NEVADA ASSEMBLER

Users' Reference Manual

Ellis Computing
600 - 41st Avenue
San Francisco, CA 94121
(415) 751-1522

NEVADA ASSEMBLER

AN ASSEMBLER FOR CP/M

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

The assembler translates a symbolic 8080 assembly language program "source code" into the binary instructions "object code" required by the computer to execute the program.

The assembler operates on standard CP/M text files. Each line of a normal text file consists of the characters of that line followed by a carriage return (0DH) and a line feed (0AH).

When the assembler is invoked, it is loaded into memory starting at location 100H. It processes the source code file in two passes. On the first pass, it builds a symbol table containing all of the labels defined in the source program. The symbol table begins at the memory location immediately following the assembler; each entry in the table is 7 bytes long. Certain errors may be detected during the first pass, causing error messages to be output to an error file (usually the console). On the second pass, the object code is generated and usually output to an object code file. In addition, a formatted listing of both source and object code may be output to a listing file, and the symbol table may be output to a file. Any errors detected during this pass cause messages to be output to the error file.

To abort the assembly process at any time, press the Control and C keys.

After the assembly runs to completion and no errors are detected, the resulting object code file (type .OBJ) can be executed by typing RUNA and its name.

EXAMPLE:

```
RUNA PROG    loads and executes a file called PROG.
```

SECTION 2

OPERATING PROCEDURES

HARDWARE REQUIREMENTS

1. 8080/Z80/8085 microprocessor.
   Z80 is a trademark of Zilog.
2. A minimum of 32K RAM.
3. Any disk drive.
4. CRT or Video display and keyboard.

SOFTWARE REQUIREMENTS

Digital Research Corp's CP/M operating system
Version 1.4 or 2.2.
CP/M is a trademark of Digital Research Corp.

FILES ON THE DISTRIBUTION DISKETTE

ASSM.COM	is the assembler program.
RUNA.COM	is the runtime loader.

FILE TYPE CONVENTIONS

Assembly source code files	.ASM
COBOL source code files	.CBL
FORTRAN source code files	.FOR
Object code run time files	.OBJ
Printer listing files	.PRN
Symbol table listing files	.SYM
Error files	.ERR
Work files	.WRK

GETTING STARTED

First, prepare a CP/M system's diskette for use as your NEVADA ASSEMBLER operations diskette.

Then insert the newly created CP/M diskette in disk drive A, and insert the NEVADA ASSEMBLER diskette in drive B and type Control C to initialize CP/M. Now copy all the files from the NEVADA ASSEMBLER diskette onto the CP/M diskette.

```
PIP A:=B:*.*[VO]
```

At this point, put the NEVADA ASSEMBLER diskette in a safe place. You will not need it unless something happens to your operations diskette. By the way, back up your operations diskette with a copy each week! If your system malfunctions you can then pat yourself on the back for

having a safe back up copy of your work.

Now, boot up the newly created NEVADA ASSEMBLER operations diskette. Notice that CP/M displays the amount of memory for which this version of CP/M has been specialized. The amount of memory available determines the size of the programs that can be assembled. The more memory available the larger the program that can be assembled.

EXECUTING THE ASSEMBLER

The assembler is invoked by a CP/M command with the following formats.

FORMAT-1:

ASSM file<CR>

FORMAT-2:

ASSM file[.uuu u$#LP0]<CR>

DESCRIPTION:

where:

file = [unit:]source-file-name
The name of the source code input file. This parameter must be present; all others are optional.

[ ] = optional parameters

unit: = disk drive unit letter. If this parameter is not included, the default drive is used.

u = the disk drive unit letter or the letter "X" for output to the console, or the letter "Z" for no output.

u for position one.
This single character code, if present, represents the drive onto which the listing file is to be written. If this argument is absent then the listing will be written on the default drive. Also, if the character is an X the listing will be sent to the console. If the character is a Z then no listing is produced.

u for position two.
The second letter of the file type represents the drive for the object (.OBJ) file. If this argument is absent then the file will be written on the default drive. If this character is a Z then no object code file will be produced.

u for position three.
The third letter of the file type represents the drive for the error (.ERR) file. If this argument is absent the console will be used to display the errors. This argument must be followed by a space or carriage return.

u for position one of the second set.
The first letter of the second set of arguments represents the drive for the symbol (.SYM) file. If this argument is absent no symbol table file will be produced.

<$options> Various assembler options may be controlled by following the $ with one or more of the following option specifiers. The list of options is terminated by a carriage return. For those options that may be preceded by + or -, the + is optional and will be assumed if absent.

+L The source file has line numbers in column 1-4 of each line.

-L The source file has no line numbers.

If neither of these is specified, the assembler will examine the first line to determine if the file has line numbers.

Instructs the assembler to generate its own line numbers in the listing in place of those in the source file (if any).

P Instructs the assembler to paginate output to the listing file. The file name of the source code file will be printed on the top left-hand corner of each page. A page number will be printed on the top right-hand corner of each page. If a TITL pseudo-operation occurs in the source code, a one- ot two-line title will be centered at the top of each page.

0,1,2, or 3 Specifies the spacing on the listing:

0 = no additional spacing
1 = 72 column output
2 = 80 column output (default)
3 = 132 column output

EXAMPLES:

```
ASSM TST
ASSM TST.AAX A$-L#P0
```

STARTUP

To assemble your program, type ASSM and the source file name. The first thing that happens is the copyright message is displayed on the screen and the disk drive(s) begin working. When the assembly process is complete, a message will be displayed and machine control will return to the operating system.

```
A>ASSM source-file<CR>
NEVADA ASSEMBLER (C) COPYRIGHT 1982
ELLIS COMPUTING
REV 2.0 ASSEMBLING

NO ASSEMBLY ERRORS.              4 LABELS WERE DEFINED.
A>
```

EXECUTING THE .OBJ FILE

To execute the program, type RUNA and the file-name. The assembly process creates a file with the extension type of (.OBJ). This object program file will be loaded into memory and executed.

```
A>RUNA file-name
```

There are several options that also can be specified with the RUNA command.

```
RUNA file-name[.ZLC]
```

Z = zero memory before loading the .OBJ file.

L = Load the program but don't execute it. Control returns to CP/M.

C = create a .COM file for later execution. Control returns to CP/M. Remember .COM files alway begin execution at location 100H.

Example:

```
A>RUNA PROG.ZC      this will zero memory and create a file
                    named PROG.COM.
A>RUNA PROG.L       this will load PROG but not execute it.
```

NOTE: These object code files (.OBJ), if properly orged, can also be loaded and executed by the NEVADA COBOL and NEVADA FORTRAN run time packages.

MEMORY USAGE

The ASSEMBLER program is read into memory starting at location 100H and uses all memory available up to the bottom of CP/M.

The runtime package RUNA loads into memory at location 100H and relocates itself to just below CP/M and then begins loading your program.

TERMINATION

The normal termination of the assembly is signaled by the display of the following messages and return to CP/M.

```
NO ASSEMBLY ERRORS.              4 LABELS WERE DEFINED.
A>
```

The assembly process can be interupted at any time by pressing the Control and C keys.

SECTION 3

STATEMENTS

INTRODUCTION

An assembly language program (source code) is a series of statements specifying the sequence of machine operations to be performed by the program.

Each statement resides on a single line and may contain up to four fields as well as an optional line number. These fields, label, operation, operand, and comment, are scanned from left to right by the assembler, and are separated by spaces. The assembler can handle lines up to 80 characters in length.

LINE NUMBERS

Line numbers in the range 0000-9999 may appear in columns 1-4. Line numbers need not be ordered and have no meaning to the assembler, except that they appear in listings. Line numbers may also make it easier to locate lines in the source code file when it is being edited. The disk and memory space required for normal text files will be increased by five bytes per line if line numbers are used; this may become significant for large files.

If line numbers are not used, the label field starts in column 1 and the operation field may not start before column 2. If line numbers are used, they must be followed by at least one space, so the label field starts in column 6 and the operand many not start before column 7.

Once the starting column for the label has been established, the same format must be followed throughout the file: either all of the lines or none of the lines can have line numbers. Any other file(s) assembled along with the main file (using COPY pseudo-operation) must conform to the format of the main file.

Example of source statements with line numbers:

```
Column
1234567
0001 LABEL ORA A    label field must start at column 6.
0002  JNZ NEXT     operation field starts at column 7
0003 ;              (minimum).
0004 LOOP MOV A,B   operation field starts one space after
0005 *              label.
```

Example of source statements without line numbers:

```
Column
1234567
LABEL ORA A    label field must start at column 1.
 JNZ NEXT      operation field starts at column 2 (minimum).
LOOP MOV A,B   operation field starts one space after label.
```

LABEL FIELD

The label field must start in column 1 of the line (column 6 if line numbers are used). A label gives the line a symbolic name that can be referenced by any statement in the program. Labels must start with an alphabetic character (A-Z,a-z), and may consist of any number of characters, though the assembler will ignore all characters beyond the fifth; e.g. the labels BRIDGE, BRIDG and BRIDGET cannot be distinguished by the assembler. A duplicate label error will occur if any two labels in a program begin with the same five letters.

