About This Remade Document

Title:	Master Operating System : A Dabhand Guide (Second Edition)
Last remade:	27-Apr-2025
Repository:	https://github.com/acheton1984/AcornDocsRemade

This is not a beautifully "remastered" document as seen elsewhere. It has the same words on the same pages in a somewhat similar layout to the original, but is presented in an easy-to-read and searchable PDF format.

Reconstruction Notes

Source scan:

https://stardot.org.uk/mirrors/www.bbcdocs.com/filebase/hardware/MasterOperatingSystem.pdf

Draft 1a (yellow highlighter)

* Recreated from a scan of the original document.
  * OCR'd text tidied up
  * Just enough tabs added for proofreading, but not to line up correctly
  * Corrections (technical and grammatical, some for consistency)
  * Pencilled notes from physical copy incorporated
  * Program listings tested and corrected
* Font sizes, margins, line spacing, and character spacing are only approximate. Some words may appear on different lines, but the overall layout and pagination are preserved.
* The memory maps on pages 210-212 have **not** been recreated.

Draft 1b (orange highlighter)

* Additional formatting applied to improve readability (but not matching the original layout).
* Moved a block of text about scratch workspace from the section about PAGE &FF on p207 to the top of the chapter on p198
* &7FFF is now the top of main RAM, not &8000.
* Typos corrected in the disc listings (mainly in comments) - no functional changes

Draft 1c (green highlighter)

* Included "E"lectron in OSBYTE &17 (p50) in response to https://www.stardot.org.uk/forums/viewtopic.php?p=445401#p445401

* A title and remade document history page have been added, along with a link to this repository.

Associated Discs

Two disc images are available to accompany this manual:

* **Dabhand MOS Listings** (DFS format) – containing the remade versions of the programs from the manual. Available from this repository.
* Official **MOS Dabhand Guide** (DFS format) – original 5.25" disc from Dabs Press. Available as "MOS: A Dabhand Guide" at https://www.flaxcottage.com/Oddments/Programming.asp

Disclaimer

The underlying content is reproduced or adapted from the original documents for educational purposes and ease of access.

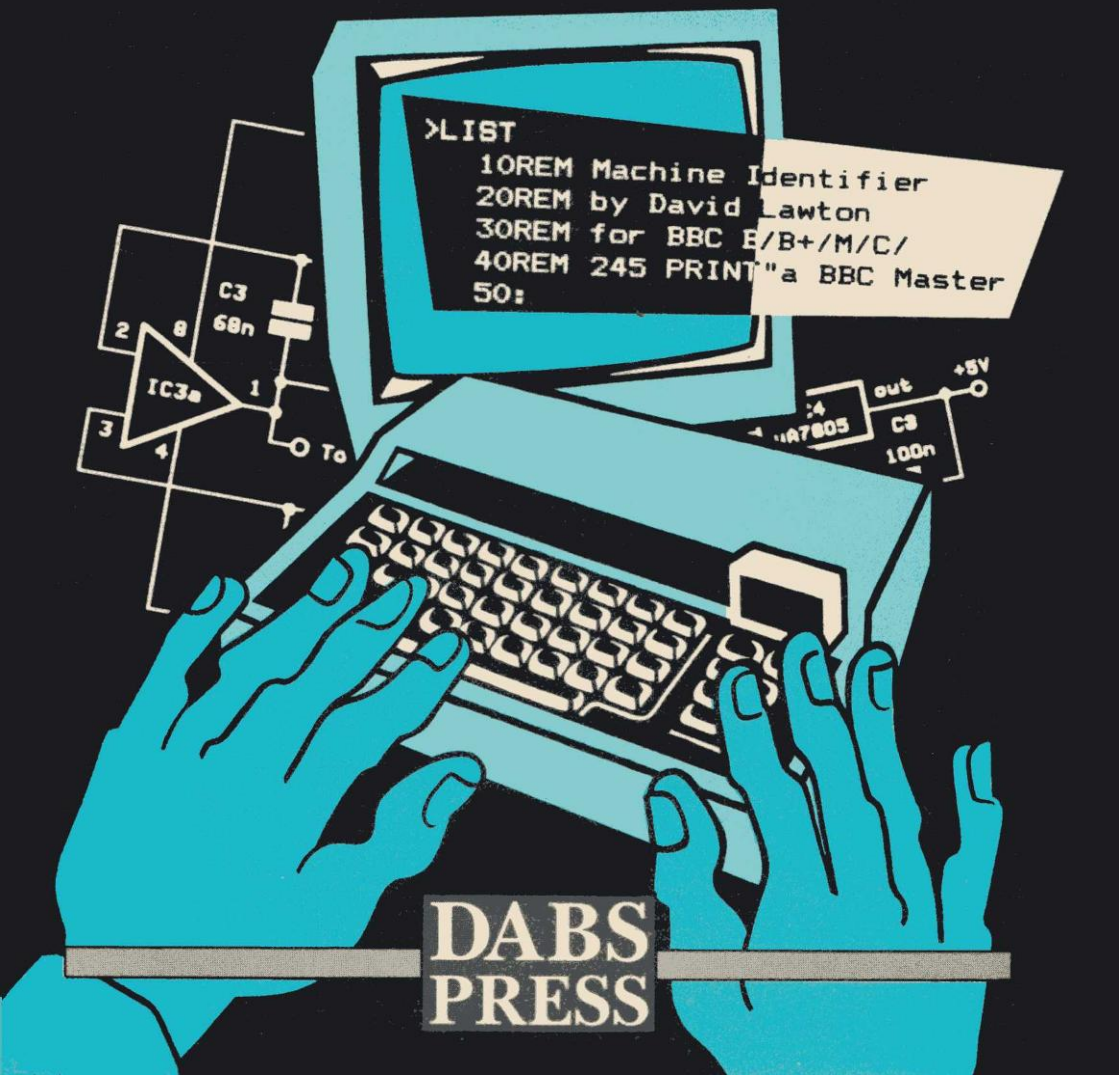
A Dabhand Guide

SECOND EDITION

DAVID ATHERTON

MASTER OPERATING SYSTEM

Including BBC B+ and Master Compact



DABS PRESS

MASTER OPERATING SYSTEM
A Dabhand Guide

David Atherton

DABS PRESS

Draft 1c - https://github.com/acheton1984/AcornDocsRemade

To Lynda

Master Operating System : A Dabhand Guide

ISBN 1-870336-01-1
First edition June 1987
Second edition January 1988
Editors: Paul Horrell and Bruce Smith
Cover: Paul Holmes
Acornsoft is a trade mark of Acorn Computers Ltd, Fulbourn Road, Cherry Hinton, Cambridge England. *MacAuthor* is published by Icon Technology Ltd, Leicester England. The *Apple Macintosh* and *Laserwriter* are produced by Apple Computer Inc. *Advanced Disc Toolkit* is produced by Advanced Computer Products, 6 Ava House, High Street, Chobham, Surrey.

Published by Dabs Press, 76 Gardner Road, Prestwich, Manchester M25 7HU England.

Typeset in 10 on 11 pt Palatino by Dabs Press using the Acornsoft VIEW word processor, MacAuthor, and an Apple Macintosh and Laserwriter. Printed and bound in Great Britain by A. Wheaton and Co Ltd, Exeter, Devon. A member of the BPCC Group.

Contents

1 : Introduction **7**
A History 7
Type Styles 9
Acknowledgments 9
About this book 10
Thank you... 10

2 : MOS Commands **11**
The A to Z of * Commands 13
Safe or not? 32

3 : The 65C02 Processor **33**
Introduction 33
Changes from the original 6502 34
Syntax 35
65C02 New Instructions 36
65C02 New Addressing Modes 43

4 : New MOS Calls **45**
OSBYTE Calls 45
OSBYTE Calls in Numeric Order 47
OSWORD Calls 75
OSWORD Calls in Numeric Order 76

5 : Shadow and SRAM **90**
Introduction 90
Shadow Memory 90
Sideways RAM 91
Using SRAM for ROMs 92
Using SRAM for Data Storage 94
Writing your own ROMs 94
Low Level Paging 95
Model B and Electron 95
Model B+ 96

Master 128 and Compact 96
High Order OSWORD &05 and &06 99

6 : Sideways ROM 105
Sideways ROM Format 105
ROM Header 106
Entering a Language ROM 108
ROM Service Calls 108
Service Calls 110
Extended Vectors 124
Clock ROM 125

7: The Tube 132
Introduction 132
Tube Host Code and OS 133
&400 - Copy Language across the Tube 133
&403 - Copy Escape Flag across the Tube 134
&406 - Writing Data across the Tube 134
Accessing Co-Processor Memory 135
Claiming and Releasing the Tube 135
Advising Tube OS of Data Transfer 136
Notes for programs on 6502 Co-Processor 140
Summary of Tube Registers 141
Tube Protocols 143

8 : Filing Systems 147
OSFIND 147
OSGBPB/OSBPUT/OSBGET 147
OSARGS 147
OSFILE 147
OSFSC 149
Prohibited Filename Characters 150
Using OSWORD &7F 150
Temporary Filing Systems 151
The Filing System Handler 153
Library Filing Systems 156
Action on Service Calls 157
Action on Reset 158
Deselecting Temporary Filing Systems 159

Programming Considerations 159
Filing System Information 160

9 : Non-Volatile RAM 161
Introduction 161
CMOS and EEPROM Memory Map 161
OSBYTE Revisited 164
CMOS RAM Editor 165

10 : Differences 172
BBC BASIC versions 172
BBC B and BBC B+ 177
BBC B and Master 128 178
Master 128 and Master Compact 179
Miscellaneous 180

Appendices 184
A : Complete OSBYTE List 185
B : Complete OSWORD List 191
C : Complete VDU List 193
VDU 23 194
VDU 25 196
D : Memory Map 198
E : OS Call and Item List 213
Baud Rates 213
Buffers 213
Events 214
Filing System Informations 214
System Calls and Vectors 215
F : Key Numbers 217
G : PCB Links 223
BBC Model B 223
BBC Model B+ 225
Master 128 227
Master Compact 229

H : Cartridge Ports **230**
I : Compact Analogue Emulator **234**
J : Connector Pinouts **241**
K : The Programs Disc **245**
L : Guide to Dabhand Guides **247**
Glossary of Terms **252**
Bibliography **259**
Index **260**

1 : Introduction

A History

When Acorn launched the BBC Microcomputer back in June 1981, they said that it was a design to last five years — a very optimistic target for a personal computer then and now for that matter. The fact that at the time of writing Acorn have been producing and selling the BBC micro for six-and-a-half years, with no abatement of interest in sight, is a remarkable tribute to the quality of the machine. Consistently praised over the years has been the excellent Machine Operating System (MOS) which unlike those on many other systems is both comprehensive, and easy to use from any language.

Acorn have always guaranteed compatibility across their product range - if programs use only those operating system routines documented by them, and do not either directly call routines in the MOS ROM, other than those documented, or access hardware directly, then they can be sure that their programs will work on future machines. Acorn have held good to this promise. Programs written in 1981 and 1982 which assiduously followed the system routine rules, can still be run on machines such as the Master Compact produced years later, without any alteration.

Eighteen months after the original BBC Models A and B, the Electron was launched as a smaller version of the Model B but without most of its expansion capabilities. Shortly after came the 6502 and Z80 second processors. By this time, it was becoming clear that compatibility and programming 'legally' (ie, using only documented system routines) were of great importance, if you wanted your programs to run across the whole range of machines.

The machine base diverged further when the BBC B+ was released in June 1985. This 64k machine was rapidly followed three months later by a 128k version in September — the BBC B+128. The B+ was similar to a Model B — in fact it was initially known as the Issue 10 BBC Micro referring to the PCB version, and compatibility problems were not severe.

January 1986 proved to be a significant month with the launch of the Master Series computers - the 128, the ET (Econet Terminal) and the various co-processor units. This was followed by the Master Compact in September 1986 , Acorn's first bundled system which also set a new trend by doing away with the standard 5.25" DFS based discs and progressing onto 3.5" ADFS format discs.

To date then there are five different base machines most of which can be enhanced or extended with a variety of second and co-processors, the options open to the programmer are enormously wide and varied. This book is aimed at the 'post-Model B' world. That is not to say that Model B owners will not benefit from it, but for B+, Master and Compact users, I hope it will become your standard reference work. Of course, whatever model you are writing software on, it only makes sense to try and make your programs run on every type of machine. This book aims to do just that by providing all the information on all aspects of the MOS for the post-B micros under one cover, something that has not been done before.

Chapters 2, 3, 4, 7 and 8 of the book cover all the Operating System commands, and also include the new 65C02 opcodes. These are standard reference sections to add to the documentation for Model B supplied in the original User Guide, and the Advanced User Guide. For reasons of space, the calls such as OSBYTE and OSWORD which are identical on all Acorn machines are not documented. Details of these should, however be found in the User Guide for your machine.

Chapters 5 and 6 take their topics further than pure reference material. These are areas where there is a lot of interest, so the text has been expanded to include examples showing you how to use Shadow and Sideways RAM, and how to write ROM format software. There is also a lot of previously unpublished information such as Tube interface documentation, Master Sideways ROM calls, and Compact technical information. Chapter 9 contains full details of the configuration bytes used in the Master CMOS RAM and EEPROM in the Master Compact, while Chapter 10 contains a list of all the differences between the various machines, both in hardware and software.

When writing programs, most people can remember the basic principles but can't commit to memory huge tables of numbers. The Appendices are extensive as needs be to encompass them and include all the OSBYTEs, OSWORDs, VDU calls, memory maps, and so on.

Throughout this book you will see references such as 'B/B+/E' or 'M/C' or 'All'. This is to show you upon which machines the items exist. B is the BBC Model B, B+ the BBC B+, E is the Electron, M the Master 128, and C, the Master Compact. Any other machines are specifically identified.

Type Styles

Certain typestyles have been used to denote specific things. When an actual key is being described, a special face is used, like this:
ESCAPE or SHIFT-BREAK.

Program listings are done in this typeface.

```
10 PRINT "A Program Listing"
```

Where you have to type something into the computer, it is always given in this typeface:

```
*INFO *.*
```

The only other face used is this one for screen output

```
Acorn MOS
```

which is deliberately small to fit 80 columns. Of course there are particular styles, such as large characters for the section headings, the use of italics for proper names such as *Acorn User* and bold text for the occasional **emphasis**. One final note, the double-bar character which appears either as a broken line or pair of lines on the BBC screen is a single vertical bar here thus |

Acknowledgements

Thanks to Bruce Smith, colleague and friend, for getting me started writing for *Acorn User* which in a roundabout way led to this book, and to Tony Quinn for publishing my articles. An enormous thank-you to David Spencer, without whom this book would never have been finished. David knows more about the BBC Micro than anyone I know, and picked up all the errors from the first edition. Sue Wall and David Bell at Acorn have always proved very helpful and deserve appreciation. Paul Horrell edited the text, Paul Holmes designed the cover, and Graham Bartram strongly influenced the typography. Thanks to all of them. Same for my former colleagues from the BBC who gave me encouragement whilst I was there. And finally a big hand for Lynda and Susan.

