nascom

Nascom 2 Microcomputer
DOCUMENTATION

Nascom Microcomputers
Division of Lucas Logic Limited
Welton Road Wedgnock Industrial Estate
Warwick CV34 5PZ
Tel: 0926 497733 Telex: 312333

Lucas Logic

PF/019

CONTENTS

1. INTRODUCTION

Nascom 8K Basic is based on Microsoft Basic, which has become the industry standard, and offers a high degree of compatability with programs published in books and magazines.

It offers 7-digit floating point numbers in the range 1.70141E38 to 2.9387E-38, full trigonometric functions, string handling and PIO control. User functions can be written in machine code or Basic to provide additional flexibility.

In addition, a number of extra features have been included:

- it works with Nasbug T2, T4 and Nas-sys
- with Nas-sys, provides powerful on screen, in line editing facilities for data and programs.
- it has clear screen and cursor positioning functions for screen formatting.
- improved LIST command for use with Nascom display.
- support of printers or terminals attached e.g. via serial interface.
- improved cassette handling with program identification and error checking of both programs and data (many other systems can allow data to be misread)
- support of the Nascom graphics options using SET, RESET and POINT commands

1-1 Introduction to this manual

This manual describes the features offered by Nascom 8K Basic. It is not intended to be used as an introduction to programming in Basic - many such books are available elsewhere (see Appendix H)

a. Conventions. For the sake of simplicity, some conventions will be followed in discussing the features of the Nascom BASIC language.

1. Words printed in capital letters must be written exactly as shown. These are mostly names of instructions and commands.

2. Items enclosed in angle brackets (< >) must be supplied as explained in the text. Items in square brackets ([ ]) are optional. Items in both kinds of brackets, [<W>], for example, are to be supplied if the optional feature is used. Items followed by dots (...) may be repeated or deleted as necessary.

3. Shift/ or Control/ followed by a letter means the character is typed by holding down the Shift or Control key and typing the indicated letter.

4. All indicated punctuation must be supplied.

b. Definitions. Some terms which will become important are as follows:

Alphanumeric character: All letters and numerals taken together are called alphanumeric characters.

Enter, Newline or Carriage Return: Refers both to the key on the terminal which causes the carriage, print head or cursor to move to the beginning of the next line and to the command that the return key issues which terminates a BASIC line.

Command Level: After BASIC prints OK, it is at the command level. This means it is ready to accept commands.

Commands and Statements: Instructions in Nascom BASIC are loosely divided into two clases, Commands and Statements. Commands are instructions normally used only in direct mode (see Modes of Operation, section 1-2). Some commands, such as CONT, may only be used in direct mode since they have no meaning as program statements. Some commands are not normally used as program statements because they cause a return to command level. But most commands will find occasional use as program statements. Statements are instructions that are normally used in indirect mode. Some statements, such as DEF, may only be used in indirect mode.

Edit: The process of deleting, adding and substituting lines in a program and that of preparing data for output according to a predetermined format will both be referred to as "editing". The particular meaning in use will be clear from the context.

Integer Expression: An expression whose value is truncated to an integer. The components of the expression need not be of integer type.

Reserved Words: Some words are reserved by BASIC for use as statements and commands. These are called reserved words and they may not be used in variable or function names.

String Literal: A string of characters enclosed by quotation marks (") which is to be input or output exactly as it appears. The quotation marks are not part of the string literal, nor may a string literal contain quotation marks. (""HI, THERE"" is not legal.)

Type: While the actual device used to enter information into the computer differs from system to system, this manual will use the word "type" to refer to the process of entry. The user types, the computer prints. Type also refers to the classifications of numbers and strings.

1-2 Modes of Operation

Nascom BASIC provides for operation of the computer in two different modes. In the direct mode, the statements or commands are executed as they are entered into the computer. Results of arithmetic and logical operations are displayed and stored for later use, but the instructions themselves are lost after execution. This mode is useful for debugging and for using BASIC in a "calculator" mode for quick computations which do not justify the design and coding of complete programs.

In the indirect mode, the computer executes instructions from a program stored in memory. Program lines are entered into memory if they are preceded by a line number. Execution of the program is initiated by the RUN commands.

1-3 Formats

a. Lines. The line is the fundamental unit of a Nascom BASIC program. The format for a Nascom BASIC line is as follows:

nnnnn <BASIC statement>[:<BASIC statement>...]

Each Nascom BASIC line begins with a number. The number corresponds to the address of the line in memory and indicates the order in which the statements in the line will be executed in the program. It also provides for branching linkages and for editing. Line numbers must be in the range Ø to 65529. A good programming practice is to use an increment of 5 or 10 between successive line numbers to allow for insertions.

Following the line number, one or more BASIC statements are written. The first word of a statement identifies the operations to be performed. The list of arguments which follows the identifying word serves several purposes. It can contain (or refer symbolically to) the data which is to be operated upon by the statement. In some important instructions, the operation to be performed depends upon conditions or options specified in the list.

Each type of statement will be considered in detail in sections 2, 3 and 4.

NB More than one statement can be written on one line if they are separated by colons (:). Any number of statements can be joined this way provided that the line is no more than 72 characters long. When used with Nas-sys in normal mode, lines cannot be greater than 48 characters. However 72 character lines can be inserted by entering Monitor mode, issuing an X0 command and returning to Basic by typing Z.

b. REMarks. In many cases, a program can be more easily understood if it contains remarks and explanations as well as the statements of the program proper. In Nascom BASIC, the REM statement allows such comments to be included without affecting execution of the program. The format of the REM statement is as follows:

REM <remarks>

A REM statement is not executed by BASIC, but branching statements may link into it. REM statements are terminated by the carriage return or the end of the line but not by a colon.

Example:
```
100 REM DO THIS LOOP:FOR I=1TO10   -the FOR statement will not be executed
101 FOR I=1 TO 10: REM DO THIS LOOP-this FOR statement will be executed
```

c. Errors When the BASIC interpreter detects an error that will cause the program to be terminated, it prints an error message. The error message formats in Nascom BASIC are as follows:

Direct statement	?XX ERROR
Indirect statement	?XX ERROR IN nnnnn

XX is the error code or message (see section 6-5 for a list of error codes and messages) and nnnnn is the line number where the error occurred. Each statement has its own particular possible errors in addition to the general errors in syntax. These errors will be discussed in the description of the individual statements.

1-4 Editing - elementary provisions

Editing features are provided in Nascom BASIC so that mistakes can be corrected and features can be added and deleted without affecting the remainder of the program. If necessary, the whole program may be deleted.

The following facilities are available with Nasbug T2 and T4 and Nas-sys in X0 mode (i.e. supporting an external terminal).

a. Correcting single characters. If an incorrect character is detected in a line as it is being typed, it can be corrected immediately with the backspace key. Each stroke of the key deletes the immediately preceding character. If there is no preceding character, a carriage return is issued and a new line is begun. Once the unwanted characters are removed, they can be replaced simply by typing the rest of the line desired.

When RUBOUT (control Z) is typed, the previous character is deleted and echoed. Each successive RUBOUT prints the next character to be deleted. Typing a new character prints the new character.

Example:
```
100 X==X Y=10          Typing two RUBOUTS deleted the '=' and 'X' which
                       were subsequently replaced by Y= .
```

b. Correcting lines. A line being typed may be deleted by typing an at-sign (@) instead of typing a carriage return. A carriage return is printed automatically after the line is deleted. Typing Control/U has the same effect.

c. Correcting whole programs. The NEW command causes the entire current program and all variables to be deleted. NEW is generally used to clear memory space preparatory to entering a new program.

1-5 Program Input with Nas-sys

When used with Nas-sys, the full range of editing facilities are available, and the line displayed on the screen is passed to the Basic interpreter by the Enter or Newline key. In addition to editing the current line, this allows you to list a program, edit lines on the screen and re-enter the updated lines. But note that data input is normally dealt with on a character by character basis as outlined in 1-4, above. (see also 5).

2. STATEMENTS AND EXPRESSIONS

2-1 Expressions

The simplest BASIC expressions are single constants, variables and function calls.

a. Constants. Nascom BASIC accepts integers, floating point real numbers or strings as constants. See section 4-1. Some examples of acceptable numeric constants follow:

```
123
3.141
0.0436
1.25E+05
```

Data input from the terminal or numeric constants in a program may have any number of digits up to the length of a line (see section 1-3a). However, only the first 7 digits of a number are significant and the seventh digit is rounded up. Therefore, the command

```
PRINT 1.234567890123
```

produces the following output:

```
1.23457
OK
```

The format of a printed number is determined by the following rules:

1. If the number is negative, a minus sign (-) is printed to the left of the number. If the number is positive, a space is printed.
2. If the absolute value of the number is an integer in the range 0 to 999999, it is printed as an integer.
3. If the absolute value of the number is greater than or equal to .01 and less than or equal to 999999, it is printed in fixed point notation with no exponent.
4. If the number does not fall into categories 2, or 3 scientific notation is used. The format of scientific notation is as follows:

```
SX.XXXXXESTT
```

Where S stands for the signs of the mantissa and the exponent (they need not be the same, of course), X for the digits of the mantissa and T for the digits of the exponent. E and D may be read "...times ten to the power...." Non-significant zeros are suppressed in the mantissa but two digits are always printed in the exponent. The sign convention in rule 1 is followed for the mantissa. The exponent must be in the range -38 to +38. The largest number that may be represented in Nascom BASIC is 1.70141E38, the smallest positive number is 2.9387E-38. The following are examples of numbers as input and as output by Nascom BASIC:

Number	Nascom BASIC Output
+1	1
-1	-1
6523	6523
1E20	1E20
-12.34567E-10	-1.23456E-09
1.234567E-7	1.23457E-07
1000000	1E+06
.1	.1
.01	.01
.000123	1.23E-04
-25.460	-25.46

In all formats, a space is printed after the number.

b. Variables.

