M0101

B A S I C / 5

User's Manual

For Use with

SOLOS, CUTER and CONSOL

PROCESSOR TECHNOLOGY CORP.
6200 Hollis Street
Emeryville, CA 94608
(415) 652-8080

SOFTWARE TECHNOLOGY CORP.
P. O. Box 5260
San Mateo, CA 94402
(415) 349-8080

I M P O R T A N T N O T I C E

S O F T W A R E W A R R A N T Y

Software Technology Corporation warrants this software product to be free from defects in material and workmanship for a period of three months from the date of original purchase.

This warranty is made in lieu of any other warranty expressed or implied and is limited to repair or replacement, at the option of Software Technology Corporation, transportation and handling charges excluded.

To obtain service under the terms of this warranty, the defective part must be returned, along with a copy of the original bill of sale, to Software Technology Corporation within the warranty period.

The warranty herein extends only to the original purchaser and is not assignable or transferable and shall not apply to any software product which has been repaired by anyone other than Software Technology Corporation or which may have been subject to alterations, misuse, negligence, or accident, or any unit which may have had the name altered, defaced or removed.

Table of Contents

Table of Contents (cont.)

APPENDICES

I. INTRODUCTION

BASIC (Beginner's All-purpose Symbolic Instruction Code) is a computer programming language characterized by versatility and ease of use. Its resemblance to standard mathematical notation and simple English statements enables novices and professionals to program solutions to a variety of problems in the shortest possible time.

BASIC is a conversational language which permits a user to sit down at his computer or terminal device and engage in a dialog with it. The results may be either immediate answers to a mathematical problem or a working computer program which may be used in the future to process new data.

There are many good books available to instruct the user in how to program in BASIC; therefore, no attempt has been made to teach BASIC in this manual. Appendix F lists several references that may be of interest.

Here, we present only Processor Technology's Sol BASIC programming language, its features and restrictions. One of the best ways to learn Sol BASIC is to experiment with your system.

Sol BASIC features include...

* Full 8-digit precision
* Multiple statement entry on one line
* BCD (Binary Coded Decimal) arithmetic for maximum accuracy in all mathematical operations
* User formatting of output data
* Program storage on, and retrieval from, cassette tape
* Data files for processing and saving numeric data
* Implementation of many function subprograms
* Execution of most program statements in direct mode for immediate calculations and enhanced debugging
* Only 8K byte memory needed to run many programs
* ARG and CALL functions facilitate linkage to 8080 machine language program segments
* Unique line editor using video display and SOLOS
* Fully formatted listings with automatic FOR-NEXT indentation.

Definition of Terms

In this manual--

exp	means	mathematical expression
num	means	any number
rel exp	means	relational expression
statement n	means	statement number
"string"	means	uninterrupted series of literal alphanumeric characters enclosed with quotation marks
var	means	variable name
( )	means	optional

Sol BASIC is a program for use with SOLOS, CONSOL and CUTER with at least 8K bytes of memory available for use by Sol BASIC. Because SOLOS and CUTER operating systems are compatible, this manual will refer to SOLOS meaning either SOLOS or CUTER. Some features described herein require that the current system output pseudo port be zero (the VDM driver of SOLOS/CUTER).

II. PROGRAM STRUCTURE

A Sol BASIC program is comprised of statements. Every statement begins with a statement number, followed by the statement body, and terminated by a CR (carriage return), line feed, or a colon in the case of multiple statements.

There are four types of statements in BASIC: declarations, assignments, control and input/output. These statement types are described in corresponding sections of this manual.

A. Statements

General statement requirements for the Sol BASIC program are as follows:

1. Every statement must have a statement number ranging between 1 and 65000. Statement numbers are used by Sol BASIC to order the program statements sequentially.

2. In any program, a statement number can be used only once.

3. Statements need not be entered in numerical order because Sol BASIC will automatically arrange them in ascending order.

4. A statement may contain no more than 80 characters including blanks.

5. Blanks, unless within a character string and enclosed by quotation marks, are not processed by Sol BASIC. BASIC removes all excess blanks as the line is processed into the file so that minimum memory area is used. When listing or editing, it automatically inserts blanks to make the line more readable.

   Example: G O T O 5 0 0

   is exactly the same as

   GOTO500

6. More than one statement can be input on a line if separated by a colon, but only one statement number is allowed.

   Example: 520 LET A=1: B=3.2: C=5E2

B. Data Format

The range of numbers that can be represented in this version of Sol BASIC is: ±.99999999E±127.

There are eight digits of significance in this version of Sol BASIC. Numbers are internally truncated on entry to fit this precision.

Numbers may be entered and displayed in three formats: integer, decimal, and exponential.

Example: 153, 34.52, 136E-2

C. Variable Names

Variables may be named any single alphabetic character or any alphabetic character followed by a single numerical digit, e.g., A, B5, X, D1.

D. REM Statement

The REM, or remark statement, is a non-executable statement which has been provided for the purpose of documenting program listings. By generous use of REM statements, a complex program may be more easily understood. REM statements are only reproduced on the program listing. They are not executed. If control is given to a REM statement, it will perform no operation. (It does, however, take a finite amount of time to process the REM statement.)

CAUTION: A REM statement cannot be terminated by a colon with statements following on the same line.

Example: 150 REM NOW HOW: LET R1=3.5E2.1

The assignment statement ("LET" above) will never be executed. The entire line is considered to be a non-executable comment.

