Appendix A

SPECIAL FEATURES OF THE BBC MICROCOMPUTER

In this appendix we shall briefly review all the features particular to the BBC microcomputer which have not been specifically dealt with elsewhere.

THE KEYBOARD

The BBC microcomputer keyboard includes a full 96 character ASCII keyboard, including lower case letters, although one symbol (ASCII code 96) is used as the pound sign instead of the grave accent (`), which is the standard symbol. It has both CAPS LOCK and SHIFT LOCK keys, the former giving upper-case letters but the lower-case option on all the special symbol keys. This is the default option when the computer is switched on, because it is most appropriate for programming, where upper case must be used for all BASIC keywords. Both CAPS LOCK and SHIFT LOCK are toggle switches rather than press and release keys as on a normal typewriter. When either key is engaged, this is shown by an illuminated LED below the key. (N.B. When both LEDs are illuminated together, this is quite a separate function, indicating that computer output is suspended. This can happen either when listing to the screen in paged mode, or during output to a printer which has a full buffer.)

A number of the keys have the standard ASCII control code functions

```
CTRL   RETURN   ESCAPE   DELETE   TAB
```

The BREAK key is the most powerful key on the computer. Unless disabled, it will interrupt and destroy the operation of any program, no matter in what state, and will clear the computer of most hung states. It clears the program, although it can normally be recovered with the command OLD, and puts the computer back into Mode 7 restoring many defaults. An even more powerful <BREAK> is obtained when the CTRL key is depressed in conjunction with the BREAK key, and this combination cannot be easily disabled. <BREAK> has yet other effects in combination with other keys, in particular the SHIFT key; <SHIFT-BREAK> is used to auto-boot the disc or Econet filing system.

When <BREAK> is pressed in conjunction with any other key, such as <SHIFT> or <CTRL>, it is necessary to keep the other key pressed while the BREAK key is released first. The reason for this is that while <BREAK> is pressed, the microcomputer is completely disabled. When <BREAK> is released, the microcomputer reads the keyboard to see whether another key is being pressed, and acts according to which key it is, if any.

The five cursor control keys (the four arrow keys plus the COPY key) on the right hand side of the keyboard are used for screen editing. Pressing any one of the arrow keys invokes a second cursor, the copy cursor, which can then be moved anywhere on the screen (with wrap-round) by the arrow keys. Wherever the copy cursor is on the screen, the COPY key can be used to ‘pick up’ the character at that point and build it into the current line of input. Thus a program line with minor errors can easily be edited without complete re-entry, by copying the old line in this way, while making any necessary corrections by typing in or deletion as necessary.

OPERATING SYSTEM COMMANDS

It is important from the start to distinguish two types of commands that can be issued to the BBC computer. Most commands, such as PRINT, MODE, VDU and CLS, are BASIC commands and are handled by the BASIC interpreter.

There is another set of commands that is handled directly by the Machine Operating System (MOS for short, or just OS). All of these commands begin with a star, and this tells the BASIC interpreter that the command that follows is to be passed directly to the OS. If the OS does not recognize the command it is passed on to any paged ROMs present in the computer, starting at the right-hand end of the bank of ROMs. Most such commands are likely to be intended for the filing system ROM. If none of the ROMs claim the command, it is treated as an abbreviation of the *RUN command, and any file in the filing system having the same name as that following the star will be executed as a machine code program. To further emphasize the difference between BASIC and Operating System commands, note that the latter can be handled even if] the computer does not possess a BASIC ROM, or is currently operating in a different language, such as Wordwise or Pascal.

Two points should be noted about Operating System commands. First, because the command is passed on to the Operating System, no BASIC commands can follow an OS command on the same line.

Second, OS commands can be issued either in upper case, like BASIC commands, or lower case. This includes filenames used for any of the filing Systems; the name will appear in any listing in the form that was used when the file was created, but it will be recognized subsequently whether in upper or lower case.

THE USER-DEFINED FUNCTION KEYS

The most eye-catching keys on the keyboard are the set of 10 red keys along the top, marked f0 to f9. These are called the user-defined function keys, because they can be programmed by the user to carry out any function he wishes. When a function key is pressed, the contents programmed into the key are issued to the computer just as if they had been typed in at the keyboard. This can be very usefully employed to issue frequently used commands, or even that part of a command which does not alter. The command to program the key marked fn (where n=0 to 9) is

```
*KEY n <any text you wish to be issued> <RETURN>
```

For instance, you could set up key 0 to issue the word RENUMBER at a single keystroke, after which you could add the starting line number and increment and press <RETURN>. The command needed would be

```
*KEY 0 RENUMBE R <RETURN>
```

Suppose, however, that you are happy with the default values for RENUMBER. It still requires two keystrokes to complete the command because you have to press the <RETURN> key. This key cannot be directly programmed into the function key, because it is already used to terminate the programming line for the key. However, there is a way of including a <RETURN> key code, or any other control code, into the function keys. You must type the symbol ll (the one over the backslash at the top right of the keyboard), followed by the key whose control version is what you want. For <RETURN>, which is actually the same as CTRL-M, you therefore use \\M. Thus you could program key f0 to save you even having to press <RETURN> when renumbering, by

```
*KEY 0 RENUMBER |M <RETURN>
```

Other control codes could be printed the same way. For instance, to switch into paged mode, LIST, and switch paged mode off automatically so that you do not accidentally forget, you could issue

```
*KEY 1 |N LIST |M |O <RETURN>
```

Other commands frequently programmed into the function keys include RUN, LIST, OLD and mode changes. Although LIST, for example, has a two character abbreviation, programming a function key still reduces three keystrokes to one, counting the <RETURN> as well.

