BBC BASIC Keywords

ABS	**Absolute value (function)**
	Returns the absolute positive value of its argument.
ACS	**Arc cosine (function)**
	Returns the arc cosine of its argument in radians. The permitted range of the argument is -1 to +1- For example, `PRINT DEG(ACS(0.5))` will print 60, because COS(60°) is 0.5.
AND (A.)	**Logical AND (operator)**
	Performs a bitwise logical AND between two operands which are internally converted to 4 byte integers before the operation. It is normally used to join two conditions in an IF or UNTIL statement; thus `IF length >10 AND width > 10 THEN PRINT "OK"` ensures that a rectangle is larger than 10 x 10.
ASC	**ASCII value (function)**
	Returns the ASCII character value of the first character of the argument string. `PRINT ASC("Cambridge Z88")` gives 90, the ASCII value of "Z". The brackets are optional, and `ASC""`, the null string, gives `-1`.
ASN	**Arc sine (function)**
	Returns the arc sine of its argument in radians. The permitted range of the argument is -1 to +1.
ATN	**Arc tangent (function)**
	Returns the arc tangent of its argument in radians.
AUTO (AU.)	**Automatic numbering (command)**

	Allows lines to be entered without first typing in the number of the line. The line numbers are preceded by the usual prompt (>).
	By default the starting line number and increment are both 10, but they may optionally be specified; for example
	`AUTO 20`
	will start with line 20, and
	`AUTO 100,20`
	will start at line 100 and increment by 20. AUTO will continue generating line numbers until you press ESC.

BGET (B.#)	**Byte from data file (function)**
	Returns a byte from the data file whose channel number is its argument. The file pointer is incremented after the byte has been read. For example, `character=BGET#c` reads the next character from file c.
BPUT# (BP.#)	**Output a byte (statement)**

Puts a byte to the data file whose channel number is the first argument. The second argument's least significant byte is written to the file. The file pointer is incremented after the byte has been written. Thus

```
BPUT#channel, char
```

writes char to file channel.

CALL (CA.) **Call machine-code (statement)**

Calls a machine code subroutine at a specified address, passing parameters in a parameter block addressed by the Z80's IX register. The IY register is set to the address of the machine code subroutine being called. The processor's A, B, C, D, E, F, H, and L registers are initialised to the least significant bytes of A%, B%, C%, D%, E%, F%, H%, and L% respectively.

This statement could cause corruption of the Cambridge Z88 memory, and should therefore only be used by experienced programmers.

The parameter block contains the following list:

number of parameters	1 byte	(1X+0)
first parameter type	1 byte	(1X+1)
first parameter address	2 bytes	(1X+2,1X+3)
parameter type	) repeated as often	
parameter address	) as necessary	

where the parameter types are as follows:

Type	Description
0	8 bit bytes (eg ?a)
4	32 bit integer variable (eg !b or c%)
5	40 bit floating point number (eg d)
128	A string at a defined address (eg $e terminated by a &0D)
129	A string variable such as f$

In the case of a string variable the parameter address is the address of a String Information Block which gives the current length of the string, number of bytes allocated, and start address, in that order.

```
muldiv=1234
CALL muldiv,A,B$,C%
```

CHAIN (CH.) **Load and run program (statement)**

Loads and runs the program whose name is specified in the argument allowing one program to load another. Information can be passed between programs using the static variables @%, and A% to Z%.

CHR$ **ASCII character (function)**

Returns the ASCII character string specified by the least significant byte of the numeric argument.
`A$=CHR$(90)`

will set A$ to 70, since ASC"Z" is 90. The characters corresponding to values 32 and above can be displayed on the screen with the line

```
FOR N%=32 TO 255: PRINT CHR$(N%) ; : NEXT
```

CLEAR (CL.) **Clear program (statement)**

Clears all variables, including strings apart from the static variables @%, and A% to Z%.

