The 6502 Second Processor User Guide

Part no 408000
Issue no 2
Date March 1984

WARNING: THE SECOND PROCESSOR MUST BE EARTHED

Important: The wires in the mains lead for the 6502 Second Processor are coloured in accordance with the following code:

Green and Yellow	**Earth**
Blue	**Neutral**
Brown	**Live**

As the colours of these wires may not correspond to the coloured markings identifying the terminals in your plug, proceed as follows:

The wire which is coloured green and yellow must be connected to the terminal in the plug which is marked by the letter E, or by the safety earth symbol or coloured green, or green and yellow.

The wire which is coloured blue must be connected to the terminal which is marked with the letter N, or coloured black.

The wire which is coloured brown must be connected to the terminal which is marked with the letter L, or coloured red.

If the socket outlet available is not suitable for the plug supplied, the plug should be cut off and the appropriate plug fitted and wire as previously noted. The moulded plug which was cut off must be disposed of as it would be a potential shock hazard if it were to be plugged in with the cut off end of the mains cord exposed. The moulded plug must be used with the fuse and fuse carrier firmly n place. The fuse carrier is of the same basic colour* as the coloured insert in the base of the plug. Different manufacturers' plugs and fuse carriers are not interchangeable. In the event of loss of the fuse carrier, the moulded plug MUST NOT be used. Either replace the moulded plug with another conventional plug wired as previously described, or obtain a replacement fuse carrier from an authorised BBC Microcomputer dealer. In the even of the fuse blowing it should be replaced, after clearing any faults, with a 3 amp fuse that is ASTA approved to BS1362.

*Not necessarily the same shade of that colour.

Within the publication the term BBC is used as an abbreviation for British Broadcasting Corporation.

Exposure

Like most electronic equipment, the 6502 Second Processor should not be exposed to direct sunlight or moisture for long periods.

Contents

About this User Guide

This User Guide is divided into two parts. Part 1 describes the 6502 Second Processor, and Part 2 explains the extended facilities of the BASIC language ROM (‘Hi-BASIC’) supplied with the 6502 Second Processor. Similarly, Appendix B explains the differences between the disc and Econet filing systems you may already have been using, and the new disc and Econet filing systems which replace these when you install the DNFS ROM.

The Different typefaces used

The different typefaces used in this book represent the following:

– Ordinary text appears like this, or like *this* for emphasis

– Text type into the computer or displayed on the screen appears **`like this`**

– Words like **`RETURN`** mean that you press the key marked RETURN rather than actually type the letters R E T U R N.

Part 1
6502 Second Processor

1 Introduction

The BBC Model B Microcomputer was designed with a second processor in mind, and this makes installation and operation very straightforward.

Here is a breif description of the extra facilities made available to you by the 6502 Second Processor.

- Up to 44K of user memory for BASIC programs in the Second Processor alone.
- Program execution is up to 50% faster than before.
- Size of user memory is not affected by the current screen mode.
- Character fonts automatically exploded: all keyboard characters may be immeadiatly redefined, as well as the spare ASCII character.
- BBC Microcomputer and Second Processor memory easily distinguishable.
- Languages can be stored on cassette or disc, as well as accessed from paged ROM.
- Paged ROM can be copied onto disc or cassette and then removed.

What is the Second Processor?

The Second Processor consists of a 6502 microprocessor system and 64K of Random Access Memory (RAM). In addition, it contains an interface between itself and the BBC Microcomputer called the Tube.

With a Second Processor fitted, the job of the BBC Microcomputer is to serve as an I/O Processor; that is, to handle all the inputs and outputs (keyboard, RS423, text and graphics output to monitors, printer output, disc drives, cassette recorders and so on).

The job of the Second Processor is to run languages such as BASIC, assembly language, word processing, and to run user programs. The Tube handles the two-way communication between the BBC Microcomputer and the Second Processor, and this leaves the Second Processor to get on with running programs, while the BBC Microcomputer looks after all the inputs and outputs as requested by the Second Processor.

The Second Processor offers you two major features: very fast program execution, and far more user memory.

From now on, the BBC Microcomputer used with the Second Processor will be referred to as the I/O Processor.

2 Installation

Checklist of items

Apart from this User Guide, you should have the following items in the box you have just openned:

- A 6502 Second Processor fitted with a flat cable and a mains lead.
- Two chips: one caled DNFS, and the other is Hi-BASIC. These are ROMs (Read Only Memory). If you have just bought your Second Processor from a dealer, these ROMs may have already been fitted to your BBC Microcomputer.
- A guarantee registration card.

Should any of these items be missing, please contact the supplier without dely.

Connecting the 6502 Second Processor to the BBC Microcomputer

If the DNFS and Hi-BASIC ROMs have not been fitted to your BBC Microcomputer, instructions on how to do this are given in Appendix A. If after reading the instructions, you do not feel qualified to fit the ROMs yourself, please call Acorn Computers Customer Service Department, on Cambridge (0223) 210111, for advice.

DNFS and Hi-BASIC ROMs

What do they do?

The DNFS ROM contains software which is loaded into the I/O Processor (ie BBC Microcomputer when used with the Second Processor) on power-up, and forms part of the I/O Processor operating system. As a bonus, this ROM also contains software to drive the disc filing system and the network filing systen. If you already have either or both systems working on your BBC Microcomputer, then you should replace either (or both) existing DFS/NFS Roms with the DNFS ROM.

The Hi-BASIC ROM is specifically for use with the Second Processor, and contains an extended version of the BASIC language in the BASIC ROM you already have installed. Both standard BASIC and Hi-BASIC will work on the Second Processor, but Hi-BASIC will not work on the BBC Microcomputer with the Second Processor disconnected. In addtion, Hi-BASIC is designed to make full use of the extra memory in the Second Processor. If you use standard BASIC, some of the Second Processor memory will be wasted. For the time being, it's a good idea to have both standard BASIC and Hi-BASIC installed in your BBC Microcomputer.

Fitting the two ROMs

Please refer to Appendix A for a detailed explanation of how to fit these two Roms if this has not aready been done.

Connecting the Second Processor

With both machines switched off and sitting on a flat surface, place the Second Processor unit at the right-hand side of the BBC Microcomputer and as as close as possible.