It is also possible to program the BREAK key to act as a function key (key number 10, which is logical, lying next to f9 as it does), although this does not alter its normal BREAK function which is carried out before issuing the programmed responses. It is quite common to use this feature to prevent users breaking out of a program, by programming the BREAK key to rerun the program with the following definition

```
*KEY 10 OLD |M RUN |M
```

Finally, if you need so many, the five cursor keys can be disabled and treated instead as yet more function keys, numbered 11 to 15, by the command

```
*FX 4,2
```

THE *TV COMMAND

The *TV command has two parameters and two purposes. The first use is to move the display up or down the screen. The default is *TV 0, but you can move the display up n lines by

```
*TV n
```

(there is a second parameter but this can be omitted unless you also want to switch off the interlace), or down n lines by

```
*TV 256-n
```

The command would be most useful to move the picture up if the current line of input is off the bottom of the screen. Note that you will not see the effect until after a mode change. Also, startling effects will be obtained if too large a value of n is used.

The second parameter to *TV determines whether the interlace is turned on or not. Interlace is the effect whereby a TV or monitor picture is built up of two interlaced halves containing alternate lines of the 625 line display. Although necessary on a TV set to display the complete picture, when displaying computer output it can give an unpleasant slight flickering effect. You can turn it off by the command

```
*TV 0,1
```

The two halves of the display will then be superimposed, giving a rock-steady picture. Note that the effect again only comes into operation after the next mode change (which could of course be into the same mode as before), and that interlace cannot be turned off in this way in Mode 7 (but see Section 6.4).

SOUND

Like graphics, the sound facilities on the BBC computer are very versatile, but the complex uses of sound, particularly the ENVELOPE command, are beyond the scope of this book.

The SOUND command by itself can be used to produce pure tones of any frequency, volume and length, and to mix up to three tones into chords. The command has the from

```
SOUND <channel>,<amplitude>,<pitch>,<duration>
```

We can briefly consider each of the parameters to SOUND in turn.

1. The first parameter, <channel>, can take four possible values, 0 to 3, but channel 0 is a special ‘noise’ channel. If a series of SOUND commands for different channels are issued in succession, this will produce the effect of chords.
2. The second parameter, <amplitude>, must be a number between 0 (zero amplitude) and − 15 (maximum loudness).
3. The third parameter, <pitch>, determines the pitch, or frequency, of the tone. It is a number from 0 to 255, each step corresponding to a change in pitch of a quarter of a semitone. Since there are 12 semitones to an octave, an increase in the pitch number of 48 gives a one octave change in pitch. Middle C has a pitch value of 53, so C in the octave below is 5, and in the octave above is 101.
4. The final parameter, <duration>, is a number which can take a value from 1 to 255, and is in units of twentieths of a second.

Try the following sequence of SOUND commands

```
FOR J=1 TO 8: SOUND 1,-15,57+4*J,6: NEXT J
```

VDU COMMANDS

VDU commands are so called because they all govern display (on the VDU) in some way or other. Many are related to graphics and are dealt with in Chapter 6, but others are of more general application, and these are described below. All the VDU commands are summarized in Appendix C.

VDU 1 can be used to send a character to the printer but not to the screen. This is useful where a special code needs to be sent to a printer to switch on underlining or perform other functions of a similar type.

VDU 2 and VDU 3 are used before sending output to a printer and to terminate the output respectively. In immediate mode, the control-codes <CTRL-B> and <CTRL-C> are more convenient.

VDU 14 (CTRL-N) is very useful when listing programs or trying to examine large amounts of program output on the screen. About twenty lines of output at a time are displayed, then the computer pauses until the SHIFT key is pressed (no other key will do). Note that it remains in effect even when a program is subsequently running, and it can cause the computer to appear to ‘hang’, without reason. VDU 15 (CTRL-O) cancels the effect of VDU 14, but does not come into effect until the current listing or program run is ended.

VDU 21 (CTRL-U) has two different effects. If issued from the keyboard, it deletes the whole of the current line. From within a program, it has the effect of stopping all subsequent output (both text and graphics) from reaching the screen. This effect is cancelled by VDU 6.

Appendix B

SELECTED *FX COMMANDS

FX is short for ‘effects’, and is the command used for a range of special facilities or effects on the BBC microcomputer. The *FX commands can also be invoked by a call to OSBYTE, address &FFF4, with the *FX number in the accumulator (or A %). Certain calls return ‘answers’ in the X and Y index registers, and OSBYTE calls are the only way to use these particular effects. A very full list of the *FX calls is given in the Advanced User Guide by Bray, Dickens and Holmes, published by the Cambridge Microcomputer Centre. Below is given a selection of the *FX calls which are most useful to the BASIC programmer.

