ABS	A shift B		**NOT**	N shift O	
AND	A shift N		**ON**		
ASC	A shift S		**OPEN**	O shift P	
ATN	A shift T		**OR**		
CHR$	C shift H		**PEEK**	P shift E	
CLOSE	CL shift O		**POKE**	P shift O	
CLR	C shift L		**POS**		
CMD	C shift M		**PRINT**	shift /	
CONT	C shift O		**PRINT#**	P shift R	
COS			**READ**	R shift E	
DATA	D shift A		**REM**		
DEF	D shift E		**RESTORE**	RE shift S	
DIM	D shift I		**RETURN**	RE shift T	
END	E shift N		**RIGHT$**	R shift I	
EXP	E shift X		**RND**	R shift N	
FN			**RUN**	R shift U	
FOR	F shift O		**SAVE**	S shift A	
FRE	F shift R		**SGN**	S shift G	
GET	G shift E		**SIN**	S shift I	
GET#			**SPC(**	S shift P	
GOSUB	GO shift S		**SQR**	S shift Q	
GO			**STATUS**	ST	
GOTO	G shift O		**STEP**	ST shift E	
IF			**STOP**	S shift T	
INPUT			**STR$**	ST shift R	
INPUT#	I shift N		**SYS**	S shift Y	
INT			**TAB**	T shift A	
LEFT$	LE shift F		**TAN**		
LEN			**THEN**	T shift H	
LET	L shift E		**TIME**	TI	
LIST	L shift I		**TIME$**	TI shift 4	
LOAD	L shift O		**TO**		
LOG			**USR**	U shift S	
MID$	M shift I		**VAL**	V shift A	
NEW			**VERIFY**	V shift E	
NEXT	N shift E		**WAIT**	W shift A	

ABS

ABS(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	A shift B	A|	DC58	B6	182

ABS returns the absolute value of a number, which is its value without a negative sign. For example, ABS(7) and ABS(-7) both return a 7.

Example:

```
FOR T = 0 TO 8: PRINT ABS(4-T): NEXT
```

This is a technique I use to create symmetrical patterns where the value decreases to zero and then increases again. This can be applied to visual effects, measuring deviations, and sorting. I find it helpful in games for distance measurement without consideration to direction.

AND

<expression> AND <expression>

Type	Short	Appears	Location	Token HEX	Token DEC
OPERATOR	A shift N	A/	CFE9	AF	175

AND is used in Boolean operations to test bits and evaluate logical conditions. It returns 1 only if both operands are 1. If either or both operands are 0, the result is 0. This is useful in logical comparisons:

```
1 AND 1 = 1          0 AND 1 = 0
1 AND 0 = 0          0 AND 0 = 0
```

Example:

```
37 AND 23 = 5        0 0 1 0 0 1 0 1 (37)
                     0 0 0 1 0 1 1 1 (23)
                     0 0 0 0 0 1 0 1  (5)

IF A=1 AND B=2 THEN PRINT "BOTH ARE TRUE"
```

This line checks for multiple conditions in combination.

ASC

ASC(<string>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	A shift S	A♥	D78B	C6	198

ASC returns a number from 0 to 255, the PETSCII value of the *first* character in the string. An empty string will produce an ?ILLEGAL QUANTITY error. See the PETSCII code chart.

Example:

```
1 PRINT ASC("A")          output is 65 for A
2 PRINT ASC("APPLE")      output is 65 for A
3 A$="TEST"
4 PRINT ASC(A$)           output is 84 for T
```

ATN

ATN(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	A shift N	A/	E308	C1	193

ATN returns the arctangent (inverse tangent) of a number. This means that it takes a ratio (a number) and tells you the angle whose tangent is that ratio. The tangent of an angle in a right triangle is the ratio of the opposite side to the adjacent side (opposite/adjacent). ATN reverses this: It takes the ratio and gives you the angle (in radians).

```
ATN(1) = 0.785398163
ATN(1)*4 = 3.14159266
```

Example:

```
1 INPUT "OPPOSITE SIDE"; O
2 INPUT "ADJACENT SIDE"; A
3 PRINT "ANGLE IS" ATN(O/A)*180/π
```

This program finds one angle of a right triangle by inputting the lengths of two sides. ATN helps find angles in trigonometry when you only know side lengths. It is useful in rotation and direction calculations.

CHR$ CHR$(<number 0-255>)

Type	Short	Appears	Location	Token HEX	Token DEC
STRING	C shift H	C|	D6E6	C7	199

CHR$ returns a PETSCII character (of string type). The number is automatically treated as an integer. CHR$ is the inverse function of ASC.

```
1 PRINT CHR$(65)          output is A
2 PRINT CHR$(147)         clears the screen
3 PRINT CHR$(28)          change text color to RED
4 PRINT CHR$(14)          change to lowercase
```

Example:

```
1 PRINT CHR$(147)
2 FOR I= 65 TO 90
3 PRINT CHR$(I) SPC(4)
4 NEXT
```

This program clears the screen then PRINTs the alphabet. CHR$ is useful for special commands such as PRINTing quotation marks without confusing the BASIC interpreter.

CLOSE CLOSE(<file number 0-255>)

Type	Short	Appears	Location	Token HEX	Token DEC
COMMAND	CL shift O	CL┌	FFC3	A0	160

CLOSE is used to close currently open files or devices.

Example:

```
1 OPEN 1,4
2 PRINT#1, "TEST"
3 CLOSE 1
```

CLR CLR

Type	Short	Appears	Location	Token HEX	Token DEC
COMMAND	C shift L		C65E	9C	156

CLR clears all variables and resets the stack and data pointer. It does not erase the program in memory but removes all stored values, making it useful for restarting calculations or freeing up memory.

