Software #1
Resident 8080 Assembler
User Manual

Processor Technology Corporation

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

Second Edition, First Printing, June, 1978
Manual Part No. 727023

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

2.1 GENERAL DESCRIPTION

The Processor Technology Software Package #1 is a self-contained program development system for the Sol Terminal Computer, or other S-100 computers based on the 8080 microprocessor using the CUTER monitor programs and CUTS module. Included in the package are an executive to handle memory files, an assembler, and a line-oriented editor.

Although this manual includes a cursory discussion of some 8060 pseudo-operations, it is not intended as an assembly language tutorial, nor does it offer an exhaustive explanation of what an assembler does. Two books which do describe such material in detail are 8080/8085 Assembly Language Programming, by Lance A. Leventhal (Adam Osborne & Associates, Berkeley, CA., 1978) and 8080/8085 Assembly Language Manual (Intel Corporation, Santa Clara, CA., 1977). The purpose of the following pages is to enable you to develop programs using the Software #1 program development system.

At least 4K of memory must be available for use by the system. This memory is allocated as follows:

0000 - 0D9F	Executive Program
0D50 - 0F5E	Special System RAM
0F60	Start of Symbol Table (Assembler Only)

Additional memory must be available for your own source, object, and text files, and for the symbol table, if assemblies are to be done. There will be more discussion of memory requirements in Sections 2.3 and 4.1, below.

2.2 LOADING SOFTWARE #1

Software Package #1 may be loaded and run from cassette using the SOLOS/CUTER command XEQ followed by a blank, the name SOFT1, and a carriage return. To load the program without running it, use GET followed by a blank, the name SOFT1, and a carriage return. To execute the program after using GET, type EXEC 0 followed by a carriage return. To exit to SOLOS/CUTER from the executive of Software Package #1, use the CUST command, unless you have defined this command to perform some other function. (The CUST command is described in Section 2.3; it permits the user to call a routine of his choice from within the executive.) In order to re-enter Software #1, start at address 3 with the command EXEC 3 followed by a carriage return.

When the program is first loaded from tape and executed, a checksum test is performed on the code in memory. If the check fails, an error message will appear, and you have the option of using a carriage return to start the program. Appendix 2 explains why it is not a good idea to start a program that has failed the checksum test. Software #1 is recorded twice on the cassette, so that if the first recording fails, you can still try the second one.

SECTION 2

THE EXECUTIVE

2.3 INTRODUCTION

When Software Package #1 is first loaded from cassette, you are in executive mode, and can use any of commands given in the summary below. All commands must be entered in upper case and begin in the first column of the video screen. At least one blank must separate the command from its first argument, and any two arguments must be separated from one another by at least one blank. Arguments must also be in upper case.

CTRL-X	Disregard anything typed since the last carriage return; start a new line and accept input from the keyboard
ENTR	Enter data into memory
DUMP	Display memory data
FILE	Create, assign or display file information
EXEC	Execute a program
ASSM	Assemble a source file to object code
LIST	List file
DELT	Delete lines of file
CUST	Execute routine whose starting address is stored at 02A8
SAVE	Save a file, or memory contents, on tape
GET	Get a file, or memory contents, from tape

There is one other type of entry that you can make in executive mode: by typing any four-digit number, you can insert a line having that line number into a memory file. (Obviously, you have to create a file first.)

A detailed discussion of all of the commands will begin in Section 2.3, below. The use of the editor to enter lines in files is discussed in Section 3.1.

Software Package #1 does input and output by calling the SINP and SOUT entry points of SOLOS/CUTER, and therefore through the current SOLOS/CUTER pseudo-ports. (See the SOLOS/CUTER User's Manual, Section IV.) The user may temporarily halt the output from a LIST, DUMP, or ASSM command by depressing the space-bar, and then cause it to resume by depressing the spacebar again. Typing the MODE SELECT (or CTRL and @ keys together) will terminate output from a command and return control to the executive.

2.4 CONVENTIONS USED IN COMMAND DESCRIPTIONS

The following conventions will be used in the discussion of command format:

1) UPPER CASE letters, and all numbers, should be entered literally in the form of a command. For example, SAVE means that the user should actually type the word SAVE.

2) Lower case letters represent the item to be typed, but are not literal. For example,

name

means "type a name."

3) Optional elements in a command are enclosed in square brackets. LIST [number] means that the LIST command need not include a number. Of course, the presence or absence of a number affects the way the command operates; default values for optional parameters are included in the discussion of each command.

In some cases, omitting a particular parameter makes it necessary to omit another. For example,

[name [/unit] [address]]

means that you can

a) include name, unit, and address, OR
b) omit name, unit, and address, OR
c) omit unit and/or address, but not name.

