Using NSC Tiny BASIC™

CHAPTER 1

1.1 Introduction

This reference guide is intended to provide you with information on the use of NSC Tiny BASIC language. This section will also provide you with information on NSC Tiny BASIC commands, statements, grammar, error messages, and control characters. A brief description of each is given along with a short example or two to demonstrate their use.

This reference guide will provide a quick method of locating basic information on NSC Tiny BASIC. For a more detailed description, and examples of NSC Tiny BASIC's use, Section 1 should be consulted.

To learn how to use NSC Tiny BASIC, you will need an INS8073 system and a teletype or CRT terminal.

CHAPTER 2

2.1 Language Expressions

2.1.1 Variables

There are twenty-six variable names which can be used with NSC Tiny BASIC. These are the letters of the English language alphabet, A through Z. The values assigned to these variables are 16-bit signed integers. There are no fractions or floating point numbers.

2.1.2 Constants

All numeric constants are decimal numbers except when preceded by a pound sign (#). If preceded by #, the number is interpreted as a hexadecimal number. The symbols 55 would be treated as a decimal number, while #55 would be treated as a hexadecimal number (equal to 85 in decimal value). Decimal constants may be in the range of -32767 to 32767.

2.1.3 Relational Operators

Relational Operators are the standard BASIC symbols:

= equal to

> greater than

< less than

<= less than or equal to

>= greater than or equal to

<> not equal to

The relational operators return either a Ø (FALSE) or -1 (TRUE) as a result. NOTE: >< is an illegal operator.

2.1.4 Arithmetic Operators

Standard arithmetic operators are provided for the four basic arithmetic functions.

+ addition

- subtraction

/ division

* multiplication

Arithmetic is accomplished by standard 16-bit twos-compliment arithmetic. Fractional quotients are truncated, not rounded; therefore, 16/3 will give 5, 17/3 will also give 5 as a result. Remainders resulting from division are dropped. No attempt is made to round off the quotient. As usual, division by zero is not permitted; it will result in an error break.

The usual algebraic rules for order in evaluating expressions is followed. The order of evaluation is controlled by parentheses, and their liberal use is advised. They provide clarity and avoid confusion in complicated expressions.

2.1.5 Logical Operators

NSC Tiny BASIC provides Logical Operators AND, OR and NOT in addition to the arithmetic operators. These perform bitwise logical operations on their 16-bit arguments and produce 16-bit results. The AND and OR operators are called binary operators because they perform an operation on TWO arguments (or operands). An example follows with binary interpretation:

```
>LIST
 10 A = 75                  A = 0000 0000 0100 1011
 20 B = 99                  B = 0000 0000 0110 0011
 30 C = A AND B             C = 0000 0000 0100 0011
 40 PRINT C

>RUN
67
```

2.1.6 Logical AND

```
>LIST
10 INPUT A
20 INPUT B
30 IF (A>50) AND (B>50) THEN GO TO 60
40 PRINT "ONE OR BOTH ARE SMALL"
50 GO TO 10
60 PRINT "BOTH ARE BIG"
70 GO TO 10

>RUN
? 51
? 52
BOTH ARE BIG
? 51
? 49
ONE OR BOTH ARE SMALL
? 49
? 49
ONE OR BOTH ARE SMALL
?^C
STOP AT 10
>
```

2.1.7 Logical OR

```
>LIST
10 INPUT A
20 INPUT B
30 IF (A>50) OR (B>50) THEN GO TO 60
40 PRINT "BOTH ARE SMALL"
50 GO TO 10
60 PRINT "ONE OR BOTH ARE BIG"
70 GO TO 10

>RUN
? 51
? 52
ONE OR BOTH ARE BIG
? 51
? 49
ONE OR BOTH ARE BIG
? 49
? 49
BOTH ARE SMALL
?^C
STOP AT 10
>
```

2.1.8 Logical NOT

The third logical operator (NOT) is a unary operator. It performs an operation on only ONE argument, as follows:

```
>LIST
>10 A = 11                A = 0000 0000 0000 1011 = 11
                                                     10
>20 B = NOT A
>30 PRINT B               B = 1111 1111 1111 0100 = -12
>RUN                                                 10
-12
```

2.2 Functions

There are several functions that may be used in arithmetic expressions in NSC Tiny BASIC. These are described below.