About this book

This book is the second title from Dabs Press, published using the latest desktop publishing and laser printing techniques. The book was written on a BBC Master 128 using VIEW, and many programming tools, particularly the excellent *Advanced Disc Toolkit* and *Advanced Disc Investigator* produced by Advanced Computer Products, and our own *Fingerprint*. The finished text was transferred using BBC Soft's *Modem Master* to an Apple Macintosh SE, using the Red Ryder shareware comms software at the other end, and a very expensive cable! On the Mac, the text was processed by *MacAuthor*, one of the few British desktop publishing programs, and output to an Apple Laserwriter. The laser printed pages were sent to the printer, A Wheaton and Co., where they were printed and bound in the normal way.

Thank you …

... for buying /borrowing/stealing this book, we hope you enjoy it and that it finds a regular place on your computer's lid! If you have any comments or suggestions to make, we'd love to hear from you. Our address is on page 2, and we're on Telecom Gold on 72:MAG11596, or Prestel 942876210. Letters to the author should be sent in the same way, and although we cannot promise all letters will be personally answered we'll try our very best.

Correspondents and mail order purchasers will be automatically advised of future Dabs product releases, unless they request otherwise. Personal details held will be provided on request in accordance with the Data Protection Act.

The Second Edition

It's an unexpected pleasure to reach the second edition of your first book! Commercially, thanks must go to our good friends at Computer Bookshops and Acorn User Merchandising, but most of all thanks to you the great British (European and worldwide) public, for your valued support. I am indebted to Sean McGoogan of Prestwick in Scotland, Andrew Benham of Southgate in London and the ubiquitous David Spencer for finding the errors in the first edition, which have been corrected here, and also to other readers too numerous to mention for their valuable suggestions — I have tried to incorporate them all.

David Atherton, Manchester, November 1987

2 : MOS Commands

This chapter contains a comprehensive list of the star (*) commands which can be executed directly from the keyboard or placed in programs. The commands can be used from within any programming language on the machine, including a NLE (no-language environment) such as CLI (command line interpreter) or from within machine code applications. Commands are executed by setting X to the low byte and Y to the high byte of an address which containing the command as an ASCII string terminated by carriage return (&0D). The comand is executed by a call to OSCLI (&FFF7). No accumulator setting is required. On non-6502 co-processors the X and Y registers are replaced by other suitable registers, for example HL on the Z80. Listing 2.1 shows how OSCLI can be used to execute the command *CAT from within a machine code program.

Listing 2.1.

```
 10 REM OSCLI in assembler
 20 REM (c) Dave Atherton 1987
 30 REM for B+/M/C
 40 REM MOS : A Dabhand Guide
 50 :
 60 DIM M% &100
 70 FOR pass=0 TO 2 STEP 2
 80 P%=M%
 90 [OPT pass
100 LDX #string MOD 256
110 LDY #string DIV 256
120 JSR &FFF7
130 RTS
140 .string
150 EQUS "*CAT"
160 EQUB 13
170 ]NEXT
180 CALL M%
```

For the sake of completeness, all the commands from the Acorn filing systems and other Acorn firmware are included. The machine list reflects this, and only the machines where the command is present as standard are listed. Where the command is documented in existing User Guides, the description here concentrates on differences on the

B+ and Master Series. For the purposes of this list, it is considered standard to have a disc interface fitted to the BBC Micro and Acorn Electron (Plus 3). Obviously upgrades can supply the other functions where they are missing - the Model B+ can be fitted with ADFS, the Model B can have a 1770 DFS fitted, while third party products supply the missing filing system commands.

Parameters to these calls should be entered as follows. On MOS 2.00 or earlier, 8-bit numbers must be entered in decimal and 16-bit addresses in hex, Exceptions to this are documented. On MOS 3.00 onwards, 8-bit parameters can also be entered in hex by using a preceding ampersand so that, for example, *FX &9C,&8,227 and *FX 156,8,&E3 are identical. Omitted numeric parameters generally default to 0 any exceptions are noted. When a MOS command expects an 8-bit parameter, giving a number above 255 or &FF will usually generate a 'Bad command' error. Parameters should be separated by one or more spaces, although in some circumstances (ie, *FX) a comma may also be used.

If MOS numeric parameters are not 8-bit, then they are 32 bit and will be expected in hex without an ampersand, for example:

```
*LOAD file FFFF1900
```

Leading zeros can be omitted as can the top 16 bits (the FFFF part in the example above) if a single processor is being used, ie, a co- or second processor is not fitted or switched on.

String parameters may be entered with or without quotation marks, but generally, failing to use quotes prevents use of the space character. Therefore in DFS, typing:

```
*TITLE "My Disc"
```

will work correctly, but

```
*TITLE My Disc
```

will title the disc as My. Quotes themselves are entered by using two quotes characters, so

```
*KEY0"LOAD ""PROG"""
```

will set the key up with LOAD "PROG". Note that |" also causes a literal quote and

```
*KEY0"LOAD |"PROG|""
```

is identical to the previous command.

All MOS commands can be abbreviated, by typing only a few letters of the command, followed by a period or full stop. The minimum abbreviation needed depends entirely on which configuration of machine you have. The MOS has first claim on all '*' commands, then each of the sideways ROMs from 15 to 0, then the current filing system, and finally the library FS. The first ROM to identify the abbreviation acts upon it. As you can't guarantee that an abbreviation will pick up the command you want, you should only use them in direct mode, and always include the full command in programs. Many people have come unstuck using *D. for *DISC and *A. for *ACCESS when these have been picked up by third party ROMs.

The commands themselves take optional parameters in two ways. Firstly, many commands take a default value if no or incomplete parameters are supplied. This is to make the commands simpler to novice users, eg *VERIFY. Some commands allow the use of wildcards to specify ambiguous names, the filing system commands in particular use these. Where a command uses an <afsp>, then * can be entered as 'any number of characters' and # can be entered as 'any character', thus *INFO * will provide information on all files in the CSD, (Currently Selected Directory) whereas *INFO #### will provide information on CSD files with names that are exactly four characters in length. *INFO ####W will list files of exactly five characters, where the fifth character is W.

Again it is not recommended that you omit parameters in your programs - if nothing else, inclusion of parameters aids clarity in the listing. Using wildcards is fine, but many of the commands that use these, such as *WIPE and *DESTROY are not really suitable for inclusion in a user program anyway.

The A to Z of * Commands

Each of the '*' commands are discussed briefly on the following pages. Commands are arranged in alphabetical order.

*** \| <text>** **MOS** **All**

This is not really a MOS command. It is more the equivalent of a REM in BASIC, and is used for comments. Anything following *| up to the carriage return is totally ignored by the MOS. An example might be a file to load in a ROM image:

```
*DIR $.ROMS
*SRLOAD utils 8000 W Q
*|
*| Now press CTRL-BREAK
*| to initialise this ROM
*|
```

The last four lines have no effect but are useful in providing a way of documenting an EXEC file of star commands.

***ADFS** **ADFS** **B+/E/M/C**
Selects the Advanced Disc FS, and tries to load the root directory.

***ACCESS <afsp> <attr>** **DFS/ADFS/Net** **All**
Sets the attributes of a file so that it can be read from (R), written to (W), and deleted, or protected from any or all of these. For Net the settings applicable to 'you' (a user looking at his own files) and for 'others' (a user looking at someone elses files) may be different, eg,

```
*ACCESS file LWR/R
```

would means that the creator could read and write to the file (but not delete it - L) but others could only read it. DFS allows file locking (L) only. ADFS allows read/write and lock and a special 'E' setting meaning that the file can only be *RUN. Once the E attribute has been set, it cannot be erased.

***APPEND <fsp>** **MOS** **M/C**
This is like *BUILD but instead of creating a new file, the existing file <fsp> is opened and the text you type is added to the end of this file. Like the *BUILD command, ESCAPE will terminate entry. If <fsp> is not present, a 'Not found' error results.

***BACK** **ADFS** **B+/E/M/C**
This ADFS command reselects the last directory you were in, ie, the CSD before the last change of directory. If you have just selected ADFS, or select the CSD twice, *BACK has no effect.

***BACKUP <src> <dest>** **DFS/ADFS** **B/B+/M/C**
Make a sector-for-sector copy of one disc surface onto another. The number of sectors actually copied is determined by the 'size' bytes in

the disc space map. No account is taken of the actual files on the disc. In ADFS, this is only implemented from version 2.00 onwards.

***BASIC (<@>)** **MOS** **All**

This command starts up the BASIC language. Of course all languages have a similar command (*WORD, *PASCAL etc) but these languages must use service code 4 to decode their startup command. BASIC does not need to do this - the MOS will do it automatically. Note that on BASIC 4 upwards, adding '@' as a parameter to the command will cause a text block to be read in and tokenised. The address of the block is determined by the contents of memory locations &00 and &01 on the language processor. The user of the '@' call is also responsible for setting up these bytes.

***BUILD <fsp>** **MOS or FS** **All**

Create a file on disc. The file <fsp> is opened using OSFIND, and a line editor, prompted by line numbers is entered. Text is accepted from the current input stream, including carriage returns, but not ASCII codes 0-31. ESCAPE terminates text entry, and closes the file.

***BYE** **ADFS/Net** **E/M/C**

This command closes all open files, and is used when you finish working on the current filing system. On NFS, the command also removes the user from the list of active users. If the maximum users as configured by fileserver setup are logged on to the network, this command is needed to allow another user to log on. On ADFS Winchester hard disc based systems, the command 'parks' the read/write head safely away from the hard disc surface. A *BYE command should always be issued before physically moving the Winchester unit.

***CAT (<drv|dir>)** **MOS** **All**

Gives a list of the files on the current drive (DFS), or directory (ADFS/network), or other medium (tape/ROM). There is no standard form of layout although DFS/ADFS/network and tape/ROM layouts are very similar. This command abbreviates to *.

***CDIR <dir>** **ADFS/Net** **B+/E/M/C**

This command is only relevant to the hierarchical filing systems ie, ADFS and Net. It creates a new directory, which can then be selected

by *DIR <dir>. The directory is empty (it contains no files) to begin with, has a title string identical to the directory name, and has a cycle number of 0. ADFS directories can contain 47 objects (files or subdirectories). Net directories can contain 244 objects.

***CLOSE** **MOS or FS** **B+/E/M/C**
This issues a call to OSFIND with A=0, and Y=0. The command closes all open files (in this filing system — see *SHUT). Before this, it updates to media: any bytes still in buffers and not actually transferred to disc etc, are transferred, and file lengths are adjusted accordingly.

***CODE (<X>) (<Y>)** **MOS** **All**
This command, identical to OSBYTE &88, places the two parameters into the X and Y registers, and jumps to the routine whose address is contained in USERV (&200/&201 on the I/O processor). Because the call is always to the I/O processor, the command is a useful way of starting I/O machine code programs from the co-processor, especially on pre-Master Series machines which do not have the *GOIO command. By default USERV points to a 'Bad command' BRK error, so until USERV has been modified, *CODE appears to have this effect.

***COMPACT <params>** **DFS/ADFS** **All**
This puts all the files on the disc onto contiguous sectors, thus making all the free space contiguous also. On DFS, the optional parameter is a single drive number 0-3. All the memory between OSHWM and HIMEM is used as a buffer. On ADFS, there are two parameters, SP and LP, denoting the start memory page and number of memory pages (in hex) required as a buffer during compaction. On a Master or Compact

```
*COMPACT 0E 72
```

performs the quickest compaction, reserving 72 pages from &0E00 (ie to &7FFF). During *COMPACT on both filing systems the disc is sometimes in an inconsistent state — so never abort a compaction. ADFS 2.00 upwards will accept the command without parameters, supplying a default of 0E 72.

***CONFIGURE <params>** **MOS** **M/C**
This writes to the MOS portion of the CMOS RAM on the Master 128, or the Compact's EEPROM. The purpose is a permanent machine setting

of a system variable, ie, the default screen mode. The first parameter is a text description, followed by 0 or more parameters. Ie,

```
*CONFIGURE MODE 135
```

will select the Teletext screen in shadow mode, as a default. The *CONFIGURE system is dealt with in more detail in Chapter 5.

***COPY <params>** **DFS/ADFS** **All**
This command has a different syntax on DFS and ADFS. On DFS, the syntax is <src drive> <dest drive> <afsp> which will make a copy on the destination drive of the file(s) <afsp> currently on the source drive, for example,

```
*COPY 0 1 V.Text
```

copies the file Text in directory V to Drive 1. On ADFS, the first parameter is the source file, and the second is the destination directory. The command therefore copies files to another directory on the same disc (*COPY V.Text $.W will duplicate 'Text' in directory V as 'Text' in directory W) or to another disc (*COPY :0.$.Source :1.$) copies 'Source' from Drive 0 to Drive 1, using only the $ directory on each. Wildcards are accepted in DFS and ADFS

***CREATE <fsp> <len>** **MOS** **M/C**
This creates a file <fsp> of <len> bytes in the current directory. The file is created using OSFILE with A=7. Only Master Series machines offer the *CREATE and the new OSFILE call.

***DELETE <fsp>** **MOS or FS** **All**
This deletes a filename from the catalogue. The file must not be locked. The actual file data stays on the disc until the space it occupies is needed by another SAVE activity. However, there is no simple way of recovering a deleted file, and no warning prompt is given.
NB: In ADFS/Net a directory can't be deleted unless it is empty.

***DESTROY <afsp>** **DFS/ADFS** **All**
This is the 'wildcard' version of *DELETE. All files which match the <afsp> are listed and a single Yes/No prompt is issued. Typing Y on DFS, or YES <RETURN> on ADFS will delete all the files listed. ADFS performs the function by individually *DELETEing each file, and

thus takes a long time. Note that under ADFS the command can only destroy files in the CSD and hierarchically inferior directories.

***DIR (<dir>)** **DFS/ADFS/Net** **All**
This command will select the directory specified

DFS: Followed by a single character directory name. *DIR $ resets the root directory. To reset Drive 0 as well *DIR :0.$ is required. Note that *DIR will retain the CSD, but on DFS 1.20 selects Drive 0.