III. PROGRAM PREPARATION

After Sol BASIC is loaded into your system, it may be started at memory address 0. (Refer to Appendix A for a complete description.) Sol BASIC is initialized to support only 8K of memory. The amount of memory to be supported may be increased by using the "SET" command.

A return to SOLOS or CONSOL can be made by giving the BYE command. Sol BASIC can then be re-entered, leaving the existing program intact, by executing location zero.

The system is then ready to accept commands or statements. The user might enter the following program, for example:

```
150  REM PROGRAM TO DEMO
160  PRINT"ENTER SOME DATA",
170  INPUT B5
180  LET P7=B5+3/2
185  PRINT
190  PRINT B5,P7
200  END
```

A. Inserting a Statement

If the user wishes to insert a statement between two others, he needs only to type a statement number that falls between the other two. For example:

```
181  REM NOW FOLLOWS THE LET STATEMENT
```

B. Replacing a Statement

If it is desired to replace a statement, a new statement is typed that has the same number as the one to be replaced. For example:

```
180  LET P7=SIN(B5)
```

replaces the previous LET statement.

C. Terminating a Line

Each line entered is terminated by a carriage return or line feed, Sol BASIC positions the print unit to the correct position on the next line.

D. Editing

The MODE key may be used to erase a character or a line that was typed in error. Also, all editing functions--as outlined in the EDIT command description--are fully functional during normal input.

IV. COMMANDS

It is possible to communicate with Sol BASIC by typing direct commands at the terminal device. Also, certain other statements can be directly executed when they are given without statement numbers. See Calculator Mode in the next Section for more information.

Commands have the effect of causing Sol BASIC to take immediate action. A Sol BASIC language program, by contrast, is first entered into the memory and then executed later when the RUN command is given.

When Sol BASIC is first ready to receive a command, the word READY is displayed on the terminal device.

Commands are typed without statement numbers. After a command has been executed, the user will either be prompted for more information, or READY will again be displayed indicating that BASIC is ready for more input--either another command or additional program statements.

CLEAR COMMAND - CLEAR

Sets all variables to zero, resets the READ pointer and initializes the program so that it may be run. CLEAR may be used as a statement in programs that exit FOR TO loops or GOSUB in a non-standard fashion. The RUN command produces an automatic clear.

LIST COMMAND - LIST (statement n)

Causes all the statements of the current program to be displayed on the user's terminal. The lines are listed in increasing numerical order by statement number. The display will begin with statement n, if given.

RUN COMMAND - RUN

Causes the current program to begin execution at the first statement number. RUN always begins at the lowest statement number. RUN resets the DATA pointer and performs a CLEAR.

New Program COMMAND - NEW

The NEW command causes working storage and all variables and pointers to be reset. The effect of this command is to erase all traces of the program from memory and to start over.

HALT COMMAND - MODE (or CTL-@ Key)

This key on the terminal console will cause BASIC to halt its current operation and to respond with a READY. BASIC will then accept further commands. This command is often used to stop a LIST command before it has completed or to halt the execution of a program.

IV. COMMANDS (cont.)

LINE CLEAR COMMAND - MODE (or CTL-@) Key

Clear the current line buffer. If the user types a line at the terminal and decides that the line is in error and should be deleted, depression of the MODE (or CTL-@) key will clear the line.

CHARACTER ERASE COMMAND - DEL Key

Single character erase. If a character is determined to have been typed in error, it may be erased by striking the "DEL" key and then entering the correct character. See EDIT command for further explanation.

MULTIPLE STATEMENT PER LINE COMMAND - :

The use of colons provides the ability to enter more than one statement on a line. Each statement must be separated by a colon, and the total number of characters may not exceed the line length of 80 characters. There may be only one statement number on a line. Therefore, one cannot transfer control to any of the appended statements except by the natural program flow.

Example: 150 LET A=A+A:B=2*A: IF A=6 THEN PRINT B

SET COMMANDS

The SET command is used to specify various system options. The command is always followed by two operands. The first operand specifies the option being selected, and the second is the new value to be associated with that option. For example, SET S=1 means that the speed is to be set to the value of "one". The following operands are allowed via a SET command. (Note that some operands are allowable as both direct execution statements and statements of a program.)

SET I=exp Direct or Program Statement

This command sets the current system input to be the indicated Sol pseudo port (of the range 0 - 3). Once processed, all further system input will come from the specified device and not from the system keyboard unless the keyboard (pseudo port 0) is specified.

SET O=exp Direct or Program Statement

This command sets the current system output to be the indicated Sol pseudo port (of the range 0 - 3). Once processed, all further system output will be directed to the specified device and not to the display unless the display (pseudo port 0) is specified.

SET S=exp Direct or Program Statement *SOLOS only*

This command specifies the speed to be used by the Sol display driver. This is the speed at which the driver will display lines

on the screen. The value of the expression must be from the range 0 (the fastest rate) through 255 (the slowest rate).

SET N=exp Direct or Program Statement

This command specifies the number of nulls to follow every line-feed. The number of nulls required is unique to each output device. The Sol display driver does not require any nulls, while some printers may require as many as 30.

SET M=exp Direct Only

This command allows the user to specify a memory size larger than the default of 8192 (8K). The expression is evaluated as an integer number of bytes indicating the maximum memory to be used.