2.2.1 MOD (a,b) Function

Returns the absolute value of the remainder a/b, where a and b are arbitrary expressions. If the value of b is zero, an error break will occur as in any division operation. As an example:

```
>10 A = 95                    2
>20 B = 44            44/ 95
>30 PRINT MOD (A,B)          88
>RUN                          7   ------MOD (95,44)
 7
```

2.2.2 RND (a,b) Function

Returns a pseudo-random integer in the range of a through b, inclusive. For the function to perform correctly, a, should be less than, b, and b-a must be less than or equal to 32767 (base 10). A typical example is:

```
>10 PRINT RND (1,100)
>RUN
 27
```

2.2.3 STAT Function

Returns the 8-bit value of the INS8073 Status Register. STAT may appear on both sides of an Assignment Statement: so, the programmer can modify the Status Register as well as read it. The Carry and Overflow Flags of the register are usually meaningless, since the NSC Tiny BASIC interpreter itself is continually modifying these flags. The Interrupt-Enable Flag may be altered by an assignment to STAT these flags. The Interrupt-Enable Flag may be altered by an assignment to STAT (such as: STAT = #FF). Location of individual flags are shown below:

Most Significant Bit — Least Significant Bit

7	6	5	4	3	2	1	0
CY/L	OV	SB	SA	F3	F2	F1	IE

Example of use:

```
>10 LET A = STAT
>20 PRINT A
>RUN
 176  --------The decimal number, 176, translates to:
                1 0 1 1 0 0 0 0 binary.
```

2.2.4 Status Register Bit Functions

The function of each bit in the Status Register is described below:

BIT DESCRIPTION

7 CARRY/LINK (CY/L): This bit is set to 1 if a carry occurs from the most significant bit during an add, a compliment-and-add, or a decimal-add machine language instruction. This bit may also be set by the operations performed by the SHIFT RIGHT WITH LINK (SRL) and the ROTATE RIGHT WITH LINK (RRL) machine language instructions. CY/L is input as a carry into the bit 0 position of the add, compliment-and-add, and decimal-add machine language instructions.

6 OVERFLOW (OV): This bit is set if an arithmetic overflow occurs during an add (ADD, ADI or ADE) or compliment-and-add (CAD, CAI or CAE) machine language instructions. Overflow is not affected by decimal-add (DAD, DAI or DAE) machine language instructions.

NOTE: The above two bits may be of little or no use in an NSC Tiny BASIC program.

5 SENSE BIT B (SB): This bit is tied to an external connector pin and may be used to sense external conditions. This is a "read-only" bit: therefore, it is not affected when the contents of the accumulator are copied into the status register by a STAT instruction. It is also the second interrupt input and may be examined by the "ON" command.

4 SENSE BIT A (SA): This bit is also tied to an external connector pin. It serves, as does SENSE BIT B, to sense external conditions. In addition, it acts as the interrupt input when the INTERRUPT ENABLE (see bit 3 of status register) is set. This bit is also a "read-only" bit. The same "ON" command may be used to sense this input. This flag is used by NSC Tiny BASIC as the serial input bit from the TTY or CRT.

3 USER FLAG 3 (F3): This bit can be set or reset as a control function for external events or for software status. It is available as an external output from the INS8073.

2 USER FLAG 2 (F2): Same as F3. This flag is used by NSC Tiny BASIC to control the paper tape reader relay.

1 USER FLAG 1 (F1): Same as F3. This flag is used by NSC Tiny BASIC as the serial output bit (with inverted data) to the TTY or CRT.

NOTE: The flag 1, 2 and 3 outputs of the status register serve as latched flags. They are set to the specified state when the contents of the accumulator are copied into the status register. They remain in that state until the contents of the status register are modified under program control.

