An Introduction to NSC Tiny BASIC

The Language of the INS8073

Jim Handy
National Semiconductor
2900 Semiconductor Dr.
Santa Clara CA 95051
408-737-4613

National Semiconductor's new INS8073 Microinterpreter is the first single IC (integrated circuit) that can be programmed in a high-level language. Since a description of the hardware in INS8073-based systems has already appeared elsewhere (see "On-Chip Tiny BASIC Dumps Development Systems," *Electronic Design*, November 22, 1980, page 235), this article will focus on the language of the new chip. Called NSC Tiny BASIC, it resembles other Tiny BASIC interpreters but has certain enhancements, which I'll describe in detail. (See table 1 for a summary of NSC Tiny BASIC features.) Especially noteworthy are NSC Tiny BASIC features that make interrupts and input/output (I/O) routines easy to handle and others that make provisions for multiprocessing.

NSC Tiny BASIC offers advantages to both the inexperienced and the experienced programmer. Using NSC Tiny BASIC, the inexperienced programmer can write comprehensive programs in much less time than equivalent assembly-language programs would require. And since NSC Tiny BASIC lets you jump back and forth between machine code and BASIC, the experienced programmer can write less important routines in BASIC while still using assembly-language for critical routines.

About the Author
At the time he wrote this article, Jim Handy was product marketing engineer for single-chip microcomputers at National Semiconductor Corporation. He is now employed by Digital Research.

Language Expressions

NSC Tiny BASIC permits the use of 26 variable names: the letters A through Z. The values assigned to these variables are 16-bit signed integers. Fractions or floating-point numbers are not allowed.

All numeric constants are decimal numbers except when preceded by a number sign (#), which indicates a hexadecimal number. For example, "55" would be treated as a decimal number, while "#55" would be treated as hexadecimal (equal to 85 decimal). Decimal constants must be in the range of −32767 to +32767. 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 0 (FALSE) or −1 (TRUE) as a result. Note that >< is an illegal operator.

Standard arithmetic operators are provided for the four basic arithmetic functions (+ addition; − subtraction; / division; * multiplication). The arithmetic is standard 16-bit two's-complement arithmetic. Fractional quotients are truncated at the right, not rounded, and remainders are dropped; therefore, both 16/3 and 17/3 give 5 as a result. As usual, division by zero is not permitted; it will result in an error break.

NSC Tiny BASIC follows the usual algebraic rules for order in evaluating expressions. Parentheses control the order of evaluation, and you should use them liberally. They insure clarity in complicated expressions.

NSC Tiny BASIC provides logical operators AND, OR, and NOT. 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). Here's an example of the use of AND, with the binary interpretation shown at right:

```
>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
```

The following program shows how the logical AND can be used with other relational operators:

```
>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
```

If we run the program, console output will look like this:

```
>RUN
? 51
? 52
BOTH ARE BIG
? 51
```

Expressions	
Variable Names A—Z	
Decimal constants in the range −32767 to +32767	
Hexadecimal numbers (preceded by a "#")	
Operators	
Relational Operators	
=	
>	
<	
>=	
<>	
Arithmetic Operators	
+	
−	
/	
*	
Logical Operators	
AND	
OR	
NOT	
The Indirect Operator	
@	Accesses absolute memory locations one byte at a time; e.g., LET X= @6800 assigns value at address 6800 to X.
Functions	
MOD (a,b)	Yields absolute value of remainder from division of a by b
RND (a,b)	Generates pseudorandom integer in range from a through b, inclusive
STAT	Represents 8-bit value of the status register
TOP	Yields address of first byte above the program in the current page of RAM
INC	Increments a memory location
DEC	Decrements a memory location
Statements	
Input/Output Statements	
INPUT	Inputs one or more expressions or numbers, separated by commas (or spaces)
INPUT $ A	Inputs a string to successive memory locations starting at location A
PRINT	Outputs information from program; gives decimal numbers and quoted strings
PRINT $ B	Prints string beginning at address B up to next carriage rturn
Assignment Statements	
LET	Sets a variable, memory location, string variable or the status register to value entered
Program-Control Statements	
GO TO	Branches to line number
GOSUB	Calls subroutine at line number
RETURN	Returns control from subroutine to line following GOSUB statement
IF/THEN	Shifts program control based on result of logical test
DO/UNTIL	Causes repetition of enclosed statements until specified condition is met
FOR/NEXT	Causes repetition for specified number of times
STEP	Sets size of increment in iterations of FOR/NEXT statements
LINK	Transfers control to a machine-language routine at a specified address
ON	Helps process interrupts; "ON interrupt-#1, 200" transfers control to GOSUB statement at line 200 when specified interrupt occurs.
STOP	Causes program to stop and prints current line number, then returns to edit mode
CONT	Resumes execution of program stopped by STOP
DELAY	Delays execution for specified number of time units up to maximum of 1040 milliseconds
CLEAR	Initializes all variables to 0, disables interrupts, enables BREAK capability, resets all stacks
Commands	
NEW	Establishes a new start-of-program address
RUN	Runs the current program
LIST	Lists the current program

