ASSM
8080 Assembler

User's Manual

Processor Technology

ASSM
Advanced 8080 Assembler User's Manual

Describes ASSM, Release 1.Ø.

Processor Technology Corporation

7100 Johnson Industrial Drive
Pleasanton, CA 94566
Telephone (415) 829-2600

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

APPENDICES

SECTION 1

GENERAL INFORMATION

1.1 INTRODUCTION

There are three programs recorded on the cassette tape that this manual accompanies. The first is ASSM, an assembler. 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.

ASSM is designed for use on the Sol Terminal Computer or another computer that uses the CUTER monitor program and CUTS module. The assembler itself occupies almost 8K of memory; additional memory is required for the symbol table, which will be described in the next subsection, and for the user program. Two cassette recorders are also required.

This manual is not intended as an assembly language tutorial. It does not discuss the 8080 instruction set (though there is a summary of the instruction mnemonics in Appendix 1), nor does it offer an exhaustive explanation of what an assembler does. Two books that do describe such material in detail are 8080A/8085 Assembly Language Programming, by Lance A. Leventhal (Adam Osborne & Associates, Berkeley, CA., 1978) and 8080/8085 Assembly Language Programming Manual (Intel Corporation, Santa Clara, CA., 1977). The purpose of the following pages is to enable you to develop programs using the ASSM assembler. The manual does include a complete discussion of the pseudo-operations recognized by the assembler.

The two other programs recorded on the tape are named PACK and UNPAC. These programs convert a cassette file from either of the two SOLOS/CUTER file formats (single-block and multiple-block) to the other. Appendix 4 of this manual describes the programs, and there are some references below to their possible use in relation to ASSM.

1.2 THE ASSEMBLER

This subsection outlines the operation of ASSM; the next subsection will describe the actual procedure for using the program.

INPUT TO THE ASSEMBLER

The assembler operates on a multiple-block source code file created in EDIT and recorded on a cassette in tape unit 1. (Any file that you create in EDIT will be a multiple-block file. The SOLOS/CUTER User's Manual, Second Edition, contains information about file structures.) If the file to be assembled is a single-block file, use the UNPAC program described in Appendix 4 to convert the file to multiple-block structure.

Because the input and processing of the source program is done block by block, there is no absolute restriction on the length of an input file. In theory, you might even use the COPY pseudo-operation (see Section 3) to create a source program that exceeded the length of a cassette tape! The real restriction on the length of a source program is that it can contain only a limited number of symbolic labels (see Section 2.3). For each 1K of memory between the 8K occupied by the ASSM program and the beginning of SOLOS or CUTER, there can be 146 labels in the source program.

Each line of an acceptable source file consists of the characters in that line and a carriage return (ØDH). A file may or may not have 4-digit line numbers in character positions 1-4 of each line; it is not permissible for only some of the lines to have line numbers.

OPERATION OF THE ASSEMBLER

Step numbers in parentheses following an explanation refer to the accompanying figure.