A label may be separated from the operations field by a colon (:) insteat of, or in addition to, a blank.

The labels A, B, C, D, E, H, L, M, PSW, and SP are pre-defined by the assembler to serve as symbolic names for the 8080 registers. They must not appear in the label field.

An asterisk (*) or semi-colon (;) in place of a label in column 1 (column 6 if line numbers are used) will designate the entire line as a comment line.

OPERATION FIELD

The operation field contains either 8080 instruction mnemonics or assembler pseudo-operation mnemonics. Appendix 1 summarizes the standard instruction mnemonics recognized by the assembler, and Appendix 4 lists several references to consult if more information on the 8080 machine instructions is needed. Assembler pseudo-operations are directives that control various aspects of the assembly process, such as storage allocation, conditional assembly, file inclusion, and listing control.

An operation mnemonic may not start before column 2 (column 7 if line numbers are used) and must be separated from a label by at least one space (or a colon).

OPERAND FIELD

Most machine instructions and pseudo-operations require one or two operands, either register names, labels, constants, or arithmetic expressions involving labels and constants.

The operands must be separated from the operator by at least one space. If two operands are required, they must be separated by a comma. No spaces may occur within the operand field, since the first space following the operands delimits the comments field.

Register Names

Many 8080 machine instructions require one or two registers or a register pair to be designated in the operand field. The symbolic names for the general-purpose registers are A, B, C, D, E, H and L. SP stands for the stack pointer, while M refers to the memory location whose address is in the HL register pair. The register pairs BC, DE, and HL are designated by the symbolic names B, D, and H, respectively. The A register and condition flags, when operated upon as a register pair, are given the symbolic name PSW.

The values assigned to be register names A, B, C, D, E, H, L, M, PSW and SP are 7, 0, 1, 2, 3, 4, 5, 6, 6, and 6, respectively. These constants, or any label or expression whose value lies in the range 0 to 7, may be used in place of the pre-defined symbolic register names where a register name is required; such a substitution of a value for the pre-defined register name is not recommended, however.

Labels

Any label that is defined elsewhere in the program may be used as an operand. If a label is used where an 8-bit quantity is required (e.g., MVI C,LABEL), its value must lie in the range -256 to 255, or it will be flagged as a value error.

If a label is used as a register name, its value must lie in th range 0 to 7, or be 0, 2, 4, or 6 if it designates a register pair. Otherwise, it will be flagged as a register error.

During each pass, the assembler maintains an instruction location counter that keeps track of the next location at which an instruction may be stored; this is analogous to the program counter used by the processor during program execution to keep track of the location of the next instruction to be fetched.

The special label $ (dollar Sign) stands for the current value of the assembler's instruction location counter. When $ appears within the operand field of a machine instruction, its value is the address of the first byte of the next instruction.

EXAMPLE:

```
FIRST EQU $       The label FIRST is set to the address
TABLE DB ENTRY    of the entry in a table and LAST
*                 points to the location immediately after
*                 the end of the table.  TABLN is then
*                 the length of the table and will remain
LAST EQU $        correct, even if later additions or
TABLN EQU LAST-FIRST  deletions are made in the table.
```

CONSTANTS

Decimal, hexadecimal, octal, binary and ASCII constants may be used as operands.

The base for numeric constants is indicated by a single letter immediately following the number, as follows:

D = decimal
H = hexadecimal
O = octal
Q = octal
B = binary

If the letter is omitted, the number is assumed to be decimal. Q is usually perferred for octal constants, since O is so easily confused with 0 (zero). Numeric constants must begin with a numeric character (0-9) so that they can be distinguished from labels; a hexadecimal constant beginning with A-F must be preceded by a zero.

ASCII constants are one or two characters surrounded by single quotes ('). A single quote within an ASCII constant is represented by two single quotes in a row with no intervening spaces. For example, the expression '''', where the two outer quote marks represent the string itself, i.e., the single quote character. A single character ASCII constant has the numerical value of the corresponding ASCII code. A double character ASCII constant has the 16-bit value whose high-order byte is the ASCII code of the first character and whose low-order byte is the ASCII code of the second character.

If a constant is used where an 8-bit quantity is required (e.g., MVI C,10H), its numeric value must lie in the range

-256 to 255 or it will be flagged as a value error.

If a constant is used as a register name, its numeric value must lie in the range 0 to 7, or be 0, 2, 4, or 6 if it designates a register pair. Otherwise it will be flagged as a register error.

Examples:

```
MVI A,128       Move 128 decimal to register A.
MVI C,10D       Move 10 decimal to register C.
LXI H,2FH       Move 2F hexadecimal to registers HL.
MVI B,303Q      Move 303 octal to register B.
MVI A,'Y'       Move the ASCII value for Y to reg A.
MVI A,101B      Move 101 binary to register A.
JMP 0FFH        Jump to address FF hexadecimal.
```

EXPRESSIONS

Operands may be arithmetic expressions constructed from labels, constants, and the following operators:

+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division (remainder discarded)

Values are treated as 16-bit unsigned 2's complement numbers. Positive or negative overflow is allowed during expression evaluation, e.g., 32767+1=7FFFH+1=-32768 and -32768-1=7FFFH=32767. Expressions are evaluated from left to right; there is no operator precedence.

If an expression is used where an 8-bit quantity is required (e.g., MVI C,TEMP+10H), it must evaluate to a value in the range -256 to 255, or it will be flagged as a value error.

Examples:

```
MVI     A,255D/10H-5
LDA     POTTS/256*OFFSET
LXI     SP,30*2+STACK
```

High- and Low-order Byte Extraction

If an operand is preceded by the symbol <, the high-order byte of the evaluated expression will be used as the value of the operand. If an operand is preceded by the symbol >, the low-order byte will be used.

Note that the symbols < and > are not operators that may be applied to labels or constants within an expression. If more than one < or > appears within an expression, the rightmost will be used to determine whether to use the high- or low-order byte of the evaluated expression as the value of the operand. That is, the rightmost < or > is treated as if it preceded the entire expression, and the others will be totally ignored.

Examples:

```
 MVI A,>TEST      Loads register A with the least
*                 significant 8 bits of the value of
*                 the label TEST.
 MVI B,<0CC00H    Loads register B with the most
*                 significant byte of the 16-bit value
*                 CC00H, i.e., CCH.
 MVI C,<1234H     Loads register C with the value 12H.
 MVI C,>1234H     Loads register C with the value 34H.
```

COMMENT FIELD

The comment field must be separated from the operand field (or operation field for instructions or pseudo-operations that require no operand) by at least one space. Comments are not processed by the assembler, but are solely for the benefit of the programmer. Good comments are essential if a program is to be understood very long after it is written or is to be maintained by someone other than its author.

An entire line will be treated as a comment if it starts with an asterisk (*) or semicolon (;) in column 1 (column 6 if line numbers are used).

Examples:

```
0001 ; is input ready?
0002 LOOP IN STAT input device status
0003      ANI 1   test status bit
0004      JZ LOOP wait for data
0005 *data is now available
```

If listing file formatting is specified in the ASM command ($=options contains 1, 2, or 3), the comment field must be preceded by at least two spaces to ensure proper output formatting. Futhermore, instructions and pseudo-operations requiring no operand must be followed by a dummy operand (a peRiod is recommended).

Examples:

```
MVI A,10  comments
RZ .  comments
```

SECTION 4

PSEUDO-OPERATIONS

Pseudo-operations appear in a source program as instructions to the assembler and do not always generate object code. This section describes the pseudo-operations recognized by the NEVADA assembler.

In the following pseudo-operation formats, <expression> stands for a constant, label, arithmetic expression constructed from constants and labels. Optional elements are enclosed in square brackets [].

Equate <label> EQU <expression>

This pseudo-operation sets a label name to the 16-bit value that is represented in the operand field. That value holds for the entire assembly and may not be changed by another EQU.

Any label that appears in the operand field of an EQU statement must be defined in a statement earlier in the program.

Examples:

```
BELL EQU 7       The value of the label BELL is set to 7.
BELL2 EQU BELL*2  The label BELL2 is ste to 7*2.
```

Set Origin [<label>] ORG <expression>

This pseudo-operation sets the assembler's instruction location counter to the 16-bit value specified in the operand field. In other words, the object code generated by the statements that follow must be loaded beginning at the specified address in order to execute properly. The label, if present, is given the specified 16-bit value.

Any label that appears in the operand field of an ORG statement must be defined in a statement earlier in the program.

If no origin is specified at the beginning of the source

code, the assembler will set the origin to 100H. If no ORG pseudo-operation is used anywhere in the source program, successive bytes of object code will be stored at successive memory locations.

Examples:

```
        ORG 4000H   Determines that the object code generated
*                   by subsequent statements must be loaded
*                   in locations beginning at 4000H.
START   ORG 100H    Determines that the object code generated
*                   by subsequent statements must be loaded
*                   in locations beginning at 100H.
```

Set Execution Address XEQ <expression>

This pseudo-operation specifies the entry point address for the program, i.e., the address at which it is to begin execution. If a program contains no XEQ pseudo-operation, the object code file will contain a starting address of 100H. If more than one XEQ appears in a program, the last will be used.

