25 BASIC keywords

This chapter contains a description of every word in the Electron BASIC language. These words are called ‘keywords’.

The syntax of each keyword is shown, and an explanation of the form used is given below.

{}	denote possible repetition of the enclosed symbols, zero or more times
[]	enclose optional items
|	indicates alternatives from which only one should be chosen
<num-const>	means a numeric constant such as 4.7 or 112
<num-var>	means a numeric variable such as Y or width
<numeric>	means either a <num-const> or a <num-var>, or a combination of these in an expression, like 4*X+1
<string-const>	means a string enclosed in quotation marks like “JONCRAWFORD”
<string-var>	means a string variable, like A$ or NAME$
<string>	means either a <string-const> or a <string-var>, or an expression such as A$+“ELK”
<testable condition>	means something which is either TRUE or FALSE. Since both TRUE and FALSE have values, it is possible to use <numeric> instead of <testable condition>
<statement>	means any BASIC statement, like PRINT or GOSUB or PROC
<variable name>	means any sequence of letters or numbers which is an acceptable variable name

BASIC keywords

ABS

Absolute Value

Abbreviation None FUNCTION

Description This function gives the modulus; that is, it strips the minus sign from the number variable or expression following it.

Examples PRINT ABS(X) will give 2 if X is -2
deviation = ABS(Temp1-Temp 2)
root = SQR(ABS(Y))

Brackets are optional where sense is not affected.

Syntax <num-var> = ABS(<numeric>)

ACS

Arc-cosine

Abbreviation None FUNCTION

Description This function gives the angle, between 0 and PI in radians, whose cosine is the number variable or expression following ACS. This expression must be between -1 and 1 inclusive.

Examples angle = ACS(0.5)
course = ACS(-0.789)
ANGLE = ACS(AD/HY)

Brackets are optional where sense is not affected.

Syntax <num-var> = ACS(<numeric>)

ADVAL

SOUND channel buffer status

Abbreviation AD. FUNCTION

Description — Gives number of free spaces in SOUND buffers. ADVAL (-5) to ADVAL (-8) correspond to SOUND channels 0 to 3 respectively.

Examples
```
X=ADVAL(-7):PRINT"Free spaces in ch. 2= "";X
IF ADVAL(-5)<>0 THEN SOUND 2, . . .
```

Syntax — <num-var> = ADVAL(<numeric>)

AND

Logical AND

Abbreviation — A. OPERATOR

Description — This is a logical operator which is most commonly used in an IF . . . THEN statement to combine two conditions and obtain a TRUE or FALSE result.

False AND False gives False
False AND True gives False
True AND False gives False
True AND True gives True

If this result is TRUE the computer will go on to the statement following the THEN. If the result is FALSE the computer will go on to the statement following the ELSE, but if the ELSE is absent it will go on to the next line.

Examples
```
IF X<5 AND X>0 THEN PROCmiddle
IF Z=17 AND Y<7 THEN PRINT"YES" ELSE PRINT ""NO""
```

Comments — AND may also be used in the conditional part of a REPEAT . . . UNTIL loop.

Syntax — <num-var> = <numeric> AND <numeric>
<num-var> = <testable condition> AND <testable condition>

ASC

ASCII code

Abbreviation None FUNCTION

Description This function gives the ASCII character value of the first character in the string which follows it. If this string is null it gives -1.

Examples PRINT ASC("JOHN") will give 74 (see ASCII table)
IF ASC(A$) = 78 THEN NEXT.
X = ASC("m")*10

Brackets are optional where sense is not affected.

Syntax <num-var> = ASC(<string>)

ASN

Arc-sine

Abbreviation None FUNCTION

Description This function gives the angle, between -PI/2 and PI/2 in radians, whose sine is the number variable or expression following ASN. This expression must be between -1 and 1 inclusive.

Examples PRINT ASN(OP/HY)
angle = ASN(0.5)

Syntax <num-var> = ASN(<numeric>)

ATN

Arc-tangent

Abbreviation None FUNCTION

Description This function gives the angle, between -PI/2 and PI/2 in radians, whose tangent is the number variable or expression following ATN.

Examples PRINT ATN(OP/AD)
angle = ATN (-3)

Brackets are optional where sense is not affected.

Syntax <num-var> = ATN(<numeric>)

AUTO

Automatic line numbering

Abbreviation AU. or FUNC A COMMAND

Description This command gets the computer to print the next line number and a space each time you press RETURN.

The command has two optional parameters: the first is the starting line number, and the second is the interval between each subsequent line number. The default value of both of these parameters is 10.

Examples AUTO 100,5 will give line numbers 100, 105, 110, 115, etc.
AUTO by itself will give line numbers 10, 20, 30, 40, 50, etc.

Comments The largest allowable line number is 32767, and the largest allowable interval is 255.

You must press ESCAPE to get out of the AUTO mode.

Syntax AUTO[<num-const> [, <num-const>]]

BGET#

Read a byte from file

Abbreviation B.# FUNCTION

Description Reads a single byte from a previously opened file whose channel number follows (see chapter on file handling).

Examples
```
byte = BGET# channel
character = BGET# A
```

Syntax <num-var> = BGET# <num-var>

BPUT#

Store a byte to file

Abbreviation BP.# STATEMENT

Description Stores a single byte on a previously opened file whose channel number follows (see chapter on file handling).

Examples
```
BPUT# channel, number
BPUT# file, Z MOD 256
```

Syntax BPUT# <num-var>, <numeric>

CALL

Call-assembled machine-code subroutine

Abbreviation CA. STATEMENT

Description Used from BASIC to call a previously assembled machine-code subroutine. Similar in operation to a PROC, being capable of passing parameters. Used in preference to a PROC where long calculation is involved, and speed is at a premium.

Examples
```
30 CALL &2000

70 fraction = &16A5
80 CALL fraction

150 CALL fraction, string$, number, integer%, ?byte
```

Syntax CALL <numeric> {Pì ,<num-var> | <string-var>}

CHAIN

Load and run a program

Abbreviation CH. or FUNC K STATEMENT

Description An instruction which LOADs and RUNs the program whose title is in the quotes. If the title is omitted the next program on the tape will be loaded. Can be used in one program to load another. NB all variables except the resident integer variables are cleared.

Examples
```
CHAIN "PROG1"
CHAIN ""
```
Syntax `CHAIN` <string>

CHR$

Character code

Abbreviation `CHR.` FUNCTION

Description Gives the character whose ASCII code is the number variable or expression following CHR$.

Examples
```
PRINT CHR$(32)
A$ = A$ + CHR$(code%)
```

Brackets are optional where sense is not affected.

Syntax <string-var> = `CHR$` (<numeric>)

CLEAR

Clear memory

Abbreviation `CL.` STATEMENT

Description This instruction takes away all the variable names in use, except the resident integer variables A% to Z%, and @%.

Examples
```
IF FNcrash > 30 THEN CLEAR
```

Syntax `CLEAR`

CLG

Clear graphics screen

Abbreviation None STATEMENT

Description Fills the graphics screen with current graphics background colour (which can be altered by the GCOL instruction). The graphics cursor is 'homed' to 0,0 bottom left of graphics screen.

Examples
```
IF X THEN CLG
```

Comments CTRL P has same effect.