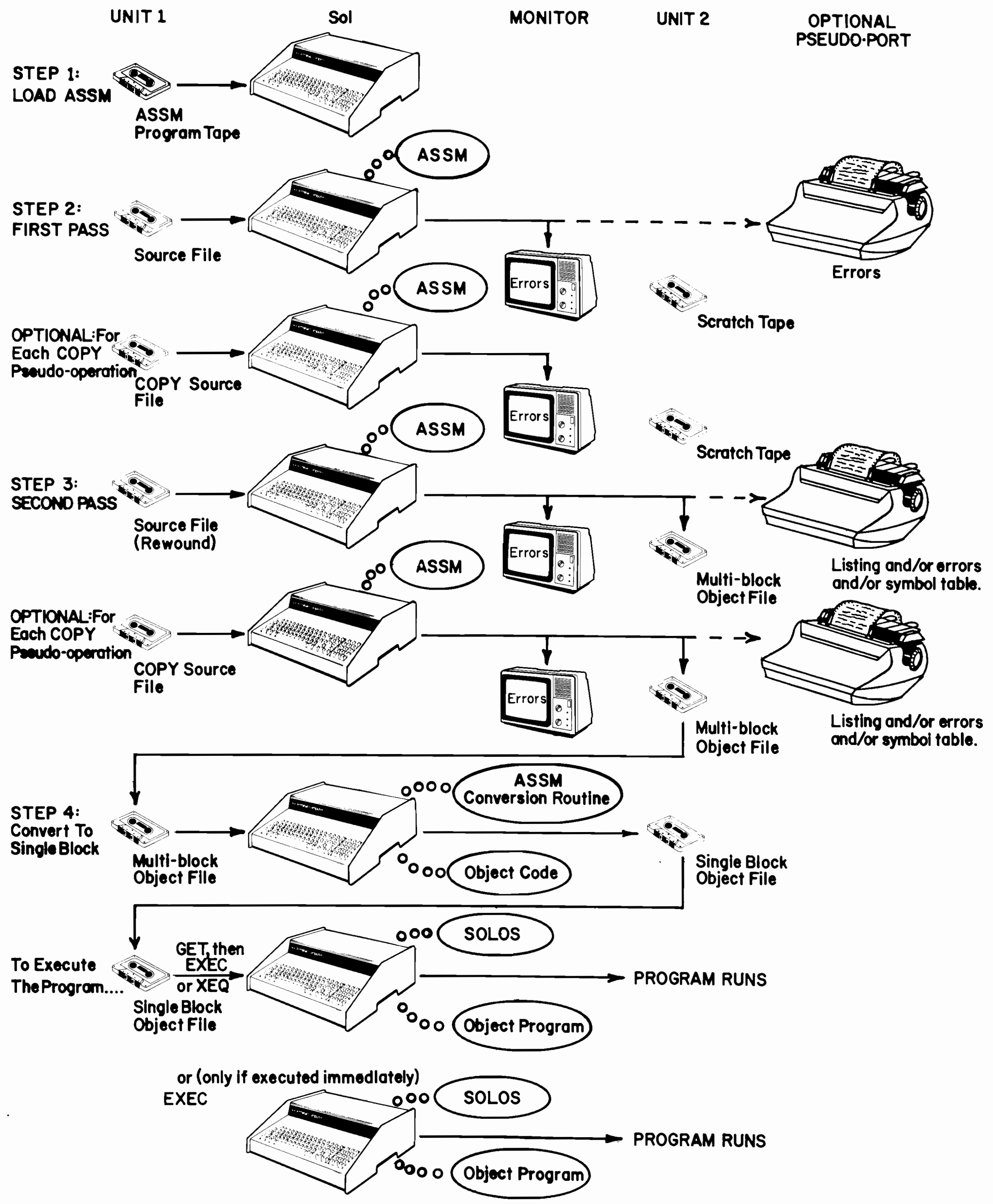
The assembler is loaded from tape unit 1 into memory starting at location Ø (step 1). It processes the source code file in two passes; although nothing is written to tape unit 2 until the second pass, you will be asked to set up both an input tape, containing a source code file, and an output tape, to contain the temporary object file, before the first pass.

On the first pass (step 2), ASSM reads the source program from tape unit 1 and builds a symbol table containing all of the labels defined in the source program. (See Section 2.3.) 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 the video display. The user may also specify that errors be sent to the current pseudo-port. For information regarding pseudo-ports, consult the SOLOS/CUTER User's Manual.

If the source program is recorded on more than one cassette tape, each additional tape must read from tape unit 1 (optional step, applying only when the source contains the COPY pseudo-operation described in Section 3).

On the second pass (step 3), ASSM again reads the source program from tape unit 1 (you will have been told to rewind the input tape), generates the object code and usually writes it out as a temporary multiple-block file on the scratch tape in unit 2. This object file is temporary because only a single-block program file can be executed conveniently in SOLOS/CUTER. In addition, a formatted listing of source and object code, errors, symbol table, or any combination of these, may be output to the current pseudo-port. Errors detected during the second pass are always output to the video display and may also be sent to the current pseudo-port.

USING ASSM



Again, if the source is recorded on more than one cassette tape, each additional tape must be read from tape unit 1 (optional step, applying only when the source contains the COPY pseudo-operation described in Section 3).