Example:

```
1 A=3.14
2 A$= "TEST"
3 PRINT A                output is 3.14
4 PRINT A$               output is TEST
5 CLR                    clears all variables
6 PRINT A                output is 0
7 PRINT A$               output is an empty string
```

CMD CMD<file number>[,<string>]

Type	Short	Appears	Location	Token HEX	Token DEC
I/O	C shift M		CA86	9D	157

CMD redirects output to a device, such as a printer or disk file, instead of the default screen device. The file number must be specified in a prior OPEN statement. Once CMD is used, all subsequent PRINT statements will send their output to the specified device.

Example:

```
1 OPEN 1,4
2 CMD 1
3 PRINT "TEST"
4 PRINT #1
5 CLOSE 1
```

CONT CONT

Type	Short	Appears	Location	Token HEX	Token DEC
COMMAND	C shift O	C┌	C857	9A	154

CONT resumes the execution of a BASIC program from the exact point where it was stopped (by the RUN/STOP key, STOP command, or. While the program is stopped, you can inspect or modify variables before resuming. CONT does not work if the program was edited, if an error caused the stop, or if an error occurred before attempting to continue.

Example:

```
1 A=3: PRINT A: STOP: PRINT A
RUN             the program displays 3 and breaks
A=5             now, change the value of A
CONT            the program continues with the new value
```

This program sets a variable and stops the program. You can modify the variable and resume the program with CONT.

COS COS(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	---		E261	BE	190

COS returns the cosine of a given angle (in radians, not degrees). Used in trigonometry for calculating angles and side lengths. This is useful in graphics (circular motions) and sound (waveform calculations).

```
COS(0) = 1          cosine of 0 radians
COS(π/2) = 0        cosine 90 degrees (π/2 radians)
COS(π) = -1         cosine 180 degrees (π radians)
```

Example:

```
1 INPUT "DISTANCE TRAVELED"; D
2 INPUT "ANGLE IN RADIANS"; A
3 PRINT "HORIZONTAL DISTANCE" D * COS(A)
```

DATA

DATA<value>[,<value> . . .]

Type	Short	Appears	Location	Token HEX	Token DEC
COMMAND	D shift A	D♣	C858	83	131

DATA stores a list of values that can be read later by the program as a continuous list using the READ statement. Any type of information can be stored here, separated by commas. If you want to include special characters such as a comma, space, or colon as data, they must be enclosed in quotation marks. The DATA statement doesn't need to be executed during the running of the program, so it may be anywhere.

Example:

```
1 READ A, B%, C$
2 PRINT A, B%, C$
3 END
4 DATA 20, 50, "TEST"
```

DEF

DEFFN<name>(<variable>)=<expression>

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	D shift E	D—	D3B3	96	150

DEF defines a custom function that can be used later in a program. The function name is the letters FN followed by any variable name (for example, FNA). The function can take one numeric variable as input and call be called multiple times throughout the program.

The DEF command is typically used to simplify calculations or create reusable formulas. The FN keyword is required to reference the function when it's called in expressions. See FN.

Example:

```
1 DEF FN A(X) = X * X
2 PRINT FN A(5)          output is 25
3 PRINT FN A(7)          output is 49
```

DIM

DIM<variable>(<d1>[,d2…])[,<var>(<d1>[d2…])]

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	D shift A	D♣	D081	86	134

DIM is used to declare arrays in BASIC. It reserves space for a list of values stored under a single variable name (numeric, string, or integer), indexed by a number. Arrays can be one-dimensional or multi-dimensional. The highest index is the number specified in the DIM statement, but since BASIC arrays start at index 0, an array declared as DIM A(5) will have six elements: A(0) through A(5).

Arrays must be declared with DIM before being used. If an array is used without a DIM statement, BASIC assumes a default size of 10.

Example:

```
DIM A(65)          1 dimensional numeric array
DIM B(7), B$(11)   two 1 dimensional arrays
DIM C%(5,3)        2 dimensional integer array
DIM D(5,3,20)      3 dimensional numeric array
```

END

END

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	E shift N	E⁄	C831	80	128

END marks the termination of a BASIC program. When encountered, it stops execution immediately, even if there are more lines of code after it. It does not clear variables, so they remain available after execution. END is not required, as execution will stop automatically when the last line runs.

Example:

```
1 PRINT "THIS IS LINE 1"
2 END
3 PRINT "THIS LINE WILL NEVER EXECUTE"
```

EXP

EXP(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	E shift X	E↑	DFED	BD	189

EXP returns the mathematical constant e (approximately 2.71828183) raised to the power of a given number (X). A value of X greater than 88.0296919 will cause an ?OVERFLOW ERROR. EXP is useful in scientific calculations, growth models, and logarithmic functions.

$$EXP(X) = e^X$$

Example:

```
1 PRINT EXP(0)          output is 1
2 PRINT EXP(1)          output is 2.718
3 PRINT EXP(2)          output is 7.389
```

FN

FN<name>(<expression>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	---		D3F4	A5	165

FN is used to call a user-defined function that was previously defined using DEF. The function name consists of FN followed by a letter and optionally one more letter or digit. When calling the function, the argument is placed in parentheses after the function name. See DEF.

Example:

```
1 REM RANDOM NUMBER
2 DEF FND(X) = INT(RND(1) * X) + 1
3 PRINT "6 SIDED DICE ROLL:" FN D(6)
4 PRINT "20 SIDED DICE ROLL:" FN D(20)

5 REM CELCIUS TO FAHRENHEIT
6 DEF FNT(C) = C * 9 / 5 + 32
7 PRINT FNT(0)          output is 32
8 PRINT FNT(100)        output is 212
```

FOR

FOR<variable>=<start>TO<end>[STEP<number>]

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	F shift O	F┌	C742	B1	129

FOR is used to create loops that repeat a block of code a specific number of times. It works with a counter variable that starts at a given value and increases (or decreases) by a set step each time NEXT is encountered, until it reaches a limit. If STEP is not specified, the loop increases by 1 each time.