1. A variable represents symbolically any number which is assigned to it. The value of a variable may be assigned explicitly by the programmer or may be assigned as the result of calculations in a program. Before a variable is assigned a value, its value is assumed to be zero. In Nascom BASIC, the variable name may be any length, but any alphanumeric characters after the first two are ignored. The first character must be a letter. No reserved words may appear as variable names or within variable names. The following are examples of legal and illegal BASIC variables:

Legal	Illegal
A	%A (first character must be alphabetic)
Z1	
TP	TO (variable names cannot be reserved words)
PSTG$	
COUNT	RGOTO (variable names cannot contain reserved words)

A variable may also represent a string. Use of this feature is discussed in section 4.

Internally, BASIC handles all numbers in binary. Therefore, some 8 digit single precision numbers may be handled correctly.

c. Array Variables. It is often advantageous to refer to several variables by the same name. In matrix calculations, for example, the computer handles each element of the matrix separately, but it is convenient for the programmer to refer to the whole matrix as a unit. For this purpose, Nascom BASIC provides subscripted variables, or arrays. The form of an array variable is as follows:

```
VV(<subscript>[,<subscript>...])
```

Where VV is a variable name and the subscripts are integer expressions. Subscripts may be enclosed in parentheses or square brackets. An array variable may have as many dimensions as will fit on a single line and can be accommodated in the memory space available. The smallest subscript is zero.

Examples:

A(5)	The sixth element of array A. The first element is A(0).
ARRAY(I,2*J)	The address of this element in a two-dimensional array is determined by evaluating the expressions in parentheses at the time of the reference to the array and truncating to integers. If I=3 and J=2.4, this refers to ARRAY(3,2).

The DIM statement allocates storage for array variables and sets all array elements to zero. **The form** of the DIM statement is as follows:

DIM VV(<subscript>[,<subscript>...])

Where VV is a legal variable name. Subscript is an integer expression which specifies the **largest possible** subscript for that dimension. Each DIM statement may apply to more than **one array** variable. Some examples follow:

```
113 DIM A(3), D$(2,2,2)
114 DIM R2 (4), B(10)
115 DIM Q1(N), Z (2+I)
```

Arrays may be dimensioned dynamically during program execution. At the time the DIM is executed, the expression within the parentheses is evaluated and the results truncated to integer.

If no DIM statement has been executed before an array variable is found in a program, BASIC assumes the variable to have a maximum subscript of 10 (11 elements) for each dimension in the reference. A BS or SUBSCRIPT OUT OF RANGE error message will be issued if an attempt is made to reference an array element which is outside the space allocated in its associated DIM statement. This can occur when the wrong number of dimensions is used in an array element reference. For example:

```
30 LET A(1,2,3)=X  when A has been dimensioned by
10 DIM A(2,2)
```

A DD or REDIMENSIONED ARRAY error occurs when a DIM statement for an array is found after that array has been dimensioned. This often occurs when a DIM statement appears after an array has been given its default dimension of 10.

d. Operators and Precedence. Nascom BASIC provides a full range of arithmetic and logical operators. The order of execution of operations in an expression is always according to their precedence as shown in the table below. The order can be specified explicitly by the use of parentheses in the normal algebraic fashion.

Table of Precedence

Operators are shown here in decreasing order of precedence. Operators listed in the same entry in the table have the same precedence and are executed in order from left to right in an expression.

1. Expressions enclosed in parentheses ()
2. ↑ exponentiation. Any number to the zero power is 1. Zero to a negative power causes a /0 or DIVISION BY ZERO error.
3. - negation, the unary minus operator.
4. *,/ multiplication and division.
5. +,- addition and subtraction
6. relational operators
   - = equal
   - <> not equal
   - < less than
   - > greater than
   - <=,=< less than or equal to
   - >=,=> greater than or equal to
7. NOT logical, bitwise negation
8. AND logical, bitwise disjunction
9. OR logical, bitwise conjunction

Relational operators may be used in any expressions. Relational expressions have the value either of True (-1) or False (0).

e. Logical Operations. Logical operators may be used for bit manipulation and Boolean algebraic functions. The AND, OR, NOT, operators convert their arguments into sixteen bit, signed, two's complement integers in the range -32768 to 32767. After the operations are performed, the result is returned in the same form and range. If the arguments are not in this range, an FC or ILLEGAL FUNCTION CALL error message will be printed and execution will be terminated. Truth tables for the logical operators appear below. The operations are performed bitwise that is, corresponding bits of each argument are examined and the result computed one bit at a time. In binary operations, bit 7 is the most significant bit of a byte and bit 0 is the least significant.

AND

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

NOT

X	NOT X
1	0
0	1

Some examples will serve to show how the logical operations work:

```
63 AND 16=16    63=binary 111111 and 16=binary 10000, so 63 AND 16=16
15 AND 14=14    15=binary 1111 and 14=binary 1110, so 15 AND 14=binary 1110=14
-1 AND 8=8      -1=binary 1111111111111111 and 8=binary 1000, so -1 AND 8=8
4 OR 2=6        4=binary 100 and 2=binary 10 so 4 OR 2=binary 110=6
10 OR 10=10     binary 1010 OR'd with itself is 1010=10
-1 OR -2=-1     -1=binary 1111111111111111 and -2= 1111111111111110, so -1 OR -2=-1
NOT 0=-1        the bit complement of sixteen zeros is sixteen ones, which is the two's
                complement representation of -1.
NOT X=-(X+1)    the two's complement of any number is the bit complement plus one.
```

A typical use of logical operations is 'masking', testing a binary number for some predetermined pattern of bits. Such numbers might come from the computer's input ports and would then reflect the condition of some external device. Further applications of logical operations will be considered in the discussion of the IF statement.

f. The LET statement. The LET statement is used to assign a value to a variable. The form is as follows:

LET ⟨VV⟩=⟨expression⟩

Where VV is a variable name and the expressionis any valid Nascom BASIC arithmetic or logical or string expression. Examples:

```
1000 LET V=X
110 LET I=I+1   the '=' sign here means 'is replaced by ....'
```

The word LET in a LET statement is optional, so algebraic equations such as: 120 V=.5*(X+2) are legal assignment statements.

A SN or SYNTAX ERROR message is printed when BASIC detects incorrect form, illegal characters in a line, incorrect punctuation or missing parentheses. An OV or OVERFLOW error occurs when the result of a calculation is too large to be represented by Nascom BASIC's number formats. All numbers must be within the range 1E-38 to 1.70141E38 or -1E-38 to -1.70141E38. An attempt to divide by zero results in the /0 or DIVISION BY ZERO error message.

For a discussion of strings, string variables and string operations, see section 4.

2-2 Branching, Loops and Subroutines

a. Branching. In addition to the sequential execution of program lines, BASIC provides for changing the order of execution. This provision is called branching and is the basis of programmed decision making and loops. The statements in Nascom BASIC which provide for branching are the GOTO, IF...THEN and ON...GOTO statements.

1. GOTO is an unconditional branch. Its form is as follows:
   GOTO ⟨mmmmm⟩
   After the GOTO statement is executed, execution continues at line number mmmmm
2. IF...THEN is a conditional branch. Its form is as follows:
   IF⟨expression⟩THEN⟨mmmmm⟩

Where the expression is a valid arithmetic, relational or logical expression and mmmmm is a line number. If the expression is evaluated as non-zero, BASIC continues at line mmmmm. Otherwise, execution resumes at the next line after the IF...THEN statement. An alternate form of the IF...THEN statement is as follows:

IF⟨expression⟩THEN⟨statement⟩

Where the statement is any Nascom BASIC statement. Examples:

```
10  IF A=10 THEN 40     If the expression A=10 is true, BASIC branches to line 40.
                        Otherwise, execution proceeds at the next line.
15  IF A<B+C OR X THEN 100  The expression after IF is evaluated and if the value
                        of the expression is non-zero, the statement branches to line 100
                        Otherwise execution continues on the next line.
20  IF X THEN 25        If X is not zero, the statement branches to line 25
30  IF X=Y THEN PRINT X     If the expression X=Y is true (its value is non-zero),
                        the PRINT statement is executed. Otherwise, the PRINT statement is not executed.
                        In either case, execution continues with the line after the IF...THEN statement.
35  IF X=Y+3 GOTO 39    Equivalent to the corresponding IF...THEN statement, except
                        that GOTO must be followed by a line number and not by another statement.
```

3. ON...GOTO provides for another type of conditional branch. Its form is as follows:
   ON⟨expression⟩GOTO⟨list of line numbers⟩

After the value of the expression is truncated to an integer, say I, the statement causes BASIC to branch to the line whose number is Ith in the list. The statement may be followed by as many line numbers as will fit on one line. If I=0 or is greater than the number of lines in the list, execution will continue at the next line after the ON...GOTO statement. I must not be less than zero or greater than 255, or an FC or ILLEGAL FUNCTION CALL error will result.

b. Loops. It is often desirable to perform the same calculations on different data or repetitively on the same data. For this purpose, BASIC provides the FOR and NEXT statements. The form of the FOR statement is as follows:

FOR⟨variable⟩=⟨X⟩TO⟨Y⟩[STEP ⟨Z⟩]

where X,Y and Z are expressions. When the FOR statement is encountered for the first time, the expressions are evaluated. The variable is set to the value of X which is called the initial value. BASIC then executes the statements which follow the FOR statement in the usual manner. When a NEXT statement is encountered, the step Z is added to the variable which is then tested against the final value Y. If Z, the step, is positive and the variable is less than or equal to the final value, or if the step is negative and the variable is greater than or equal to the final value, then BASIC branches back to the statement immediately following the FOR statement. Otherwise, execution proceeds with the statement following the NEXT. If the step is not specified, it is assumed to be 1.