Finally, the assembler must make one pass over the object code file (step 4). For this purpose, you will be asked to insert the cassette that contains the temporary file in tape unit 1, and the cassette that will contain the final output file in tape unit 2. ASSM will load the multiple-block object file as though it were going to execute the program: that is, each byte is loaded at the address that was assigned to it when the object code was generated. (Thus, memory must be available at the same locations that the program is intended to occupy when it runs.) The object code is then written to tape unit 2 in the single-block structure that is required if the program is to be loaded and executed within SOLOS/CUTER.

OUTPUT FROM THE ASSEMBLER

If the assembly runs to completion and no errors are detected, the resulting object code file is a file that you can execute using the SOLOS/CUTER XEQ command (if the source code contained an XEQ pseudo-operation - see Section 3). To load the file without executing it, or to execute a file that does not contain an XEQ pseudo-operation, use the SOLOS/CUTER GET and EXEC commands. It is also possible to execute the program from memory, because it is loaded at the appropriate address during the last stage of assembly; give a SOLOS/CUTER EXEC command that specifies the starting address for the program.

1.3 PROCEDURE FOR USING ASSM

While still in SOLOS/CUTER, decide whether you want to send output to another pseudo-port in addition to the screen. If you do, use the SOLOS/CUTER SET command (see your manual) before loading ASSM.

To load ASSM from cassette into memory, place the cassette containing the program in tape unit 1, and put the recorder in PLAY mode. (If you are not familiar with the operation of cassette recorders, you might want to look at Appendix 5, entitled "About Cassette Recorders and Cassette Files." If you are not sure how to connect your recorder to the computer, look at Section 7 of your Sol Manual.)

If you want to load ASSM and execute it immediately, type the SOLOS/CUTER command XEQ ASSM and then press RETURN; if you want to load but not execute the program, type GET ASSM and press RETURN. Once the program is loaded with the GET command, it can be executed with the command EXEC Ø. The loading process can be aborted with MODE SELECT or by pressing the CTRL and @ keys simultaneously.

After the assembler begins to execute, it will display the following series of instructions and questions. At any time during this interactive portion of ASSM, the ESCape key will cause the program to start over again with the first instruction.

```
CASSM 1.Ø Cassette Assembler
Copyright (C) 1978, Processor Technology Corporation

Enter source file name:
```

The name entered at this point must be the one to five character file name of the multiple-block source program file. If you plan to use the COPY pseudo-operation to assemble several files, give the name of the first file to be assembled. The DELete key can be used to delete an erroneous character at any time before the carriage return is entered.

```
Enter object file name (optional):
```

If the assembler is to generate and write an object code file, the one to five character file name that will be written to the file header must be specified at this point. If no file name is entered before the carriage return terminates the response, the assembler will not generate an object file. The DELete key can be used to delete an erroneous character at any point before the carriage return is entered.

```
Does source file have line numbers? (Y/N)
```

As will be seen in Section 2, the column in which a statement begins is significant to the assembler. For this reason it is necessary to specify whether or not each line of the source file begins with a line number. Type Y for yes and N for no. A carriage return is not required to terminate this response.

```
Should errors, listing, and/or symbol table
  be output to the current pseudo-port? (E/L/S):
```

Specify one or more of these options by typing the appropriate letter(s). To select more than one option, type the two or three letters consecutively without an intervening blank, e.g., EL for errors and listing. (Consult your SOLOS/CUTER User's Manual for information concerning pseudo-ports.) If no options are specified, lines that contain errors will still appear on the video display, but no output will go to the current pseudo-port (unless the pseudo-port coincides with the display). Enter a carriage return to terminate the response.

```
Should printed output be Paginated and/or have
  new line numbers added? (P/N):
```

If the response to the last question determined that no output will be sent to the current pseudo-port, this question is not asked. If the P option is selected, output to the current pseudo-port will be paginated. The name of the source program file will be printed on the top left-hand corner of each page, and a page number will be printed on the top right-hand corner. A page length of 66 lines will be assumed. If the TITL pseudo-operation occurs in the source program, the title provided in the instruction will be centered at the top of each page. If the N option is selected, new line numbers will be assigned to the source code. To select both options, type PN. Enter a carriage return to terminate the response.

```
For what width should printed output be formatted;
 no formatting, 72, 80, 132?  (0/1/2/3):
```

If formatting is requested, additional spaces will be inserted between fields in the output to fit the width of the paper or output device.