Syntax
```
CLG
```

CLOSE#

Close a file

Abbreviation CLO.# STATEMENT

Description Tells the computer you have completely finished with the file whose channel number follows (see chapter on file handling).

Examples
```
CLOSE# (Channel)
CLOSE# file1
```

Comments `CLOSE#0` closes all files.

Syntax `CLOSE#<numeric>`

CLS

Clear text screen

Abbreviation None STATEMENT

Description Fills text screen with current text background colour (which can be altered by the COLOUR instruction). The text cursor is homed to 0,0 the top left of the text screen.

Examples
```
IF X THEN CLS
```

Comments CTRL L has same effect.

Syntax CLS

COLOUR

Abbreviation C. or FUNC C STATEMENT

Description Used to select text screen foreground and background colour.

Standard colours, with their logical values, in each mode are as follows:

Foreground colour			**Background colour**		
Logical no.		Actual colour	Logical No.		Actual colour
Modes 0, 3, 4, 6					
0		Black (0)	128		Black (0)
1		White (7)	129		White (7)
Modes 1,5					
0		Black (0)	128		Black (0)
1		Red (1)	120		Red (1)
2		Yellow (3)	130		Yellow (3)
3		White (7)	131		White (7)
Mode 2					
0		Black (0)	128		Black (0)
1		Red (1)	129		Red (1)
2		Green (2)	130		Green (2)
3		Yellow (3)	131		Yellow (3)
4		Blue (4)	132		Blue (4)
5		Magenta (5)	133		Magenta (5)
6		Cyan (6)	134		Cyan (6)
7		White (7)	135		White (7)
8	Flashing	black/white (8)	136	Flashing	black/white (8)
9	Flashing	red/cyan (9)	137	Flashing	cyan/red (9)
10	Flashing	green/magenta (10)	138	Flashing	magenta/green (10)
11	Flashing	yellow/blue (11)	139	Flashing	yellow/blue (11)
12	Flashing	blue/yellow (12)	140	Flashing	blue/yellow (12)
13	Flashing	magenta/green	141	Flashing	magenta/green (13)
14	Flashing	cyan/red (14)	142	Flashing	cyan/red (14)
15	Flashing	white/black (15)	143	Flashing	white/black (15)

COLOUR takes one parameter, which is the logical value of the particular colour required, as given in the tables..

Examples

```
COLOUR 2
COLOUR 131
```

Comments — Colours used in each mode may be changed using VDU19;.

See chapter 20.

Syntax — COLOUR <numeric>

COS

Cosine

Abbreviation — None — FUNCTION

Description — This function gives the cosine of an angle, which must be in radians.

Examples

```
PRINT COS(3.142)
X = COS(y)
```

Brackets are optional where sense is not affected.

Syntax — <num-var> = COS (<numeric>)

COUNT

Count characters

Abbreviation — COU. — FUNCTION

Description — Counts the number of characters printed using PRINT since last carriage return.

Examples

```
10 PRINT '"Happy Birthday '"";COUNT
20 PRINT '"Happy ""Birthday '';COUNT
>RUN
Happy Birthday 15
Happy
Birthday 9
```

Comments — Different from POS, which gives the position of the cursor from the left hand margin.

Syntax — <num-var> = COUNT

DATA

Data in program

Abbreviation D. STATEMENT

Description This enables you to store information in a program and to recall it using a READ instruction. The information can be string or numeric. (See chapter on READ and DATA).

Examples
```
10 READ A,B$,century
20 DATA 3,GEORGE,18
```

Syntax DATA <str-const> | <num-const> | <num-var> {Pì , <str-const> | <num-const> | <num-var>}

DEF

Define function or procedure

Abbreviation None STATEMENT

Description Informs the computer than an FN or PROC is about to be defined. (See chapters on procedures and functions.)

Examples
```
10 DEF FNdouble(X) = X*2
10 DEF PROCdouble
20 X = X*2
30 ENDPROC
```

Syntax DEF FN| PROC <name>|(<string-var> | <num-var{Pì , <string-var> | <num-var>})]

DEG

Degrees

Abbreviation FUNC H FUNCTION

Description Converts radians into degrees.

Examples
```
angle = DEG(PI/6)
angle = DEG(ACS(0.78))
```

Brackets are optional where sense is not affected.

Syntax <num-var> = DEG<numeric>

DELETE — Delete program lines

Abbreviation DEL. — COMMAND

Description This command will delete a section of program from the first line number stated to the second inclusive. Cannot be used in a program.

Examples
```
DELETE 100,150
```

Comments To delete a single line, just type the line number and press RETURN.

Syntax DELETE <num-const>, <num-const>

DIM

Dimension of an array

Abbreviation None — STATEMENT

Description Informs the computer of how much memory to reserve for a named array. (See chapter on arrays.)

Examples
```
DIM Date$(12,31)
DIM X(100)
```

Comments DIM is also used to allocate space for machine-code programs.

Syntax
DIM <num-var> | <str-var>
(<numeric>{, <numeric>})
DIM <num-var> <numeric>

DIV

Integer division

Abbreviation: None — OPERATOR

Description: This tells the computer to divide one number into another using integer arithmetic; this means the result will always be a whole number.

Examples: 17 DIV 2 gives 8, i.e. the number of times that 2 can be subtracted from 17 with a positive or zero remainder.

Comments: If numbers or variables are used which are not integers, then they will be truncated before the division is carried out.

8.1 DIV 2.9 gives 4.

Syntax: <num-var> = <numeric> DIV <numeric>

DRAW

Draw line on screen

Abbreviation: DR. or FUNC D — STATEMENT

Description: Will draw a line from the previous coordinates of the graphics cursor to the new ones given, in all graphics modes (0, 1, 2, 4 and 5). To move the graphics cursor use the MOVE instruction.

The screen is always 0 to 1279 on the X axis and 0 to 1024 on the Y axis, regardless of which graphics mode you are in. The line is drawn in the current graphics foreground colour which can be changed by using the GCOL instruction.

Examples:

```
10 MODE 4
20 MOVE 0,512
30 DRAW 1279,512
```

will draw a horizontal line half way up the screen.

See chapter 20.

Syntax: DRAW <numeric> , <numeric>

ELSE

Abbreviation EL. or FUNC E (See IF)

Description Used to provide an alternative course of action if the result of an IF statement is false.

Examples
```
IF A=0 THEN PRINT"YES" ELSE PRINT"NO"
IF B THEN 100 ELSE 200
```

Syntax IF <testable condition> THEN <statement> ELSE <statement>

END

Abbreviation None STATEMENT

Description Can be used to halt execution of a program. Its other use is to reset TOP after a PAGE move.

Examples
```
PAGE = &1600:END
60 IF finished THEN END
```

Syntax END

ENDPROC

End of procedure

Abbreviation E. STATEMENT

Description This statement must conclude a DEF PROC as it tells the computer you have finished defining the procedure.

Examples
```
100 DEF PROCname
110 REM statement
120 REM statement
130 ENDPROC
```

Syntax ENDPROC

ENVELOPE

Abbreviation ENV. STATEMENT

Description This statement is used in conjunction with the SOUND statement to control the pitch of a sound. The ENVELOPE statement is followed by 14 parameters.