***FX 0**

Reports the version number of the operating system. (At the time of publication, the current operating system is 1.2, with earlier versions being 0.1 and (briefly) 1.0.)

***FX2,n**

Selects the source of input, which can be either the keyboard or the RS423 port.

n=0 selects the keyboard (default)
n=1 selects the RS423 port
n=2 selects both keyboard and RS423 port

***FX3,n**

Selects which output streams receive future output. The first three bits of I! control: the RS423 output (bit 0, set to enable output), the screen (bit 1, clear to enable output) and the printer port (bit 2, clear to enable Output). Thus *FX3,1 enables all three streams, *FX3,6 disables all streams. The default is n=0 (screen and printer enabled). The printer must still be selected by VDU 2, (or CTRL-B) but setting bit 3 will both enable and select the printer regardless of whether a VDU 2 has been issued.

*FX4,n

Controls the action of the cursor control keys and the COPY key. n=0 gives normal editing (default), n=1 makes the keys act as normal keys with ASCII codes 135 to 139, while n=2 makes them act as extra programmable function keys, 11 to 15 (see Section 11.2.2 for further details).

*FX5,n

Selects the printer type. n=1 selects the parallel printer port (default), n=2 selects the RS423 port, n=4 selects the Econet output. See Appendix K also.

*FX6,n

Selects the character (ASCII code n) to be suppressed for printer output. See Appendix K for the use of this call.

*FX7,n and *FX8,n

These calls set the baud rate for the RS 423 port for input (*FX7) and output (*FX8) according to the following table

n=1	75 baud
n=2	150 baud
n=3	300 baud
n=4	1200 baud
n=5	2400 baud
n=6	4800 baud
n=7	9600 baud
n=8	19200 baud

*FX9,n and *FX10,n

Set the duration of the first (*FX9) and second (*FX10) flashing colours. n is the duration in fiftieths of a second (default n=25).

*FX11,n

Sets the delay time before keys start to auto-repeat. n is the delay in centiseconds, n=0 disables auto-repeat (default n=32).

*FX12,n

Sets the auto-repeat period to n centiseconds. A small value of n thus gives rapid repetition. n=0 restores the default value (8 centiseconds) and also the default for the auto-repeat delay.

*FX15,n and *FX21,n

These calls flush, or empty, various internal buffers. *FX15,0 flushes all buffers, and *FX15,1 flushes the currently selected input buffer (see *FX2), which will usually be the keyboard buffer. *FX21 flushes specified buffers, as follows

n=0	keyboard buffer (will usually be equivalent to *FX15,1)
n=1	RS423 input buffer
n=2	RS423 output buffer
n=3	printer output buffer
n=4-7	flush SOUND channels 0 to 3 respectively
n=8	speech synthesis buffer

*FX16,n

Enables n of the ADC channels. (n=0 to 4. n=0 disables all channels.)

*FX17,n

Forces the start of a conversion on ADC channel n.

*FX18

Cancels any strings stored in the user-defined function keys.

*FX20

Used to ‘explode’ or expand the number of user-defined characters. See Section 6.4.3.

*FX119

Closes any files open for *SPOOL output or *EXEC input.

OSBYTE call 121 (&79)

This *FX must be made as an OSBYTE call, because a value is returned in the X register (see Section 10.8.4). If called with X=0, it scans the keyboard and returns the value of any key pressed, or 255 (&FF) if no key is pressed. Unlike INKEY(-X) it can be used not just to detect a particular key, but Whether any key is pressed, and which it is. The number of the key pressed, which is returned in the X register, is related to the INKEY number, being ABS(INKEY number)-1. For example, SHIFT is 0, SHIFT LOCK is 80, → is 121.

OSBYTE call 131 (&83)

This OSBYTE call returns the value of the OSHWM (the OSHWM is the default value of PAGE − see Section 10.3) in the X and Y registers (low byte in X, high byte in Y).

OSBYTE call 132 (&84)

Returns the address of the bottom of the section of RAM being used for screen display (default value of HIMEM).

OSBYTE call 135 (&87)

Returns in the X register the character value of the character at the text cursor position, and in the Y register the graphics MODE number.

***FX138,m,n**

Inserts character with ASCII code n into the input buffer specified by m (m=0 for the keyboard buffer, and m=1 for the RS423 input buffer). See Section 10.7 .2 for the way in which *FX138 can be used.

***FX200,n**

This affects the action of <ESCAPE> and <:BREAK>. If bit 0 is set to I it will disable <ESCAPE:>, and if bit 1 is set to I it will clear memory on <BREAK>. Thus *FX200,3 will prevent users from breaking out of a program with <ESCAPE>, and stop them getting back to the program with OLD after a <BREAK>.

***FX201,n**

*FX201,1 will disable the keyboard, *FX201,0 restores normal keyboard action.