Neither unit nor address may be specified if name is omitted.

4) <cr> means "type a carriage return." Unless an exception is noted, the user should enter a carriage return to terminate any line of input. The computer will supply a carriage return at the end of every line of output.

5) aaaa and bbbb are used to represent hexadecimal addresses.

2.5 COMMAND DESCRIPTIONS

ENTR aaaa Enter data into memory

This command is used to enter data into memory starting at address aaaa. Enter data in hexadecimal format; when you have finished typing the data that you want to enter, type a slash (/). If you enter more than one item into memory, as in the example, the items that you type will occupy successive memory locations.

Example:

```
ENTR 1500<cr>
0 0A 30 44 FF FE/
```

DUMP aaaa bbbb Dump Contents of Memory

This command is used to examine the contents of memory. The values contained in memory from locations aaaa to bbbb are displayed in hexadecimal. Each line of display consists of an address followed by the contents of the next 16 memory locations. If bbbb is not specified, or if it is a number less than or equal to aaaa, only the contents of location aaaa will be displayed. The space-bar may be used to stop the listing at any point; the next key depressed will cause the listing to resume. If you want to discontinue the listing and return to the executive, type MODE SELECT (CTRL-@).

Example:

```
-------------------------------------------------------------
DUMP 1140 1152<cr>
0040: 0A D8 D6 07 C9 DB 00 E6 45 00 D0 01 D3 02 F8 CF
0050: E6 7F C9
-------------------------------------------------------------
```

FILE[/name/[aaaa]] Create, assign, or display file information

This command is used to create and delete files, to examine their beginning and ending addresses, and to establish whether or not a given file, or any file, will be "current," i.e., available for editing or assembly. Up to six files can exist simultaneously, with any one of these files regarded as "current". Depending on the form of the command and its parameters, the following functions are performed.

* NOTE * Slashes around a filename are literal.

FILE /name/ aaaa	Create a file with the specified name, starting at address aaaa, and make it current. If a file with the same name already exists, display the error message NO NO.
FILE /name/ 0	Delete the named file and make no file current. Note, no file can start at address 0.
FILE /name/	Save all parameters of the existing current file; then make the named file the current file.
FILE	Display parameters of the "current" file in the following format, with aaaa and bbbb being the beginning of file and end of file addresses: name aaaa bbbb
FILES	Display the parameters of all files currently known by the system.

Example:

```
FILE /ONE/ 1000<cr>                 (user creates file ONE, begin-
                                     ning address 1060)
ONE 1000 1000                       (parameters of file displayed)
0001 THIS IS THE FIRST LINE.<cr>    (user enters lines into file)
0026 THIS IS THE SECOND. <cr>
0029 LINE 3<cr>
FILE /TWO/ 1100<cr>                 (user creates file TWO, begin-
                                     ning at address 1100)
TWO 1160 1160                       (parameters of file displayed)
FILE /THREE/ 1200<cr>               (user creates file THREE,
                                     beginning at address 1200)
THREE 1200 1200                     (parameters of file displayed)
0010 DEAR JOHN<cr>                  (user enters lines into file)
0012 PAY ME OR I WON'T<cr>
0014 BE YOUR FRIEND.<cr>
0015 SEE YOU SOON,<cr>
0017 IGOR<cr>
FILE /TWO/<cr>                      (user saves parameters of cur
                                     rent file THREE and makes TWO
TWO 1100 1100                       current- parameters displayed)
1300 FILE TWO GETS THIS LINE.<cr>   (user enters lines into file)
1984 UPPER CASE OKAY.<cr>
1000 NO LOWER CASE!<cr>             (lines needn't be entered in
2710 END TWO<cr>                    order)
FILE<cr>                            (user requests parameters of
                                     current file)
TWO 1100 1159                       (parameters of file displayed)
FILES<cr>                           (user requests parameters of
                                     all files)
TWO 1100 1159                       (parameters of files displayed)
THREE 1200  127E
ONE 1000 1045
```

FILE STRUCTURE AND MEMORY MANAGEMENT

Files created in Software #1 have a structure in which the first byte of a line contains a count of the number of characters' in that line, including the counter itself, the line number, and the carriage return that ended the line. Thus

```
0008  THIS IS A POSSIBLE LINE.
```

contains 32 bytes: a counter, 3E characters of text, and a carriage return. Note that the line number, and the space or spaces that follow the line number, are included in the line. The end of a file is indicated by a 01 (Hex) character; a final count of the number of bytes in a file will reflect the presence of this character.