CLG **Clears Graphics Window**

	This clears the graphics window (only); it does not affect the position of the graphics cursor. **Note:** CLS can be used to clear the text window and leave the graphics window unchanged.
CLOSE (CL.#)	**Close channel (statement)**
	Closes a specified channel. For example, `CLOSE#c` closes channel c.
CLS	**Clear text area (statement)**
	Clears the text area of the screen. The text cursor is moved to the 'home' position (0,0) at the top left-hand character position of the text area.
COLOUR	
	not implemented.
COS	**Cosine (function)**
	Returns the cosine of its argument in radians. `X=COS(angle)`
COUNT (COU.)	**Character count (function)**
	Returns the number of characters sent to the output stream (VDU or printer) since the last new line. For example, `PRINT A$::REPEAT PRINT ".";:UNTIL COUNT = 72` will pad the line with dots to 72 characters
DATA (D.)	**Data (statement)**
	Introduces lists of data for use by the READ statement (see READ).
DEF	**Define function/procedure (statement)**

	Precedes declaration of a user-defined function (FN) or procedure (PROC). DEF must be used at the start of a program line.
	
	For example,
	
	`DEF FNcelsiusff) = (f-32)*5/9`
	
	defines a function to convert Fahrenheit to Celsius.
	
	Executing
	
	`PRINT FNcelsius(98.4)`
	
	will convert 98.4 to Celsius.

DEG	**Degrees (function)**
	Returns the argument converted from radians to degrees. For example `PRINT=DEG(PI/2)` will print 90
DELETE (DEL.)	**Delete lines (command)**

	Deletes a group of lines from the program. Both start and end lines of the group will be deleted. For example `DELETE 123,456` will delete all lines between 123 and 456 inclusive, which need not exist.
DIM	**Dimension array (statement)**

	Dimensions an array, or reserves an area of memory for special applications. For example,
	`DIM a$(10,20)`
	dimensions a two-dimensional string array a$ with elements a$(0,0) up to a$(10,20). Arrays may have one or more dimensions, and may be string arrays, floating-point arrays, or integer arrays.
	`DIM X%24`
	reserves 25 bytes and puts the address of byte 0 in the variable X%.

DIV	**Integer divide (operator)**
	Gives the integer quotient of two items. The result is always an integer. X=A DIV B y=(top+bottom+1) DIV 2
DRAW x,y	**Draw black straight line**
	Draws a straight line (in black) between the current position of the graphics cursor and the specified coordinates, then moves the graphics cursor to the specified position. This statement is identical to PLOT 5.
ELSE (EL.)	**Else clause (statement)**
	An optional part of the IF...THEN, or ON...GOSUB, ON...GOTO statements, it introduces the action to be taken if the testable condition evaluates to FALSE, or the ON expression is out of range.
END	**End program (statement)**
	Returns to direct mode.
ENDPROC	**End procedure (statement)**
	Denotes the end of a procedure defined with DEF PROC.
EOF#	**End of file (function)**

	Returns -1 (TRUE) if the end of the specified data file has been reached. For example,
	`REPEAT`
	`char%=BGET#data`
	`...`
	`UNTIL EOF#data`
	will read characters until the end of the file whose channel number is the variable `data`.

EOR	**Logical Exclusive-OR (operator)**
	Performs a bitwise integer logical exclusive-or between two operands which are internally converted to 4 byte integers before the operation.
ERL	**Error line (function)**
	Returns the line number of the line where the last error occurred. For example, `PRINT "Error number" ERR "at line" ERL`
ERR	**Error code (function)**
	Returns the error code number of the last error which occurred.

EVAL (EV.)	**Evaluate string (function)**

	Returns the result of evaluating the given expression supplied as a string. For example,
	`a=6 : b=7`
	`PRINT EVAL ("a + b")`