***FX202,n**

As an OSBYTE call, this allows you to read (from the X register) the status of the keyboard locks etc, according to the following list, but is probably most useful as a means of controlling the SHIFT LOCK and CAPS LOCK from within a program, to force user input into the case desired for example. Bit 4 controls CAPS LOCK (cleared to 0 to set the lock) and bit 5 controls SHIFT LOCK (again 0 sets the lock). Thus *FX202,32 will switch on the CAPS LOCK. The full effect of all bits is

bit 3	returns 1 if SHIFT pressed
bit 4	1 for no CAPS LOCK
bit 5	1 for no SHIFT LOCK
bit 6	returns 1 if CTRL pressed
bit 7	1 to enable SHIFT

SHIFT enable allows the SHIFT key to work in reverse when a lock is in operation, so that you can type in lower case without disengaging CAPS LOCK or SHIFT LOCK.

***FX212,n**

Sets the amplitude for the BELL sound. The value of n should be set to 8>< (<amplitude value>+31) where <amplitude value> is the usual number 0 to -15 as used in the SOUND command. The default value for n is 144.

*FX213,n

Sets the pitch of the BELL sound. n is the normal pitch parameter of the SOUND command (default value 101).

*FX214,n

Sets the duration of the BELL. n is the usual duration parameter of the SOUND command (default value 7).

*FX219,n

Sets the ASCII code returned by the TAB key to be n. The default value is 9 (move cursor right) but it can be set to any value, including any control-code or soft key code. Thus *FX219,12 would turn the TAB key into a screen clear key.

*FX225-228,n

These calls control the effect of the function keys in various combinations, as follows

`*FX225,n`	function keys alone (default n=1)
`*FX226,n`	SHIFT+function keys (default n= 128)
`*FX227,n`	CTRL+function keys (default n=144)
`*FX228,n`	CTRL-SHIFT+function keys (default n=0)
n=0	makes the key combinations have no effect
n=1	makes them act as function keys
n>1	n acts as a base number, such that the key produces the ASCII code n+function key number

The default values for *FX226 and *FX227 are chosen to enable teletext colour and graphics codes to be produced readily.

*FX229,n

n=1	causes the ESCAPE key to generate its ASCII code (27) instead of the normal ESCAPE action
n=0	restores the normal ESCAPE action

Appendix C

SUMMARY OF THE VDU COMMANDS

In the following list, items marked with an asterisk (*) are 'standard' effects of those particular control-codes, such as the <RETURN> character, CTRL-M. Items marked with a plus sign (+) are equivalents of BASIC commands.

VDU number	*Control control required*	*Extra bytes*	*Action*
0	@	0	None
1	A	1	Next character not sent to the screen (used to send output to the printer only)
2	B	0	Future output sent to the printer
3	C	0	Stop output to the printer
4	D	0	Cancel the effect of VDU 5
5	E	0	Future text written at graphics cursor position
6	F	0	Cancel the effect of VDU 21
*7	G	0	Create a short beep
*8	H	0	Backspace (move cursor left one space)
*9	I	0	Forwardspace (move cursor right one space)
*10	J	0	Linefeed (move cursor down one line)
*11	K	0	Reverse linefeed (move cursor up one line)
+*12	L	0	Clear text screen (new page, CLS)
*13	M	0	<RETURN> (carriage return)
14	N	0	Switch on page mode for output
15	O	0	Switch off page mode
+16	P	0	Clear graphics screen (CLG)
+17	Q	1	COLOUR command

VDU number	*Control character*	*Extra bytes required*	*Action*
+18	R	2	GCOL command
19	S	5	Reassign logical colour
20	T	0	Cancel all VDU 19 assignments
21	U	0	Immediate mode: cancel line Program mode: switch off future output
22	V	1	Changes display mode but does *not* reset HIMEM
23	W	9	Specify user-defined character
24	X	5	Set graphics window
+25	Y	5	PLOT command
26	Z	0	Cancel VDU 24 and VDU 28 and home text and graphics cursors
*27	[	0	ESCAPE character. When issued by a program, does nothing
28	\	4	Set text window
29	]	0	Set a new graphics origin
3	0	^	0Home text cursor to top left of screen
31	_	0	TAB(X,Y)
+127		0	DELETE character

Appendix D

BUGS AND UNEXPECTED EFFECTS IN BBC BASIC

Inevitably there are some bugs in the BBC microcomputer software. A comprehensive list is published by the MEP Software Unit. The MEP list includes bugs in the assembler, network filing system and quirks as well as bugs. In this appendix the most serious bugs in those parts of the system covered by this book are reported.

1 Renumbering on BREAK and OLD

When BREAK is pressed, the first byte of the program in memory is set to &FF to clear the program. This byte normally represents the high byte of the first line number. When the OLD command is issued, it simply sets the first byte of the program to zero to remove the end of program marker. If the line number was originally greater than 255, it will be reduced to the low byte of the number − e.g. 300 will become 44, 1000 will become 232.