When you create and develop a file in Software #1, you decide where that file will be placed in memory. The system will not allow you to use the same name for more than one file,

but it places no restriction on the number of times you can place data in the same location. You can create a file, store an assembled program or enter data in a memory location that you have already used; the new information will replace the previous contents of that location.

The only way that you can protect against losing information accidentally is to be aware of how much memory you have, how you will need to use the available storage area, and how you can best avoid storing over data that you might still want to use. Be sure that you are aware of the beginning and ending addresses of each of your files, and make a note of locations in which you have entered data using ENTR. When you create a file, consider its length in bytes and whether the file is likely to grow so large as to encroach on the beginning of another file. If you plan to assemble a program, remember to leave enough room for the symbol table. More specific suggestions related to assembly of source programs are included in the discussions of the ASSM command and the assembler.

EXEC aaaa Execute a program

This command is used to execute a program at address aaaa; if you want to return control to the executive when execution is complete, be sure that you have included a RET instruction in the program. If the OUST command is in use and you want to return to SOLOS or CUTER, you can use the EXEC command and assign to aaaa the beginning address of SOLOS (C000) or CUTER. To re-enter the executive, give the EXEC 3 command from SOLOS/CUTER.

LIST [number] List file

This command is used to display the lines entered by the user into the current file. The output consists of the lines in the file, starting at the specified line number. If no number is specified, display starts at the beginning of the file. The LIST command does not change the format of the file in any way, e.g., by reducing multiple blanks to one; each line appears precisely as you entered it. Lines are listed with line numbers in ascending order, as they exist in the file. Use MODE SELECT or the CTRL and @ keys simultaneously to discontinue a listing and return to the executive.

Example:

```
-----------------------------------------------------------------
FILE   /SIMPL/    1A2B<cr>        (user creates current file)
0000 WAIT EI<cr>                  (user enters lines into file)
0010  JMP WAIT+1<cr>
0020 *  THIS ENABLES INTERRUPTS AND WAITS<cr>
0024       END<cr>
LIST 10<cr>                       (user says start list at 10)
0010  JMP WAIT+1                  (lines of file are displayed)
0020 *  THIS ENABLES INTERRUPTS AND WAITS
0024       END
LIST<cr>                          (user says list current file)
0000 WAIT EI                      (list starts at beginning of
0010  JMP WAIT+1                  file, by default)
0020 *  THIS ENABLES INTERRUPTS AND WAITS
0024 _     END
-----------------------------------------------------------------
```

DELT linel [line2] Delete line(s) from file

This command is used to delete lines from the current file. All lines from linel to line2, inclusive, are deleted from the file. If line2 is not specified, only linel is deleted.

Example:

```
-----------------------------------------------------------------
(see example under LIST)
DELT 10 20                        (user says delete lines 10-20)
LIST                              (user asks for list of edited
0000 WAIT EI                      file)
0024       END                    (lines 10 and 20 are gone)
-----------------------------------------------------------------
```

SAVE [/name/ aaaa bbbb] Save a file or other portion of memory on cassette tape

This command is like the SOLOS/CUTER SAVE command, except that here the filename is surrounded by slashes. The command records the contents of a file or of a specified area of memory on a cassette placed in tape unit 1. If there are arguments given, as in the example above, the contents of memory from address aaaa to address bbbb are saved, and a file name of up to five characters is recorded in the cassette file header, (with type "P" for Program). The command has a simpler form, SAVE with no arguments, which saves the whole current file on tape. The name which has already been assigned to the file will also be its name on the cassette file header, and it will have a file type "T" (Text).

GET /name/ [aaaa] Read a cassette tape file into file or memory

This command reads the specified file from tape unit one into the area of memory beginning at aaaa. If aaaa is absent from the command, the file will be read into the area occupied by the current file. The filename used must be the same as when the file was SAVEd; a filename must be specified. The form of this command is like that of the SOLOS/CUTER GET command, except that, as with the SAVE command, the filename must be surrounded by slashes.

CUST Execute program whose beginning address is stored at 02A8

This command is a call to the address stored at 02A8. (Actually, the high byte of the address must be stored at 02A8, the low byte at 02A9.) If you have not entered a different address, 02A8 and 02A9 will contain a return entry point to SOLOS/CUTER. To enter the address of a program other than SOLOS/CUTER, use the ENTR command:

```
ENTR 02A8
<high byte> <low byte>/
```

where the high and low bytes are entered in hexadecimal form, with a space between them. The routine to be called must be an object program starting at the given address. If an 8080 RET instruction is included in the program, and if proper stack operations are maintained, control will return to the executive.