Table 1: *A summary of NSC Tiny BASIC features.*

```
? 49
ONE OR BOTH ARE SMALL
? 49
? 49
ONE OR BOTH ARE SMALL
? ^C
STOP AT 10
>
```

The following similar program shows the use of a logical OR with other relational operators:

```
>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
```

Here's a sample run:

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

The third logical operator (NOT) is unary; i.e., it performs an operation on only one argument. An example follows, again showing the binary interpretation at right:

```
>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
```

Tiny BASIC Functions

NSC Tiny BASIC offers several functions for use in arithmetic expressions. For example, the MOD (or modulo) function returns the absolute value of the remainder from the division 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. Here's an example of the use of MOD:

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

The NSC Tiny BASIC random-number generator is called RND, and it returns a pseudo-random integer in the range from a through b, inclusive. For the RND function to perform correctly, a should be less than b, and b−a must be less than or equal to 32767 (decimal). A typical example is:

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

The STAT function returns the 8-bit value of the INS8073 status register. STAT may appear on either side of an assignment statement, enabling you to modify the status register as well as read it. The carry and overflow flags of the status register are usually meaningless, because the NSC Tiny BASIC interpreter itself is continually modifying these flags. The interrupt-enable flag may be altered by an assignment to STAT (e.g., STAT = #FF). Locations of individual flags in the status register are shown in

Reliable Business Software

DATASMITH software requires no previous computer experience, so it can be used effectively by your present office staff. The menu-driven systems feature extensive error detection and correction facilities, so they are "friendly" to the user.

- GENERAL LEDGER. Everything you need to keep the books. Features easy-to-use data entry and error correction, trial balance, fast post, and a variety of comprehensive reports. Automatic error detection keeps the books in balance. Writes checks and makes journal entries in one operation.
- PAYROLL. A very flexible system that adapts to a wide variety of needs. Features Federal, state, and local tax calculations, EI credit, and special pay and deduction amounts. Prints all necessary reports, paychecks, and W-2 forms.
- DATA MANAGER. A powerful generalized data management system that lets you define, enter, update, sort, select, and print reports from a database of your own design. Applicable to almost any job where records must be kept, this system can replace literally hundreds of programs.

Put your computer to work with these sophisticated systems now. Programs are available for 48K or larger two-disk systems in your choice of code for Microsoft BASIC-80® under CP/M® or Micropolis® BASIC. Write or call for complete details. Customized systems also available.

DATASMITH

Box 8036, Shawnee Mission, KS 66208, (913) 381-9118

Circle 138 on inquiry card.

table 2. The function of each bit in the status register is shown in table 3. Here is an example of the use of the STAT function:

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

Other Functions

The TOP function returns the address of the first byte of RAM (random-access read/write memory) above the NSC Tiny BASIC program that is available to the user. This will be the address of the highest byte in the NSC Tiny BASIC program, plus 1. A program can use all the memory in the RAM above and including TOP as scratchpad storage. As an example:

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

The INC and DEC functions increment or decrement a memory location X. Here are some examples:

```
10 LET X=1032
20 A=INC(X)
   .
   .
   .
50 B=DEC(X)
60 X=INC(6000)
70 Y=DEC(6000)
```

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

Statements

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

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

Table 2: *A representation of the INS8073 status register. Bits 5 through 0 can be either read or set from NSC Tiny BASIC.*

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

When the statement at line 10 is executed, NSC Tiny BASIC prompts you with a question mark. You type in a number, which is assigned to the variable A after you press the RETURN key. NSC Tiny BASIC then prompts

