BBC BASIC (Z80)

BBC BASIC (Z80) for the Sinclair Spectrum

1. Introduction

BBC BASIC (Z80) has been designed to be as compatible as possible with Version 4 of the 6502 BBC BASIC resident in the BBC Micro Master series. The language syntax is not always identical to that of the 6502 version, but in most cases the Z80 version is more tolerant. BBC BASIC for the Spectrum is based on the implementation for the Acorn Z80 CoProcessor using BBC file I/O.

BBC BASIC can be installed in a sideways ROM selected with port &7FFD or in DOCK or EX. It can also be loaded into main RAM, resulting in there being about 17K less space free for user programs and data.

When BASIC starts, the computer will display:

```
Acorn BBC BASIC Version 2.20
(C) Copyright R.T.Russell 1983
>
```

If the system allows seperate processes, then *Quit will return to the calling process. *Help will report the current host interface version.

2. Memory utilisation

BBC BASIC runs in low memory in ROM/RAM at &0000-&3FFF or in high memory at &BB00-&EFFF. About about 1K of memory is reserved after the Spectrum screen for Spectrum system variables and data, resulting in a value of PAGE of &6000. The remainder of the user memory is available for BASIC programs, variables (heap) and stack. Depending on the system configuration, HIMEM can have a value up to &F000.

If a Timex MMU is present and RAM is available at &4000-&7FFF, then the screen is paged out, resulting in PAGE starting at &4000.

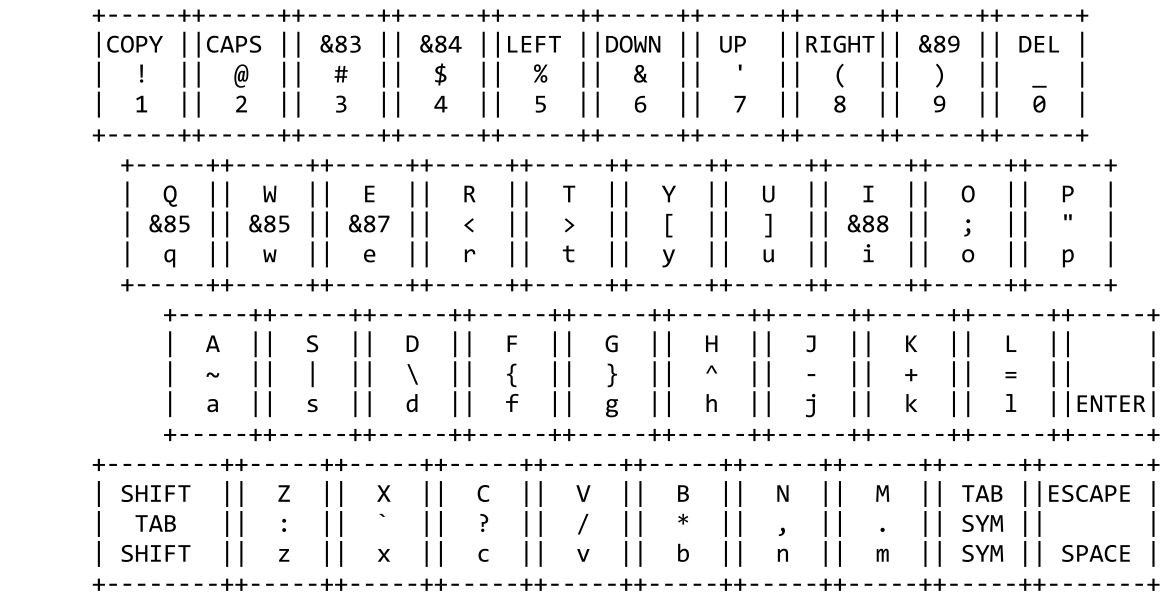
PAGE may be raised or HIMEM lowered by other parts of the host system reserving memory.

3. Commands, statements and functions

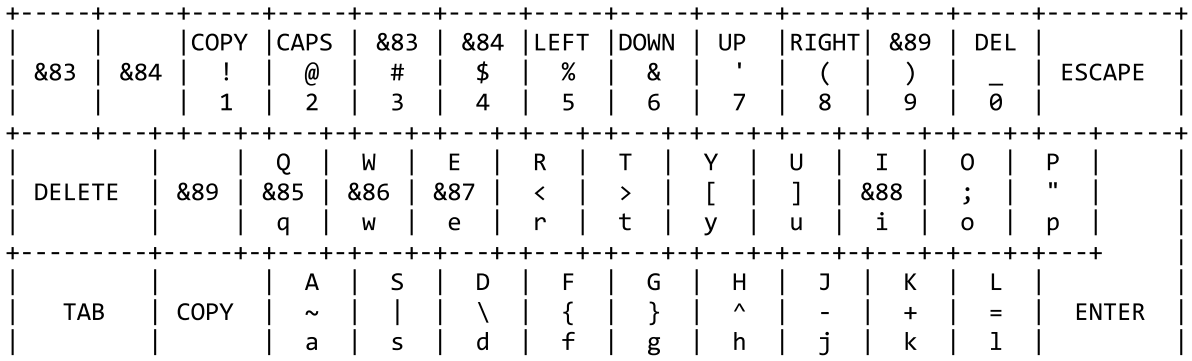
The syntax of BASIC commands, statements and functions are the same as the 6502 BASIC for the BBC Micro version (BASIC 4). Some commands and functions are implemented by calling operating system entry points at &FF00-&FFFF and so their functionality depends on the level of support implemented by the host interface. The differences in the current host interface (version 0.4x) are listed here.

