

Basic Reference

Binary Operators

| Precedence | Operator | Notes |
|------------|----------|--|
| 4 | * | |
| | / | Forward slash is floating point divide. 22/7 is 3.142857 |
| | \ | Backward slash is integer divide, 22/7 is 3 |
| | % | Modulus of integer division ignoring signs |
| | >> | Logical shifts up to 32 places, inserting zeros at the appropriate ends. |
| | << | |
| 3 | + | |
| | - | |
| 2 | < | Compares as numbers or strings. If either is floating point it is compared as such, and the match is not exactly equal, but about 1 part in 100,000. Returns -1 for true, 0 for false. |
| | <= | |
| | > | |
| | >= | |
| | <> | |
| | = | |
| 1 | & | Binary operators on integers, but can be used as logical operators. Equivalent to and, or and exclusive or. |
| | | |
| | ^ | |

Unary Operators (General)

| Operator | Notes |
|------------------|--|
| alloc(n) | Allocate n bytes of 65C02 memory, return adress |
| asc(s\$) | Return ASCII value of first character or zero for empty string |
| atan(n) | Arctangent of n in degrees |
| chr\$(n) | Convert ASCII to string |
| cos(n) | Cosine of n, n ls in degrees. |
| deek(a) | Read word value at a |
| event(v,r) | event takes an integer variable and a fire rate (r) in 1/100s, and uses the integer variable to return -1 at that rate. See samples. |
| exp(n) | e to the power n |
| inkey\$() | Return the key stroke if one is in the keyboard buffer, otherwise returns a n empty string. |
| int(n) | Whole part of the float value n. Integers are unchanged. |
| isval(s\$) | Converts string to number, returns -1 if okay, 0 if fails. |
| key(n) | Return the state of the given key. The key is the USB HID key scan code. |
| left\$(a\$,n) | Left most n characters of a\$ |
| len(a\$) | Return length of string in characters. |
| log(n) | Natural Logarithm (e.g. ln2) of n. |
| max(a,b) | Return the largest of a and b (numbers or strings) |
| mid\$(a\$,f[,s]) | Characters from a\$ starting at f (1 indexed), s characters, s is optional and defaults to the rest of the line. |
| min(a,b) | Return the smaller of a and b (numbers or strings) |
| peek(a) | Read byte value at a |
| rand(n) | Random integer $0 < x < n$ (e.g. 0 to n-1) |
| right\$(a\$,n) | Rightmost n characters of a\$ |
| rnd(n) | Random number $0 < x < 1$, ignores n. |

| | |
|-----------------------|---|
| <code>sin(n)</code> | Sine of n, n is in degrees. |
| <code>sqr(n)</code> | Square root of n |
| <code>str\$(n)</code> | Convert n to a string |
| <code>tan(n)</code> | Tangent of n, n is in degrees. |
| <code>time()</code> | Return time since power on in 100 th of a seconds. |
| <code>val(s\$)</code> | Convert string to number. Error if bad number. |

BASIC Commands (General)