Example:

```
1 FOR I = 1 TO 10: PRINT I: NEXT
```

FOR loops can be "nested" inside one another, which means you can have a FOR loop inside another FOR loop. This allows you to create more complex iterations over multiple dimensions or ranges. Each nested loop controls its own counter variable and has its own set of start, stop, and step values. The inner loop will complete its entire cycle before the outer loop moves to the next iteration. The VIC allows up to 10 nested loops.

Example:

```
1 FOR A = 1 TO 3
2 FOR B = 1 TO 5
3 PRINT "A="; I, "B="; T
4 NEXT: NEXT
```

FRE

FRE(<dummy number/string>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	F shift R	F_	D37D	B8	184

FRE returns the amount of free memory (RAM) available for a program and variables.

Example:

```
PRINT FRE (0)          output is number of unused bytes
```

GET

GET<variable>

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	G shift E	G⁻	FFE4	A1	161

GET retrieves the next character typed and stores it in a variable (numeric, integer, or string). It allows a program to capture a single character from the keyboard (or keyboard buffer) without waiting for the user to press the Return key.

GET uses real-time key detection; it does not pause for input. If no key is pressed when the statement executes, the variable remains its default (0 for numeric variables and an empty string for string variables. If the input is incompatible with the expected variable type—for example, if a letter is pressed when a numeric variable is used—an error will occur.

Example:

```
1 GET A$
2 IF A$= "" THEN PRINT "NO KEY PRESSED"
3 IF A$<> "" THEN PRINT "YOU PRESSED:"; A$
4 GOTO 1
```

GET#

GET<file number>,<variable>[,<variable>…]

Type	Short	Appears	Location	Token HEX	Token DEC
I/O	---		---	---	---

GET retrieves a single character from a device, such as a file or a communication channel, without waiting for a full line of input. It is similar to GET, but it works with files and other input sources instead of the keyboard. The device number is assigned when the file was opened with OPEN. The read character will be stored as a variable.

Example:

```
1 OPEN 1,8,2,"TESTFILE,S,R"
2 GET#1, A$, B
3 IF A$<>"" THEN PRINT A$; : GOTO 2
4 CLOSE 1
```

GO

GO TO<line number>

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	---		C8A0	B6	203

GO itself does not perform any action. When used with TO, it functions as the equivalent of GOTO. GO alone results in a syntax error. GO TO uses two tokens instead of one, taking up more memory and processing slightly slower. See GOTO.

Example:

```
1 PRINT "TEST": GO TO 1
```

GOSUB

GOSUB<line number>

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	GO shift S	GO♥	C883	8D	141

GOSUB goes to a subroutine in a BASIC program. When GOSUB is executed, the program jumps to the specified line number and continues running from there. When a RETURN statement is encountered, the program jumps back to the line immediately after the GOSUB command and continues execution from that point.

This allows code to be reused without duplication. Multiple subroutines can be called consecutively, but they follow a “Last In, First Out” (LIFO) order—meaning the last subroutine called will be the first to return. If too many subroutines are nested at once, the system runs out of memory.

Example:

```
1 PRINT "START OF PROGRAM"
2 GOSUB 10
3 PRINT "CONTINUE PROGRAM"
4 GOSUB 10
5 PRINT "END OF PROGRAM"
6 END
10 PRINT "SUBROUTINE"
11 RETURN
```

GOTO

GOTO<line number>

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	G shift O	G⁻	C8A0	89	137

GOTO causes the program to jump directly to a line in the program, skipping any lines in between. This is useful for controlling the flow of execution, such as jumping to a different part of the program or looping back to a previous section. It's possible to create loops with GOTO that never end, requiring the user to interrupt it with the RUN STOP key.

Example:

```
1 PRINT "HELLO WORLD"
2 GOTO 1
```

IF

IF<expression>THEN<line number or statement>

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	---		C928	8B	139

IF is used for conditional branching in a program. It evaluates an expression, and if the expression is true, the following statement is executed. This can be a command, an expression, or a line redirection. If false, everything else on that line is skipped (not executed), and the program jumps directly to the next program line.

IF can be paired with GOTO instead of THEN. If THEN is immediately followed by a number, BASIC interprets it as GOTO that line number. IF statements support comparisons like <, >, =, <=, >=, <> and logical operators like AND, OR, and NOT for more complex conditions.

Example:

```
1 INPUT "ENTER A NUMBER"; N
2 IF N=0 OR N>10 THEN 1
3 IF N=5 THEN PRINT "FIVE": END
4 IF N=6 OR N=7 THEN PRINT "SIX OR 7": GOTO 1
5 IF N=>8 THEN PRINT "EIGHT OR MORE"
6 IF N<4 THEN PRINT "LESS THAN FOUR"
```

INPUT

INPUT[“<prompt>”;]<variable>[,<variable>…]

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	---		FFCF	85	133

INPUT defines a variable from user input. It displays a question mark then pauses execution and waits for the user to type a response, which is then stored in a specified variable. If multiple variables are listed, the user must enter values separated by commas. A prompt can be displayed by including a string followed by a semicolon or a comma before the variable name. User input must end with pressing the Return key. If no input is received, the contents of the variables remain unchanged. The only way to end a program during an INPUT statement is to hold down the RUN/STOP key and hit RESTORE.

Example:

```
1 INPUT A
2 INPUT B, C, D
3 INPUT "ENTER A NUMBER"; E
4 PRINT A, B, C, D, E
```

INPUT#

INPUT#<file number>,<variable>[,<variable>…]

Type	Short	Appears	Location	Token HEX	Token DEC
I/O	I shift N	I⁄	CBA5	84	132

INPUT# defines a variable from the next piece of data from a previously opened file stored on peripheral device media such as disk or tape. It reads complete data consisting of maximum 80 characters into variables and not only single characters as the GET# command. If the file has no more data, the program will automatically continue to the next line of code.