ENVELOPE n,s,Pi1,Pi2,Pi3,Pr1,Pr2,Pr3,126,0,0,-126, 126,126

Parameter	Range	Effect
n	1 to 4	Envelope number
bits 0-6	0 to 127	Length of each step in 1/100 of a second
s bit 7	0 or 1	0 = auto-repeat the envelope 1 = no auto-repeat
Pi1	-128 to 127	Change in pitch per step in section 1
Pi2	-128 to 127	Change in pitch per step in section 2
Pi3	-128 to 127	Change in pitch per step in section 3
Pr1	0 to 255	Number of steps in section 1
Pr2	0 to 255	Number of steps in section 2
Pr3	0 to 255	Number of steps in section 3

See chapter 22.

Syntax ENVELOPE <var-num>, <var-num>, <var-num>, <var-num>, <var-num>, <var-num>, <var-num>, <var-num>, <var-num>, <var-num>, <var-num>, <var-num>, <var-num>, <var-num>

EOF#

End of file check

Abbreviation None FUNCTION

Description This function is used to discover whether the end of an open file has been reached. The function gives a -1 if the end has been reached and a 0 if not. EOF# must be followed by the channel number.

Examples
```
IF EOF# (channel) THEN PROCclose
REPEAT . . . :UNTIL EOF#(X)
```

Syntax <num-var> = EOF#(<var-num>)

EOR

Logical exclusive-OR

Abbreviation None OPERATOR

Description This is used in an IF . . . THEN or REPEAT . . . UNTIL loop to combine two conditions in the following way:

False EOR False gives False
False EOR True gives True
True EOR False gives True
True EOR True gives False

In other words, if the results of the two conditions combined by an EOR are different then the result is true.

Examples
```
IF A=6 EOR B < 10 THEN GOSUB 420
```

Syntax <num-var> = <numeric> EOR <numeric>

EQUB

Abbreviation None STATEMENT

Description Used to insert a byte of data into an Assembly Language program. EQUB can only be used inside the square brackets enclosing a piece of Assembly Language.

Examples
```
EQUB 13
EQUB A%
```

Syntax EQUB <numeric>

EQUD

Abbreviation None STATEMENT

Description Used to insert a double-word (4 bytes) of data into an Assembly Language program. EQUD can only be used inside the square brackets enclosing a piece of Assembly Language.

Examples
```
EQUD 10000000
EQUD F%
```

Syntax EQUD <numeric>

EQUS

Abbreviation None STATEMENT

Description Used to insert the ASCII values of a string into an Assembly Language program. EQUS can only be used inside the square brackets enclosing a piece of Assembly Language.

Examples
```
EQUS "Too big"
EQUS L$
```

Comments — Used, among other things, for printing error messages in Assembly Language programs. Unlike the indirection operator $, EQUS does not add a RETURN (&D) to the end of the string.

Syntax — EQUS <string>

EQUW

Abbreviation — None — STATEMENT

Description — Used to insert a word of data (2 bytes) into an Assembly Language program. EQUW can only be used inside the square brackets enclosing a piece of Assembly Language.

Examples
```
EQUW &FFE0
EQUW Z%
```

Syntax — EQUW <numeric>

ERL

Error line number

Abbreviation — None — FUNCTION

Description — A function which gives the line number in which the last error occurred.

Examples
```
X = ERL
REPORT:PRINT " at line ";ERL
```

Syntax — <num-var> = ERL

ERR

Error

Abbreviation None FUNCTION

Description A function which gives the numeric code for the last error which occurred. This is useful for error trapping.

Examples
```
IF ERR=17 THEN CLOSE#(channel)
```

Syntax <num-var> = ERR

EVAL

Evaluate

Abbreviation EV. FUNCTION

Description Mainly used to enable you to type an expression, such as a mathematical equation, into the computer while a program is running. The equation is entered as a string, e.g. A$ = "COS(X/20)", and EVAL(A$) will work it out.

Examples
```
A$ = "COS(X/20)"
Y = EVAL(A$)
```

Syntax
<num-var> = EVAL (<string>)
<str-var> = EVAL (<string>)

EXP

Exponent

Abbreviation None FUNCTION

Description This mathematical function calculates the exponential *e* (2.7183 ...) raised to any specified power.

Examples
```
Y = EXP(X)
```
i.e. Y = e to the power of X

Syntax <num-var> = EXP (<numeric>)

EXT#

Reserved for future use.

FALSE

Abbreviation FA. CONSTANT

Description This is a condition which the computer understands to be the number 0. If the computer decides a certain condition is false it will represent it as 0, and will act accordingly.

Examples
```
REPEAT . . . : UNTIL FALSE
IF A=FALSE THEN . . .
```

Comments
```
PRINT 1 = 2
```

gives 0, because 1 is not equal to 2, and so 1=2 is FALSE.

Syntax <num-var> = FALSE

FN

User-definable function

Abbreviation None Prefix

Description FN is a prefix that identifies a function, both in a DEF statement and in a function call.

Examples
```
10 INPUT A
20 answer=FNsquare(A)
30 PRINT answer
40 END
50 DEF FNsquare(number)=number*number

DEF FNe = 2.7182818
```

Syntax DEF FN <name> [(<num-var>|<str-var>{Pì , <num-var>|<str-var>})]

FOR

Abbreviation F. STATEMENT

Description FOR is used to initiate the control variable of the FOR. . . NEXT loop and will always take the following format:

FOR		=	number or	TO	STEP
(numeric variable)			numeric variable)	(n or nv)	(n or nv)
control variable			start parameter	finish parameter	increment

If the STEP is omitted it is assumed to be 1.

When executing a FOR . . . NEXT loop the computer sets the control variable to the start parameter.

Each time NEXT is encountered the computer increments the control variable and loops back to the instruction just after FOR. This is repeated until the control variable is greater than the finish parameter. NB the increment can be negative.

Syntax FOR<num-var> = <numeric> TO<numeric> [STEP<numeric>]

GCOL

Graphics colour

Abbreviation GC. STATEMENT

Description Used to select graphics screen foreground and background colour, and to control effect of mixing colours. Takes two parameters, the second being the logical value of the colour required, the first the way in which two colours mix.

Action of first parameters is as follows:

0 Plot the colour specified
1 OR the colour specified with those already on the screen

2 AND the colour specified with those already on the screen
3 EOR the colour specified with those already on the screen
4 Plot the logical inverse of the colour specified

This mixing is carried out on a bit by bit basis. For 1, 2 and 3, each binary digit in the plotted colour's logical value is ORed, ANDed, or EORed with its respective digit in the screen colour's logical value, to produce the logical colour which is to be plotted on the part of the screen.

Inversion 4, only involves the plotted colour, all its binary digits being inverted. In bit by bit logic, 0 is false and 1 is true.

The truth tables for OR, AND and EOR are as follows:

0 OR 0 gives 0
0 OR 1 gives 1
1 OR 0 gives 1
1 OR 1 gives 1

0 AND 1 gives 0
0 AND 0 gives 0
1 AND 0 gives 0
1 AND 1 gives 1

0 EOR 0 gives 0
0 EOR 1 gives 1
1 EOR 0 gives 1
1 EOR 1 gives 0

Examples

```
GCOL 2,1
GCOL RND(5)-1, RND(8)-1
GCOL mix%, 129
```

Comments

Colours used in each mode may be changed using VDU19;

See chapter 20.