Being careful of any cables already connected to the BBC Microcomputer, gently move both units so that the front edge of each unit overlaps the edge of the surface, until the row of connectors underneath the BBC Microcomputer case is accessible.

Plug the flat cable connector from the Second Processor into the socket marked 'tube®'. If you are unsure which way round the flat cable connector plugs in, then look closely at it and find a small ▲ symbol. When plugged into the 'tube' socket, this arrow must be in line with the ▲ symbol printed above the socket.

Note: Although the software which handles the disc filing system and Econet® is now fitted to your machine, neither program will work unless the full disc or Econet upgrade has been installed.

3 Starting Up

Switch on both machines. For the I/O Processor to ‘recognise’ the Second Processor, the I/O Processor must undergo a ‘reset’ (equivalent to pressing CTRL BREAK) *after* the Second Processor has been switched on. So either swith on the Second Processor first, then the I/O Processor (which automatically undergoes a ‘reset’ on switch-on), or press CTRL BREAK. The following message, or something similar, should appear on your screen.

```
Acorn TUBE 64K

Acorn DFS

BASIC

>_
```

If **Acorn TUBE 64K** does not appear on your screen, first check that the DNFS ROM is fitted, and if so, then contact your nearest Acorn Service Centre.

Using the BBC Microcomputer on its own

If you want to use the BBC Microcomputer on its own, simply switch the Second Processor off at the switch on the back, and press CTRL BREAK. (Remember that you may have been using a language stored on disc or cassette which was running on the Second Processor. If you want to run this language on the BBC Microcomputer, it will have to be available in ROM. In this case you may have to switch the BBC Microcomputer off and insert the required language ROM if it isn’t already plugged in.) The following message, or something similar, should apear on your screen:

```
BBC Computer 32K

Acorn DFS

BASIC

>_
```

4 System Memory

Introduction

The BBC Microcomputer contains memory which is used for storing the machine operating system program, storing a language program like BASIC, storing the information which appears on the screen, storingyour own work (ie a BASIC program or a text file if you are using a word processing language) and so on.

The BBC Microcomputer's memory contains 65536 untis which can each store a number from 0 to 255. This memory is commonly known as '64K' of memory, and is normally referred to in hexadecimal numbers. So we can say that the memory is numbered from 0000 to &FFFF, and each memory unit can store anumber from 00 to &FF. Please refer to the *BBC Microcomputer System User Guide* for a more detailed explanation, and a description of how the memory is devided up (the memory map).

The 6502 Second Processor also contains 64K of memory, but much more of it is available for storing your own programs than in the BBC Microcomputer.

The I/O Processor memory map

Figure 1 shows the memory map for the BBC Microcomputer when working with the 6502 Second Processor.

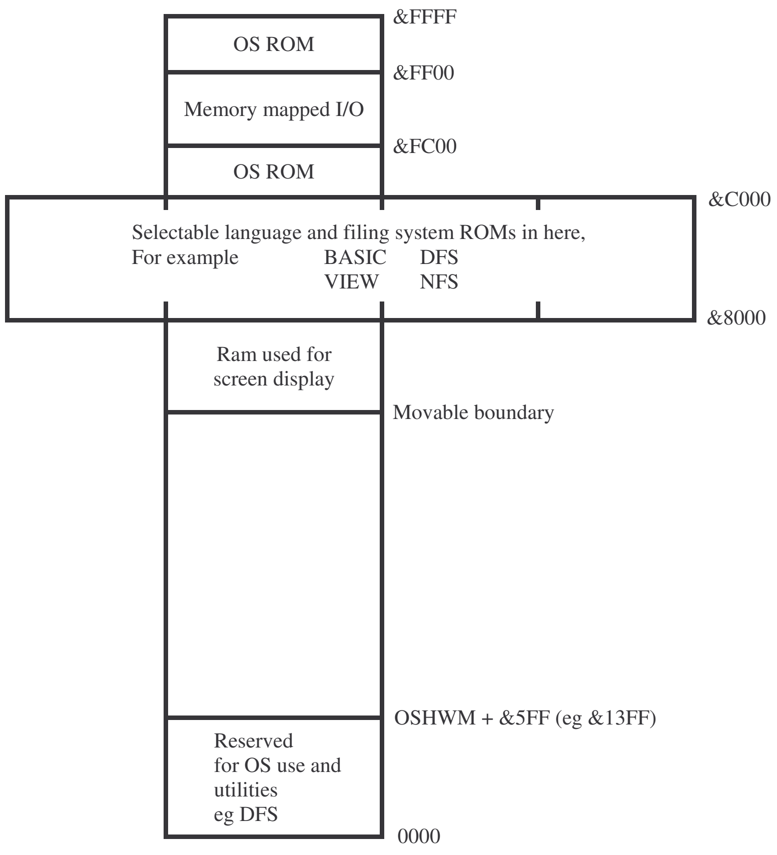

Figure 1

OSHWM stands for operating system hight water mark. This is the point at which the memory reserved by the operating system ends, and will depend on which filing system ROMs are fitted in selectable ROM. For example, with no filing system ROMs fitted, OSHWM = &E00; with a disc filing system fitted (ie including a DFS ROM), OSHWM = &1900.

With the Second Processor connected, an extra &5FF is also reserved to allow sufficient memory for holding new ASCII character definitions. This extra memory allows characters &20 to &FF to be redefined, in addition to the normal redefinable characters. This is exactly equivalent to issuing a `*FX20,6` (refer to chapter 42 in the *BBC Microcomputer System User Guide*). When the Second Processor is fitted, this happens automatically.

The 6502 Second Processor memory map

Figure 2 shows the memory map for the 6502 Second Processor.

