ASSM
Advanced 8080 Assembler
User’s Manual

Describes ASSM, Release 1.0.

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

Copyright (C) 1978, Processor Technology Corporation
Manual Part No. 727120
All rights reserved.
<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-1</td>
</tr>
<tr>
<td>1.1</td>
<td>1-1</td>
</tr>
<tr>
<td>1.2</td>
<td>1-1</td>
</tr>
<tr>
<td>1.3</td>
<td>1-4</td>
</tr>
<tr>
<td>2</td>
<td>2-1</td>
</tr>
<tr>
<td>2.1</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2</td>
<td>2-1</td>
</tr>
<tr>
<td>2.3</td>
<td>2-2</td>
</tr>
<tr>
<td>2.4</td>
<td>2-2</td>
</tr>
<tr>
<td>2.5</td>
<td>2-2</td>
</tr>
<tr>
<td>2.5.1</td>
<td>2-3</td>
</tr>
<tr>
<td>2.5.2</td>
<td>2-3</td>
</tr>
<tr>
<td>2.5.3</td>
<td>2-3</td>
</tr>
<tr>
<td>2.5.4</td>
<td>2-4</td>
</tr>
<tr>
<td>2.5.5</td>
<td>2-5</td>
</tr>
<tr>
<td>2.6</td>
<td>2-5</td>
</tr>
<tr>
<td>3</td>
<td>3-1</td>
</tr>
<tr>
<td>4</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1</td>
<td>4-1</td>
</tr>
<tr>
<td>4.2</td>
<td>4-2</td>
</tr>
</tbody>
</table>
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 2.1). 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 (0D0A). 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 0 (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.

\[ C \text{ ASSM} \]
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, but no machine code is loaded at this 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 located 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 0. 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 1.0 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 DELE 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 DELE 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, listings, 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'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 TITLE 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.

6 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-6 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 COPI 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 rewind)

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 SGLOS/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, BRIDGE 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, 0, 6 and 7, 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., VDI, 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 $  
    TABLE DB ENTRY  
   **  
    LAST EQU $  
    TABLN EQU LAST-FIRST  

The label FIRST is set to the address 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 correct, even if later additions or 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. O is usually preferred for octal constants, since O is so easily confused with Ø (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 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,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 8 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,18D  Move 18 decimal to register C.
LXI H,2FH  Move 2F hexadecimal to register pair HL.
MVI B,38Q  Move 38 octal to register B.
MVI A,'Y'  Move the ASCII value for Y to register A.
JMP 0FFH  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
/ 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=8000H-32768 and -32768+1=8000H+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.

An expression used as a register name must evaluate to a value in the range 0 to 7, or to 0, 2, 4, or 6 if it designates a register pair. Otherwise, it will be flagged as a register error.

Examples:

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

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,<CC00H  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.

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:

```plaintext
LOOP IN STA INPUT DEVICE STATUS
AN1 TEST STATUS BIT
J2 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,16 COMMENT
RZ . COMMENT
```

SECTION 3

PESEUDO-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:

```plaintext
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 0000H for SOLOS, or at 8000H 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:

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
DB OFH,30H,10H,11H,10H,11H,3,2,7,10
TABLE DB 'A','B','C','D',0
```