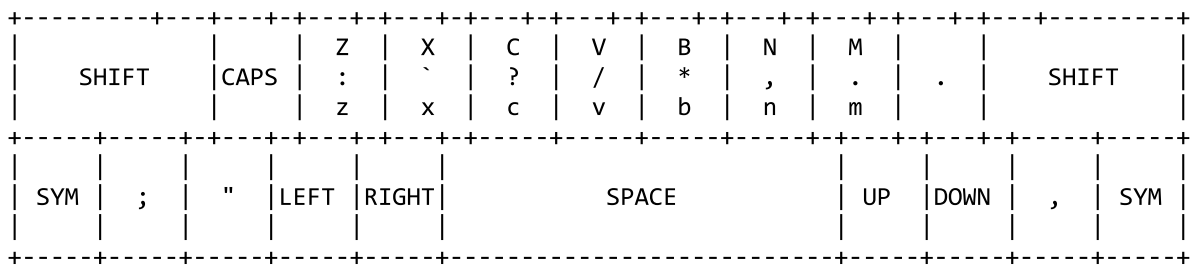
4. Keyboard layout

The Spectrum uses the following keyboard layout when running BBC BASIC.



The Spectrum Plus uses the following keyboard layout when running BBC BASIC.





5. Special characters

!	32-bit indirection
$	String variable or string indirection
%	Integer variable
&	Precedes hexadecimal constant
'	New line in PRINT or INPUT
*	Precedes an "operating system" statement
:	Separates statements typed on the same line
;	Introduce comment in assembler, suppress action in PRINT
?	8-bit indirection (PEEK & POKE)
[	Enter assembler
]	Exit assembler
~	Convert to hex (PRINT and STR$)

6. Variables

Variable names may be of unlimited length and all characters are significant. Variable names must start with a letter. They can only contain the characters A..Z, a..z, 0..9 and underline. Embedded keywords are allowed. Upper and lower case variable names are distinguished.

The following types of variable are allowed:

A	real numeric
A%	integer numeric
A$	string

The variables A%..Z% are regarded as special in that they are not cleared by the commands or statements RUN, CHAIN and CLEAR. In addition A%, B%, C%, D%, E%, F%, H%, L%, X% and Y% have special uses in CALL and USR routines and O% & P% have special meanings in the assembler (code origin and program counter respectively). The special variable @% controls numeric print formatting. The variables @%..Z% are called "static variables", all other variables are called "dynamic variables".

Real variables have a range of approximately +-1E-38 to +-1E38 and numeric functions evaluate to 9 significant figure accuracy. Internally

every real number is stored in 40 bits.

Integer variables are stored in 32 bits and have a range of -2,147,483,648 to 2,147,483,647.

String variables may contain from 0 to 255 characters.

All arrays must be dimensioned before use.

All statements can also be used as direct commands.

7. Immediate Commands

Immediate commands can only be enter at the BASIC prompt. An attempt to use them in a program will give the 'Mistake' error.

Command	Syntax	Function
AUTO	AUTO [start][,inc]	Generate line numbers.
DELETE	DELETE start,end	Delete program lines.
LIST	LIST [line][,line]	List all or part of program.
LISTO	LISTO number	Control indentation in LIST.
LOAD	LOAD "filename"	Load a program into memory.
NEW	NEW	Delete current program & variables.
OLD	OLD	Recover a program deleted by NEW.
RENUMBER	RENUMBER [start][,inc]	Renumber the program lines.
SAVE	SAVE "filename"	Save the current program to disk. If the first line of the program is a line 'REM > filename' then SAVE with no filename will use the filename in the REM statement.

8. Statements

The following statements can be used in programs or directly from the BASIC prompt.

Statement	Syntax	Function

```
CALL      CALL address[,arg list]       Call assembly language routine.

CHAIN     CHAIN string                  Load and run a program.

CLEAR     CLEAR                         Clear dynamic variables.

CLS       CLS                           Clear the screen.

DEF       DEF FNname[(arg list)]        Define a function.
          DEF PROCname[(arg list)]      Define a procedure.

DIM       DIM var(sub1[,sub2...])[,..]  Dimension one or more arrays.
          DIM var exp [,var exp...]     Reserve space for assembler etc.

END       END                           Terminate program and close files.

ENDPROC   ENDPROC                       Return from a procedure.

FOR       FOR var=exp TO exp [STEP exp] Begin a FOR...NEXT loop.

GOSUB     GOSUB exp                     Call a BASIC subroutine.

GOTO      GOTO exp                      Branch to specified line.

IF        IF exp THEN stmts [ELSE stmt] Do statement(s) if exp non-zero.
          IF exp THEN line [ELSE line]  Branch if exp non-zero.

LET       [LET] var = exp               Assignment.

LOCAL     LOCAL var[,var...]            Declare variables local to
                                        function or procedure.

NEXT      NEXT [var[,var...]]           End FOR...NEXT loop.

ON        ON exp GOTO line,line.. [ELSE line]     Computed GOTO.
          ON exp GOSUB line,line.. [ELSE line]    Computed GOSUB.

ON ERROR  ON ERROR stmts                Do statement(s) on error.
          ON ERROR OFF                  Restore default error handling.

PROC      PROCname[(parameter list)]    Call a procedure.

REM       REM any text                  Remark

REPEAT    REPEAT                        Begin a REPEAT...UNTIL loop.

REPORT    REPORT                        Print error message for last error.

RESTORE   RESTORE [line]                Reset data pointer to beginning or
```

		to specified line.
RETURN	RETURN	Return from subroutine.
RUN	RUN	Run the current program.
STOP	STOP	Stop program and print message.
TRACE	TRACE ON	Start trace mode.
	TRACE OFF	End trace mode.
	TRACE exp	Trace lines less than exp.
UNTIL	UNTIL exp	Terminate loop if exp is non-zero.
WIDTH	WIDTH exp	Set print output width.