Examples:

```
10 FOR I=2 TO 11        The loop is executed 10 times with the variable I taking on
                        each integral value from 2 to 11.
```

```
20 FOR V=1 TO 9.3                       This loop will execute 9 times until V is
                                        greater than 9.3
30 FOR V=10*N TO 3.4/% STEP SQR(R)      The initial, final and step expressions need
                                        not be integral, but they will be evaluated
                                        only once, before looping begins.
40 FOR V=9 TO 1 STEP -1                 This loop will be executed 9 times.
```

FOR...NEXT loops may be nested. That is, BASIC will execute a FOR...NEXT loop within the context of another loop. An example of two nested loops follows:

```
100 FOR I=1 TO 10
120 FOR J=1 TO 1
130 PRINT A(1,J)
140 NEXT J
150 NEXT I
```

Line 130 will print 1 element of A for I=1, 2 for I=2 and so on. If loops are nested, they must have different loop variable names. The NEXT statement for the inside loop variable (J in the example) must appear before that for the outside variable (I). Any number of levels of nesting is allowed up to the limit of available memory.

The NEXT statement is of the form:

NEXT[⟨variable⟩[,⟨variable⟩...]]

Where each variable is the loop variable of a FOR loop for which the NEXT statement is the end point. NEXT without a variable will match the most recent FOR statement. In the case of nested loops which have the same end point, a single NEXT statement may be used for all of them. The first variable in the list must be that of the most recent loop, the second of the next most recent, and so on. If BASIC encounters a NEXT statement before its corresponding FOR statement has been executed, an NF or NEXT WITHOUT FOR error message is issued and execution is terminated.

c. Subroutines. If the same operation or series of operations are to be performed in several places in a program, storage space requirements and programming time will be minimized by the use of subroutines. A subroutine is a series of statements which are executed in the normal fashion upon being branched to by a GOSUB statement. Execution of the subroutine is terminated by the RETURN statement which branches back to the statement after the most recent GOSUB. The format of the GOSUB statement is as follows:

GOSUB⟨line number⟩

where the line number is that of the first line of the subroutine. A subroutine may be called from more than one place in a program, and a subroutine may contain a call to another subroutine. Such subroutine nesting is limited only by available memory. Subroutines may be branched to conditionally by use of the ON...GOSUB statement, whose form is as follows:

ON ⟨expression⟩ GOSUB ⟨list of line numbers⟩

The execution is the same as ON...GOTO except that the line numbers are those of the first lines of subroutines. Execution continues at the next statement after the ON...GOSUB upon return from one of the subroutines.

d. OUT OF MEMORY errors. While nesting in loops, subroutines and branching is not limited by BASIC, memory size limitations restrict the size and complexity of programs. The OM or OUT OF MEMORY error message is issued when a program requires more memory than is available. See Appendix B for an explanation of the amount of memory required to run programs.

2-3 Input/Output

a. INPUT. The INPUT statement causes data input to be requested from the terminal. The format of the INPUT statement is as follows:

INPUT⟨list of variables⟩

The effect of the INPUT statement is to cause the values typed on the terminal to be assigned to the variables in the list. When an INPUT statement is executed, a question mark (?) is printed on the terminal signalling a request for information. The operator types the required numbers or strings separated by commas and types "enter". If the data entered is invalid (strings were entered when numbers were requested, etc.) BASIC prints 'REDO FROM START?' and waits for the correct data to be entered. If more data was requested by the INPUT statement than was typed, ?? is printed on the terminal and execution awaits the needed data. If more data was typed than was requested, the warning 'EXTRA IGNORED' is printed and execution proceeds. After all the requested data is input, execution continues normally at the statement following the INPUT. An optional prompt string may be added to an INPUT statement.

INPUT["⟨prompt string⟩";]⟨variable list⟩

Execution of the statement causes the prompt string to be printed before the question mark. Then all operations proceed as above. The prompt string must be enclosed in double quotation marks (") and must be separated from the variable list by a semicolon (;). Example:

```
100 INPUT "WHAT'S THE VALUE";X,Y causes the following output:
WHAT'S THE VALUE?
```

The requested values of X and Y are typed after the ? A carriage return or enter in response to an INPUT statement will cause execution to continue with the values of the variables in the variable list unchanged.

b. PRINT. The PRINT statement causes the Nascom to print data on the CRT or other terminal as appropriate. The simplest PRINT statement is:

PRINT

which prints a carriage return. The effect is to skip a line. The more usual PRINT statement has the following form:

PRINT list of expressions

which causes the values of the expressions in the list to be printed. String literals may be printed if they are enclosed in double quotation marks (").

The position of printing is determined by the punctuation used to separate the entries in the list. Nascom BASIC divides the printing line into zones of 14 spaces each. A comma causes printing of the value of the next expression to begin at the beginning of the next 14 column zone. A semicolon (;) causes the next printing to begin immediately after the last value printed. If a comma or semicolon terminates the list of expressions, the next PRINT statement begins printing on the same line according to the conditions above. Otherwise, a carriage return is printed.

c. DATA, READ, RESTORE.

1. The DATA statement. Numerical or string data needed in a program may be written into the program statements themselves, input from peripheral devices or read from DATA statements. The format of the DATA statement is as follows:

DATA<list>

where the entries in the list are numerical or string constants separated by commas. The effect of the statement is to store the list of values in memory in coded form for access by the READ statement. Examples:

```
10 DATA 1,2,-1E3,.04
20 DATA "ARR", NASCOM
```

Leading and trailing spaces in string values are suppressed unless the string is enclosed by double quotation marks.

2. The READ statement. The data stored by DATA statements is accessed by READ statements which have the following form:

READ<list of variables>

where the entries in the list are variable names separated by commas. The effect of the READ statement is to assign the values in the DATA lists to the corresponding variables in the READ statement list. This is done one by one from left to right until the READ list is exhausted. If there are more names in the READ list than values in the DATA lists, an OD or OUT OF DATA error message is issued. If there are more values stored in DATA statements than are read by a READ statement, the next READ statement to be executed will begin with the next unread DATA list entry. A single READ statement may access more than one DATA statement, and more than one READ statement may access the data in a single DATA statement.

An SN or SYNTAX ERROR message can result from an improperly formatted DATA list. The line number in the error message refers to the actual line of the DATA statement in which the error occurred.

3. RESTORE statement. After the RESTORE statement is executed, the next piece of data accessed by a READ statement will be the first entry of the first DATA list in the program. This allows re-READing the data.

d. CSAVEing and CLOADing Arrays. Numeric arrays may be saved on cassette or loaded from cassette using CSAVE* and CLOAD* The formats of the statements are:

CSAVE*<array name>

and

CLOAD*<array name>

The array is written out in binary with four octal 210 header bytes to indicate the start of data. These bytes are searched for when CLOADing the array. The number of bytes written is four plus

4*<number of elements> for the array

When an array is written out or read in, the elements of the array are written out with the leftmost subscript varying most quickly, the next leftmost second, etc:

```
DIM A(10)
CSAVE*A
```

writes out A(0),A(1),...A(10)

```
DIM A(10,10)
CSAVE*A
```

writes out A(0,0), A(1,0)...A(10,0),A(10,1)...A(10,10)

Using this fact, it is possible to write out an array as a two dimensional array and read it back in as a single dimensional array, etc.

Nascom Basic also generates a sumcheck at the end of the data. If the sumcheck fails on input, the message "Bad" is displayed and it is then possible to restart the program and try to read the data again.

e. Miscellaneous Input/Output.

1. WAIT. The status of input ports can be monitored by the WAIT command which has the following format:

WAIT<I,J>[,<K>]

where I is the number of the port being monitored and J and K are integer expressions. The port status is exclusive ORd with K and the result is ANDed with J. Execution is suspended until a non-zero value results. J picks the bits of port I to be tested and execution is suspended until those bits differ from the corresponding bits of K. Execution resumes at the next statement after the WAIT. If K is omitted, it is assumed to be zero. I,J and K must be in the range 0 to 255. Examples:

WAIT 20,6	Execution stops until either bit 1 or bit 2 of port 20 are equal to 1. (Bit 0 is least significant bit, 7 is the most significant.) Execution resumes at the next statement.
WAIT 10,255,7	Execution stops until any of the most significant 5 bits of port 10 are one or any of the least significant 3 bits are zero. Execution resumes at the next statement.

2. POKE, PEEK Data may be entered into memory in binary form with the POKE statement whose format is as follows:

POKE <I,J>

where I and J are integer expressions. POKE stores the byte J into the location specified by the value of I. I must be less than 32768. J must be in the range 0 to 255. Data may be POKEd into memory above location 32768 by making I a negative number. In that case, I is computed by subtracting 65536 from the desired address. To POKE data into location 45000, for example, I is 45000-65536=-20536. Care must be taken not to POKE data into the storage area occupied by BASIC or the system may be POKEd to death, and BASIC will have to be restarted.

The complementary function to POKE is PEEK. The format for a PEEK call is as follows:

PEEK(I)

where I is an integer expression specifying the address from which a byte is read. I is chosen in the same way as in the POKE statement. The value returned is an integer between 0 and 255. A major use of PEEK and POKE is to pass arguments and results to and from machine language subroutines.

