reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Vector Graphic to notify any person of such revision or changes, except when an agreement to the contrary exists.

## Revisions

The date and revision of each page herein appears at the bottom of each page. The revision letter such as A or B changes if the MANUAL has been improved but the PRODUCT itself has not been significantly modified. The date and revision on the Title Page corresponds to that of the page most recently revised. When the product itself is modified significantly, the product will get a new revision number, as shown on the manual's title page, and the manual will revert to revision A, as if it were treating a brand new product. EACH MANUAL SHOULD ONLY BE USED WITH THE PRODUCT IDENTIFIED ON THE TITLE PAGE.

# FOREWORD

**Audience** — This manual is intended for ASSEMBLY LANGUAGE programmers. It assumes a moderate technical knowledge of small computers, and familiarity with the basic operation of the Vector Graphic system, the CP/M operating system, and the Z-80 instruction set.

**Scope** — This manual will describe the operation of the ZSM Assembler for CP/M, including all pseudo operations and syntax.

**Organization** — It assumes the user knows how to program in assembly language using the 8080 superset form of Z-80 mnemonics. Vector Graphic can supply a description of these mnemonics in relation to the Zilog/Mostek mnemonics.

# TABLE OF CONTENTS

Section Page



ASSEMBLER ERRORS

DAD IY,BC is ASS. as DAD IY,IY

# I PERSPECTIVE

## 1.1 Introduction

ZSM for CP/M is a program which converts Z-80 assembly language text (source code) into a sequence of machine language instructions (object code). The latter can then be loaded into the computer's memory and executed.

ZSM is a "disk to disk" assembler, meaing that it takes a named source file from a disk and puts the resultant object file back onto the disk. If requested it can print or display on the console an "assembly listing" showing both source and object code, or it can put the assembly listing on disk for printing at a later time.

For users familiar with the MDOS version of ZSM, the CP/M version is similar in most respects. The main difference is that you call it using a syntax nearly the same as the CP/M 1.4 assembler, ASM.

Like the MDOS ZSM, the CP/M version uses the 8080 superset style of mnemonics, as a convenience to programmers who began by learning 8080 code. Also, programs written for the 8080 can be assembled using ZSM. Unlike the MDOS ZSM, the CP/M ZSM does not have either the END or TAB pseudo operations, and it has several new pseudo operations, such as TITLE, RADIX, and MARGIN.

If you are not familiar with the 8080 superset style of mnemonics, Vector Graphic can supply a booklet which compares it instruction-by-instruction with the Zilog/Mostek mnemonics. It is important to grasp how indexing is handled. Under Zilog mnemonics, an operand might appear as (IX+d) where d is the offset and IX is the index register. Under ZSM, it would be d(X). Thus instead of

```
LD    H,(IX+12)
```

the following notation is used:

```
MOV   H,12(X)
```

The same is true of IY, only it would appear as (Y) instead of (X). In addition, an offset of zero may be omitted entirely. That is, (IX+0) need not be written as 0(X), it can simply be (X).

After ZSM has assembled a source file, it puts the object file onto the desired disk, giving it the file type .HEX. If you are familiar with CP/M, you know that you first have to change the type of this file to .COM if you want to be able to call it up as if it were a CCP command, just by typing its name. To do this, you type LOAD XXXXX.HEX (return) after the CCP prompt A>, where XXXXX is the name of the file. You can then type XXXXX (return) to execute the program.

## 1.2 Differences between Version 2.2 and 2.5

This section is included for users already familiar with ZSM for CP/M.

ZSM 2.5 clears up the problems in version 2.2. Problems corrected are as follows:

### Print-file output

1. Pass 1 errors now print only once, not twice.
2. Tab characters in the source file are converted to spaces properly.
3. Errors that occur on a line that is also on the border of a page are now handled correctly.
4. Pass 1 errors are now at the beginning of the print file.
5. The output format of pass 1 errors is now neatly aligned.