EXP	**Exponent (function)**
	Returns 'e' (2.71828183) to the power of the argument.
EXT#	**Extent of file (function)**
	Returns the total length of the file whose channel number is its argument. The file must have been opened with OPENIN, OPENUP, or OPENOUT.
FALSE (FA.)	**False (function)**
	Returns the value zero representing logical false. For example, `REPEAT PRINT "*" : UNTIL FALSE` will continue forever.
FN	**Function (statement)**
	Introduces a user-declared function. The first character of the function name can be a letter, underline, or a number. No spaces are allowed between the function name and the opening bracket of the parameter list (if any).
FOR (F.)	**Start FOR loop (statement)**
	Initialises a FOR ... NEXT loop. The loop is executed at least once for each of the values of the control variable in the specified range. FOR card=1 TO 6 PRINT card; NEXT card will print 1 2 3 4 5 6
GCOL	
	not implemented.
GET/GET$	**Wait for key (function)**
	Waits for a key to be pressed on the keyboard. GET returns the ASCII value, and GET$ returns the corresponding single-character string. For example, `REPEAT UNTIL GET = 13` waits for to be pressed.
GOSUB (GOS.)	**Call subroutine (statement)**

	Calls a section of a program as a subroutine at a specified line number. Control returns to the next statement when RETURN is encountered in the subroutine. One subroutine may call another subroutine (or itself).
	
	`100 GOSUB 120`
	`110 END`
	`120 PRINT "Hello"`
	`130 RETURN`

GOTO (G.)	**Go to line (statement)**
	Transfers program control to a line with a specified or calculated line number. For example, `GOTO 100 GOTO (X*10)` The use of the calculated GOTO, as in the second example, is not recommended as it will not be renumbered correctly by the RENUMBER command.
HIMEM (H.)	**High memory bound (function)**
	A pseudo-variable which contains the address of the first byte of free memory.
IF	**Condition (statement)**

	Sets up a test condition which can be used to control the subsequent flow of the program. It is part of the IF ... THEN ... ELSE structure. The word THEN is optional under most circumstances.
	
	`IF length-5 THEN 110`
	`IF A<C OR A>D GOTO 110`
	`IF A>C AND C>-D THEN GOTO 110 ELSE PRINT "CCL"`

INKEY/INKEY$	**Read key (function)**
	Waits for up to a specified number of clock ticks (10ms each). If no key is pressed in the time limit, INKEY will return -1 and INKEY$ will return a null string; otherwise the INKEY function will return the ASCII value of the key pressed.
INPUT (I.)	**Input value (statement)**
	Inputs values from the keyboard. The INPUT statement normally prints a ? prompt for each variable in the list. Alternatively strings can be included in the list of variables to be printed as prompts; omitting the comma after a string will suppress the question mark. For example: `INPUT"Enter your age:" age%, "and your name",name$`
INPUT LINE	**(statement)**
	Identical to INPUT except that the entire line, including commas, quotes and leading spaces is input into a string variable. `INPUT LINE A$`
INPUT#	**Input from file (statement)**
	Reads data from a file into specified variables. The data should have been written to the file with a corresponding PRINT# statement.
INSTR	**Substring (function)**

	Returns the position of a substring within a string, optionally starting the search at a specified place in the string. The leftmost character position is 1. If the sub-string is not found, 0 is returned.
	For example,
	`PRINT INSTR("Cambridge Z88"."8")`
	will print 12, and
	`PRINT INSTR("PipeDream","e".5)`
	will start the search at character 5 and print 7.

INT	**Integer (function)**

	Converts a real number to the next lower or equal integer.
	`INT(99.8)` is `99`
	`INT(-12)` is `-12`
	`INT(-12.1)` is `-13`.

LEFT$	**Left of string (function)**
	Returns a specified number of characters from the left of a string. If there are insufficient characters in the source string, all the characters are returned. Thus, if `A$=" BANANA"` `PRINT LEFT$(A$,3)` would print `"BAN"`.
LEN	**Length of string (function)**
	Returns the length of the argument string. For example, `X=LEN"fred"` will set X to 4.
LET	**Assignment (statement)**
	Optional before an assignment statement.
LIST (L.)	**List program (command)**
	Lists the program. Examples: `LIST` lists the entire program `LIST ,99` lists up to line 99 `LIST 11,` lists from line 11 to the end `LIST 11,99` lists lines 11 to 99 inclusive `LIST 55` lists line 55 only To obtain a listing of a program to a printer connected to the Cambridge Z88: Attach and turn on the printer. Type `LIST`, type ◻+P, and press ENTER When finished, type ◻ -P.
LISTO	**LIST options (command)**