0	for no additional spacing
1	for 72- column paper or output device
2	for 80- column paper or output device
3	for 132- column paper or output device

Enter a carriage return to terminate the response.

```
Set up tapes.  (Hit return when ready)
```

Place the tape containing the source program file in tape unit 1, position the tape a little before the file, and put the recorder in PLAY mode. Place a scratch tape in tape unit 2 and put the recorder in RECORD mode. Remember that the scratch tape will receive the temporary multiple-block object file on the second pass. When a carriage return is entered to terminate this response, ASSM will begin the first pass.

A MODE SELECT or CTRL-@ can be used to abort the assembly at any time after this point. If assembly is aborted by this means, ASSM will have to be reloaded from cassette in order to be executed again.

If a COPY pseudo-operation is encountered, ASSM will request that the appropriate cassette be put in tape unit 1 (see Section 3).

At the end of the first pass, ASSM reports:

```
Done with pass 1, all input tapes need to be rewound.
Rewind tape unit 1.  (Hit return when rewound)
```

Power is applied to tape unit 1 so that the tape(s) containing the source file(s) can be rewound. When this step has been completed, place the original source tape in tape unit 1, and put the recorder in PLAY mode. When the carriage return is entered, ASSM will begin the second pass. If the source program contains a COPY pseudo-operation, ASSM will request that the appropriate cassette be inserted in tape unit 1. Any output that is being sent to the current pseudo-port will be sent during the second pass. If an object file is being generated, ASSM will give this instruction after the second pass:

```
Place scratch tape in tape unit one, and final output tape
in tape unit two.  (Hit return when done)
```

Place the tape containing the temporary multiple-block object file in tape unit 1, and the tape to receive the final single-block object file in tape unit 2. When the carriage return is entered, ASSM will make the final pass on the object file. When this pass is complete, control will return to SOLOS/CUTER.

SECTION 2

STATEMENTS

2.1 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.

2.2 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, but they may make it easier to locate lines in the source code file when it is being edited. The tape 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 field may 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 the COPY pseudo-operation) must conform to the format of the main file.

Example of source statements with line numbers:

```
column
1234567
-------
0000 LABEL ORA A   Label field must start at column 6.
0001  JNZ NEXT     Operation field starts at column 7 (minimum).
0002 LOOP MOV A,B  Operation field starts one space after 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.
```

2.3 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 operation field by a colon (:) instead 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 (see Section 2.5.1). 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; see Section 2.6.

2.4 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. More information on the 8080 machine instructions can be found in the two books mentioned in Section 1.1, above. Assembler pseudo-operations are directives that control various aspects of the assembly process, such as storage allocation, conditional assembly, file inclusion, and listing control. The pseudo-operations are described in Section 3.

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).

2.5 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 comment field.

2.5.1 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 the 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.

2.5.2 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 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.

During each pass over the source code, 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 first 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.
```

2.5.3 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 preferred for octal constants, since O is so easily confused with Ø (zero). Numeric constants must begin with a numeric character (Ø-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 ASCII value of a single quote mark (') is represented by the expression '''', where the two outer quote marks are the delimiters of the ASCII string, and the two inner 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. (Appendix 2 contains a list of ASCII codes.) 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,1ØH), 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 Ø to 7, or be Ø, 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,1ØD         Move 1Ø decimal to register C.
        LXI H,2FH         Move 2F hexadecimal to register pair HL.
        MVI B,3Ø3Q        Move 3Ø3 octal to register B.
        MVI A,'Y'         Move the ASCII value for Y to register A.
        MVI A,1Ø1B        Move 1Ø1 binary to register A.
        JMP ØFFH          Jump to address FF hexadecimal.
```

2.5.4 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=8ØØØH=-32768 and -32768-1=8ØØØH-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+1ØH), it must evaluate to a value in the range -256 to 255, or it will be flagged as a value error.

An expression used as a register name must evaluate to a value in the range Ø to 7, or to Ø, 2, 4, or 6 if it designates a register pair. Otherwise, it will be flagged as a register error.

