Numbers and misc

```
;                   ;comment
*                   ;comment when in 1st column
a=1234              ;base 10
a=$1234             ;base 16
a=%11010            ;base 2
a=a + &             ;line continuation
  b
a$="Howdy"          ;strings -- " or ' allowed delimeters
a$=!5"hola"!13      ;like chr$(5)+"hola"+chr$(13)
a$=!s"hey"          ;!s -- use screen codes instead of ascii
FIXME
                    ;!r -- use regular (ascii) codes
                    ;!n -- negate (ora $80) following codes
Slang is CaSe InSeNsItIvE
```

Variable types and declarations

```
byte, ubyte         ;1 byte, signed/unsigned
int, uint           ;2 bytes, signed/unsigned
float               ;5 bytes, MFLPT format
byte blah@$d020           ;@-var: locate variable at $d020
int blah(3,3)             ;Create 2D array
int screen(40,25)@$0400 ;Create 2D array, locate at text screen
int blah(3,3)=[ [1 2 3] &
              [4 5 6] &
              [7 8 9] ]   ;Create 2D array and pre-initialize values
ubyte str(20)=['..can also use strings to pre-initialize']
blah(2,0) = 1             ;Note that array indices start at 0
deftype foo               ;Compound variable type declaration
  int .a                  ;note leading .
  byte .b(10)
  float .x
defend
type foo yak1, yak2       ;Create two variables of compound type foo
yak1.a = 10               ;...which can then access individual elements
yak.b(3) = -8
yak1.x = 3.1415926
```

note: currently, arrays of compound variables are not allowed (although arrays may be inside compound vars).

```
VarBlock @$c000           ;Define all subsequent variable starting at $c000
  ubyte t1
  int t2
EndVarBlock               ;between the VarBlock and EndVarBlock
byte b
int c
c=#b+3                    ;Set c to the _address_ of b plus three
```

Strings
note: strings are null-terminated.

```
byte str(20)            ;Strings are byte arrays
byte str2d(5,20)        ;2D array of bytes/strings
str(1) = 1              ;Treat as byte array
str$="Hey"!13           ;Using the $ sign treats as string
if str$(3:5) = "bla"    ;Can specify substring ranges
  str$(3)="blah"        ;str will be xxxblah00
```

Byte arrays (actually any type) can be treated as a string by using $ after the variable name. Strings are really more of an addressing mode than a variable type.

Subroutines
note: <- indicates backarrow key

```
sub blah()                          ;Subroutine with no input or output variables
sub blah2(int x, byte r)            ;Subroutine with two input variables
sub blah3(int z)<-byte s,int t      ;Subroutine with one input variable (z)
                                    ;and two output variables (s, t)
sub blah4(@ax)          ;One input variable, in the .A (lo) and .X (hi)
                        ;registers. @x @a @xy @yx etc. are all available,
                        ;where @ax syntax -> @(lo)(hi) 16-bits.
sub asm chrout@$ffd2(@a)      ;Create subroutine interface, without
                              ;starting new subroutine (e.g. for calling
                              ;external routines)
endsub                  ;End subroutine (rts)
```

Usage:

```
blah3(10)               ;Call subroutine on its own
a=blah3<-s              ;Use subroutine variables in expressions
b=blah3<-t + a + 1

a=blah3(12)<-s          ;can also combine operations
b=blah3<-t + a - 1
```

Operators

```
=                ;Assignment operator, e.g. a=1

bitand, bitor, biteor   ;bitwise operators
+                       ;addition, string concatenation
- * / ^                 ;standard math operators
                        ;note: integer * and / require core library
                        ;      float operations require BASIC switched in
<< >>                   ;shift operators (note: fixed only, not float)
< <= > >= = !=          ;comparison operators.  Returns 0 (false) or 1 (true)
and, or, eor            ;logical comparison operators.  Returns 0 (false)
                        ;or nonzero (true).  Note: identical to bitand etc.
                        ;but lowest precedence
```

note: comparison operators can be used with strings

Functions
Unlike keywords, functions can be used in expressions.

```
byte, ubyte, int, uint, float ;Type conversion
a = int(x)+c                  ;Convert x to type int, then add
```

floating-point functions: (BASIC ROM routines)

```
abs         ;absolute value
atn         ;arctangent (note: will rename as atan)
cos         ;cosine
exp         ;exponential
trunc       ;truncate
log         ;logarithm
rnd         ;random number
sgn         ;sign
sqrt        ;square root
tan         ;tangent
```

misc functions:

```
not         ;logical not (0->1, nonzero->0)
```

VIC functions:

```
SpriteColSpr(number)    ;Check sprite to sprite collision.  Returns 0/nonzero
```

Flow control, loops
The following flow commands may be followed by a ! to execute the block when the condition is _false_:

```
      if!         ;ifnot -- if not true then...
      elseif!
      while!      ;whilenot -- while not true...
      until!      ;untilnot -- until not true...
      case!       ;casenot -- like ifnot

if (expr)         ;standard if-block
  [code]
elsif (expr)
  [code]
else
  [code]
endif


for x=start:end [step y]      ;standard for-loop.  step is optional
  [code]
endfor -or- next              ;next is alias for endfor


while (expr)                  ;standard while-loop
  [code]
endwhile


repeat            ;standard repeat-loop
  [code]
until (expr)
```

```
repeat            ;infinite repeat loop
  [code]
forever
```

test-case commands -- Essentially an if-then shorthand

```
testall                 ;test all of the following cases (inclusive test)
testelse                ;test cases like if-elsif-elsif (exclusive test)
  case (expr)           ;execute following code if (expr) true
  case (expr)           ;multiple case statements may follow
endtest                 ;end statement
```