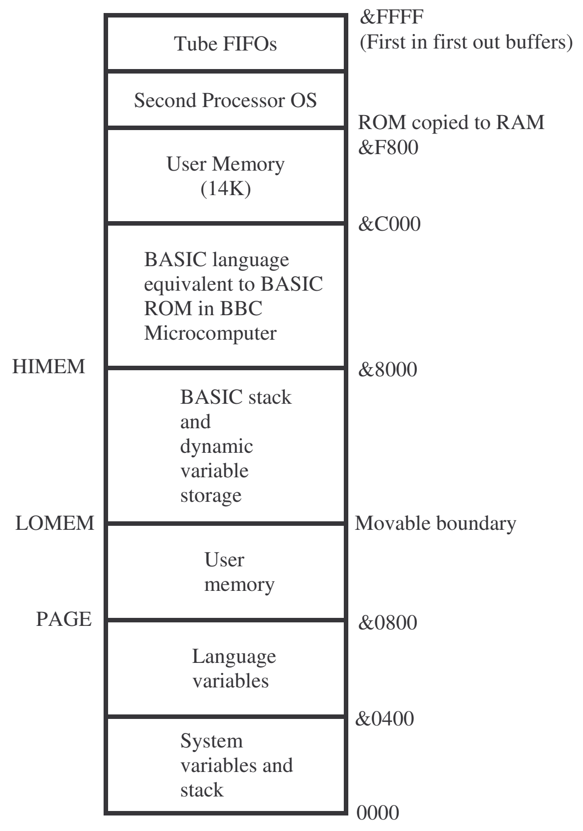

Figure 2

The complete memory map of the Second Processor is in RAM. The drawing above shows the memory map for normal BASIC. This is typical of languages designed for the BBC Microcomputer. Note that there is a 14K block of memory above the language code which, whilst available for use for storing BASIC variables, cannot be used for storing BASIC programs (see chapter 6). However, no memory space is taken by the screen display or filing systems.

On switch-on

The Tube software is copied from the DNFS ROM in the I/O Processor to the 4, 5, 6 and 7 in the I/O Processor.

The Second Processor operating system is copied from a ROM inside the Second Processor to &F800-&FFFF.

The language ROM automatically selected by the I/O Processor is copied to &8000-&C000 in the Second Processor, except Hi-BASIC which is copied to &B800-&F800.

5 Using standard BASIC in the Second Processor

BASIC works in exactly the same way in the Second Processor as in the BBC Microcomputer. The only differences are as follows:

Keywords such as **PAGE**, **HIMEM, LOMEM** and **TOP** will give different values from normal. This is because they work on the Second Processor user memory where your BASIC programs are stored, and no longer ‘see’ the I/O Processor’s user memory.

Unless you specify otherwise, **PAGE** will always be at &800. In the BBC Microcomputer, the default value of **PAGE** depends o any filing system ROMs you have fitted, and whether or no the character fonts have been exploded for redefinition (***FX20**).

The value **HIMEM** of is always &8000, regardless of the current screen mode. This is because the memory used for the screen is in the I/O Processor itself, and not the Second Processor.

In other words, running BASIC in the Second Processor is the same as BASIC in the BBC Microcomputer, expcept that more memory is available.

6 Hi~Basic

Hi-BASIC is an improved and extended version of standard BASIC. Part 2 of the book describes Hi-BASIC in detail.

Standard BASIC and Hi-BASIC

Using standard BASIC (which is automatically copied into the Second Processor), the memory from &C000 to &F800 (14K) can be used for holding assembly language subroutines (either from BASIC or from disc/cassette), or storing data using the **?** or **!** operators. It is also available to standard BASIC for storing variables: set **LOMEM** to &C000, and **HIMEM** to &F800. BASIC variables are then stored in this area rather than taing up BASIC program space.

Hi-BASIC makes this additional 14K available to BASIC programs by relocating the BASIC language code to just below the Second Processor operating system memory. Compare the following map in figure 3 with the one in chapter 4.

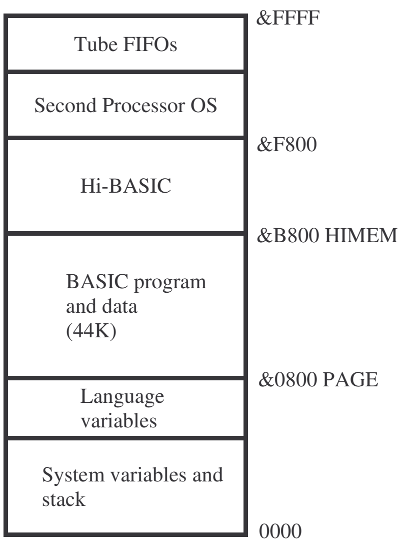

Figure 3

44K is now available for BASIC programs and data (note that **`HIMEM`** for Hi-BASIC is &B800 – regardless of screen mode).

Using Hi-BASIC

If you have both standard BASIC and Hi-BASIC fitted, and the ROMs have been correctly installed (see Appendix A), then standard BASIC will be selected on switch-on. To select Hi-BASIC, you must tell the operating system which socket contains the Hi-BASIC ROM. If you only have Hi-BASIC fitted, Hi-BASIC will be selected on switch-on, and you do not need to do anything. The four sockets are numbered 12, 13, 14, 15 (see figure 4 below).

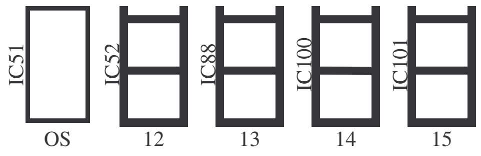

Figure 4

If you don't know where Hi-BASIC has been fitted, remove the lid on the I/O Processor, and take a look. To access and run Hi-BASIC, type **`*FX142,`** followed by the number of the Hi-BASIC socket, then press RETURN. If your dealer has not already done so, note here which socket contains Hi-BASIC in *your* BBC Microcomputer. For example, if the Hi-BASIC ROM is plugged into socket number 14, then type

```
*FX142,14  RETURN
```

The message

```
BASIC

>_
```

should appear on your screen. If not, then you have probably typed in the wrong socket number – press CTRL BREAK and try again. If you only ever wish to use your system with the Second Processor connected, you can remove the standard BASIC ROM (marked 'PB01' or 'PB05) from its socket. When you switch on, Hi-BASIC will be automatically loaded into the Second Processor and you will not have to issue the **`*FX`** call. However, you will not be able to use BASIC without the Second Processor.

If you try to load a filing system ROM, you will get the error message

```
This is not a language
```

The filing system ROM will have been loaded into the Second Processor, despite the error message.

Note: To check that you are running Hi-BASIC and not standard BASIC on the Second Processor, type

```
PRINT ~HIMEM RETURN
```

If the answer is

```
B800
```

Then you know that you are running Hi-BASIC.