3. DOKE, DEEK. These are double length versions of POKE and PEEK i.e. J may be in the range +32767 to -32768 calculated as for I on previous page. The least significant byte is stored in address I, and the most significant in I+1, so it can be used for storing 16 bit numbers or addresses for use by machine code subroutines.

4. OUT, INP. The format of the OUT statement is as follows:

OUT <I,J>

where I and J are integer expressions. OUT sends the byte signified by J to output port I. I and J must be in the range 0 to 255.

The INP function is called as follows:

INP(<I>)

INP reads a byte from port I where I is an integer expression in the range 0 to 255. Example:

```
20 IF INP(J)=16 THEN PRINT "ON"
```

3. FUNCTIONS

Nascom BASIC allows functions to be referenced in mathematical function notation. The format of a function call is as follows:

⟨name⟩(⟨argument⟩)

where the name is that of a previously defined function and the argument is an expression. Only one argument is allowed. Function calls may be components of expressions, so statements like

```
10 LET T=(F*SIN(T))/P    and
20 C=SQR(A↑2+B↑2+2*A*B*COS(T))
```

are legal.

3-1 Intrinsic Functions

Nascom BASIC provides several frequently used functions which may be called from any program without further definition. For a list of intrinsic functions, see section 6-3.

3-2 User-Defined Functions

a. The DEF statement. The programmer may define functions which are not included in the list of intrinsic functions by means of the DEF statement. The form of the DEF statement is as follows:

DEF⟨function name⟩(⟨variable name⟩)=⟨expression⟩

where the function name must be FN followed by a legal variable name and a 'dummy' variable name. The dummy variable represents the argument variable or value in the function call. Only one argument is allowed for a user-defined function. Any expression may appear on the right side of the equation, but it must be limited to one line. User-defined string functions are not allowed. Examples of valid functions are:

```
10 DEF FNFTOC(T)=(T-32)*5/9
12 DEF FNRAD(DEG)=3.14159/180*DEG When called with the measure of an angle in degrees,
                                  returns the radian equivalent.
```

A function may be redefined by executing another DEF statement with the same name. A DEF statement must be executed before the function it defines may be called.

b. USR. The USR function allows calls to assembly language subroutines. See appendix D

3-3 Errors

An FC or ILLEGAL FUNCTION CALL error results when an improper call is made to a function. Some places this might occur are the following:

1. A negative array subscript. LET A(-1)=0, for example.
2. An array subscript that is too large (>32767)
3. Negative or zero argument for LOG
4. Negative argument for SQR
5. A↑B with A negative and B not an integer.
6. A call to USR with no address patched for the machine language subroutine.
7. Improper arguments to MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, INSTR, STRING$, SPACE$ or ON...GOTO.

An attempt to call a user-defined function which has not previously appeared in a DEF statement will cause a UF or UNDEFINED USER FUNCTION error.

A TM or TYPE MISMATCH error will occur if a function which expects a string argument is given a numeric value or vice-versa.

4. STRINGS

In Nascom BASIC expressions may either have numeric value or may be strings of characters. Nascom BASIC provides a complete complement of statements and functions for manipulating string data. Many of the statements have already been discussed so only their particular application to strings will be treated in this section.

4-1 String Data

A string is a list of alphanumeric characters which may be from 0 to 255 characters in length. Strings may be stated explicitly as constants or referred to symbolically by variables. String constants are delimited by quotation marks at the beginning and end. A string variable name ends with a dollar sign ($). Examples:

```
A$="ABCD"       Sets the variable A$ to the four character string "ABCD"
B9$="14A/56"    Sets the variable B9$ to the six character string "14A/56"
FOOFOO$="E$"    Sets the variable FOOFOO$ to the two character string "E$"
```

Strings input to an INPUT statement need not be surrounded by quotation marks.

String arrays may be dimensioned exactly as any other kind of array by use of the DIM statement. Each element of a string array is a string which may be up to 255 characters long. The total number of string characters in use at any point in the execution of a program must not exceed the total allocation of string space or an OS or OUT OF STRING SPACE error will result. String space is allocated by the CLEAR command which is explained in section 6-2.

4-2 String Operations

a. Comparison Operators. The comparison operators for strings are the same as those for numbers:

```
=  equal
<> not equal
<  less than
>  greater than
=<, <= less than or equal to
=>, >= greater than or equal to
```

Comparison is made character by character on the basis of ASCII codes ubtil a difference is found. If, while comparison is proceeding, the end of one string is reached, the shorter string is considered to be smaller. ASCII codes may be found in Appendix A. Examples:

```
A<Z        ASCII A is 065, Z is 090
1<A        ASCII 1 is 049
" A">"A"   Leading and trailing blanks are significant in string literals.
```

b. String Expressions. String expressions are composed of string literals, string variables and string function calls connected by the + or concatenation operator. The effect of the catenation operator is to add the string on the right side of the operator to the end of the string on the left. If the result of concatenation is a string more than 255 characters long, an LS or STRING TOO LONG error message will be issued and execution will be terminated.

c. Input/Output. The same statements used for input and output of normal numeric data may be used for string data, as well.

1. INPUT, PRINT. The INPUT and PRINT statements read and write strings on the terminal. Strings need not be enclosed in quotation marks, but if they are not, leading blanks will be ignored and the string will be terminated when the first comma or colon is encountered. Examples:

```
10 INPUT ZOO$,FOO$          Reads two strings
20 INPUT X$                 Reads one string and assigns it to the variable X$
30 PRINT X$,"HI, THERE"     Prints two strings, including all spaces and punctuation in the
                            second.
```

2. DATA, READ. DATA and READ statements for string data are the same as for numeric data. For format conventions, see the explanation of INPUT and PRINT above.

4-3 String Functions

The format for intrinsic string function calls is the same as that for numeric functions. For the list of string functions, see section 6-3. String function names must end with a dollar sign.

5. ADDITIONAL COMMANDS

Nascom 8K Basic has a number of commands which are not normally found in 8K Basics. In addition to the DEEK and DOKE functions described in section 2-3 facilities are provided to call the monitor, manipulate the Nascom display screen, support add-on terminals and printers, and support graphics hardware.

a. MONITOR. This command transfers command to the monitor. Control can be restored to Basic from Nas-sys by using the monitor command J for a complete re-start, or Z for a warm start preserving programs etc. Note that, if breakpoints are set in order to de-bug a machine code subroutine, you should return to Basic by issuing an EFFFA, or EFFFD command as J and Z do not set breakpoints. The monitor command is often useful for issuing an XO command to turn on a printer.

b. WIDTH N. Input and output lines are normally assumed to be 47 characters long. However this can be inconvenient when supporting a printer. WIDTH (N) changes the assumed line buffer to N characters, where N is in the range 1 to 255.

On output to the serial port a newline will be generated automatically after N characters. Newlines will be generated every 48 characters on the Nascom display in addition to an additional one after N characters.

On input, the assumed line input buffer will be N or 72 characters, whichever is the longer. When entering data or programs with Nasbug T2 or T4, an internal newline will be generated after 47 characters too. When entering programs under Nas-sys normal mode the line width is assumed to be 48 characters and is not affected by the WIDTH command.

Note that WIDTH does not affect the calculation which determines whether a number can be printed in the current line (see 2-1(a)). This can be modified by

```
POKE 4163, (INT(N/14)-1)*14
```

where N is the line width in characters.

c. CLS clears the screen under all monitors

d. SCREEN X,Y sets the cursor position to character position X, line Y. X is in the range 1 to 48, Y is in the range 1 to 16. Note that line 16 is at the top of the screen and is not scrolled. Line 1 is the next line down, and line 15 is at the bottom.

The Commands

```
SCREEN 24,8
PRINT "HELLO"
```

will result in the message HELLO being printed at character position 24, line 8.

Note that with Nas-sys, only one character at a time can be printed on line 16, as the act of incrementing the cursor position results in it being set to the bottom of the screen, but this can be overcome by using the routine in Appendix I.

e. Lines N Normally, a LIST command will scroll five lines of program and wait for a newline character to be typed on the keyboard before scrolling another five lines. This default of five can be changed by the LINES command e.g. to 14 to scan larger pages of program, or to a larger number to allow listing to a serial printer.

N must be in the range +32767 to -32768. Negative numbers are treated as 65536 + N.

The following three commands are for use with the simple graphics option on Nascom 1 or Nascom 2 with the graphic character set. Points can be set black or white on a grid 96 points wide by 48 points high, with 0,0 being in the top left hand corner.

f. SET(X,Y) sets point X,Y bright
g. RESET (X,Y) if point X,Y is bright, sets it dark
h. POINT (X,Y) an integer function which returns the value 1 if point X,Y is set bright, 0 if not.

Note that X and Y must be in the range 0 to 95 and 0 to 47 respectively. Points with Y values in the range 45-47 appear on the top (unscrolled) line, otherwise points will appear on appropriate lines, descending as the value of Y increases.