Ø INTERRUPT ENABLE FLAG (IE): The processor recognizes the interrupt inputs if this flag is set. This bit can be set and reset under program control. When set, NSC Tiny BASIC recognizes external interrupt requests received via the SENSE A or B inputs. When reset, it inhibits the INS8073 from recognizing interrupt requests.

2.2.5 TOP Function

Returns the address of the first byte above the NSC Tiny BASIC program in the current page which is available to the user. This will be the address of the highest byte in the NSC Tiny BASIC program plus 1. All the memory in the RAM above and including TOP can be used by the NSC Tiny BASIC program as scratchpad storage. As an example:

```
>10 PRINT TOP
>RUN
4400      ------4400 is the first address of unused RAM
```

2.2.6 INC (X) and DEC (X) Functions

These statements increment or decrement a memory location X. Examples:

```
>10 LET X=1032
>20 INC (X)
     .
     .
     .
     .
>50 DEC (X)
>60 INC (6000)
>70 DEC (6001)
```

These instructions are used for multiprocessing and are non-interruptable. This means that if two 8073's are used on the same bus, whenever one executes an INC (X) or DEC (X) instruction, other processors must remain idle. These instructions are used, generally, for communications between processors in a multiprocessor system.

2.3 Statements

2.3.1 INPUT Statement

Data can be input to an NSC Tiny BASIC program by using the INPUT statement. One or more items (variables, expressions etc.), separated by commas, may be entered according to the following formats:

```
10 INPUT A
20 INPUT B,C
```

When the statement at Line 10 is executed, NSC Tiny BASIC prompts the user with a question mark. The user types in a number which is assigned to the variable A after the RETURN key is pressed. NSC Tiny BASIC then prompts the user with another question mark. The user types in two expressions, separated by commas, which will be assigned to B and C in that order.

```
RUN
? 45
? 237, 4455
```

NSC Tiny BASIC would now continue with execution of the program. String input is also allowed. See the String Handling section in this chapter for more information.

NSC Tiny BASIC accepts both numbers and expressions typed in response to an INPUT request. For example:

```
>10 A=10
>20 INPUT B,C
>30 PRINT B,C
>RUN
?A+1,A*2
 11  20
```

The comma between the entered expressions is not mandatory and can be replaced by spaces if the second expression does not start with a plus or minus sign.

There must be at least as many expressions in the input list as variables in the INPUT statement. If an error occurs when NSC Tiny BASIC tries to evaluate the typed-in expression, the message:

```
RETYPE
```

is printed along with the error message, and the question mark (?) prompt will appear again so that the user can type the expressions correctly.

The correct response to an `INPUT $factor' statement is a string, terminated by a carriage return. Quotation marks are not used for input.

INPUT may not be used in the command mode.

2.3.2 PRINT Statement (Output)

The PRINT Statement is used to output information from the program. Quoted strings are displayed exactly as they appear with the quotes removed. Numbers are printed in decimal format. Positive numbers will be preceded by a space, and negative numbers will be preceded by a minus (-) sign. There is a trailing space for all numbers. A semicolon (;) at the end of a PRINT Statement suppresses the usual carriage return and line feed with which NSC Tiny BASIC terminates the output.

Strings stored in memory (such as those generated by a String Input Statement) may also be printed. Refer to the String Handling Section in this chapter for more information. Typical example:

```
>PRINT "THIS IS A STRING"
>20 A=10
>30 B=20
>40 PRINT "10 PLUS 20=", A+B
>RUN
THIS IS A STRING
10 PLUS 20=30
```

2.3.3. LET Statement (Assignment)

The word, LET, may be used or omitted in an Assignment Statement. The execution of an assignment statement is faster if the word LET is used. The left portion of an Assignment Statement may be a simple variable (A-Z), STAT or a memory location indicated by an @ followed by a variable, number or an expression in parentheses, (refer to Indirect Operator for more information). Examples:

```
LET X=7
X=7
LET E=I*R
E=I*R
STAT=#70
LET @A=255
@(T+36)=#FF
```

Conditional assignments may be made without using an IF statement. The method hinges on the fact that all predicates are actually evaluated to yield -1 if true, and 0 if false. Thus, if a predicate is enclosed in parentheses, it may be used as a multiplier in a statement as:

```
LET X= -A*(A>=0)+A*(A<0)
```

which would assign the absolute value of A to X.