Syntax

GCOL<numeric>,<numeric>

GET

Get code from keyboard

Abbreviation None FUNCTION

Description This instruction causes the computer to read a character from the keyboard buffer. If there is none, the computer will wait for a key to be pressed. It then gives the ASCII code for that key (see ASCII table) before continuing.

Examples `Key=GET`

Comments

The keyboard buffer may be flushed by `*FX15`.

Syntax <num-var> = `GET`

GET$

Get character from keyboard

Abbreviation `GE.` FUNCTION

Description This instruction is the same as `GET`, but gives a string containing the character before continuing.

Examples `Key$=GET$`

Syntax <string-var> = `GET$`

GOSUB

Go to a subroutine

Abbreviation `GOS.` STATEMENT

Description This instruction tells the computer to go to a subroutine and start executing instructions form the specified line number until the instruction `RETURN`, when the computer must return to the instruction immediately after the `GOSUB` call. No more than 26 nested subroutines are allowed.

Examples
```
GOSUB 1000
ON A GOSUB 10, 20 30
```

Comments — It is possible to use an expression, and brackets must then be used:

```
GOSUB (10*A)
```

but this will not work if the program is RENUMBERed.

Syntax — GOSUB <numeric>

GOTO

Go to a line number

Abbreviation — G. or FUNC G — STATEMENT

Description — This instruction tells the computer to jump to the specified line number and start executing instructions there.

Examples
```
GOTO 100
ON A GOTO 10, 20, 30
```

Comments — It is possible to use an expression:

```
GOTO (10*A)
```

but this will not work if the program is RENUMBERed.

Syntax — GOTO <numeric>

HIMEM

Abbreviation — H. — VARIABLE

Description — Address pointer containing the address of the lowest location in memory used by screen display. Its value may change depending upon which mode you are

using. No BASIC instructions or variables are stored above this point. HIMEM can be altered by the user, to preserve space for machine-code programs between BASIC and the screen. HIMEM contains the highest memory location that BASIC may use. This may be changed after a MODE statement.

Examples

```
PRINT HIMEM
HIMEM = &4000
```

Syntax

<num-var> = HIMEM

IF

Conditional IF

Abbreviation None STATEMENT

Description A word forming part of the IF. . . THEN . . . ELSE statement. If must always be followed by a testable condition, and the result of this test (TRUE or FALSE) will control the subsequent action by the computer.

Examples

```
IF A<5 THEN . . .
IF IQ>140 THEN PRINT"GENIUS"
```

Comments

```
IF A THEN 110
```

means

```
IF A<>0 THEN GOTO 110
```

Syntax

IF <testable condition> THEN<statement>
IF <testable condition> THEN<statement>
[ELSE<statement> | <line number>]

INKEY

Abbreviation None FUNCTION

Description (i) This instruction will wait a specified length of time for a key to be pressed before continuing. If a key is pressed, its ASCII code is given but if no key is pressed a -1 is given. The number in brackets is the time delay measured in 1/100 sec, and can have any

value between 0 and 32767.

Examples

```
key = INKEY(100)
```

Comments

In that this instruction and GET, GET$ and INKEY$ will actually test the keyboard buffer. This means that an INKEY instruction will respond to any previously pressed key whose code has gone into this buffer (memory), even if you are not pressing it at this moment.

To get over this problem, the keyboard buffer can be flushed using a *FX15,1 instruction just before using INKEY. Also, the autorepeat can be turned off by using *FX11,0.

Description (ii)

In addition to the above, the INKEY instruction can be used to test for a single key directly. Using a negative number in the brackets, one for each key according to the table shown below, INKEY gives -1 if the key is pressed, 0 if it is not. INKEY used in this way does not read the buffer — it reads the key itself. See the table which follows.

Examples

```
IF INKEY(-99) THEN . . .
```

will be TRUE when the space-bar is pressed.

Brackets are optional where sense is not affected.

Key	Number	Key	Number
A	-66	1	-49
B	-101	2	-50
C	-83	3	-18
D	-51	4	-19
E	-35	5	-20
F	-68	6	-53
G	-84	7	-37
H	-85	8	-22
I	-38	9	-39
J	-70	0	-40
K	-71	-	-24
L	-87	;	-88
M	-102	:	-73
N	-86	,	-103
O	-55	.	-104
P	-56	/	-105
Q	-17	SPACE BAR	-99
R	-52	ESCAPE	-113
S	-82	CAPS LK	-65
T	-36	CTRL	-2
U	-54	SHIFT	-1
V	-100	DELETE	-90
W	-34	COPY	-106
X	-67	RETURN	-74
Y	-69	↑	-58
Z	-98	↓	-42
		←	-26
		→	-122

Syntax <num-var> = INKEY (<numeric>)

INKEY$

Abbreviation INK. FUNCTION

Description This instruction halts the program, print a ? on the screen, and waits for some information to be entered followed by RETURN.

Examples
```
Key$ = INKEY$(100)
```
Brackets are optional where sense is not affected.

Syntax <string-var> = INKEY (<numeric>)

INPUT

Abbreviation I. or FUNC I STATEMENT

Description This instruction halts the program, prints a ? on the screen, and waits for some information to be entered followed by RETURN.

INPUT must be followed by a variable.

Examples
```
INPUT X
INPUT name$, number
```

Syntax Too complex for a simple description.

INPUT# Input from file

Abbreviation I.# STATEMENT

Description INPUT# is like INPUT, but instead of receiving information from the keyboard the computer takes it from a previously opened file. Must be followed by channel number (see chapter on file handling).

Examples

```
INPUT# channel, make$, price
INPUT# C,B,A,Z$
```

Syntax

INPUT# <num-var>, <num-var> | <string-var> {Pì , <num-vat> | <string-var>}

INSTR

In string

Abbreviation INS. Includes (FUNCTION

Description

This function will give the position of one string within another, the leftmost character position being 1. The search normally starts from the beginning of the string but an optional third parameter provides the facility to start the search from any specific character position. The number given by INSTR is the position of the second string within the first. A search for a null string ("") will always give 1. If the search fails (the two strings are not the same at any position) then 0 is given.

Examples

```
position = INSTR(first$, second$, start)
PRINT INSTR("MONOTONOUS","ON")
```

will print 2, whereas

```
PRINT INSTR("MONOTONOUS","ON",3)
```

will print 6

Syntax

<num-var> = INSTR(<string>,<string> [,<numeric>])

INT

Integer part

Abbreviation None FUNCTION

Description

This function returns the next whole number below the value of the number variable or expression in brackets. In other words the number is truncated.

Examples

```
INT (1.7) is 1
INT (-1.7) is -2
x=INT(Y)
```

Brackets are optional where sense is not affected.

Syntax <num-var> = INT <numeric>

LEFT$

Left string

Abbreviation LE. Includes (Left string

Description A function which gives the specified number of leftmost characters in a string.

Examples

```
PRINT LEFT$("ELECTRON",5)
```

will give ELECT

```
A$ = LEFT$(B$,C)
```

Syntax <string-var> = LEFT$(<string>,<numeric>)

LEN

Length of string

Abbreviation None FUNCTION

Description A function which gives the number of characters (including spaces) in the specified string.