	Controls the appearance of a listed program. The number following the command specifies which of the following formatting options are required. **Value** / **Option** 0 — No inserted spaces 1 — Space after line number 2 — FOR ... NEXT loops indented 4 — REPEAT ... UNTIL loops indented The numbers can be added to combine options, the default being 7.
LN	**Natural logarithm (function)**
	Returns the natural (Naperian) logarithm of its argument.
LOAD (LO.)	**Load program (command)**

	Loads a new program from a file and clears the variables. For example
	
	`LOAD "STAT4"`
	
	or
	
	`LOAD ":RAM.O/CONVERTER.BAS"`

LOCAL (LOC.)	**Local variables (statement)**
	Declares variables for local use inside a function (FN) or procedure (PROC). LOCAL A$,X,Y%
LOG	**Logarithm (function)**
	Returns the base-10 logarithm of its argument.
LOMEM (LOM.)	**Lower memory bound (function)**
	A pseudo-variable which controls where in memory the dynamic data structures are to be placed. The default is TOP, the first free address after the end of the program.
MID$	**Middle of string (function)**
	Returns a string consisting of a specified number of characters of the string starting from a given position. For example `A$ = MID$(B$, start, length)` sets A$ to the substring of B$ starting at position 'start', and of length 'length'. If 'length' is omitted, or if there are insufficient characters in the string, then all the characters from 'start' onwards are returned. Thus `PRINT MID$("DOZY",2,2)` will print "OZ"
MOD	**Modulo (operator)**

	Gives the signed remainder of the integer division.
	
	`X=A MOD B`
	
	is equivalent to
	
	`X = A - ((A DIV B) *B)`

MODE n	**MODE statement**

	The MODE statement allows selection of the normal text-only mode (MODE 0) or a text-and-graphics mode (MODE 1). In MODE 1 the display is split into two parts: a text-window on the left and a graphics-window on the right. The text window consists of 8 rows of 50 characters, and the graphics window is 64 pixels high by 256 pixels wide; you cannot (normally) mix text and graphics in the same window. Points in the graphics window are addressed as x,y coordinates from 0,0 (the bottom-left corner) to 255,63 (the top-right corner), although the origin can be moved using the PLOT -1 statement (q.v.). Although MODE 1 sets up the window positions and sizes as described, it is possible to change these using the VDU statement. However the method of doing this is outside the scope of this document. It is not advisable to cause the text and graphics windows to overlap, although this may occasionally be useful. MODE clears the display (both text and graphics windows), moves the text cursor to 0,0 (the top left of the text window), resets the graphics origin and moves the graphics cursor to 0,0 (the bottom left of the graphics window). In MODE 0 (the normal 94-column text mode) the other graphics statements have no effect.
MOVE x,y	**Move graphics cursor**
	Moves the graphics cursor to the specified coordinates, but does not affect what is displayed. This statement is identical to PLOT 4.
NEW	**New program (command)**
	Initialises the interpreter for a new program to be typed in. An old program may be recovered with the OLD command provided no program lines have been typed in. The variables @% and A% to Z% are preserved even after a NEW command.
NEXT (N.)	**End FOR loop (statement)**
	Ends a FOR ... NEXT loop. NEXT takes an optional control variable; if this is not the same as the variable supplied in the corresponding FOR statement, an error will occur.
NOT	**Logical NOT (operator)**
	A unary operator (the same priority as unary –) giving a bit-by-bit binary inversion of the constant, variable, or mathematical or boolean expression to its right. Usually used in IF ... THEN or UNTIL statements to invert the sense of the condition. Expressions must be enclosed in brackets.
OLD	**Recover old program. (command)**
	Undoes the effect of NEW provided no lines have been typed in or deleted, no variables have been created, and no popdown or application has been entered.
ON	**Multi-way switch (statement)**
	Provides a multi-way GOTO or GOSUB, depending on the value of a control variable. The line numbers in the list may be constants or calculated, and the unwanted ones are skipped without calculation. For example: `ON action GOSUB 1000,2000,3000,4000`
ON ERROR	**Error trap (statement)**

	Provides error trapping. If an ON ERROR statement has been encountered, BASIC will transfer control to it (without taking any reporting action) when an error is detected. This allows error reporting/recovery to be controlled by the program. However, the program control stack is still cleared when the error is detected and it is not possible to return to the point where the error occurred.
	