The line which a point appears on is calculated as $L = INT(Y+3) + 1$

The character position a point appears on is calculated as $C = INT(X+2) + 1$

Characters other than the graphic characters COH to FFH are overwritten by SET and ignored by RESET and POINT.

i. Data Input under Nas-sys. Under Nas-sys in normal mode, data is input one character at a time, as it is with Nasbug T2 and T4. This allows INPUT statements to appear at any position on the screen with complete flexibility. Two additional modes of data input are provided.

```
DOKE 4175, -25
```

causes a newline to be generated on INPUT and data is input using the Nas-sys editing features.

```
DOKE 4175, -6670
```

causes similar results, but omits the newline, so the data input will include anything else on the line eg. the question mark and any prompt message printed by the INPUT statement. The additional characters can be stripped off by string manipulation in the usual way.

```
DOKE 4175, -6649
```

restores normal operation. Care should be taken when using these commands, as incorrect arguments will cause the system to crash. The values may change at a future date with new issues of the Basic Interpreter.

j. Printer Support. In addition to printers and terminals attached via the serial interface, printers can be interfaced to the PIO on the Nascom. Routines to drive such printers can be included and switched on and off using DOKE and POKE commands (see appropriate monitor manual for detailed information).

k. Aborting Programs. Escape (Shift and Newline) stops and aborts a running program (see 6-4) but only when typed on the Nascom keyboard. Generating an NMI (non maskable interrupt) has the same effect. It is not possible to abort in this way from a cassette read or write operation, but a similar effect can be achieved by hitting reset and entering the "Z" command.

6. LISTS AND DIRECTORIES

6-1 Commands

Commands direct Nascom BASIC to arrange memory and input/output facilities, to list and edit programs and to handle other housekeeping details in support of program execution. Nascom BASIC accepts commands after it prints 'OK' and is at command level. The table below lists the commands in alphabetical order.

Command
CLEAR
Sets all program variables to zero
CLEAR[<expression>]
Same as CLEAR but sets string space (see 4-1) to the value of the expression. If no argument is given, string space will remain unchanged. When Nascom BASIC is started, string space is set to 50 bytes.
CLOAD<string expression>
Causes the program on cassette tape designated by the first character of <STRING expression> to be loaded into memory. A NEW command is issued before the program is loaded.
CLOAD?<string expression>
Verifies that the program specified is loadable and error free.
CLOAD*<array name>
Loads the specified array from cassette tape. May be used as a program statement.
CONT
Continues program execution after an Escape has been typed or a STOP or END statement has been executed. Execution resumes at the statement after the break occurred unless input from the terminal was interrupted. In that case, execution resumes with the reprinting of the prompt (? or prompt string). CONT is useful in debugging, especially where an 'infinite loop' is suspected. An infinite loop is a series of statements from which there is no escape. Typing Escape causes a break in execution and puts BASIC in command level. Direct mode statements can then be used to print intermediate values, change the values of variables, etc. Execution can be restarted by typing the CONT command, or by executing a direct mode GOTO statement, which causes execution to resume at the specified line number.

Execution cannot be continued if a direct mode error has occurred during the break. Execution cannot continue if the program was modified during the break.
CSAVE<string expression>
Causes the program currently in memory to be saved on cassette tape under the name specified by the first character of <string expression>.
CSAVE*<array name>
Causes the array named to be saved on cassette tape. May be used as a program statement.

LIST
Lists the program currently in memory starting with the lowest numbered line. Listing is terminated either by the end of the program or by typing Escape (Shift & Newline)
LIST[<line number>]
Prints the current program beginning at the specified line. The LIST command will print n lines (where n is 5 or as modified by the LINES command) and wait for enter/newline to be typed before continuing. Typing ESCAPE returns control to Basic.
NEW
Deletes the current program and clears all variables. Used before entering a new program.
NULL<integer expression>
Sets the number of nulls to be printed at the end of each line. For 10 character per second tape punches, <integer expression> should be >=3. For 30 cps punches, it should be >=3. When tapes are not being punched, <integer expression> should be 0 or 1 for Teletypes* and Teletype compatible CRT's. It should be 2 or 3 for 30 cps hard copy printers. The default value is 0. Note that the Nascom monitors will normally ignore nulls on output, but a suitable delay can usually be generated by setting NULL 255. Nulls are passed to user I/O drivers.
RUN[<line number>]
starts execution of the program currently in memory at the line specified. If the line number is omitted, execution begins at the lowest line number.

6-2 Statements

The following table of statements is listed in alphabetical order. In the table, X and Y stand for any expressions allowed in the version under consideration. I and J stand for expressions whose values are truncated to integers. V and W are any variable names. The format for a Nascom BASIC line is as follows:
<nnnnn> <statement>[:<statement>...]
where nnnnn is the line number

DATA DATA<list>
specifies data to be read by a READ statement. List elements can be numbers or strings. List elements are separated by commas.

DEF DEF FNV(<W>)=<X>
Defines a user-defined function. Function name is FN followed by a legal variable name. Definitions are restricted to one line (72 characters).

DOKE DOKE <I>,<J>
Stores J in memory location I and I+1. If I or J are negative, they are interpreted as 65536 + I or J, as appropriate.

* Teletype is a registered trademark of the Teletype Corporation.

DIM DIM.<V>(<I>[,J...])[,...]
Allocates space for array variables. More than one variable may be dimensioned by one DIM statement up to the limit of the line. The value of each expression gives the maximum subscript possible. The smallest subscript is 0. Without a DIM statement, an array is assumed to have maximum subscript of 10 for each dimension referenced. For example, A(I,J) is assumed to have 121 elements, from A(0,0) to A(10,10) unless otherwise dimensioned in a DIM statement.

END END
Terminates execution of a program.

FOR FOR<V>=<X>TO<Y>[STEP<Z>]
Allows repeated execution of the same statements. First execution sets V=X. Execution proceeds normally until NEXT is encountered. Z is added to V, then, IF Z<0 and V>=Y, or if Z>0 and V<=Y, BASIC branches back to the statement after FOR. Otherwise, execution continues with the statement after NEXT.

GOTO GOTO<nnnnn>
Unconditional branch to line number

GOSUB GOSUB<nnnnn>
Unconditional branch to subroutine beginning at line nnnnn.

IF...GOTO IF <X> GOTO<nnnnn>
Same as IF...THEN except GOTO can only be followed by a line number and not another statement.

IF...THEN IF<X>THEN<Y>
or IF<X>THEN<statement>[:statement...]
If value of X<>0, branches to line number or statement after THEN. Otherwise, execution proceeds at the line after the IF...THEN.

INPUT INPUT<V>[,<W>...]
Causes BASIC to request input from terminal. Values typed on the terminal are assigned to the variables in the list.

LET LET <V>=<X>
Assigns the value of the expression to the variable. The word LET is optional.

LINES LINES <A>
Sets the number of lines printed by a LIST command before pausing to n.

NEXT NEXT [<V>,<W>...]
Last statement of a FOR loop. V is the variable of the most recent loop, W of the next most recent and so on. NEXT without a variable terminates the most recent FOR loop.

ON...GOTO ON<I>GOTO<list of line numbers>
Branches to line whose number is Ith in the list. List elements are separated by commas. If I=0 or > number of elements in the list, execution continues at next statement. If I<0 or >255, an error results.

ON...GOSUB ON <I> GOSUB <list>
Same as ON...GOTO except list elements are initial line numbers of subroutines.

OUT OUT<I>,<J>
Sends byte J to port I. 0<=I,J<=255.

POKE POKE<I>,<J>
Stores byte J in memory location derived from I. 0<=J<=255; -32768<I<65536. If I is negative, address is 65536+I, if I is positive, address=I.

PRINT PRINT<X>[,<Y>...]
Causes values of expressions in the list to be printed on the terminal. Spacing is determined by punctuation.

Punctuation	Spacing - next printing begins:
,	at beginning of next 14 column zone
;	immediately
other or none	at beginning of next line.

String literals may be printed if enclosed by (") marks.

READ READ<V>[,<W>...]
Assigns values in DATA statements to variables. Values are assigned in sequence with the first value in the first DATA statement.

REM REM[<remark>]
Allows insertion of remarks. Not executed, but may be branched into. In extended versions, remarks may be added to the end of a line preceded by a single quotation mark (').

RESTORE RESTORE
Allows data from DATA statements to be reread. Next READ statement after RESTORE begins with first data of first data statement.

RETURN RETURN
Terminates a subroutine. Branches to the statement after the most recent GOSUB.

SCREEN SCREEN <X>,<Y>
Sets the cursor position to character position X, line Y.

STOP STOP
Stops program execution. BASIC enters command level and prints BREAK IN LINE nnnnn.

WAIT WAIT<I>,<J>[,<K>]
Status of port I is XOR'd with K and AND'ed with J.
Continued execution awaits non-zero result. K defaults to 0. 0<=I,J,K<=255.

WIDTH WIDTH <n>
Sets the width of input and print buffers to n (see 5 (b)).

6-3 Intrinsic Functions

Nascom BASIC provides several commonly used algebraic and string functions which may be called from any program without further definition. The functions in the following table are listed in alphabetical order. The notation to the right of the Call Format is the versions in which the function is available. As usual, X and Y stand for expressions, I and J for integer expressions and X$ and Y$ for string expressions.

Function	Call Format

ABS — ABS(X)
Returns absolute value of expression X. ABS(X)=X if X>=0,-X if X<0.

ASC — ASC(X$)
Returns the ASCII code of the first character of the string X$. ASCII codes are in appendix A.

ATN — ATN(X)
Returns arctangent (X). Result is in radians in range -pi/2 to pi/2.

CHR$ — CHR$(I)
Returns a string whose one element has ASCII code I. ASCII codes are in Appendix A.

COS — COS(X)
Returns cos(X). X is in radians.

EXP — EXP(X)
Returns e to the power x. X must be <=87.3365.

FRE — FRE(0)
Returns number of bytes in memory not being used by BASIC. If argument is a string, returns number of free bytes in string space.

INP — INP(I)
Reads a byte from port I.

INT — INT(X)
Returns the largest integer <=X

LEFT$ — LEFT$(X$,I)
Returns leftmost I characters of string X$

LEN — LEN(X$)
Returns length of string X$. Non-printing characters and blanks are counted.