Examples:

```
        MVI              A,255D/1ØH-5
        LDA              POTTS/256*OFFSET
        LXI              SP,3Ø*2+STACK
```

2.5.5 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,<ØCCØØH     Loads register B with the most significant
                          byte of the 16-bit value CCØØH, i.e., CCH.
        MVI C,<1234H      Loads register C with the value 12H.
        MVI C,>1234H      Loads register C with the value 34H.
```

2.6 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:

```
LOOP IN  STAT INPUT DEVICE STATUS
     ANI 1    TEST STATUS BIT
     JZ  LOOP WAIT FOR DATA
*DATA IS NOW AVAILABLE
```

If listing file formatting is requested in answer to the question, "For what width should printed output be formatted...?" the comment field must be preceded by at least two spaces. Furthermore, instructions and pseudo-operations requiring no operand must be followed by a dummy operand (a period is recommended).

Examples:

```
MVI A,10  COMMENT
RZ .  COMMENT
```

SECTION 3

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 ASSM.

In the following pseudo-operation formats, <expression> stands for a constant, label, or 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  Label BELL2 is set 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.

During the pass that converts the multiple-block object code file into a single-block file of the kind that SOLOS/CUTER can load, the object code is actually loaded at the origin given in the program, or at the default origin. The routine that performs the conversion runs in the SOLOS/CUTER stack area: that is, in the 1K of memory beginning at address C800H for SOLOS, or at 800H above the first address assigned to CUTER. Thus, the program to be assembled must NOT be given an origin in the SOLOS/CUTER stack area.

Examples:

ORG 64	Determines that the object code generated by subsequent statements must be loaded in locations beginning at 64 (40H).
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 no start address; if its name is typed in a SOLOS/CUTER XEQ command, it will be loaded but not executed, and an error message will be displayed. (It is, however, possible to execute such a file using the SOLOS/CUTER GET and EXEC commands, or to call it from another program.) 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	Reserve one byte.
DS 400	Reserve 400 bytes.
RES 177Q	Reserve 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 are defined.
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 occurrence of the same delimiter, or at a carriage return.

The ASCZ pseudo-operation is the same except that it appends a NUL (ØØH) 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 Ø
ASCF 1

If the operand field contains a Ø, 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 arguments 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 Ø (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.

```
YES EQU 1        Sets the value of the label 'YES' to 1.
NO EQU Ø         Sets the value of the label 'NO' to Ø.
*
 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 (Ø), so the code
 MVI A,'N'        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 <file name>[/<unit>]

This pseudo-operation copies source code from a tape file into a program being assembled. The code from the copied file will be assembled starting at the current address.

The resulting object code will be exactly like what would be generated if the copied source code were appended to the original file, but the COPY pseudo-operation does not actually alter any source file.

If the COPY pseudo-operation is used, it must be the last instruction in the source file in which it appears. When the assembler encounters the COPY during each pass on the source file, it will ask you to

```
     "Insert <name specified in operand field> into tape
       unit 1.  (Hit return when done)"
```

The original source tape should be removed from tape unit 1, and the tape containing the requested file should be inserted. Put the recorder in PLAY mode.

A copied file may in turn copy another. Note that one file must not copy another which in turn copies the original file--the assembly will generate duplicate label errors and assemble the same code over and over until the object code file overflows the tape on which it is being written.

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.

Listing Control NLST
LST

The NLST pseudo-operation suppresses output of the listing to the current pseudo-port. If object code is being generated, it will still be output to the object code file; lines containing errors will still be output to the video display (and to the current pseudo-port, if the E option was selected). The LST pseudo-operation re-enables output of the listing to the current pseudo-port.

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

If the P option is specified in answer to the question, "Should printed output be Paginated ...," the one- or two-line title specified by this pseudo-operation will be centered at the top of each page of the listing.

Page Eject PAGE

If the P option is specified in answer to the question, "Should printed output be Paginated ...," 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 source code input file will also terminate each pass, the END statement is unnecessary in most cases.

SECTION 4

ERROR MESSAGES

4.1 CONSOLE ERROR MESSAGES

A number of console messages may be generated in response to errors in reading or assembly of a source file. Any of the following errors will cause assembly to be aborted and control to return to SOLOS/CUTER.

Read Error- Bad tape file

This message usually indicates that a tape has been recorded incorrectly, or that MODE SELECT or CTRL-@ has been entered during the reading of a tape file.