	Note that under some circumstances ON ERROR can cause BASIC to generate repeated errors, requiring a soft-reset. This can be avoided by including a call to INKEY$, as in the following example, which will allow you to exit from BASIC to the Index and *KILL the activity:
	
	`10 ON ERROR REPORT: 0$=INKEY$(100)`
	`20 ERROR`

OPENIN (OP.)	**Open file for input (function)**
	Opens a file for reading or updating and returns the 'channel number' of the file, or 0 on failure. This number must be used in subsequent references to the file with BGET#, INPUT#, EXT#, PTR#, EOF# or CLOSE#.
OPENOUT	**Open file for output (function)**

	Opens a file for writing and returns the 'channel number' of the file, or 0 on failure. This number must be used in subsequent references to the file with BPUT#, PRINT#, EXT#, PTR# or CLOSE#. `X=OPENOUT(A$) X=OPENOUT(":RAM.0/DATA.DAT")`
OPENUP (OPENU.)	**Open file for update (function)**
	Opens a file for update and returns the channel number, or 0 on failure. Once a file is opened you can update it or extend it.
OPT	**Assembler options (statement)**

	An assembler pseudo operation controlling the method of assembly.
	
	It is followed by a number in the range 0 to 3 to specify the method of assembly:
	
	Option / **Action**
	0 — errors suppressed, no listing
	1 — errors suppressed, listing
	2 — errors reported, no listing
	3 — errors reported, listing
	
	The code is assembled into memory at the address specified by P%. For example:
	
	`10 DIM code 100`
	`20 FOR pass = 0 TO 3 STEP 3`
	`30 P%=code`
	`40 [`
	`50 OPT pass`
	`60  .....`
	`70 \ backslash introduces a comment`
	`80 \ standard Z80 mnemonics are used`
	`90 ]`
	`100 NEXT pass`
	`110 END`
	
	Alternatively the assembled code can be assembled into memory at the address specified by O%, with labels generated according to the value of P%, by adding 4 to each of these option values.
	
	For more details, read the BBC BASIC and the in-line assembler in the Developers' Notes.

OR	**Logical OR (operator)**
	Gives the bitwise integer logical OR between two operands which are internally converted to 4 byte integers before the operation.
OSCLI	**Operating-system command (statement)**
	Allows a string expression to be passed to the operating system. For example, in the BASIC editor (see Editing BASIC programs, p. 191) `60220 OSCLI "*CLI.<" + B$`
PAGE (PA.)	**Program area (function)**
	A pseudo-variable controlling the starting address of the current user program area. The lower byte of PAGE is always zero.
PI	**Pi (function)**

	Returns 3.141592653.
PLOT n,x,y	**Plotting statement**
	A multi-purpose plotting statement, whose effect is controlled by the first parameter n:

n	action
-1	Move the graphics origin to x,y.
0	Move the graphics cursor relative to the last point.
1	Draw a line, in "black", relative to the last point.
2	Draw a line, in "inverse", relative to the last point.
3	Draw a line, in "white", relative to the last point.
4	Move the graphics cursor to the absolute position x,y.
5	Draw a line, in "black", to the absolute position x,y.
6	Draw a line, in "inverse", to the absolute position x,y.
7	Draw a line, in "white", to the absolute position x,y.
8-15	As 0-7, but plot the last point on the line twice (i.e. in the "inverting" modes omit the last point).
16-31	As 0-15, but draw the line dotted.
32-63	As 0-31, but plot the first point on the line twice (i.e. in the "inverting" modes omit the first point).
64-71	As 0-7, but plot a single point at x,y.
72-79	Draw a horizontal line left and right from the point x,y until the first "lit" pixel is encountered, or the edge of the window. This can be used to fill shapes.
80-87	Plot and fill a triangle defined by the two previously visited points and the point x,y.
88-95	Draw a horizontal line to the right of the point x,y until the first "unlit" pixel is encountered, or the edge of the window. This can be used to "undraw" things.
96-103	Plot and fill a rectangle whose opposite corners are defined by the last visited point and the point x,y.