2.3.4 The GO TO Statement

NSC Tiny BASIC allows GO TO Statements to allow program branches to a specific line number or a line number called by an arbitrary expression. As examples:

```
10 GO TO 50
```

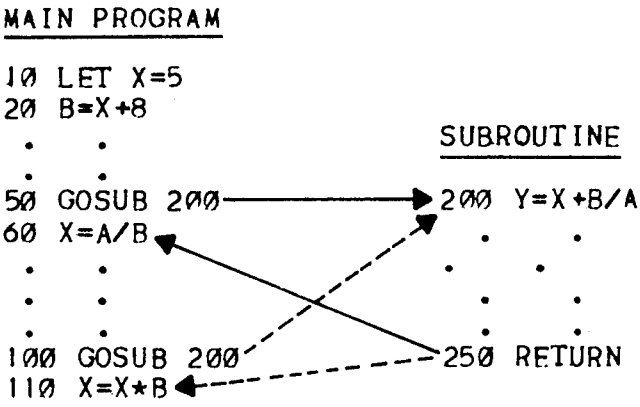
would cause the program to jump from Line 10 directly to Line 50, but

```
10 GO TO X+5
```

would cause the program to jump from Line 10 to Line X+5. Thus, the value of X is variable allowing dynamic control of program execution at this point.

2.3.5 GOSUB/RETURN Statements

These instructions are useful when a computation or operation must be performed at more than one place in a program. Rather than write the routine at each place where needed, a GOSUB instruction is used to "call" the computation or operation (referred to as a SUBROUTINE). After the subroutine has been executed, a RETURN instruction (the last instruction of the subroutine) causes the program to resume execution at the next line number following the original GOSUB instruction. As an example:



On the first GOSUB call, the order of execution follows the solid arrows. At the second GOSUB call (Line 100), the order of execution follows the dashed arrows.

NOTE: GOSUBs may be nested up to 8 levels deep (including interrupt levels).

2.3.6 IF/THEN Statement

This instruction allows for program control to be modified by a logical test condition. The test condition follows the IF clause of the statement. When the test condition is true (non-zero), the THEN portion of the statement will be executed. When the test condition is false (zero), the THEN portion is ignored and execution continues at the next numbered line of the program.

```
50 IF X>J THEN GO TO 140
```

NSC Tiny BASIC allows the omission of the word THEN from an IF/THEN Statement. This omission, also allowed on some larger BASICs, enhances the clarity of the program. The above statements then become:

```
50 IF X>J GO TO 140
```

2.3.7 DO/UNTIL Statements

This instruction is not available in standard BASICs. This statement is used to program loops, keeping GO TO statements to a minimum. The overall effect is to greatly improve readability and clarity of NSC Tiny BASIC programs. The following example shows the use of DO/UNTIL Statements to print numbers less than 100:

```
10 PRINT 1: PRINT
20 PRINT 2
30 I=3                          :REM I IS NUMBER TESTED
40 DO
50 J=I/2                        :REM J IS THE LIMIT
60 N=1                          :REM N IS THE FACTOR
70 DO                           :REM SEEKS A DIVISIBLE FACTOR OF I
80 N=N+2
90 UNTIL (MOD(I,N)=0 OR (N>J))
100 IF N>J PRINT I              :REM NO DIVISIBLE FACTOR
110 I=I+2
120 UNTIL (I>100)               :REM ENDS THE SEARCH
```

By enclosing a zero or more statements between the DO and the UNTIL <condition> statement (where the <condition> is any arbitrary expression), the statements between will be repeated as a group until the <condition> evaluates to a non-zero number (a true condition). DO/UNTIL loops can be nested, and NSC Tiny BASIC will report an error if the nesting level becomes too deep, (more than eight levels).