RETURN TO SOLOS/CONSOL COMMAND - BYE Direct or Program Statement

This command returns control to SOLOS or CONSOL, whichever is installed.

STORE PROGRAM COMMAND - SAVE name Direct Only *SOLOS Only*

This command, when used with SOLOS, records the current BASIC program onto cassette tape under the name specified. Only unit 1 is used for program storage. The tape speed is set to "high". CONSOL does not support this function.

READ PROGRAM COMMAND - GET name Direct Only *SOLOS Only*

This command when used with SOLOS will read from the specified CUTS tape and find the named program which must be the name assigned when the program was recorded. Once the program has been found, the entire program as recorded will be read into memory and become the current BASIC program. If the program is larger than memory defined by "SET M=", the entire program will be loaded followed by an "SO" error message.

AUTO RUN COMMAND - XEQ name Direct Only *SOLOS Only*

This command is similar to GET, but once the program has been read into memory and becomes the current BASIC program, an automatic RUN will be performed.

EDIT COMMAND - EDIT line-number *SOLOS Only*

This command allows the user to edit the specified line of the current BASIC Program. The Sol BASIC editor allows a line of program to be edited without having to re-enter the entire line. To accomplish this, various special keys found on the Sol keyboard are used to direct this editing process in conjunction with the display driver of SOLOS. They are:

DEL or Rubout	This key causes the current character (under the cursor) to be deleted and the remainder of this line to be shifted to the left.
← or CTL-A	The left arrow functions as a cursor control by moving the cursor to the left one character. This is used to position the cursor prior to making a change in the line.
→ or CTL-S	The right arrow moves the cursor to the right one character.
↑ or CTL-W	The up-arrow activates the insert mode. When entering characters to the editor, there are two possible modes: insert and non-insert. Non-insert mode is the standard mode specifying that any character entered replaces the character at the cursor location. In insert mode, every character of the line from the cursor to the end of the line is shifted to the right to make room for the character being entered. Insert mode provides an easy way to enter a letter or word at any point on a line.
↓ or CTL-X	The down-arrow deactivates the insert mode.
RETURN	The carriage return terminates the editing of this line by clearing the line from the cursor to the end of the line.
LINE FEED	The line feed also terminates the editing of this line, but leaves the line exactly as on the screen.

The use of the REPEAT key facilitates rapid movement of the cursor through the line.

All of the edit functions are available at all times during input to Sol BASIC.

Pausing the Display

Sol BASIC offers a feature that is quite useful whether outputting to the display screen or any other device. At every carriage return, Sol BASIC looks to see if the space bar of the keyboard has been pressed. If the space bar has been pressed, Sol BASIC will pause until any other key on the keyboard has been pressed. What this means, for example, is that pressing the space bar while listing a program causes the listing to stop to keyboard control.

V. DIRECT EXECUTION - CALCULATOR MODE

Sol BASIC facilitates computer utilization for the immediate solution of problems--generally of a mathematical nature--which do not require iterative program procedures. To clarify: Sol BASIC may be used as a sophisticated electronic calculator by means of its "Direct" statement execution capability.

While BASIC is in the command mode, some BASIC statements may be entered without statement numbers. BASIC will immediately execute such statements. This is called the direct mode of execution:

```
Example:  A=1.5
          PRINT A
```

Statements that are entered with statement numbers are considered to be program statements which will be executed later.

In the following sections of this manual, all Sol BASIC statements are described. Only those statements which are flagged with the word "Direct" may be used in the direct mode.

Another use for direct execution is as an aid in program development and debugging. Through use of direct statements, program variables can be altered or read, and program flow may be directly controlled.

VI. DECLARATION STATEMENTS

NUMERICAL ARRAY DIMENSION STATEMENT - DIM var (exp) Direct

Allocates memory space for an array. In Sol BASIC, only single dimension arrays are allowed. Maximum array size is determined by available memory. All array elements are set to zero by the DIM statement.

If an array is not explicitly defined by a DIM statement, it is assumed to be defined as an array of 10 elements upon the first reference to it in a program.

CAUTION: An array can be dimensioned only once in a program--dynamically or statically.

```
DATA STATEMENT      - DATA num (,num...,num)
READ STATEMENT      - READ var (,var...,var)
RESTORE STATEMENT   - RESTORE
```

The DATA and READ statements are used in conjunction with each other as one of the methods to assign values to variables.

Every time a DATA statement is encountered, the values in the argument field are assigned sequentially to the next available positions of a data buffer. All DATA statements, no matter where they occur in a program, cause data to be combined into one data list.

READ statements cause values in the data buffer to be accessed sequentially and assigned to the variables named in the READ statement.

Example:
```
110 DATA 1,2,3.5
120 DATA 4.5,7,70
130 DATA 80,81
140 READ B2,B3,D5,Z6

is the equivalent of:

10 LET B2=1
20 LET B3=2
30 LET D5=3.5
40 LET Z6=4.5
```

The RESTORE statement causes the data buffer pointer, which is advanced by the execution of READ statements, to be reset to point to the first position in the data buffer.