Examples

```
PRINT LEN("Donald Duck")
```

will be 11

```
Length=LEN(A$)
```

Brackets are optional where sense is not affected.

Syntax <num-var> = LEN(<string>)

LET

Abbreviation — None — STATEMENT

Description — This is an optional keyword which is used to assign a value to a variable.

Examples — LET X=10 has the same effect as X=10
LET A$="JOHN" has the same effect as A$="JOHN"

Comments — May not be used with LOMEM, HIMEM, PAGE and TIME.

Syntax — LET <var> = <expression>

LIST

Abbreviation — L. or FUNC L — COMMAND

Description — This command instructs the computer to list the current program on the screen. It has two optional parameters which control the first and last lines to be listed.

Examples — LIST 100,200 will list from 100 to 200
LIST ,200 will list up to 200
LIST 100, will list from 100

Comments — Since LIST is a command it cannot be used in a program or in a multi-statement line.

If you press CTRL N beforehand, LIST will list your program a screen-full at a time. When you want to see the next screen-full, press SHIFT. This paged mode can be cancelled using CTRL O.

Syntax — LIST [<num-const>[,]<num-const>]

LISTO

List option

Abbreviation: None — COMMAND

Description: This command must be followed by a number which controls the way in which a program is LISTed, as follows:

0 List just as stored in computer's memory
1 Inserts a space after each line number
2 Indent FOR . . . NEXT loops
4 Indent REPEAT . . . UNTIL loops

Any combination of the above may be obtained by adding the required values.

Examples: LISTO5 will insert a space after the line number and indent REPEAT . . . UNTIL loops.

Comments: Since LISTO is a command it cannot be used in a program or in a multi-statement line.

Syntax: LISTO <num-const>

LN

Natural logarithm

Abbreviation: None — FUNCTION

Description: A mathematical function to calculate the natural logarithm of the given number variable or expression.

Examples: X = LN(Y)

Brackets are optional where sense is not affected.

Syntax: <num-var> = LN <numeric>

LOAD

Load program from file

Abbreviation LO. or FUNC , COMMAND

Description A command which instructs the computer to LOAD the named program from the file. If the name is omitted then the next program is loaded.

When the computer prints 'Loading' on the screen, the old program has been deleted and all variables cleared except the resident integer variables.

Examples
```
LOAD "BUGZAP!"
LOAD "" loads the next program (from tape only)
```

Comments During LOADing, the computer prints up the number of pages of memory being used.

Syntax LOAD <string>

LOCAL

Variable declaration

Abbreviation LOC. or FUNC Q STATEMENT

Description Informs the computer that the named variables are LOCAL to the PROC or FN in which they are declared.

LOCAL variables are totally independent of variables with the same name outside the PROC or FN.

Examples
```
LOCAL I
LOCAL price%
```

Syntax LOCAL <string-var>|<num-var>{Pì ,<string-var>|<num-var>}

LOG

Common logarithm

Abbreviation None FUNCTION

Description A mathematical function to calculate the common logarithm of the given number variable or expression.

Examples
```
Y = LOG(X)
rate = LOG(cone)
```

Syntax <num-var> = LOG<numeric>

LOMEM

Abbreviation LOM. VARIABLE

Description Address pointer containing the address above which all the BASIC program's variables are stored. It is usually set to be the same as TOP, but can be altered by the user at the start of a program.

Examples
```
PRINT LOMEM
0 LOMEM = &FA2
```

Comments If LOMEM is changed during program execution the computer will lose all its BASIC variables

Syntax
```
LOMEM = <numeric>
<num-var> = LOMEM
```

MID$

Middle string

Abbreviation M. Includes (FUNCTION

Description This function gives a subsection of a string; the position of the first character of the substring and the number of characters being specified. If the length is

omitted, the whole string to the right of the start position is given.

Examples

```
PRINT MID$(Main$, start, length)

PRINT MID$("MICROCOMPUTER",6,7)
```
will be COMPUTE

```
PRINT MID$("MICROCOMPUTER",6)
```
will be COMPUTER

```
X$=MID$(A$,S,L)
```

Syntax <string-var> = MID$(<string>, <numeric> [,<numeric>])

MOD

Modulo

Abbreviation None OPERATOR

Description This function gives the remainder when an integer division is carried out.

Examples

5 MOD 2 is 1
55 MOD 5 is 0
-10 MOD 4 is -2
numerator% MOD denominator%

Comments If this function is used with decimal numbers or variables, these values are truncated before the division takes place.

4.1 MOD 3.9 is 1.

Syntax <num-var> = <numeric> MOD <numeric>

MODE

Graphics mode

Abbreviation: MO. or FUNC M — STATEMENT

Description: Here is a list of the seven modes and their characteristics:

Mode	Graphics	Colours	Text
0	640 × 256	2	80 × 32
1	320 × 256	4	40 × 32
2	160 × 256	16	20 × 32
3	Text only	2	80 × 25
4	320 × 256	2	40 × 32
5	160 × 256	4	20 × 32
6	Text only	2	40 × 25

This instruction tells the computer to change screen mode. Changing mode clears the screen and must not be used within a PROC or FN. MODE resets the value of HIMEM.

Examples:

```
MODE 0
MODE x
10 MODE mode
```

Comments: Text coordinates change from mode to mode, but graphics coordinates are scaled to be the same in all graphics modes: 0 to 1279, 0 to 1023.

Syntax: MODE <numeric>

MOVE

Move graphics cursor

Abbreviation: MOV. — STATEMENT

Description: This instruction moves the graphics cursor to any position, on or off the screen.

Examples:

```
MOVE 640,52
10 MOVE X,Y
```

Comments — The graphics origin may be moved to any position on or off the screen by using the instruction

```
VDU 29,X;Y;
```

where X,Y are the coordinates you wish to become 0,0.

See chapter 20.

Syntax — MOVE <numeric>, <numeric>

NEW

New program

Abbreviation — None — COMMAND

Description — This command deletes a program from the computer's memory by adjusting certain internal pointers. A program can be retrieved by typing OLD which resets the internal pointers, provided no new program lines have been entered and no new variables have been created. Since it is a command it cannot be used in a program or multi-statement line.

Examples

```
NEW
```

Syntax

```
NEW
```

NEXT

Abbreviation — N. or FUNC N — STATEMENT

Description — This is used to step the control variable in a FOR. . . NEXT loop. The control variable after the NEXT is optional (see FOR).

Examples

```
NEXT X%
NEXT loop
NEXT
```

Syntax — NEXT [<num-var>] {, <num-var>}

NOT

Logical NOT

Abbreviation: None — FUNCTION

Description: Normally used in conjunction with a testable condition to reverse the logic of the result, i.e. TRUE becomes FALSE and FALSE becomes TRUE.

Examples:
```
IF NOT(A=5) THEN money = 70
```

Comments: TRUE and FALSE are represented as -1 and 0 respectively.

NOT 0 is -1, NOT -1 is 0.

Beware of trying to use NOT with other values for TRUE. NOT1 is -2, which still acts as TRUE.

Syntax:
```
<num-var> = NOT <numeric>
<testable condition> = NOT (<testable condition>)
```

OLD

Old program

Abbreviation: O. or FUNC O (includes RETURN) — COMMAND

Description: This command is used to recover a program which has recently been deleted by NEW, or by pressing the BREAK key, or CTRL BREAK.