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 subtract, or any instruction that alters the status register. 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.
6	OVERFLOW (OV): This bit is set if an arithmetic overflow occurs during a machine-language add or subtract instruction.
	NOTE: Bit 7 and bit 6 may be of little or no use in an NSC Tiny BASIC program.
5	SENSE BIT B (SB): Tied to an external connector pin, this bit can be used to sense external conditions. This is a "read-only" bit; it is not affected when the contents of the accumulator are copied into the status register by a STAT instruction. Sense bit B is also the second interrupt input and may be examined by use of the "ON" command.
4	SENSE BIT A (SA): Like sense bit B, this bit is tied to an external connector pin and serves to sense external conditions. In addition, sense bit A acts as the interrupt input when the INTERRUPT ENABLE (see bit 3 of status register below) is set. This bit is also a "read-only" bit and can be examined with the "ON" command. NSC Tiny BASIC uses this bit 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): Similar to F3. This flag is used by NSC Tiny BASIC to control the paper-tape reader relay.
1	USER FLAG 1 (F1): Similar to F3. This flag is used by NSC Tiny BASIC as the serial-output bit (with inverted data) to the TTY or CRT.
	NOTE: The outputs of flags 1, 2, and 3 of the status register serve as latched flags. These flags are set to the specified state when the contents of the status register are modified by an assignment to STAT, and remain in that state until the contents of the status register are modified under program control.
0	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 the interrupt-enable flag is set, NSC Tiny BASIC recognizes external interrupt requests received via the SENSE A or B inputs. When reset, this interrupt-enable flag prevents the INS8073 from recognizing interrupt requests.

Table 3: *A summary of the function of each bit in the INS8073 status register.*

you with another question mark, and you type in two numbers, separated by commas. These numbers will be assigned to B and C in that order. A sample run follows:

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

NSC Tiny BASIC would now continue execution of the program.

NSC Tiny BASIC accepts expressions as well as numbers in response to an INPUT request. For example:

```
>LIST
  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 you can type the expressions correctly.

NSC Tiny BASIC allows string input, as described in the section on string handling, found later in this article. INPUT may not be used in the command mode.

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. A space precedes positive numbers, and a minus sign precedes negative numbers. All numbers have a trailing space. 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. A typical example:

```
>LIST
  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
```

The word LET may be used or omitted in an assignment statement, but 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 through Z), STAT, or a memory location, which is indicated by an @ followed by a variable, a number, or an expression in parentheses. Here are some sample assignments:

```
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 shown here:

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

This statement would assign the absolute value of A to X.

Program Control

NSC Tiny BASIC provides an assortment of program-control statements. The GO TO statement permits program branches to a specific line number or a line number called by an arbitrary expression. For example,

```
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. The value of X is variable, allowing dynamic control of program execution at this point.

The GOSUB and RETURN statements are useful when a computation or operation must be performed at more than one place in a program. Rather than write the routine over again each time it is needed, you employ a GOSUB instruction 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. An example is shown in figure 1. GOSUBs may be nested up to eight levels deep (including interrupt levels).

The IF. . .THEN statement allows 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 (nonzero), the THEN portion of the statement will be executed. When the test condition is false (zero), the THEN portion will be ignored and execution will continue at the next numbered line of the program. For example:

```
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 previous example would become:

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

The DO. . .UNTIL statement is unique to NSC Tiny BASIC. Borrowed from Pascal, this statement is used to program loops, thus keeping GO TO statements to a minimum. The DO. . .UNTIL statement makes NSC Tiny BASIC programs clear in structure and easy to read. 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=1/2            :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 0 or more statements between the DO and the UNTIL <condition> statement (where the <condition> is any arbitrary expression), you cause repetition of the enclosed statements as a group until the <condition> evaluates to a nonzero 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).