An example of the difference between ORG and XEQ is that a program whose first 100 bytes are occupied by data will have an ORG address 100 bytes lower in memory than its XEQ address.

Example:

```
        XEQ 100H    The entry point address for the assembled
*                   program is set to 100H.
```

Define Storage [<label>] DS <expression>
[<label>] RES <expression>

Either of these pseudo-operations reserves the specified number of successive memory locations starting at the current address within the program. The contents of these locations are not defined and are not initialized at load time.

Any label that appears in the operand field of a DS or RES statement must be defined in a statement earlier in the program.

Examples:

```
SPEED DS 1      Reserves one byte.
 DS 400         Reserves 400 bytes.
 RES 177Q       Reserves 177 (octal) bytes.
```

Define byte [<label>] DB <expression>[,<expression>,...]

This pseudo-operation sets a memory location to an 8-bit value. If the operand field contains multiple expressions separated by commas, the expressions will define successive bytes of memory beginning at the current address. Each expression must evaluate to a number that can be represented in 8 bits.

Examples:

```
 DB 1   one byte is defined.
 DB 0FFH,303Q,100D,11010011B,3*BELL,-10  multiple bytes.
TABLE DB 'A','B','C','D',0      multiple bytes are defined.
```

Define Word [<label>] DW <expression>

This pseudo-operation sets two memory locations to a 16-bit quantity. The least significant (low-order) byte of the value is stored at the current address and the most significant byte (high-order) is stored at the current address + 1.

Examples:

```
SAVE DW 1234H   1234H is stored in memory, 34H in the
*               low-order byte and 12H in the high-order
*               byte.
YES DW 'OK'     The ASCII value for the letters 'O' and 'K'
*               is stored with the 'K' at the lower memory
*               address.
```

Define Double Byte [<label>] DDB <expression>

This pseudo-operation is almost the same as DW, except that the two bytes are stored in the opposite order: high-order byte first, followed by the low-order byte.

Example:

```
FIRST DDB 1234H    1234H is stored in memory, 12H in the
;                  low-order byte and 34H in the high-order
;                  byte.
```

Define ASCII String **[<label>] ASC #<ASCII string>#**
[<label>] ASCZ #<ASCII string>#

The ASC pseudo-operation puts a string of characters into successive memory locations starting at the current location. The special symbols # in the format are "delimiters"; they define the beginning and end of the ASCII character string. The assembler uses the first non-blank character found as the delimiter. The string immediately follows this delimiter, and ends at the next occurence of the same delimiter, or at a carriage return.

The ASCZ pseudo-operation is the same except that it appends a NUL (00H) to the end of the stored string.

Examples:

```
WORDS ASC "THIS IS AN ASCII STRING"
      ASCZ "THIS IS ANOTHER STRING"
```

Set ASCII List Flag **ASCF 0**
ASCF 1

If the operand field contains a 0, the listing of the assembled bytes of an ASCII string will be suppressed after the first line (four bytes). Likewise, only the first four assembled bytes of a DB pseudo-operation with multiple arguments will be listed. If a program contains many long strings, its listing will be easier to read if the ASCF pseudo-operation is used.

If the operand field contains a 1, the assembled form of subsequent ASCII strings and DB pseudo-operations with multiple argumrnts will be listed in full. This is the default condition.
See Appendix 3 for an example of the listing format.

Conditional Assembly **IF <expression>**

```
.
source code
.
ENDF
```

The value of the expression in the operand field governs whether or not subsequent code up to the matching ENDF will be assembled. If the expression evaluates to a 0 (false), the code will not be assembled. If the expression evaluates to a non-zero value (true), the code will be assembled. Blocks of code delimited by IF and ENDF ("conditional code") may be nested within another block of conditional code.

Any label that appears in the operand field of an IF...ENDF pseudo-operation must be defined in a statement earlier in the program.

Example:

```
YES EQU 1       Sets the value of the label 'YES' to 1.
NO EQU 0        Sets the value of the label 'NO' to 0.
*
 IF YES         the expression here is true (1), so the
 MVI A,'Y'      code on this line will be assembled.
 IF NO          The expression here is false (0), so the
 MVI A,'N'      code on this line will not be assembled.
 ENDF           This terminates the NO conditional.
 ENDF           This terminates the YES conditional.
```

List Conditional code **IFLS**

This pseudo-operation enables listing of conditional source code even though no object code is being generated because of a false IF condition. The assembler will not list such conditional source code if this pseudo-operation is not used.

Copy file **COPY [<unit:>]<file-name>**

This pseudo-operation copies source code from a disk file into a program being assembled. The code from the copied file will be assembled starting at the current address. When the copied file is exhausted, the assembler will continue to assemble from the original file. The resulting object code will be exactly like what would be generated if the copied source code were part of the original file, but the COPY pseudo-operation does not actually alter any source file.

A copied file may not copy another file. And all files

that are accessed by the COPY pseudo-operation must be of the same format as the main source file, i.e., either having or not having line numbers. The files must be type (.ASM).

EXAMPLES:

```
COPY FILE1
COPY B:FILE2
```

Listing Control **NLST**
LST

The NLST pseudo-operation suppresses all output to the listing file. Object code will still be output to the object code file and the lines containing errors will still be output to the error file. The LST pseudo-operation re-enables output to the listing file.

Listing Title **TITL <first line>"<second line>**

If the P option is specified in the ASM command, the one- or two-line title specified by this pseudo-operation will be printed centered at the top of each page of the listing.

Page Eject **PAGE**

If the P option is specified in the ASM command, this pseudo-operation causes a skip to the top of the next page of the listing.

End of Source file **END**

This pseudo-operation terminates each pass of the assembly. Only one END statement should be in the file or files to be assembled, and it should be the last statement encountered by the assembler. Since an end-of-file on the sourec code input file will also terminate each pass, the END statement is unnecessary in most cases.

SECTION 5

ERROR CODES AND MESSAGES

ASSEMBLER COMMAND ERRORS

A number of console messages may be generated in response to errors in the ASM command. When an error of this sort occurs, the assembly is aborted and control returns to CP/M.

EXPECTED NAME The source code input file name is missing.

ILLEGAL OPTION An unrecognized option specifier follows $.

91 ERROR IN EXTENDING THE FILE
92 END OF DISK DATA - DISK IS FULL
93 FINE NOT OPEN
94 NO MORE DIRECTORY SPACE - DISK IS FULL
95 FILE CANNOT BE FOUND
96 FILE ALREADY OPEN
97 READING UNWRITTEN DATA

ASSEMBLY ERRORS

If a statement contains one of the following errors, there will be a single letter error code in column 19 of the line output to the listing and/or error files. An error detected during both the first and the second pass of the assembler will be flagged twice in the listing(s). If the error is not an opcode error, NULs will be output as the second and , if appropriate, third bytes of object code for that instruction. If the error is an opcode error, the instruction will be assumed to be a three-byte instruction, and three NULs will be written to the listing and/or error files. The error codes are:

A	ARGUMENT ERROR	An illegal label or constant appears in the operand field. This might be 1) a number with a letter in it, e.g., 2L, 2) a label that starts with a number, e.g., 3STOP, or 3) an improper representation of a string, e.g., '''A''' in the operand field of a statement containing the ASCII pseudo-operation.
D	DUPLICATE LABEL	The source code contains multiple labels whose first five characters are identical.
L	LABEL ERROR	The symbol in the label field contains illegal characters, e.g., it starts with a number.
M	MISSING LABEL	An EQU instruction does not have a symbol in the label field.
O	OPCODE ERROR	The symbol in the operation field is not a valid 8080 instruction mnemonic or an assembler pseudo-operation mnemonic.
R	REGISTER ERROR	An expression used as a register designator does not have a legal value.
S	SYNTAX ERROR	A statement is not in the format required by the assembler.
U	UNDEFINED SYMBOL	A label used in the operand field is not defined, i.e., does not appear in the label field anywhere in the program, or is not defined prior to its use as an operand in an EQU, ORG, DS, RES, or IF pseudo-operation.
V	VALUE ERROR	The value of the operand lies outside the allowed range.