ADFS: Directory names, like filenames can be up to ten characters. Also the special character ^ means 'up a level', *not* 'back a level', and commands like:

```
*DIR ^.Fred
```

are not unusual. *DIR $ reselects the root directory. *DIR & and without a parameter reselects the user's root directory, which on ADFS is always $, but on Net may not be - using *DIR & gives Net compatibility.

Net: Is similar to ADFS, except that *DIR and *DIR & select the user's own directory, which, if he is not a system user, will not be $. Acorn Econet does not support the '^' concept at all, although this is supported by competitive products.

***DISC** **DFS** **B/B+/M/C**
This command selects the Disc Filing System (DFS). The command itself is implemented in the DFS. *DISK is also acceptable.

***DISMOUNT <drv>** **ADFS** **E/M/C**
This is the opposite of *MOUNT, and deselects use of an ADFS drive. It should be used prior to *MOUNTing a new drive, although in practice this is not usually necessary. However, the ADFS map sometimes demands it, generating a 'Disc changed' error when you try to *MOUNT or *DIR the new drive.

***DRIVE <drv>** **DFS/ADFS** **B/B+/M/C**
This is a DFS command and is used to select one of the four floppy disc surfaces (0-3) for read/write accesses. It is implemented in ADFS 2.00 onwards for compatibility and is translated to *DIR :<drv>.$

***DUMP <fsp> (<sta> <ofs>)** **MOS or FS** **All**
This sends to the current output stream a hex and ASCII dump of the file <fsp>, without corrupting user memory. The dump has lines of a relative hex address (4 digits), eight hex bytes (2 digits) and eight ASCII characters. Characters below 32 or above 126 are shown as dots.

The start and offset parameters, if used, dump the file starting at byte <start>, using <offset> to determine the relative address displayed. These parameters both default to zero, and are not available on the Model B and Electron. On the Electron *DUMP is provided as a disc-based utility.

***ENABLE** **DFS** **All**
This command does nothing except set a flag in memory. Certain dangerous DFS commands will test this flag (at &FFFF10C8 on DFS 1.20) and if the previous command was not *ENABLE, these commands work differently. Every MOS command issues a call to OSFSC with A=8, and DFS uses this to work out if there were any MOS calls between *ENABLE and the dangerous command.

***EX <adir>** **MOS or FS** **B+/M/C**
This will show the same display as *INFO for a whole directory. If used without parameters, the directory displayed is the CSD. On ADFS *INFO causes the directory to be reloaded, whereas *EX (on the CSD) does not. Under CFS/RFS any parameters given are ignored.

***EXEC <fsp>** **MOS** **All**
This opens the file <fsp> and sends all bytes to the current input stream, having the effect of 'typing' the file. No key expansions are performed. The MOS will automatically close the file when all bytes have been read or an ESCAPE condition occurs.

***FADFS** **MOS** **B+/E/M/C**
As *ADFS, this selects the Advanced Disc Filing System, but does not try to load a directory. *FADFS is invaluable because (a) without this command, ADFS formatters could not start as the disc in the drive could not be mounted, and (b) the disc may not be in Drive A, which is always assumed by *ADFS.

***FORM <40|80> (<drv>)** **DFS** **B+/M**

From DFS 2.00 onwards, there is a ROM resident disc formatting command, in earlier versions it is supplied as a disc based command. The first parameter is 40 or 80 depending how many tracks you wish to be formatted, although any number between 0 and 255 is accepted, and the correct free space is calculated. This is useful if you are duplicating discs, as it makes *BACKUP quicker if you don't format or copy unnecessary tracks. Note that if you specify 40 on an 80 track drive, you will get a 40 track disc, but at 80 track density — meaning that it can only be used on an 80-track drive. The second DFS parameter is the drive number(s). Using *FORM 80 0213 will first format the disc in Drive 0 (both sides) then the disc in drive 1 (both sides).

***FORMAT <drv> <size>** **ADFS** **C**

In ADFS 2.00 a formatting command has been introduced. The first parameter is the drive number. On the Compact this will always be 0 or 1, but could be higher when ADFS 2.00 is used on a Master 128. The second parameter is S, M or L. S, or small, means a single sided 40 track ADFS format (160k), M, or medium, a single sided 80 track format (320k), and L means a double sided 80 track format (640k). Standard Compact discs use the M format, although most users will format data discs as 'L'. *FORMAT is available on the Master 128 in MOS 3.21 onwards. *FORMAT is a disc-based utility on the Electron.

***FREE (<drv>)** **DFS/ADFS/Net** **B+/E/M/C**

This shows the number of bytes free (in decimal) on this device, and for DFS/ADFS the number of sectors free (in hex). Additionally for DFS, the number of filenames free is shown. Conversely, the number of bytes, sectors, and filenames used is shown - so this command can also quickly tell you the size of a disc.

***FX <A>[,<X>[,<Y>]]** **MOS** **All**

This provides keyboard access to those OSBYTE routines which perform actions, rather than giving results. ie, to speed up the keyboard repeat speed, type *FX12,4. On the Master series only, the parameters can optionally be given in hex — see the start of this chapter.

***GO (<addr>)** **MOS** **B/B+/M/C**
This command is supplied in the Tube MOS with Models B and B+, and in the MOS on the Master Series. It simply starts a machine code routine at <addr>. Only the bottom 16 bits are regarded, so *GO FFFF2000 would still start code at 2000 on the language processor, if a second processor is active. No checking is made as to whether the bytes at <addr> are valid machine code.

The command can also be used to drop into the CLI. This is done by typing no parameter on an I/O machine, or *GO F800 on a 6502 second processor. A '*' is prompted and anything typed is acted on as a '*' command. This provides a useful interface whereby *LOAD and other memory-altering commands can be issued, when all of the processor workspace is required. On a 6502 second processor, entering CLI provides 61k of user RAM (&0400 to &F7FF).

***GOIO <addr>** **MOS** **M/C**
This command performs as *GO, but in a two processor environment, will always call an address on the I/O processor.

***HELP (<subject>)** **MOS** **All**
This command shows useful text about the firmware fitted to the machine. When typed without a parameter, it will show names and version numbers of the firmware fitted, plus any words that can be used as <subject> in the above. The actual text supplied when a parameter is given depends entirely on the firmware. A call (OSBYTE &8F with X=9) is passed by the MOS to each ROM, when a *HELP command is issued, and it depends entirely on the ROM what action is taken. Note that the built-in help on DFS and ADFS will not spool to a disc file, although they will go to the printer.

***IGNORE (<n>)** **MOS** **M/C**
When used with a parameter, this is similar to OSBYTE 6, which sets the character for printer to ignore. However *IGNORE can be issued without a parameter, resulting in the MOS sending all characters to the printer. It is not possible on Models B and B+ or the Electron to have no ignore character.

***INFO <afsp>** **MOS** **All**
This command will show the name, load address, execution address, and length of the file or files specified. Also shown is the relative sector address where the file is stored. All parameters are shown in hex. The command is implemented as a filing system command on machines prior to the Master. It is implemented in the MOS on the Master series, but does not work in the Cassette (CFS) and ROM (RFS) Filing Systems.

***INSERT <n>** **MOS** **M/C**
This will reinstate a previously unplugged ROM at the next hard reset. See *UNPLUG. The *ROMS command shows which, if any ROMs are unplugged.

***KEY <n> (<string>)** **MOS** **All**
This will assign <string> to red function key <n>. When the function keys are pressed (if enabled - see OSBYTE &E1-&E4) it will be as if <string> had been typed. See also *SHOW

***-<fsname>-LIBFS** **MOS** **M/C**
This command allows a second library to be available which is not in the same filing system as that currently in use. The library offered is the CSD, followed by the Currently Selected Library — CSL (see *LIB for library order) of the other FS, which must of course be present. An example would be to supply a network utility library to Disc/ ADFS users. The syntax is *-<fsname>-LIBFS, for example *-NET-LIBFS. *LIBFS alone cancels this effect, as it effectively says the current FS is the library FS.

***LCAT** **ADFS/Net** **E/M/C**
This performs a *CAT on the CSL. The CSD is not changed. On the Net it is implemented as a library file.

***LEX** **ADFS/Net** **E/M/C**
This performs a *EX command on the CSL. The CSD is not changed. On the Net it is implemented as a library file.

***LIB (<library>)** **DFS/ADFS/Net** **All**

This selects a library, the CSL. A library is like a directory for machine code programs. If a machine code program is *RUN (or */ or *<filename> is used), the CSD is searched first for the file. If it is not there, the CSL is searched, followed by any secondary library (secondary CSD and secondary CSL — see *LIBFS). The system allows you to have a number of machine code programs online, without cluttering up the CSD, and accessible whichever directory you are in. The system is not really necessary in DFS, which is probably why it has not been used much! In ADFS and Net however, it can be very useful. EXEC files can also be held in the library because ADFS/Net and DFS 2.00 onwards will *EXEC any file that you try to *RUN if the execution address is &FFFFFFFF.

***LIST <fsp>** **MOS or FS** **B/B+/E/M/C**

This is similar to *TYPE, except that line numbers are placed at the beginning of each line. Where *TYPE deviates on Master series machines, *LIST also deviates.

***LINE (<string>)** **MOS** **All**

This command places the start address of the string parameter in the X (lo) and Y (hi) registers, sets A=1 and jumps to the I/O processor address pointed to by USERV (&200/&201). The string is terminated in memory by a byte containing 13 (&0D). Like *CODE, you will get a 'Bad command' error until you modify USERV.

***LOAD <fsp> (<addr>)** **All FS** **All**

This command transfers a file directly into memory. The optional address is 32 bits in length, and thus loading can be directed into I/O or language processor. The address is the start point from where the file will be loaded. If the address is omitted the file will be loaded to the load address given in the catalogue or header (CFS/RFS). No checking is done of the validity of the load address. Once loading commences 64k wraparound takes place if necessary.

***MAP (<drv>)** **DFS/ADFS** **B+/E/M/C**

This command in DFS 2.00 onwards and ADFS shows the free space(s) on a disc. Each free area is listed, with start sector and number of sectors

free, in hex. A fully compacted disc should only display a single pair of numbers.

***MOTOR (<n>)** **MOS** **B/B+/E/M**
This command, identical to OSBYTE &8C, switches the cassette motor relay. n=0 or no parameter turns the motor off. n=1 to n=255 turns it on. n>255 causes a 'Bad command' error. On the Compact, the command doesn't cause an error, and has no effect.

***MOUNT (<drv>)** **ADFS** **B+/E/M/C**
This transfers the catalogue of an ADFS disc into memory, and selects directory $. It is inherently performed by the *ADFS command (but not by the *FADFS command). It should be used each time a new disc is placed in the ADFS drive. If no parameter is given, the last selected drive is reMOUNTed.

***MOVE <-fs-fsp> <-fs-fsp>** **MOS** **M/C**
This command will move a single file from one filing system to another. Typical syntax might be:

```
*MOVE -ADFS-:1.$.Letters.Jack -DISC-:0.L.JackLtr
```

If a full pathname is not given for ADFS/Network, the CSD only is searched. If a FS name is not given, the current FS is assumed. The file must exist on the source FS, any destination subdirectories must exist on the destination FS and both source and destination FS must be selectable with their standard command, which means that their workspace memory may have to be cleared. (*FX200,3 then BREAK will usually suffice). Note that *MOVE without the different filing systems acts as a 'copy' or 'duplicate' command so that in DFS:

```
*COPY 0 1 FRED
```

performs the same function as:

```
*MOVE :0.FRED :1.FRED
```

Note that unlike *COPY, *MOVE does not take any user memory, instead using the buffers in HAZEL, and if available, shadow screen memory. Thus *MOVE runs best in non-shadow modes, and worst of all in Modes 128-130.

The CSD, if expressed, may be abbreviated to @ in common with other Master Series filing system commands. If in the example

above, you were already in ADFS, Drive 1, directory Letters, you could have typed @.Jack as the first argument.

***OPT (<n>) (<n>)** **MOS** **All**
Identical to OSBYTE &8B, this sets various filing system options. See the original User Guide for full details.

***PRINT <fsp>** **MOS** **M/C**
This command sends the contents of a file direct to the VDU in a similar manner to *TYPE, except that it actions control codes and top-bit-set characters rather than showing them in GSREAD format. It is thus identical to *TYPE on machines prior to the Master 128, except for this: the old *TYPE dealt with carriage returns through the OSASCI routine, causing double-line spacing if the *TYPEd file contains linefeed characters. *PRINT is directed through OSWRCH, printing raw ASCII data.This will affect any applications which count characters being put through the VDU drivers, and also the appearance of text being *TYPEd out by programs ie, it is not sufficient just to change *TYPE to *PRINT in your programs.

***REMOVE <fsp>** **MOS** **M/C**
This is identical to *DELETE except that it does not generate an error when the file to be removed is absent. In MOS 3.21 and 5.10, a safety device is included. If any text follows the <fsp>, a 'Bad command' is generated. This avoids:

```
*REM. file1 file2
```

where you meant to type *REN. for *RENAME.

***RENAME <fsp> <new fsp>** **DFS/ADFS/Net** **All**
This alters the name of a file or directory (which must be unlocked, but need not have write access). On ADFS the command is also used to move a file to another directory. This is achieved merely by renaming it using the new directory pathname.

***ROM** **MOS** **All**
This selects the ROM filing system (RFS) as the current FS.

***ROMS** **MOS** **B+/M/C**

This lists the names and binary version numbers of the contents of each ROM slot, which may be a ROM, EPROM or a ROM image in sideways RAM. Slots where the '(C)' identifier cannot be found are denoted as '?' On MOS 3.21/5.10 and higher, duplicate ROMs are not shown (the first 256 bytes of data are used to perform comparisons) and the RAM slots bear the message RAM, not ROM. On the Model B+, the layout of the command is slightly different - the ASCII version no, not the binary is shown

```
Rom 08 : ( L) ADT 1.20 (Model B+ layout)
ROM 6 ADT 00 unplugged (Master/Compact layout)
```

A bracketed pair of letters indicates whether the ROM is a language (L) or a service (S) type ROM.

On the Model B+ layout, an empty socket is omitted, thus a display may have less than 16 items. On the Master layout all sockets are shown. Empty or duplicate slots are shown as 'ROM n ?' followed by 'unplugged' where appropriate.

***RUN <fsp>** **MOS** **All**

This command performs a *LOAD of the file, without a parameter, ie, it loads the file to the load address in the catalogue/file header, and then jumps to the file's execution address (which may be on I/O or language processor).