POINT(x,y)	**Returns state of pixel**
	This function returns the state of the pixel at the specified location, as 0 (unlit) or 1 (lit). If the specified point is outside the graphics window (taking into account the position of the graphics origin), or if MODE 0 is selected, the value -1 is returned.
POS	**Cursor position (function)**
	Returns the horizontal position of the cursor on the screen. The left-hand column is 0 and the right-hand column is one less than the width of the display.
PRINT (P.)	**Print text (statement)**

	Prints characters on the screen or printer.
	
	Items may be separated by commas, semi-colons, or no separator:
	
	`PRINT "ONE" I% "TWO"`
	
	Numbers are printed in a format determined by the value of the variable @%. This is set to a hexadecimal value as follows:
	
	@% = &SSNIVPPWW
	
	where
	
	SS determines the format of strings created by STR$.
	
	If SS=01 then STR$ will use the format specified by 9%, otherwise @% will be ignored.
	
	NN determines the notation format.
	
	NN=00 General notation: integers will be printed with no decimal places, numbers between 0.1 and 1 will be printed as 0.1 etc, and numbers less than 0.1 will use scientific notation.
	
	NN=01 Scientific notation: eg 100 is printed as 1E2.
	
	NN=02 Fixed format notation: if the number will fit into the specified field width, it will be displayed with the number of decimal places specified by PP. Otherwise general notation will be used.
	
	PP determines the number of decimal places to be printed.
	
	WW determines the overall print field width.
	
	By default, @% = &0000090A, giving general notation with a field width of 10 characters. Its value can be printed, in hexadecimal, with
	
	`PRINT ~@%`
	
	Numbers can be printed in hexadecimal by prefixing them with '-'.

PRINT# (P.#)	**Write to file (statement)**
	Writes the internal form of a list of variables, separated by commas, to a specified data file.
PROC	**Procedure (statement)**
	Introduces a user-declared procedure. The first character of a procedure name can be a letter, underline, or a number. No spaces are allowed between the procedure name and the opening bracket of the parameter list (if any). The procedure returns to the calling program with an ENDPROC statement.
PTR#	**File pointer (function)**
	A pseudo-variable allowing the random-access pointer of a specified file to be read and changed. For example, `PTR#F=PTR#F+5` moves to the next floating-point number in the file with channel number F, since 5 bytes are allocated to each number.
PUT	**Output to port (statement)**
	Outputs data to a Z80 port. Should only be used by experienced programmers as incorrect use could damage the Cambridge Z88. The full Z80 extended addressing is available. `PUT A,N :REM output N to port A.`
RAD	**Radians (function)**
	Returns its argument converted from degrees to radians. For example: `RAD(90)`
READ	**Read DATA statements (statement)**

	Assigns to variables values read from the DATA statements in the program. Strings must be enclosed in double quotes if they have leading spaces or contain commas.
	
	`READ A%,B,C$`
	`DATA 27,-12.34,"Hello"`

REM	**Comment (statement)**
	Introduces a comment, causing the rest of the line to be ignored.
RENUMBER (REN.)	**Renumber program (command)**
	Renumbers the lines and corrects the cross references inside a program. The options are as for AUTO.
REPEAT (REP.)	**REPEAT loop (statement)**
	Introduces a REPEAT...UNTIL loop. For example: `REPEAT PRINT "*": UNTIL COUNT = 80` will print 80 stars on the screen or printer.
REPORT (REPO.)	**Report error (statement)**
	Prints out the error string associated with the last error which occurred. If no error has occurred, prints the copyright string.
RESTORE (RES.)	**Restore READ (statement)**
	Sets the line from which subsequent READ statements will read data.
RETURN (R.)	**Return from subroutine (statement)**
	Causes a RETURN to the statement after the most recent GOSUB statement.
RIGHT$	**Right of string (function)**
	Returns a specified number of characters from the right-hand end of a string. If there are insufficient characters in the string then all are returned. For example: `PRINT RIGHT$("DOZY",3)` will print 0 Z Y -
RND	**Random number (function)**
	Returns a random number. The type and range of the number returned depends upon the optional parameter, as follows:

Value	Result of RND(X)
X<0	Returns X and resets random number generator to -X.
X=0	Repeats last random number given by RND(1).
X=1	Returns a random number between 0 and 0.999999.
X>1	Returns a random integer between 1 and X inclusive.