JUMP

C3	JMP	Adr
C2	JNZ	
CA	JZ	
D2	JNC	
DA	JC	
E2	JPO	
EA	JPE	
F2	JP	
FA	JM	
E9	PCHL	

CALL

CD	CALL	Adr
C4	CNZ	
CC	CZ	
D4	CNC	
DC	CC	
E4	CPO	
EC	CPE	
F4	CP	
FC	CM	

RETURN

C9	RET	
C0	RNZ	
C8	RZ	
D0	RNC	
D8	RC	
E0	RPO	
E8	RPE	
F0	RP	
F8	RM	

RESTART

C7	RST	0
CF	RST	1
D7	RST	2
DF	RST	3
E7	RST	4
EF	RST	5
F7	RST	6
FF	RST	7

ROTATE†

07	RLC	
0F	RRC	
17	RAL	
1F	RAR	

CONTROL

00	NOP	
76	HLT	
F3	DI	
FB	EI	

MOVE (cont)

58	MOV	E,B	
59	MOV	E,C	
5A	MOV	E,D	
5B	MOV	E,E	
5C	MOV	E,H	
5D	MOV	E,L	
5E	MOV	E,M	
5F	MOV	E,A	
60	MOV	H,B	
61	MOV	H,C	
62	MOV	H,D	
63	MOV	H,E	
64	MOV	H,H	
65	MOV	H,L	
66	MOV	H,M	
67	MOV	H,A	
68	MOV	L,B	
69	MOV	L,C	
6A	MOV	L,D	
6B	MOV	L,E	
6C	MOV	L,H	
6D	MOV	L,L	
6E	MOV	L,M	
6F	MOV	L,A	
70	MOV	M,B	
71	MOV	M,C	
72	MOV	M,D	
73	MOV	M,E	
74	MOV	M,H	
75	MOV	M,L	

77	MOV	M,A	
78	MOV	A,B	
79	MOV	A,C	
7A	MOV	A,D	
7B	MOV	A,E	
7C	MOV	A,H	
7D	MOV	A,L	
7E	MOV	A,M	
7F	MOV	A,A	

ACCUMULATOR*

80	ADD	B	
81	ADD	C	
82	ADD	D	
83	ADD	E	
84	ADD	H	
85	ADD	L	
86	ADD	M	
87	ADD	A	
88	ADC	B	
89	ADC	C	
8A	ADC	D	
8B	ADC	E	
8C	ADC	H	
8D	ADC	L	
8E	ADC	M	
8F	ADC	A	
90	SUB	B	
91	SUB	C	
92	SUB	D	
93	SUB	E	
94	SUB	H	
95	SUB	L	
96	SUB	M	
97	SUB	A	
98	SBB	B	
99	SBB	C	
9A	SBB	D	
9B	SBB	E	
9C	SBB	H	
9D	SBB	L	
9E	SBB	M	
9F	SBB	A	
A0	ANA	B	
A1	ANA	C	
A2	ANA	D	
A3	ANA	E	
A4	ANA	H	
A5	ANA	L	
A6	ANA	M	
A7	ANA	A	
A8	XRA	B	
A9	XRA	C	
AA	XRA	D	
AB	XRA	E	
AC	XRA	H	
AD	XRA	L	
AE	XRA	M	
AF	XRA	A	
B0	ORA	B	
B1	ORA	C	
B2	ORA	D	
B3	ORA	E	
B4	ORA	H	
B5	ORA	L	
B6	ORA	M	
B7	ORA	A	
B8	CMP	B	
B9	CMP	C	
BA	CMP	D	
BB	CMP	E	
BC	CMP	H	
BD	CMP	L	
BE	CMP	M	
BF	CMP	A	

CONSTANT DEFINITION

0BDH, 1AH	Hex
105D, 105	Decimal
72O, 72Q	Octal
11011B, 00110B	Binary
'TEST', 'A', 'B'	ASCII

OPERATORS

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

MOVE IMMEDIATE

06	MVI	B,	D8
0E	MVI	C,	
16	MVI	D,	
1E	MVI	E,	
26	MVI	H,	
2E	MVI	L,	
36	MVI	M,	
3E	MVI	A,	

Acc IMMEDIATE*

C6	ADI	D8
CE	ACI	
D6	SUI	
DE	SBI	
E6	ANI	
EE	XRI	
F6	ORI	
FE	CPI	

LOAD IMMEDIATE

01	LXI	B,	D16
11	LXI	D,	
21	LXI	H,	
31	LXI	SP,	

STACK OPS

C5	PUSH	B
D5	PUSH	D
E5	PUSH	H
F5	PUSH	PSW
C1	POP	B
D1	POP	D
E1	POP	H
F1	POP	PSW*
E3	XTHL	
F9	SPHL	

MOVE

40	MOV	B,B	
41	MOV	B,C	
42	MOV	B,D	
43	MOV	B,E	
44	MOV	B,H	
45	MOV	B,L	
46	MOV	B,M	
47	MOV	B,A	
48	MOV	C,B	
49	MOV	C,C	
4A	MOV	C,D	
4B	MOV	C,E	
4C	MOV	C,H	
4D	MOV	C,L	
4E	MOV	C,M	
4F	MOV	C,A	
50	MOV	D,B	
51	MOV	D,C	
52	MOV	D,D	
53	MOV	D,E	
54	MOV	D,H	
55	MOV	D,L	
56	MOV	D,M	
57	MOV	D,A	

DOUBLE ADD†

09	DAD	B
19	DAD	D
29	DAD	H
39	DAD	SP

SPECIALS

EB	XCHG	
27	DAA*	
2F	CMA	
37	STC†	
3F	CMC†	

INPUT/OUTPUT

D3	OUT	D8
DB	IN	D8

INCREMENT†*

04	INR	B
0C	INR	C
14	INR	D
1C	INR	E
24	INR	H
2C	INR	L
34	INR	M
3C	INR	A
03	INX	B
13	INX	D
23	INX	H
33	INX	SP

DECREMENT†*

05	DCR	B
0D	DCR	C
15	DCR	D
1D	DCR	E
25	DCR	H
2D	DCR	L
35	DCR	M
3D	DCR	A
0B	DCX	B
1B	DCX	D
2B	DCX	H
3B	DCX	SP

LOAD/STORE

0A	LDAX	B
1A	LDAX	D
2A	LHLD	Adr
3A	LDA	Adr
02	STAX	B
12	STAX	D
22	SHLD	Adr
32	STA	Adr

PSEUDO INSTRUCTION

ORG	Adr	
END		
EQU	D16	
DS	D16	
DB	D8	[,]
DW	D16	[,]

STANDARD SETS

A	SET	7
B	SET	0
C	SET	1
D	SET	2
E	SET	3
H	SET	4
L	SET	5
M	SET	6
SP	SET	6
PSW	SET	6

NEVADA ASSEMBLER

D8 = constant, or logical/arithmetic expression that evaluates to an 8 bit data quantity.

* = all Flags (C,Z,S,P) affected

D16 = constant, or logical/arithmetic expression that evaluates to a 16 bit data quantity.

† = only CARRY affected

Adr = 16 bit address

** = all Flags except CARRY affected; (exception: INX & DCX affect no Flags)

APPENDIX I.

00	NOP		28	---		50	MOV	D,B	78	MOV	A,B	A0	ANA	B	C8	RZ		F0	RP				
01	LXI	B,D16	29	DAD	H	51	MOV	D,C	79	MOV	A,C	A1	ANA	C	C9	RET		F1	POP	PSW			
02	STAX	B	2A	LHLD	Adr	52	MOV	D,D	7A	MOV	A,D	A2	ANA	D	CA	JZ		F2	JP	Adr			
03	INX	B	2B	DCX	H	53	MOV	D,E	7B	MOV	A,E	A3	ANA	E	CB	---		F3	DI				
04	INR	B	2C	INR	L	54	MOV	D,H	7C	MOV	A,H	A4	ANA	H	CC	CZ	Adr	F4	CP	Adr			
05	DCR	B	2D	DCR	L	55	MOV	D,L	7D	MOV	A,L	A5	ANA	L	CD	CALL	Adr	F5	PUSH	PSW			
06	MVI	B,D8	2E	MVI	L,D8	56	MOV	D,M	7E	MOV	A,M	A6	ANA	M	CE	ACI	D8	F6	ORI	D8			
07	RLC		2F	CMA		57	MOV	D,A	7F	MOV	A,A	A7	ANA	A	CF	RST	1	F7	RST	6			
08	---		30	---		58	MOV	E,B	80	ADD	B	A8	XRA	B	D0	RNC		F8	RM				
09	DAD	B	31	LXI	SP,D16	59	MOV	E,C	81	ADD	C	A9	XRA	C	D1	POP	D	F9	SPHL				
0A	LDAX	B	32	STA	Adr	5A	MOV	E,D	82	ADD	D	AA	XRA	D	D2	JNC	Adr	FA	JM	Adr			
0B	DCX	B	33	INX	SP	5B	MOV	E,E	83	ADD	E	AB	XRA	E	D3	OUT	D8	FB	EI				
0C	INR	C	34	INR	M	5C	MOV	E,H	84	ADD	H	AC	XRA	H	D4	CNC	Adr	FC	CM	Adr			
0D	DCR	C	35	DCR	M	5D	MOV	E,L	85	ADD	L	AD	XRA	L	D5	PUSH	D	FD	---				
0E	MVI	C,D8	36	MVI	M,D8	5E	MOV	E,M	86	ADD	M	AE	XRA	M	D6	SUI	D8	FE	CPI	D8			
0F	RRC		37	STC		5F	MOV	E,A	87	ADD	A	AF	XRA	A	D7	RST	2	FF	RST	7			
10	---		38	---		60	MOV	H,B	88	ADC	B	B0	ORA	B	D8	RC							
11	LXI	D,D16	39	DAD	SP	61	MOV	H,C	89	ADC	C	B1	ORA	C	D9	---							
12	STAX	D	3A	LDA	Adr	62	MOV	H,D	8A	ADC	D	B2	ORA	D	DA	JC	Adr						
13	INX	D	3B	DCX	SP	63	MOV	H,E	8B	ADC	E	B3	ORA	E	DB	IN	D8						
14	INR	D	3C	INR	A	64	MOV	H,H	8C	ADC	H	B4	ORA	H	DC	CC	Adr						
15	DCR	D	3D	DCR	A	65	MOV	H,L	8D	ADC	L	B5	ORA	L	DD	---							
16	MVI	D,D8	3E	MVI	A,D8	66	MOV	H,M	8E	ADC	M	B6	ORA	M	DE	SBI	D8						
17	RAL		3F	CMC		67	MOV	H,A	8F	ADC	A	B7	ORA	A	DF	RST	3						
18	---		40	MOV	B,B	68	MOV	L,B	90	SUB	B	B8	CMP	B	E0	RPO							
19	DAD	D	41	MOV	B,C	69	MOV	L,C	91	SUB	C	B9	CMP	C	E1	POP	H						
1A	LDAX	D	42	MOV	B,D	6A	MOV	L,D	92	SUB	D	BA	CMP	D	E2	JPO	Adr						
1B	DCX	D	43	MOV	B,E	6B	MOV	L,E	93	SUB	E	BB	CMP	E	E3	XTHL							
1C	INR	E	44	MOV	B,H	6C	MOV	L,H	94	SUB	H	BC	CMP	H	E4	CPO	Adr						
1D	DCR	E	45	MOV	B,L	6D	MOV	L,L	95	SUB	L	BD	CMP	L	E5	PUSH	H						
1E	MVI	E,D8	46	MOV	B,M	6E	MOV	L,M	96	SUB	M	BE	CMP	M	E6	ANI	D8						
1F	RAR		47	MOV	B,A	6F	MOV	L,A	97	SUB	A	BF	CMP	A	E7	RST	4						
20	---		48	MOV	C,B	70	MOV	M,B	98	SBB	B	C0	RNZ		E8	RPE							
21	LXI	H,D16	49	MOV	C,C	71	MOV	M,C	99	SBB	C	C1	POP	B	E9	PCHL							
22	SHLD	Adr	4A	MOV	C,D	72	MOV	M,D	9A	SBB	D	C2	JNZ	Adr	EA	JPE	Adr						
23	INX	H	4B	MOV	C,E	73	MOV	M,E	9B	SBB	E	C3	JMP	Adr	EB	XCHG							
24	INR	H	4C	MOV	C,H	74	MOV	M,H	9C	SBB	H	C4	CNZ	Adr	EC	CPE	Adr						
25	DCR	H	4D	MOV	C,L	75	MOV	M,L	9D	SBB	L	C5	PUSH	B	ED	---							
26	MVI	H,D8	4E	MOV	C,M	76	HLT		9E	SBB	M	C6	ADI	D8	EE	XRI	D8						
27	DAA		4F	MOV	C,A	77	MOV	M,A	9F	SBB	A	C7	RST	0	EF	RST	5						