Example:

```
1 OPEN 1,8,2,"TESTFILE"
2 INPUT#1, A$
3 PRINT A$;
4 CLOSE 1
```

INT

INT(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	---		DCCC	B5	181

INT returns the integer part of a given number. It essentially removes any digits after the decimal point of a positive number. If the number is negative, INT rounds it down to the next lower integer (more negative).

Example:

```
1 PRINT INT(3.14)          output is 3
2 PRINT INT(-3.14)         output is negative 4
3 PRINT INT(3.14 + 0.5)    output is 4
```

INT always round downward. Line 3 demonstrates a technique for rounding a number up or down by adding 0.5.

LEFT$

LEFT$(<string>,<integer number 0-255>)

Type	Short	Appears	Location	Token HEX	Token DEC
STRING	LE shift F	LE_	D700	C8	200

LEFT$ returns a specified number (from 0 to 255) of characters from the left (beginning) of a string. If the specified length is 0, an empty string ("") is returned.

If the specified length is greater than the string length, the entire string is returned. If length is negative or greater than 255, an error occurs.

Example:

```
1 A$="TEST"
2 PRINT LEFT$(A$, 1)       output is T
3 PRINT LEFT$(A$, 3)       output is TES
4 PRINT LEFT$("ABC", 2)    output is AB
5 PRINT LEFT$("ABC", 5)    output is ABC
```

LEN LEN(<string>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	---		D77C	C3	195

LEN returns the number of characters in a string, including spaces and non-printing characters. The result is an integer representing the length of the string. An empty string returns zero.

Example:

```
1 A$="TEST"
2 PRINT LEN(A$)              output is 4
3 PRINT LEN("A B C")         output is 5
4 PRINT LEN("")              output is 0
5 PRINT SPC(11-LEN(A$)/2)    centers a string on screen
```

Line 5 demonstrates a technique for centering text on screen. The center of a 22 column screen is 11. By subtracting half the length of a string from 11, one can determine the number of spaces to roughly center different length strings on a line.

LET [LET]=<expression>

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	L shift E	L‾	C9A5	88	136

LET assigns a value to a variable. It is optional, meaning you can write assignments with or without it. Since LET is not required, most programmers omit it. Remember, variables are shortened to two characters. So, SCORE would automatically be the same as SC.

Example:

```
1 LET A = 17.6
2 LET A$ = "TEST"
3 LET B% = A + 15
4 LET SCORE = 10 : REM SC = 10
```

LIST

LIST[[line number]-[line number]]>

Type	Short	Appears	Location	Token HEX	Token DEC
COMMAND	L shift I	L┐	C69C	9B	155

LIST displays the BASIC program currently in memory. By default, this listing is shown on the screen. However, it can also be redirected to other output devices, such as a printer, by using specific commands or redirection settings. LIST is also used to display disk directories. During the listing of a long program, the user can cancel the listing immediately by pressing <RUN/STOP>. To slow down the scrolling speed, the user can hold down the <CTRL> key.

Example:

```
LIST                displays the entire program
LIST 10             displays line 10 only
LIST 10-            displays from line 10 on
LIST -10            displays from start to line 10
LIST 10-20          displays line 10 through line 20
LIST 0              displays the entire program
```

LOAD

LOAD[“<file name>”[,<device>[,<number>…]]]

Type	Short	Appears	Location	Token HEX	Token DEC
COMMAND	L shift O	L┌	FFD5	93	147

LOAD transfers a previously SAVED program, file, or data from an external device (such as a disk or tape) into the computer’s memory. LOAD works with file-name pattern matching and special characters explained in the device section on page XXX.

Example:

```
LOAD                    next program on tape
LOAD "TEST"             tape program by file name
LOAD A$                 tape program by string
LOAD "TEST" ,8          disk program
LOAD "TEST" ,8 ,1       disk in same saved memory
```

LOG

LOG(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	---		D9EA	BC	188

LOG returns the natural logarithm of a positive number greater than zero. The result of LOG(x) is the power to which e must be raised to get the value x.

e is approximately equal to 2.71828. The natural logarithm is commonly used in mathematics, physics, and engineering for dealing with exponential growth or decay.

```
1 PRINT LOG(1)                output is 0
2 PRINT LOG(2.71828184) = 1
3 PRINT LOG(100)/LOG(10) = 2
```

MID$

MID$(<string>,<start>[,<length>])

Type	Short	Appears	Location	Token HEX	Token DEC
STRING	M shift I	M-,	D737	CA	202

MID$ returns a substring from a string, starting at a given position and (optionally) with a specific length. Both the start position and length must be between 1 and 255, and they are interpreted as integer values. If the start position exceeds the length of the string, or if the specified length is negative, the result will be an empty string.

Example:

```
1 A$="ABCDEFGHIJ"
2 PRINT MID$(A$, 2, 2)      output is BC
3 PRINT MID$(A$, 3, 5)      output is CDEFG
4 PRINT MID$(A$, 5, 12)     output is EFGHIJ (end)
```

NEW NEW

Type	Short	Appears	Location	Token HEX	Token DEC
COMMAND	---		C642	A2	162

NEW clears the current program from memory. It removes all program lines and resets any variables or data stored during the program's execution without confirmation. However, it does not reset or affect system settings. There is no undo function for NEW in BASIC.

Example:

```
1 PRINT "TEST"
NEW             can be used in direct mode or in program
LIST            there is no longer a program in memory
```

NEXT NEXT[<counter variable>][,<variable>]. . .

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	N shift E	N⁻	CD1E	82	130

NEXT advances the counter variable in a FOR...NEXT loop. The counter variable is incremented (or decremented) by the specified STEP value.