9. I/O Statements

Statement	Syntax	Function
BPUT	BPUT #chan,exp	Write LS byte of exp to disk file.
CLOSE	CLOSE #chan	Close open file. IF chan=0 close all files.
DATA	DATA list	Data for READ statement.
EXT#	EXT#chan=exp	Sets the extent of an open file.
INPUT	INPUT ["prompt"[,]]var[,var]	Request input from user. Comma after prompt causes question mark.
	INPUT LINE ["prompt....	As INPUT but accept whole line including commas, quotes etc.
INPUT#	INPUT #chan,var[,var...]	Read data from open file.
OSCLI	OSCLI string	Pass string to "operating system".
PRINT	PRINT [TAB(x[,y])][SPC(n)]['][;][~][exp[,exp...][;]	Print data to output stream.
PRINT#	PRINT #chan,exp[,exp...]	Write data to open file.
PTR#	PTR#chan=exp	Sets the current pointer for an open file.
PUT	PUT port,exp	Output LS byte of exp to port.

READ	READ var[,var...]	Read data from DATA statement(s).
VDU	VDU exp[,exp...]	Send LS byte of exp to output stream.
	VDU exp;[exp;...]	Send LS 16 bits of exp to output stream as two characters (LS byte first).

10. Print Formatting

By default, strings are printed left-justified and numbers are printed right-justified in a print zone. Numeric quantities will be printed left-justified if preceded by a semicolon (;). A comma (,) causes a tab to the beginning of the next print zone, unless the cursor is already at the start of a zone. An apostrophe (') in a PRINT or INPUT statement forces a new-line. A trailing semicolon in a PRINT statement suppresses the new-line. TAB(x), TAB(x,y) and SPC(n) may be used in PRINT and INPUT statements to position the cursor. A tilde (~) causes numbers to be printed in hex.

The variable @% controls numeric formatting as follows:

```
LS byte:     Width of print zone, 0-255.  Normally 10.
Byte 2 :     Number of significant figures or decimal places. Maximum 10.
Byte 3 :     Print format type:  0 - General format (default)
                                 1 - Exponential format
                                 2 - Fixed format.
MS byte:     STR$ flag. If zero then STR$ formats in G9 mode. If nonzero
             zero then STR$ formats according to bytes 2 & 3 of @%.
```

```
Examples                     Result
@%=&2010A                    01234567890123456789

PRINT "HELLO",8              HELLO            8.0
PRINT "HELLO" 8              HELLO       8.0
PRINT "HELLO";8              HELLO8.0
PRINT "HELLO",;8             HELLO     8.0
```

Value	G9 @%=&90A	G2 @%=&20A	E2 @%=&1020A	F2 @%=&2020A
.001	1E-3	1E-3	1.0E-3	0.00
.006	6E-3	6E-3	6.0E-3	0.01
.01	1E-2	1E-2	1.0E-2	0.01
.1	0.1	0.1	1.0E-1	0.10
1	1	1	1.0E0	1.00
10	10	10	1.0E1	10.00

```
100              100           1E2          1.0E2        100.00
1000            1000           1E3          1.0E3       1000.00
```

11. Operators

Symbol	Function
+	Addition or string concatenation.
-	Negation or subtraction.
*	Multiplication.
/	Division.
^	Involution (raise to power).
NOT	One's complement (integer).
EOR	Bitwise exclusive-OR (integer).
OR	Bitwise OR (integer).
AND	Bitwise AND (integer).
MOD	Modulus (integer result).
DIV	Integer division (integer result).
=	Equality.
<>	Inequality.
<	Less than.
>	Greater than.
<=	Less than or equal.
>=	Greater than or equal.

The precedence of operators is:

1. Expressions in parentheses, functions, negation, NOT.
2. ^
3. *, /, MOD, DIV
4. +, -
5. =, <, >, <>, <=, >=
6. AND
7. OR, EOR

12. Arithmetic Functions

Function	Action
ABS(exp)	Absolute value of expression.

ACS(exp)	Arc-cosine of expression, in radians.
ASN(exp)	Arc-sine of expression, in radians.
ATN(exp)	Arc-tangent of expression, in radians.
COS(exp)	Cosine of radian expression.
DEG(exp)	Value in degrees of radian expression.
EXP(exp)	e raised to the power of expression.
INT(exp)	Largest integer less than expression.
LN(exp)	Natural logarithm of expression.
LOG(exp)	Base-ten logarithm of expression.
RAD(exp)	Radian value of expression in degrees.
RND[(exp)]	RND returns random 32-bit integer. RND(-n) seeds sequence. RND(0) repeats last value in RND(1) form. RND(1) returns number between 0 and 0.999999999 RND(n) returns random integer between 1 and n.
SGN(exp)	1 if exp>0, 0 if exp=0, -1 if exp<0.
SIN(exp)	Sine of radian expression.
SQR(exp)	Square root of expression.
TAN(exp)	Tangent of radian expression.

13. String Functions

Function	Action
ASC(str)	Returns ASCII value of first character of string. Returns -1 if null string.
CHR$(exp)	Returns one-character string with ASCII value of exp.
EVAL(str)	Evaluates str as an expression and returns resulting number or string.
GET	Waits for keypress and returns ASCII value.
GET$	Waits for keypress and returns one-character string.