RUN	**Run program (statement)**
	Starts execution of the program after clearing all but the static variables @%, and A% to Z%.
SAVE (SA.)	**Save program (statement)**
	Saves the current program area to a file, in internal (tokenised) format. `SAVE "myprog.BAS" SAVE A$`

SGN	**Sign (function)**
	Returns -1, 0, or + 1 depending on whether the argument is negative, zero or positive respectively. `result=SGN(answer)`
SIN	**Sine (function)**
	Returns the sine of its argument taken in radians.
SPC	**Print spaces (statement)**
	Prints a specified number of spaces. SPC can only be used as part of an INPUT or PRINT list; for example: `PRINT "Name"; SPCM; "Age"; SPC(6): "Address"`
SQR	**Square root (function)**
	Returns the positive square root of its argument. `z=SQ R (XA 2+y A2)`
STEP (S.)	**FOR loop increment (statement)**

	Part of the FOR statement, this optional section specifies the step size. For example,
	`FOR i=1 TO 20 STEP 5`
	`PRINT i;`
	`NEXT`
	will print: `1 6 11 16`

STOP	**Stop program (statement)**
	Syntactically identical to END, STOP also prints a message `STOP at line X` where X is the line number.
STR$	**String (function)**
	Returns the string form of the numeric argument as it would have been printed. A number A% can be converted to a string in hexadecimal format with the function: `STR$~A%`
STRING$	**Repeat strings (function)**
	Returns a given number of repetitions of a string. `A$=STRING$(5,"HA")` will set AL$ to "HAHAHAHAHA"
TAB	**Move to screen position (statement)**
	Moves the cursor to a given screen position. TAB can only be used as part of a PRINT or INPUT statement. There are two forms: TAB(X) will print spaces until the cursor reaches column X (on the same line, or next line). TAB(X,Y) will move the cursor directly to character position X,Y on the screen, where 0,0 corresponds to the top left-hand corner.
TAN (T.)	**Tangent (function)**
	Returns the tangent of its argument taken in radians.
THEN (TH.)	**Condition clause (statement)**

	An optional part of the IF ... THEN ... ELSE statement. It introduces the action to be taken if the testable condition evaluates to TRUE.
TIME (TI.)	**Time (function)**

	A pseudo-variable which sets and reads the elapsed time clock. The value of TIME must be initialised before it is used; for example
	`TIME=100`
	resets the value of TIME to 100 centiseconds, and
	`X=TIME`
	sets X to the value of the current elapsed time.

TIME$	**Time string (function)**

	Returns a string giving the current date and time; for example:
	`Wednesday 29 April 1987, 10:12:32`
	The functions LEFT$, MIID$, and RIGHT$ can be used to extract parts of this string. For example, the time alone can be obtained by
	`A$ = RIGHT$(TIME$,8)`
	`PRINT A$`
	10:12:32

TO	**Upper bound of FOR loop (statement)**
	Introduces the terminating value f6r the loop in a FOR ... TO ... STEP statement. When the loop control variable exceeds the value following 'TO', the loop is terminated.
TOP	**Top of program (function)**
	Returns the value of the first free location after the end of the current program.
TRACE (TR.)	**Trace program (command)**
	TRACE can be used to provide information on the execution of a program. TRACE ON causes the interpreter to print executed line numbers when it encounters them, for debugging. The facility can be turned off by typing TRACE OFF.
TRUE	**True (function)**
	Returns the value —1, representing logical true.
UNTIL (U.)	**End REPEAT loop (statement)**
	The end of a REPEAT ... UNTIL structure.
USR	**Call machine-code (function)**
	Enters a machine code routine at the address specified in its argument, passing the least-significant bytes of the integer variables A%, B%, C%, D%, E%, F%, H%, L%, and F% into the correspondingly-named registers of the Z80 on entry. Unlike CALL, USR returns a 32-bit result composed of the contents of the Z80's H, L, H', and L' registers, most-significant to least-significant. This function should only be used by experienced programmers.
VAL	**Value of string (function)**
	Converts a character string representing a number into numeric form. `X=VAL(a$)`
VDU (V.)	**Output bytes to screen (statement)**

