**Tokenisation**

The following tokens are stored in low/high order. Each is 16 bits wide except for that representing an ASCII string.

End Token : 0000 0000 0000 0000 $0000

This token is used to mark the end of a sequence of tokens (e.g. a line of tokenised code)

ASCII String : 0000 0000 LLLL LLLL $0000-$00FF

This represents a single ASCII string. The token identifies it as a string and gives the overall length in bytes of the string. This is not the same as the length of the string. It also includes the header word (2 bytes/1 word) and a trailing 00 byte. The latter is mandatory. The string is stored in such a way that the address of the following token is itself the ASCIIZ string. If there is an even length string (i.e. with the terminating NULL there is an even number of bytes) this should be padded out so the overall token size is zero.

Constant Shift : 0001 CCCC CCCC CCCC $1000-$1FFF

This allows the extension of constants from 15 bits to 27 bits, e.g. enough to encompass the address range of a 65816 CPU. When there is a constant shift, it is shifted left 15 times and added to the 15 bit standard constant. This clears the constant shift.

Identifier : 001E DDDD DDCC CCCC $2000-$3FFF

The identifier token has 2 6 bit ASCII values packed into a 16 bit word. CCCCCC and DDDDDD are calculated by subtracting 32 from the upper case ASCII value 32-95, only 0-9 @ A-Z are allowed at present. 0-9 are not permitted for the first character. If the identifier has an odd length, the name is padded with DDDDDD = 0.

Tokens are sequenced together to form long identifiers. E is used as a continuation bit, e.g. it is clear for the last identifier token. This means that single character variables must be 0010 0000 00CC CCCC where CCCCCCC is 32-58 (@-Z) which means the permanent fast variables @ and A-Z can be easily identified.

Keywords : 01TT TTKK KKKK KKKK $4000-$7FFF

These represent tokenised keywords. The numbers are not continuous though the KKKK KKKK KK value is – this is the actual keyword token. TTTT identifies it’s type as follows.z

0000-0111 Binary Operators. 0000 is the lowest level

(e.g. & | ^)

1000-1100 Unused at present

1101 Unary function. Note some binary functions such

as ! ? and – are also unary ones.

1110 Syntactic only keywords – things that aren’t a

command, such as , ; and TO

1111 An executable keyword command LET LIST etc

Constant : 1CCC CCCC CCCC CCCC $8000-$FFFF

Normally represents the constant 0-32767. The constant shift can extend this to 2^27-1. This shift is always zeroed after being applied.

**Program Storage**

Programs are a collection of records in line number order. The header is 2 words ; firstly an ***offset*** to the next line. If this is zero this indicates the program end. Following that is the tokenised line number, from 0-32767 – this has bit 15 set as per the tokenised constant.

Following that is a sequence of tokens ending in the token $0000.

**Variable Storage**

Variable storage immediately follows the end of program storage (except A-Z). Each record occupies 8 bytes. The *address* of the next variable. The *address* of the identifier as a sequence of tokens (stored in program code normally) , and finally a four byte word representing the current value. Because names are stored in program code as is normal in BASIC any program editing clears the variables (except @ and A-Z)

All addresses are absolute addresses, so if the BASIC workspace area is 3000-C000, then those addresses are independent of that.

Variables are stored in linked lists, ending with a next variable address (the link) of $0000. There is an array of word addresses which is accessed via a hash on the first token, which is obtained by xoring the two bytes of the token.

For ease of readability there is a gap of 2 words containing the values $EEEEEEEE between the program code and the start of variable / data space.

**Memory Usage**

All are offset from the base address.

+0000 Module identifier “BASC” (in that order)

+0010-+007F The variables A – Z, 4 bytes each. @ is at

$0010, A at $0014 … to Z at $0078

+0080-+009F Up to 16 linked list pointers. The number

actually used is 1,2,4,8 or 16

+00A0 Next free byte after program store (reset on

CLEAR, any program editing, NEW etc.)

+00A2 Byte after last used byte ; top of memory.

+00C0 First program record (offset to next)

**Allocated Memory**

Allocated memory – for strings or arrays, is stored from the top of memory down, e.g. from the value stored at offset ($A2) to the top of the memory block.

With protection turned on, the binary indirection operators will only write to permitted blocks.

Unary indirection operators are unprotected and will work anywhere. If you do !<address> and it blows something up, it’s your problem.