2 DIM statement

The BASIC interpreter should trap DIM statements which require more memory than is available, and report a ‘Bad DIM’ error. Occasionally the interpreter misses the error; this occurs when at least the lower six bits of every dimension are set − e.g. DIM A(63,127 ,255). Other dimensions such as DIM A(64,126,255) will be trapped correctly. The result can be disastrous, because the interpreter clears all the memory set aside for the array, and may even overwrite crucial areas of memory used by the system. The only solution then will be to switch off the computer.

3 DATA statement

A DATA statement must be the first statement on a line, or it will be ignored. This is not perhaps a bug, but certainly a limitation of which you should be aware.

More seriously, if the special BBC feature of adding a line number to RESTORE is used to set the DATA pointer to that line instead of back to the very start of the program, data will be read from that line even if it is not a

DATA statement. In this case, data is assumed to start after the first comma on the line.

4 RENUMBER command

If, on RENUMBER, line numbers would exceed 32767, they are treated MOD 32768, that is they ‘wrap round’ instead of generating an error.

5 Undefined variables

A useful safety feature of BBC BASIC is that variables must be assigned a value before they can be used. There is an exception to this when the variable appears on both sides of an assignment statement. The initial value of the variable is taken as zero, so the statement

```
J=J+10
```

will result in J taking the value 10 if it has not previously been assigned a value.

6 INKEY statement

If INKEY(X) is used with a value of X greater than 255, then the first time that the statement is executed, the effective value of X is reduced by 256. The solution is to issue a previous INKEY with an argument of 1.

7 CLEAR statement

Users should be aware that the CLEAR statement does not merely clear variables, it also empties the BASIC stack and clears the data pointer and LOMEM. The BASIC stack contains addresses of the start of loops and return addresses of subroutines, procedures and functions, so CLEAR should never be issued from within a FOR or REPEAT loop or sub-program.

8 Variable names

Variable names can include underline symbols and pound signs as well as letters and numbers. The former is particularly useful to give an impression of a space, as in New_X.

9 Double-height characters

When using double-height characters in Mode 7, the line immediately below a line containing the double-height character 141 becomes a special line. Characters following another character 141 generate lower halves of the characters, thus enabling the double-height characters to be completed, but ‘normal’ characters, before the 141 or after a character 140, do not show up. This is not really a bug, but can be a restriction in certain circumstances. A consequent effect is that, when the upper of a pair of double height character lines scrolls off the top of the screen, the second line acts as an upper half and suppresses normal characters on the next line.

10 Incorrect catalogue

When discs are in use, the latest catalogue information is stored in the computer on pages 14 and 15 (&E and &F). Under certain circumstances, particularly if the disc drive has not stopped spinning since the last disc access, the DFS assumes that it still knows what the disc contains and does not read the catalogue again. If you have swapped discs in the meantime, the catalogue information would be incorrect, and any writing process using the incorrect catalogue could be a disaster.

Try loading a program, and before the disc drive has stopped spinning, open the drive door and remove the disc. The drive will keep running, and while it is empty type *CAT. The old catalogue list will appear even though the drive is empty.

The key point to remember is always to let the drive stop spinning before issuing any command that writes to a disc.

11 Limitations on *SPOOL

Text produced by the DFS cannot be written into an ASCII file.

The following bugs occur in BASIC I, but have been corrected in BASIC II.

12 ABS function

In BASIC I, ABS may not be preceded by the unary minus operator (i.e. a minus sign to invert the sign of the function, rather than to subtract it from another value). Thus

```
PRINT -ABS(-1)
```

is not allowed, although

```
    PRINT 3-ABS (-1)
```

is permissible.

13 COUNT function

In BASIC I, the value returned by COUNT is not zeroed after a mode change, so that after a mode change in the middle of a line of output COUNT would include the old line in its total.

14 ON...ELSE

The structure of combining ELSE with ON...GOTO and ON...GOSUB is not allowed within procedures and functions in BASIC I.

15 EVAL function

In BASIC I, the EVAL function will not correctly evaluate pseudo-variables. Instead it treats them as ordinary variables, and reports ‘no such variable’.

16 INSTR command

In BASIC I, this command contains a serious bug whereby the stack is corrupted if the search string is larger than the original string. This bug is explained more fully in Section 4.4.

17 ON ERROR trap

Certain line numbers cannot be used with ON ERROR GOTO in BASIC I. Line numbers such as 9999 cause the machine to hang up completely if an error occurs.

Appendix E

DIFFERENCES BETWEEN BASIC I AND BASIC II

The second release of BASIC on the BBC microcomputer has a number of improvements over the original version, and these are listed below. In addition, a number of the bugs have been corrected. These bugs are described in Appendix D, and are simply listed below in brief.

The first release of BASIC is now commonly referred to as BASIC I, and the revision is called BASIC II. The easiest way to determine which version of BASIC you have is to press <BREAK>, and then type the command REPORT. The computer replies with a copyright message, which includes the dates 1981 and 1982 for BASIC I and BASIC II respectively. The following bugs have been corrected in BASIC II

- ABS can now be used with a preceding minus sign.
- COUNT is now properly reset on a mode change.
- ELSE may now be used with ON...GOTO and ON...GOSUB within procedures and functions.
- EVAL will now handle pseudo-variables.
- INSTR can now handle a search string longer than the source string.
- ON ERROR will now handle all line numbers correctly.