INKEY(exp)	Waits exp centiseconds for keypress and returns ASCII value. If no keypress then returns -1.
INKEY$(exp)	Waits exp centiseconds for keypress and returns one-character string. If no keypress returns null string.
INSTR(r,s[,n])	Returns position of string s in string r, optionally starting at position n.
LEFT$(str,exp)	Returns leftmost exp characters of string.
LEN(str)	Returns length of string (0-255).
MID$(str,m[,n])	Returns sub-string from position m, of length n or to end.
RIGHT$(str,exp)	Returns rightmost exp characters of string.
STR$[~](exp)	Returns string representation of exp in decimal (or hex).
STRING$(n,str)	Returns a string consisting of n copies of str.
VAL(str)	Returns numeric value of str. IF str does not begin with a signed or unsigned numeric constant, VAL returns zero.

14. I/O And Special Functions

Function	Action
BGET#chan	Returns a single byte from an open file.
COUNT	Number of characters printed since last new line.
END	Returns end of BASIC's variable heap.
EOF#chan	Returns TRUE if open file is at its end.
ERL	Line number of last error.
ERR	Code of last error.
EXT#chan	Returns length of open file.
FALSE	Returns zero.
FNname[(parameter list)]	User-defined numeric or string function.
GET(port)	Returns contents of Z80 port.

MODE	Returns current screen mode.
OPENIN(str)	Opens file for input and returns channel number.
OPENOUT(str)	Opens file for output and returns channel number.
OPENUP(str)	Opens file for update and returns channel number.
PI	Returns 3.14159265.
POS	Returns current cursor column (LHS=0).
PTR#chan	Reads the current pointer for an open file.
TOP	Returns first address after end of user's program.
TRUE	Returns -1.
USR(address)	Calls machine-code routine and returns integer.
VPOS	Returns current cursor line (top line=0).

15. Pseudo-variables

Pseudo-variables allow the user both to read and modify system variables.
They may be used either side of an assignment statement, eg TIME=TIME+50
(but note that LET is not permitted).

Name	Function
PAGE	Memory address of current user's program.
HIMEM	Top of memory used by BASIC.
LOMEM	Start address of dynamic variable storage.
TIME	Elapsed time clock, counts in centiseconds.
TIME$	Real Time Clock time and date in format "Day,dd Mon yyyy.hh:mm:ss".

16. Error codes

Immediate Mode Only (error code 0):

Silly	RENUMBER space	LINE space

Disastrous and untrappable:

```
   Bad program          No room              Sorry

Trappable:
   1 Out of range        4 Mistake
   5 Missing ,           6 Type mismatch
   7 No FN               9 Missing "
  10 Bad DIM            11 DIM space
  12 Not LOCAL          13 No PROC
  14 Array              15 Subscript
  16 Syntax error       17 Escape
  18 Division by zero   19 String too long
  20 Too big            21 -ve root
  22 Log range          23 Accuracy lost
  24 Exp range          26 No such variable
  27 Missing )          28 Bad HEX
  29 No such FN/PROC    30 Bad call
  31 Arguments          32 No FOR
  33 Can't match FOR    34 FOR variable
  36 No TO              38 No GOSUB
  39 ON syntax          40 ON range
  41 No such line       42 Out of DATA
  43 No REPEAT          45 Missing #

 190 Directory full    192 Too many open files
 196 File exists       198 Disk full
 200 Close error       204 Bad name
 214 File not found    222 Channel
 253 Bad string        254 Bad command
```

17. Indirection operators

Indirection is the process which is provided by PEEK and POKE in other dialects of BASIC. There are three indirection operators:

	Name	Purpose	No. of bytes affected
?	query	byte indirection operator	1
!	pling	word indirection operator	4
$	dollar	string indirection operator	1 to 256

```
   Y=PEEK(X) is equivalent to Y=?X
   POKE X,Y  is equivalent to ?X=Y
```

! acts on four successive bytes. For example, !M=&12345678 would load &78 into address M, &56 into address M+1, &34 into address M+2 and &12 into address M+3. $ writes a string, followed by carriage return, into memory at a specified address, e.g. $M="ABCDEF" will place the ASCII characters A to F in locations M to M+5 and will load &0D into address M+6.

Query (?) and pling (!) can also be used as binary operators, e.g. M?3 means "the contents of memory location M+3". The left-hand operand must be a variable, not a constant.

The power of indirection operators is in the way they can be used to create your own data structures. For example you may need a structure consisting of a 10 character string, an 8-bit number and a reference to a similar structure. If M is the address of the start of the structure then:

```
   $M   is the string
   M?11 is the 8-bit number
   M!12 is the address of the related structure.
```

In this way you can create and manipulate linked lists and tree structures in memory, very easily.

18. Access to machine code

The USR function and the CALL statement provide a flexible interface between BASIC and machine-code routines. Both USR and CALL initialise the Z80's registers prior to the machine-code call as follows:

```
 A register = LS byte of A%
 F register = LS byte of F%
 B register = LS byte of B%
 C register = LS byte of C%
 D register = LS byte of D%
 E register = LS byte of E%
 H register = LS byte of H%
 L register = LS byte of L%
IY register = address of machine-code routine (=PC)
IX register = address of parameter block (CALL statement only)
```

USR(address) Calls the machine-code routine and returns a 32-bit integer made up of the contents of the H,L,H' and L' registers (most-significant to least-significant) on return from the routine.