***SAVE <params>** **MOS** **All**

*SAVE transfers a contiguous block of memory to to file. The parameters of the *SAVE command are: <fsp> <start address> <end address|length> (<optional execution address>) (<optional reload address>), ie:

```
*SAVE File2 FFFF0E000 +100 FFFF0E01
```

would save a block of memory from &E00 to &EFF on the I/O processor. The execution address is stored as &E01 located on the I/O Processor. Note that the end address quoted will always be one higher than the last byte saved. ie,

```
*SAVE Block 1000 2000
```

would save byte &1FFF but not &2000.

The memory transferred can be defined by start and end addresses, or by start plus length. The third optional parameter is the execution address. If not given, this defaults to the start address. The fourth optional parameter is the reload address. If not given, this too defaults to the start address.

***SHADOW (<n>) MOS B+/M/C**
This command performs OSBYTE &72, with <n> in the X register. *SHADOW or *SHADOW 0 cause the shadow screen to be selected for display and writing by VDU code (not direct access) at the next mode (VDU 22) change, irrespective of whether or not the mode number has the top bit set. SHADOW <1-255> causes main or shadow memory to be selected as appropriate according to the top bit of the mode number following VDU 22. When a *SHADOW 0 is in force, it is impossible to select a non-shadow screen mode using VDU 22 followed by a top-bit-clear number. The default condition is *SHADOW 1, even on a Master 128 or Compact set to start in a shadow mode.

***SHOW (<n>) MOS M/C**
This command lists any strings programmed on the function keys. The Master 128 MOS 3.20 must take a parameter and lists the contents key <n>. MOS 3.21/5.10 will work without a parameter and list all the keys contents. The single key display just prints the string in GSREAD format, enclosed in quotes. The full display gives, for each key, the text “Key” followed by the number in hex, followed by the text of the string in the same format.

***SHUT MOS M/C**
This command is similar to *CLOSE except that it closes all open files in all Filing Systems, whereas *CLOSE only closes files in the current Filing System.

***SPOOL (<fsp>) MOS All**
This opens a file (using OSFIND A=&80) and then sends all VDU driver output to the file specified. The output continues until the file is closed. Closure is acheived by typing *SPOOL or *SPOOLON without a parameter, which would then form the last few characters of the file.

***SPOOLON (<fsp>)** **MOS** **M/C**
This opens an existing file (using OSFIND A=&C0) and then works exactly like *SPOOL. A 'Bad command' error (not 'Not found') will occur if the existing file is absent.

***SRDATA <id>** **MOS** **B+/M/C**
This determines that the RAM bank <id> will be addressed using the pseudo-addressing system. There is no equivalent system routine in assembly language. Note that to obtain more than 16k of pseudo-addressable sideways RAM, the banks defined by SRDATA must be adjacent. Ie:

```
*SRDATA W
*SRDATA Y
*SRROM X
*SRROM Z
```

will only provide 16k of pseudo-addressable sideways RAM, as the second bank (X) is not defined for pseudo-addressing. See Chapter 5 for more details of these SR commands. In the B+ and Master the SR commands are coded in the DFS ROM, so if this ROM is UNPLUGged, the commands will not work.

***SRLOAD <fsp> <addr> (<id>) (Q) (I)** **MOS** **B+/M/C**
This command transfers a file <fsp> into sideways RAM at <addr>. If pseudo-addressing is in use, then <addr> starts at 0, and the <id> must be omitted. If absolute addressing is used, the <id> is mandatory, and the lowest <addr> usable is &8000. The optional Q parameter speeds up loading (by using OSFILE instead of OSBGET) but uses I/O main memory as a buffer, from OSHWM onwards. This will corrupt data in memory (unless you are using a co-processor).

On MOS 3.21/5.10 and higher, an additional parameter 'I' may be added after or instead of the 'Q'. After the load has happened, and provided it happens successfully, this 'I' causes a byte in I/O location (&2A1 + n) to be set to the value of the byte at &8006 in the sideways bank n, where n is the bank specified in the *SRLOAD command. This only has any meaning where the file being loaded is a sideways ROM image. It has the effect of 'initialising' the ROM image, making it able to accept service calls and hence commands. It does not however cause a workspace claim to take place, so it is unsuitable as a method of initialising ROMs which need to claim workspace.

***SRREAD <addr> <addr> <addr> (<id>) MOS B+/M/C**
This transfers data from sideways RAM to main memory (including Tube memory). The first parameter is the main memory start address (32 bits). The second is the end address. Alternatively, the number of bytes to transfer can be given preceded by a '+'. Thus:

```
*SRREAD FFFF2000 FFFF20FF 8000 W
```

has the same effect as

```
*SRREAD FFFF2000+FF 8000 W
```

NB: The Sideways RAM start address does not need to be in the range &8000-&BFFF, and if used with, say, lower values, the call can be used as a straight memory shifter. A bonus here is that Tube transfer is automatic. Use *SRREAD to transfer data to the Tube, and *SRWRITE to transfer from Tube to I/O processor.

***SRROM <id> MOS B+/M/C**
This determines that the RAM bank <id> will be addressed using the absolute addressing system. There is no direct assembler system equivalent. Note that this command is not necessary if you are loading a ROM image with a command such as:

```
*SRLOAD Database 8000 W Q
```

unless the bank (W in this example) has previously been allocated to the pseudo-system with *SRDATA.

***SRSAVE <fsp> <addr> <addr> (<id>) (Q) MOS B+/M/C**
This command saves sideways RAM memory contents to disc. The first parameter is the file to save to, the second and third are start and end (or + the length) of sideways RAM, in absolute or pseudo range. The optional <id> parameter determines whether the sideways address parameters are absolute or relative. As in *SRLOAD, the Q parameter causes a faster transfer, but again at the overhead of corrupting main memory.

***SRWRITE <addr> <addr> <addr> (<id>) (I) MOS B+/M/C**
This transfers data to sideways RAM to main memory (including Tube memory). The first parameter is the main memory start address (32 bits). As with the other commands, the second parameter is the end address or + the length. The third parameter is the start of sideways

RAM, in absolute or pseudo-address terms, and the optional <id> specifies which RAM bank to write to.
The 'I' option is supported on MOS 3.21/5.10. See *SRLOAD.

***STATUS (<param>)** **MOS** **M/C**
This command displays the settings of the various operating system variables stored in CMOS RAM/EEPROM. On MOS 3.20, the settings are displayed in the order they appear in the MOS. On MOS 3.21 and 5.10 they are displayed in alphabetical order. However, any additional items added by sideways ROMs responding to service call &41 (see Chapter 6) are always listed last.

***TAPE (<n>)** **MOS** **All**
Identical to OSBYTE &8C, this command selects the Tape Filing System, or more strictly the audio tone filing system. If n=3 the 300 baud system is used. Any other or no value of n selects the 1200 baud system. Although accepted by the Compact it has no real effect as the Compact is not fitted with a cassette interface.

***TIME** **MOS** **M/C**
This is only available on the Master /Compact machines, but note that the Teletext Filing System, supplied on earlier machines adds a command of this name to the system. On the Master 128, the call is routed to OSWORD &0E, with (YX)+0=0, and a 24 character string is sent to the VDU drivers in the format "Sun,17 Aug 1986.15:49:03" (without quotes). On the Compact, which has no real time clock, the action taken is to print the string

```
Fri,31 Dec 1999,23:59:59
```

On Econet Compact machines, where a clock and fileserver are present, the time and date are read from the fileserver (using OSWORD &14, function code 16), and supplied to the OSWORD &0E routine. The fileserver does not supply the day of the week, so the string supplied starts with three spaces, then the comma.

***TITLE <title>** **DFS/ADFS** **All**
In DFS, this command takes a string of up to 12 characters, which is placed as a title string on the currently selected disc, deleting any previous string. Any characters are valid in the string, even *#:.$!"

and so on. On ADFS, a new title up to 19 characters can be given to each directory. This is done by selecting the directory and issuing a *TITLE command. The default title for each directory is the same as the name of the directory.

***TV <n> (<n>)** **MOS** **All**

This is exactly the same as OSBYTE &90. The first parameter sets the vertical positioning of the picture, the second parameter, if included, turns interlace off (0) or on (1). Note that although any value can be set on the vertical positioning, the Master Series CONFIGURE system only stores settings between 252 (-4) and 3.

***TYPE <fsp>** **All FS** **All**

This command, like *LIST and *PRINT sends a file byte by byte to the VDU driver. On the Models B and B+ it is implemented within filing systems, and sends each byte to the VDU drivers, irrespective of the effect this will have. On the Master Series, the command sends characters under 32 to the drivers in GSREAD format, that is, the format used by the MOS command of that name. This means that all non printable characters (except linefeed and carriage return) are shown in the form "|" followed by the ASCII character for 64+code. If the code is above 128, then "|!" is shown followed by the character for the code minus 128, which may in turn involve another "|". The GSREAD format for ASCII 2 is |B. The format for 134 is |!|F.

***UNPLUG <n>** **MOS** **M/C**

This disables ROM software fitted to the computer, or loaded into a sideways RAM socket. It does this on hard reset by failing to enter the ROM type byte in the table at &2A1-&2B0 on the I/O processor. If the MOS finds no entry here, it will not offer commands to that slot, so the ROM never gets a chance to initialise itself. Thus an UNPLUG only takes effect at the next hard reset. Certain utility ROMs which direct commands at a specific ROMs will therefore get round 'UNPLUGging'. See also *ROMS which reports the unplugged /inserted state.

*UNPLUG and *INSERT also exist on some versions of the 1770 DFS, and as such are available to Model B/B+ users. Similar commands are provided on some third-party products.

***VERIFY (<drv>)... 1770 DFS/ADFS B+/M/C**
This will verify the formatting of a disc. Each track is tested to check if the ID and CRC marks are correct for standard DFS or ADFS format. No attempt is made to check validity of the disc contents. The number of tracks checked is determined by the 'size' bytes in Track 0, Sector 1. This command is supplied as a disc utility on other versions of the DFS, ie, those based on the 8271 chip.

***WIPE <afsp> DFS B/B+/M**
Removes specified files from the catalogue and rearranges the catalogue. Asks for confirmation that each file conforming to the specification is to be deleted.

***X MOS M/C**
This command is used in conjunction with a 'Tube splitter', a small piece of hardware which can be built to share the Tube between two external co-processors. Typing *X must be followed by a hard reset, which will toggle the active co-processor. Without this additional hardware, the command serves no purpose. It appears in Compact MOS 5.10 even though there is no Tube.

Safe or not?
Certain * commands by their nature can corrupt the contents of main memory. The list below details the 'safe' and 'unsafe' commands, some 'unsafe' commands may be made 'safe' by specifying parameters, see the command descriptions above to check.

Safe Commands: *ACCESS, *APPEND, *BACK, *BUILD, *CAT, *CDIR, *CLOSE, *CREATE, *DELETE, *DESTROY, *DIR, *DISMOUNT, *DRIVE, *DUMP, *EX, *FREE, *INFO, *LCAT, *LEX, *LIB, *LIST, *MAP, *MOUNT, *OPT, *REMOVE, *RENAME, *SAVE, *SPOOL, *SRLOAD, *SRSAVE, *TITLE, *VERIFY, *WIPE.

NB. Commands *SRLOAD and *SRSAVE are only safe if the Q option is omitted.

Unsafe Commands: *BACKUP, *EXEC, *COMPACT, *COPY, *FORM *FORMAT, *LOAD, *PRINT, *RUN, *SRLOAD using Q, *SRSAVE using Q, *TYPE

3 : The 65C02 Processor

Introduction

The Master 128 and Compact all contain a central processing unit (CPU) that is more powerful than that of the Model B and B+. This processor, known as a 'CMOS type', contains all the instructions of the standard 6502 and quite a few extra ones. There are several different CMOS type 6502s. The Master and Compact contain a 65SC12, the external '6502 second processor' contains a R65C02, and the Master Turbo unit has a R65C102. Although there are many internal differences in the family, all these chips appear identical to the programmer, There are however some important areas of difference from the 6502 CPU. These are:

1. Indexed addressing across a page boundary is now corrected.
2. Execution of invalid opcodes is now predictable.
3. Decimal flag operations have been improved.
4. There has been a cycle time change on memory addressing.
5. The effect of interrupts during BRK execution has changed.
6. The safety of invalid read operations has been improved.
7. Some new instructions and addressing modes have been added.

Detailed explanations of the first six items and documentation of the new instructions and addressing modes are given in this chapter.

Several companies manufacture the CMOS 6502, and the chips from all but one are very similar in operation. However, Rockwell Instruments, who prefix their chips with the letter 'R' (ie, R65C02) has included four extra instructions - BBR, BBS, RMB and SMB. These occupy 32 opcodes as each instruction has a different opcode for bits 0 to 7. These are not 'chance' instructions - they are documented on the Rockwell data sheet. The only place you will normally find a Rockwell R65C02 chip is on a Master Turbo or 6502 second processor board, although some enthusiasts have been fitting them to other BBC Micro boards to replace their 6502. Note that Acorn do not guarantee to use a Rockwell chip on these units.

You should not include the Rockwell extra instructions in your programs unless you are sure that the target machine will definitely contain a Rockwell processor. This can only be assumed if you know all the machines that the software will ever run on. The BBC BASIC assembler does not support these codes. The other new 65C02 opcodes should only be used on software intended for the BBC B+, Master 128 and Compact.

Changes from the original 6502

1. The original 6502 has a well known bug where an indirect jump which crossed a page boundary ~~cross~~ does not add one to its internal counter for the new page. If the instruction in question was JMP (&12FF) and location &12FF contained &12, location &1300 contained &34 and location &1200 contained &56, you would expect the processor to jump to &3412 whereas in fact it would jump to &5612, taking the high byte from &1200, not &1300. This has been cured on the CMOS range, and the correct jump in this case would be performed.

2. Some of the original 6502 CPUs could execute a number of the extra CMOS opcodes. However the chip testing machines at the factory did not test for those codes, so no-one could guarantee whether they would be there on any particular machine, nor indeed what effect executing the codes might have. Certainly STZ worked on the majority of 6502s, but DEC A almost never did. The CMOS CPUs do not have this problem - all opcodes not allocated are executed as NOPs and take two cycles.

3. The 6502 leaves its decimal flag in an indeterminate state after a machine reset, and the N, V and Z flags are invalid in a decimal operation. The CMOS family clear the decimal flag after reset (ie, perform a CLD) and use the N, V and Z flags when the decimal mode is active. Note that the ADC and SBC opcodes (in all modes) now take one more clock cycle in decimal mode. This is to deal with this flag updating.