Using Hi-BASIC with a disc system

Having loaded Hi-BASIC into the Second Processor as described above, you can now make a copy of it onto disc. This has two advantages: firstly, you can directly access Hi-BASIC off disc, regardless of the current language (and create a !**BOOT** file which will boot the Second Processor into Hi-BASIC by pressing SHIFT BREAK – see the *Disc Filing System User Guide*). Secondly, you can remove the Hi-BASIC ROM from the I/O Processor, which leaves you a spare ROM socket.

Type

```
*SAVE HIBASIC B8900+4000 RETURN
```

This will save the Hi-BASIC code currently at &B800-&F800 onto disc, as a file called **HIBASIC**. Loading and running the language from disc is done in the standard way:

```
*HIBASIC RETURN
```

7 Expanding the memory

If you want to run machine-code programs in the Second Processor, and do not need to load a language, then effectively you can add another 16K to the Second Processor user memory by using up the memory normally occupied by a language. If you are currently running BASIC, type

`CALL &F800` RETURN

The > prompt is replaced by a * prompt. Because you are no longer running a language, the only commands you can use are operating system and disc/net filing system commands. You can now load and run machine-code programs from disc (or cassette, or Econet if fitted). Although the previous language is still in memory, it can be overwritten.

Because you may want to do this from a language other than BASIC, for example VIEW, or from a cassette/disc-based program which allows OS calls from the keyboard, the Second Processor's operating system provides you with an additional command which is **`*GO`**. The procedure is similar to **`CALL`** in BASIC. Type

`*GO F800` RETURN

To get back to the previous language (as long as you haven't written all over it!) type

`*GO` RETURN

or

`*GO 8000` RETURN

or whatever the start address of the previous language is.

To reload the previous language from cassette or disc type

`*FRED` RETURN

where **FRED** is the filename of the language stored on disc or cassette. Remember that ***RUN** can be shortened to ***** for a disc-basic program, but if you are using cassette, the minimum abbreviation is ***/** for ***RUN**.

Alternatively, you can copy the previous language from ROM by name.

```
*BASIC RETURN
```

(to get back to BASIC for example), or press the **CTRL** and **BREAK** keys, which will copy the language ROM normally accessed on power-up into the Second Processor.

Or copy it from ROM, using the ROM socket number

```
*FX142,14 RETURN
```

(to get back to Hi-BASIC for example, where the Hi-BASIC ROM is plugged into socket number 14).

8 Copying ROMs

Using the 6502 Second Processor, it is possible to store code on disc that would normally be placed in paged ROMs inside the BBC Microcomputer. If an unknown * command is entered for which no paged ROM is present, the operating system will look for a file of that name on disc (if a disc filing system is installed and selected), and will load the file into the Second Processor for execution. This means that you can keep a 'library' of paged ROMs on disc, which can be called up (provided the disc is installed) in the same way as ROMs installed inside the BBC Microcomputer.

However, in using this facility you must ensure that you do not infringe the rights of the owner of the copyright of the program contained in the ROM.

Where this copyright is owned by Acorn, the program may be copied from ROM in the circumstances described in Acorn's 'Terms and Conditions of Sale and Use of Software' but not otherwise. A copy of these terms and conditions may be obtained from Customer Services, Acorn Computers Limited, Fulbourn Road, Cherry Hinton, Cambridge.

Where this copyright is not owned by Acorn, specific permission must be obtained from the owner of the copyright before making such copies.

To copy a language ROM onto disc, first insert the ROM into one of the spare ROM sockets inside the BBC Microcomputer (see Appendix A). Switch the computer on and load the appropriate language by typing the appropriate command – eg

```
*LANG RETURN
```

Then type

```
*SAVE LANG 8000 +4000 RETURN
```

The ROM may than be removed. To recall the language from disc, type

```
*LANG
```

in the normal way. (Note that the procedure for saving Hi-BASIC is different – see chapter 6.)

9 Distinguishing between memories

You will notice from the memory maps that the I/O Processor and the Second Processor memories are both numbers from 0000 to &FFFF. With the Second Processor working, the disc/cassette/Econet filing systems must somehow distinguish between the two memories in order to **SAVE** or **LOAD** programs or data in the correct place. This is done by defining Second Processor memory as running from 0000 to &FFFF, and I/O Processor memory from &FFFF0000 to &FFFFFFFF.

For example, memory location &C000 resides, by definition, in the Second Processor: memory locations &FFFFC000 resides in the I/O Processor.

Suppose you run a machine-code program which resides in the Second Processor from memory location &6000 to &6500. This program draws a pretty picture on the screen in **MODE 1**, and you would like to save both the program and the resulting picture onto disc.

To save the program onto disc, you would type

```
*SAVE GRAFPRG 6000 6500 RETURN
```

where **GRAFPRG** is the filename you have chosen. To load and run the programs from disc, you would type

```
*RUN GRAFPRG RETURN
```

The program would be loaded back into the Second Processor rather than the I/O Processor (no leading ‘FFFFs’ in the load address and end address), and would run.

To save the resulting picture onto disc, you would have to save the screen memory used in **MODE 1** graphics, and this resides in the I/O Processor.

The screen memory for **MODE 1** starts at &FFFF3000, and ends at &FFFF7FFF (ie memory locations &3000 to &7FFF in the I/O Processor). To save the resulting picture onto disc, you would type

```
*SAVE PICTURE FFFF3000+5000 RETURN
```

(where **PICTURE** is the filename you have chosen). If you now reload this by typing

```
*LOAD PICTURE RETURN
```

the file **PICTURE** will be loaded into the I/O Processor, and as long as you are already in **MODE 1**, the picture will reappear. (From this you can probably deduce why games programs which *directly access screen memory* fail to work on the Second Processor).

You may have noticed that when you execute a *INFO on a disc file, addresses are preceded by either 00 or FF, rather than 0000 or FFFF. For example, typing

```
*INFO PICTURE RETURN
```

might display

```
$.PICTURE  FF3000 FF3000 FF4FFF 07A
```

and typing

```
*INFO GRAFPRG RETURN
```

might display

```
$.GRAFPRG  006000 006000 000500 05B
```

This disc filing system ignores the two most significant bytes of the addressing – the cassette filing system displays the addressing in full.

Important: It is only the filing systems that recognise these ‘extended’ addresses. Current versions of BASIC and Hi-BASIC make no distinction. Only the memory in the Second Processor can be directly accessed from BASIC. This means that using the **!** and **?** operators to access memory locations in the I/O Processor from the Second Processor doesn’t work – for example

```
?&FFFF2000 = &55 RETURN
```

will put the value &55 into memory location &2000 in the Second Processor, and *not* into memory location &2000 in the I/O Processor as specified by the leading FFFFs.