Example:
```
110 DATA 1,2,3.5
120 DATA 4.5,7,70
130 DATA 8,81
140 READ B2,B3
150 RESTORE
160 READ D5,D6
```

In this example, the variables would be assigned values equal to:

```
    100  LET B2=1:B3=2:D5=1:D6=2
```

VII. ASSIGNMENT STATEMENTS

LET STATEMENT - LET var=exp Direct

The LET statement is used to assign a value to a variable. The use of the word LET is optional.

```
Example:  100 LET B=827
          110 LET B5=87E2
          120 R=(X*Y)/2*A
```

The equal sign does not mean equivalence as in ordinary mathematics. It is the replacement operator. It says, replace the value of the variable named on the left with the value of the expression on the right. The expression on the right can be a simple numerical value or an expression composed of numerical values, variables, mathematical operations, and functions.

MATHEMATICAL OPERATORS

The mathematical operators used to form expressions are:

```
- (unary)  .. Negate (requires only one operand)
* ........... Multiplication
/ ........... Division
+ ........... Addition
- ........... Subtraction
```

The unary minus (negate) may appear in sequence with any other mathematical operator (e.g., A#-B). No other mathematical operators may appear in sequence, and no operator is ever assumed: A++B and (A+2) (B-3) are not valid.

An arithmetic expression is evaluated in a particular order of precedence: negation is performed first, then multiplication and division, and last, addition and subtraction.

In cases of equal precedence, the evaluation is performed from left to right.

The order of evaluation can be controlled explicitly through use of pairs of parentheses. The expression inside the innermost pair is evaluated first; the outermost last.

```
Example:  150 LET R=A+B-C/2*3

                is evaluated as:

          Temp1= A + B - Temp2
          R = A + B - Temp2

Example:  137 LET R= ((A+B)-C)/(2*3)

                is evaluated as:

          Temp1= A+B   Temp2=Temp1 - C
          Temp3 = 2*3   R=Temp2/Temp3
```

VIII. CONTROL STATEMENTS

Control statements are used to control the natural sequential progression of program statement execution. They can be used to transfer control to another part of a program, terminate execution, or control iterative processes (loops).

FOR AND NEXT STATEMENTS - FOR var=exp1 TO exp2 (STEP exp3)

```
                          :
                          :
                        NEXT (var)
```

The FOR and NEXT statements are used together for setting up program loops. A loop causes the execution of one or more statements for a specified number of times. The variable in the FOR-TO statement is initially set to the value of the first expression (expl). Subsequently, the statements following the FOR are executed. When the NEXT statement is encountered, the named variable is added to the value indicated by the STEP expression in the FOR-TO statement, and execution is resumed at the statement following the FOR-TO. If the addition of the STEP value develops a sum that is greater than the TO expression (exp2), the next instruction executed will be the one following the NEXT statement. If no STEP is specified, a value of one will be assigned. If the TO value is initially less than the initial value, the FOR-NEXT loop will still be executed once.

Example:
```
110 FOR I=1 TO 10
120   INPUT X
130   PRINT I,X,X/5.6
140 NEXT I
```

Although expressions are permitted for the initial, final, and STEP values in the FOR statement, they will be evaluated only once--first time the loop is entered.

If the variable in the NEXT statement is not given by name, Sol BASIC will properly add the STEP value to the variable in the last FOR statement,

Example:
```
110 FOR K=1 TO 350
      :
      :
120 FOR L=1 TO 80
      :
      :
130 NEXT
135 NEXT
      :
      :
```

In the preceding example, the NEXT at statement number 130 will STEP the FOR loop beginning at statement 120. The NEXT at 135 will STEP the FOR loop beginning at 110.

It is not possible to use the same variable in two loops if they are nested. In the preceding example, the variable in line 120 could not be K.

When the statement after the NEXT statement is executed, the variable is equal to the value that caused the loop to terminate, not the TO value itself. In the first example, I would be equal to 11 when the loop terminates.

Sol BASIC lists FOR .. NEXT loops indented such that the nesting of loops produces a graphic demonstration of this nesting. This form of indention of loops has proved useful both in problem solving (debugging) and in structured programming.

STOP STATEMENT - STOP

The STOP statement causes the program to stop executing. Sol BASIC returns to the command mode. The STOP statement differs from the END statement in that it causes Sol BASIC to display the statement number where the program halted, and the program can be restarted by a GOTO. The message displayed is:

"STOP IN LINE XXXX"

END STATEMENT - END

The END statement causes the program to stop executing. Sol BASIC returns to the command mode. In Sol BASIC, END may appear more than once and need not appear at all. When END is encountered, the message "Sol BASIC" will be displayed. When a program terminates without an END, then only the message "READY" will be displayed.

GOTO STATEMENT - GOTO statement n Direct

The GOTO statement directs Sol BASIC to execute the specified statement unconditionally. Program flow continues from the new statement. The GOTO must be the last or only statement on a line. Any additional characters following the GOTO will be considered remarks.

Example: 150 GOTO 270

IF STATEMENT Direct

IF relational exp THEN statement n

IF relational exp THEN ONE OR MORE BASIC statements

The IF statement is used to control the sequence of program statements to be executed, depending on specific conditions. If the relational expression given in the IF is "true", then control is given to the statement after the THEN, If the relational expression is "false", program execution continues at the line following the line with the IF statement.

It is also possible to provide a BASIC statement after, the THEN in the IF statement. If this is done and the relational expression is true, the BASIC statement will be executed and the program will continue at the statement following the expression. When the IF is false, program execution continues at the line following the line with the IF statement.

When evaluating relational expressions, arithmetic operations take precedence in their usual order, and the relational operators are given equal weight and are evaluated last.