D8 = constant, or logical/arithmetic expression that evaluates to an 8 bit data quantity

D16 = constant, or logical/arithmetic expression that evaluates to a 16 bit data quantity.

Adr = 16 bit address

HEX-ASCII TABLE

HEX-ASCII TABLE

Non-Printing

00	NULL
07	BELL
09	TAB
0A	LF
0B	VT
0C	FORM
0D	CR
11	X-ON
12	TAPE
13	X-OFF
14	
1B	ESC
7D	ALT MODE
7F	RUB OUT

Printing		Characters	
30	0	40	@
31	1	20	space
32	2	21	!
33	3	22	"
34	4	23	#
35	5	24	$
36	6	25	%
37	7	26	&
38	8	27	'
39	9	28	(
		29	)
41	A	2A	*
42	B	2B	+
43	C	2C	,
44	D	2D	-
45	E	2E	.
46	F	2F	/
47	G	3A	:
48	H	3B	;
49	I	3C	<
4A	J	3D	=
4B	K	3E	>
4C	L	3F	?
4D	M	5B	[
4E	N	5C	\
4F	O	5D	]
50	P	5E	↑ (^)
51	Q	5F	← (-)
52	R		
53	S		
54	T		
55	U		
56	V		
57	W		
58	X		
59	Y		
5A	Z		

APPENDIX 2
TABLE OF ASCII CODES (Zero Parity)

Paper tape 1 2 3 . 4 5 6 7 P	Upper Octal	Octal	Decimal	Hex	Character		
.	0000	000	0	00	ctrl @	NUL	
● .	0004	001	1	01	ctrl A	SOH	Start of Heading
● .	0010	002	2	02	ctrl B	STX	Start of Text
●● .	0014	003	3	03	ctrl C	ETX	End of Text
●.	0020	004	4	04	ctrl D	EOT	End of Xmit
● ●.	0024	005	5	05	ctrl E	ENQ	Enquiry
●●.	0030	006	6	06	ctrl F	ACK	Acknowledge
●●●.	0034	007	7	07	ctrl G	BEL	Audible Signal
.●	0040	010	8	08	ctrl H	BS	Back Space
● .●	0044	011	9	09	ctrl I	HT	Horizontal Tab
● .●	0050	012	10	0A	ctrl J	LF	Line Feed
●● .●	0054	013	11	0B	ctrl K	VT	Vertical Tab
●.●	0060	014	12	0C	ctrl L	FF	Form Feed
● ●.●	0064	015	13	0D	ctrl M	CR	Carriage Return
●●.●	0070	016	14	0E	ctrl N	SO	Shift Out
●●●.●	0074	017	15	0F	ctrl O	SI	Shift In
. ●	0100	020	16	10	ctrl P	DLE	Data Line Escape
● . ●	0104	021	17	11	ctrl Q	DC1	X On
● . ●	0110	022	18	12	ctrl R	DC2	Aux On
●● . ●	0114	023	19	13	ctrl S	DC3	X Off
●. ●	0120	024	20	14	ctrl T	DC4	Aux Off
● ●. ●	0124	025	21	15	ctrl U	NAK	Negative Acknowledge
●●. ●	0130	026	22	16	ctrl V	SYN	Synchronous File
●●●. ●	0134	027	23	17	ctrl W	ETB	End of Xmit Block
.●●	0140	030	24	18	ctrl X	CAN	Cancel
● .●●	0144	031	25	19	ctrl Y	EM	End of Medium
● .●●	0150	032	26	1A	ctrl Z	SUB	Substitute
●● .●●	0154	033	27	1B	ctrl [	ESC	Escape
●.●●	0160	034	28	1C	ctrl \	FS	File Separator
● ●.●●	0164	035	29	1D	ctrl]	GS	Group Separator
●●.●●	0170	036	30	1E	ctrl ^	RS	Record Separator
●●●.●●	0174	037	31	1F	ctrl _	US	Unit Separator
. ●	0200	040	32	20	Space		
● . ●	0204	041	33	21	!		
● . ●	0210	042	34	22	"		
●● . ●	0214	043	35	23	#		
●. ●	0220	044	36	24	$		
● ●. ●	0224	045	37	25	%		
●●. ●	0230	046	38	26	&		
●●●. ●	0234	047	39	27	'		
.● ●	0240	050	40	28	(		
● .● ●	0244	051	41	29	)		
● .● ●	0250	052	42	2A	*		
●● .● ●	0254	053	43	2B	+		
●.● ●	0260	054	44	2C	,		
● ●.● ●	0264	055	45	2D	-		
●●.● ●	0270	056	46	2E	.		
●●●.● ●	0274	057	47	2F	/		
. ●●	0300	060	48	30	0		
● . ●●	0304	061	49	31	1		
● . ●●	0310	062	50	32	2		
●● . ●●	0314	063	51	33	3		
●. ●●	0320	064	52	34	4		
● ●. ●●	0324	065	53	35	5		
●●. ●●	0330	066	54	36	6		
●●●. ●●	0334	067	55	37	7		
.●●●	0340	070	56	38	8		
● .●●●	0344	071	57	39	9		
● .●●●	0350	072	58	3A	:		
●● .●●●	0354	073	59	3B	;		
●.●●●	0360	074	60	3C	<		
● ●.●●●	0364	075	61	3D	=		
●●.●●●	0370	076	62	3E	>		
●●●.●●●	0374	077	63	3F	?		