The same goes for saving and loading BASIC programs. If you save a BASIC program to disc or cassette from the BBC Microcomputer, executing a **`*INFO`** will display addresses with leading FFs. Saving a BASIC program from the Second Processor and executing a **`*INFO`** will display addresses with leading 00s. However, BASIC will always load either program into whichever processor is currently running BASIC. This means that your BASIC programs will be totally compatible with either BBC Microcomputer, or I/O Processor with Second Processor.

However, it is possible to read and write to memory locations in the I/O Processor, and this is done using OSWORD calls 5 and 6. OSWORD calls are dealt with in chapter 43 of the *BBC Microcomputer System User Guide*. Here is a useful program to write to and read from the I/O Processor.

```
 10 DIM X% 20: Y%=X% DIV 256 : OSWORD=&FFF1
900 END
1000 DEF FNREADIO(addr): !X%=addr: A%=5: CALL
OSWORD
1010 DEF PROCWRITEIO(data,addr):
!X%=addr:X%?4=data
1030 A%=6: CALL OSWORD: ENDPROC
```

A friendly word of warning: the Tube hardware is at addresses &FEF8 to &FEFF in the memory map. It is ‘read sensitive’, which means that if you read any of these locations then you could cause the system to crash. Likewise, don’t use indexed instructions based in this area, for example

```
LDA &FEF8,Y
```

when **`Y`** is **`&FE`** will correctly access zero page &00f8, but as a ‘side effect’ it will also read from the Tube at &FEF8 – the beginning of the Tube operating system.

10 Using the Second Processor from assembly language

Introduction

All communication between the Second Processor and the BBC Micro-computer is done via the Tube. All file handling and operating system calls are made in exactly the same way as on the BBC Microcomputer. In other words, an understanding of how the Tube reads and writes to the BBC Microcomputer is unnecessary.

For a description of the OS calls available from assembly language, please refer to the *BBC Microcomputer System User Guide*.

Points to remember

The following is a summary of special points to remember when programming in assembly language on the Second Processor.

OS calls inside an event: In the Second Processor no OS calls are allowed inside event routines. In the BBC Microcomputer all OSBYTE and some OSWORD calls are allowed.

OSBYTE calls made from the Second Processor: Low numbered OSBYTE calls (0-127) return a value only in the X and Y register, high numbered OSBYTE calls return values in the X and Y registers, and the carry flag.

Note: OSBYTE call &82 always returns 00 in X and 00 in Y.

OSBYTE call &83 always return 00 in X and 08 in Y.

OSBYTE call &84 always gives the execution address of the currently running program; eg &8000 for standard BASIC and &B800 for Hi-BASIC.

Zero page to page eight are used as follows:

Page 0 to &EE is available
Page 1 = stack
Page 2 = OS indirections and user memory
Page 3 = error messages
Page 4 – 7 are available to the user, although not in BASIC
Page 8 - &F7 available to the user

Interrupts generated by devices connected to the BBC Microcomputer cannot be serviced by the Second Processor.

Note: The saving and transmission of values across the Tube is done as economically as possible. For example, with OSBYTE calls 0 to &7F, only X and A are transmitted and only X is returned. Non-standard use of these calls will cause hardship.

Part 2
Hi~BASIC (BASIC II)

Introduction

Part 2 of this User Guide explains the differences between standard BASIC and BASIC II. Hi-BASIC is the same as BASIC II, but is relocated in memory to &B800-&F800.

In many respects BASIC II is similar to BASIC; however there are several differences between the two. Some of the BASIC keywords have been extended so they are more powerful and additional ones have been incorporated. Furthermore several of the arithmetic routines have been recoded so that they give greater precision.

For the assembler programmer four new useful operations have been added and the OPT instruction has been extended to enable programs to be relocated easily.

Apart from the extra facilities which BASIC II offers, please refer to the *BBC Microcomputer system User Guide* for a detailed account of BBC BASIC.

1 Alterations and extensions to BASIC keywords

ABS

The unary operator may be used in BASIC II, eg

PRINT –ABS(1)

This will now give the value –1

In BASIC this gave a **`Type mismatch`** error.

COUNT

This counts the number of characters printed using **`PRINT`**, since that last new line. This has now been altered so that **`COUNT`** is reset to zero after a change of mode, eg

```
10 PRINT “Hello”;
20 MODE 3
40 PRINT “Goodbye”;
40 PRINT COUNT
```

In BASIC this would leave the screen showing:

```
Goodbye    12
```

With BASIC II the following is obtained:

```
Goodbye    7
```

ELSE

In BASIC **ON…GOTO/GOSUB...ELSE** could not be used inside procedures or functions. Only the **ON…GOTO/GOSUB** part was available. Attempting to use an **ELSE** part caused an error message to be printed. In BASIC II the **ON…GOTO/GOSUB** has been extended so that it can be used with an **ELSE** part anywhere in a program.

Example

```
 10 PRINT "If you want to know what all the
little piggies were doing this afternoon, give
the number of the little piggy you are
interested in when asked."
 20 INPUT "Which little piggy",A
 30 PROCPIG(A)
 40 GOTO 20
 50 DEFPROCPIG(X)
 60 ON X GOTO 70,80,90,100,110 ELSE GOTO 120
 70 PRINT "The first little piggy went to
market.": ENDPROC
 80 PRINT "The second little piggy stayed at
home.": ENDPROC
 90 PRINT "The third little piggy had roast
beef.": ENDPROC
100 PRINT "The fourth little piggy had none.":
ENDPROC
110 PRINT "The fifth little piggy ran all the
way home.": ENDPROC
120 PRINT "There were only five little piggies.":
ENDPROC
```

EVAL

This function has been extended. In BASIC, **EVAL A$** could be used to evaluate strings such as:

```
A$ = "SIN(X/120) + COS (X/30)"
A$ = "PI"
```

In addition, in BASIC II the **EVAL** function can be used to evaluate pseudo-variables such as **TIME**, **PAGE**, **HIMEM** and **LOMEM**, eg

```
A$ = "TIME"
PRINT EVAL (A$)
```

Will print out the current value of the pseudo-variable **TIME**.