The example below illustrates the use of the CUST command to return to SOLOS/CUTER. These are the steps to follow if you want to use CUST to execute a program other than SOLOS/CUTER:

1) Using the FILE command, create a current file to contain the source program.
2) Using the editor, enter the source program.
3) Using the ASSM command (described below), assemble the source program to object code. If you are likely to want to use the program again, SAVE the object code on cassette tape.
4) Using the ENTR command, store the beginning address of the object program in locations 02A8 and 02A9.

If you have followed these instructions, your program will be executed whenever you give the CUST command from the executive.

Example: (assumes a printer using the serial port)

```
CUST<cr>                (user enters command to return
                        to SOLOS)
>SET O=1<cr>            (user sets serial output port)
>EX 3<cr>               (user re-enters Software #1)

LIST                    (user requests a listing of the
                        current file on the present
                        output device; in this case,
                        the listing will be sent
                        to the printer through the
                        serial port)
```

ASSM[E] aaaa [bbbb] Assemble a source file to object code .

In order to be executed, a source program must be assembled to object code. This command initiates assembly of the CURRENT file, if that file contains a source program; here is a description of what happens to the current file when you enter the ASSM command (see Section 4 for more details):

1) The assembler will create a symbol table, beginning at 0F60, to contain all of the labels that you have defined in your source program. (See the description of source program statements in Section 4.2.) The symbol table requires seven bytes per symbol; when you determine where in memory to locate a file or an object program, be sure to allow for the anticipated size of this symbol table.

2) Next, the assembler will translate each statement to its corresponding binary representation and assign an address in memory to each byte of the resulting object code. Consecutive bytes of object code will be assigned consecutive addresses beginning at aaaa, unless you use the ORG pseudo-operation (described in Section 4.6) to override this procedure.

3) If you include the bbbb parameter, the object program will be stored at location bbbb. If no value is specified for bbbb, the code is placed at aaaa. Remember to assign to aaaa the address at which the program will eventually be loaded; the only purpose of bbbb is to allow a temporary displacement of the code. For example, if the source program starts at aaaa and you do not specify a value for bbbb, the object program will replace the source program in memory. If the assembler listing then indicates that there are errors in the source program, you will have to reenter the source code in order to be able to correct it. A program that is assembled to run at aaaa, but that is actually stored at bbbb, must be SAVEd and reloaded, if it is to run properly.

4) If you do not use the optional E in the command, the assembler produces a complete listing; each line of the listing consists of a) the beginning address assigned to the line, b) a hexadecimal representation of the object code (note that the number of bytes of object code will vary with the instruction), c) a single-letter error code, where appropriate (see Section 4.8), and d) the original source program line. If you specify the E, only lines containing errors will be listed. The listing will appear on the screen, unless you have changed the output pseudo-port in SOLOS or CUTER.

The second line of the sample program below contains an opcode error signified by the letter 0. In order to show that JMP is an operation, rather than a label, you must separate it from the line number by MORE THAN ONE blank space; this requirement will be discussed along with other assembler requirements in Section 4.2.

Examples:

```
LIST<cr>                    (user asks for listing of file)
0000 WAIT EI                (LIST starts at file beginning)
0010 JMP WAIT+1
0020 *  THIS SETS INTERRUPT AND WAITS
0024       END

ASSM 1A2B 2A2B<cr>          (user says assemble the current
(assembler listing nor-     file, using 1A2B as the first
mally begins on the line    address, and store the object
right after the command)    code beginning at 2A2B)

1A2B FB             0000 WAIT EI
1A2C 00 00 00 O     0010 JMP WAIT+1
1A2F                0020 * THIS SETS INTERRUPT AND WAITS

0010 JMP WAIT+1<cr>         (user enters correction)
ASSME 1A2B 2A2B<cr>         (user says reassemble file,
                            showing only lines with errors)
                            (there are no errors)
SAVE /OBJ/ 1A2B 1A2F        (user saves file on tape, with
                            filename OBJ)
GET /OBJ/ 1A2B              (user reloads file at appropri-
                            ate address)
EXEC 1A2B                   (user executes object program)
```

2.6 ERROR MESSAGES:

The executive has three error messages:

1) WHAT? indicating an improper command or an error in the parameters following the command;

2) NO CURRENT FILE indicating that you have entered a command that references a current file when none exists; and

3) NO NO indicating that you have attempted to create a file that already exists in memory.

SECTION 3

THE EDITOR

Whenever you are in executive mode and begin an entry with a four digit number, the editor inserts a line having that line number (and consisting of whatever text follows it) into the current file. If you have already begun to experiment with the system, you have probably noticed that only upper case letters can be entered from the keyboard. Also, to be incorporated into the current file, the line MUST begin with a four digit line number in column 1.