Examples:
```
OLD
```

Syntax:
```
OLD
```

ON

Abbreviation None STATEMENT

Description This instruction can be used in conjunction with GOTO, GOSUB and ERROR.

Firstly GOTO and GOSUB.

```
ON X GOTO 100, 300, 350, 470
```

If X=1 then the program will go to 100. If X=2 then it will go to 300. If X=3 then it will go to 350 and so on.

```
ON X GOSUB 475, 205, 310
```

If X=1 then the program will 'gosub' 475. If X=2 then it will 'gosub' 205 and so on.

An ELSE can be included at the end to trap out of range values without causing an error.

```
ON X GOTO 70, 190, 310 ELSE ENDPROC
```

Secondly ERROR

```
ON ERROR GOTO 1000
ON ERROR RUN
ON ERROR PROCerror
```

This instruction is used for *error trapping*. This enables the program to deal with errors, rather than letting the computer halt the program and print up an error message.

Comments Errors may be accepted once again by typing

```
ON ERROR OFF
```

when the computer will halt and print messages as usual.

Syntax

ON<num-var> GOTO<numeric>,{Pì ,<numeric>}
or
ON<num-var> GOSUB<numeric>{Pì ,<numeric>}
or
ONERROR <statement>
or
ONERROR OFF

OPENIN

Open input file

Abbreviation OP. FUNCTION

Description This function opens a file from the current filing system, e.g. cassette to the computer and returns the channel number allocated by the computer's file system. If the file does not exist then a 0 (FALSE) is returned. The file is opened for input only. See chapter on file handling.

Examples

```
file = OPENIN "SCREENLOAD"
X = OPENIN A$
```

Syntax <num-var> = OPENIN (<string>)

OPENOUT

Abbreviation OPENO. FUNCTION

Description This function opens a file from the computer to the current filing system, e.g. cassette and returns the channel number allocated by the computer's file system. If the file does not exist then one will be created. If a file of the same name already exists then one will be deleted and a new one created. The file is opened for output only.

Examples

```
file = OPENOUT "SCREEND"
X = OPENOUT A$
```

Syntax <num-var> = OPENOUT(<string>)

OPENUP

Open file for update

Abbreviation None FUNCTION

Description This function opens a file from the current filing system, e.g. cassette to the computer, in the same way as OPENIN. The file is opened for input and output.

Examples
```
edit = OPENUP "“Accounts"
Z = OPENUP name$
```

Syntax <num-var> = OPENUP(<string>)

OPT

Assembly option

Abbreviation None STATEMENT

Description Used to select whether error messages are reported, or listings are given during assembly of a machine-code subroutine. OPT can only be used inside the square brackets enclosing a piece of Assembly Language.

It may take eight different values of parameter.

OPT 0 Report no errors, list no machine-code.
OPT 1 Report no errors, list the machine-code.
OPT 2 Report any errors, list no machine-code.
OPT 3 Report any errors, list the machine-code.

OPT 4 to OPT 7 are the same as OPTs 0 to 3 except that the machine-code is generated at the origin O% instead of the program counter P%.

Examples
```
50 OPT 2
100 FOR I=0 TO 3 STEP 3
110 [OPT I
120 ....
130 ....]
140 NEXT
```

Comments — The second example above is commonly used in two-pass assembly, which will always give errors on the first pass.

Syntax — OPT <numeric>

OR

Logical OR

Abbreviation — None — OPERATOR

Description — This is a logical operator which combines two testable conditions in the following way:

False OR False gives False
False OR True gives True
True OR False gives True
True OR True gives True

Examples

```
IF A=5 OR A=10 THEN ...
IF Altitude OR B% THEN ...
IF X<2 OR NOT(Y=4) THEN ...
```

Syntax — <num-var> = <numeric> OR <numeric>

OSCLI

Operating System Command Line Interpreter

Abbreviation — OS. — STATEMENT

Description — Used to pass BASIC variables as parameters to Operating System calls, such as *FX or *KEY. Must be used with a string, the variables being converted using STR$. The complete string must be the same as would be used normally, but without the asterisk.

Examples

```
OSCLI"FX"+STR$ action+","+STR$ parameter
OSCLI"KEY"+STR$ keynumber+keystring$
```

PAGE

Abbreviation PA. VARIABLE

Description Address pointer containing the address above which the BASIC program is stored. It is usually set to &E00 on cassette only machine, but can be altered by the user to locate more than one BASIC program in memory.

Examples
```
PRINT PAGE
PAGE = &1600
```

Comments The two least significant hex digits of PAGE are always zero - it points to the base of a page of memory, which always contains 256 bytes.

Syntax
```
PAGE = <numeric>
<num-var> = PAGE
```

PI

Abbreviation None CONSTANT

Description PI has the value 3.1459265 and can be used just like a number.

Examples
```
circumference = 2*PI*radius
area = PI*radius^2
```

Syntax
```
<num-var> = PI
```

PLOT

Plot graphics

Abbreviation PL. or FUNC P STATEMENT

Description The PLOT instruction is used to draw single points, lines, dotted lines and triangles.

PLOT takes the form:
PLOT A,X,Y

which will plot at or to the point X,Y in the manner determined by the value of the first parameter A.

The effect of the value of the first parameter is:

0 Move relative to last point.

1 Draw line relative in the current graphics foreground colour.

2 Draw line relative in the logical inverse colour.

3 Draw line relative in the current graphics background colour.

4 Move to absolute position.

5 Draw line absolute in the current graphics foreground colour.

6 Draw line absolute in logical inverse colour.

7 Draw line absolute in current graphics background colour.

High values of A have other effects which are related to the effects given by the values 0 to 7

8-15 As 0-7 but with the last point in the line omitted in 'inverting actions' — eg using GCOL4.

16-23 As 0-7 but with a dotted line.

24-31 As 0-7 but with a dotted line and without the last point on the line.

32-63 Are reserved for the Graphics Extension ROM.

64-71 As 0-7 but only a single point is plotted.

72-79 Line fill.

80-87 As 0-7 but plot and fill a triangle. When filling solid triangles with colour the computer fills the triangle between the coordinates given and the last two points visited.

88-95 Line fill.

See chapter 20.

Syntax PLOT <numeric>, <numeric>, <numeric>

POINT

Point at graphics screen colour position

Abbreviation PO. FUNCTION

Description This function gives the logical colour at the specified point on the screen. If this point is off the screen then a -1 is given.

Examples colour = POINT(X,Y)

Comments This function is used in the MARSLANDER program to test whether the capsule has touched down, and whether it is on a flat surface.

Syntax <num-var> = POINT (<numeric>, <numeric>)

POS

Position of text cursor

Abbreviation None FUNCTION

Description This function tells how far across the text screen the text cursor is. The value to the right is determined by the mode, and by the size of text window.

Examples X = POS

Syntax <num-var> = POS

PRINT

Print on screen

Abbreviation: P. or FUNC / STATEMENT

Description: This instruction is used for all character output to the screen.

Examples:

```
PRINT "Anywhere"
PRINT A,B,length,moon$,BILL%
PRINT HEIGHT*DEPTH;CHR$127;99;
```

PRINT CHR$X; is almost the same in operation and effect as VDUX, and the two are interchangeable.