NEXT is used with the FOR statement to mark the end of the loop's code block and returns to the corresponding FOR statement (If a variable is not included, this is the most recent active loop's counter). If the counter limit is reached, the loop ends, and the program continues with the next statement or line. Multiple loop counters can be terminated by a single NEXT statement if the variable names are listed in order, separated by commas, from the innermost loop to the outermost loop. The VIC allows up to 10 nested loops.

Example:

```
1 FOR A = 1 TO 10
2 PRINT A: NEXT
3 FOR B = 1 TO 3: FOR C = 1 TO 5
4 PRINT B, C: NEXT C, B
```

NOT

<expression> NOT <expression>

Type	Short	Appears	Location	Token HEX	Token DEC
OPERATOR	N shift O	NOT	CED4	A8	168

NOT performs a bitwise complement on integer numbers, flipping every bit and interpreting the result as a signed integer. For example, the number 5 in binary (16-bit signed integer format) returns negative 6:

```
0000 0000 0000 0101        5 in binary, 16-bit signed integer
1111 1111 1111 1010        NOT 5 in binary, negative 6
```

When NOT is applied to a boolean expression (like a comparison), it treats 0 as false (0) and any non-zero value as true (-1)

Example:

```
1 PRINT NOT 0          output is -1
2 PRINT NOT 5          output is -6
3 PRINT NOT -1         output is 0
4 PRINT NOT (5=5)      output is 0 (false)
```

ON

ON<expression>GOSUB/GOTO<line>[,<line>]. . .

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	---		C94B	91	145

ON is used for conditional branching based on a numeric expression (evaluated as an integer). It selects one of several possible GOTO or GOSUB destinations. If the result is less than 1 or greater than the number of line numbers listed, the program continues without jumping.

Example:

```
1 INPUT A
2 ON A GOTO 3, 4, 5: END
3 PRINT "ONE": END
4 PRINT "TWO": END
5 PRINT "THREE": END
```

OPEN

OPEN<file no.>[,<device>[,<channel>[,<name>]]

Type	Short	Appears	Location	Token HEX	Token DEC
I/O	O shift P	O┐	FFC0	9F	159

OPEN establishes a communication channel between the computer and an external device, such as a disk drive, printer, or user port. This channel allows data to be transferred between the program and the device. See I/O section for device numbers and more information.

Example:

```
OPEN 1,8,15,"<command>"
```

Opens channel 1 to device 8 (disk drive), with 15 as the command channel to send the specified disk command.

OR

<expression> OR <expression>

Type	Short	Appears	Location	Token HEX	Token DEC
OPERATOR	---		CFE6	B0	176

OR is used in Boolean operations to test bits and evaluate logical conditions. It returns 1 if either or both operands are 1. If returns 0 only if both operands are 0. This is useful in logical comparisons:

```
1 OR 1 = 1          0 OR 1 = 1
1 OR 0 = 1          0 OR 0 = 0
```

Example:

```
5 AND 3 = 7         0 0 0 0 0 1 0 1 (5)
                    0 0 0 0 0 0 1 1 (3)
                    0 0 0 0 0 1 1 1  (7)

IF A=1 OR B=2 THEN PRINT "BOTH ARE TRUE"
```

Checks for multiple conditions, returning true if at least one condition is met.

PEEK PEEK(<memory address>)

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	P shift E	P¯	D80D	C2	194

PEEK returns the byte value stored at the specified memory address. The address must be a number between 0 and 65535. The result is an integer between 0 and 255, representing the data at that location. This function is often used to read system settings, hardware registers, or memory-mapped data.

Example:

```
1 PRINT PEEK(7680)
```
output is the value in memory

π π

Type	Short	Appears	Location	Token HEX	Token DEC
CONSTANT	---		CEA8	FF	255

π is a built-in constant representing an approximation of pi.

Example:

```
1 PRINT π
```
output is 3.14159265

POKE POKE<memory address>,<value>

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	P shift O	P⌐	D824	97	151

POKE stores a byte value at a specified memory address. The address must be a number between 0 and 65535, representing the memory location. The value must be between 0 and 255.

Example:

```
1 POKE 7680, 42
```
puts the value in memory

POS

POS(<dummy number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	---		D39E	B9	185

POS returns the current position of the output cursor in the current text window (or device). Specifically, it tells you the column number of the cursor on the current line of output. The position is returned as an integer, which corresponds to the column where the next output would appear.

Example:

```
1 PRINT "TEST";
2 PRINT POS(X)              output is TEST 4
```

PRINT

PRINT[<expression>] [[;|,]<expression>...]

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	shift /	?	FFD2	99	153

PRINT displays text, numbers, or expressions on the screen. It is used to output information during program execution. The content can be separated by commas or semicolons to control formatting. This means the cursor automatically moves to the next line after the output.

Expressions can be text strings, numbers, variables, or special characters (like cursor controls). Semicolon (;) prints the next item immediately after the previous one without space. Comma (,) aligns the next item at the next tab stop (usually every 10 characters on the VIC-20). If no separator follows the expression, the cursor moves to the next line.

Example:

```
1 PRINT "TEST"       outputs TEST and moves to next line
2 PRINT "A";"B"      outputs AB
3 PRINT A            outputs 0, the value of A
4 PRINT A,B          outputs 0      0 with tab spacing
```

PRINT#

PRINT#<file number>,<variable>[,<variable>…]

Type	Short	Appears	Location	Token HEX	Token DEC
I/O	P shift R	P_	CA80	98	152

PRINT# sends output to an open communication channel, such as a disk drive, printer, or other external device, instead of the screen. The channel must have been previously opened by the OPEN command. Expressions can include text, numbers, or variables.

Example:

```
1 OPEN 1,8,2,"TESTFILE"
2 PRINT#1, A$
3 CLOSE 1
```

READ

READ<variable> [,<variable>]...

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	R shift E	R	CC06	87	135

READ retrieves values from a list of data items provided by DATA statements in the program. READ assigns the next value from the DATA list to one or more variables, moving sequentially through the list each time it is called. The location of the DATA statement in the program is irrelevant, as it does not affect the program's flow.