### Error handling

1. Some errors previously reported in pass 1 are now reported in pass 2.
2. There is improved detection of type "M" errors on lines that require a label but have none.
3. Type "V" errors are now detected for 16 bit values. Previously, only 8 bit values were checked for overflow. This modification affects "DW", "DD", and "DS" pseudo-ops.
4. No error now occurs when there is not a space or tab between an operand and a following comment. That is, ";" is now a valid end-of-operand character.
5. Type "M" errors no longer destroy the value of the last preceeding entry in the Symbol Table.
6. Overlapping hex code, due to more than one ORG statement, is now detected for up to 10 ORG's. More ORG's are unaffected, but also unchecked. A message is produced if more than 10 ORG's occurred, but it is not counted as an error. The message serves only to remind the user that the possibility of overlapping code exists.
7. Errors concerning the MOV and MVI opcodes (i.e. MOV M,2(X) or MOV M,M) are now detected and reported.

The following features are new:

1. The LINK pseudo-op now allows nesting up to seven levels instead of only one.
2. Pass numbers are now reported on the screen to mark the assembler's progress.
3. Durng LINKing, the name of the file currently being LINKed is reported, in order to mark proress through the first pass.
4. The total number of errors are now counted (in decimal) and displayed at the end of the assembly.
5. There is a new pseudo-op: MARGIN. It provides a left-hand margin on PRN files. It is documented in the body of the manual.

# II USING ZSM FOR CP/M

## 2.1 Calling ZSM from CCP (the CP/M executive)

Make sure you have a CP/M 2 System Diskette in drive A, from which you have done a warm or cold boot. Make sure your source file is on a diskette in one of the drives. You invoke ZSM by typing, after the CCP prompt A>, one of the following two forms:

ZSM <filename> (return)

OR

ZSM <filename>.<3 parameters> (return)

In both cases, <filename> represents the name of an assembly language source file of the form <filename>.ASM. In other words, to assemble USERCUST.ASM, simply reference the filename USERCUST. ZSM assumes you mean USERCUST.ASM.

In the first case, the assembler looks for the source file on the drive currently "logged in" under CP/M (usually drive A) and puts the object file back onto the same drive. It also puts the assembler listing file onto that drive, rather than printing it or displaying it on the console.

Whenever ZSM puts an object file onto a disk, the object file is always of the form <filename>.HEX. The filename is the same name used when calling ZSM. Whenever ZSM puts an assembler listing file onto disk, this file is always of the form <filename>.PRN, again where the filename is the same name used when calling ZSM. For example, if you assembler USERCUST.ASM, by typing ZSM USERCUST (return), the result will be two new files on the same diskette: USERCUST.HEX and USERCUST.PRN. If errors occur during assembly, they will be listed in the PRN file as well as at the console.

The second command form is used to specify the origin of the source file, the destination of the hex file, and the destination of the print file, if any of these are different than the currently logged in drive. Each of the 3 parameters is a single letter, which have the following meaning:

The first - designates the disk drive which contains the source file.
Use A, B, C, or D.

The second - designates the drive which will receive the the hex file.
Use A, B, C, D, or Z.
Z skips generation of the hex file altogether.

The third - designates the drive which will receive the print file.
Use A, B, C, D, X, Y or Z.
X prints the listing immediately, rather than putting it on a disk.
Y places the listing on the console, rather than a disk.
Z skips generation of the print file.

Thus, the command ZSM USERCUST.ABC (return) indicates that the source file USERCUST.ASM is to be taken from disk A, that USERCUST.HEX is to be put on drive B, and that USERCUST.PRN is to be put on drive C.

The command ZSM USERCUST.ABX (return) is the same except that it prints the listing immediately rather than putting USERCUST.PRN on the disk.

The command ZSM USERCUST.BZY (return) takes USERCUST.ASM from drive B, does not generate a hex file at all, and displays the assembly listing on the console. This kind of command might be used for a quick assembly to check for program syntax errors.

The command ZSM USERCUST.AAA (return) is exactly the same as typing ZSM USERCUST (return).