Example: 5+6*5 > 15*2 evaluates to be true

Relational expressions will have a value of -1 if they are evaluated to be "true", and a value of zero if they evaluate to "false".

Example: (12 > 10)=-1 or (A <> A)=0

The relational operators are:

=	means	Equal
<>	means	Not equal
<	means	Less than
>	means	Greater than
<=	means	Less than or equal
>=	means	Greater than or equal

```
Examples:  110 IF A > B+3 THEN 160
           180 IF A=B+3 THEN PRINT "VALUE A ",A: GOTO 140
           190 IF A < B THEN T1=B
```

IX. INPUT/OUTPUT STATEMENTS

INPUT STATEMENTS — Direct or Program Statement

INPUT ("string"(,))var(,var..var)(,)

The INPUT statement allows the user to enter data from the current system input device, usually the keyboard.

```
    Example:  110 INPUT A,B,C
              120 INPUT ""V(1),R,V(2)
```

When the program comes to an INPUT statement, a question mark will be displayed to the current output device, The user then enters the requested data separated by commas and followed by a carriage return. If no data is entered, or if insufficient data is given, the system will prompt the user with a double question mark: "??". Constants are the only data which may be entered in response to an INPUT. If the last variable is followed by a comma, the carriage return-linefeed will be suppressed if the input is also terminated with a linefeed.

If the optional preceding string expression is given, it causes the carriage return/line feed and the "?" prompt to be suppressed, When the optional "string" is given, the "?" prompt will be replaced with the user supplied "string" as the prompt, A null string ("") may also be used to suppress the prompt.

```
PRINT STATEMENTS - PRINT var
                   PRINT exp
                   PRINT %(Z)(E)(N)% (var,exp)
```

The PRINT statement directs Sol BASIC to output the specified values to the output device.

The value of expressions, literal values, simple variables, or text strings may be printed out, the various forms may be combined in the print list by separating each with a comma. If the entire list is terminated by a comma, the terminating carriage return/line feed will be suppressed.

Sol BASIC prints data in fixed width fields, Each datum begins at the leftmost position within each field, The number of digits, trailing zeroes, etc., are specified by the format specification, if any. If a (;) is used as a separator between elements of a print list instead of a comma, the next field will begin after the last character of the preceding field regardless of field widths.

```
    Example:  X=0:Y=0:Z=1: PRINT X;Y;Z

                   prints:

              0 1 1
```

If the next position to be printed is greater than or equal to position 56, then a carriage return/line feed is given before the next value is printed.

PRINT given with no arguments causes one line to be skipped.

All PRINT strings have a special feature--an easy means to output control characters. Whenever an ampersand (&) is encountered in a PRINT string it will not be printed but will cause the very next character of the string to be output as the control of that character.

```
Example:  "&A"=CTL-A;
          "&C"=CTL-C;
          "&&"=&.
```

The TAB Function

The TAB function is used in the PRINT statement to cause data to be printed in exact locations. TAB tells Sol BASIC which position to begin printing the next value in the print list. The argument of TAB may be an expression.

```
Example:  110 PRINT TAB(2),B,TAB(2*R),C
```

Note: The print positions are numbered zero to 71.

Formatted Print

Sol BASIC enables the user to control the format of the printed output by specifying: free format, exponential format, trailing zeroes, and the number of places of accuracy to the right of the decimal point.

If no specification is made, Sol BASIC will print eight places of precision with the low order digit rounded and trailing zeroes suppressed. It will also automatically select between the decimal, integer, and exponential formats, depending on the magnitude of the value to be printed.

It is possible for the user to override Sol BASIC's automatic formatting by including a format specification in the output list, A format specification is two percent signs with interposed code characters.

Format Specification %(Z) (E) (F) (N)%

F = Free Format (BASIC selects format)
Z = Print Trailing Zeroes
E = Print in Exponential Format
N = Print N (N=1-8) places to right of decimal point

All parameters are optional, but once a format specification is given, it will continue to be used until a new format specification is given. To force Sol BASIC to return to its usual default format, a format specification of %% must be given.

Example: 110 PRINT %5E%

245 PRINT %Z2%,A,B: PRINT %Z3%,CD,%%

Example: LIST

```
 5 FOR I = 1 TO 150 STEP 7.5
 6       B=I: GOSUB 50
 7       PRINT %Z2%;TAB(9);"$";TAB(M);B
 8       B=I*15/2: GOSUB 50
 9    PRINT %Z3%; TAB(M+10) ;B
10 NEXT
20 END
50 M=13: IF B < 1 THEN RETURN
55 M=12: IF B < 10 THEN RETURN
60 M=11: IF B < 100 THEN RETURN
65 M=10: IF B < 1000 THEN RETURN
70 M=9: RETURN
READY

RUN

    $   1.00      7.500
    $   8.50     63.750
    $  16.00    130.000
    $  23.50    176.250
    $  31.00    232.500
    $  38.50    288.750

                          etc. ...
```

Try running this program yourself.

X. SUBPROGRAMS

A subprogram is a sequence of instructions which perform some task that would have utility in more than one place in a Sol BASIC program. To use such a sequence from more than one place, Sol BASIC provides subroutines and functions.

A subroutine is a program unit that receives control upon execution of a GOSUB statement, Upon completion of the subroutine, control is returned to the statement following the GOSUB by execution of a RETURN statement.