One byte is defined.
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
YES DW 'OK'
```

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

The ASCII 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 (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 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 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.

YES EQU 1
NO EQU 0

* IF YES
  MVI A,'Y'
* IF NO
  MVI A,'N'
ENDF
ENDF

Sets the value of the label 'YES' to 1.
Sets the value of the label 'NO' to 0.
The expression here is true (1), so the code on this line will be assembled.
The expression here is false (0), so the code on this line will not be assembled.
This terminates the NO conditional.
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.
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 UNFAC. The Appendix entitled "About Cassette Recorders and Cassette Files" explains what the message means and why it is important.
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., ZL, 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.

---

**APPENDIX I**

- **ACCUMULATOR**
  - A (Accumulator) A,
  - B (Accumulator) B

- **OPERATORS**
  - ADD
  - SUB
  - MUL
  - DIV

- **CONDITIONAL INSTRUCTIONS**
  - JZ (Jump if Zero) if Z
  - JS (Jump if Sign) if S
  - JP (Jump if Positive) if P

- **RESTART INSTRUCTIONS**
  - RST

- **INPUT/OUTPUT**
  - DI (Input) DI
  - DI (Output) DO

- **MOVEL/LOAD**
  - MOV
  - LDA

- **STORAGE**
  - STA
  - DDA

- **DECREMENT**
  - DEC
  - INCR

- **INCREMENT**
  - INC
  - DCR

- **JUMP**
  - JP
  - JS
  - JZ

- **CALL**
  - CALL

- **RETURN**
  - RET

- **MOVE IMMEDIATE**
  - MOV
  - LDA

- **STACK**
  - PUSH
  - POP

- **Specials**
  - EBD, XCHG, CMPS, ESC, etc.

- **Addressing Modes**
  - Direct, Immediate, Register, Indirect, etc.

- **Disassembly**
  - Discrete code of logical-numeric expression that evaluates D, I, and C, and Carry, Condition, etc.

- **Memory**
  - Bytes, words, etc., at PFS, CS, SP, etc.
APPENDIX 2

TABLE OF ASCII CODES (Cont'd) (Zero Panify)

<table>
<thead>
<tr>
<th>Paper tape</th>
<th>Upper</th>
<th>Octal</th>
<th>Decimal</th>
<th>Hex</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>123 456 7 8</td>
<td>0000</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>a</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0001</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>b</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0002</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>c</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0003</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>d</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0004</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>e</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0005</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>f</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0006</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>g</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0007</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>h</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0008</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>i</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0009</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>j</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>000A</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>k</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>000B</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>l</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>000C</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>m</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>000D</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>n</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>000E</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>o</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>000F</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>p</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0010</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>q</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0011</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>r</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0012</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>s</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0013</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>t</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0014</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>u</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0015</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>v</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0016</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>w</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0017</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>x</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0018</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>y</td>
</tr>
<tr>
<td>123 456 7 8</td>
<td>0019</td>
<td>0010</td>
<td>0012</td>
<td>0013</td>
<td>z</td>
</tr>
</tbody>
</table>

APPENDIX 3

ASSEMBLER LISTING

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>MACH CODE</th>
<th>ERROR FND</th>
<th>LABEL</th>
<th>OPERATION</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example of TOXCH use:

0013

APPENDIX 2

ASM

A2-2

A3-1

C ASSM
APPENDIX 3
ASSEMBLER LISTING (Cont'd)

SYMBOL TABLE LISTING

<table>
<thead>
<tr>
<th>Label Addr</th>
<th>Label Addr</th>
<th>Label Addr</th>
<th>Label Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT46</td>
<td>0135</td>
<td>T31</td>
<td>0112</td>
</tr>
<tr>
<td>T52</td>
<td>0105</td>
<td>T50</td>
<td>010F</td>
</tr>
<tr>
<td>T53</td>
<td>0111</td>
<td>T26</td>
<td>0100</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Label Addr</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT46</td>
<td>0135</td>
</tr>
<tr>
<td>T31</td>
<td>0112</td>
</tr>
<tr>
<td>T50</td>
<td>010F</td>
</tr>
<tr>
<td>T53</td>
<td>0111</td>
</tr>
<tr>
<td>T26</td>
<td>0100</td>
</tr>
</tbody>
</table>

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 record 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.

<table>
<thead>
<tr>
<th>ITEM</th>
<th>INPUT FILE</th>
<th>OUTPUT FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALS-8*</td>
<td>SB</td>
<td>SB</td>
</tr>
<tr>
<td>ASSM</td>
<td>MB/ less than or equal to 1024</td>
<td>No text output</td>
</tr>
<tr>
<td>Ext. BASIC</td>
<td>MB/ 256</td>
<td>MB/ 256</td>
</tr>
<tr>
<td>EDIT</td>
<td>MB/ 256 to 1024</td>
<td>MB/ 256 to 1024</td>
</tr>
<tr>
<td>SOFTWARE #1</td>
<td>SB</td>
<td>SB</td>
</tr>
</tbody>
</table>

* 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.)

A3-2

A4-1
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-Ø 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 Ø<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-Ø to abort it and return to SGLOS/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 Ø<CR>, 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-Ø 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 501 or CUTS, the Panasonic KG-413AS or Realistic CT-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. Reactivate 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 "Empty" 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 each file on the tape is read, information will be displayed on the screen. If you have recorded the empty file called END as suggested, you will know whn 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 SAVING 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 CATalog 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.