Each variable receives one item from the DATA list in the order they appear. The DATA statement holds literal values separated by commas. The program must include enough DATA items for all READ statements, or an OUT OF DATA error will occur. RESTORE can reset the data pointer, allowing the same DATA items to be read again.

Example:

```
1 READ A, B$
2 PRINT A, B$        outputs 17   TEST
3 END
4 DATA 17, "TEST"
```

REM REM[<any text>]

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	---		C93B	8F	143

REM insert comments in a BASIC program. Anything following REM on the same line is ignored during execution, serving only as a note for the programmer. Though primarily for comments, some programmers use REM to create placeholders or visual markers in the code for easier navigation with LIST.

When using LIST, graphic characters that are not enclosed in quotation marks might be interpreted as tokens. This can potentially change the computer's behavior during LIST.

Example:

```
1 REM THIS IS AN EXAMPLE PROGRAM
2 PRINT SC : REM SC IS A VARIABLE FOR SCORE
```

RESTORE RESTORE

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	RE shift S	RE♥	C81D	8C	140

RESTORE resets the position of the DATA pointer. Once RESTORE is called, the next READ command will retrieve data from the reset position. It can be used multiple times within the program to revisit DATA values, making it useful in loops or conditional structures.

Example:

```
1 READ A, B$
2 PRINT A, B$        outputs 17   TEST
3 RESTORE
4 GOTO 1
5 DATA 17, "TEST"
```

RETURN RETURN

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	RE shift T	RE|	C8D2	8E	142

RETURN transfers control back to the statement immediately following the GOSUB that called the subroutine. It marks the end of a subroutine, allowing the program to resume execution at the point where the subroutine was invoked.

Example:

```
1 PRINT "START OF PROGRAM"
2 GOSUB 10: PRINT "CONTINUE PROGRAM"
3 GOSUB 10: PRINT "END OF PROGRAM"
4 END
10 PRINT "SUBROUTINE"
11 RETURN
```

RIGHT$ RIGHT$(<string>,<integer number 0-255>)

Type	Short	Appears	Location	Token HEX	Token DEC
STRING	R shift I	R-,	D72C	C9	201

RIGHT$ returns a specified number (from 0 to 255) of characters from the rightmost portion of a string, extracting a specified number of characters from the end of the string. If the specified length is 0, an empty string ("") is returned.

If the specified length is greater than the string length, the entire string is returned. If length is negative or greater than 255, an error occurs.

Example:

```
1 A$="TEST"
2 PRINT RIGHT$(A$, 1)      output is T
3 PRINT RIGHT$(A$, 3)      output is EST
4 PRINT RIGHT$("ABC", 2)   output is BC
5 PRINT RIGHT$("ABC", 5)   output is ABC
```

RND

RND(<dummy number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	R shift N	R/	E094	BB	187

RND generates a pseudo-random number between 0 and 1 (exclusive) using VIA 1 timer A and timer B. The result is always greater than or equal to 0 and less than 1.

In RND(X), the X parameter can affect randomness in certain cases, positive values essentially act the same, making X a dummy for those inputs. 0 produced a less varied sequence of random numbers because the result depends on the system timer, which may follow predictable patterns. RND(π) has been suggested as an ideal value for achieving both fast execution and an even distribution of random numbers.

Example:

```
1 PRINT RND(1)              output is between 0 and 1
2 PRINT INT(RND(1)*7)       output is integer 0 to 6
3 PRINT INT(RND(1)*7)+1     output is integer 1 to 7
```

RUN

RUN[<line number>]

Type	Short	Appears	Location	Token HEX	Token DEC
COMMAND	R shift U	R ⌐	C871	8A	138

RUN starts the execution of a BASIC program from the lowest-numbered line in memory. It clears any previous variables and resets the DATA pointer to the first DATA statement unless the program is started with the optional line number after RUN format. Without a line number, RUN begins execution from the first line.

Pressing RUN/STOP during execution halts the program.

Example:

```
RUN          begins execution from the first line
RUN 10       begins execution from line 10
```

SAVE

SAVE["<file name>"[,<device>[,<number>…]]]

Type	Short	Appears	Location	Token HEX	Token DEC
COMMAND	S shift A	S♠	FFD8	94	148

SAVE is used to save a program or data to a storage device, such as a cassette tape or disk. It allows users to preserve their work for future use or transfer. The SAVE command specifies the device number, the file name, and the type of data being saved (typically a program or data file). The SAVE command is typically used to preserve programs or data files for later loading with the LOAD command.

Example:

```
SAVE                    saves to tape without name
SAVE "TEST"             save to tape with file name
SAVE A$                 tape program by string
SAVE "TEST" ,8          disk program
SAVE "TEST" ,8 ,1       disk in same saved memory
```

SGN

SGN(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	S shift G	S|	DC39	B4	180

SGN returns the sign of a numeric expression. It indicates whether the number is positive, negative, or zero. The SGN function is useful for testing the direction of numbers, especially in mathematical operations or decision-making within a program.

Example:

```
1 PRINT SGN(1)          output is 1
2 PRINT SGN(36879)      output is 1
3 PRINT SGN(0)          output is 0
4 PRINT SGN(-7)         output is -1
```

SIN

SIN(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	S shift I	S¬	E268	BF	191

SIN returns the trigonometric sine of a given angle in radians. The result will always be between -1 and 1. To convert degrees to radians (since SIN expects radians), multiply the angle by PI/180:

```
SIN(0) = 0          sine of 0 radians
SIN(π/2) = 1        sine 90 degrees (π/2 radians)
SIN(π) = 0          sine 180 degrees (π radians)
SIN(-π/2) = -1      sine -90 degrees (-π/2 radians)
```

Example:

```
1 INPUT "DISTANCE TRAVELED"; D
2 INPUT "ANGLE IN RADIANS"; A
3 PRINT "VERTICAL DISTANCE" D * SIN(A)
```

SPC(

SPC(<integer number 0-255>)

Type	Short	Appears	Location	Token HEX	Token DEC
STRING	S shift P	S┐	CAF8	A6	166

SPC(inserts a specified number of spaces (between 0 and 255) into printed output. The function is used within PRINT statements to control text formatting. A semicolon is not necessary before or after a SPC statement. If the specified length is negative or greater than 255, an error occurs. If SPC() is the last element in the line, the next PRINT statement will resume printing immediately after the inserted spaces, without moving to a new line.