The following improvements have been made in BASIC II

INPUT

This will now accept the semicolon as a separator for all items, as well as the comma. This brings it into line with other implementations of BASIC, where the command has the form

```
INPUT "Leading string";A,B,C
```

OPENUP

The command OPENUP is introduced for opening data files for reading and writing (i.e. updating), and OPENIN is reserved for reading only. This feature is fully discussed in Section 9.3.1.

OSCLI

The command OSCLI followed by a string or string variable passes the string to the command line interpreter. In BASIC I it was necessary to load the string into memory, set a pointer to the string via X% and Y% and CALL &FFF7. Use of the CLI is discussed in detail in Section 10.7.1.

LN, LOG, SIN, COS

These functions have all been increased in accuracy. The effect will be most noticeable near the limits of range of the functions.

PRINT (and STR$)

Numbers are printed with slightly greater precision.

EQUB, EQUW, EQUD, EQUS

Four new assembler pseudo-operations are provided, to transfer values to bytes within machine code. The operations respectively define a single byte, word (double-byte), double-word (four bytes) and string. The instructions are used in statements such as

```
.values EQUB &0D
 EQUW &1234
 EQUD &01020304
 EQUS "END OF MESSAGE"
```

which store the specified values in memory starting at the address specified by the label ‘values’.

Appendix F

THE RESIDENT VARIABLE @%

The detailed features of the layout of numbers output by the PRINT command are governed by the resident integer variable @%. Like all integer variables it is a four byte number, and each byte serves a separate function. It is most conveniently specified in hexadecimal (that is, with a leading &) since then each pair of digits represents one byte. If the four bytes have values B4, B3, B2 and B1, then for the number

```
@%=&12345678
```

we would have B4=&12, B3=&34, B2=&56 and B1=&78.

Alternatively, if B1 and B4 are treated as variables with separate values, then @% can be denoted by

```
@%=B1+&100*(B2+&100*(B3+&100*B4))
```

or

```
@%=B1+256*(B2+256*(B3+256*B4))
```

which will be the best way to set up @% if you want to vary it from time to time.

B4

B4 has a rather marginal effect − it determines whether the format of a string created by STR$ is controlled by @)%. If B4=01, @% is used in determining the format, if B4=00, @J% is ignored. The default value for B4 is zero and this is why, as pointed out in Section 4.4.2, STR$ may not always produce quite what you would expect.

B3

B3 selects from 3 types of format as follows

B3=00	General format (or G format)
B3=01	Exponent format (or E format)
B3=02	Fixed format (or F format)

G format is the default state, in which a number is printed as a straightforward fixed point number (for example 123.45) if it is neither too large for the specified range, nor less than 0.1. Outside these limits it is printed in exponent form or scientific notation as a floating point number, with a mantissa (a fixed point number between I and 10) and an exponent or power of 10. Thus 0.0123 would be written as 1.23E-2 and 12 milliard would be written as 1.2E10.

In E format, numbers are always written in exponent form.

F format is the most interesting. All numbers are printed with a fixed number of decimal places (determined by B2) unless they exceed the value which can be stored internally as an exact number (more than nine figures) in which case they revert to exponent form. Numbers which are too small, however, are printed as zero. F format is designed for, and is ideal for, tabulated columns of numbers, so that, combined with right justification, decimal points will be aligned.

B2

B2 is the other factor affecting the format of printed numbers. It plays different roles according to the value of B3.

In G format, it specifies the maximum number of significant figures to be printed, and hence also the largest number which can be printed in fixed point, before reverting to exponent form. The default value is 9, which is one figure less than the machine accuracy, so that rounding errors are normally suppressed.

In exponent form numbers, B2 again determines the maximum number of significant figures of the mantissa. Including a possible leading minus sign, the decimal point, the E, up to two digits for the power of 10 and another possible minus sign for the power, exponent form numbers in G format occupy up to B2+6 characters in all. One difference occurs for E format numbers, in that the power occupies three columns, including trailing spaces, and B2 figures are always printed in the mantissa, so numbers always occupy B2+5 or B2+6 columns, depending on whether there is a minus sign.

In F format, B2 plays a quite different role, specifying the number of digits to follow the decimal point. The maximum number of significant figures is always set to nine, being governed by the machine accuracy, so that numbers do not revert to exponent form below 109. Even integers are printed With a decimal point and B2 trailing zeros. Note, however, that if the number of figures printed would exceed nine, the display reverts to G format. For example, with B2=4, 123456 would be printed as an integer, whereas with B2=3 it would be printed as 123456.000.

The allowed range of B2 is as follows

G format 1 to 9
E format 1 to 9
F format 0 to 9

For numbers outside these limits, B2 reverts to the default of 9.

As an example of the use of B2, Table F .1 shows how the number 12.3456 would be printed in each of the formats, for values of B2 of 3, 6 and 9.