4. Read /modify/write instructions at an effective address, ie, INC &400,X take two read and one write cycle on the 65C02 and take one read and two write cycles on the 6502. This may improve speed where the location being modified is an I/O port. It will not affect the speed of modification of RAM.

5. If an interrupt has occurred when a BRK has been fetched but has yet to be executed, the BRK will be executed on the CMOS family. On the 6502 it is abandoned.

6. When the 6502 forms a read across a page boundary — for example LDA (&3FFF),Y with Y=2, an extra read would be performed on &3FFF+3. This can cause problems if &4002 is not RAM but a write-only I/O port. On the 65C02, the extra read is performed at the memory address pointed to by the program counter, which by definition must be RAM or ROM and not I/O map. If the instruction just mentioned was assembled at &9000 in memory, an extra read of &9003 will occur. Note that the extra read is not seen by the program.

7. There are 12 new instructions and two new addressing modes which provide 59 new opcodes. The rest of the chapter is devoted to explaining these in detail.

Syntax

In the description of the new opcodes below the following syntax is used throughout:

A = Accumulator
X = X Register
Y = Y Register
M = Memory
N = Negative flag
0 = Zero
1 = One
Z = Zero Page
SP = Stack Pointer
PC = Program Counter

A description such as A=A+1 would infer that the contents of the accumulator have one added to them. Similarly SP=SP+1,(SP)->X would read as 'the stack pointer is incremented by one and the contents of the byte pointed to by the stack pointer is copied into the X register'.

65C02 New Instructions

The extra R65C02 commands are identified by the remark 'Rockwell only'.

BIT Test bits with accumulator mask (New modes)

Flags set by A AND M
The BIT command has been extended to allow immediate operations ie, BIT #&AA, where no memory is altered, but the flags are set as a result of ANDing the argument with the accumulator. Note that the accumulator is not altered. However the operator is of limited use as only Z is altered, not N and V.

Also BIT can now be used with addressing modes zero page,X and absolute,X, in which case N and V are affected.

Opcodes:

```
89 FF     BIT #&FF
34 70     BIT &70,X
3C 00 70  BIT &7000,X
```

Clock cycles: Immediate/zero page,X 2, Absolute,X 3

BBR Branch on Bit Reset

IF ZP(bit N)=0 THEN PC=PC+M *Rockwell only*
This new command is a conditional branch based on the state of a single bit of zero page memory. The syntax is BBRn zp bb where n is the bit (0-7), zp is the memory address in zero page and bb is the distance of the branch, +127 to -128 as usual. The command is useful as it can replace masking, accumulator use and flag setting. There is an associated BBS command.

The branch is relative to the position of the next instruction. This means that a branch of -3 would be a loop back to the start of the original instruction. Normally you don't have to worry about this, as the assembler takes care of it, but no BBC Micro assembler supports R65C02 instructions, so you will have to hand-code the instruction should you wish to use it - this can be done by using the EQUB instruction to assemble the bytes concerned.

Flags affected : None
Addressing mode : Relative

Opcodes : &0F=BBR0 &1F=BBR1 &2F=BBR2 &3F=BBR3
&4F=BBR4 &5F=BBR5 &6F=BBR6 &7F=BBR7

```
0F 70 F0  BBR0,&70 -16        (3 bytes)
```

Clock cycles : 5 (+1 if branch occurs)
(+1 more if branch crosses page boundary)

Example program

```
LDA port \ Put port value in &70
STA &70
BBR3,&70 switch  \ If bit 3=0 jump to 'switch'
```

BBS Branch on Bit Set

IF ZP (bit N)=1 THEN PC=PC+M *Rockwell only*
This is similar to BBR above, except that the branch occurs if the memory bit in question is set, ie, contains 1.

Flags affected : None
Addressing mode: Relative
Opcodes : &8F=BBS0 &9F=BBS1 &AF=BBS2 &BF=BBS3
&CF=BBS4 &DF=BBS5 &EF=BBS6 &FF=BBS7

```
FF 25 20  BBS7,&25 32         (3 bytes)
```

Clock cycles : 5 (+1 if branch occurs)
(+1 more if branch crosses page boundary)

BRA Branch Always

PC=PC-M
This command is a member of the Branch set, but will always branch, irrespective of the state of the flags. It is therefore functionally equivalent to JMP except that it uses relative addressing, thus being relocatable and shorter. 'nn' is the distance of the branch.

The branch is relative to the position of the next instruction. See BBR.

Flags affected : None
Addressing mode: Relative
Opcode : &80 &nn

```
80 F1     BRA -15   (2 bytes)
```

Clock cycles : 3

Example program

```
.wait
JSR OSRDCH       \ Get a key press in A
CMP #32  \ Is it a space
BEQ spc  \ If so go to spc routine
CMP #13  \ or is it a RETURN
BEQ cr           \ go to cr routine
LDA #21  \ clear key presses
LDX #0
JSR OSBYTE
BRA wait \ and loop, unconditionally back
```

As it uses only OS calls, the whole section of code is relocatable.

DEC A Decrement the Accumulator

A=A-1
This new call, an obvious omission from the 6502, allows the accumulator to be decremented directly, in the same way that X, Y and memory can be. It is over twice as fast as zero page memory decrementing. BBC BASIC IV onwards and Acornsoft MASM assembler allow the alternative mnemonic DEA.

Flags affected : N Z
Addressing mode: Accumulator
Opcode : &3A

```
3A        DEC A     (1 byte)
```

Clock cycles : 2

INC A Increment the Accumulator

A=A+1
This new call is, like DEC A, an obvious omission from the 6502. It allows the accumulator to be incremented directly, in the same way as X, Y and memory. BBC BASIC IV onwards and Acornsoft MASM assembler allow the alternative mnemonic INA.

Flags affected : N Z
Addressing mode : Accumulator
Opcode : &1A

```
1A      INC A    (1 byte)
```

Clock cycles : 2

PHX Push the X Register onto the Stack

X-> (SP),SP=SP-1
This places the contents of the X register in the location pointed to by SP, the machine stack pointer. SP is then decremented by one. X remains unchanged. The command is useful for storing the value of X perhaps before calling a subroutine where you know it will be corrupted. On the 6502 only the A register can be stacked in this way and consequently other registers had first to be transferred to A.

Flags affected : None
Addressing mode : Implied
Opcode : &DA

```
DA      PHX      (1 byte)
```

Clock Cycles : 3

Example program

```
.start
PHP
PHA     \ store all registers
PHX     \ for later retrieval
PHY
....
```

PHY Push the Y Register onto the Stack

Y->(SP),SP=SP-1
This places the contents of the Y register in location pointed to by SP, the machine stack pointer, which is then decremented by one. Y remains unchanged. The command is useful for storing the value of Y perhaps before calling a subroutine where you know it will be corrupted.

Flags affected : None
Addressing mode : Implied
Opcode : &5A

```
5A      PHY      (1 byte)
```

Clock Cycles : 3

Example program : see PHX

PLX Pull the X Register from the Stack

SP=SP+1,(SP) ->X
This increments SP by one, and then takes the byte pointed to by the stack pointer (SP) and places it in the X register. The actual stack memory is not altered. The old contents of X are lost. This command is useful for restoring registers saved with PHX, but also can be used for data swapping, see the example program.

Flags affected : N Z
Addressing mode : Implied
Opcode : &FA

```
FA      PLX     (1 byte)
```

Clock Cycles : 4

Example program : data swapping between X and Y

```
.swap
PHX     \ store X and Y
PHY
PLX     \ X into Y
PLY     \ and Y into X, without corrupting A
```

PLY Pull the Y Register from the Stack

SP=SP+1, (SP)->Y
After incrementing SP by one, this takes the byte pointed to by the stack pointer (SP) and places it in the Y register. The actual stack memory is not altered. The old contents of Y are lost. This command is useful for restoring registers saved with PHY, but also can be used for data swapping.

Flags affected : N Z
Addressing mode : Implied
Opcode : &7A

```
7A      PLY     (1 byte)
```

Clock Cycles : 4

RMB Reset Memory Bit

0 -> ZP (bit N) *Rockwell only*

This instruction resets (ie, loads with zero) a single bit of a zero page memory location without altering any of the other bits in the byte. This can be very useful, especially in conjunction with BBR and BBS above, when using zero page memory locations, in effect, as additional flag registers. It cannot unfortunately be used for I/O bit-toggling as only zero page addressing is allowed, and there isn't any I/O on page zero. There is a complementary instruction SMB (Set Memory Bit).

Flags affected : None
Addressing Mode : Zero page
Opcodes : &07=RMB0 &17=RMB1 &27=RMB2 &37=RMB3 &47=RMB4 &57=RMB5 &67=RMB6 &77=RMB7

```
17 70     RMB1,&70  (2 bytes)
```

Clock cycles : 5

SMB Set Memory Bit

1 -> ZP (bit N) *Rockwell only*

This is the opposite instruction to RMB. It will set (ie, load with 1) a single bit of a zero page memory location, and is useful for the same reasons as RMB.

Flags affected : None
Addressing Mode : Zero page
Opcodes : &87=SMB0 &97=SMB1 &A7=SMB2 &B7=SMB3 &C7=SMB4 &D7=SMB5 &E7=SMB6 &F7=SMB7

```
C7 43     SMB4,&43  (2 bytes)
```

Clock cycles : 5

STZ Store Zero in Memory

0->M

This allows you to set a byte in memory to zero without altering any registers. It is very useful, as counters and so on often have to be

initialised to zero, but for one reason or another the contents of the registers need to be preserved. It is equivalent to the 6502 instructions

```
LDA #0
STA mem
```

but of course, does not affect the accumulator contents. It is available in the same addressing modes as STY plus the additional mode, Absolute,X.

Flags affected : None
Addr. Modes : Zero page, Zero page,X
Absolute, Absolute,X

Opcodes :

```
64 70     STZ &70           (2 bytes)
74 70     STZ &70,X         (2 bytes)
9C 00     0E STZ &E00       (3 bytes)
9E 00     0E STZ &E00,X     (3 bytes) *
```

Clock cycles : zp=3 : zp,X=4 : abs=4 : abs,X=5

TRB Test and Reset Bits

(M AND (NOT A)) -> M
This command will reset bits in the memory location specified, using a mask contained in the accumulator. Only those bits which are set in the accumulator will be affected in the memory being reset. Unlike the RMB and SMB instructions this instruction works on non page zero memory and therefore is very useful for I/O bit toggling.

Flags affected : Z=1 if all masked bits already reset
Additionally NV affected on Rockwell CPUs
Addr.Modes : Absolute, zero page
Opcodes : &14 (zero page), &1C (main memory)

```
14 70     TRB &70   (2 bytes)
1C 00 0E  TRB &E00  (3 bytes)
```

Clock cycles : zp =5 : abs =6

Example program :

```
\ switch on M128 internal Tube
\ only disturb the relevant bit.
LDA #&10         \ Mask for bit 4
TRB acccon       \ Reset bit 4 of &FE34
BEQ int          \ If Z, already reset
```

```
RTS
.int BRK          \ Tube already on
EQUB 253          \ so why are you doing it?
EQUS "Already on"
EQUB 0
```

TSB Test and Set Bits

M OR A->M

This command is the exact opposite of TRB. It will set bits in the memory location specified, using an accumulator mask. It is also useful for I/O toggling. In both TRB and TSB, the accumulator bits should be set where action is required. A common mistake is to set bits in the mask prior to TSB and reset bits prior to TRB.

Flags affected :	Z=1 if all masked bits are reset. Additionally NV affected on Rockwell CPUs
Addr. Modes :	Absolute, zero page
Opcodes :	&04 (zero page), &0C (main memory)

```
0C 70     TSB &70   2 bytes)
0C 00 0E  SB &E00   3 bytes)
```

Clock cycles : zp =5 : abs =6

65C02 New Addressing Modes

There are two new addressing modes on the 65C02 family. These are :

Indexed Absolute Indirect

PC <= (M+X) + (M+X+1)*256

This is a new mode and only contains one instruction, JMP. The instruction JMP (IND,X) takes the 16-bit value of IND and adds to it the 8-bit contents of X. This gives a 16-bit result. The contents of this memory address, plus the next address are the effective address jumped to. The effect is like a JMP (address) where 'address' is IND+X. What does not happen is the addition of X to the contents of IND and IND+1. Early editions of the Rockwell R65C02 data sheet give the instruction as JMP (IND),X which is presumably a typesetting error, but nevertheless watch out for it in disassemblers.

Flags affected :	None
Addressing Mode :	Indexed Absolute Indirect
Opcode :	&7C &nn &nn

```
7C 00 0E  JMP (&E00),X       (3 bytes)
```

Clock cycles : 6

Example program:

```
\ Select from a jump table
\ (only even values of X are meaningful)
LDX #4                    \ Select 'close'
JMP (table, X)   \ and jump to it.
.table
EQUW read
EQUW write
EQUW close
EQUW open
```

Indirect Zero Page (No Index)

This is equivalent to zero page indirect with Y (ie LDA (&70),Y) where Y is zero. Of course, the X and Y contents are irrelevant to the operation. It is very annoying to have to clear Y on the 6502 for this sort of operation, so the new mode is very valuable.

Flags affected : N Z
Addressing Mode : Indirect Zero Page
Opcodes :

```
12 70   ORA (&70)       (2 bytes)
32 70   AND (&70)       (2 bytes)
52 70   EOR (&70)       (2 bytes)
72 70   ADC (&70)       (2 bytes)
92 70   STA (&70)       (2 bytes)
B2 70   LDA (&70)       (2 bytes)
D2 70   CMP (&70)       (2 bytes)
F2 70   SBC (&70)       (2 bytes)
```

Clock cycles : 5 (+1 for ADC and SBC in decimal mode)

Example program

```
.prog
LDA #&20 \ This routine will
STA &70  \ return with A containing
LDA #&50 \ the CONTENTS of &5020.
STA &71  \
LDA (&70)        \ New CMOS instruction
RTS
```

4 : New MOS Calls

OSBYTE calls

Most of the routines in the Operating System are available to the user through the documented Operating System calls resident in page &FF of memory. Many of these are available through OSBYTE, a general purpose entry point located at &FFF4. The action taken on calling OSBYTE is totally dependent on the values of the registers on entry. The contents of the Accumulator (A) defines the operation to be performed, X and Y contain any parameters required.