2.3.8 FOR/NEXT Statements

These statements are identical to the FOR/NEXT Statements in standard BASICs. The STEP in the FOR statement is optional. If it is not included, a STEP value of +1 is assumed. The value of the STEP may be either positive or negative. Starting and ending values of the FOR/NEXT loop are included in the FOR statement. The loop is repeated when the NEXT statement has been executed provided the upper limit of the FOR statement has not been reached. When the upper limit is reached, the program will exist from the FOR/NEXT loop. NSC Tiny BASIC causes an error break if the variable in the NEXT statement is not the same variable as that used in the FOR statement.

FOR/NEXT loops may be nested, and NSC Tiny BASIC will report an error if the nesting level becomes too deep; a depth of four levels of FOR loop nesting is allowed. A FOR loop will be executed at least once, even if the initial value of the control variable already exceeds its bounds before starting. The following program would do nothing but print the odd integers less than 100.

```
10 N=100                        :REM UPPER LIMIT
20 FOR I=1 TO N STEP 2          :REM START AT 1 WITH STEP OF 2
30 PRINT I                      :REM PRINT A NUMBER
40 NEXT I                       :REM REPEAT (at Line 20)
```

2.3.9 LINK Statement

Control may be transferred from an NSC Tiny BASIC program to an INS 8073 machine language routine by means of a LINK Statement. This allows the user to make use of routines which may be more efficiently performed in machine language. A statement of the form LINK <address> will cause control to be transferred to the INS8073 machine language routine starting at <address>. Control is transferred by execution of a JSR instruction. The pointers may be modified by the routine. P3's value is unpredictable, and P2 points at the start of A-Z variable storage. Variables are stored in alphabetically ascending order, two bytes each, low order byte first then high-order byte.

Example:

```
>10 LINK #1800                          NSC Tiny BASIC transfers to
>20 IF A=0 THEN PR "SENSE A IS LOW"        address #1800 to read
>30 IF A=1 THEN PR "SENSE A IS HIGH"       sensor.
>99 STOP                                Program transfers back to
>RUN                                       NSC Tiny BASIC
SENSE A IS HIGH

STOP AT 99
>RUN
SENSE A IS LOW

STOP AT 99
```

```
1              .TITLE SENSE
2 0000         .=01800              ;HEXADECIMAL
3 1800 06      LD    A,S
4 1801 D410    AND   A,=16
5 1803 6C02    BZ    LOW
6 1805 C401    LD    A,=1
7 1807 CA00    ST    A,0,P2         ;STORES ACCUMULATOR INTO LOCATION
8 1809 5C      RET                      OF VARIABLE A
9       0000   .END
```

2.3.10 ON Statement

This statement is used for processing interrupts. The format of the statement is:

ON interrupt-#1, line-number

When numbered interrupt (interrupt-#) occurs, NSC Tiny BASIC executes a GOSUB statement beginning at line number "line-number". If "line-number" is zero, the corresponding interrupt is disabled at the software level. Interrupt numbers may be 1 or 2. Use of the ON statement disables console interrupts (BREAK function). Interrupts must also be enabled at the hardware level by setting the Interrupt Enable bit in the status register (for example, using STAT=1).

2.3.11 STOP Statement

Although the last line of a program does not need to be a STOP statement, it is a useful debugging tool for programs. The STOP statement may be inserted as breakpoints in an NSC Tiny BASIC program.

When NSC Tiny BASIC encounters a STOP statement, it prints a stop message and the current line number. It then returns to the edit mode. Thus, the programmer can see whether his program reached the desired point. Any number of STOP statements may appear in the program. By removing the STOP statements, one by one, a program can be tested by parts until the debugging process is completed.

Execution of a stopped program may be continued after the STOP by a CONT (continue) command.