LOG — LOG(X)
Returns natural log of X. X>0

MID$ — MID$(X$,I[,J])
Without J, returns rightmost characters from X$ beginning with the Ith character. If I>LEN(X$), MID$ returns the null string. 0<I<255. With 3 arguments, returns a string of length J of characters from X$ beginning with the Ith character. If J is greater than the number of characters in X$ to the right of I, MID$ returns the rest of the string. 0<=J<=255.

POS — POS(I)
Returns present column position of terminal's print head. Leftmost position =0.

RND — RND(X)
Returns a random number between 0 and 1. X<0 starts a new sequence of random numbers. X>0 gives the next random number in the sequence. X=0 gives the last number returned. Sequences started with the same negative number will be the same.

RIGHT$ — RIGHT$(X$,I)
Returns rightmost I characters of string X$. If I=LEN(X$), returns X$.

SGN — SGN(X)

If X>0, returns 1, if X=0 returns 0, if X<0, returns -1. For example,
ON SGN(X)+2 GOTO 100,200,300 branches to 100 if X is negative, 200 if X is 0 and 300 if X is positive.

SIN — SIN(X)
Returns the sine of the value of X in radians. COS(X)=SIN(X+3.14159/2).

SPC — SPC(I)
Prints I blanks on terminal. 0<=I<=255

SQR — SQR(X)
Returns square root of X. X must be >=0

STR$ — STR$(X)
Returns string representation of value of X.

TAB — TAB(I)
Spaces to position I on the terminal. Space 0 is the leftmost space, 71 the rightmost. If the carriage is already beyond space I, TAB has no effect. 0<=I<=255. May only be used in PRINT statements.

TAN — TAN(X)
Returns tangent(X). X is in radians.

USR — USR(X)
Calls the user's machine language subroutine with argument X.

VAL — VAL(X$)
Returns numerical value of string X$. If first character of X$ is not +,-,& or a digit, VAL(X$)=0.

6-4 Special Characters

Nascom BASIC recognizes several characters in the ASCII font as having special functions in carriage control, editing and program interruption. Characters such as Control/C, Control/S, etc., are typed by holding down the Control key and typing the designated letter. The special characters are listed in the table below. Note that those marked with an asterisk do not apply when inputting using Nas-sys in normal mode.

* @ @ (Escape or shift + Newline for Nas-sys)
Erases current line and executes carriage return

(backspace)
Erases last character typed. If there is no last character types a carriage return.

* _ _ (underline)
same as backspace.

Newline (or Enter)
Returns print head or cursor to beginning of the next line.

Escape (shift + newline)
Interrupts execution of current program or list command. Takes effect after execution of the current statement or after listing the current line. Program execution can be continued by typing any character, except that typing a further Escape causes BASIC to go to command level and types OK.
CONT command resumes exeuction. See section 6-1.

Note: The keys must be depressed on the Nascom keyboard. Pressing ESC on an attached serial terminal has no effect.

: :
Separates statements in a line.

? ?
Equivalent to PRINT statement.

* Rubout (Control Z)
Deletes previous character on an input line. First Rubout prints the last character to be printed. Each successive Rubout prints the next character to the left and deletes it typing a new character causing the new character to be printed.

* Control/R
Prints newline and echos the line being input.

Control/U
Same as @

Lower Case Input
Lower case alphabetic characters are always echoed as lower case, but LIST and PRINT will translate lower case to upper case, if the lower case characters are not part of string literals, REM statements or single quote (') remarks.

6-5 Error Messages

After an error occurs, BASIC returns to command level and types OK. Variable values and the program test remain intact, butthe program cannot be continued by the CONT command. All GOSUB and FOR context is lost. The program may be continued by direct mode GOTO however. When an error occurs in a direct statement, no line number is printed. Format of error messages:

Direct Statement	?XX ERROR
Indirect Statement	?XX ERROR IN YYYYY

where XX is the error code and YYYYY is the line number where the error occurred. The following are the possible error codes and their meanings:

ERROR CODE	EXTENDED ERROR MESSAGE
BS	SUBSCRIPT OUT OF RANGE

An attempt was made to reference an array element which is outside the dimensions of the array. This error can occur if the wrong number of dimensions are used in an array reference. For example:
LET A (1,1,1)=Z
when A has already been dimensioned by DIM A (10,10)

DD	REDIMENSIONED ARRAY

After an array was dimensioned, another dimension statement for the same array was encountered. This error often occurs if an array has been given the defult dimension of 10 and later in the program a DIM statement is found for the same array.

FC	ILLEGAL FUNCTION CALL

The parameter passed to a math or string function was out of range. FC errors can occur due to:

1. a negative array subscript (LET A (-1)=0)
2. an unreasonably large array subscript (>32767)
3. LOG with negative or zero argument
4. SQR with negative argument
5. A B with A negative and B not an integer
6. a call to USR before the address of a machine language subroutine has been entered.
7. calls to MID$, LEFT$, RIGHT$, INP, OUT, WAIT, SCREEN, WIDTH, SET, RESET, POINT, DEEK, DOKE, PEEK, POKE, TAB, SPC, or ON...GOTO with an improper argument.

MO	MISSING OPERAND

No operand has been given to the command. CSAVE without a file name is illegal.

ID ILLEGAL DIRECT

INPUT and DEF are illegal in the direct mode.

NF NEXT WITHOUT FOR

The variable in a NEXT statement corresponds to no previously executed FOR statement.

OD OUT OF DATA

A READ statement was executed but all of the DATA statements in the program have already been read. The program tried to read too much data or insufficient data was included in the program.

OM OUT OF MEMORY

Program is too large, has too many variables, too many FOR loops, too many GOSUBs or too complicated expressions. See Appendix C.

OV OVERFLOW

The result of a calculation was too large to be represented in BASIC's number format. If an underflow occurs, zero is given as the result and execution continues without any error message being printed.

SN SYNTAX ERROR

Missing parenthesis in an expression, illegal character in a line, incorrect punctuation, etc.

RG RETURN WITHOUT GOSUB

A RETURN statement was encountered before a previous GOSUB statement was executed.

UL UNDEFINED LINE

The line reference in a GOTO, GOSUB, or IF... THEN was to a line which does not exist.

/0 DIVISION BY ZERO

Can occur with integer division and MOD as well as floating point division. 0 to a negative power also causes a DIVISION BY ZERO error.

CN CAN'T CONTINUE

Attempt to continue a program when none exists, an error occurred, or after a modification was made to the program.

LS STRING TO LONG

An attempt was made to create a string more that 255 characters long.

OS OUT OF STRING SPACE

String variables exceed amount of string space allocated for them. Use the CLEAR command to allocate more string space or use smaller strings or fewer string variables.

ST STRING FORMULA TOO COMPLEX

A string expression was too long or too complex. Break it into two or more shorter ones.

TM TYPE MISMATCH

The left hand side of an assignment statement was a numeric variable and the right hand side was a string, or vice-versa; or a function which expected a string argument was given a numeric one or vice-versa.

UF UNDEFINED USER FUNCTION

Reference was made to a user defined function which had never been defined.

6-6 Reserved Words

Some words are reserved by the Nascom BASIC interpreter for use as statements, commands, operators, etc. and thus may not be used for variable or function names. The reserved words are listed below. In addition to these words, intrinsic function names are reserved words.

RESERVED WORDS

CLEAR	NEW	AND	OUT
DATA	NEXT	CONT	POINT
DIM	PRINT	DEF	RESET
END	READ	DOKE	POKE
FOR	REM	FN	SCREEN
GOSUB	RETURN	LINES	SET
GOTO	RUN	NOT	SPC
IF	STOP	NULL	WAIT
INPUT	TO	ON	WIDTH
LET	TAB	OR	
LIST	THEN		
	USR		

7 RUNNING BASIC

Basic is distributed on 1 x 8K byte ROM or 8 x 1K byte EPROM's and should be situated from E000H to FFFFH.

The following entry points are available for running basic.

a) E000H - cold start entered by typing EE000 - when used with Nas-sys also resets the monitor
b) FFFAH - normal cold start. Entered by typing J under Nas-sys or EFFFA under Nasbug - does not reset the monitor
c) FFFDH - warm start. Entered by typing Z under Nas-sys or EFFFD under Nasbug - retains any programs etc. in store and can only be used after the system has been initialised by entering at E000 or FFFA

when initialised, the system responds with the message
Memory Size?
You should then type either

a) a newline or enter character, after which the Basic will use all available store above 1000H, or
b) a decimal address representing the highest store location you wish Basic to use. In this way you can reserve space at the top of store for user machine code routines.

when successfully started, Basic prints the message

```
NASCOM ROM BASIC Ver 4.7
Copyright (c) 1978 by Microsoft
<n> Bytes free
```

where <n> is the number of bytes available for program and data, and enters Basic command mode. Programs or direct commands can then be entered.

APPENDIX A

ASCII CHARACTER CODES

DECIMAL	CHAR.	DECIMAL	CHAR.	DECIMAL	CHAR.
000	NUL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	-	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	\
007	BEL	050	2	093	]
008	BS	051	3	094	^
009	HT	052	4	095	<
010	LF	053	5	096	'
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	=	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	{
038	&	081	Q	124	|
039	'	082	R	125	}
040	(	083	S	126	~
041	)	084	T	127	DEL
042	*	085	U		

LF=Line Feed FF=Form Feed CR=Carriage Return DEL=Rubout

Using ASCII codes -- the CHR$ function.

CHR$(X) returns a string whose one character is that with ASCII code X. ASC(X$) converts the first character of a string to its ASCII decimal value.