## 2.2 Language elements

The source file has a general format as follows:

LABEL: OPCODE OPERANDS;comment

Note that line numbers are not required.

Each element, except for the comment, must be separated from the preceding one by at least one space character or a tab character. Tab characters cause the elements to print on columns which are even multiples of 8 from the left edge.

The LABEL is optional. If present, it will be entered into the symbol table. Whether or not it is present, its position must be followed by a space or colon. That is,

LABEL OPC or LABEL: OPC or  OPC

are valid, while OPC

is not, because it is not preceded a space. Labels may include any of the following characters:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
Ø 1 2 3 4 5 6 7 8 9 @ . [ ] { } \ | ` ~ ^ _
```

To avoid ambiguity, however, the first character may not be . or Ø-9. In addition, a label may be of any length up to 47 characters. All characters are significant. In normal use, though, up to 12 characters should suffice; and over 14 characters will look a little strange on the listing.

The OPCODE must either be a Z-8Ø opcode or a pseudo-op. Both are explained later.

The OPERANDS vary. There can be any number of them, depending on whether they are operands for an opcode or a pseudo-op. There are also instances where there are no operands, and therefore this field can, in some cases, be omitted. If more operands are supplied that are needed, the extras are ignored.

The COMMENT field is totally ignored by the assembler, except for printing it on the listing. Comments are used only for documentation or clarity, and can be omitted altogether. If present, comments should be preceeded by a semicolon (;). The semicolon will cause a TAB to the third TAB setting, whereas its absence will result in the comment appearing immediately to the right of the operand field.

There is one exception to the above format, and that is the case of an all-comment line. If the first character of the line (after the line number and space) is either an asterisk (*) or semicolon, the entire line will be treated as a comment.

## 2.3 Constants

ZSM provides for constants of two varieties, numeric and ASCII.

ASCII constants are indicated by enclosing the appropriate character in single quotes ('). Any ASCII character can appear between the quotes, except for (1) control characters, having an ASCII code of under 20 hex; (2) the single quote character, ASCII code 27 hex; and (3) the DEL character, 7F hex.

Numeric constants may be in any of four bases - 2, 8, 10, and 16. A specific base is indicated as follows:

###H indicates hexadecimal (base 16) - for example 1C7H
###Q indicates octal (base 8) - for example 62Q
###B indicates binary (base 2) - for example 10101B
###D or just ### indicates decimal (base 10) - for example 193D or 193

Regardless of base, all numeric constants must begin with a digit, 0-9. (This is to prevent ambiguity with labels.) Thus A07 hex would have to be written as 0A07H.

There is one special numeric constant, denoted by the symbol $. This constant is always equal to the address of the current line; that is, the memory location that the current line will be written into when it is loaded. Note that this reflects the address of the beginning of the current line, not the next line (as in some assemblers). As an example, consider that