Too many characters in name of COPY statement: <name>

The name displayed after the colon appears in the operand field of a COPY statement in the source program and has more than five characters in it. A file name that has more than five characters is not valid within SOLOS/CUTER.

Error- COPY statement cannot be other than at EOF.

A source file contains a COPY statement that is not the last statement of the file.

Error- Symbol table overflow.

The source program contains more labels than will fit in the amount of memory available. Remember that the symbol table requires seven bytes per symbol and begins immediately after the code for ASSM.

The other message that you might encounter is CHECKSUM TEST FAILED. This error would occur during an attempt to load ASSM, PACK, or UNPAC. The Appendix entitled "About Cassette Recorders and Cassette Files" explains what the message means and why it is important.

4.2 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 current pseudo-port. An error detected during both the first and the second pass of the assembler will be flagged twice in the listing. 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 8Ø8Ø 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	Adr
CA	JZ	Adr
D2	JNC	Adr
DA	JC	Adr
E2	JPO	Adr
EA	JPE	Adr
F2	JP	Adr
FA	JM	Adr
E9	PCHL	

CALL

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

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 IMMEDIATE

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

Acc IMMEDIATE*

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

LOAD IMMEDIATE

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

DOUBLE ADD†

09	DAD	B
19	DAD	D
29	DAD	H
39	DAD	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	

SPECIALS

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

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

INPUT/OUTPUT

D3	OUT	D8
DB	IN	D8

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

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

PSEUDO INSTRUCTION

ORG	Adr
END	
EQU	D16
DS	D16
DB	D8 | |
DW	D16 | |

CONSTANT DEFINITION

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

OPERATORS

+, -

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

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		

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

HEX-ASCII TABLE

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			

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

APPENDIX I

APPENDIX 2
TABLE OF ASCII CODES (Zero Parity)

Paper tape 123 . 4567P	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 123 . 4567P	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	[	shift K
`  ●.●● ● `	0560	134	92	5C	\	shift L
`● ●.●● ● `	0564	135	93	5D	]	shift M
` ●●.●● ● `	0570	136	94	5E	^	shift N
`●●●.●● ● `	0574	137	95	5F	_	shift O
`   .  ●● `	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	}	Alt Mode
` ●●.●●●● `	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
```

ASSEMBLER LISTING (Cont'd)

SYMBOL TABLE LISTING

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

CROSS REFERENCE LISTING

(Printed in place of Symbol Table Listing if X option is specified.)

```
Label Addr. References
 CTABL  0135   0092
 TS1    0102   0131
 TS2    010F   0192
 TS3    0111   0239
 TS4    0115   0307
 TSRCH  0100   0367 0374
```

APPENDIX 4

PACK AND UNPAC

In Section 5.2.7 of the SOLOS/CUTER User's Manual, there is a discussion of the two types of files generated and utilized by SOLOS/CUTER. It is important to remember that any given item of software may be able to handle only one of these file structures, and that a program that handles multiple-block files may require that the blocks be of a particular size or in a particular range of sizes. The PACK and UNPAC programs recorded on the same cassette as the software you have purchased convert files of either type into the opposite type; PACK converts a multiple-block file to single-block format, and UNPAC converts a single-block file to multiple-block format.

The following chart is a summary of the file requirements of cassette software. SB stands for single-block, and MB stands for multiple-block. If MB is followed by a slash and a number, that number indicates the block size required or generated by the program.

ITEM	INPUT FILE	OUTPUT FILE
ALS-8*	SB	SB
ASSM	MB/ less than or equal to 1Ø24	No text output file
Ext. BASIC	MB/ 256	MB/ 256
EDIT	MB/ 256 to 1Ø24	MB/ 256 to 1Ø24
SOFTWARE #1	SB	SB

* uses SOLOS/CUTER SAVE and GET commands

If you want to use the output file from one of these programs as the input file for another, and if the file does not have the structure required by the second program, you will need to use PACK or UNPAC to create a file of the correct structure. (Actually, the original file will not be altered; each of these programs reads a file from tape unit 1 and RECORDS A DIFFERENTLY FORMATTED VERSION of the same file on tape unit 2.)

PACK and UNPAC can be used to convert a multiple-block file having one block size into a multiple-block file with another block size. First PACK the file, and then UNPAC it, specifying the block size that you want. (If you have the EDIT program, you will probably find it more convenient to use the n; command than to execute PACK, then UNPAC, for this purpose.)

PACK and UNPAC have only one error message:

Read error - Bad tape file.

This message is displayed during the reading of a file from unit 1; it usually indicates that the file was recorded incorrectly, or that a MODE SELect or CTRL-@ was entered from the keyboard while the tape was being read.

PACK

PACK reads a multiple-block text file from tape unit 1 and writes it out as a single-block file on tape unit 2. When you execute PACK, either by typing XEQ PACK<CR> or the sequence GET PACK<CR> followed by EX 0<CR>, the screen will display:

```
Multi- to single- block converter

Enter name of multi-block file:
```

Type the name of the multiple-block file that you want to PACK. PACK will also give this name to the single-block file that it creates. A file name should contain one to five characters, no blanks or slashes; when you have typed the whole name, hit the carriage return key. If you make a typing error BEFORE YOU HIT THE RETURN KEY, use DELete to erase the last character typed. If you hit the return and then discover a typing error in the file name, you can use the ESCape key to restart the series of questions.

```
Should output file be in ALS-8 format? (Y/N):
```

If the file is intended for use in the ALS-8 system, answer Y; otherwise, answer N. In ALS-8 format each line of a file begins with a count of the number of bytes in that line. Note that the input file for PACK will never be in ALS-8 format, because ALS-8 format is not used for multiple-block files.

```
Set up tapes. (Hit return when ready)
```

Insert the tape containing the multiple-block file in tape unit 1, and put the recorder in PLAY mode. Insert the tape that will contain the single-block file in tape unit 2, and put that recorder in RECORD mode. When you hit the return key, the program will begin reading from tape unit 1 and writing to tape unit 2.

At any time before you hit this final carriage return, you can restart the question-answer section of PACK by hitting the ESCape key. Once you have given the carriage return, however, the only way to interrupt the activity of the program is to use MODE SELECT or CTRL-@ to abort it and return to SOLOS/CUTER. If you abort the program and want to execute it again, you will have to reload it from cassette.

UNPAC

UNPAC reads a single-block text file from tape unit 1 and writes it out as a multiple-block file on tape unit 2. When you execute UNPAC by typing either XEQ UNPAC<CR> or the sequence GET UNPAC<CR> followed by EX 0, the screen will display:

```
Single- to multi- block converter

Enter name of single-block file:
```

Enter the name of the file that you want to UNPAC. The same name will be given to the multiple-block output file. The file name specifications are the same as those for PACK, and the DELete and ESCape keys serve the same purpose as in PACK (see above).

```
Is input file in ALS-8 format? (Y/N):
```

If the input file is in ALS-8 format, answer Y; otherwise, answer N. The output file will not be in ALS-8 format. (The ALS-8 byte counts will be stripped from the input file before it is written out to the output file.)

```
Enter desired block size for output file:
```

Enter the block size (in number of bytes per block) as a decimal number, and then type a carriage return.