One of the most common uses of CHR$ is to send a special character to the user's terminal. The most often used of these characters is the BEL (ASCII 7). Printing this character will cause a bell to ring on some terminals and a beep on many CRT's. This may be used as a preface to an error message, as a novelty, or just to wake up the user if he has fallen asleep. Example:

```
PRINT CHR$(7);
```

Another major use of special characters is on those CRT's that have cursor positioning and other special functions (such as turning on a hard copy printer). For example, on some CRT's a form feed (CHR$(12)) will cause the screen to erase and the cursor to "home" or move to the upper left corner.

Some CRT's give the user the capability of drawing graphs and curves in a special point-plotter mode. This feature may easily be taken advantage of through use of Nascom BASIC's CHR$ function.

APPENDIX B

SPACE AND SPEED HINTS

A. Space Allocation

The memory space required for a program depends, of course, on the number and kind of elements in the program. The following table contains information on the space required for the various program elements.

Element	Space Required
Variables	
numeric	6 bytes
Arrays	
strings and floating pt.	(no of elements)* 6 + 5 + (no of dimensions)*2 bytes
Functions	
intrinsic	1 byte for the call
user-defined	6 bytes for the definition
Reserved Words	1 byte each
Other Characters	1 byte each
Stack Space	
active FOR loop	16 bytes
active GOSUB	5 bytes
parentheses	6 bytes each set
temporary result	10 bytes

Basic itself occupies 8K of ROM

B. Space Hints

The space required to run a program may be significantly reduced without affecting execution by following a few of the following hints:

1. Use multiple statements per line. Each line has a 5 byte overhead for the line number, etc., so the fewer lines there are, the less storage is required.
2. Delete unnecessary spaces. Instead of writing
   ```
   10 PRINT X,Y,Z
   ```
   use
   ```
   10 PRINTX,Y,Z
   ```
3. Delete REM statements to save 1 byte for REM and 1 byte for each character of the remark.
4. Use variables instead of constants, especially when the same value is used several times. For example, using the constant 3.14159 ten times in a program uses 40 bytes more space than assigning
   ```
   10 P=3.14159
   ```
   once and using P ten times.
5. Using END as the last statement of a program is not necessary and takes one byte extra.
6. Reuse unneeded variables instead of defining new variables.
7. Use subroutines instead of writing the same code several times.
8. Use the zero elements of arrays. Remember the array dimensioned by
   ```
   100 DIM A(10)
   ```
   has eleven elements, A(0) through A(10).

C. Speed Hints

1. Deleting spaces and REM statements gives a small but significant decrease in execution time.
2. Variables are set up in a table in the order of their first appearance in the program. Later in the program, BASIC searches the table for the variable at each reference. Variables at the head of the table take less time to search for than those at the end. Therefore, reuse variable names and keep the list of variables as short as possible.
3. Use NEXT without the index variable.
4. Use variables instead of constants, especially in FOR loops and other code that must be executed repeatedly.
5. String variables set up a descriptor which contains the length of the string and a pointer to the first memory location of the string. As strings are manipulated, string space fills up with intermediate results and extraneous material as well as the desired string information. When this happens, BASIC's "garbage collection" routine clears out the unwanted material. The frequency of garbage collection is inversely proportional to the amount of string space. The more string space there is, the longer it takes to fill with garbage. The time garbage collection takes is proportional to the square of the number of string variables. Therefore, to minimize garbage collection time, make string space as large as possible and use as few string variables as possible.

APPENDIX C

MATHEMATICAL FUNCTIONS

1. Derived Functions

The following functions, while not intrinsic to Nascom BASIC, can be calculated using the existing BASIC functions:

Function	BASIC equivalent
SECANT	SEC(X) = 1/COS(X)
COSECANT	CSC(X) = 1/SIN(X)
COTANGENT	COT(X) = 1/TAN(X)
INVERSE SINE	ARCSIN(X) = ATN(X/SQR(-X*X+1))
INVERSE COSINE	ARCCOS(X) = -ATN (X/SQR(-X*X+1)) +1.5708
INVERSE SECANT	ARCSEC(X) = ATN(XSQR(X*X-1)) +SGN(SGN(X)-1)*1.5708
INVERSE COSECANT	ARCCSC(X) = ATN(1/SQR(X*X-1)) +(SGN(X)-1)*1.5708
INVERSE COTANGENT	ARCCOT(X) = ATN(X)+1.5708
HYPERBOLIC SINE	SINH(X) = (EXP(X)-EXP(-X))/2
HYPERBOLIC COSINE	COSH(X) = (EXP(X)+EXP(-X))/2
HYPERBOLIC TANGENT	TANH(X) = EXP(-X)/EXP(X)+EXP(-X)) *2+1
HYPERBOLIC SECANT	SECH(X) = 2/(EXP(X)+EXP(=X))
HYPERBOLIC COSECANT	CSCH(X) = 2/(EXP(X)-EXP(-X))
HYPERBOLIC COTANGENT	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X)) *2+1
INVERSE HYPERBOLIC SINE	ARCSINH(X) = LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X) = LOG(X+SQR(X*X+-1))
INVERSE HYPERBOLIC TANGENT	ARCTANH(X) = LOG((1+X)/(1-X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X) = LOG((SQR(-X*X+1)+1)/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X) = LOG((SGN(X)*SQR(X*X+1)+1)/X
INVERSE HYPERBOLIC COTANGENT	ARCCOTH(X) = LOG((X+1)/(X-1))/2

APPENDIX D

BASIC AND ASSEMBLY LANGUAGE

Nascom BASIC has provisions for interfacing with assembly language routines. The USR function allows Nascom BASIC programs to call assembly language subroutines in the same manner as BASIC functions.

The first step in setting up a machine language subroutine for a BASIC program is to set aside memory space. When BASIC asks, "MEMORY SIZE?" during initialization, the response should be the top of memory available minus the amount needed for the assembly language routine. BASIC uses all the bytes it can find from location 4096 up, so the topmost locations in memory can be used for user supplied routines. Locations from 0C00H to 1000H not used by the Nascom monitor can also be used for user written routines. If the answer to the MEMORY SIZE? question is too small, BASIC will ask the question again until it gets all the memory it needs. See Appendix C for Nascom BASIC's memory requirements.

The assembly language routine may be loaded into memory in the usual way, or from a BASIC program by means of a POKE or DOKE statement.

The starting address of the assembly language routine goes in USRLOC, a two byte location in memory situated at 1004H and 1005H (least significant byte in 1004H most significant in 1005H). The function USR calls the routine whose address is in USRLOC. Initially, USRLOC contains the address of ILLFUN, the routine which gives the FC or ILLEGAL FUNCTION CALL error. If USR is called without an address loaded in USRLOC, an ILLEGAL FUNCTION CALL error results.

When USR is called, the stack pointer is set up for 8 levels (16 bytes) of stack storage. If more stack space is needed, BASIC's stack can be saved and a new stack set up for use by the assembly language routine. BASIC's stack must be restored, however, before returning from the user routine.

All memory and all the registers can be changed by a user's assembly language routine. Of course, memory locations within BASIC ought not to be changed, nor should more bytes be popped off the stack than were put on it.

USR is called with a single argument. The assembly language routine can retrieve this argument by calling the routine whose address is in locations E00BH and E00CH. The low-order byte of the address is in E00BH and the high-order in E00CH. This routine (DEINT) stores the argument in the register pair [D,E].

The argument is truncated to an integer and if it is not in the range -32768 to 32767, an FC error occurs.

To pass a result back from an assembly language routine, load the value in register pair [A,B]. This value must be a signed, 16 bit integer as defined above. Then call the routine whose address is in locations E00DH and E00EH. If this routine is not called, USR(X) returns X. To return to BASIC, then, the assembly language routine executes a RET instruction.

Any interrupt handling routines should save the stack, registers A-L and the PSW. They should also reenable interrupts before returning since an interrupt automatically disables all further interrupts once it is received.

There is only one way to call an assembly language routine but this does not limit the programmer to only one assembly language routine. The argument of USR can be used to designate which routine is being called. Additional arguments can be passed through the use of POKE or DOKE and values may be passed back by PEEK or DEEK.

APPENDIX E

USING THE CASSETTE INTERFACE

Programs may be saved on cassette tape by means of the CSAVE command. CSAVE may be used in either direct or indirect mode, and its format is as follows:

CSAVE <string expression>

The program currently in memory is saved on cassette under the name specified by the first character of the <string expression>. Note that the program named A is saved by CSAVE "A". After CSAVE is completed, BASIC always returns to command level. Programs are written on tape in BASIC's internal representation. Variable values are not saved on tape, although an indirect mode CSAVE does not affect the variable values of the program currently in memory. The number of nulls (see NULL command) has no effect on the operation of CSAVE. Before using CSAVE, turn on the cassette recorder. Make sure the tape is in the proper position then put the recorder in RECORD mode.

CSAVE first writes a block header containing the first character of the string expression and then calls the appropriate monitor routine to dump the program.

Progams may be loaded from cassette tape by means of the CLOAD command, which has the same format as CSAVE. The effect of CLOAD is to execute a NEW command, clearing memory and all variable values and loading the specified file into memory. When finished Nascom BASIC returns to command level. Reading starts by searching until 3 consecutive zeros are read. BASIC will not return to command level after a CLOAD if it could not find the requested file, or if the file was found but was mis-read. In that case, the computer will continue to search until it is stopped and restarted. When a program is found the message

File program identifier Found

is displayed. The method of displaying the program as it is input depends on the monitor in use. Data may be read and written with the CSAVE* and CLOAD* commands. The formats are as follows:

CSAVE*<array variable name>

and

CLOAD*<array variable name>

See section 2-4d for a discussion of CSAVE* and CLOAD* for array data.