INPUT

If more than one string or value is to be input at a time then the variable identifiers have to be separated from each other. In BASIC this was done using commas, eg

```
INPUT NAME$, AGE, HEIGHT
```

In BASIC II either commas or semicolons may be used, eg

```
INPUT NAME$, AGE; HEIGHT
```

When entering values, however, these should be separated by commas as before, not semicolons, eg

```
MICHAEL,21,170
```

INSTR

In BASIC, **INSTR** could be used to find the occurrence of one string inside another, eg

```
PRINT INSTR("Hello","l")
```

Would print 3 since the first 'l' is in the third character position. However, the second entry had to be shorter than the first for the function to operate correctly, eg

```
INSTR("l","Hello")
```

Would not work correctly inside a function or procedure.

In BASIC II, **INSTR** has been extended so that if a longer string is searched for inside a shorter one, as in the example, then this will result in **INSTR** returning the value **0**.

This means that it is now possible to use, for example

```
100 INPUT "What is the longer string",A$
110 INPUT "What string do you wish to search
for",B$
120 X = INSTR(A$,B$)
```

Inside a procedure of function without having to check the A$ is longer than B$.

ON ERROR

In BASIC it was possible to jump to most line numbers using **the ON ERROR GOTO…**, but not all lines could be jumped to in this way, for example **ON ERROR GOTO 9999** could not be used. This command has now been altered to accommodate all line numbers.

OPENIN and OPENUP

In BASIC and BASIC II an existing file can be opened to allow data to be read or altered, or to allow more data to be added to the end. In BASIC, this function was performed by the instruction **OPENIN**; in BASIC II it is done by **OPENUP**. Since these keywords give exactly the same result, the token for them both is &AD. Hence, if a program containing the instruction **OPENUP** is written on a BBC Microcomputer containing BASIC II then the instruction will become tokenised to &AD. If this program is then saved and loaded onto a machine containing BASIC, the program will work in exactly the same way, but when listed it will display the instruction as **OPENIN**. This will apply the other way around as well so existing programs do not need to be altered to run in BASIC II.

The keyword OPENIN does exist in BASIC II but it has a different meaning. BASIC II uses the keyword **OPENIN** to open a file for read-only operations; this was not possible in BASIC. Since this is a new facility it has a new token, &8E. Note that programs written in BASIC II which contain the instruction **OPENIN** will not run in BASIC. This applies to all programs containing any of the new instructions.

Examples

```
10 DIM A$(20)
20 channel = OPENIN(“name”)
30 FOR N = 1 TO 20
40 INPUT#channel,A$(N)
50 NEXT

10 channel = OPENUP(“name”)
20 FOR N = 1 TO 20
30 P = PTR#channel,A$
40 INPUT#channel,A$
50 IF A$ = “Jim” THEN PTR#channel = P :
PRINT#channel,”Joe”
60 NEXT
```

2 Error handling

The standard error handling procedures in BASIC II do not use any space no the software stack. This means that when a program runs out of all free space the error message printed out is no longer followed by a **`No room`** message.

Fatal errors

`STOP` and **`No room`** are now classed as errors; they have an error number of 0. They are, however, 'fatal errors' in that when they are processed they have an **`ON ERROR OFF`** effect, eg

```
10 ON ERROR GOTO 30
20 STOP
30 PRINT "HELLO"
```

This will result in **`STOP at line 20`** being printed since the error is not trapped.

Any error with number 0 is classed as a fatal error in BAIC II. Hence fatal errors can now be generated by the user in assembler, eg

```
 10 ON ERROR PRINT "An error has occurred"
    ................................
100 [
110 .stophere    BRK
120              EQUB 0
130              EQUS "stop here"
140              EQUB 0
150 ]
    ................................
200 CALLstophere
```

See page 41 for an explanation of **`EQUB`** and **`EQUS`**.

The call of ‘stophere’ will result in **`Stop here`** being printed. However, if line 120 is replaced by

```
120               EQUB 20
```

then the call of ‘stophere’ will result in **`An error has occurred`** being printed. This is because originally the error is given the number 0 and so is treated as a fatal error and is not trapped by the **`ON ERROR`** command. In the second case, however, the error is given the number 20 and is treated like a normal error.

Alterations to error messages

The message printed out by the command **`STOP`** has also been changed to **`STOP`**. In BASIC it printed **`STOP at line 0`**.

An illegal **`DIM`** statement, for example **`DIM P% -2`**, will now give the error message **`Bad DIM`**.

New error message

A new error message has been introduced, error 45 **`Missing #`**.

This is printed if any of the following are used without a ‘#’ sign:

`PTR`, **`EOF`**, **`BGET`**, **`BPUT`**, **`EXT`**.

3 Increases in precision and efficiency

LN and LOG

These functions have been rewritten to make them more accurate. This has a greater effect at the limits of the number range allowed.

PRINT and STR$

With these functions in BAIC II, ten figures of precision on printing are allowed instead of nine. This allows the maximum positive integer 2147483647 to be printed out, eg

```
X = 2147483647
@% = &00000A0A
PRINT X
```

In BASIC the result is 2.1474365E9.

In BASIC II the result is 2147483647.

To retain compatibility with **PRINT**, **STR$** has been given the default value of ten figures. This will result in **STR$** giving different values from before, eg 7.7 which is a recurring binary fraction will be converted to 7.699999999.

SIN and COS

These have been recoded in order to increase their accuracy.

String handling procedures

The allocation of space for string is more efficient in BASIC II, eg

```
REPEAT A$ = A$ + “*” : UNTIL LEN A$ = 255
```

In BASIC this would have allocated a total of 3882 bytes, whereas in BASIC II only 263 bytes are being used. However, the optimisation is only implemented when just one string is being used at a time, eg

```
REPEAT A$ = A$ + “*” :B$ = B$ + “!” :
UNTIL LEN A$ = 255
```

In both BASIC II and BASIC this uses 7764 bytes.

4 New Commands

OPENUP

See ‘OPENIN and OPENUP’ in chapter 1.

OSCLI

OSCLI takes a string expression and gives it to the operating system command line interpreter. Hence it acts in a similar way to the operating system commands, eg

```
OSCLI”CAT”
```

acts like

```
*CAT
```

However **OSCLI** is more powerful than this, eg

```
A$ = “CAT”
OSCLI A$
```

Is also equivalent to *CAT whereas

```
A$ = “CAT”
*A$
```

cannot be used.