| Command | Notes |
|---|--|
| ' <string> | Comment. This is a string for syntactic consistency. The tokeniser will process a line that doesn't have speech marks as this is not common. REM this is a comment is now ' "this is a comment" and can be typed in as ' this is a comment |
| assert <expr> | Error generated if <expr> is zero. Used for checking parameters and/or enforcing contracts. |
| call <name>(p1,p2,p3) | Call named procedure with optional parameters. |
| cat | Show contents of current directory |
| clear | Clear out stack, strings, reset all variables. |
| cls | Clear screen to current background colour. |
| data <const>,.... | DATA statement. For syntactic consistency, strings must be enclosed in quote marks e.g. data "John Smith". |
| dim <array>(n,[m]), ... | Dimension a one or two dimension string or number array, up to 255 elements in each dimension (e.g. 0-254) |
| do ... exit ... loop | General loop you can break out of at any point. |
| doke <addr>,<data> | Write word to address |
| end | End Program |
| fkey <key>,<string> | Define the behaviour of F1..F10 – the characters in the string are entered as if they are typed (e.g. fkey 1,chr\$(12)+"list"+chr\$(13) clears screen and lists the program |
| for <var> = <start> to/downto <end> ... next | For loop. Note this is non standard, Limitations are : the index must be an integer. Step can only be 1 (to) or -1 (downto). Next does not specify an index and cannot be used to terminate loops using the 'wrong' index. |
| gosub <expr> | Call subroutine at line number. For porting only. See goto. |
| goto <expr> | Transfer execution to line number. For porting only. Use in general coding is a capital offence. If I write RENUMBER it will <u>not</u> support these. |
| if <expr> then | Standard BASIC if, executes command or line number. (IF .. GOTO doesn't work, use IF .. THEN nn) |
| if <expr>: .. else .. endif | Extended multiline if, without THEN. The else clause is optional. |

| | |
|--|--|
| input <stuff> | Input has an identical syntax and behaviour to Print except that variables are entered via the keyboard rather than printed on the screen. |
| let <var> = <expr> | Assignment statement. The LET is optional. |
| list [<from>][,][<to>] list <procedure>() | List program to display by line number or procedure name. |
| load "file"[,<address>] | Load file to BASIC space or given address. |
| local <var>,<var> | Local variables, use after PROC, restored at ENDPROC variables can be simple strings or numbers <i>only</i> . |
| new | Erase Program |
| poke <addr>,<data> | Write byte to address |
| print <stuff> | Print strings and numbers, standard format - , is used for tab ; to separate elements. |
| proc <nm>(p1,p2,p3 ...) endproc | Delimits procedures, optional parameters, must match call. |
| read <var>,... | Read variables from data statements. Types must match those in data statements. |
| repeat .. until <expr> | Execute code until <expr> is true |
| restore | Restore data pointer to program start |
| return | Return from subroutine called with gosub. |
| run | Run Program |
| save "file"[,<adr>,<sz>] | Save BASIC program or memory from <adr> length <sz> |
| stop | Halt program with error |
| sys <address> | Call 65C02 machine code at given address. Passes contents of variables A,X,Y in those registers. |
| while <expr> .. wend | Repeat code while expression is true |

The Inline Assembler

The inline assembler works in a very similar way to that of the BBC Micro, except that it does not use the square brackets [and] to delimit assembler code. Assembler code is in normal BASIC programs.

A simple example shown below (in the samples directory) :

It prints a row of 10 asterisks.

| | |
|----------------------------|---|
| 100 mem = alloc(32) | Allocate 32 bytes of memory to store the program code. |
| 110 for i = 0 to 1 | We pass through the code twice because of forward referenced labels. This actually doesn't apply here. |
| 120 p = mem | P is the code pointer – it is like * = <xx> - it means put the code here |
| 130 o = i * 3 | Bit 0 is the pass (0 or 1) Bit 1 should display the code generated on pass 2 only, this is stored in 'O' for options. |
| 140 .start | Superfluous – creates a label 'start' – which contains the address here |
| 150 ldx #10 | Use X to count the starts |
| 160 .loop1 | Loop position. We can't use loop because it's a keyword |
| 170 lda #42 | ASCII code for asterisk |
| 180 jsr \$fff1 | Monitor instruction to print a character |
| 190 dex | Classic 6502 loop |
| 200 bne loop1 | |
| 210 rts | Return to caller |
| 220 next | Do it twice and complete both passes |
| 230 sys mem | BASIC instruction to 'call 6502 code'. Could do sys start here. |

Most standard 65C02 syntax is supported, except currently you cannot use lsr a ; it has to be just lsr (and similarly for rol, asl, ror, inc and dec)

You can also pass A X Y as variables. So you could delete line 150 and run it with

X = 12: sys start

which would print 12 asterisks.