2.3.12 DELAY Statement

This statement delays NSC Tiny BASIC for "expr" time units (nominally milliseconds, 1-1040). Delay 0 gives the maximum delay of 1040 milliseconds. The format is:

```
DELAY expr
```

Example:

```
>10 DELAY 100          Delay 100 milliseconds.
```

2.3.13 CLEAR Statement

This statement initializes all variables to 0, disables interrups, enables BREAK capability from the console, and resets all stacks (GOSUB, FOR-NEXT, DO-UNTIL).

Example:

```
>10 ON (2,250)         Break is disabled, Interrupt 2 is enabled.
 .
 .
 .
 .
 .
>300 CLEAR             Break is re-enabled, Interrupt 2 is disabled.
```

2.4 Indirect Operator

The Indirect Operator is an NSC Tiny BASIC exclusive, at least in the realm of BASIC. It accomplishes the functions of PEEK and POKE with a less cumbersome syntax. The Indirect Operator is a way to access absolute memory location although its applications are not limited to that. Its utility is especially significant for microprocessors, such as the INS8073, where interfacing is commonly performed through memory addressing.

An "at" sign (@) which preceeds a constant, a variable or an expression in parentheses causes that constant, variable or expression to be used as an unsigned 16-bit address at which the value is to be obtained or stored. Thus, if an input device has an address of #6800 (hexadecimal), the statement:

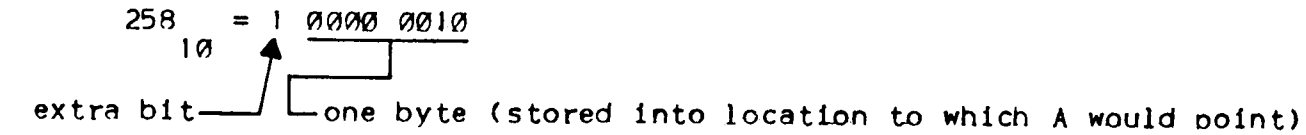
```
LET X=@#6800
```

would input from that device and assign the value of the input to the variable X. If the address of an output device was #6801, the statement:

```
@#6801=Y
```

would output the least significant byte of Y to the device.

The indirect operator accessing memory locations only one byte at a time. An assignment such as @A=248 changes the memory location pointed to by A to 248 (1111 1000) binary, since 248 can be expressed as one byte. However, an assignment such as @A=258 changes the memory location pointed to by A to 2 because the value of 258 cannot be expressed by a single byte, as shown below:



Only the least significant byte of 258 (which is 2) is stored at that location. The extra bit would be lost forever.

Any place that a variable, such as B, would be legal, the construct "@B" would also be legal. The meaning of @B is: the byte located at the memory location whose address is the value of B. Other examples:

```
40 LET B=6000          Assigns 6000 to B.
50 LET @ B=100         Stores decimal 100 in memory location 6000.
60 LET C=@B            Sets C=to 100.
70 PRINT @6000         Prints 100.
80 LET D=@(A+10*D)     Sets D=the value stored in memory location
                       (A+10*D).
```

Parentheses are required when applying @ to an expression.

2.5 Multiple Statements On A Line

More than one statement can be placed on one program line. This is accomplished by placing a colon between the statements. Readability of the program can be improved, and memory space can be saved by using this technique. As an example of the use of multiple statements:

```
200 PRINT "MY GUESS IS",Y:PRINT "INPUT A POSITIVE NUMBER";:
          INPUT X:IF X <=0 GO TO 200
```

If X is negative or zero, the user will be instructed to enter a positive number, and the program returns to Line 200 for a new guess. If the user had entered a positive number correctly, the program would have proceeded to the next numbered line after Line 200.

Care in use of multiple statements per line must be exercised. The above example shows that if the condition of the IF STATEMENT is false, control is passed to the next program line. Anything else on the line containing multiple statements will be ignored.

2.6 String Handling

String input may be accomplished by executing a statement of the form INPUT $ F, where F is a Factor syntactically (see Grammar). When the program reaches this statement during program execution, NSC Tiny BASIC prompts the user with a question mark (?). All line editing characters may be used (back space, line delete, etc.). If a control-U is typed to delete an entered line, NSC Tiny BASIC will continue to prompt for a line until a line is terminated by a carriage return. The line is stored in consecutive locations starting at the address pointed to by F, up to and including the carriage return. Example:

```
20 INPUT $ A    may also be written    20 INPUT $A
```

and inputs a string to successive memory locations starting at A.