Table F.1 Effect of B2 on the format of numbers.

	G format	*E format*	*F format*
B2=9	12.3456	1.23456000E1	12.3456
B2=6	12.3456	1.23456E1	12.345600
B2=3	12.3	1.23E1	12.346

Note that with B2=9, the F format display has reverted to G format.

B1

As we saw earlier, B1 sets the print field width. This is not the maximum number of characters that will be printed, but is simply the width between each 'tab position' as determined by commas in PRINT statements. It does, however, limit the number of characters that will be printed right justified, and hence with the decimal points aligned. Moreover, right justification will only occur if items are separated by commas, so in principle this form of layout would have to be in columns of equal width. In practice, the TAB(X, Y) form overrides the left margin origin from which B1 measures, so it could be used to set out columns of figures of different widths (but not less than B1). The range of B1 is any value from 0 to 255. However, the practical upper limit is probably more like 20.

Example F.1

The following program generates numbers whose magnitudes and number of figures, as well as values, are random, and prints them in successive columns of width 10, 13, 17, 12, 20 in Mode 0, with respectively 0, 3, 4, 2, 5 decimal places, properly aligned.

```
10 MODE 0
20 DIM A(15)
30 CLS
40 FOR ROW=1 TO 20
50 FOR COL=1 TO 5
60 REPEAT
70 A(COL)=FN_rannum
80 UNTIL ABS(A(COL))<1E8
```

```
   90 NEXT COL
  100 @%=&2000A: PRINT TAB(0,ROW) A(1)
  110 @%=&2030D: PRINT TAB(10,ROW) A(2)
  120 @%=&20411: PRINT TAB(23,ROW) A(3)
  130 @%=&2020C: PRINT TAB(40,ROW) A(4)
  140 @%=&20514: PRINT TAB(52,ROW) A(5)
  150 NEXT ROW
  160 END
10000 DEF FN_rannum
10010 LOCAL DECPLACES, NUM
10020 DECPLACES=10+(RND(6)-1)
10030 NUM=INT((RND(1)-0.5)*10+RND(9))
10040 =NUM/DECPLACES
```

Note that the PRINT statements must not end with semicolons. The TAB(X, Y) form forces all the output for one row to appear on the same line. This cannot, of course, be used in conjunction with a printer.

Note also that the formatting still goes wrong occasionally for very large numbers, especially in columns 3 and 5 where several decimal points are required. This can be prevented by restricting the size of the numbers to less than 10^6 (alter the last RND on line 1030 to RND(5)). It is of course a very demanding test to format numbers of such varying type, but it does serve to demonstrate that although the @% variable provides great flexibility, it is difficult to use.

One final property of B1 may sometimes be useful. Provided it is set to at least 2, numbers separated by commas will be printed with at least one space between them. Thus if a series of variables is to be printed where alignment of columns is immaterial, but where the normal large gaps are unnecessary, setting B1 to 2 will be much more convenient than separating each variable by " ".

Appendix G

PSEUDO-VARIABLES IN BBC BASIC

The following are BBC BASIC pseudo-variables. They always have a value, and may be altered by the system as well as by the user, but in other respects they can be handled just like ordinary variables.

`HIMEM`	Pointer to top of available RAM
`LOMEM`	Pointer to the bottom of variable storage space
`PAGE`	Pointer to the start of program storage (must be a multiple of 256)
`PTR#`	Pointer to position in a data file
`TIME`	An internal timer reading in centiseconds

The following are strictly functions, but could be thought of as read-only pseudo-variables.

`COUNT`	Counts the number of characters printed since the last <RETURN>
`EOF#`	Boolean variable which becomes true when the end of a data file is reached
`ERL`	Line number where the last error occurred
`ERR`	Error number of the last error (see Appendix J)
`EXT#`	Current size of data file
`FALSE`	The Boolean value ‘false’ (returns a numerical value of 0)
`PI`	The mathematical quantity π, with value 3.14159265
`POS`	Horizontal position of the cursor (left hand side=0) Pointer to the top of a program
`TRUE`	The Boolean value ‘true’ (returns a numerical value of -1)
`VPOS`	Vertical position of the cursor (top=0)

Appendix H

ASCII CODES

Character	*Decimal*	*Hex*
CTRL-@	0	00
CTRL-A	1	01
CTRL-B	2	02
CTRL-C	3	03
CTRL-D	4	04
CTRL-E	5	05
CTRL-F	6	06
CTRL-G	7	07
CTRL-H	8	08
CTRL-I or TAB	9	09
CTRL-J	10	0A
CTRL-K	11	0B
CTRL-L	12	0C
CTRL-M or RETURN	13	0D
CTRL-N	14	0E
CTRL-O	15	0F
CTRL-P	16	10
CTRL-Q	17	11
CTRL-R	18	12
CTRL-S	19	13
CTRL-T	20	14
CTRL-U	21	15
CTRL-V	22	16
CTRL-W	23	17
CTRL-X	24	18
CTRL-Y	25	19
CTRL-Z	26	1A
CTRL-[	27	1B
CTRL-\	28	1C
CTRL-]	29	1D
CTRL-^	30	1E
CTRL-_	31	1F