APPENDIX 2
TABLE OF ASCII CODES *(Cont'd)* (Zero Parity)

Paper tape 1 2 3 . 4 5 6 7 P	Upper Octal	Octal	Decimal	Hex	Character	
. . . . ●	0400	100	64	40	@	
● . . . . . . ●	0404	101	65	41	A	
. ● . . . . . ●	0410	102	66	42	B	
● ● . . . . . ●	0414	103	67	43	C	
. . ● . . . . ●	0420	104	68	44	D	
● . ● . . . . ●	0424	105	69	45	E	
. ● ● . . . . ●	0430	106	70	46	F	
● ● ● . . . . ●	0434	107	71	47	G	
. . . ● . . . ●	0440	110	72	48	H	
● . . ● . . . ●	0444	111	73	49	I	
. ● . ● . . . ●	0450	112	74	4A	J	
● ● . ● . . . ●	0454	113	75	4B	K	
. . ● ● . . . ●	0460	114	76	4C	L	
● . ● ● . . . ●	0464	115	77	4D	M	
. ● ● ● . . . ●	0470	116	78	4E	N	
● ● ● ● . . . ●	0474	117	79	4F	O	
. . . . ● . . ●	0500	120	80	50	P	
● . . . ● . . ●	0504	121	81	51	Q	
. ● . . ● . . ●	0510	122	82	52	R	
● ● . . ● . . ●	0514	123	83	53	S	
. . ● . ● . . ●	0520	124	84	54	T	
● . ● . ● . . ●	0524	125	85	55	U	
. ● ● . ● . . ●	0530	126	86	56	V	
● ● ● . ● . . ●	0534	127	87	57	W	
. . . ● ● . . ●	0540	130	88	58	X	
● . . ● ● . . ●	0544	131	89	59	Y	
. ● . ● ● . . ●	0550	132	90	5A	Z	
● ● . ● ● . . ●	0554	133	91	5B	[	
. . ● ● ● . . ●	0560	134	92	5C	\	
● . ● ● ● . . ●	0564	135	93	5D	]	
. ● ● ● ● . . ●	0570	136	94	5E	^	
● ● ● ● ● . . ●	0574	137	95	5F	_	
. . . . . ● ●	0600	140	96	60	`	
● . . . . ● ●	0604	141	97	61	a	
. ● . . . ● ●	0610	142	98	62	b	
● ● . . . ● ●	0614	143	99	63	c	
. . ● . . ● ●	0620	144	100	64	d	
● . ● . . ● ●	0624	145	101	65	e	
. ● ● . . ● ●	0630	146	102	66	f	
● ● ● . . ● ●	0634	147	103	67	g	
. . . ● . ● ●	0640	150	104	68	h	
● . . ● . ● ●	0644	151	105	69	i	
. ● . ● . ● ●	0650	152	106	6A	j	
● ● . ● . ● ●	0654	153	107	6B	k	
. . ● ● . ● ●	0660	154	108	6C	l	
● . ● ● . ● ●	0664	155	109	6D	m	
. ● ● ● . ● ●	0670	156	110	6E	n	
● ● ● ● . ● ●	0674	157	111	6F	o	
. . . . ● ● ●	0700	160	112	70	p	
● . . . ● ● ●	0704	161	113	71	q	
. ● . . ● ● ●	0710	162	114	72	r	
● ● . . ● ● ●	0714	163	115	73	s	
. . ● . ● ● ●	0720	164	116	74	t	
● . ● . ● ● ●	0724	165	117	75	u	
. ● ● . ● ● ●	0730	166	118	76	v	
● ● ● . ● ● ●	0734	167	119	77	w	
. . . ● ● ● ●	0740	170	120	78	x	
● . . ● ● ● ●	0744	171	121	79	y	
. ● . ● ● ● ●	0750	172	122	7A	z	
● ● . ● ● ● ●	0754	173	123	7B	{	
. . ● ● ● ● ●	0760	174	124	7C	|	
● . ● ● ● ● ●	0764	175	125	7D	}	
. ● ● ● ● ● ●	0770	176	126	7E	~	Prefix
● ● ● ● ● ● ●	0774	177	127	7F	DEL	Rubout

APPENDIX 3

ASSEMBLER LISTING

```
ADDRESS  ASSEMBLED CODE   ERROR FLAG  LINE NO.  LABEL   OPERATION  OPERAND    COMMENT

                          0000 *
                          0001 *SEARCH TABLE FOR MATCH TO STRING
                          0002 *EACH TABLE ENTRY IS FOLLOWED BY A TWO-BYTE DISPATCH ADDRESS.
                          0003 *TABLE MUST HAVE AT LEAST ONE ENTRY AND IS TERMINATED BY A
                          0004 *ZERO BYTE.
                          0005 *ON ENTRY:  HL POINTS TO STRING
                          0006 *           DE POINTS TO TABLE
                          0007 *           C  IS NUMBER OF CHARACTERS IN TABLE ENTRIES
                          0008 *ON RETURN: ZERO FLAG SET IF NO MATCH, ELSE DE POINTS TO
                          0009 *           DISPATCH ADDRESS
                          0010 *
0100 E5                   0011 TSRCH  PUSH   H      SAVE STRING ADDRESS
0101 41                   0012        MOV    B,C    INITIALIZE CHARACTER COUNT
0102 1A                   0013 TS1    LDAX   D      COMPARE CHARACTERS
0103 BE                   0014        CMP    M
0104 C2 11 01             0015        JNZ    TS3
0107 23                   0016        INX    H      CHARACTERS MATCH, GO ON TO NEXT
0108 13                   0017        INX    D
0109 05                   0018        DCR    B
010A C2 02 01             0019        JNZ    TS1
010D F6 01                0020        ORI    1      MATCHING ENTRY FOUND
010F E1                   0021 TS2    POP    H
0110 C9                   0022        RET
0111 B7                   0023 TS3    ORA    A      TEST FOR END OF TABLE
0112 CA 0F 01             0024        JZ     TS2
0115 13                   0025 TS4    INX    D      SKIP TO NEXT ENTRY
0116 05                   0026        DCR    B
0117 C2 15 01             0027        JNZ    TS4
011A 13                   0028        INX    D
011B 13                   0029        INX    D
011C E1                   0030        POP    H
011D C3 00 01             0031        JMP    TSRCH
                          0032 *
                          0033 *EXAMPLE OF TSRCH USE:
                          0034 *
                          0035 *(ASSUME HL POINTS TO A FOUR-CHARACTER COMMAND STRING)
0120 11 35 01             0036        LXI    D,CTABL  DE POINTS TO COMMAND TABLE
0123 0E 04                0037        MVI    C,4    TABLE ENTRIES ARE FOUR CHARACTERS LONG
0125 CD 00 01             0038        CALL   TSRCH
0128 CA 00 00        U    0039        JZ     ERROR  COMMAND NOT IN TABLE
012B EB                   0040        XCHG   .      SET UP STACK FOR RETURN TO MAIN ROUTINE
012C 11 00 00        U    0041        LXI    D,COMMAND
012F D5                   0042        PUSH   D
0130 7E                   0043        MOV    A,M    DISPATCH TO APPROPRIATE COMMAND ROUTINE
0131 23                   0044        INX    H
0132 66                   0045        MOV    H,M
0133 6F                   0046        MOV    L,A
0134 E9                   0047        PCHL
                          0048 *
                          0049 *COMMAND TABLE
                          0050 *
0135 43 4F 4D 31          0051 CTABL  ASC    'COM1'  FIRST ENTRY
0139 00 00           U    0052        DW     SUB1   ADDRESS OF SUB1
013B 43 4F 4D 32          0053        ASC    'COM2'  SECOND ENTRY
013F 00 00           U    0054        DW     SUB2   ADDRESS OF SUB2
0141 00                   0055        DB     0      END OF TABLE MARK
```

SYMBOL TABLE LISTING

Label	Addr.	Label	Addr.	Label	Addr.	Label	Addr.
CTABL	0135	TS1	0102	TS2	010F	TS3	0111
TS4	0115	TSRCH	0100				