Comments: PRINT is also used to issue control-codes to the computer during program execution.

PRINT CHR$9 will move the text cursor forward one square, for example.

Syntax: PRINT {["][,|;]<string>|<numeric>}["][;]

PRINT#

Print on file

Abbreviation: P.# STATEMENT

Description: PRINT# is like PRINT but instead of printing information to the screen it prints it on to a previously opened file. Must be followed by channel number (see chapter on file handling).

Examples:

```
PRINT# channel,make$,prices
PRINT# C,B,A,Z$
```

Syntax: PRINT# <num-var>{["][,<numeric>|<string>}

PROC

Procedure

Abbreviation — FUNC X — Prefix

Description — This prefix is used when defining a named procedure (see DEF) and to call this named procedure from anywhere in the program.

Examples

```
PROCrotate
IF . . . THEN PROCfire
DEF PROCfire
PROCvolume(radius,height)
```

Comments — PROC must be followed by a name without any spaces. Parameters may be passed in brackets.

Syntax — DEF PROC <variable-name> [(string-var> | <num-var> {["][,<string-var> | <num-var>}]

PTR#

Reserved for future use.

RAD

Radians

Abbreviation — FUNC J — FUNCTION

Description — This function gives an angle in radians which is equivalent to the specified angle in degrees. There are 2*PI radians in a circle of 360°.

Examples

```
X = RAD(X)
angle = RAD(angle)
answer = SIN(RAD(angle))
```

Brackets are optional where sense is not affected.

Syntax — <num-var> = RAD <numeric>

READ

Read data into variable(s)

Abbreviation None STATEMENT

Description This instruction tells the computer to copy information from a DATA statement into the variables which follow the READ instruction. The types of variables must match; numbers must be copied into numeric variables and strings into string variables. See DATA and chapter on READ and DATA.

Examples
```
READ name$,Tel,credit
```

Syntax READ <num-var>|<string-var> {,<num-var>| <string-var>}

REM

Remark

Abbreviation None STATEMENT

Description This instruction tells the computer to ignore the rest of the program line, thus enabling the programmer to insert comments in the listing without affecting the program.

Examples
```
10 REM Marslander
1035 REM Move Alien
```

Syntax REM <anything>

RENUMBER

Renumber program line

Abbreviation REN. or FUNC 8 COMMAND

Description This command has two optional parameters which control the way in which a program is to be renumbered. The value of the first parameter is the starting line number. The second is the increment for

each subsequent line number. If either or both parameters are omitted they will default to 10.

Examples

If a program looks like this

```
10 xxx
15 xxx
17 xxx
30 xxx
32 xxx
```

RENUMBER 100,5 will change it to

```
100 xxx
105 xxx
110 xxx
115 xxx
120 xxx
```

Comments

This command also renumbers all GOTOs, GOSUBs and any other cross references, except calculated ones like GOTO (8*W) and GOSUB (100+Z).

Syntax

RENUMBER [<num-const> [,<num-const>]]

REPEAT

Abbreviation

REP. or FUNC Y

STATEMENT

Description

This instruction initiates a REPEAT . . . UNTIL loop which tells the computer to REPEAT a set of instructions UNTIL a testable condition becomes true.

REPEAT . . . UNTIL loops may be nested up to a depth of 20.

Examples

```
NOW = TIME:REPEAT UNTIL TIME - NOW >= 500
```

Syntax

REPEAT

REPORT

Abbreviation REPO. STATEMENT

Description This instruction will print up on the screen what the last error was, in words.

Examples
```
REPORT: PRINT " at line ";ERL
```

Syntax REPORT

RESTORE

Abbreviation RES. or FUNC W STATEMENT

Description This instruction restores the 'DATA pointer' to the beginning of a specified line. The DATA pointer points to the next piece of information to be read by a READ instruction. If no line is specified the DATA pointer is restored to the beginning of the first DATA instruction in the program.

Examples
```
RESTORE
RESTORE 2500
```

Syntax RESTORE <numeric>

RETURN

Return from subroutine

Abbreviation R. STATEMENT

Description This instruction informs the computer that it has reached the end of a subroutine and that it must now RETURN to the instruction immediately after the GOSUB at which the subroutine was called. A subroutine must not be exited other than by using RETURN.

Examples
```
RETURN
IF A=0 THEN RETURN
```

Comments Not to be confused with the RETURN key.

Syntax `RETURN`

RIGHT$

Right string

Abbreviation RI. Includes (FUNCTION

Description A function which gives the specified number or right-most characters in a string.

Examples `PRINT RIGHT$("FLAVOUR", 3)` will give OUR.

Syntax <string-var> = `RIGHT$`(<string>,<numeric>)

RND

Random number generator

Abbreviation None FUNCTION

Description This function, which may be followed by a number in brackets, returns a random number.

`RND` by itself generates a random whole number between -2147483648 and 2147483647

`RND(-X)` gives the value -X and resets the random number generator to a number based on X.

`RND(0)` repeats the last random number given by `RND(1)`.

`RND(1)` generates a random number between 0 and 0.999999.

`RND(X)` generates a random whole number between (and possibly including) 1 and X.

The brackets are compulsory and must immediately follow the word `RND` with no intervening space.

Examples X = RND(10)

Syntax <num-var> = RND [(<numeric>)]

RUN

Abbreviation FUNC R (includes RETURN) STATEMENT

Description This instruction makes the computer initiate execution of the numbered program lines in its memory. It also clears all variables except the resident integer variables.

Examples RUN
IF velocity > 100 THEN RUN

Comments A program can be RUN without clearing the variables by using the command GOTO, followed by the first line number.

Syntax RUN

SAVE

Abbreviation SA. or FUNC . STATEMENT

Description This transfers the program from the computer's current program area (between the system variables PAGE and TOP) on to cassette, and in future expansions, disc as well. When used with tape, SAVE must be followed by a name of up to 10 characters, inside quotation marks.

Examples SAVE "BUGZAP!"

Syntax SAVE <string>

SGN

Sign

Abbreviation None FUNCTION

Description This function tells you whether the specified number, variable or expression is positive, zero or negative.

The function gives:

-1 for a negative number
0 for a zero
+1 for a positive number

Examples `X = SGN(Y)`

Brackets are optional where sense is not affected.

Syntax <num-var> = SGN(<numeric>)

SIN

Sine

Abbreviation None FUNCTION

Description This function gives the sine of an angle, which must be in radians.

Examples
```
PRINT SIN(3.142)
X = SIN(y)
```

Brackets are optional where sense is not affected.

Syntax <num-var> = SIN(<numeric>)

SOUND

Abbreviation SO. STATEMENT

Description Makes the computer generate a sound on the internal loudspeaker.

The format is:

SOUND Q, A, P, D

Q is the channel number, 0 to 3.

A is the envelope number, 0 to 4. If A is 0 then that sound channel is turned off. If A is negative (-1 for example) then a pure tone is produced.

P is the pitch, 0 to 255.

D is the duration, 1 to 255 in twentieths of a second.

See chapter 22.

Syntax SOUND <numeric>, <numeric>, <numeric>, <numeric>

SPC

Space

Abbreviation None (See PRINT, INPUT)

Description This statement is used in conjunction with PRINT or INPUT to give the specified number of spaces. This number may not be greater than 255.