```
Set up tapes. (Hit return when ready)
```

See the explanation under PACK, above. The ESCape and MODE keys function as they do with PACK; like PACK, UNPAC must be reloaded from cassette in order to be executed after MODE SELECT or Control-@ has been used.

APPENDIX 5

ABOUT CASSETTE RECORDERS AND CASSETTE FILES

Successful and reliable results with cassette recorders and cassette files require a good deal of care. You need to use consistent and careful methods, and you need to know what to expect, when you try to read a manufacturer's tape, or your own. The following methods are recommended:

1) Use only a recorder recommended for digital usage. For use with the Processor Technology Sol or CUTS, the Panasonic RQ-413AS or Realistic CTR-21 is recommended.

2) Keep the recorder at least a foot away from equipment containing power transformers or other equipment which might generate magnetic fields, picked up by the recorder as hum.

3) Keep the tape heads cleaned and demagnetized in accordance with the manufacturer's instructions.

4) Use high quality brand-name tape, preferably low noise, high output tape. Poor tape can give poor results, and rapidly wear down a recorder's tape heads.

5) Bulk erase tapes before reusing. It can be hard to find the file you want in a jumble of old file pieces. Bulk erasing also decreases the noise level of the tape.

6) Keep cassettes in their protective plastic covers, in a cool place, when not in use. Cassettes are vulnerable to dirt, high temperature, liquids, and physical abuse.

7) Experimentally determine the most reliable settings for volume and tone controls, and use these settings only.

8) On some cassette recorders, the microphone can be live while recording through the AUX input. Deactivate the mike in accordance with the manufacturer's instructions. In some cases this can be done by inserting a dummy plug into the microphone jack.

9) If you record more than one file on a side, SAVE an empty file, named "END" for example, after the last file of interest. Once you read its name, you will know not to search beyond it for files you are seeking. One way to avoid having to search for files is to record only one file per cassette, at the beginning of the tape, if you can afford the extra cassettes.

10) Do not record on the first or last minute of tape on a side. The tape at the ends gets the most physical abuse. Do not be impatient when trying to read the first file on a tape. You, or the manufacturer of a pre-recorded program, may have recorded a lot of empty tape at the beginning.

11) Record a file more than once, before it leaves memory. This redundancy can protect you from bad tape, equipment malfunction, and accidental erasure.

12) Most cassette recorders have a feature that allows you to protect a cassette from accidental erasure. On the edge of the cassette opposite the exposed tape are two small cavities covered by plastic tabs, one at each end of the cassette. If one of the tabs is broken out, then one side of the cassette is "write protected." An interlock in the recorder will not allow you to press the record button. A piece of tape over the cavity will remove this protection.

13) Use the tape counter to keep track of the position of files on the cassette. Always rewind the cassette and set the counter to zero when first putting a cassette into the recorder. Time the first 30 seconds and note the reading of the counter. Always begin recording after this count on all cassettes. Record the beginning and ending count of each file for later reference. Before recording a new file after other files, advance a few counts beyond the end of the last file to insure that it will not be written over.

14) The SOLOS/CUTER command CATalog can be used to generate a list of all files on a cassette. In SOLOS/CUTER, type CAT <CR>, rewind to the beginning of the tape, and press PLAY on the recorder. As the header of each file is read, information will be displayed on the screen. If you have recorded the empty file called END, as suggested, you will know when to search no further. If you write down the the catalog information along with the tape counter readings and a brief description of the file, you will be able to locate any file quickly.

15) Before beginning work after any modification to the system, test by SAVEing and GETting a short test program. This could prevent the loss of much work.

In addition to using the above procedures methodically, you need to know the various ways in which programs may be recorded on tapes you have purchased:

1) If you cannot read a file consistently, and suspect the tape itself, do not despair. The same file may have been recorded elsewhere on the tape. Processor Technology often records a second version, later on the same side of the tape. When you first get a tape, CATalog it with SOLOS or CUTER so you will know exactly what it contains. Write down the tape counter readings at the same time.

2) An empty file named END is sometimes placed at the end of the recorded portion of a tape. When SOLOS CATalogs a file, the file header information is displayed as soon as the beginning part of the file passes the tape head, but nothing is displayed when the end of the program passes by. If another filename such as END is displayed, you know you have just passed the end of the previous file.

3) Some of the programs supplied by Processor Technology contain a checksum test within their code, in addition to the checksum test which SOLOS performs. When a program containing this test is first executed after loading, the checksum test reads all of the program in memory, and calculates a checksum number which is compared with a correct value. If the numbers match, the program in memory is correct. Nothing is displayed when the numbers match, but if they do not match, the message CHECKSUM TEST FAILED, or a similar message, is displayed. The message may be followed by two numbers, representing the correct and incorrect checksum numbers.

Even though the checksum test was failed, it may be possible to enter the program anyway by typing the carriage return key. The bad data may not even be apparent, if it is in a portion of the program you do not use. It is best, however, to try to find and correct the problem causing the error so the checksum test is passed. The error can be caused by the cassette interface circuitry, bad memory locations, bad tape, a faulty recording, improper alignment or settings on the cassette recorder, or other equipment problems.

Processor Technology Corporation 7100 Johnson Industrial Way, Pleasanton, CA 94566

Manual Number 727121