Line numbers can range from 0000 to 9999 Decimal, so that 10,000 lines can exist in each file (if enough storage is available.) When you enter the carriage return that terminates every input line, the line, including the carriage return, becomes part of the file in the current file area. The editor places all line numbers in sequence and automatically overwrites an existing line in the file if you enter a new line with the same line number. Because you are not required to enter the lines of the file in order, the display will not always mirror the contents of the current file. (If you want to see the file with lines and line numbers in the proper sequence, use the LIST command.)

THE EDITOR DOES NOT AUTOMATICALLY ASSIGN LINE NUMBERS; any input line which does not begin with a four digit line number will be regarded as a command to the executive. (Use leading zeros for numbers less than 1000.) An entry to the editor may include a maximum of eighty characters plus the carriage return that terminates the line. On the Sol video display or VDM-1, characters following the sixty-fourth are displayed on the next line of the screen.

To insert a line between two lines that already exist, you will have to assign it a line number between the line numbers of the two existing lines. Thus, if you anticipate that you will want to insert lines after the initial entry of a program or text file, you should space your original line numbers to permit insertions. (For example, if line 0010 is followed by line 0015, it is no problem to add line 0012, whereas if line 0010 is followed by line 0011, the user can make no insertions without renumbering the lines.)

Other constraints on the format of input lines are related to the operation of the assembler.

SECTION 4

THE ASSEMBLER

4.1 GENERAL OPERATION

When you give the ASSM command, the assembler translates an 8080 assembly language (source) program into 8080 machine (object) code. The assembler, like the editor, will operate only on the current file. The remainder of this manual will be devoted to describing the features of the Software #1 assembler.

The assembler translates each line of the current file into object code. Although each line must contain a line number, the line number does not become part of the object program. It is however- reproduced on the assembler listing. The character position immediately after the line number should be left blank; thus the "first source code character position" is the second position following the last digit of the line number.

The Software #1 assembler operates in two "passes": that is, it passes over the source program twice, from beginning to end, in the course of creating the corresponding object program. If you are not already familiar with the syntax of source program statements, you might want to look at Section 4.2, below before reading the rest of this discussion.

On the first pass, the assembler creates a symbol table, defining all symbols in the order that they appear as labels in the source program. BIG EQU LITL is meaningful only if LITL appears in the label field of an earlier statement. With the minimum 4K of memory, there is room for 22 five-character symbols in the symbol table. (Only the first five characters are used to identify a symbol, although you may enter more than five characters.) Each entry in the symbol table requires seven bytes (five for the symbol, two for the value of that symbol), so that for each additional 1K of memory, there is room for 146 more symbols in the table.

Remember to calculate how much memory you need to store your source and object programs. You can figure out the length of a source program or text file by counting characters, including line numbers, spaces, carriage returns, byte counters, and the end of file character; one character equals one byte. You can determine the length of an object program by knowing the number of bytes generated by each instruction and then adding, but this procedure is long and tedious - nobody follows it. A good practice is to allow plenty of space; if you give your object program about 20% as much room as your source file, you will almost certainly have made ample provision for it.

On the second pass, each source program instruction is translated into the corresponding object code; all expressions,

symbols, operation codes, and ASCII constants are assigned their absolute values, and each line of code is given an address. Remember that the correct form of the ASSM command is

ASSM[E] aaaa [bbbb]

Addresses assigned to the object code begin at aaaa, unless you use the ORG pseudo-operation in the source program to override this procedure.

As the object program is generated it is placed in memory starting at bbbb. If bbbb has been omitted from the command, assembled code is stored beginning at aaaa. The listing, also produced on the second pass, indicates exactly what data has been assigned to each location in memory. Because the SOUT entry point of SOLOS/CUTER is used the listing will appear on the video screen, unless you have used the SOLOS/CUTER SET command to direct output to a different pseudo-port. (See your SOLOS/CUTER User's Manual.)

If the E option is included in the ASSM command, only lines which contain errors will be listed.

The description of the ASSM command in Section 2.3 contains and explains an example of an assembler listing.

4.2 SOURCE PROGRAM STATEMENTS

STATEMENTS may contain either symbolic 8080 machine instructions or assembler pseudo-operations. A statement may include as many as four fields, separated by blanks:

[label] operation [operand] [comment]

The label field, if present, must begin in the first source code character position, that is, it must be separated from the line number by exactly one blank space. The symbol in the label field can contain any number of characters, but only the first five characters are used in the symbol table; thus, the label SUBSET1 and the label SUBSET2 will be regarded as identical and, if they appear in label fields in the same program, will cause an error message to be displayed. All symbols in the label field must begin with a letter, and may contain only letters and numbers.

The operation field contains either an 8080 operation mnemonic or a pseudo-operation code; this field must be occupied, unless the line is a comment line.