OSCLI is particularly useful for loading files, eg

```
OSCLI “LOAD “ + FILES$ + “ “ + STR$˜(TOP−2)
```

Will load a BASIC program onto the end of another BASIC program.

The **STR$** in the above example is a function which takes the value of TOP-2 as a number and returns the equivalent string. The ˜ sign means that the hexadecimal representation of the number is taken rather than the decimal representation.

```
*LOAD FILE$ TOP-2
```

cannot be used instead of the **OSCLI** command since the ***LOAD** needs the address to be hexadecimal value and will not accept a variable such as TOP-2.

The shortest abbreviation for **OSCLI** is **OS.**, and its token value is &FF. It has no unique errors of its own, just the normal **Type mismatch** error.

5 New features for assembler programmers

ASC

In BASIC II **`ASC ":"`** may be used in the assembler. In the original BASIC this led to confusion.

EQUB, EQUW, EQUD, EQUS

These four operations are available in the assembler. They all take a single argument and put its value into the assembly code. The last letter in the name determines whether it is a byte, word, double word or string that it enters. **`EQUS`** puts all the characters of a string into the code without a carriage return; if one is required this should be added to the string itself. An example of the use of **`EQUS`** is shown below:

```
 10                    DIM CODE 200
 20                    oswrch = &FFEE
 30                    nop    = &FE
 40                    addr   = &70
 50                    FOR pass = 0 TO 3 STEP 3
 60                    P% = CODE
 70                    [
 80                    OPT pass
 90.go                 JSR vstring
100                    EQUS "hello"
110                    NOP
120                    RTS
   .............................
200.vstring            PLA : STA addr
210                    PLA : STA addr + 1
220                    LDY #0
230.vloop              INC addr
240                    BNE nocarry
250                    INC addr + 1
260.nocarry            LDA (addr),Y
270                     CMP # nop
280                     BEQ out
290                     JSR oswrch
300                     JMP vloop
310.out                 JMP (addr)
320                     ]
330                     NEXT pass
340                     CALL go
```

In the above program **EQUS** inserts all the characters from the string **"hello"** into the machine code. **JSR vstring** then prints out all the characters until it reaches **NOP**.

EQUS can also be used within assembler to generate macros, eg

```
 10 FOR pass = 4 TO 7 STEP 3
 20 [
 30 OPT pass
    .....................
100 EQUS FNADD(X,Y)
    .....................
200 ]
210 DEF FNADD(X,Y)
220 [
    .....................
300 ]
320 = ""
```

This would place the code generated by the function **FNADD** in the main body of the code where the function was called. The **= ""** is there to signal the end of the macro since it ends the function call by returning a null string.

The four operations listed above have no errors of their own apart from the **Type mismatch** error.

Note that **EQUB**, **EQUW**, **EQUD** and **EQUS** have to be in upper case to be recognised.

OPT

In BASIC the lowest bit of the OPT statement's operand determines whether the assembled code is listed or not and the second bit is used to suppress or report errors. In BASIC II the third bit is used as well to determine whether the code is put at O% (the code origin) or P% (the program counter). If the bit is set then the code will be put at O% and both O% and P% will be incremented. If it is unset only P% will be used. For example OPT 4 (100 in binary) will suppress assembler errors, give no listing and will put the code at O% but with all references referring to the locations they would be pointing to if the code were located at P%. This means that it would then be straightforward to relocate the code to P% and run it.

In BASIC only the first and second bits were relevant so setting the third bit had no effect, therefore the code was always put at P%.

Example

```
 10 REM This compiles a program to produce code
to run at &400 which is in BASIC workspace
so direct assembly is not possible.
 20 FOR pass = 4 TO 7 STEP 3
 30 O% = &2000
 40 P% = &400
 50 [
 60 .score      EQUD 0
 70             OPT pass
 80 .scoreadd   TXA          \The Lo byte of the
 90             CLC          \score to be added is
100             ADC score    \stored On X
110             STA score
120             TYA          \The Hi byte of the
130             ADC score + 1\score to be added is
140             STA score + 1\stored in Y
150             BCC nocarry
160             INC score + 2
170 .nocarry    RTS
180 ]
190 NEXT pass
```

The above assembly code is a routine to increment the score in a game and is intended to be part of a larger program which initialises the score and then jumps to the subroutine **.scoreadd** whenever it is required. When run, the program will compile the machine code generated to &2000 but it has to be run at &400 since the address references are fixed to point into this area.

To save the program you need to know the length of it. To find this, type

```
PRINT~O% - &2000
```

This will print out the number of bytes of object code produced by the program. If this was &28A for example you could then type

```
*SAVE Score 2000 228A 400 400
```

where &228A = &2000 + &28A.

Alternatively the OSCLI command described earlier could be used:

```
OSCLI "SAVE Score 2000 " + STR$~(O%) + " 400 400"
```

In either case to run the program type

```
*RUN Score
```

Appendix A

Fitting the Hi-BASIC and DNFS ROMs

Important: The DNFS ROM *must* be fitted to your BBC Microcomputer for the Second Processor to work.

There are three ways you can organise your BBC Microcomputer to run BASIC:

Option 1 – standard BASIC only
Option 2 – standard BASIC *and* Hi-BASIC
Option 3 – Hi-BASIC only

Option 1 will work on both the BBC Microcomputer (Second Processor switched off) and on the Second Processor. However, full use is not made of the Second Processor's memory.

Option 3 will work on the Second Processor *only*. Please note that there are a few games and other programs which, for the sake of extra speed or other reasons, access the hardware of the I/O Processor directly and not though the operating system. These will not work on the Second Processor.

Option 2 is the most flexible. Where standard BASIC and Hi-BASIC are both fitted, standard BASIC must be placed in a ROM socket to the right of the Hi-BASIC ROM.

Fitting the Hi-BASIC ROM

To use Hi-BASIC you should insert the Hi-BASIC ROM into one of the free sockets. The ROM sockets are located on the front right-hand side of the circuit board inside the BBC Microcomputer casing. Please refer to figure 5 at the end of this Appendix.