Character	*Decimal*	*Hex*
SPACE	32	20
!	33	21
"	34	22
#	35	23
$	36	24
%	37	25
&	38	26
'	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F

Character	*Decimal*	*Hex*	*Character*	*Decimal*	*Hex*
@	64	40	£	96	60
A	65	41	a	97	61
B	66	42	b	98	62
C	67	43	c	99	63
D	68	44	d	100	64
E	69	45	e	101	65
F	70	46	f	102	66
G	71	47	g	103	67
H	72	48	h	104	68
I	73	49	i	105	69
J	74	4A	j	106	6A
K	75	4B	k	107	6B
L	76	4C	l	108	6C
M	77	4D	m	109	6D
N	78	4E	n	110	6E
O	79	4F	o	111	6F
P	80	50	p	112	70
Q	81	51	q	113	71
R	82	52	r	114	72
S	83	53	s	115	73
T	84	54	t	116	74
U	85	55	u	117	75
V	86	56	v	118	76
W	87	57	w	119	77
X	88	58	x	120	78
Y	89	59	y	121	79
Z	90	5A	z	122	7A
[	91	5B	{	123	7B
\	92	5C	|	124	7C
]	93	5D	}	125	7D
ˆ	94	5E	˜	126	7E
_	95	5F	DELETE	127	7F

Appendix I

INKEY CODES

When the keyboard is examined directly by using INKEY with a negative argument, the code for the key has no relationship to the ASCII code of the character, but is an internal code related to the structure of the keyboard. A closely related table of codes is used by OSBYTE call 121.

INKEY code	*OSBYTE code*	*symbol*
-1	0	<SHIFT>
-2	1	<CTRL>
-17	16	Q
-18	17	3
-19	18	4
-20	19	5
-21	20	f4
-22	21	8
-23	22	f7
-24	23	-
-25	24	ˆ
-26	25	←
-33	32	f0
-34	33	W
-35	34	E
-36	35	T
-37	36	7
-38	37	9
-39	38	I
-40	39	0
-41	40	_
-42	41	¯
-49	48	1
-50	49	2
-51	50	D
-52	51	R
-53	52	6
-54	53	U
-55	54	O

INKEY code	*OSBYTE code*	*symbol*
-56	55	P
-57	56	[
-58	57	↑
-65	64	<CAPS LOCK>
-66	65	A
-67	66	X
-68	67	F
-69	68	Y
-70	69	J
-71	70	K
-72	71	@
-73	72	:
-74	73	<RETURN >
-81	80	<SHIFT LOCK>
-82	81	S
-83	82	C
-84	83	G
-85	84	H
-86	85	N
-87	86	L
-88	87	;
-89	88	]
-90	89	<DELETE>
-97	96	<TAB>
-98	97	Z
-99	98	<SPACE>
-100	99	V
-101	100	B
-102	101	M

INKEY code	*OSBYTE code*	*symbol*
-103	102	,
-104	103	.
-105	104	/
-106	105	<COPY>
-113	112	<ESCAPE>
-114	113	f1
-115	114	f2

INKEY code	*OSBYTE code*	*symbol*
-116	115	f3
-117	116	f5
-118	117	f6
-119	118	f8
-120	119	f9
-121	120	\
-122	121	→

Appendix J

ERROR CODES

Code	*Error*
0	**No room** (cannot be trapped)
0	**STOP** (BASIC II only)
1	**Out of range** (assembly language error)
2	**Byte** (assembly language error)
3	**Index (**assembly language error)
4	**Mistake**
5	**Missing ,**
6	**Type mismatch**
7	**No FN**
8	**$ range** (Trying to address locations less than 256 (&100) with the $ indirection operator
9	**Missing "**
10	**Bad DIM**
11	**DIM spa ce**
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**
25	**Bad MODE**
26	**No such variable**
27	**Missing)**
28	**Bad hex**
29	**No such FN/PROC**
30	**Bad call**

Code	*Error*
31	**Arguments**
32	**No FOR**
33	**Can't match FOR**
34	**FOR variable**
35	**Too many FORs** (only 10 FORs may be nested)
36	**No TO**
37	**Too many G0SUBs** (only 26 nested subroutines allowed)
38	**No GOSUB**
39	**ON syntax**
40	**ON range**
41	**No such line**
42	**Out of DATA**
43	**No REPEAT**
44	**Too many REPEATs (only 20 REPEATs may be nested)**
45	**Missing # (BASIC II only)**
216	**Data?** (cassette filing system error)
217	**Header?** (cassette filing system error)
218	**Block?** (cassette filing system error)
219	**File?** (cassette filing system error)
220	**Syntax** (filing system error)
222	**Channel** (filing system error)
223	**EOF** (filing system error)
251	**Bad key**
253	**Bad string**
254	**Bad command**

The following errors have no error code

Bad program (cannot be trapped)
Failed at <line number> (error during RENUMBERing)
LINE space (no room in memory to insert line)
Silly (occurs with AUTO if a step size greater than 255 is specified)