The operand field contains arguments required by the operation in the operation field. If two arguments are present, they must be separated by a comma (BUT NO BLANKS), as in the first line of the example below:

```
0015 FLOP  MOV M,B  COMMENT
0020 * COMMENT
0025       JMP  BEG
0030       CALL FLOP
0035 BEG   ADI  8+6-4
0040       MOV  A,B
```

The comment field is for explanatory remarks. It is reproduced on the listing during the second pass. (Line 0015, above, has a comment field.) There is a distinction between a comment field and a comment line. A comment line, like line 0020 in the example above, does not translate into any object code; whenever a line begins with an asterisk (*) in the first source code character position, the assembler disregards that line. The comment field of a program statement does not generate any code, either, but the rest of the line is assembled normally.

Notice that all of the fields are separated from one another by one or more blank spaces. Here is a summary of the rules concerning blanks.

1) The label field must be separated from the line number by EXACTLY ONE blank. If there is more than one blank following the line number, the assembler will regard the next character as the first of an operation or pseudo operation mnemonic. If there is no blank immediately following the line number, the line will be found to contain an error.

2) If the label field is omitted, the operation field must be separated from the line number by MORE THAN ONE blank. Otherwise, the operation or pseudo-operation mnemonic will be interpreted as a label in the label field.

3) A comment line must begin with an asterisk separated from the line number by EXACTLY ONE BLANK.

4) With the one exception of the comment field, there can be NO BLANKS within fields, including within arithmetic expressions. Blanks are regarded as field separators.

In all other cases, fields may be separated by any number of blanks.

4.3 SYMBOLIC LABELS

To assign a symbolic label to a statement, put the desired symbol in the label field. When a label begins a statement, the assembler assigns it a value that corresponds to the address of the next byte to be assembled, that is, to the first address assigned to the statement. The only exception to this rule is that the EQU pseudo-operation causes the symbol in the label field to assume the value indicated in the operand field.

(For example, the assembler will respond to the statement

```
POTTS EQU 128
```

by assigning the value 128 to the label POTTS.) Any label that has been defined, whether by an EQU or by another statement, can be used in the operand field of another statement in the program. In fact, because the assignment of values to labels occurs on the first pass, it is unnecessary to define a label in an earlier statement than the one in which it appears in the operand field; for example, line 0025 in the last example says jump to BEG, when BEG does not appear in the label field until later in the program. The only exception to this rule is that a label can not appear in the operand field of an EQU statement unless it has been defined earlier in the program.

There are eight symbols which the assembler has reserved and associated with particular values. These names may not be used; except in the operand field, and the user has no control over their definition. They are

```
A- the accumulator  (7)
B- Register B       (0)
C- Register C       (1)
D- Register D       (2)
E- Register E       (3)
H- Register H       (4)
L- Register L       (5)
M- Memory           (6)
```

The numbers in parentheses are the decimal equivalents of the binary codes used to represent these entities. In a program statement, either the single-letter abbreviation or the decimal number may be used. For example, the two instructions

```
INR 5   and    INR L
```

assemble to the same byte of object code.

The 8080 mnemonics for the Stack Pointer (SP) and Program Status-Word-(PSW) are not predefined in Software #1. In the context of an 8080 assembly language program, they must be associated with a value of 6; therefore, in order to use SP or PSW in a program, you must insert the lines

```
PSW EQU 6
SP EQU 6
```

in the text of your source program file.

In addition to the above reserved symbols, there is the single special character symbol ($). This symbol changes in value as the assembly progresses. It is always equated with the next address to receive object code after the current instruction is assembled. $ may only be used in the operand field.

Examples:

```
JMP $       means jump to the location following this
MOV A,B     instruction, i.e., to the MOV instruction.
            (This JMP is obviously redundant.)
```

4.4 RELATIVE SYMBOLIC ADDRESSING

If the name of a particular location is known, a nearby location may be represented by an expression including the known name and a numeric offset. The offset is expressed in bytes of object code rather than in source program lines, so that the variable length of instructions must be taken into account. Only + and - may be used in the expression of an offset.

Example:

```
UPDATE STA TEMP+3    SAVE ACCUMULATOR IN TEMP LOC
       LDA IT        GET IT
       INR A         INCREMENT IT
       STA IT        STORE IT
       LDA TEMP+3    GET BACK THE ACCUMULATOR
       ....
       ....
TEMP   DB 10
       DB 20
       DB 30
       DB 40         TEMP+3 POINTS HERE
```

In this example, the instruction STA TEMP+3 causes the contents of the accumulator to be saved in a location three bytes beyond that denoted by TEMP. The instruction LDA TEMP+3 means get the contents of the location three bytes beyond TEMP, and put those contents into the accumulator.