In this book we have decided not to repeat information that is readily available elsewhere. So OSBYTE calls which are identical on Model B are not documented here - you will find them in the Model B User Guide and Advanced User Guide. However, Appendix A contains a complete list of the calls. OSBYTE calls can be entered through the command line by using the *FX command. ie:

```
*FX 156, 8, 227
```

which is equivalent to:

```
LDA #156
LDX #8
LDY #227
JSR &FFF4
```

For calls in the range &00 to &19 the value of Y on entry is irrelevant, is not corrupted, and is never used to return a value.

OSBYTE calls from &A6 to &FF allow the user to read a value, write a value, or just read or write some bits. This is done by setting X and Y on entry, following the rule:

NEW VALUE = (OLD VALUE AND Y) EOR X

meaning that to read a value, X=0 and Y=&FF, to write a value, X=value and Y=0.

All the values used by OSBYTEs &A6-&FF are stored in memory from OSVARS, any particular value being stored at OSVARS + A register.

OSVARS itself is determined by OSBYTE &A6 and &A7. It is &0190 on all machines to date, For example OSBYTE &C2/&C3 read/set how each colour flashes in colours 8-15. The MOS looks at the variables to time the flashing, which are &190+&C2 = &252 and &190+&C3 = &253. You can see the effect like this:

```
MODE 4
VDU 19,1,15;0;          (Watch the effect)
?&252=10:?&253=3        (Now watch that)
?&252=1:?&253=100       (and that)
```

(This is not the recommended way of altering flash rates!)

All OSBYTE calls can be made from a second processor.

OSBYTE &00 (0) **Give MOS version number** **All**

Entry : X=0 Generate error message containing 'OS' and version number.
X>0 Return value in normal way.

Exit :

X=0	OS 1.00	Early Model B/Electron
X=1	OS 1.20	Model B (UK or US)
X=2	OS 2.00	Model B+
X=3	MOS 3.20	Master 128
X=4	MOS 4.00	Master ET
X=5	MOS 5.00	Master Compact

Also see OSBYTE &81 with X=&00 and Y=&01, and OSBYTE &F0. Between these three calls you can work out exactly what machine your software is running on.

OSBYTE &01 (1) **User call (read I/O location &281)** **All**
OSBYTE &02 (2) **Set input stream** **All**
OSBYTE &03 (3) **Set output stream** **All**
No change in function on the B+ and Master Series. See Model B Guides.

OSBYTE &04 (4) **Select cursor state** **All**

Entry:

X=0 Cursor keys used for editing and COPYing.

X=1 Assigns ASCII codes to (brown) cursor keys. COPY=135, LEFT=136, RIGHT=137, DOWN=138, UP=139.

X=2 Cursor keys return strings set by *KEY where COPY=11, LEFT=12, RIGHT=13, DOWN=14, UP=15.

X=3 On Compact only, the cursor keys will emulate the joysticks and will supply values to the ADVAL (OSBYTE &80) routines. The cursor key switch positions (pressed=1) are logically ORed with any physical joystick.

COPY is ORed with Fire Button 1.
LEFT is ORed with Joystick Left.

RIGHT is ORed with Joystick Right.
UP is ORed with Joystick Up.
DOWN is ORed with Joystick Down.

Remember that the Compact joystick system is digital. See OSBYTE &80 for a full explanation.

OSBYTE &05 (5)	**Select printer type**	**All**
OSBYTE &06 (6)	**Select printer ignore character**	**All**
OSBYTE &07 (7)	**Select RS423 receive rate**	**All**
OSBYTE &08 (8)	**Select RS423 transmit rate**	**All**
OSBYTE &09 (9)	**Set flash rate of 1st screen colour**	**All**
OSBYTE &0A (10)	**Set flash rate of 2nd screen colour**	**All**
OSBYTE &0B (11)	**Set keyboard auto-repeat delay**	**All**
OSBYTE &0C (12)	**Set keyboard auto-repeat rate**	**All**
OSBYTE &0D (13)	**Disable event**	**All**
OSBYTE &0E (14)	**Enable event**	**All**
OSBYTE &0F (15)	**Flush input/all buffers**	**All**

No change in function on the B+ and Master Series. See Model B Guides.

OSBYTE &10 (16)	**Write number of ADC channels**	**All**
OSBYTE &11 (17)	**Write next channel to be sampled**	**All**

No change in function on the B+ and Master 128. The Master Compact also offers a full emulation of these calls. However the only useful activity is to turn off the 'conversion' using *FX16,0.

OSBYTE &12 (18) **Reset function key definitions** **All**

On all machines this call resets the soft key strings to nulls. Note that on the Master Series the strings are in a totally different place in memory (see Appendix D) and are stored in a different way.

OSBYTE &13 (19) **Wait for vertical sync pulse** **All**

No change in function on the B+ and Master Series. See Model B Guides.

OSBYTE &14 (20) Font explosion/definition All

On Model B and B+, this call reserves space so that character matrixes can be defined. Space can be reserved for between 32 (default) and 224 defined characters. Attempts to define a character without having first reserved the correct space will result in unwanted alterations to other characters. The first 32 characters are stored at &C00-&CFF, and each further block of 32 is stored from OSHWM upwards. OSHWM, as read by OSBYTE &B4 is increased. Remember however that PAGE in BASIC does not increase automatically, and must be manually altered. Note that any OSBYTE &14 call has a side effect of restoring ROM definitions to matrixes in the range &20-&7E.

Entry	Definable range
X=0	&80-&9F (default)
X=1	&80-&BF
X=2	&80-&DF
X=3	&80-&FF
X=4	&20-&3F,&80-&FF
X=5	&20-&5F,&80-&FF
X=6	&20-&FF

Note: It is not normally possible to send matrix 127 through the VDU drivers as it is a control code (DELETE)

On the Master series, the call serves a different purpose. The Master series font is permanently exploded, and all characters from &20 to &FF are redefinable at any time. This call is still implemented and now merely restores the ROM definitions of the entire character set (&20-&FF). An OSBYTE &20 call with X=6 will result in an exploded font and default matrixes on all machines. See also OSBYTE &19.

OSBYTE &15 (21) Flush specific buffer All

No change in function on the B+ and Master Series (Buffer 8 (speech buffer) is present but unused on the Master). See Model B Guides.

OSBYTE &16 (22) Inc. ROM polling semaphore E/M/C

The call increments a byte in memory known as the ROM polling flag. When the flag is non-zero, the Operating System issues a call (OSBYTE &8F, X=&15) to each ROM, 100 times a second, to which ROMs may respond. This enables you to write background routines

without having to set up interrupt or event vectoring. The reason the flag is incremented, rather than toggled, is that several ROMs may turn the flag on, then off, and only when all ROMs present have turned the system off will it cease to run. See Chapter 6 for more details of the service call.

Entry : No parameters
Exit: X=preserved, Y=corrupt

OSBYTE &17 (23) Dec. ROM polling semaphore E/M/C
This call decrements the ROM polling flag. Every ROM which issues OSBYTE &16 to start polling should issue this call when polling is no longer required. Hard reset sets the polling semaphore to zero.

OSBYTE &18 (24) Select external sound system E
On the Electron, this call is used to select an external sound system, to which subsequent sound command (OSWORD &07) data is sent. The system was however, never implemented, but if it were, would take a parameter in X to select a system. See also OSBYTE &74

This call is not implemented on Model B/B+. It does not do anything on the Master series where it is reserved for Acorn use.

OSBYTE &19 (25) Restore font definitions M/C
The call defines a group of characters from the font definitions in the ROM, as opposed to OSBYTE &14 which defines the entire set. OSBYTE &19 with X=0 is equivalent to OSBYTE &14.

Entry:	Restored characters	
X=0	&20-&FF	32-255
X=1	&20-&3F	32-63
X=2	&40-&5F	64-95
X=3	&60-&7F	96-127
X=4	&80-&9F	128-159
X=5	&A0-&BF	160-191
X=6	&C0-&DF	192-223
X=7	&E0-&FF	224-255

Exit : X=corrupt, Y=corrupt

OSBYTE &1A(26) to &31(49) **All**

Calls to OSBYTE with A set between &1A and &31 cause a 'Bad command' error.

OSBYTE &32 (50) Econet transmit errors **NFS**

This call, serviced by NFS/ANFS, is used after a data transmission, to check if the transmission worked correctly. If X=0, the transmission worked.

Entry : None
Exit : A=corrupt, Y=corrupt, C=corrupt

X	b7	0=complete	1=in progress
	b6	0=successful	1=failed
	b5	Always 0	
	b0-4	Error code	0=successful

Errors: &40=Line jammed
&41=Some part of four-way handshake lost/corrupt
&42=No scout acknowledgement
&43=No clock
&44=Bad transmit control block (TXCB)

OSBYTE &33 (51) Econet receive errors **NFS**

This call, serviced by NFS/ANFS, is used to check whether data has been received. The call would usually be continuously issued until data was present.

Entry : X=Receive control block (RXCB) number
Exit : A=corrupt, Y=corrupt, C=corrupt

X	b7	0=no data rec'd	1=data received
	b6-0	corrupt.	

The appropriate code to check that a message had been received would be:

```
.loop
LDA #&33
LDX rxcb      \ Already stored
JSR osbyte
TXA
AND #&80
BEQ loop
```

```
.recvd \ Message received.
....
```

OSBYTE &34 (52) Delete Econet RX control block NFS
This call, serviced by NFS/ANFS, deletes a receive control block (RXCB). Also, it will enable/disable network receive events (ANFS only)

Entry : X=RXCB number, or
X=100 Disable network receive events
X=150 Enable network receive events
Exit : X=preserved, Y=corrupt, C=corrupt

OSBYTE &35 (53) Sever a remote connection NFS
This call, equivalent to the NFS command *ROFF, severs the remote connection to another machine. The other machine returns control to its own keyboard.

Entry : No parameters
Exit : All corrupt

OSBYTE &44 (68) Test for Sideways RAM B+/M/C
The call tests whether sideways RAM (SRAM) is fitted to the machine. As you might expect it only knows about Acorn Sideways RAM which means the B+ 128k expansion, and the standard 64k of SRAM in the Master and Compact. However, it will detect 16k of SRAM that is permanently write-wired such as Solidisk or the RAM fitted to ROM boards such as ATPL. The call is used by the BBC B+ to display its startup message, which is why a standard BBC plus 1770 DFS plus 16k of Sideways RAM will display 'Acorn OS 80K' on hard reset. Listing 4.1, on the next page, illustrates how the call is used.

On the Model B and B+, and the first Master 128 MOS, the code to service this call is contained within the 1770 DFS. On MOS 3.21 and all Compacts, the code is contained in the main MOS. Note that when the call is used with non-Acorn sideways RAM, unusual results may occur. Generally though, the example program should work correctly. The MOS tests the RAM by writing to the byte at &8008, using the &FE30 latch to change ROM bank. It does not use the OSRDRM routine.

Entry : A=&44 X, Y=irrelevant
Exit : A=&44
X b0 set if Bank 4 of SRAM present and usable
X b1 set if Bank 5 of SRAM present and usable
X b2 set if Bank 6 of SRAM present and usable
X b3 set if Bank 7 of SRAM present and usable

Non-Acorn SRAM should return a correct value if it is selected by writing to &FE30, but will return a value of &0F if it is permanently write selected ie all write operations above &8000 are to a SRAM bank. It is however very hardware dependent and no completely correct result can be predicted.

There is further information on Sideways RAM including programs in Chapter 5.

Listing 4.1.

```
 10 REM Demo of OSBYTE &44
 20 REM (c) Dave Atherton 1987
 30 REM for B+128/M/C
 40 REM MOS : A Dabhand Guide
 50 :
 60 osbyte=&FFF4
 70 oswrch=&FFEE
 80 DIM M% 40
 90 FOR pass=0 TO 3 STEP 3
100 P%=M%
110 [OPT pass
120 \ LDA #0
130 \ STA mem
140 LDA #&44
150 JSR osbyte
160 TXA
170 LDX #4
180 .loop
190 CLC
200 ROR A
210 BCC not
220 INC mem
230 .not
240 DEX
250 BNE loop
260 LDA mem
270 CLC
280 ADC #ASC"0"
290 JSR oswrch
300 RTS
310 .mem
320 EQUB 0
```

```
330 ]:NEXT
340 PRINT"There are ";
350 CALL M%
360 PRINT" banks of SW RAM."
```

OSBYTE &45 (69) Sideways RAM allocations B+/M/C
This call is implemented on the BBC B+, Master and Compact. It is also available on BBC B with 1770 DFS, as it is again implemented in the DFS ROM.

The call reads the status of any Sideways RAM banks indicating whether they are being used with pseudo or absolute addressing. Like OSBYTE &44, the results are bit significant.

Entry :	A=&44	X,Y irrelevant
Exit :	A=&44	
	X b0	set if Bank 4 of SRAM using pseudo, clear if not.
	X b1	set if Bank 5 of SRAM using pseudo, clear if not.
	X b2	set if Bank 6 of SRAM using pseudo, clear if not.
	X b3	set if Bank 7 of SRAM using pseudo, clear if not.

There is an example program to display size of pseudo area in Chapter 5.

OSBYTE &46 (70) to &5F (97) All
Calls to OSBYTE with A set between &46 and &5F cause a 'Bad command' error.

OSBYTE &60 (96) Terminal commands M
This call is used only by the Terminal software supplied in the Master 128 Mega-Bit ROM. It has six functions only:

Entry:	X=&40	Receive flow control (RFC) disabled
	X=&41	Receive flow control enabled
	X=&62	Transmit flow control (TFC) disabled
	X=&63	Transmit flow control enabled
	X=&80	Remove INSV and REMV intercepts
	X=&81	Setup INSV and REMV intercepts

OSBYTE &61 (97) to &6A (106) All
Calls to OSBYTE with A set between &61 and &6A cause a 'Bad command' error.

OSBYTE &6B (107) Select int/ext 1MHz bus M

On the Master 128 (only) there is an internal 2MHz bus which is brought out to the cartridge sockets and an external 1MHz bus. Only one can be active at once, and this call selects between them.

Entry : X=0:Y=0 Select external 1MHz bus
X=1:Y=0 Select internal 2MHz bus
Exit : X=preserved, Y=corrupt

NB: The call does not work with MOS 3.20 due to a bug, and access to the switch must be made directly. See Chapter 5.

OSBYTE &6C (108) Select screen for direct access M/C

This call is implemented on the Master series only, and concerns selection of one of the two planes of 20k of RAM from &3000 to &7FFF. The three OSBYTES &6C, &70 and &71 control direct access, VDU driver access, and display, all of which may be different on the Master. For the more limited control of shadow memory on the B+, see Chapter 5.