A function is a program unit to which control is passed by a reference to the function name in an expression. A value is computed for the function name, and control is returned to the statement that invoked the function.

```
GOSUB STATEMENT - GOSUB statement n
                    .
                    .
                    .
                  statement n
                    .
                    .
                  RETURN
```

The GOSUB statement causes control to be passed to the given statement number. It is assumed that the given statement number is an entry point of a subroutine. The subroutine returns control to the statement following the GOSUB statement with a RETURN statement. A RETURN must be the last or only statement on a line. Any additional characters following the RETURN will be considered remarks.

Subroutine example:

```
100 X=1
110 GOSUB 200
120 PRINT X
125 X=5.1
130 GOSUB 200
140 PRINT X
150 STOP
200 X=(X+3)*5.32E3
210 RETURN
211 END
```

Subroutines may be nested; that is, subroutines can use GOSUB to call another subroutine which in turn can call another. A subroutine cannot call itself. Subroutine nesting is limited to six levels.

X. SUBPROGRAMS (cont.)

BASIC FUNCTIONS

ABS (exp)	Gives the absolute value of the expression.
INT (exp)	Gives the largest integer less than or equal to its argument.
RND (exp)	Generates pseudo-random numbers ranging between 0.0 and 1.0. The argument is required for syntax but does not alter the function. The random number generator is reset by the CLEAR command.
SGN (exp)	Gives a value of +1, if argument is greater than 0. Gives a value of -1 if argument is negative. Gives a value of 0 if argument is 0.
SQR (exp)	Gives the square root of the argument.
SIN (exp)	Gives the sine of the argument, when the argument is given in radians.
COS (arg)	Gives the cosine of the argument, when the argument is given in radians.
TAN (exp)	Gives the tangent of the argument, when the argument is given in radians.
TAB (exp)	See PRINT statement. Used to position output characters.
ARG (exp)	ARG and CALL are used together to link to assembly language program segments. Both may be used in the
CALL (exp)	direct mode.

ARGUMENT AND CALL FUNCTIONS - ARG and CALL

When the ARG function appears in some Sol BASIC statement such as B=ARG(V1), the argument will be evaluated as a sixteen bit integer and temporarily stored in the BASIC monitor. Should linkage be made to an assembly language (8080) program segment via the CALL function, the previously stored sixteen bits will be passed to the assembly language code in the B,C register pair.

When the CALL function is invoked by coding it into some BASIC statement such as X6=CALL(5.2*A4); the argument of the CALL function will be evaluated as a sixteen bit address. Sol BASIC will transfer control to that address using an 8080 "CALL" instruction.

X. SUBPROGRAMS (cont.)

Your machine language code loads registers B,C with any desired information. This information is then passed back into the Sol BASIC program as the value of the CALL.

```
Example:  110 REM LINK TO ASSY LANG PROG
          120 LET X=12: R3=3192
          130 B=ARG(X/5)
          140 LET M=CALL(R3)
          150 PRINT M
          160 END
```

In this example, B is assigned the value of the ARG argument, linkage is made to assembly language program at address 3192, and M is set to whatever was returned in B,C.

To let back into ALS8 the user can use B=CALL(57440).

XI. FILES

Sol BASIC, when used with SOLOS, has provision for writing data to and reading data from cassette data files. Two cassette recorders are supported, so one file may be read from one unit, the data processed, and the processed data written to the other unit. When using files, the cassette operations--other than correctly placing the cassettes--are under control of Sol BASIC running with SOLOS.

Prior to using the FILE OPERATIONS, a few concepts are important, The term FILE, for example, refers to a collection of data, and no assumptions are made by Sol BASIC as to the structure of the FILE. It only provides convenient access to the file information while you, or your program, can define any structure,

For example, the following program:

```
4 FILE #1
5 FOR I=1 TO 10
6 INPUT "NEXT NUMBER"A
7 PRINT #1, I,A,SQR(I+A)
8 NEXT I
9 CLOSE #1
```

would produce a file with the following definable structure:

1. The FILE is 30 "ELEMENTS" long, (3 elements "printed" to the file 10 times.)

2. Every third ELEMENT, starting with the first (1,4,7...) will be a number one larger than the preceding element. (1,20 the value of "I".)

3. Every third element, starting with the second, will be a number as input by the user at line 6.

4. Every third element, starting with the third, will be a number determined by the square root of the sum of the preceding two elements.

5. The END OF FILE follows the 30th element.

Using this type of "structuring" the file "characteristics" are as follows:

Element - one data unit

Record - A series of elements determined by the user. (Three elements per record in the preceding example.)

Length - The number of elements in a file or the number of records. (30 elements, 10 records in the preceding example.)

EOF - The end of the file which is important if the length of the file is unknown or may vary.

The following example illustrates another type of file which produces a usable structure:

```
 5  FILE #1
10  INPUT "PART NUMBER?"  A
15  IF A=0 THEN 100
20  PRINT
25  INPUT "QUANTITY ON HAND?"  B
30  PRINT
35  INPUT "MINIMUM QUANTITY?"  C
40  PRINT
45  INPUT "COST PER UNIT?"  D
50  PRINT
55  PRINT #1,A,B,C,D
60  GOTO 10
100  CLOSE #1
105  END
```

A. File Operations

FILE #1, FILE #2

In order for the file to be accessed, Sol BASIC must first set up a number of internal parameters. This is known as an OPEN operation, and it must be performed once--and only once--prior to any attempt to access a file. If an OPEN is attempted on an already open file, the program will abort giving a DM error message.