APPENDIX 4

This is a sample program. The loader source code.

```
0001 ***************************************************
0002 *
0003 *                  RUNA file-name[.ZCL]
0004 *
0005 *    An .OBJ file consists of one or more segments that
0006 *    have the format:
0007 *       #BYTES          DESCRIPTION
0008 *         2          Number of code and data bytes in
0009 *                    segment
0010 *         2          Load address of code and data
0011 *                    belonging to the segment.
0012 *       Variable     Code and/or data.
0013 *
0014 *    The run time package will load each segment at the
0015 *    specified address until a starting address is
0016 *    encountered.  A starting address is represented as
0017 *    load address with a zero byte count.
0018 *
0019 * ************************************************
0020 *
0021 RELOC EQU 0    ;4200H FOR TRS-80 MOD 1
0022 BDOS EQU 5+RELOC  ;CP/M
0023 BLKSIZ EQU 128
0024 OFCB EQU 5CH+RELOC  ;IN CP/M
0025 OEX EQU OFCB+12
0026 OCR EQU OFCB+32
0027 OBUF EQU 80H+RELOC  ;IN CP/M
0028 *
0029 CSTART EQU $
0030  LXI SP,STK
0031  MVI C,0CH  ;RETURN VERSION #
0032  CALL BDOS
0033  MOV A,L
0034  ORA A
0035  JNZ VER2X
0036  LDA 4+RELOC  ;CPM 1.4 DEFAULT DRIVE
0037  CPI 5
0038  JC SETDF
0039  XRA A
0040 SETDF EQU $  ;11-30-81 FOR MP/M II
0041  STA ODRIVE  ;DEFAULT DRIVE
0042 *  GET OPTIONS FROM TYPE FIELD
0043  LXI H,5CH+8
0044  MVI C,4
0045 NEXT EQU $
0046  INX H
0047  DCR C
0048  JZ NOOPTIONS
0049  MOV A,M
0050  CPI ' '
```

```
0051  JZ NOOPTIONS
0052  CPI 'Z'
0053  JZ ZEROFIL
0054  CPI 'C'
0055  JZ COMFILE
0056  CPI 'L'
0057  JZ NOEXEC
0058 * ERROR ILLEGAL OPTION
0059  LXI H,MESGA
0060  CALL DISPLAY
0061  JMP 0+RELOC
0062 *  GET SIZE OF INSTRUCTION
0063 GETSZ LXI H,TBL-1
0064 AGAIN MOV A,C
0065  INX H
0066  MOV B,M
0067  ANA B
0068  JZ BYTE1
0069  INX H
0070  MOV B,M
0071  XRA B
0072  INX H
0073  JNZ AGAIN
0074  MOV A,M
0075  RET .  EXIT
0076 BYTE1 MVI A,1
0077  RET .  EXIT
0078 *
0079 REL EQU $  RELOCATION
0080  PUSH H
0081  PUSH D
0082  PUSH PSW
0083  INX H
0084  MOV E,M
0085  INX H
0086  MOV A,M
0087  ORA A  WE DON'T RELOCATE BELOW 100+RELOC
0088  JZ NOREL
0089  MOV D,A
0090  PUSH H
0091  LHLD BASE
0092  DAD D  ADDRESS IS NOW ADJUSTED
0093  XCHG
0094  POP H
0095  MOV M,D  PUT IT BACK
0096  DCX H
0097  MOV M,E
0098 NOREL EQU $
0099  POP PSW
0100  POP D
0101  POP H
0102  RET
0103 *
0104 VER2X EQU $
0105  MVI C,19H  ;GET CPM 2.X DEFAULT DRIVE
0106  CALL BDOS
0107  JMP SETDF
0108 *
```

```
0109 MESGA ASC 'ILLEGAL OPTION'
0110  DB 0DH,0AH
0111  ASC 'RUN D:FILE.ZCL<CR>'
0112  DB 0DH,0AH,0
0113 *
0114 TBL DB -1,11101001B,1
0115  DB -1,11001101B,3
0116  DB 11000111B,11000100B,3
0117  DB -1,11000011B,3
0118  DB 11000111B,11000010B,3
0119  DB 11000111B,11000111B,1
0120  DB -1,11001001B,1
0121  DB 11000111B,11000000B,1
0122  DB 11001111B,1,3
0123  DB 11100111B,00100010B,3
0124  DB 11110111B,11010011B,2
0125  DB 11000111B,6,2
0126  DB 11000111B,11000110B,2
0127  DB 0              END OF TABLE
0128 *
0129 BASE DW 0  BASE ADJ TO ADD TO ADDRESS TO BE RELOCATED
0130 START DW 0  STARTING ADDR OF RELOCATED CODE
0131 *
0132 ZEROFILL EQU $
0133  STA ZX
0134  JMP NEXT
0135 *
0136 COMFILE EQU $
0137  STA CX
0138  JMP NEXT
0139 *
0140 NOEXEC EQU $
0141  STA LX
0142  JMP NEXT
0143 *
0144 OSET EQU $
0145  LXI D,OBUF
0146  MVI C,26  ;SET DMA
0147  CALL BDOS
0148  LDA ODRIVE
0149  MVI D,0
0150  MOV E,A
0151  MVI C,14  ;SET DRIVE
0152  CALL BDOS
0153  LXI D,OFCB
0154  RET
0155 *
0156 NOOPTIONS EQU $
0157  LXI H,080H+3+RELOC
0158  MOV A,M
0159  CPI ':'  ;WAS DRIVE REQUESTED?
0160  JNZ DEFDRIVE  ;DEFAULT IS SET
0161  DCX H
0162  MOV A,M
0163  CPI 'A'
0164  JC DEFDRIVE
0165  SUI 'A'
0166  STA ODRIVE
```

```
0167 DEFDRIVE EQU $
0168  CALL SETFCB
0169  MVI M,'O'
0170  INX H
0171  MVI M,'B'
0172  INX H
0173  MVI M,'J'
0174  CALL OSET
0175  MVI C,15  ;OPEN
0176  CALL BDOS
0177  CPI -1
0178  JZ OERR  ;OPEN ERROR
0179  XRA A
0180  STA OCR
0181 *  RELOCATE CODE TO JUST BELOW CP/M
0182  LHLD 6+RELOC
0183  DCX H  HIGHEST ADDR
0184  LXI B,LAST-LOADFILE  SIZE OF CODE TO BE RELOCATED
0185  MOV A,L
0186  SUB C
0187  MOV L,A
0188  MOV A,H
0189  SBB B
0190  MOV H,A
0191 ; H&L= STARTING ADDRESS
0192  SHLD START
0193  PUSH H
0194  LXI D,LOADFILE
0195  MOV A,L
0196  SUB E
0197  MOV L,A
0198  MOV A,H
0199  SBB D
0200  MOV H,A
0201  SHLD BASE
0202  POP H
0203  LXI B,CONSTANTS-LOADFILE  SIZE OF INSTRUCTION MOVE
0204  XCHG
0205 NXTI EQU $
0206  PUSH H
0207  PUSH D
0208  PUSH B
0209  MOV C,M  GET OPCODE
0210  CALL GETSZ  GET SIZE OF INSTRUCTION
0211  POP B
0212  POP D
0213  POP H
0214  CPI 3
0215  JC SKPREL
0216  CALL REL  RELOCATE ADDR IN THIS 3 BYTE INST
0217 SKPREL EQU $
0218  PUSH B
0219  PUSH PSW
0220  MOV C,A  SIZE
0221 NXTM EQU $
0222  MOV A,M
0223  STAX D
0224  INX H
```

```
0225  INX D
0226  DCR C
0227  JNZ NXTM
0228  POP PSW
0229  POP B
0230 NXTD EQU $
0231  DCX B
0232  DCR A
0233  JNZ NXTD
0234  MOV A,C
0235  ORA B
0236  JNZ NXTI
0237 *  RELOCATE CONSTANTS
0238  LXI B,LAST-CONSTANTS  SIZE OF CONSTANTS
0239 NXTC EQU $
0240  MOV A,M
0241  STAX D
0242  INX H
0243  INX D
0244  DCX B
0245  MOV A,C
0246  ORA B
0247  JNZ NXTC
0248  LHLD START
0249  PCHL .  CODE HAS BEEN RELOCATED NOW GO TO IT
0250 *
0251 ****************************************************
0252 *  RUNA  A:FILE.OBJ<CR>
0253 *
0254 *  MOVE PARAMETERS AND CHECK
0255 ****************************************************
0256 *
0257 LOADFILE EQU $
0258  LXI SP,STK  SET STACK AFTER RELOCATION
0259  LDA ZX  ZERO FILL MEMORY?
0260  ORA A
0261  JZ SKPCLR
0262  LXI D,LOADFILE-1
0263  MVI H,1     STARTING ADDR  +RELOC
0264  MVI L,0
0265 CLEAR EQU $
0266  XRA A
0267  MOV M,A
0268  INX H
0269  MOV A,L
0270  SUB E
0271  MOV A,H
0272  SBB D
0273  JC CLEAR
0274 SKPCLR EQU $
0275  CALL ORD  ;GET 1ST RECORD OF .OBJ FILE
0276 OLOAD EQU $
0277  CALL GETOP
0278 MAO MOV A,M  ;MOVE 4 BYTES FROM BUF TO WORK
0279  STAX D
0280  INX H
0281  INX D
0282  DCR C
```

```
0283  CZ ORD
0284  DCR B
0285  JNZ MAO
0286 ; H&L = BUFFER  C=COUNT
0287  XCHG
0288  LHLD OWRK  ;SIZE OF NEXT READ
0289  MOV A,L
0290  ORA H
0291  JZ CLOSE
0292  SHLD OSIZE
0293  LHLD OWRK+2
0294  XCHG
0295 MAOA MOV A,M  ;MOVE FROM BUF TO OBJ ADDR
0296  STAX D
0297  INX H
0298  INX D
0299  DCR C
0300  CZ ORD
0301  PUSH H
0302  LHLD OSIZE
0303  DCX H
0304  SHLD OSIZE
0305  MOV A,L
0306  ORA H
0307  POP H
0308  JNZ MAOA
0309  CALL SAVOP
0310  JMP OLOAD
0311 *
0312 GETOP EQU $  ;GET O POINTERS
0313  LXI D,OWRK
0314  LHLD OCBA  ;BUF ADDR
0315  MVI B,4  ;LENGTH OF WRK
0316  LDA OCBC  ;BUF CNT
0317  MOV C,A
0318  RET
0319 *
0320 ORD EQU $
0321  PUSH B
0322  PUSH D  ;OPNT
0323  LXI D,OFCB
0324  MVI C,20  ;READ
0325  CALL BDOS
0326  POP D
0327  POP B
0328  ORA A
0329  JNZ RERR
0330  LXI H,OBUF
0331  MVI C,BLKSIZ
0332  RET
0333 *
0334 SAVOP EQU $
0335  SHLD OCBA  ;BUFF ADDR
0336  MOV A,C
0337  STA OCBC  ;BUF CNT
0338  LDA HIGH
0339  CMP D
0340  RNC
```

```
0341  MOV A,D
0342  STA HIGH
0343  RET
0344 *
0345 CLOSE EQU $
0346  LXI D,OFCB
0347  MVI C,16  ;CLOSE
0348  CALL BDOS
0349  LDA CX
0350  ORA A
0351  JNZ GENCOM
0352  LDA LX
0353  ORA A
0354  JNZ 0+RELOC  LOAD BUT DON'T EXECUTE
0355  LHLD OWRK+2  ;STARTING ADDRESS
0356  PCHL
0357 *
0358 SETFCB EQU $
0359  XRA A
0360  STA OFCB
0361  STA OCR
0362  LXI H,OEX
0363  MVI C,4
0364 EXLUP EQU $  ;10-2-81 ZERO CPM EXT AREA
0365  MOV M,A
0366  INX H
0367  DCR C
0368  JNZ EXLUP
0369  LXI H,OBUF
0370  SHLD OCBA
0371  MVI A,BLKSIZ
0372  STA OCBC
0373  LXI H,5CH+9+RELOC  ;CP/M FILE TYPE
0374  RET
0375 *
0376 CREATE EQU $
0377  LXI D,OFCB
0378  MVI C,22  CREATE
0379  CALL BDOS
0380  CPI -1
0381  RNZ
0382 OERR EQU $
0383  LXI H,MESGO  OPEN ERROR
0384  CALL DISPLAY
0385  JMP OXT1
0386 *
0387 GENCOM EQU $  GENERATE .COM FILE
0388  CALL SETFCB
0389  MVI M,'C'
0390  INX H
0391  MVI M,'O'
0392  INX H
0393  MVI M,'M'  .COM IN FCB
0394 *OPEN
0395  LXI D,OFCB
0396  MVI C,15  OPEN .COM FILE
0397  CALL BDOS
0398  CPI -1
```

```
0399  CZ CREATE
0400  XRA A
0401  STA OCR
0402 *WRITE
0403  LDA HIGH
0404  DCR A
0405  MOV H,A
0406  MVI L,0FFH
0407  SHLD SIZ  OF THIS WRITE
0408  MVI D,1  STARTING ADDRESS   +RELOC
0409  MVI E,0
0410  LXI H,OBUF  BUFFER ADDRESS
0411  MVI C,BLKSIZ  BUFFER SIZE
0412 NXTW EQU $
0413  LDAX D
0414  MOV M,A
0415  INX H
0416  INX D
0417  DCR C  BUFF COUNT
0418  CZ WRITE
0419  PUSH H
0420  LHLD SIZ
0421  DCX H
0422  SHLD SIZ
0423  MOV A,L
0424  ORA H
0425  POP H
0426  JNZ NXTW
0427  CALL WRITE  LAST BLOCK
0428 * CLOSE
0429  LXI D,OFCB
0430  MVI C,16  CLOSE
0431  CALL BDOS
0432  JMP 0+RELOC
0433 *
0434 WRITE EQU $
0435  PUSH D
0436  LXI D,OFCB
0437  MVI C,21  WRITE
0438  CALL BDOS
0439  POP D
0440  ORA A
0441  JNZ ERRW
0442  LXI H,OBUF
0443  MVI C,BLKSIZ
0444  RET
0445 *
0446 *++************************************************
0447 *$$  DISPLAY A MESSAGE TO THE CONSOLE
0448 *     ENTRY H&L CONTAIN STARTING ADDRESS OF THE MESSAGE
0449 *           THE MESSAGE TEXT IS TERMINATED BY 0 HEX
0450 *           CALL DISPLAY
0451 *--************************************************
0452 *
0453 DISPLAY EQU $
0454  MOV A,M
0455  ORA A
0456  RZ .  EXIT TO CALLING ROUTINE **
```

```
0457  MOV E,A
0458  MVI C,2
0459  PUSH H
0460  CALL BDOS  ;PUT THE CHAR TO THE CONSOLE
0461  POP H
0462  INX H
0463  JMP DISPLAY
0464 *
0465 ERRW EQU $
0466  LXI H,MESGW  WRITE ERROR
0467  CALL DISPLAY
0468  JMP OXT1
0469 *
0470 RERR EQU $
0471  LXI H,MESGR  READ ERROR
0472  CALL DISPLAY
0473 OXT1 EQU $
0474  LXI H,OFCB+1  ;FILE NAME
0475  CALL DISPLAY
0476  JMP 0+RELOC  RETURN TO CP/M
0477 ****************************************************
0478 CONSTANTS EQU $
0479 HIGH DB 0  HIGHEST PAGE USED FOR .COM
0480 ZX DB 0  DEFAULT NO CLEAR ;Z=ZERO FILL BEFORE LOADING
0481 CX DB 0  DEFAULT NO .COM ;C=.COM FILE
0482 LX DB 0  DEFAULT EXECUTE ;L=LOAD BUT NO EXECUTION
0483 ODRIVE DB 0
0484 OWRK DB 0,0,0,0
0485 OCBA DW OBUF  ;CURRENT BUFFER ADDRESS
0486 OCBC DB BLKSIZ  ;CURRENT BUFFER COUNTER
0487 OSIZE DW 0  ;SIZE OF NEXT OBJ BLOCK
0488 SIZ DW 0  SIZE OF COM FILE CODE
0489 MESGO ASC 'OPEN ERROR '
0490  DB 0
0491 MESGR ASC 'READ ERROR '
0492  DB 0
0493 MESGW ASC 'WRITE ERROR '
0494  DB 0
0495  DS 30
0496 STK DB 'S'
0497 LAST DB 0
```