Entry : X=0 Main memory selected for direct access
X=1 Shadow memory (LYNNE) selected for direct access
Exit : X=preserved, Y=corrupt

OSBYTE &6D (109) Make FS permanent M/C

This call is only implemented on the Master and Compact. If made while a temporary filing system command is active, that temporary filing system becomes permanent. A simple example can be seen by typing :

```
*DISC
*-TAPE-CAT
(Press ESCAPE)
*CAT              (has come back to disc)
*-TAPE-FX109
*CAT              (now looking at tape)
```

OSBYTE &70 (112) Select screen for VDU access M/C
This is similar to OSBYTE &6C, but controls VDU access. The Master Series MOS allows direct code (LDAs and STAs) to access the bank of screen RAM determine by OSBYTE &6C, but this is not necessarily the same as the bank addressed by MOS VDU code. See Chapter 5 for more details.

Entry : X=0 Select main or shadow according to current mode (ie if MODE 128-135 used, select shadow)
X=1 Select main memory (immediately)
X=2 Select shadow memory (immediately)
Exit : X=old setting, Y=corrupt

OSBYTE &71 (113) Select screen for video display M/C
This is similar to OSBYTE &6C, but controls display. When both the shadow and main RAM areas from &3000 to &7FFF are full of picture data (in the same mode), calling OSBYTE &71 can flick instantly between the two. See Chapter 5 for more details.

Entry : X=0 Select main or shadow according to current mode (ie if MODE 128-135 used, select shadow)
X=1 Select main memory (immediately)
X=2 Select shadow memory (immediately)
Exit : X=old setting, Y=corrupt

OSBYTE &72 (114) Select next screen mode B+/M/C
This is not really a shadow selection command as such. It allows programs to automatically select shadow mode (always) at mode change even if the mode chosen is 0-7 (rather than 128-135). This offers shadow mode to programs already written and issued, which contain such commands as MODE 0. The call is exactly equivalent to *SHADOW, which calls this routine. The setting remains in force until the next hard reset. What is actually selected is the VDU access and display. Direct writes to shadow or main memory are not affected. On the B+, this is the only shadow control call.

Entry : X=0 Always select shadow memory at each mode change regardless of mode number.
X=1 Select shadow memory at mode change if mode in range 128-135, else select main memory.

OSBYTE &73 (115) Blank or restore palette **E**

On the Electron only this call will temporarily set all colours to black, or restore them. It is useful because NMI routines cause problems with the screen update circuitry.

Entry : X=0 Restore the palette
X>0 Set all palette colours to black, if in Modes 0, 1 or 2

The call gives a 'Bad command' on Model B. On the B+/M/C it has no effect.

OSBYTE &74 (116) Reset internal sound system **E**

This call is allocated, for the Electron only, to resetting the internal sound system. This is in connection with the external sound system supported by OSBYTE &18, and would take a parameter in X. It causes a 'Bad command' error on the Model B, and has no effect on the B+/M/C.

OSBYTE &75 (117) Read VDU status byte **All**

This call reads the VDU status byte (at &D0 on the I/O processor on all machines). The VDU status byte is a primary reference, and although no call exists to write legally to it, a write will cause the desired effects. For example ?&D0=?&D0 OR 2 will cause the screen to stop scrolling. Acorn have indicated in the past that because of failure to provide a write capability to OSBYTE &75, it is extremely unlikely that the byte would ever be moved from &D0, giving access to this byte (using OSWORD &05 and &06) a quasi-legal status.

Entry : A=&75:X=irrelevant:Y=irrelevant
Exit : A=&75:X=status byte:Y=corrupt

The status byte has the following meaning. The bits of the status register are set when the conditions described occur.

b0 Printer enabled by VDU 2
b1 Scrolling disabled
b2 Paged scrolling enabled by VDU 14

b3 Text window enabled by VDU 28
b4 Currently using shadow screen (B+/M/C)
b5 Graphics text enabled by VDU 5
b6 Input and output cursors separated
b7 Screen disabled by VDU 21

OSBYTE &76 (118) Update keyboard LEDs All
No change in function on the B+ and Master series. Remember that the Master 128 has its LEDs in a different keyboard position, and also that the Electron, while servicing this call only has a CAPS LOCK LED. On exit Xb7 is set if CTRL is being pressed at the time.

OSBYTE &77 (119) Close all *SPOOL etc. files All
OSBYTE &78 (120) Write keys pressed information All
OSBYTE &79 (121) Keyboard scan All
OSBYTE &7A (122) Keyboard scan from &10 All
No change in function on the B+ and Master series. The Master 128 and Compact have of course more keys. See Appendix F for a complete key number table.

OSBYTE &7B (123) Printer driver going dormant All
OSBYTE &7C (124) Clear ESCAPE condition All
OSBYTE &7D (125) Set ESCAPE condition All
OSBYTE &7E (126) Acknowledge ESCAPE condition All
OSBYTE &7F (127) Check for end of opened file All
OSBYTE &80 (128) Read ADC channel/buffer status All
No change in function on the B+ and Master Series. See Model B Guides. As with all A-to-D calls, the Electron passes OSBYTE &80 with a positive value to the ROMs as an unknown OSBYTE. The Compact emulates the analogue conversion - see Appendix I for full details.

OSBYTE &81 (129) Read key All
This call, which is called by INKEY in BASIC has three functions. The first is to read a key within a time limit, the second to read if a key is being pressed at that exact moment — including non-character keys like CTRL and CAPS LOCK, and the third, a special case, will read the machine type.

1. Read key in time limit

Entry : A=&81
X=Time limit (in centiseconds) AND &FF
Y=Time limit (in centiseconds) DIV &100
Exit : If key pressed in time limit (except ESCAPE):
A=&81:X=ASCII value of key pressed:Y=0, C=0
If no key hit by time limit: A=&81:X=&FF:Y=&FF, C=1
If ESCAPE pressed (the key that generates an ESCAPE event, not necessarily the key marked ESCAPE):
A=&81:X=corrupt, Y=&1B:Carry set

2. Read if specific key is being held down

Entry : A=&81
X=Internal key number EOR &FF : Y=&FF
(See Appendix F for a list of internal key numbers)
Exit : A=&81:X=&FF:Y=&FF Key is being pressed
A=&81:X=0:Y=0 Key is not being pressed

3. Read machine type

Entry : A=&81:X=0:Y=&FF
Exit : X=machine type:Y=0
Machine types:

A=0	BBC Micro MOS 0.10
A=1	Acorn Electron MOS 1.00
A=&FF	BBC Micro MOS 1.00 or MOS 1.20.
A=&FE	BBC Micro MOS 1.00 or MOS 1.10 USA
A=&FD	BBC Master 128 MOS 3.20
A=&FC	BBC Micro MOS 1.20 West Germany
A=&FB	BBC Micro B+ MOS 2.00
A=&FA	Acorn ABC MOS
A=&F5	BBC Master Compact MOS 5.10
A=&F4	BBC Master 128 MOS 3.26

OSBYTE &82 (130) Read higher order address All
OSBYTE &83 (131) Read OSHWM All
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &84 (132) Read top of user RAM (HIMEM) All

This call returns in X(lo), Y(hi) the address above the last address free to the user. If shadow mode is selected for VDU access the address returned is &8000. If a second processor is active, the address returned is that of the last piece of machine code to be started with *RUN or *GO, or a language startup. However — starting BASIC after *RUNning machine code on a 6502 second processor doesn't alter the value.

Listing 4.2.

```
 10 REM Read HIMEM in non-BASIC environment.
 20 REM (c) Dave Atherton 1987
 30 REM for B/B+/E/M/C
 40 REM MOS : A Dabhand Guide
 50 :
 60 DIM M% &200
 70 oswrch=&FFEE:osbyte=&FFF4
 80 FOR pass=0 TO 3 STEP3
 90 P%=M%
100 [OPT pass
110 LDA #132
120 JSR osbyte
130 TYA
140 JSR hexout
150 TXA
160 .hexout
170 PHA
180 LSR A
190 LSR A
200 LSRA
210 LSR A
220 JSR hexout2
230 PLA
240 .hexout2
250 AND #&0F
260 ORA #&30
270 CMP #ASC"9"+1
280 BCC under10
290 ADC #6
300 .under10
310 JMP oswrch
320 ]NEXT
330 CALL M%
```

OSBYTE &85 (133) Read HIMEM for a given mode All

No change in function on the B+ and Master Series. See Model B Guides.

OSBYTE &86 (134) Read input cursor position All

This call returns the current text cursor position. When cursors are separated during cursor editing (ie, COPYing), this returns the position of the input cursor, that's the one you are moving with the cursor keys, represented by the machine cursor — not a block character. OSBYTE &A5 is a new call for the Master Series, which returns the output cursor position (the block character). This was not previously readable.

Entry : No parameters
Exit: X=Horizontal character position
Y=Vertical character position

OSBYTE &87 (135) Read character at cursor position All

This returns the character at the cursor position (as returned by OSBYTE &86). Note that on B+ onwards, the screen mode returned in Y will be in the range 0-7. The shadow screen setting must be checked separately (by OSBYTE &75 reading bit 4 for the VDU status byte).

Entry : No parameters
Exit : X=ASCII code of character, or
X=0 Code not recognised as a character

OSBYTE &88 (136) Perform *CODE All

No change in function on the B+ and Master Series. See Model B User Guides,

OSBYTE &89 (137) Perform *MOTOR All

On all machines except Compact, this switches the cassette relay. On the Compact the command is accepted, but it has no effect, so any programs transferred onto Compact ADFS discs, will be able to survive *TAPE and *MOTOR commands in the software.

Entry : X=0 Motor relay off
X>0 Motor relay on
Exit : X=corrupt:Y=corrupt

The CFS calls this routine with Y=0 when writing to, and Y=1 when reading from tape.

OSBYTE &8A (138) Insert value into buffer All
OSBYTE &8B (139) Perform *OPT All
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &8C (140) Perform *TAPE All
On all machines except the Compact, this selects the CFS. On the Compact, the command is accepted, but has no effect. See also OSBYTE &89.

Entry : X=3 Select 300 baud CFS
X<>3 Select 1200 baud CFS
Exit : X=preserved:Y=preserved

OSBYTE &8D (141) Perform *ROM All
OSBYTE &8E (142) Enter language ROM All
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &8F (143) Paged ROM service request All
No change in function on the B+ and Master Series. However, there are many new service calls available. See Chapter 6 where are the calls available through this OSBYTE are documented.

OSBYTE &90 (144) Perform *TV All
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &91 (145) Get character from buffer All
OSBYTE &92 (146) Read from FRED, 1MHz bus All
OSBYTE &93 (147) Write to FRED, 1MHz bus All
OSBYTE &94 (148) Read from JIM, 1MHz bus All
OSBYTE &95 (149) Write to JIM, 1MHz bus All
OSBYTE &96 (150) Read from SHEILA All
OSBYTE &97 (151) Write to SHEILA All
No change in function on the B+ and Master 128. On the Compact, the calls are implemented, but can only respond if hardware is connected.

OSBYTE &98 (152) Examine buffer status All
OSBYTE &99 (153) Insert character into buffer All
OSBYTE &9A (154) Write to video ULA ctrl reg All
OSBYTE &9B (155) Write to video ULA palette reg All
OSBYTE &9C (156) Read/Write ACIA ctrl reg All
OSBYTE &9D (157) "Fast" Tube BPUT All
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &9E (158) Read from speech processor B/B+
OSBYTE &9F (159) Write to speech processor B/B+
No change in function on the B+. On the Electron and Master Series, speech is not implemented or offered.

OSBYTE &A0 (160) Read VDU variable base address All
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &A1 (161) Read CMOS RAM/EEPROM M/C
This reads bytes from the 146818 CMOS chip on the Master 128, or the EEPROM chip on the Compact. Fifty locations are readable on the Master 128 (X=0 to X=49), and either 128 or 256 on the Compact. A special call (X=255) for the Compact only determines which if any type of EEPROM is present. Note that this call will not read the clock registers of the 146818.

Entry : X=byte to be read (0-49 on Master 128, 0-127/254 on Compact)
X=255 (Compact only) - Is EEPROM a 128 or 256 byte device?
Exit : X=corrupt:Y=contents of RAM/EEPROM location
If X=255 on Compact:

Y=0	No EEPROM present
Y=&7F	128 byte EEPROM present
Y=&FF	256 byte EEPROM present

OSBYTE &A2 (162) Write CMOS RAM/EEPROM M/C
This writes bytes to CMOS RAM or EEPROM. The address (see OSBYTE &A1 for range) is placed in X and the data in Y. Note that this call will not allow you to change byte 0 on either machine, or bytes 127, 128 or 255 on the Compact with a large EEPROM. This is for Econet security reasons. See Appendix K for details of the CMOS RAM/EEPROM usage.

Entry : X=address on CMOS RAM/EEPROM
Y=Byte to be written
Exit : X=preserved:Y=corrupt

OSBYTE &A3 (163) Reserved for third parties M/C
This call is reserved by Acorn for allocation to third party software houses, so therefore it controls a variety of functions.

OSBYTE &A4 (164) Check if data is 6502 code M/C
Entry : Y=high byte of address of code to be checked
X=low byte of address of code to be checked

The routine checks that the location pointed to by XY+7 contains &00 then "(C)". If it does, the processor code type byte at XY+6 is checked. If bit 7 is clear, a BRK occurs with the fatal error (ERR=0) "This is not a language". If bit 7 is set but bits 3 to 0 are not 0000 (&0) or 0010 (&2), the fatal error (ERR=0) "I cannot run this code" occurs. The routine is used when an attempt is made to start a ROM as a language, to test whether (a) it is a language, and (b) if it whether or not it is 6502 machine code or exclusively written for a certain co-processor.

A Tube Host program on a non-65C02 co-processors should do the "code" test on its own code types in bits 3-0 of &8007. These are listed in Chapter 6.

OSBYTE &AS5 (165) Read output cursor position M/C
This call, new to the Master series, returns the output cursor position, when cursors are separated for editing. The call is complementary to OSBYTE &86, which returns the input cursor position. If cursors are not separated, both calls return the same result.

Entry : No parameters

Exit : X=horizontal cursor position
Y=vertical cursor position.