CLOSE #1, CLOSE #2

This operation informs Sol BASIC that no further accesses are going to be made to the file. The operation must be performed each time you are through with a file.

PRINT #1, PRINT #2

This operation writes data to a file. Any number of expressions or variables may follow the comma.

READ #1, READ #2

This operation reads data from a previously written file. Any number of variables can follow the comma.

XI. FILES (cont.)

B. END OF FILE - EOF

When Sol BASIC detects an END of file, it makes a special return to your program. After each successful READ, Sol BASIC returns to the line following the READ statement. Notice this is the next valid line preceded by a line number.

When the END of file is encountered, Sol BASIC searches for the first colon following the READ #1, and executes the statement found there. The following statements--

```
10  FOR I=1 to 1000000
15  READ #1,A:PRINT "END OF FILE";:GOTO 30
20  PRINT I,A
25  NEXT I
30  CLOSE #1
35  END
```

read elements of a file, printing each one until the EOF is reached (assuming there are fewer than one million elements!!). Upon encountering the end, Sol BASIC would print "END OF FILE", close the file and then stop.

C. Let's Use the File Operations

1. Put a cassette in Unit 1. Be sure the ON-OFF control is properly connected, and set the unit to the RECORD mode. Note the counter position so you can rewind the tape to the starting point later.

2. INPUT and RUN the following program:

```
 5  FILE #1
 6  FOR I=1 TO 250
 7  PRINT #1;I,SIN(I), SQR(1)
 8  NEXT I
10  CLOSE #1
```

3. You have now written a file to the cassette tape. Take the recorder out of the RECORD mode and rewind the tape to the start of the file.

4. Place the recorder in the PLAY mode.

5. INPUT and RUN the following program:

```
 5  FILE #1
10  FOR I=1 TO 300
15  READ #1,A,B,C:PRINT "END OF FILE";:GOTO 30
20  PRINT A,B,C
25  NEXT I
30  CLOSE #1
35  END
```

A P P E N D I C E S

Appendix A

Loading Sol BASIC

Sol BASIC is distributed on a single cassette. Side one of this tape is 1200 Baud CUTS format while Side two is 300 Baud Kansas City format. CONSOL, SOLOS and CUTER provide the commands necessary to read in this tape.

Sol BASIC may be read into memory and automatically executed by entering "XEQ BASIC". If "GET BASIC" is entered, Sol BASIC will be read into memory but will not automatically be executed.

Sol BASIC is always executed beginning at location zero (EXEC 0). The first execution of Sol BASIC will perform initialization procedures. Once initialized, Sol BASIC may be re-entered at location zero which will leave the BASIC program intact.

A very special feature of Sol BASIC is the capability of specifying the end-of-statement character (usually colon ":") and the print-concatenate character (usually semi-colon ";"). As part of the initialization procedures, Sol BASIC will display the characters currently being used for the end-of-statement and print-concatenate as well as the memory locations which the user may alter to specify any other character to be used.

This capability provides the means to make Sol BASIC appear compatible with most other BASIC's. For example, some BASIC's use a back slash (\) as an end-of-statement character. By changing one location in memory, Sol BASIC will accept and list programs with a back-slash instead of a colon.

The characters altered in this manner by the user should not be used for any other purpose. Should the same character be specified for both end-of-statement and print-concatenate, this character will then be used solely for end-of-statement.

Appendix B

ABBREVIATIONS

Sol BASIC offers the capability of entering abbreviations for each of the reserved words. This capability will greatly reduce the number of keystrokes improving efficiency. Once entered, all abbreviations will be converted to the complete reserved word when listed. Abbreviations are indicated by a character string terminated with a period which first matches an entry in the table of reserved words.

Reserved Word	Shortest Way To Enter
ABS	ABS
ARG	ARG
BYE	B.
CALL	CA.
CLEAR	CLE.
CLOSE	CLO.
COS	COS
DATA	D.
DIM	DIM
EDIT	ED.
END	E.
FILE	FI.
FOR	F.
GET	GET
GOSUB	GOS.
GOTO	G.
IF	IF
INPUT	I.
INT	INT
LET	LET
LIST	L.
NEXT	N.
NEW	NEW

Reserved Word	Shortest Way To Enter
PRINT	P.
READ	READ
REM	REM
RESTORE	RES.
RETURN	R.
RND	RND
RUN	RUN
SAVE	SA.
SET	SET
SGN	SGN
SIN	SIN
SQR	SQR
STEP	STEP
STOP	S.
TAB	TAB
TAN	TAN
THEN	TH.
TO	TO
XEQ	X.

Appendix C

LINE EDITING

The line editing capability of Sol BASIC makes use of the enhanced display driver available with SOLOS. This driver is always monitoring the character sequences being displayed. Whenever an "escape" character is to be displayed, the driver treats this as the beginning of a command and not to be displayed. An escape is followed by one or two bytes which further indicate the nature of the command as well as various options. For example, an escape followed by a byte containing a binary eight (8) allows the display speed to be varied. Table C-1 defines the various escape sequences supported by the driver contained within SOLOS. In each case the bytes) following the escape character is a binary value.

Table C-1. Escape Sequences Supported by SOLOS Display Driver.