The FOR. . .NEXT statement in NSC Tiny BASIC is identical to the FOR. . .NEXT statement in standard

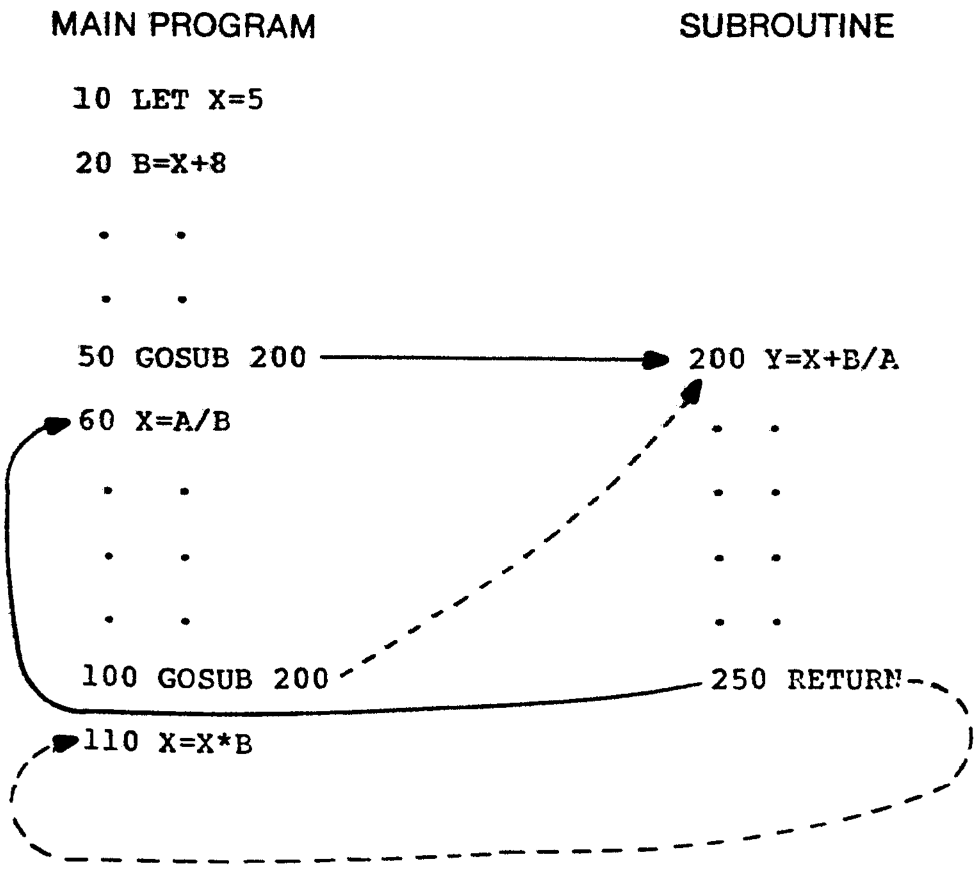


Figure 1: *The effects of the GOSUB and RETURN statements in NSC Tiny BASIC. On the first GOSUB call (line 50), the order of execution follows the solid arrows. On the second GOSUB call (line 100), the order of execution follows the dashed arrows.*

BASICs. A STEP function in the FOR statement may be used to specify the size of the increment in each iteration of the statement. In the absence of a specified STEP, NSC Tiny BASIC assumes a STEP value of +1. 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 exit 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 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)
                       20)
```

When increased execution speed is needed, you can use a LINK statement to transfer control from an NSC Tiny BASIC program to an INS8073 machine-language routine. A statement of the form LINK <address> will cause transfer of control to the INS8073 machine-language routine, starting at the specified address. Control is transferred by execution of a JSR (Jump to Subroutine) instruction.

The INS8073 has two address pointers, P2 and P3, in addition to the program counter and the stack pointer. When a LINK statement transfers control to a machine-language routine, the routine can modify the pointers P2 and P3. The value of pointer P3 is unpredictable; P2 points at the starting location of the storage of A through Z variables. These variables are stored in ascending alphabetical order, two bytes each, low-order byte first. Here is an example:

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

The program above requires the machine-language program described below to be loaded into address 1800 hexadecimal.

```
1              .TITLE SENSE     ;THIS PROGRAM
                                READS SENSE A PIN
2 0000         .=01800
3 1800 06      LD   A,S         ;TINY BASIC JUMPS
                                HERE
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 AC-
                                CUMULATOR INTO
                                LOCATION OF
                                VARIABLE A
8 1809 5C      RET              ;RETURN TO TINY
                                BASIC
9      0000    .END
```

The ON statement helps process interrupts. The format of the statement is:

ON interrupt—#, line-number

When the numbered interrupt (interrupt—#) occurs, NSC Tiny BASIC executes a GOSUB statement beginning at the line number given. If the line-number given 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 (e.g., using STAT=1).

Although the last line of a program does not have to be a STOP statement, use of a STOP in this way does help in debugging. The STOP statement may be inserted as a breakpoint in an NSC Tiny BASIC program. On encountering a STOP statement, NSC Tiny BASIC prints a stop message and the current line number, then returns to the edit mode. Thus, you can see whether your program has reached the desired point. Any number of STOP statements may appear in the program. By removing the STOP statements one by one, you can test the program in parts until debugging is completed. Execution of a stopped program may be continued after the STOP by a CONT (continue) command.

Other Useful Features

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

DELAY expr

For example:

```
>10 DELAY 100        Delay 100 ms.
```

The CLEAR statement initializes all variables to 0, disables interrupts, enables BREAK capability from the console, and resets all stacks (GOSUB, FOR. . .NEXT, and DO. . .UNTIL). For example:

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

The indirect operator is an NSC Tiny BASIC exclusive, at least in the realm of BASIC. This operator performs the functions of PEEK and POKE with a less cumbersome syntax. The indirect operator can access absolute memory locations and can service input/output devices as well. Its utility is especially significant for microprocessors like the INS8073, for which interfacing is commonly performed through memory addressing.

When an "at" sign (@) precedes a constant, a variable, or an expression in parentheses, that constant, variable, or expression is taken as an unsigned 16-bit address at which a 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 accesses memory locations only one byte at a time. An assignment such as @A=248 changes the memory location pointed to by A to 248 binary (1111 1000), 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 expressing the value 258 causes a carry to a ninth bit, which is lost, as shown below:

```
258₁₀ = 1     0000  0010
      extra   one byte (stored into location to
       bit    which A would point)
```

Only the least significant byte of 258 (which is 2) is stored at the location to which A would point.

In any place where a variable, such as B, would be legal, the construct "@B" (which means the byte located at the memory location whose address is the value of B) would also be legal. Here are some 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 equal to 100.
70 PRINT @6000         Prints 100.
80 LET D=@(A+10*D)     Sets D equal to the value
                       stored in memory location
                       (A+10*D).
```

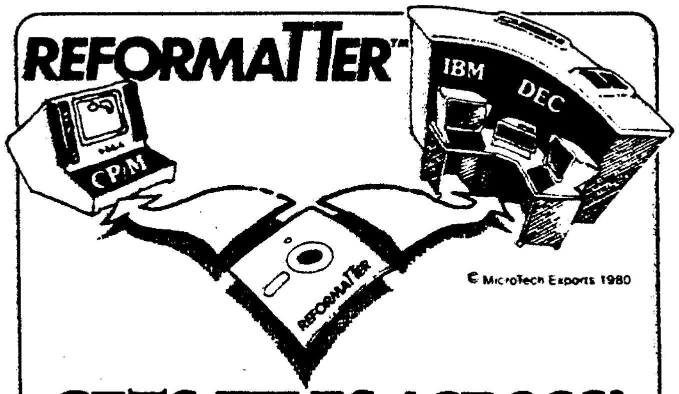
Parentheses are required when @ is applied to an expression.

More than one statement can be placed on one program line by placing a colon between the statements. This technique can improve readability of the program and can save memory space. Here is 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, you will be instructed to enter a positive number, and the program will return to line 200 for a new guess. If you had entered a positive number correctly, the program would have proceeded to the next numbered line after line 200.

You must use multiple statements per line with care. The above example shows that if the condition of the IF



GETS FILES ACROSS!

With ***REFORMATTER*** disk utilities you can read and write IBM 3740 and DEC RT-11 single density formatted diskettes on your CP/M® system.

REFORMATTER enables you to access large system databases, improve data exchange with other organizations, increase program development capabilities, and use your micro in distributed processing.

REFORMATTER programs feature bi-directional data transfer and full directory manipulation. ASCII/EBCDIC conversion provided with CP/M ↔ IBM. *MP/M is now fully supported.*

Program Data Sheets, Application Guides, and Machine Compatibility Guides available.

Each program $195.00 from stock. Specify CP/M ↔ IBM or CP/M ↔ DEC. Order from MicroTech Exports, Inc., 467 Hamilton Ave., Suite 2, Palo Alto, CA 94301 □ Tel: 415/324-9114 □ TWX: 910-370-7457 MUH-ALTOS □ Dealer and OEM discounts available.

CP/M® is a registered trademark of Digital Research

Circle 276 on inquiry card.

statement is false, control passes to the next program line. Anything else on the line containing multiple statements will be ignored.