2.6.1 String Output

An item in a PRINT statement can include a string variable in the form of $B, where B is a factor. When the print statement is encountered during program execution, the string will be printed beginning at the address B up to, but not including, a carriage return. A keyboard interrupt will also terminate the printing of the string if detected before the carriage return. Example:

```
50 PRINT $B    prints the string beginning at the location
                      pointed to by "B".
```

2.6.2 String Assignment

String variables can be assigned to characters in quotes just as other variables are assigned numerical values. A statement of the form $C= "THIS STRING IS A STRING" (when encountered during program execution) would cause the characters in quotes to be stored in memory starting at the address indicated by C up to and including the carriage return. Example:

```
70 $D="THIS IS A STRING WITH NO INPUT STATEMENT."
A "T" is stored at location "D", and H at location "D+1" etc.
```

2.6.3 String Move

Strings can be moved from one memory block to another memory block using this feature. A statement of the form $A=$B (where A and B are Factors) will transfer the characters in memory beginning with the address B to the memory beginning with address A. The last character, normally a carriage return, is also copied. Also, a statement such as $(A+1)=$A would be disasterous since it causes the entire contents of the RAM to be filled with the first character of $A.

2.6.4 String Examples

```
10 A=TOP                 :REM A POINTS TO EMPTY RAM ABOVE TOP OF
                          PROGRAM
20 C=TOP+100             :REM C POINTS TO RAM 100 BYTES ABOVE A
30 D=TOP+200             :REM D POINTS TO RAM 100 BYTES ABOVE C
40 INPUT $A              :REM STORES CHARACTERS WHERE A POINTS
50 PRINT $A
60 LET $C= "IS THE STRING INPUT AT LINE 10"
70 $D=$C                 :REM STORES CHARACTERS WHERE D POINTS
80 PRINT $D
```

2.7 Commands

2.7.1 NEW expr

This command establishes a new start-of-program address equal to the value of "expr". NSC Tiny BASIC then executes its initialization sequence which clears all variables, resets all hardware/software stacks, disables interrupts, enables BREAK capability from the console, and performs the nondestructive RAM search described in part one. If the value of "expr" points to a ROM address, the NSC Tiny BASIC program which begins at this address will be automatically executed. Program memory (including the end-of-program pointer used by the editor) is not altered by this command.

Example:

```
>NEW 1000
```

NEW used without an argument sets the end-of-program pointer equal to the start-of-program pointer so that a new program may be entered. If a program already exists at the start-of-program address, it will be lost.

Example:

```
>NEW 1000        Sets program pointer to 1000
 NEW             Sets end-of-program pointer to 1000
```

2.7.2 RUN

Runs the current program.

Example:

```
>RUN     Execution begins at lowest line number
```

2.7.3 CONT

Continues execution of the current program from the point where execution was suspended (via a STOP, console interrupt, or reset). An NSC Tiny BASIC program that is executing can be interrupted by pressing the BREAK or RESET keys on the keyboard. Execution can be resumed by entering the CONT command.

Example:

```
>RUN
 THIS IS THE STRING INPUT AT LINE 10
 THIS IS THE STRING INPUT AT LINE 10
 THIS IS THE STRING INPUT AT LINE 10
 THIS IS THE STRING INPUT AT LINE 10  Press  BREAK or RESET.
^C
>CONT
 THIS IS THE STRING INPUT AT LINE 10
 THIS IS THE STRING INPUT AT LINE 10
 And so on...
```

2.7.4 LIST (expr)

Lists the current program (optionally starting at the line number specified by (expr)).

Example:

```
>LIST 10

10 INPUT $A
20 PRINT $A
30 LET $C="IS THE STRING INPUT AT LINE 10"
40 $D=$C
50 PRINT $D
```