4.5 CONSTANTS

The assembler allows you to write positive or negative numbers in the operand field. These numbers will be regarded as decimal constants, and their binary equivalents will be used appropriately. All unsigned numbers are considered positive.

Hexadecimal constants must be have an "H" after them, to indicate that they are not decimal constants. 25H is an example of a hexadecimal constant; 25 and 25D will both be regarded as decimal. (The D is okay, but not required.)

A hexadecimal constant may not start with a letter; use a leading zero to show that an item in the operand field is a number, rather than a label.

```
A5H will be assumed to be a label.
0A5H will be assumed to be a hexadecimal constant.
```

ASCII constants are represented within single quote marks (SHIFT-7). A constant consisting of two characters may be represented within one set of single quotes, e.g., LXI H,'AB' is acceptable. To handle a longer string, use a series of statements containing the Define Byte pseudo-operation (see the next section). For example,

```
DB 'E'
DB 'R'
DB 'R'
DB 'O'
DB 'R'
```

will cause the string ERROR to be stored in successive locations in memory.

4.6 EXPRESSIONS

An expression is a sequence of one or more symbols, constants or other expressions separated by the arithmetic operators plus or minus. Remember not to include any blanks; a blank will always be regarded as a field separator. Expressions may be used only in the operand field.

```
PAM+3
ISAB-'A'+52
LOOP+32H-5
```

Expressions are calculated using 16 bit arithmetic; thus, all arithmetic is done modulo 65536. Single byte data cannot contain a value greater than 255 or less than -256. Any value outside this range will result in an assembler error.

4.7 PSEUDO-OPERATIONS

Pseudo-operations are written as ordinary statements, but they direct the assembler to perform certain functions which do not always develop 8080 machine code. This section describes the pseudo-operations that you can use in Software #1 source programs.

ORG--Set Program Origin: [label] ORG expression

This pseudo-operation determines the first address that the assembler will assign to subsequent statements. If a label is included, it is assigned the value of the expression. If ORG is used at the beginning of a program, the value of the expression in the operand field becomes the origin, or first address, to be assigned to the program; ORG OVERRIDES THE aaaa PARAMETER OF THE ASSM OR ASSME COMMAND (see section 2.3). Similarly, an ORG statement within a program overrides the rule of assigning consecutive addresses to consecutive statements; the ORG statement indicates the next address to be assigned, regardless of previous addresses. The statements that follow

the ORG statement are given addresses beginning at the one represented by the expression.

END--End of Assembly

This pseudo-operation informs the assembler that the last source statement has been read. The assembler will proceed to the second pass or, if both passes have been completed, terminate and return control to the executive. This pseudo-operation is not required because the assembler will stop when it reaches the end of file indicator (01H).

EQU--Equate Symbolic Value: label EQU expression

This pseudo-operation assigns the value of the expression to the given label. For example,

```
GREEN EQU 3
COLOR EQU GREEN
BLUE EQU GREEN+1
```

will assign a value of 3 to both GREEN and COLOR, and a value of 4 to BLUE. Notice that GREEN had to be defined before it was used in the operand field of another statement.

DS--Define Storage: [label] DS expression

This pseudo-operation reserves a specified number of successive memory locations but does not store anything in them. The expression determines how many locations will be reserved; the assembler "skips" these, beginning at the current position in the program, and makes its next assignment to the location immediately following the reserved area. The expression must evaluate to a 16-bit number, i.e., it must be possible to represent that number in 16 or fewer bits.

DB--Define Byte: [label] DB expression

This pseudo-operation reserves a single memory location and assigns to it a value determined by the expression. The expression must evaluate to a number which can be represented in 8 or fewer bits.

DW--Define Word: [label] DW expression

This pseudo-operation reserves two bytes of storage and assigns to them a value determined by the expression. The expression must evaluate to a number which can be represented in 16 or fewer bits; the low order byte of the value is stored at the lower memory address, in conformity to the Intel format for two-byte operands. For example, DW 'AB' stores 'B' at a lower memory location than 'A'.

4.8 ERROR MESSAGES:

The following error messages are displayed on the assembler listing when the error occurs. Some of the errors are only displayed during the first pass.

A	Argument Error	An improper argument 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 a string longer than 2 characters, e.g., 'ABC'.
D	Duplicate Label	Two labels in the program are the same, i.e., the first five characters are identical.
L	Label Error	The symbol in the label field contains illegal characters.
M	Missing Label	Error A label has not been included in an EQU instruction.
O	Opcode	Error The symbol in the operation field is not a valid 8080 instruction mnemonic or assembler pseudo-operation mnemonic.
R	Register Error	An invalid symbol was used to designate a register.
S	Syntax Error	The syntax of a statement does not comply with the require ments of the assembler.
U	Undefined Symbol	A label in the operand field can not be found in the symbol table, i.e., it does not appear in the label field anywhere in the source program.
V	Value Error	The expression in the operand field is outside the allowed range.