CALL address[,parameter list]

Sets up a parameter block containing the following information:

```
Number of parameters - 1 byte
Parameter type - 1 byte     ) repeated as often
Parameter address - 2 bytes ) as necessary.
```

Parameter types are as follows:

```
 0: 8-bit quantity (e.g. ?X)
 4: 32-bit integer variable (e.g. !X or X%)
 5: 40-bit floating-point variable (e.g. V)
```

128: a fixed string (e.g. $X, terminated by &0D)
129: a string variable (e.g. A$)

In the case of a string variable, the parameter address is the address of a String Descriptor containing start address, current length of string and number of bytes allocated.

Parameters are passed by reference and may be changed by the machine-code routine.

19. Resident Z80 assembler

BBC BASIC includes a full assembler for the appropriate host processor, in this case the Z80. It is an in-line assembler which loads its object code directly into the target memory area; the static variable P% is the program counter.

The in-line assembler is accessed in exactly the same way as the 6502 assembler in the BBC Micro version of BBC BASIC. That is, '[' enters assembler mode and ']' exits assembler mode.

Error reporting and listing are controlled by the pseudo-op OPT as follows:

OPT 0	Inhibits error reporting and gives no listing.
OPT 1	Inhibits error reporting but gives a listing.
OPT 2	Reports errors but gives no listing.
OPT 3	Reports errors and gives a listing.
OPT 4	As OPT 0 but puts code at O% rather than P%.
OPT 5	As OPT 1 but puts code at O% rather than P%.
OPT 6	As OPT 2 but puts code at O% rather than P%.
OPT 7	As OPT 3 but puts code at O% rather than P%.

Normally the first pass of the assembler will be with OPT 0 and the second pass with OPT 2 (if no listing is required) or OPT 3 (if a listing is required). Do not confuse the pseudo-op OPT with the Operating System Command *OPT.

All standard Zilog mnemonics are accepted: ADD, ADC and SBC must be followed by A or HL. For example, ADD A,C is accepted but ADD C is not. However, the brackets around the port number in IN and OUT are optional. Thus both OUT (5),A and OUT 5,A are accepted. The instruction IN F,(C) is not accepted, but the equivalent code is produced from IN (HL),C.

Pseudo-ops accepted by the assembler are DEFB, DEFW and DEFM; these write an 8-bit byte, a 16-bit word and a multi-character string respectively.

Example of use of assembler:

```
100 DIM code 15        :REM Reserve space for 16 bytes of code
110 bdos=5
120 FOR pass=0 TO 1    :REM Implement 2-pass assembly
130 P%=code            :REM Set program counter at start of each pass
140 [OPT pass*3        ;Enter assembler and select listing on pass 2
150 LD D,95:LD E,ASC"!"
160 .loop              ;A label
170 LD C,2             ;Source statements
180 PUSH DE:CALL bdos:POP DE
190 INC E:DEC D:JR NZ,loop
200 RET:]              :REM Exit assembler
210 NEXT pass
220 CALL code          :REM Execute assembly language routine
```

In this particular example a single pass would have sufficed as there are no "forward references".

20. Operating system interface

All Operating System ("star") commands are passed to the host to implement.

CALLs and USRs to addresses in the range &FF00 to &FFFF provide access to the machine operating system, as with the BBC microcomputer. The processor's A, H, L and E registers are initialised to the least significant bytes of the integer variables A%, X%, Y% and E% respectively. In the case of USR, the returned 32-bit value is composed of the processor's F, H, L and A registers corresponding to the 6502's P, Y, X and A registers, most significant to least significant.

The host interface provides the following entries to the operating system: OSFIND, OSGBPB, OSBPUT, OSBGET, OSARGS, OSFILE, OSRDCH, OSASCI, OSNEWL, OSWORD, OSBYTE, OSCLI at &FFCE to &FFF7.

21. Host System Interface

Some BBC BASIC statements and functions are implemented by calling operating system entry points at &FF00-&FFFF and so their functionality depends on the level of support implemented by the host interface. The differences in the current host interface (version 0.4x) are listed here.

ADVAL
Returns information about input devices such as a joystick or mouse, if fitted, or the number of free spaces in a buffer.

ADVAL(0) joystick buttons ADVAL(-1) bytes in keyboard buffer

```
ADVAL(1) joystick 1 X-position  ADVAL(-2) bytes in serial input buffer
ADVAL(2) joystick 1 Y-position  ADVAL(-3) free space in serial output
ADVAL(3) joystick 2 X-position  ADVAL(-4) free space in printer output
ADVAL(4) joystick 2 Y-position  ADVAL(-5) free space in chn 0 SOUND queue
                                ADVAL(-6) free space in chn 1 SOUND queue
ADVAL(7) mouse X position       ADVAL(-7) free space in chn 2 SOUND queue
ADVAL(8) mouse Y position       ADVAL(-8) free space in chn 3 SOUND queue
```

The Spectrum host currently does not return anything in response to ADVAL.

CALL

CALL passes control to a machine code subroutine. A CALL to an address in the &FF00-&FFFF range accesses various OS functions as detailed in the section detailing the OS interface.

CLG

Clears the graphics area of the screen and sets it to the currently selected graphics background colour using the current background plotting action (set by GCOL). Sends CHR$16 to the VDU. The current host clears the entire screen to the current colours.

BGET#

Reads a byte from a channel previously opened with OPENIN or OPENUP. The current host does not implement OPEN.

CLOSE#

Closes the specified channel. The current host does not implement OPEN, so CLOSE is also ignored.