A sumcheck is generated on input and output. If the sumcheck fails on input, the message "bad" is displayed and Basic returns to command level. Note that the incorrect data will have been put in store.

In case of a misread, a direct GOTO or CLOAD command can be used to cause the data to be re-read.

Under Nas-sys, CLOAD?<string expression> reads the program from tape without putting it in store and can be used to check that a program has been correctly saved.

Note that you can also fool Basic into LISTing or PRINTing to tape or INPUTting programs or data from tape by using the XO command under Nasbug T4 and Nas-sys. This is a useful way of storing libraries of subroutines which can be input and used as part of different programs.

Further examples of storage and retrieval of tape data are contained in appendix I.

APPENDIX F

CONVERTING BASIC PROGRAMS

NOT WRITTEN FOR NASCOM COMPUTERS

Though implementations of BASIC on different computers are in many ways similar, there are some incompatibilities between Nascom BASIC and the BASIC used on other computers.

1. Strings

A number of BASICs require the length of strings to be declared before they are used. All dimension statements of this type should be removed from the program. In some of these BASICs, a declaration of the form A$(I,J) declares a string array of J elements each of which has a length I. Convert DIM statements of this type to equivalent ones in Nascom BASIC: DIM A$(J). Nascom BASIC uses "+" for string concatenation, not "," or " & ." Nascom BASIC uses LEFT$, RIGHT$ and MID$ to take substrings of strings. Some other BASICs use A$(I) to access the Ith character of the string A$, and A$(I,J) to take a substring of A$ from character position I to character position J. Convert as follows:

OLD	NEW
A$(I)	MID$(AS,I,1)
A$(I,J)	MID$(AS,I,J-I+1)

This assumes that the reference to a subscript of A$ is in an expression or is on the right side of an assignment. If the reference to A$ is on the left hand side of an assignment, and X$ is the string expression used to replace characters in A$, convert as follows:

OLD	NEW
A$(I)=X$	A$=LEFT$(A$,I-1)+X$+MID$(A$,I+1)
A$(I,J)=X$	A$=LEFT$(A$,I-1)+X$+MID$(A$,J+1)

2. Multiple assignments.

Some BASICs allow statements of the form:

```
500 LET B=C=0
```

This statement would set the variables B and C to zero. In Nascom BASIC, this has an entirely different effect. All the "=" signs to the right of the first one would be interpreted as logical comparison operators. This would set the variable B to -1 if C equaled 0. If C did not equal 0, B would be set to 0. The easiest way to convert statements like this one is to rewrite them as follows:

```
500 C=0:B=C
```

3. Some BASICs use "\" instead of ":" to delimit multiple statements on a line. Change each "\" to ":" in the program.

4. Programs which use the MAT functions available in some BASICs will have to be rewritten using FOR...NEXT loops to perform the appropriate operations.

APPENDIX G

STORAGE USED

Nascom Basic leaves locations between 0C80H (3200 decimal) and 1000H (4096 decimal) for use by user machine code routines. It uses locations 1000H to 3E11H (4414) for workspace and resides in E000H to FFFFH. Locations 3E12H to E000H are therefore available for the users Basic program and data.

APPENDIX H

USEFUL BOOKS

You may find the following books useful. They are not intended to be a complete bibliography, merely a list of books which we at Nascom have seen and used.

1. General Introduction to Pragramming in Basic

   Basic Programming by John G. Kemery and Thomas E Kurtz, Pub. Wiley

   Instant Basic by Jerald R Brown, Pub. Dilithium Press

   Basic Basic by James S Coan, Pub. Jayden

   Advanced Basic by James S Coan, Pub. Hayden

2. Games and Useful Programs

   What To Do After You Hit Return (or PCC's First Book of Computer Games). Pub. People's Computer Company

   Basic Computer Games, Ed. David H Ahl, Pub. Workman Publishing

   Basic Software Library (six volumes) R W Brown, Pub. Scientific Research Institute

APPENDIX 1

USEFUL ROUTINES

1. Writing to line 16 (non-scrolled) under Nas-sys

```
1  REM THIS ROUTINE WRITES TO LINE 16
2  REM USING NAS-SYS
10 CLS
20 SCREEN 1,15
25 REM THAT PUTS IT ON BOTTOM LINE
30 PRINT "HEADER";
35 REM NOW WE ARE GOING TO COPY IT TO TOP
40 FOR C=2954 to 3000 STEP 2
50 DOKE C+64,DEEK(C)
60 NEXT C
65 REM CHR$(27) GENERATES ESC=LINE DELETE
70 PRINT CHR$(27);
80 REM REST OF PROGRAM CAN START HERE
```

2. Program to convert Hex numbers to Decimal

```
4  CLS
5  PRINT
10 INPUT"ENTER HEX No.";H$
20 T=0:D=1
30 FOR P=LEN(H$)-1 TO 0 STEP-1
40 C=ASC(MID$(H$,D,1))
50 D=D+1
60 IF (C>=48)*(C<=57) THEN C=C-48:GOTO 100
70 IF (C>=65)*(C<=70) THEN C=C-55: GOTO 100
80 PRINT "+ve Hex with no D.P. please":GOTO 5
100 T=T+C*16↑P
110 NEXT
120 PRINT "Hex ";H$;" in Decimal is";T
130 GOTO 5
```

A better way of writing lines 60 & 70 is :

```
60   IF C>= 48 AND C<= 57 THEN... etc.
70   IF C>= 65 AND C<= 70 THEN... etc.
```

3. To set X Mode under Nas-sys

```
10 DOKE 3189 , 1925
20 DOKE 3187 , 1917
30 POKE 3112 c <option>
```

where option is 0, 1, 16, 17, 32, 33, 48 or 49, as appropriate
e.g. to set X0, option = 0
This routine can be used to turn a serial printer on under program control

4. To set N Mode under Nas-sys

```
10 DOKE 3189 , 1922
20 DOKE 3187 , 1919
```

This routine can be used to turn off a serial printer

5. To set U Mode under Nas-sys

```
10 DOKE 3189 , 1921
20 DOKE 3187 , 1918
```

This can be used to set U mode to support user-written I/O drivers.

6. To set Keyboard (K) modes under Nas-sys

```
10 POKE 3111, <option>
```

where option = 0, 1, 4, or 5 to set normal, typewriter, graphics or typewriter and graphics mode.

7. To scan the keyboard for input under Nas-sys

The following code sets up a USR call which scans the keyboard and returns the ASCII value of any character typed on the keyboard, or zero if no character has been typed.

```
10 DOKE 3200 , 25311
20 DOKE 3202 , 312
30 DOKE 3204 , 18351
40 DOKE 3206 , 10927
50 DOKE 3208 , -8179
60 POKE 3210 , 233
70 DOKE 4100 , 3200
```

The USR call can be used as follows, for example.

```
80 IF USR(0)<>0THEN GOTO 500
```

The body of the USR subroutine is as follows:

```
                    RST  SCAL
0C80  DF 62         DEFB ZIN        ; scan inputs
0C82  38 01         JR C, CHAR      ; skip if char
0C84  AF            XOR A           ; clear A
0C85  47            CHAR LD B, A    ; put char in B
0C86  AF            XOR A           ; clear A
0C87  2A 0D E0      LD HL(#E00D)    ; get address in HL
0C8A  E9            JP (HL)         ; jump and return
```

APPENDIX J

SINGLE CHARACTER INPUT OF RESERVED WORDS

Internally, Basic stores reserved words in programs in the form of a single character reserved word token, which has bit 8 set and bits 1 to 7 representing an index to a reserved word table.

The reserved word tokens can be generated directly, reducing the amount of typing required and enabling "longer" statements to be typed in on a single line.

To generate these tokens, it is necessary to hold down the "graphics" key on the keyboard plus one or more other keys. Note that some characters may require "shift" to be held down (e.g.↑ * etc.) and others may also require "control" to be held down too. When typing a program in this way, a graphics character will be generated, but the program will LIST correctly with the reserved words generated in full.

The keys required for each reserved word are as follows:

Reserved Words	Keys Graphics plus	Reserved Words	Keys Graphics plus
END	Control Shift @	NEW	$
FOR	Control A	TAB(	%
NEXT	Control B	TO	&
DATA	Control C	FN	'
INPUT	Control D	SPC(	(
DIM	Control E	THEN	)
READ	Control F	NOT	*
LET	Control G	STEP	+
GOTO	Control H	AND	1
RUN	Control I	OR	2
IF	Control J	SGN	6
RESTORE	Control K	INT	7
GOSUB	Control L	ABS	8
RETURN	Control M	USR	9
REM	Control N	FRE	:
STOP	Control O	INP	;
OUT	Control P	POS	<
ON	Control Q	SQR	=
NULL	Control R	RND	>
WAIT	Control S	LOG	?
DEF	Control T	EXP	shift @
POKE	Control U	COS	A
DOKE	Control V	SIN	B
SCREEN	Control W	TAN	C
LINES	Control X	ATN	D
CLS	Control Y	PEEK	E
WIDTH	Control Z	DEEK	F
MONITOR	Control [	POINT	G
SET	Control \	LEN	H
RESET	Control]	STR$	I
PRINT	Control ↑	VAL	J
CONT	Control _ (underline)	ASC	K
LIST	space	CHR$	L
CLEAR	!	LEFT$	M
CLOAD	"	RIGHT$	N
CSAVE	£	MID$	O