```
XXX:    JUMP    $
```

would cause an infinite loop, since it would jump to itself.

## 2.4 Operators

ZSM recognizes 10 operators. They are as follows:

| | |
|---|---|
| + | addition |
| - | subtraction, or negative (as in -1) |
| * | multiplication |
| / | division |
| % | modulo (remainder of division) |
| & | logical AND |
| ! | logical OR |
| # | logical EXCLUSIVE-OR |
| > | rotate right (11010 1B>3 yields 10111 0B) |
| < | rotate left (11101 10B<1 yields 11011 01B) |

All arithmetic operators treat their operands as unsigned 16-bit quantities, and answers are truncated to 16 bits. All logical operators perform their function on a bit-by-bit basis, and they also treat their operands as 16-bit values.

Operators combine with constants to form expressions. In an expression, all operators are evaluated in a strict left-to-right order, with no precedence of operators.

Thus consider the following situation:

TEST has been assigned the value 1000H.
INC has been assigned the value 6.

The expression encountered is TEST*6+INC!7<8.

The procedure would be TEST*6 (6000H) +INC (6006H) !7 (6007H) <8 (0760H). Thus the resulting value is 760H.

## 2.5 Registers

The Z-8Ø has a number of registers, all of which have a specific symbolic reference. ZSM supports these references, as follows.

| register | | designation | |
|---|---|---|---|
| register B | - | B | Also may be called BC for register-pair instructions |
| register C | - | C | |
| register D | - | D | Also may be called DE for register-pair instructions |
| register E | - | E | |
| register H | - | H | Also may be called HL for register-pair instructions |
| register L | - | L | |
| accumulator | - | A | |
| memory | - | M | Although not supported by ZSM, also called (HL) |
| A & flags | - | PSW | Program Status Word, may also be called AF |
| Stack Ptr | - | SP | |
| Index reg X | - | IX | Also may be called X for brevity |
| Index reg Y | - | IY | Also may be called Y for brevity |

Of course, the Z-8Ø also has registers A', B', C', D', E', H', L', F', PC, I, and R, but these are never explicitly referred to in an instruction, so no special designation is needed.

## 2.6 Pseudo-ops

ZSM supports a large number of pseudo-ops. They will be explained now.

### ORG Set origin

The ORG pseudo-op specifies where the object code is to be put. Assembled code and data is assembled starting at the address specified as the operand to the ORG psuedo-op, and proceeds upward, until the end of the program or another ORG. A program can contain as many ORGs as desired. Since ORG is handled in pass 1, any symbol appearing in the operand must already be defined.

### LINK Link to a file

The LINK pseudo-op allows separate program files on the disk to be 'linked together' and assembled as one file. The LINK operand is a source file name, enclosed in single quotes.

Linking to a file is like a subroutine; that is, when the linked-to file is exhausted, assembly of the original program will continue from where it was left off at. For example,

```
XXX:    LXI     H,4000H
XXX:    LINK    'TEST'
XXX:    MOV     A,M
```

will cause the entirety of the file TEST to be assembled between the LXI and the MOV. One note, though: files that are linked to must not contain an END pseudo-op. Up to seven levels of LINK files may be nested.

### EQU Equate

The EQU pseudo-op simply equates the label associated with it to the value of the operands.

```
TEN:    EQU     10
TWENTY: EQU     2*10
```

The above code would cause the label TEN to have the value 10, and TWENTY to have the value 20.

REQ Request value

The REQ pseudo-op is similar to the EQU pseudo-op, only instead of an explicit value being specified, the system console is prompted for the value. The prompt is specified as the operand. For example,

```
TEST:   REQ 'Input:'
```

Would cause the message

Input:

to be displayed on the console during pass 1 of the assembly. The operator must then type the value to be associated with the label. For example, if the operator had typed '56H' in response to the prompt, then TEST would have a value of 56 hex.

PRT Print

The PRT pseudo-op allows information to be displayed on the console during pass 2. If operands are present, they are displayed, otherwise, just a carriage return/linefeed is printed. For example,

```
TEST:   EQU     7000H
        PRT     'This is a test ',TEST
```

would cause

This is a test 7000

to be printed on the console during pass 2.

NLIST No list

The NLIST pseudo-op will cause code following it not to be listed. Note that this overrides any options which may have been specified in the command string; If the E option was used, nothing will be listed (errors or not) after a NLIST.

LIST List

The LIST pseudo-op cancels the effect of the NLIST pseudo-op. If there has been no NLIST, then this has no effect.

FORM Form feed

The FORM pseudo-op produces a formfeed in the listing when encountered.

IFF If false - conditional assembly

The block of code following the IFF pseudo-op will be assembled only if the operand evaluates to Ø.

IFT If true - conditional assembly

The block of code following the IFT pseudo-op will be assembled only if the operand evaluates to anything other than Ø.

ENDIF End of IF block

The ENDIF pseudo-op is used to mark the end of an IFT or IFF block.

DB Define byte

The DB pseudo-op assigns its operands to successive memory locations. Either numeric or ASCII operands may be present, but either one must evaluate to only 8 bits. This means that only one ASCII character may be included per operand. For example,