CLS

Clears the text area of the screen and sets it to the currently selected text background colour. The text cursor is moved to the 'home' position (0,0) at the top left-hand corner of the text area. Sends CHR$12 to the VDU.

COLOUR

Sets the text foreground, background or border colour. Sends CHR$17;CHR$n to the VDU. On the Spectrum there are no logical colours, only physical colours. The colours are RGB encoded, rather than the Spectrum's native BRG encoding, with individual bits having the following meaning:

- Bit 0 - Red
- Bit 1 - Green

  Bit 2 - Blue
  Bit 3 - Bright
  Bit 4 - Flash
  Bit 5 - Change colour only, don't change Flash and Bright
  Bit 6 - Change Flash and Bright only, don't change colours
  Bit 7 - 0=Set foreground or flash/bright, 1=Set background or border

This results in the colour being set as follows:

  &00+n (0-63)   - text foreground colour
  &40+n (64-127) - flash and bright without changing colours
  &80+n (128-191) - text background colour
  &C0+n (192-255) - border colour

Foreground	Background	Border	Colour
0 &00	128 &80	192 &C0	Black
1 &01	129 &81	193 &C1	Red
2 &02	130 &82	194 &C2	Green
3 &03	131 &83	195 &C3	Yellow
4 &04	132 &84	196 &C4	Blue
5 &05	133 &85	197 &C5	Magenta
6 &06	134 &86	198 &C6	Cyan
7 &07	135 &87	199 &C7	White

The border cannot set to flashing or bright, so setting those bits are ignored.

DRAW
Draws a line in the current graphics foreground colour from the last point visited to the X and Y specified. The screen is always 1024 pixels wide and 1280 pixels high in all screen modes. DRAW x,y is equivalent to PLOT 5,x,y. The current Spectrum host ignores DRAW.

ENVELOPE
Used in conjunction with SOUND to control the pitch and/or amplitude of a sound whilst it is playing. The current Spectrum host ignores ENVELOPE.

EOF#
Returns TRUE if at the end of the opened file specified by the handle. The current host does not implement OPEN, so always returns FALSE.

EXT#
Returns the total length of the opened file specified by the handle. The current host does not implement OPEN, so always returns zero.

GCOL
Sets the current graphics foreground or background colour and plotting mode. Sends CHR$18;CHR$m;CHR$n to the VDU. The modes are:

  0 Plot the colour specified.
  1 OR the colour with the colour that is already there.
  2 AND the colour with the colour that is already there.
  3 Exclusive-OR the colour with the colour that is already there.
  4 Invert the colour that is already there.

The colour is specified as with COLOUR:

  &00+n (0-63)   - graphics foreground colour
  &40+n (64-127) - graphics flash and bright
  &80+n (128-191) - graphics background colour
  &C0+n (192-255) - undefined

The current Spectrum host ignores GCOL.

GET/GET$
Waits for the next character from the current input stream, usually the keyboard.

INKEY/INKEY$
With a positive argument waits for the specified maximum time in centiseconds for a character from the current input stream, or returns "" or -1 if no key pressed. With a negative argument returns the host type or checks for an individual keypress using the following BBC-compatible keycodes:

-001	Shift	-017	Q	-033		-049	1
-002	Symbol-Shift	-018	3	-034	W	-050	2
-003		-019	4	-035	E	-051	D
-004		-020	5	-036	T	-052	R
-005		-021		-037	7	-053	6
-006		-022	8	-038	I	-054	U
-007		-023		-039	9	-055	O
-008		-024		-040	0	-056	P
-009		-025		-041		-057	

-066	A	-082	S	-098	Z	-114	
-067	X	-083	C	-099	Space	-115	
-068	F	-084	G	-100	V	-116	
-069	Y	-085	H	-101	B	-117	
-070	J	-086	N	-102	M	-118	

-071	K	-087	L	-103	-119
-072		-088		-104	-120
-073		-089		-105	-121
-074	Enter	-090		-106	-122

The current Spectrum host does not implement keyboard scanning, returning FALSE. INKEY-256 returns &E0 to specify "Spectrum".

INPUT#

Reads data from an opened file with multiple calls to BGET. The current host does not implement OPEN, so INPUT# does not return data.

MODE

Sets the screen display mode. The screen is cleared and all the graphics and text parameters (colours, origin, etc) are reset to their default values. Sends CHR$22;CHR$n to the VDU. The current Spectrum host uses the same screen layout for all screen modes - 32 characters by 24 lines. This can be seen as a near equivalent to the BBC's mode 6.

MOVE

Moves the graphics cursor to an absolute position without drawing a line. The screen is always 1024 pixels wide and 1280 pixels high in all screen modes. MOVE x,y is equivalent to PLOT 4,x,y. The current Spectrum host ignores MOVE.

OPENIN

Opens a file for reading and returns the channel number of the file. A returned value of zero signifies that the specified file was not found, or could not be opened for some other reason. The current Spectrum host does not implement OPEN, and so returns zero.

OPENOUT

Opens a file for writing and returns the channel number of the file. If the specified file does not exist it is created. If the specified file already exists it is truncated to zero length and all the data in the file is lost. A returned value of zero indicates that the specified file could not be created. The current Spectrum host does not implement OPEN, and so returns zero.

OPENUP

Opens a file for update (reading and writing) and returns the channel number of the file. A returned value of zero signifies that the specified file was not found on the disk, or could not be opened for some other

reason. The current Spectrum host does not implement OPEN, and so returns zero.

OSCLI
Passes a string to be interpreted as an 'operating system' command.