Escape Code	Meaning
01	The next byte indicates the position within the current line at which to position the cursor.
02	The next byte indicates the line number at which to position the cursor. The top most line is known as Line 1, and the bottom line is Line 16.
03	This command is useful only to a machine language program. The current line number of the cursor is returned in Register "B", and the current position within that line is returned to Register "C",
04	This command is useful only to a machine language program. The absolute memory address of the cursor within the display RAM is returned in Register Pair "BC".
05 06 07	These three commands cause whatever character is in Register "B" to be output directly to the current cursor location of the display. This is useful both to display the escape character itself and to display characters in inverse video.
08	The next byte following indicates the speed which the display is to occur. A binary zero (0) is the fastest and a binary 255 is the slowest.
09	This escape command functions the same as does the 01 above.

Although these escape sequences appear to be of value only to the machine language programmer, let your imagination run wild. With these sequences, you could write a BASIC program which might move the cursor around the screen to produce various patterns . . . or who knows what?

Remember that these escape sequences are formed by outputting this data to the display driver. Instead of displaying the escape, and following byte(s), this data is treated like an internal command to the driver.

Appendix D

ERROR MESSAGES

ERRORS	EXPLANATION
BA	Bad argument. A command has been given an illegal argument.
SN	Bad syntax.
CS	Control stack error. For example, FOR has no corresponding NEXT, illegal FOR-NEXT, GOSUB-RETURN nesting, or control stack too deep.
DI	Direct input error. User has tried to give BASIC a command which it cannot process in the direct mode.
DM	Dimension error. Attempt to dimension (DIM) array more than once in program, attempt to open an already opened file used for reads.
FP	Floating point arithmetic error. User has attempted to divide by zero, or a calculation has resulted in a number too large to be represented in BASIC's number format. Note: Underflow will result in zero with no error indication.
IN	Input error. User has given a number in incorrect format in response to an INPUT statement.
LL	Line too long. User has attempted to input a line of more than 72 characters.
LN	Line number error. Line number specified in a GOTO, GOSUB, or IF statement was not found.
NA	Negative argument for square root function.
OB	Out of bounds. An array index, TAB value or other integer has exceeded its permissible limit. Also an attempt to load a BASIC program above available memory.
RD	Read error. No more data in data buffer, the number of READ statements has exceeded the number of DATA values given, or an error during cassette read operations.
SO	Storage overflow. Working memory has insufficient room for text, symbol table, array space, or program is too large.

Appendix E

THE BASIC CHARACTER SET

I	II	III	IV	V
←	@	:	+	%
↑	?	9-0	*	$
]	>	/	)	#
\	=	.	(	"
[	<	-	'	!
Z-A	;	,	&	

Appendix F

Sol BASIC Statement-Summary

CLOSE #n	Terminates processing of a CUTS tape file where n-1 or 2. This statement is required for an output tape.
DATA num(,num...,num)	Supplies data for READ state ment.
DIM var(exp)	Used to dimension numerical arrays containing a subscript greater than 10.
END	Halts program execution.
FILE #n	Prepares BASIC for later INPUT's or PRINT's to a CUTS tape file where n=1 or 2.
FOR var=exp TO exp(STEPexp) . NEXT (var)	Loop control statements; var must be the same in both statements (if used).
GOSUB statement n . statement n . RETURN	Transfers control to the sub-routine beginning at statement n, and then returns control to the statement following GOSUB.
GOTO statement n	Branches to statement n.
IF relational exp THEN statement n IF rel. exp THEN statement n	If the relational expression is true, branches to statement n, or executes statement n. The next program line will be executed if false.
INPUT ("string"(,))var(,var…var)(,) READ #n,var(,var...,var);	A statement processed at end of file. Requests numerical data at program execution time.
(LET) var=exp	Assigns value of expression to variable.
PRINT var PRINT "string" PRINT exp PRINT #n;var or exp	Outputs variable or literal values. Forms may be combined (except as noted). An "&" within a string outputs the next character in con-trol mode (e.g., &X means CTL-X).
READ var(,var...,var)	Reads numerical values from DATA statements.

REM anything	Comment statement.
RESTORE	Resets READ pointer to beginning of first DATA statement.
SET I=exp SET O=exp SET S=exp SET N=exp	Select the system input device. Select the system output device. Select the display speed. Select number of nulls.
BYE	Return control to the CONSOL or SOLOS personality module.
STOP	Terminates program.

Appendix G

REFERENCES

Entering BASIC, J. Sack and J. Meadows, Science Research Associates, 1973.

BASIC: A Computer Programming Language, C. Pegels, Holden-Day, Inc., 1973.

BASIC Programming, J. Kemeny and T. Kurtz, Peoples Computer Co., 1010 Doyle (P.O. Box 310), Menlo Park, CA 94025, 1967.

BASIC, Albrecht, Finkle and Brown, Peoples Computer Co., 1010 Doyle (P.O. Box 310), Menlo Park, CA 94025, 1973.

A Guided Tour of Computer Programming in BASIC, T. Dwyer, Houghton Mifflin Co., 1973.

Programming Time Shared Computer in BASIC, Eugene H. Barnett, Wiley-Interscience, L/C 72-175789 ($12.00).

Programming Language #2, Digital Equipment Corp., Maynard, MA 01754.

101 BASIC Computer Games, Software Distribution Center, Digital Equipment Corp., Maynard, MA 01754 ($7.50).

What to Do After You Hit Return, Peoples Computer Co., 1010 Doyle (P.O. Box 310), Menlo Park, CA 94025 ($6.95).