1. To get to the board, undo the four screws which hold the casing together – on some computers these will be marked 'FIX'. Two of these screws are underneath the computer, and the other two can be found on the back.

2. Once the top is removed, release the bolts holding down the keyboard assembly. These are located on either side of the keyboard. Some machines have two bolts, others may have three.

3. There is no need to disconnect the keyboard completely, so the multi-wire connector to the main board can be left in place. Carefully displace the keyboard, rotating it clockwise through about 20 degrees so that the front right-hand side is accessible.

4. Locate the row of five large sockets. The one on the left contains the operating system. The BBC BASIC ROM should be in one of the other four. It can be identified by looking at the second line of writing on its upper surface which is a series of letters and number ending in the series ‘B01’ or ‘B05’.

Language and filing system ROMs – operating procedure

The four ROM sockets have an operating priority, decreasing from right to left. On a hard reset, or when the computer is switched on, the language ROM in the rightmost ROM socket takes priority over the others. Similarly, the filing system ROM in the rightmost position takes priority over any other filing system. So the position of the Hi-BASIC ROM in relation to any other languages present will determine whether your machine start up in Hi-BASIC or in another language.

If you want to start up in Hi-BASIC then you must insert your ROM to the right of all other language ROMs

Removing the BASIC ROM

To avoid bending any pins, the ROM must be removed very carefully. Take a screwdriver or similar tool and gently prise up each end, a bit at a time.

Inserting the Hi-BASIC ROM

1. Before taking the ROM out of its protective packaging, identify Pin 1 on the ROM. It is either marked with a dot on the top, in the corner or Pin 1, or the half-moon notch at one end of the ROM identifies the end of the ROM nearest Pin 1. Pin 1 should be on the left if the notch is held up.

2. Hold the ends of the ROM between finger and thumb, and line up all the pings over the destination socket. Pin 1 and the half-moon notch should point towards the back of the computer casing.

3. Now apply firm pressure to the ROM, but try not to force it! When the ROM is in, it appears to be slightly raised. Check that all the pins do enter the socket and that none are bent out or underneath.

Removing the DFS and NFS ROMs

If you already have a disc or Econet filing system fitted – or both – you will need to remove the DFS and NFS ROMs in the same way as the BASIC ROM (see above).

Inserting the DNFS ROM

Following the steps above for inserting the Hi-BASIC ROM.

The disc filing system has priority over the Econet filing system. To access the Econet filing system, type

```
*NET RETURN
```

and to revert to the disc filing system, type

```
*DISC RETURN
```

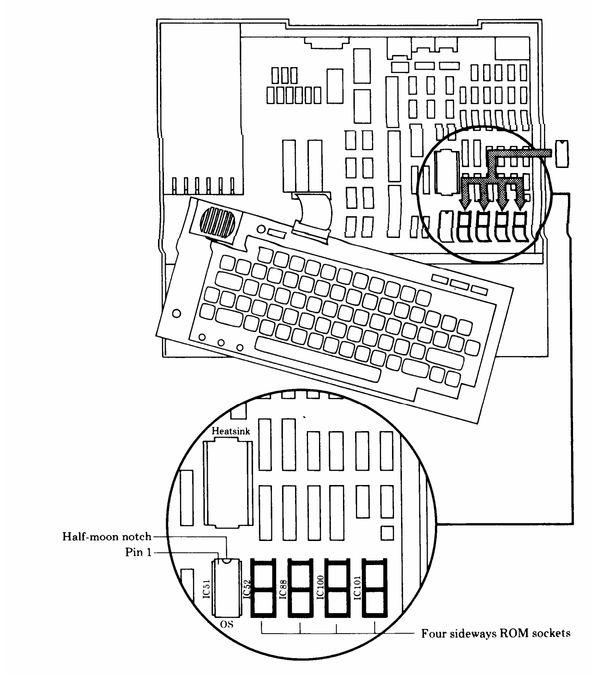

Figure 5 Inserting the Hi-BASIC ROM

Appendix B

Differences between the filing systems

If you have been using the disc filing system or Econet on your BBC Microcomputer, you may notice that there are differences between the filing system software you are used to, and the new software provided by the DNFS ROM. More specifically, these differences occur between version 0.90 and version 1.00 (and above) of the disc filing system, and version 3.34 and version 3.40 (and above) of the Econet filing system.

Changes to the disc filing system

1. Version 1.00 does not allow control characters, top bit set characters and **DELETE** in filenames or disc titles. For example, this means that coloured filenames in **MODE 7** are not possible.

2. The filing system commands **LIB** and **DIR**, and filename searching are different *if* any part of the full filename is not specified. For example

```
*DIR X  RETURN
*DIR :2 RETURN
```

In version 0.90, this will leave you in drive **2**, directory **$**. In version 1.00 this will leave you in drive **2**, directory **X**.

```
LOAD ":2.FILE" RETURN
```

Version 0.90 looks for **:2.$.FILE**, regardless of the current directory. Version 1.00 *will* observe the current directory, and if you are currently in directory **X**, then version 2 will search for **:2.X.FILE**.

The **LIB** command behaves in the same way.

3. Version 1.00 does not distinguish between upper case or lower case directory names, unlike version 0.90. For example

```
SAVE T.Fred
```

And

```
SAVE t.Fred
```

Will **`SAVE`** the file **`Fred`** into the same directory.

4. Some filing system messages have been rephrased in version 1.00

5. Version 1.00 provides an extra option for OSWORD &7F. If the drive number given is greater than &7F (ie a minus umber), the filing system reverts to the currently selected drive.

6. Open file handles, **`OPT1`**, **`DIR`**, **`LIB`** etc are preserved over a soft reset in version 1.00.

7. Version 1.00 provides line numbers with leading zeros for **`BUILD`** and **`LIST`**.

Changes to the Econet filing system

1. Version 3.40 prints catalogues in multiple columns as opposed to single columns in version 3.34.

2. In version 3.40, passwords can be masked at ‘log on’ for security.

3. Printing protocols in version 3.40 allow control characters to be sent to the printer server, enabling graphics dumps and ‘diablo’-type printers to be used.

4. The Econet systems can be run as IRQ task with net and disc activities in the foreground. This means that these activities can occur whilst a program is running, and not interfere with the program.

5. Stations fitted with version 3.40 and above do not recognise ‘privileged’ station numbers, and will be protected against immediate operations from *any* station.