PLOT
A multi-purpose drawing statement. Three numeric values follow the PLOT keyword: the first specifies the type of point, line, triangle, circle etc. to be drawn; the second and third give the X and Y coordinates to be used (in that order). The two most commonly used statements, PLOT 4 and PLOT 5, have the duplicate keywords MOVE and DRAW. PLOT sends CHR$25; CHR$k;CHR$(x MOD 256);CHR$(x DIV 256);CHR$(y MOD 256);CHR$(y DIV 256); to the VDU.

The following PLOT codes are defined:

PLOT 0 Move to a position relative to the last point.
PLOT 1 Draw line relative in graphics foreground colour.
PLOT 2 Draw line relative in logical inverse colour.
PLOT 3 Draw line relative in graphics background colour.
PLOT 4 Move to an absolute position.
PLOT 5 Draw line absolute in graphics foreground colour.
PLOT 6 Draw line absolute in logical inverse colour.
PLOT 7 Draw line absolute in graphics background colour.

PLOT 8-15 Last point in line omitted when inverted plotting used.

PLOT 16-23 Lines are drawn dotted.

PLOT 24-31 Lines are drawn dotted, omiting last point.

PLOT 32-39 Solid line, first point omitted.

PLOT 40-47 Solid line, both end points omitted.

PLOT 48-55 Dotted line, initial point omitted.

PLOT 56-63 Dotted line, both end points omitted.

PLOT 64-71 Single point is plotted.

PLOT 72-79 Horizontal line filling.

PLOT 80-87 Plot and fill a triangle formed by the specified position and the last two points visited.

PLOT 88-95 Horizontal line blanking.

The current Spectrum host does not implement PLOT.

POINT
Returns the colour of the screen at the coordinates specified. If the point is outside the graphics window, then -1 is returned. The current Spectrum host does not implement POINT.

POS
Returns the current horizontal position of the text cursor on the screen. The left hand column is 0 and the right hand column is one fewer than the number of text columns in the current display MODE.

PRINT#
Writes data to an opened file with multiple calls to BPUT. The current host does not implement OPEN, so the data written with PRINT# is ignored.

In contrast to other versions of BBC BASIC (Z80), BBC BASIC for the Spectrum, being a converson of BBC BASIC for the Acorn Z80 CoProcessor using BBC file I/O, reads and writes data files in the same format as 6502 and ARM BASIC.

PTR#
A pseudo-variable allowing the random-access pointer of the file whose file handle is its argument to be read and changed. The current host does not implement OPEN, so writing PTR# is ignored and reading PTR# returns no data.

SOUND
Generates sounds according to the paramters as follows:

```
SOUND channel,loudness,pitch,duration
```

channel
Bits 0 & 1 are the channel number (1-3). If bit 4 is set, the sound queue is flushed and the new sound is started immediately.

loudness
Values from -15 to -1 select a sound of amplitude 15 to 1 respectively with constant pitch, zero selects silence and values from 1 to 15 select an envelope (see ENVELOPE statement).

pitch

This selects the initial pitch. Middle C is 53, and a semitone change in pitch is a change of 4.

duration
Values from 0 to 254 select the duration of the sound, in units of approximately 1/20 second. The value -1 causes an indefinite sound, which can be stopped only by issuing another SOUND statement with the "flush" bit set or by pressing the ESCape key.

The current Spectrum host does not implement SOUND.

TAB
A keyword available in PRINT or INPUT. TAB(X) outputs sufficient spaces to move to column X according to the value of COUNT. TAB(X,Y) will move the cursor on the screen to character cell X,Y (column X, row Y) if possible by sending CHR$31;CHR$x;CHR$y to the VDU.

TIME
A pseudo-variable which reads and sets the elapsed 100Hz time clock. TIME is dereived from the Spectrum's FRAMES system variable, which is a 50Hz clock, so TIME always has bit 0 clear.

TIME$
A 24 character long string pseudo-variable which reads and sets the system real-time clock. The format of the character string is:

```
Day,dd Mon yyyy.hh:mm:ss
```

Where:

```
Day  is the day of the week (Mon, Tue, etc).
dd   is the day of the month (01, 02, etc).
Mon  is the abbreviated month name (Jan, Feb ,etc).
yyyy is the year (2003, 2004, etc).
hh   is hours (00 to 23).
mm   is minutes (00 to 59).
ss   is seconds (00 to 59).
```

The current Spectrum host does not implement TIME$ and returns a null string on reading and ignores writing.

USR
Calls a machine code routine and returns a value.

VDU

Sends a list of numeric arguments to the VDU. A 16-bit value can be sent if the value is followed by a ';'. It is sent as a pair of characters, least significant byte first.

VPOS
Returns the vertical position of the text cursor. The top row is row 0 and the bottom row is one fewer than the number of text rows in the current display MODE.

22. The VDU system

Control Characters:

The following VDU codes are defined. Not all of them are currently implemented, but the host correctly parses the VDU stream for unrecognised VDU sequences.

VDU 0 Null.

VDU 1,n Send byte to printer. Currently ignored.

VDU 2 Enable printer. Currently ignored.

VDU 3 Disables printer. Currently ignored.

VDU 4 Causes text to be written at the text cursor. Currently ignored.

VDU 5 Causes text to be written at the graphics cursor. Currently ignored.

VDU 6 Enables VDU output. Cancels the effect of VDU 21.

VDU 7 Causes a "beep". Currently ignored.

VDU 8 Moves the text cursor left one character. If at (0,1), then Spectrum ROM bug prevents the cursor moving to (31,0). If at (0,0), the cursor does not move.