Examples
```
PRINT "Name";SPC(10);"Tel.";SPC(10);"CREDIT"
INPUT "Amount" SPC(3) A
```

Brackets are optional where sense is not affected.

Syntax
PRINT SPC (<numeric>)
INPUT SPC (<numeric>)

SQR

Square root

Abbreviation None FUNCTION

Description This function gives the square root of a positive number.

Examples
```
X = SQR(Y)
ans = SQR(ABS(NUMBER))
```

Brackets are optional where sense is not affected.

Syntax <num-var> = SQR(<numeric>)

STEP

Abbreviation S. or FUNC S (See FOR)

Description This is part of the FOR . . . TO . . . STEP . . . NEXT statement. The number following STEP is the amount by which the control variable is incremented in each loop, and can be positive or negative, integer or real.

Examples
```
FOR X=0 TO 50 STEP 5
FOR I=10 TO 0 STEP -1
FOR J=7 TO 11.3 STEP Z*.7
```

Syntax FOR <num-var> = <numeric> TO <numeric> [STEP <numeric>]

STOP

Abbreviation None STATEMENT

Description This instruction has the same effect as END but it prints the message STOP, and the line number, on the screen.

Examples
```
IF P > Temp THEN STOP
```

Syntax STOP

STR$

String

Abbreviation STR. FUNCTION

Description This function converts any number or expression in the brackets into a string. STR$ has an opposite effect to that of VAL.

Examples
```
A$ = STR$(X)
B$ = STR$(-1.23)
```

Brackets are optional where sense is not affected.

Syntax <string-var> = STR$ (<numeric>)

STRING$

Abbreviation STRI. Includes (FUNCTION

Description This instruction produces a long string consisting of a specified number of copies of a shorter string.

Examples
```
Line$ = STRING$(40,"-")
```

Syntax <string-var> = STRING$ (<numeric>,<string>)

TAB

Abbreviation None (See PRINT, INPUT)

Description Used with either PRINT or INPUT to set the position of the text cursor on the screen.

TAB(X) will move the cursor forward to position X on the current line. X can be between - and 19, 0 and 39, or 0 and 79 depending upon which mode is in use.

TAB(X,Y) will move the cursor directly to position X,Y. The text cursor has its origin 0,0 at the top left. If a text window is in use, then TAB will treat the top left corner of that as 0,0.

Examples

```
PRINTTAB(6,11);"HELLO"
INPUT "How much" TAB(10), cost
```

Comments — The brackets are compulsory, and there must be no space between TAB and (.

Syntax

```
PRINT TAB(<numeric>, [,<numeric>])
INPUT TAB(<numeric>, [,<numeric>])
```

TAN

Abbreviation — T. — FUNCTION

Description — This function gives the tangent of an angle, which must be in radians.

Examples

```
PRINT TAN(PI/2)
ratio = TAN(.6)
```

Brackets are optional where sense is not affected.

Syntax

```
<num-var> = TAN<numeric>
```

THEN

Abbreviation — TH. or FUNC T — (See IF)

Description — Provides the course of action if the result of an IF statement is true.

It is an optional keyword, but the program is neater and easier to follow when it is left in.

Examples IF A = B THEN PROCab

Comments THEN is not optional when assigning certain resident variables such as TIME, and when GOTO is omitted.

Syntax IF <testable condition> THEN <statement> [ELSE <statement>]

TIME

Abbreviation TI. VARIABLE

Description An integer variable which is incremented every hundredth of a second. It serves as an elapsed time clock, and can be set to any initial value by the user.

Examples
```
TIME =0
T% = TIME
PRINT T%
```

Syntax
```
TIME = <numeric>
<num-var> = TIME
```

TO

Abbreviation None (See FOR)

Description Used in the FOR . . . NEXT loop to set the limiting value of the control variable.

Examples FOR I = 0 TO 11

Syntax FOR <num-var> = <numeric> TO <numeric> [STEP <numeric}]

TOP

Abbreviation None VARIABLE

Description Address pointer containing the address of the first free memory location after the top of the BASIC program. TOP-PAGE will give the length of your BASIC program in bytes.

Examples
```
PRINT TOP-PAGE
```

Syntax <num-var> = TOP

TRACE

Abbreviation TR. STATEMENT

Description Debugging device which prints BASIC line numbers in order or execution. Is turned on by TRACE ON, and off by TRACE OFF. TRACE X will only give line numbers below X.

Syntax TRACE ON OFF <numeric>

TRUE

Abbreviation None CONSTANT

Description This is a condition which the computer understands to be the number -1. If the computer decides a certain condition is true, it will represent it as -1, and will act accordingly.

Examples
```
IF A=TRUE THEN . . .
Test = TRUE
```

Comments — In practice, any number other than 0 is taken by the computer to be TRUE. Care is needed though. NOT(-1) is 0. NOT(1) is -2. This is because the NOT function simply inverts the binary digits, and does not consider them true or false at all.

Syntax — <num-var> = TRUE

UNTIL

Abbreviation — U. or FUNC U (See REPEAT)

Description — Conditional part of REPEAT . . . UNTIL loop. The loop is executed until the conditional statement after UNTIL goes true.

Examples

```
REPEAT
xxx
xxx
UNTIL X=7
```

```
REPEAT          These are       REPEAT
xxx             the same        xxx
xxx             and will        xxx
UNTIL 0         loop forever    UNTIL FALSE
```

Syntax — UNTIL <testable condition>

USR

Abbreviation — None FUNCTION

Description — Used from BASIC to call a previously assembled machine-code function. Similar in operation to an FN, but is not able to pass parameters. Used in preference to an FN where long calculation is involved, and speed is at a premium.

Examples

```
X = USR(&1750)

20 address = &30A9
30 PRINT USR(address)
```

Comments — See chapter on assembler.

Syntax — <num-var> = USR (<numeric>)

VAL

Abbreviation — None — FUNCTION

Description — Gives the numeric part of a string as a number. The string must start with +, -, or a number, otherwise 0 is given.

Examples

```
number = VAL(-762*12)
```

will put number equal to -762.

Syntax — <num-var> = VAL(<string>)

VDU

Abbreviation — V. or FUNC V — STATEMENT

Description — VDU has almost the same function as PRINT CHR$. It can be used to give any character or control code from the ASCII table in Appendix F.

Examples

```
VDU5    Link text and graphics cursors.
VDU8    Move text cursor back one square.
VDU23   Re-define character.
```

Comments — Sends code directly to the VDU drivers. Is quicker to type than PRINT CHR$.

PRINT TAB(X,Y) is equivalent to
VDU 31,X,Y

Syntax VDU <numeric> {, | ; <numeric>} [;]

VPOS

Abbreviation VP. FUNCTION

Description Gives the distance of the text cursor from the top of the screen or text window.

Examples
```
Y = VPOS
PRINT VPOS
```

Syntax <num-var> = VPOS

WIDTH

Abbreviation W. STATEMENT

Description Sets the width used by the computer to print on the screen. This is normally unlimited, and the computer runs on to the next line at the right boundary of the screen or text window.

Examples WIDTH 10 will cause the text to be printed in a column 10 characters wide.

Syntax WIDTH <numeric>