```
LOCATE: DB      1,20H,11B,'D',TEST,14
```

would put each operand into a successive memory location.

'Z' is a special case of the DB pseudo-op, and it is equivalent to DB Ø. For example,

```
XXX:    Z           and
XXX:    DB Ø
```
are equivalent.

DW Define word

The DW pseudo-op is basically similar to DB, only it defines two bytes at a time, rather than 1. Also, the two bytes are in Intel standard low/high format.

DD Define data

The DD pseudo-op is exactly like DW, only the two bytes are put in high/low format.

DT Define text

The DT pseudo-op allows ASCII text to be put into memory. The desired text must be enclosed by single quotes. For example,

```
TEST:   DT 'ABCDEF'
```

would produce the following object code: 41 42 43 44 45 46 (hex).

DTH Define text terminated high

The DTH pseudo-op is like DT, only the last character is ORed with 80H before it is written out. In the above example, the last byte would be C6 hex.

DTZ Define text terminated with zero

The DTZ pseudo-op is like DT also, only it causes a byte of 00 to be appended to the text string. Thus the example would be 41 42 43 44 45 46 00.

DS Define storage

The DS pseudo-op causes the assembler to skip over the number of bytes specified by the operand. Since the object file is scatter loaded, the area skipped over will remain undisturbed.

FILL Fill storage

The FILL pseudo-op is similar to DS, only it fills the area with a constant, rather that skipping over it. The constant to fill with is specified with the second operand. For example,

```
XXX:    FILL 5,3
```

would produce the output

03 03 03 03 03.

**TITLE** Prints a title at the top of every page in the assembly listing

Type TITLE <title>, with no quotation marks around <title>. For example,

TITLE This is a program

will cause "This is a program" to print at the top of every page in the assembly listing.

**RADIX** This sets the number system base ("radix")

This pseudo op sets the radix for all numbers used in the source program which do not have a specific number base designation such as H or D attached. It can appear anywhere in the text, and will apply to all numbers in the text because it is read by the assembler in the first pass. Only use it once within the text. If you do not use the RADIX pseudo op, ZSM defaults to a radix of 10. For example, to specify that all numbers without another number base designation are hexadecimal, set the radix to 16. To use binary, set the radix to 2.

The format is RADIX n, where n is the desired radix.

**MARGIN** This creates a left-hand margin in printed output (PRN files).

The MARGIN pseudo-op requires a parameter in absolute or symbolic form that evaluates to a value from 0 to 131. This number designates the column where printing should start.

## 2.7 Assembly errors

There are ten assembly errors. Note that an error doesn't necessarily cause the program to assemble wrong, particularly if the error is a syntax error in something like a TAB statement. Nevertheless, all errors should be avoided.

The errors are as follows.

A Argument error - This is caused by an invalid character in an operand field, or an ASCII constant which is out of range.

D Duplicate label error - This indicates that a symbolic name was used more than once as a label. The first value will be used.

J Jump error - This indicates a relative jump (JR, JRZ, JRNZ, JRC, JRNC, DJNZ) to a label which is out of range. The relative jump should be replaced with an absolute one. There is one small ambiguity with this error: If you have a relative jump to a non-existant label, you will get a J error instead of a U error. Although perhaps the U would be more appropriate, the way the errors are handled gives J priority.

L Label error - This is caused by a label which contains invalid characters.

M Missing label error - This indicates that an EQU or REQ pseudo-op was encountered, but there was no label on the line. Obviously, a label is necessary for either of these.

O Opcode error - This is caused by an illegal or missing opcode. Actually, misspelling is the most common cause of this error, or starting the opcode in the first column instead of the second.

R Register error - This indicates that an illegal value was found where a register was expected.

S Syntax error - This is caused by missing operands or improper use of operators.

U Undefined symbol error - This indicates that a symbol was used, but that the symbol has not been defined.

V Value error - This indicates that the value computed is out of range for the operation being used, specifically a two-byte instruction, or a DB.