APPENDIX 5

REFERENCES

8080/8085 Assembly Language Programming Manual, Intel Corporation, Santa Clara CA., 1977.

Leventhal, Lance A., 8080A/8085 Assembly Language Programming Adam Osborne & Associates, Berkeley, CA., 1978.

ELLIS COMPUTING, NEVADA COBOL Application Packages Book1, ELLIS COMPUTING, 1980.

ELLIS COMPUTING, NEVADA EDIT, ELLIS COMPUTING, 1982.

ELLIS COMPUTING, NEVADA SORT, ELLIS COMPUTING, 1982.

ELLIS COMPUTING, NEVADA COBOL, ELLIS COMPUTING, 1979.

Starkweather, J., NEVADA PILOT, ELLIS COMPUTING, 1981.

OPERAND FIELD

Most machine instructions and pseudo-operations require one or two operands, either register names, labels, constants, or arithmetic expressions involving labels and constants.

The operands must be separated from the operator by at least one space. If two operands are required, they must be separated by a comma. No spaces may occur within the operand field, since the first space following the operands delimits the comments field.

Register Names

Many 8080 machine instructions require one or two registers or a register pair to be designated in the operand field. The symbolic names for the general-purpose registers are A, B, C, D, E, H and L. SP stands for the stack pointer, while M refers to the memory location whose address is in the HL register pair. The register pairs BC, DE, and HL are designated by the symbolic names B, D, and H, respectively. The A register and condition flags, when operated upon as a register pair, are given the symbolic name PSW.

The values assigned to be register names A, B, C, D, E, H, L, M, PSW and SP are 7, 0, 1, 2, 3, 4, 5, 6, 6, and 6, respectively. These constants, or any label or expression whose value lies in the range 0 to 7, may be used in place of the pre-defined symbolic register names where a register name is required; such a substitution of a value for the pre-defined register name is not recommended, however.

Labels

Any label that is defined elsewhere in the program may be used as an operand. If a label is used where an 8-bit quantity is required (e.g., MVI C,LABEL), its value must lie in the range -256 to 255, or it will be flagged as a value error.

If a label is used as a register name, its value must lie in th range 0 to 7, or be 0, 2, 4, or 6 if it designates a register pair. Otherwise, it will be flagged as a register error.

During each pass, the assembler maintains an instruction location counter that keeps track of the next location at which an instruction may be stored; this is analogous to the program counter used by the processor during program execution to keep track of the location of the next instruction to be fetched.

The special label $ (dollar Sign) stands for the current value of the assembler's instruction location counter. When $ appears within the operand field of a machine instruction, its value is the address of the first byte of the next instruction.

EXAMPLE:

```
FIRST EQU $      The label FIRST is set to the address
TABLE DB ENTRY   of the entry in a table and LAST
*                points to the location immediately after
*                the end of the table.  TABLN is then
*                the length of the table and will remain
LAST EQU $       correct, even if later additions or
TABLN EQU LAST-FIRST  deletions are made in the table.
```

CONSTANTS

Decimal, hexadecimal, octal, binary and ASCII constants may be used as operands.

The base for numeric constants is indicated by a single letter immediately following the number, as follows:

D = decimal
H = hexadecimal
O = octal
Q = octal
B = binary

If the letter is omitted, the number is assumed to be decimal. Q is usually perferred for octal constants, since O is so easily confused with 0 (zero). Numeric constants must begin with a numeric character (0-9) so that they can be distinguished from labels; a hexadecimal constant beginning with A-F must be preceded by a zero.

ASCII constants are one or two characters surrounded by single quotes ('). A single quote within an ASCII constant is represented by two single quotes in a row with no intervening spaces. For example, the expression '''', where the two outer quote marks represent the string itself, i.e., the single quote character. A single character ASCII constant has the numerical value of the corresponding ASCII code. A double character ASCII constant has the 16-bit value whose high-order byte is the ASCII code of the first character and whose low-order byte is the ASCII code of the second character.

If a constant is used where an 8-bit quantity is required (e.g., MVI C,10H), its numeric value must lie in the range