	Takes a list of numeric arguments and sends their least-significant bytes as characters to the current 'output stream'.
VPOS (VP.)	**Vertical position (function)**
	Returns the vertical cursor position. The top of the screen is line 0.
WIDTH (W.)	**Screen width (statement)**
	Controls output overall field width. Initially WIDTH is 94 (the default). For example, before printing it is a good idea to set `WIDTH 80` to give a new line after every 80 characters of output.
***CLI**	**Send line to CLI (OZ command)**
	Sends a command to the command line interpreter. For example, in the BASIC editor (see Editing BASIC programs): `60090 *CLI .*:RAM.O/EE.CLI`
***EDIT (*E.)**	***EDIT line number**
	This command allows you to edit a specified program line. It results in the line being displayed (after a short delay) with the cursor positioned at the end, and you can then edit the line using any of the usual line-editing features, as follows:

⇦	Move cursor left one character
⇨	Move cursor right one character
SHIFT ⇦	Move cursor left one word
SHIFT ⇨	Move cursor right one word
◇ ⇦	Move cursor to start of line
◇ ⇨	Move cursor to end of line
DEL	Backspace and delete
SHIFT	Delete character under cursor
◇ DEL	Delete entire line
◇ **D**	Delete from cursor to end of line
◇ **G**	Delete character under cursor
◇ **S**	Swap case
◇ T	Delete up to next space
◇ **U**	Insert space at cursor position
◇ **V**	Toggle between insert and overtype

To enter the edited line into the program press ENTER; to abandon the edit and leave the line unchanged press ESC.

You can also use ***EDIT** to concatenate two or more program lines, by specifying the first line and last line separated by commas (e.g. *EDIT 10,30). In this case you will have to edit out the line numbers of the second and subsequent lines (and delete the old lines afterwards).

*EDIT may be abbreviated to *E. (the dot is required).

*ERASE	Erase a file (OZ command)

	Erases a specified file. For example,
	`*ERASE filename`
	or using the CLI
	`OSCLI "*ERASE "+file$`

***NAME**	**Names the BASIC activity (OZ command)**
	Gives the BASIC activity a name, which will be displayed in the list of SUSPENDED ACTIVITIES in the Index. `*NAME MYPROG`
***RENAME**	**Rename a file (OZ command)**

	Renames a file; for example:
	`*RENAME oldfile newfile`
	`OSCLI "*RENAME "+oldfile$+" "+newfile$`

Notes on V3.10 BBC BASIC

1. You are advised to select MODE 0 before entering Pipedream, since it seems to get confused by the presence of the graphics window.
2. If you reply to the INPUT statement with a very long string (more than 252 characters) the machine will crash, so you must avoid doing so.
3. Using graphics statements in an ON ERROR routine may give anomalous results. For example:

```
10 ON ERROR MODE 0 : REPORT : END
20 MODE 1
30 REPEAT
40 DRAW RND(256)-1,RND(64)-1
50 UNTIL FALSE
```

The above program can be exited only by pressing ESC. The intention is that this will cause the display to clear and themessage "Escape" to be displayed. In practice, the message actually displayed will be "Sorry, not implemented" since, the MODE statement still affects REPORT, ERR and ERL.

Special Cambridge Z88 information

The following system information is available:

`PRINT ~PTR# -1`	The number of channels left for use and the ROM release number
`PRINT EXT# -1`	Prints the estimated free memory in bytes
`PRINT EOF# -1`	Returns 0 for an unexpanded Cambridge Z88 and -1 for an expanded one. See Expanded/Unexpanded.