Example:

```
1 PRINT SPC(5) "A"          output is      A
2 PRINT "A" SPC(5) "B"      output is A    B
```

SQR

SQR(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	S shift Q	S●	DF71	BA	186

SQR calculates the square root of a positive numeric expression. The result is the non-negative number that, when multiplied by itself, equals the input value. The function only works with non-negative numbers. It will not handle imaginary numbers or negative square roots.

Example:

```
1 PRINT SQR(25)          output is 5
2 PRINT SQR(2)           output is 1.41421356
3 PRINT SQR(0)           output is 0
```

STATUS

STATUS

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	ST	ST	C795	---	---

STATUS returns a series of numbers that reflect the status of connected I/O devices such as disk drives, printers, or other peripherals. When used with a specific device number, it shows whether the device is ready for use or if there's an error.

Example:

```
1 PRINT STATUS
2 PRINT ST AND 64          displays a distinct bit
```

BIT	VALUE	TAPE	SERIAL	RS-232
0	0	OK	OK	OK
0	1		Write time out	Parity error
1	2		Read time out	Framing error
2	4	Data block too		Rec buffer
3	8	Data block too		
4	16	Verify read error		CTS signal
5	32	Checksum error		
6	64	End of file (EOI)	End of file (EOI)	RTS signal
7	128		Device not present	Break detected

STEP . . .[STEP<number>]

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	ST shift E	ST	C82F	A9	169

STEP is used in conjunction with the FOR loop in BASIC programming to control the increment or decrement of the loop counter. By default, the counter increases by 1 each time the loop runs, but STEP allows you to change this value to any number.

Example:

```
1 FOR I = 1 TO 10 STEP 2          increment by 2
2 PRINT I: NEXT
3 FOR I = 1 TO 10 STEP .02        increment by 0.02
4 PRINT I: NEXT
5 FOR I = 10 TO 1 STEP -1         decrement by 1
5 PRINT I: NEXT
```

STOP STOP

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	S shift T	S	D465	90	144

STOP halts the execution of a program and triggers a "BREAK" message, displaying the line number where the stop occurred. This is useful for debugging.

Example:

```
1 PRINT "START"
2 STOP
3 PRINT "END"
RUN
START

BREAK IN 2
READY.
```

STR$ STR$(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
STRING	ST shift R	ST_	E127	C4	196

STR$ converts a numeric value into a string. The result is a string that can be printed, concatenated, or manipulated just like any other string.

Example:

```
1 X= 17
2 A$= STR$(X)
3 PRINT A$                    output is 17
```

SYS SYS<memory address>

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	S shift Y	S	E127	9E	158

SYS calls a machine language routine by jumping directly to a specific memory address. This allows BASIC programs to execute custom machine language programs or built-in system routines stored in ROM memory. It is essential to know the correct memory addresses and how the machine language routines work before using SYS, as incorrect usage can crash the system.

To pass parameters to machine language routines, you typically store values in specific memory locations or CPU registers before issuing the SYS command: 780 (accumulator), 781 (X register), 782 (Y register), 783 (status register/flags).

The number 96 for RTS (Return from Subroutine) ends the machine code routine and returns control to BASIC. SYS expects the routine to end with RTS, or the program may crash or behave unpredictably.

Example:

```
SYS 828        calls a user-defined routine at location 828
SYS 64802      calls the ROM routine to reset the computer
```

TAB(

TAB(<integer number 0-255>)

Type	Short	Appears	Location	Token HEX	Token DEC
STRING	T shift A	T♣	CAFB	A3	163

TAB(moves the print position to a specified column on the screen. It is used with the PRINT statement to control text layout. If the current print position is already beyond the target, the cursor moves to the next line before positioning.

Unlike SPC(x), which inserts a fixed number of spaces, TAB(x) directly moves the cursor to the specified column: (the *X+1* position) on the current line. If X is less than the current cursor position, TAB() is ignored.

Example:

```
1 PRINT TAB(5) "A"          output is     A
2 PRINT "A" TAB(5) "B"      output is A   B
```

TAN

TAN(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	---		E2B1	C0	192

TAN returns the tangent of an angle in radians. In a right triangle, the tangent of an angle is the ratio of the length of the opposite side to the length of the adjacent side. The TAN function can be used in BASIC to compute this value for any given angle.

```
TAN(0) = 0          a flat line, 0 radians
TAN (π/4) = 1       45 degree angle, equal sides, ratio: 1
```

Example:

```
1 INPUT "HORIZONTAL DISTANCE"; D
2 INPUT "ANGLE IN RADIANS"; A
3 PRINT "VERTICAL DISTANCE" D * TAN(A)
```

THEN

IF<expression>THEN<line number or statement>

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	T shift H	T H	---	A7	167

THEN is part of the IF-THEN statement structure, which is used to separate the condition from the action that should be taken if that condition is true. It can be followed by one or more commands that should be executed if the condition is true.

If the condition is false, the command(s) after THEN are skipped, and the program continues to the next line.

Example:

```
1 INPUT "ENTER A NUMBER"; N
2 IF N=>5 THEN PRINT "MORE THAN 4": END
3 IF N<5 THEN PRINT "LESS THAN 5": GOTO 1
```

TIME

TIME

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	TI	T I	---	---	---

TIME (often abbreviated from TI) is a special function used to measure the time interval since the last reset or power-on of the computer. It is a system variable that stores the time in 1/60th second intervals.