VDU 9 Moves the text cursor right one character. Bypasses the Spectrum ROM bug to ensure correct function.

VDU 10 Moves the text cursor down one line.

VDU 11 Moves the text cursor up one line. If on line 1, the Spectrum ROM bug prevents the cursor moving up and the cursor ends up at (0,1).

VDU 12 CLS: Clears the text window to the current text background colour and moves the text cursor to (0,0).

VDU 13 Moves the text cursor to the left-hand edge of the window, but does not move it vertically.

VDU 14 Enter paged mode. Currently unsupported.

VDU 15 Stop paging. Currently unsupported.

VDU 16 CLG: Clears the graphics window using the current background GCOL action and colour. The current host clears the whole screen.

VDU 17,n COLOUR n: Sets the text foreground, background or border colour.

VDU 18,a,c
        GCOL a,c: Sets the graphics colour and plot action. Ignored on the current host.

VDU 19,l,p,r,g,b
        Sets the logical to physical colour mapping. Currently ignored.

VDU 20 Sets text and graphics colours to their default values (background black, foreground white) and resets the palette.

VDU 21 Disable VDU output. All VDU commands except 6 are ignored.

VDU 22,n MODE n: Selects a new screen mode and resets all screen driver variables (colours, palette, windows, cursor positions, graphics origin etc.). The current host selects the standard Spectrum 32x24 screen for all modes.

VDU 23,n,r1,r2,r3,r4,r5,r6,r7,r8
        Program user-defined graphics characters, and various VDU functions. Currently only characters &90-&9F are redefinable and characters &20-&7E if CHARS is set to point to RAM.

VDU 24,leftx;bottomy;rightx;topy;
        Define graphics window. Currently unimplemented.

VDU 25,n,x;y;
        PLOT k,x,y: Performs a PLOT action. Currently unimplemented.

VDU 26 Reset text and graphics windows to their default positions (filling the whole screen), home text cursor, move graphics cursor to 0,0 and reset the graphics origin to 0,0.

VDU 27 Do nothing.

VDU 28,leftx,bottomy,rightx,topy
Set a text window. The text cursor is moved to the new home position. Currently unimplemented.

VDU 29,x;y;
Move the graphics origin to the specified coordinates. Currently unimplemented.

VDU 30 Home the text cursor, to the top-left hand corner of the text window.

VDU 31,x,y
TAB(x,y): Moves the cursor to the position (x,y) if within the text window. If the coordinates are invalid they are ignored.

VDU 127 Backspace the cursor by one position and delete the character there.

Printable Characters:

Characters 32-126 (&20-&7E) are the usual Spectrum characters pointed to by CHARS. Characters 128-143 (&80-&8F) are initially set to the Spectrum block graphics. Characters 144-159 (&90-&9F) are the UDCs pointed to by UDC, and are initially set to inverted forms of "A" to "P". Characters 160-255 are initially set to copies of &20-&7F, resulting in CHR$255 being the Spectrum copyright symbol.

23. Operating System Commands

The following commands are implemented by the host code that interfaces BBC BASIC to the Spectrum system.

*|	*| comment	Everything after the | is ignored.
*/	*/filename [parameters]	Abbreviation for *Run.
*d:	*drive:	Select d as current device/drive.
*BASIC	*BASIC [filename]	Enter BBC BASIC.
*CAT	*CAT [drive]	Catalogue the default or specified drive.
*DELETE	*DELETE file	Erase file.
*FX	*FX a[,b[,c]]	

*GO	*GO [addr [parameters]]	Call code at address, or enter MOS command prompt if no address.
*HELP	*HELP [words]	Display information on system.
*LOAD	*LOAD file [aaaa]	Load file to hex address aaaa. The address can be FFFFxxxx to specify screen memory.
*QUIT	*QUIT	Return to calling process, if supported.
*RUN	*RUN file [parameters]	Load and execute the specified file, passing it any parameters.
*SAVE	*SAVE file ssss eeee	Save RAM from hex address ssss to address eeee-1.
	*SAVE file ssss +llll	Save RAM from address ssss with length llll. The addresses can be FFFFxxxx to specify creen memory.

If a *command is not recognised, the host tries to load and run it from the current drive.

Filenames are [d:]filename. If d: is present it can be T: for Tape or 1: to 8: for microdrive 1 to 8. On reset, T: is selected if no Interface 1 is preset, otherwise 1: is selected.

Operating system commands may be abbreviated and/or entered in lower case.

A "star" command cannot contain variable names and must be the last item on a program line. To include a variable name use the OSCLI statement, e.g. to delete a file whose name is known only at run time:

```
OSCLI "DELETE "+filename$
```

24. Random access files

BBC BASIC supports both random access and the ability to modify (update) a previously written file. Random access is performed by a single pointer (PTR#chn) which can be positioned anywhere in the file. The pointer is automatically incremented after every read or write operation (using BGET#, BPUT#, INPUT# or PRINT#).

Examples:

```
100 REM Read a file backwards
110 fin=OPENIN(filename$):size=EXT#fin
120 FOR point=size-1 TO 0 STEP -1
130 PTR#fin=point : PRINT CHR$(BGET#fin);
140 NEXT : CLOSE #fin
```

```
100 REM Update a "record" in a random-access file
110 fin=OPENUP(filename$)
120 PTR#fin=record_number*record_length
130 PRINT #fin,new_data,new_data$
140 CLOSE #fin
```

25. Spectrum Host Environment

To be written.