APPENDIX 1

USING CASSETTES

Successful and reliable results with cassette recorders and cassette files requires 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 inteface circuitry, bad memory locations, bad tape, a faulty recording, improper alignment or settings on the cassette recorder, or other equipment problems.

(Appendix II -- 8080 Opcode table sorted by function omitted)

ProcessorTechnology

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

Software #1 Update 731070

The procedure described below corrects an error in the way that Software #1, Release 1.6 (MOD 0), accepts input from a driver other than the Sol keyboard. It is recommended that all users of the program make this "patch"; otherwise, an attempt to use an input device other than the Sol keyboard will have undesirable consequences. (Also, the documentation for any future patches will assume that this patch has been made.) In the examples the symbol <cr> is used to denote a carriage return, and the symbol > is the SOLOS/CUTER prompt.

The patch is made by reading the program into memory and making alterations there, then saving the corrected program on a cassette tape other than the production cassette. The commands used are those described in the SOLOS/CUTER User's Manual. This notice is intended to enable the user to make the necessary changes in SOFT1, and to have a minimal understanding of the processes involved, without having to refer to that manual. If you are not very familiar with the operation of cassette recorders, you may want to consult the appendix entitled "Using Cassettes" in your Software #1 Resident Assembler User's Manual.

1) Set up a cassette recorder for reading, and load SOFT1 with the SOLOS/CUTER GET command:

>GET SOFT1<cr>

2) When the prompt (>) returns, type

>EN 39<cr>

to indicate that the next number entered (see step 3) `should replace the present contents of memory location 39H. If step 3 involved a list of numbers, those numbers would be placed in consecutive memory locations beginning at 39H.

3) When a colon (:) appears on the screen, type

47/<cr>

to enter the number 47H at location 39H.

4) When the prompt (>) returns, type

>EN FAF<cr>

to indicate that the next number entered should replace the present contents of memory location FAFH. Subsequent entries (i.e., the 51 in step 5) will occupy consecutive locations following FAFH.

5) When a colon (:) appears on the screen, type:

F9 51/<cr>

Notice that the list of entries must again be terminated by a slash. (In step 3 there was only one entry in the list, but the slash was still required.) When the prompt (>) returns, the patch has been completed.

6) Set up a cassette recorder for writing. Insert and position a cassette tape other than the production cassette. Be sure to leave at least half a minute at the beginning of the tape unrecorded.

7) After hitting the MODE SELECT key to return to SOLOS/CUTER command mode, type:

>SET XE=0<cr>

to set the starting address that will be written in the tape file header.

8) Type

>SAVE SOFT1 0 FB1<cr>
or
>SAVE SOFT1/2 0 FB1<cr>

(The first version assumes that the file is being written to a cassette in unit 1; the second assumes that the file is being written to a cassette in unit 2.) When the file has been recorded completely, the prompt (>) will be displayed again.

9) Before copying the patched version of SOFT1 to the other side of your cassette tape (NOT the production cassette), load the program FROM THE TAPE THAT YOU HAVE JUST WRITTEN to make sure that it works. If the input pseudo-port setting that you use when you execute SOFT1 is the Sol keyboard and you have access to another input device, try using the COST command to return to SOLOS/CUTER, the SET command to change the input pseudo-port, and the EXEC command to restart SOFT1. (Use the example on top of page 2-8 in the Software #1 Resident Assembler User's Manual, but type "SET I= " instead of "SET O= ", and use the port designation that matches your device.) If you do not have another input device, just try several of the commands to see that nothing is amiss.

If the program seems to be working properly, use the CUST command to return to SOLOS/CUTER. If not, load SOFT1 again (as in step 1) and repeat steps 2 through 8, writing the new patched version over the old patched version of the program and testing your results.

10) To copy the patched version of SOFT1 to the other side of the tape on which you have recorded it, load the program again (as in step 1) from the ORIGINAL side of that cassette, and repeat steps 6 through 8 above, making sure you write the file to the REVERSE side of the cassette.

11) Note on your duplicate cassette that you have updated your copy of the Software #1 program from MOD 0 to MOD 0/1. When you execute the program, it will continue to identify itself as MOD 0; it is therefore very important that you make a record of the correct MOD number, so that you will know whether any further update notices apply to you.

Remember that Processor Technology software is copyrighted. Your duplicate is for your use for maintenance and backup purposes only. No copies should be given away or sold.