Printing and I/O commands

```
sprint [string]         ;Simple print (or String print) -- prints strings.
sprintln                ;Like sprint, but adds a CR at the end.
sprint(10,3) "yo!"      ;Print to row 10, column 3 -- () is optional
print                   ;Main print command.  Prints strings and numbers/
                        ;expressions, optionally separated by commas.
print !5"Hey!"!13
print "x=" x " and y=",y
print(10,3) "b=" float(b+10*sin(x))
println                 ;Print, followed by CR
```

notes: print requires the core library for strings and fixed expressions; float expressions use the BASIC print routines.

```
getchar b               ;read a char, store in b.  Does not wait for char
waitchar b              ;wait for keypress, store in b
waitchar                ;wait for keypress
input b$                ;input a string using kernal CHRIN, store in b
inputstr(maxlen) b$     ;input using core library routine.  Max string
                        ;length is in maxlen.
load "filename",dev,address   ;load file to memory
save "filename",dev,start,end ;save memory to file
```

note: with SuperCPU, address/start/end can be 24-bit values, provided pppatch is running.

Misc

```
poke                    ;similar to BASIC poke
wait address,value      ;loop until address = value
wait $d012,255          ;wait for $d012=255
waitne, waitge, waitlt  ;wait until address !=, >=, or <= value
waitbitsclear           ;wait for specific bits to be clear.  Value is ANDed
                        ;with address
waitbitsset             ;wait for specific bits to be set
fillmem(start,end,fill byte)  ;Fill a range of memory with a byte
done                    ;end program (rts)
donebrk                 ;end program (brk)
endsub                  ;end subroutine
endirq                  ;end irq routine
put 'filename'          ;PUT the specified file into the code at current
                        ;location
```

```
saveobj "@0:testcode.o" ;If compile successful, save object code
autorun                 ;If compile successful, automatically run program
SetZPTemp address       ;Set temporary ZP stack location (default=$02)
SetEvalTemp address     ;Set location for temporary results (default=$0110)
SetStrBuf address       ;Set location for string buffer (default=$0200)
```

Interrupts

```
irq name(varlist)       ;irq is an alias for sub -- create an IRQ routine
endirq                  ;...but end it with endirq instead of endsub
jmpkernalirq            ;or exit through the normal ROM IRQ
SetIRQRoutine(name)     ;Set the system IRQ vector ($0314) to point to the
                        ;specified routine
DisableInterrupts       ;Disable Interrupts (sei)
EnableInterrupts        ;Enable (cli)
EnableTimerAIRQ         ;Enable CIA#1 timer A IRQ (system IRQ)
DisableTimerAIRQ        ;Disable
SetTimerA(value)        ;set CIA#1 timer A value
AckTimerAIRQ            ;Acknowledge timer A IRQ
EnableRasterIRQ         ;Enable the VIC raster IRQ
DisableRasterIRQ        ;Disable
AckRasterIRQ            ;Acknowledge VIC raster irq
SetRaster(value)        ;Set raster value for IRQ to occur at
```

VIC commands

```
UseSpriteStuff          ;Use before using below sprite commands
SetSpriteX(#,xpos)      ;Set X-position for sprite number #
SetSpriteY(#,ypos)      ;Set y-position
SetSpriteColor(#,color) ;Set color
SetSpritePtr(#,block)   ;Sets the sprite pointer to block.  This command
                        ;will be updated in the future to something more
                        ;sensible/useful
SpriteOn(#)             ;Turn sprite number # on
SpriteOff(#)            ;Turn it off
SetScrollX(value)       ;Set the fine x-scroll value (0-7)
SetCol38                ;Set 38 columns
SetCol40                ;Set 40 columns
SetCharData(address)    ;Set the character dot data to address
SetVideoMatrix(address) ;Sets the video (screen matrix) address
SetVICBank(bank)        ;Sets the 16k graphics bank to 0-3
MemConfig number        ;Sets the memory configuration (location $01, 0-7)
```

Linker

There are just three linker commands: REL, ENT, and EXT

REL - Assemble as RELocatable file. Placed at top of code.
ENT - Declare label/variable/subroutine as an ENTry point, available to other modules.
EXT - Declare label/variable/subroutine as EXTernal to the current module (declared as an ENT in some other module).

Names are then resolved at link time.

Format of link file:

```
link $1000
module1.l
module2.l
...
```

Examples:

```
byte ent blah     ;Make "blah" available to other modules
byte ext blah     ;Declare "blah" as a variable external to other modules.
sub ent routine(int a, int b) ;Make subroutine available to other modules
```

Editor

Ctrl-h from editor brings up all available commands.
SYS54016 will re-enter editor upon reset, etc.

Jammon (ML monitor)

Please see the jammon docs (they're short!) for details.

Memory map

While Slang environment is running:

$0801-$9F00	Slang core
$A000-$B800	Compiler commands
$BE00-$BFFF	Used by slang for temporary results
$C000+	Variable storage during compilation
bank 2:	Code assembled to bank 2
bank 3:	Used for temporary storage during assembly
bank 8+:	Editor text stored here
hi bank:	Used to store slang environment

Upon exiting Slang environment (exit to monitor, running program, etc.):

- monitor in bank 0 at specified address, if exited to monitor
- PPPatch at $CE80-$CFFF or so
- Re-entry code stored to $01d300
- Editor text moved up in memory as high as it can, freeing lower banks
- Slang environment, etc. stored in highest available bank, e.g. on a 16M system this will be bank $F5.