TIME is read-only and cannot be set. Variable TIME$ can however be set and the TIME variable will follow. For example, TI$="000000" will set TI to 0. The TI value resets to zero when the VIC-20 is powered off or reset.

Example:

```
1 PRINT "TIME ELAPSED (1/60 SECS): "; TI
2 TI$= "000000"              reset with TI$
3 FOR T=1 TO 999: NEXT       brief delay
4 PRINT "NOW: "; TI          the time it takes to execute
```

TIME$ TIME$

Type	Short	Appears	Location	Token HEX	Token DEC
STRING	TI$	T I $	---	---	---

TIME$ (often abbreviated from TI$) is a special system variable that holds the current time of day as a string in the format HH:MM:SS, where HH is hours, MM is minutes, and SS is seconds.

The time is based on the VIC-20's continuous internal clock. When the computer is turned on, the clock starts at "000000" (midnight) and begins counting from there. Resetting TIME$ resets TIME too.

You can set the time by assigning a string in the correct format based on a 24 hour clock: for example, "153045" would represent 3:30:45 PM. After 23:59:59, the clock wraps around to "000000" automatically.

Example:

```
1 PRINT TI$                displays time elapsed
2 TI$= "000000"            resets the time
3 FOR T=1 TO 999: NEXT     brief delay
4 PRINT "NOW: "; TI$       the time it takes to execute
```

TO TO<number>

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	---	---	---	A4	164

TO serves two primary purposes: As part of a FOR-NEXT loop, and as part of the "GO TO" construct (the same as GOTO). It defines the limit of a loop or the line number of a GOTO statement. TO cannot be used as a variable name or part of one.

Example:

```
1 FOR A = 1 TO 10          as part of FOR TO NEXT loop
2 PRINT A: NEXT
3 GO TO 1                  as part of GO TO (GOTO)
```

USR

USR(<number>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	U shift S	U♥	---	B7	183

USR calls user-defined machine language routines whose starting address is stored in memory locations 1 and 2 (low byte and high byte of the address, respectively) and returns a value when the subroutine finishes. The subroutine receives the expression as an argument in the floating point accumulator. The machine code routine can access this argument and perform operations based on it. The result of the routine must be placed back into the floating point accumulator before returning, so BASIC can access the final value.

Example:

```
USR(8)          calls a user-defined routine with a value of 8
```

VAL

VAL(<string>)

Type	Short	Appears	Location	Token HEX	Token DEC
NUMERIC	V shift A	V♠	D7AD	C5	197

VAL converts a string of numeric characters into a numeric value. It scans the string from left to right, extracting any valid numeric characters it encounters. Any non-numeric characters returns 0.

The first dot (.) is interpreted as decimal point and the first e or E as exponent. It stops converting as soon as it reaches a character that isn't part of a valid number (like a letter or a space).

Example:

```
1 PRINT VAL("3")            output is 3
2 PRINT VAL ("3.14")        output is 3.14
3 PRINT VAL ("-7")          output is -7
4 PRINT VAL ("ABC123")      output is 0
5 PRINT VAL ("123ABC")      output is 123
```

VERIFY

VERIFY["<file name>"[,<device>[,<number>…]]]

Type	Short	Appears	Location	Token HEX	Token DEC
COMMAND	V shift E	V—	FFDB	95	149

VERIFY checks whether the data on a storage device (like a disk or a cassette) matches the program stored in memory. It is typically used after a program has been saved and needs to be checked for accuracy when reloading. This is essential for ensuring the program has been saved correctly and can be loaded back without errors.

Example:

```
VERIFY                    checks first file on tape
VERIFY "TEST"             checks tape with file name
VERIFY A$                 tape program by string
VERIFY "TEST" ,8          disk program
VERIFY "TEST" ,8 ,1       disk in same saved memory
```

WAIT

WAIT<memory address>,<AND-mask>[,<XOR-mask>]

Type	Short	Appears	Location	Token HEX	Token DEC
STATEMENT	W shift A	W♠	D82D	92	146

WAIT pauses the execution of a program until a specific condition is met. The memory location is usually linked to system variables that track certain aspects of the computer, such as timing or input events.

The WAIT command with and-mask and flip-mask allows you to create more precise conditions for when the program should continue. By using bitwise operations, you can monitor specific bits in a memory location and wait for changes. The and-mask is used in a bitwise AND operation with the byte value at the given memory address. After masking the byte, the flip-mask is applied.

Example:

```
WAIT 197, 64          wait until no key is pressed
WAIT 197, 64, 64      wait until any key is pressed
WAIT 653, 1           wait until SHIFT key is pressed
WAIT 198, 8           wait until 8 keys are in key buffer
```

OPERATORS

OPERATORS perform mathematical, logical, or string operations on data.

Name	Appears	Example	Result
ADD	+	PRINT 5+3	8
SUBTRACT	−	PRINT 7−3	4
MULTIPLY	*	PRINT 6*7	42
DIVIDE	/	PRINT 20/4	5
EXPONENT	↑	PRINT 2↑3	8
EQUAL TO	=	PRINT 5=5	−1 (true)
LESS THAN	<	PRINT 4<6	−1 (true)
GREATER THAN	>	PRINT 7>9	0 (false)
NOT EQUAL TO	<> or ><	PRINT 5<>3	−1 (true)
LESS THAN OR EQUAL TO	<= or =<	PRINT 5<=5	−1 (true)
GREATER THAN OR EQUAL TO	>= or =>	PRINT 8>=3	−1 (true)
AND	AND	PRINT 5AND3	1
OR	OR	PRINT 5OR3	7
NOT	NOT	PRINT NOT5	−6
CONCATENATE	+	PRINT"A"+"B"	AB