String Handling

To input string data, a statement of the form

```
INPUT $ F
```

where F is a starting address, is used. When the program reaches this statement during program execution, NSC Tiny BASIC prompts you 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. For example,

```
20 INPUT $ A
```

may also be written

```
20 INPUT $A
```

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

An item in a PRINT statement can include a string variable in the form of $B. 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 the interrupt is detected before the carriage return. For example,

```
50 PRINT $B
```

prints the string beginning at the location pointed to by "B".

Characters in quotes can be assigned to string variables just as numerical values are assigned to other variables. 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 and going up to and including the carriage return at the end of the line. For example:

```
70 $D="THIS IS A STRING WITH NO
   INPUT STATEMENT."
```

A T is stored at location "D", an H at location "D+1", etc.

Strings can be moved from one memory block to another. A statement of the form

```
$A=$B
```

(where A and B are addresses) will transfer the characters in memory beginning with address B to memory beginning with address A. The last character, normally a carriage return, is also copied. Note that a statement such as

```
$(A+1)=$A
```

would be disastrous, because it fills all of RAM with the first character of $A. Here is an example of moving one memory block to another location:

```
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
```

Commands

The NEW 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 a nondestructive search of RAM. If the value of "expr" points to a ROM (read-only memory) address, the NSC Tiny BASIC program that begins at that address will be automatically executed. The NEW command does not alter memory (including the end-of-program pointer used by the editor). For 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. For example:

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

The RUN command runs the current program. For example:

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

The CONT (continue) command 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. For 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...
```

The LIST(expr) command lists the current program (optionally starting at the line number specified by "expr"). For 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
```

Conclusion

NSC Tiny BASIC and the INS8073 Microinterpreter Chip offer many advantages to the programmer. NSC Tiny BASIC's indirect operator represents a substantial improvement over the usual PEEK and POKE statements. The DO . . . UNTIL statement brings the advantages of structured programming into the realm of Tiny BASICs for the first time. These and other advanced features of NSC Tiny BASIC offer you the convenience of a high-level language as well as new possibilities for elegance and efficiency in process-control and other applications often reserved for assembly language.

Furthermore, with NSC Tiny BASIC and the INS8073, transferring programs from RAM to ROM is remarkably simple. Because the INS8073 executes ASCII (American Standard Code for Information Interchange) data, if the program will run in RAM, it will run in ROM. You don't have to put anything in ROM except what you put on paper.

Programmers have already used the INS8073 and NSC Tiny BASIC for a wide variety of applications, including precision measurement of conditions in oil wells and testing the feasibility of the digital design of an FM tuner. In the coming years, the INS8073 and NSC Tiny BASIC will simplify many other complex tasks.■

CATCH THE S-100 INC. BUS!

MEMORY EXTRAVAGANZA	LIST PRICE	OUR SPECIAL CASH PRICE
GODBOUT - RAM-17-64K STATIC A&T	795.00	579.00
MORROW DESIGNS - 65K STATIC A&T	775.00	579.00
MEMORY MERCHANT 16K STATIC A&T	179.00	149.00
SEATTLE COMPUTER PRODUCTS 8/16 RAM - STATIC 64K A&T	995.00	795.00
NORTH STAR RAM - 32 DYNAMIC A&T	739.00	399.00
SSM MB64 64K STATIC A&T	849.00	695.00
CCS 64K DYNAMIC A&T	750.00	519.00

Subject to Available Quantities • Prices Quoted Include Cash Discounts. Shipping & Insurance Extra.

S-100, inc.

14425 North 79th Street, Suite B
Scottsdale, Arizona 85260
Order Number 800-528-3138
Technical 602-991-7870

CAN'T MAKE ENDS MEET?

We couldn't either. At least, not until 1978 when we began connecting RS-232 devices to IEEE-488 computers with our family of serial interfaces. Three units available for use with Commodore's PET/CBM, the HP-85, Osborne-1 and others-plus our IEEE-488 auto-answer/auto-dial 103 Modem and data communications software. Each interface complete with cabinet, one-year warranty, documentation. Priced from $129-389. Details from George Masters:

TNW CORPORATION

Dept. B., 3444 Hancock St., San Diego, CA 92110
(714) 296-2115 • TWX 910-335-1194
VISA/MasterCard • Dealer Inquiries Welcome