OSBYTE &A6 (166) Start address of OS variables (lo) All
OSBYTE &A7 (167) Start address of OS variables (hi) All
OSBYTE &A8 (168) ROM pointer table (lo) All
OSBYTE &A9 (169) ROM pointer table (hi) All
OSBYTE &AA (170) ROM info table (lo) All
OSBYTE &AB (171) ROM info table (hi) All
OSBYTE &AC (172) Key translation table (lo) All
OSBYTE &AD (173) Key translation table (hi) All
OSBYTE &AE (174) VDU variables (lo) All
OSBYTE &AF (175) VDU variables (hi) All
OSBYTE &B0 (176) R/W CFS timeout counter All
OSBYTE &B1 (177) R/W input source All
OSBYTE &B2 (178) R/W keyboard semaphore All
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &B3 (179) OSHWM/ROM semaphore All
On Models B/B+/E this call reads and writes the primary OSHWM ignoring any space claimed by character fonts. On the Master Series, the font is permanently exploded without loss of user RAM, so OSBYTEs &B3 and &B4 will always be the same. Thus, &B3 has been used for a different purpose - it now returns the value of the ROM semaphore set by OSBYTE &16/&17. The call follows the NEW=(OLD AND Y) EOR X rule.

B/B+/E
Entry : X=0:Y=255 returns hi-byte of OSHWM in X
X=value:Y=0 assigns new OSHWM value

M/C
Entry : X=0:Y=255 returns semaphore value in X
X=value:Y=0 assigns new value directly to semaphore

OSBYTE &B4 (180) **R/W current OSHWM** **All**
OSBYTE &B5 (181) **R/W RS423 mode** **All**
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &B6 (182) **Font explosion/NoIgnore** **All**
On B/B+/E this was used to determine the size of the font explosion caused by the last OSBYTE &14, effectively by returning the X parameter of that last call. On the Master Series, this information is unnecessary as the font is permanently completely exploded, so the call has been used to determine whether there is a printer ignore character, (not what the character is - that comes from OSBYTE &F6). On Models B/B+/E you have no choice, there must be at least one ignore character.

B/B+/E
Entry : X=0:Y=255 returns explosion state (0-7) in X
X=value:Y=0 writes value, just as OSBYTE &14

M/C
Entry : X=0:Y=255
Exit : X>127 = There is no ignore character
X<128 = There is an ignore character. OSBYTE &F6 will reveal it.

OSBYTE &B7 (183) **CFS/RFS switch** **All**
On all machines prior to the Compact, this call returned 0 in X if the last filing system selected (of the two) was tape (CFS), and 2 if it was ROM (RFS). Any disc etc selections had no effect. On the Compact the location starts containing 0, but if ROM is selected, it then returns 2. BREAK resets the value (stored in &247) to 0.

OSBYTE &B8 (184) **Read copy of video ULA ctrl reg** **All**
OSBYTE &B9 (185) **Read "" video ULA palette reg** **All**
OSBYTE &BA (186) **R/W ROM active at last BRK** **All**
OSBYTE &BB (187) **R/W ROM socket of BASIC** **All**
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &BC (188) Read current ADC channel All
On the B/B+/M this returns the channel currently being processed - the channel least suitable for reading at that moment. On the Compact, a figure is supplied which tallies with the rest of the Compact ADC emulation. The only difference to note is that the Compact defaults to 2-channel 'conversion'. 4-channel 'conversion' can be of course be set up with OSBYTE &10. See Appendix I. This call should not be used in write mode - use OSBYTE &11 instead. The call is not implemented on an Electron.

OSBYTE &BD (189) R/W max ADC channel no. All
This call returns the maximum ADC channel number, as set with OSBYTE &10. The Compact emulation totally supports this. See Appendix I. The call is not implemented on an Electron.

OSBYTE &BE (190) R/W ADC conversion type All
On the B+ and Master 128, the call sets returns the conversion resolution.

To write: X=&00 or &0C (12 bit conversion)
X=&08 (8-bit conversion), and Y=0.
To read: X=&00 and Y=&FF.

Faster but less accurate is 8-bit conversion — however in ADVAL you lose the full accuracy anyway. On the Compact the call is used to provide a variety of controls over the pseudo A-to-D system. Details are provided in Appendix I. The call is not implemented on an Electron.

OSBYTE &BF (191) **R/W RS423 use flag** **All**
OSBYTE &C0 (192) **Read 6850 control flag** **All**
OSBYTE &C1 (193) **R/W flash counter** **All**
OSBYTE &C2 (194) **R/W mark period count** **All**
OSBYTE &C3 (195) **R/W space period count** **All**
OSBYTE &C4 (196) **R/W auto-repeat delay** **All**
OSBYTE &C5 (197) **R/W auto-repeat period** **All**
OSBYTE &C6 (198) **R/W *EXEC file handle** **All**
OSBYTE &C7 (199) **R/W *SPOOL file handle** **All**
OSBYTE &C8 (200) **R/W ESCAPE,BREAK effect** **All**
OSBYTE &C9 (201) **R/W keyboard disable status** **All**
OSBYTE &CA (202) **R/W keyboard status byte** **All**
OSBYTE &CB (203) **R/W RS423 handshake extent** **All**
OSBYTE &CC (204) **R/W RS423 suppression flag** **All**
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &CD (205) **R/W cassette/RS423 flag** **All**
No change in function on the B+ and Master Series. The Compact has no cassette interface, so calling this command with &40 (or any other value) will not alter the fact that RS232 output is sent to any RS232 port present.

OSBYTE &CE (206) **R/W OS call intercept status** **All**
OSBYTE &CF (207) **R/W OSRDCH intercept status** **All**
OSBYTE &D0 (208) **R/W OSWRCH intercept status** **All**
These calls are concerned with Econet interception. They can however be used for other purposes. No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &D1 (209) **R/W speech suppression status** **All**
This call does not exist on the Master Series or Electron, as speech is not offered as an option.

OSBYTE &D2 (210)	**R/W sound suppression status**	**All**
OSBYTE &D3 (211)	**R/W BELL channel**	**All**
OSBYTE &D4 (212)	**R/W BELL env. no/amplitude**	**All**
OSBYTE &D5 (213)	**R/W BELL frequency**	**All**
OSBYTE &D6 (214)	**R/W BELL duration**	**All**
OSBYTE &D7 (215)	**R/W startup message etc.**	**All**
OSBYTE &D8 (216)	**R/W length of soft key string**	**All**
OSBYTE &D9 (217)	**R/W paged mode count**	**All**
OSBYTE &DA (218)	**R/W VDU queue length**	**All**
OSBYTE &DB (219)	**R/W TAB character value**	**All**
OSBYTE &DC (220)	**R/W ESCAPE character value**	**All**

No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &DD (221)	**R/W char &C0 to &CF status**	**All**
OSBYTE &DE (222)	**R/W char &D0 to &DF status**	**All**
OSBYTE &DF (223)	**R/W char &E0 to &EF status**	**All**
OSBYTE &E0 (224)	**R/W char &F0 to &FF status**	**All**
OSBYTE &E1 (225)	**R/W function key status**	**All**
OSBYTE &E2 (226)	**R/W SHIFT-function key status**	**All**
OSBYTE &E3 (227)	**R/W CTRL-function key status**	**All**
OSBYTE &E4 (228)	**R/W CTRL-SHIFT-fn key status**	**All**

On B/B+/E and Master 128, these commands, which determine how a character insertion/function keypress shall be interpreted, have at the moment only three possibilities:

X=0	Ignore the ins/key
X=1	Return soft string no. (ins/key AND &0F)
X=2-255	Return character X + (ins/key AND &0F)

where ins/key equals the ASCII code inserted or the number of the function key pressed, depending on which call is being used.

On the Master Compact, a new feature is implemented, X=2 no longer returns a 2 for f0/insert &C0, 2 for f1/&C1 and so on. Instead, it returns two characters to the current input stream (ie, to OSRDCH), the first being NUL (&00) and the second being the appropriate key number/inserted code. For the function keys the key numbers are:

Normal	&80 - &8F
Shifted	&90 - &9F
Control	&A0 - &AF
SHIFT/CTRL	&B0 - &BF

The idea behind this system is that you can use the code key on the Compact to generate ASCII characters in the range 128-255, yet still use the function keys, doing this by interpreting say ASCII 134 as a character, but 0 followed by 134 as function key 6.

OSBYTE &E5 (229) R/W ESCAPE key status All
OSBYTE &E6 (230) R/W ESCAPE effects flags All
OSBYTE &E7 (231) R/W IRQ bit mask - user 6522 All
OSBYTE &E8 (232) R/W IRQ bit mask - 6850 All
OSBYTE &E9 (233) R/W IRQ bit mask - system 6522 All
OSBYTE &EA (234) Read Tube flag All
No change in function on the B+ and Master 128. On the Compact, the call is supported. Reading will always return X=0 (No Tube). Writing will have no effect. Acorn do not recommend using OSBYTE &EA in write mode. In practice *FX234,0,0 followed by a language start-up, will happily return to the I/O processor. Pressing BREAK will of course reselect the co-processor.

OSBYTE &EB (235) Read speech processor presence All
This call does not exist on the Master Series and Electron as speech is not an option on these machines. It always returns X=0 (no speech system present). On the B+, it works normally.

OSBYTE &EC (236) R/W *FX3 status All
OSBYTE &ED (237) R/W *FX4 status All
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &EE (238) Set base for numeric keypad M/C
This new call sets the codes returned by the pad. Individual codes cannot be set — instead a base code is used. All ASCII codes generated are thus calculated from that base code. The displacements for each key are calculated as the ASCII value of keycap legend minus 48. So keypad 0 is set to the base code, keypad 1 to the base code+1, keypad

RETURN to the base code minus 35 (13-48=-35) and so on. The default setting is 48.

Entry : X=0:Y=255 Read base setting
X=value:Y=0 Write new base
Exit: X=old value:Y=contents of next location

OSBYTE &EF (239) R/W *FX114 status B+/M/C
This call reads the shadow screen state as set by OSBYTE &72. However, MOS 1.20, which does not support shadow screens, will return a value of X=0 for this call. Usually, you would want to treat a Model B as 'no shadow', so a machine type call (OSBYTE &00, OSBYTE &81 X=0:Y=&FF) should be issued first, and this call only issued when you are sure the machine in question is a B+ or Master Series.

Entry : X=0:Y=255
Exit : X=0 Shadow mode is selected or will be at the next mode change (or this is a Model B with MOS 1.20)
X=1 This is not the case. A *FX114,1 state prevails.

OSBYTE &F0 (240) Read UK/US flag M/C
This call returns a number indicating the country for which the MOS is intended. If you wish, your program may check this call, and if you find a USA machine, adjust your screen depths accordingly — 25 lines where UK has 32, 22 lines where UK has 25, 20 lines in Mode 7, 200 graphic pixels in the Y axis.

Entry : X=0:Y=255
Exit : X=0 UK MOS, X=1 USA MOS
Y=contents of the next byte

OSBYTE &F1 (241) R/W *FX1 (user) setting All
OSBYTE &F2 (242) Read RAM copy of serial ULA All
OSBYTE &F3 (243) R/W timer switch state All
OSBYTE &F4 (244) R/W soft key consistency flag All
OSBYTE &F5 (245) R/W *FX5 (printer) setting All
OSBYTE &F6 (246) R/W *FX6 (ignore) setting All
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &F7 (247) R/W BREAK intercept code All
OSBYTE &F8 (248) R/W BREAK intercept code All
OSBYTE &F9 (249) R/W BREAK intercept code All
No change in function on the B+ and Master Series. See Model B User Guides. Note that a BREAK with a memory clear, set with OSBYTE &C8, used to cause a hang-up if a BREAK intercept was in force. Master Series machines do not hang-up in this situation.

OSBYTE &FA (250) Read *FX112 status M/C
This reads the current setting of OSBYTE &70, the bank of memory (shadow or main) to which data is written.

Entry : X=0:Y=255
Exit : X=0 Writing as specified by mode
X=1 Always writing to main memory
X=2 Always writing to shadow memory

It is not possible to use this call to write. Instead use OSBYTE &70.

OSBYTE &FB (251) Read *FX113 status M/C
This reads the current setting of OSBYTE &71, which select whether main or shadow RAM will be displayed as the visible screen.

Entry : X=0:Y=255
Exit : X=0 Display determined by mode
X=1 Always displaying main memory
X=2 Always displaying shadow memory

It is not possible to use this call to write. Instead use OSBYTE &71.

OSBYTE &FC (252) R/W current language ROM no All
OSBYTE &FD (253) R/W last BREAK type All
No change in function on the B+ and Master Series. See Model B User Guides.

OSBYTE &FE (254) RAM size /SHIFT key effect All
On the Model B, this call returns the main RAM fitted - 16k for a Model A (X=&40), and 32k for a Model B (X=&80). This was modifiable, and by typing *FX254,64 then CTRL-BREAK — a Model B would

pretend to be a Model A. On the B+ and Electron the call has no effect, but will return X=0 on an Electron and X=1 on B+.

On the Master Series, the call has a totally different purpose, to enable or disable (default) the effect of the SHIFT key on the numeric pad. If enabled, then when SHIFT or SHIFT LOCK is pressed, the keypad keys will return their shifted equivalents. Those legends whose main keyboard equivalents are already shifted return the unshifted key from the main keyboard.

Model B/B+/E

Entry : X=0:Y=255 Read RAM size
X=value,Y=0 Write RAM size
Exit : X=0 Electron
X=1 B+
X=&40 Model A
X=&80 Model B
NB : Writing &80 to a Model A causes a crash when MODE 0-3 selected

Master /Compact

Entry : X=0:Y=255 to read
X=0 enable, X>0 disable:Y=0
Exit : X=0 enabled,X>0 disabled:Y=contents of next location

OSBYTE &FF (255) R/W start up options All

This call reads/writes eight system setup bits. On the Model B/B+ only, the default setting of these bits is determined by eight DIL switches (ON=0 OFF=1) which can be fitted on the keyboard, thus giving some small degree of system configuration. If the switch is not fitted it acts as though all bits are off, and gives a default of &FF.

With the Master non-volatile system, these switches were no longer required, and thus the call changes slightly. Note that screen mode and disc drive settings are now made with *CONFIGURE MODE and *CONFIGURE FDRIVE, not with this call.

Model B/B+/Electron

b0-2 Screen Mode (000=0 to 111=7)
b3 1=!BOOT on SHIFT-BREAK, not on BREAK
0=!BOOT on BREAK, not on SHIFT-BREAK

b4-5 Set disc speed (00=fastest, 11=slowest)
b6 Reserved
b7 0=DNFS starts in DFS
1=DNFS starts in NFS
(NB : No keyboard switch equivalent on Electron)

Master 128/Compact

b0-2 No effect
b3 1=!BOOT on SHIFT-BREAK
0=!BOOT on BREAK
(irrespective of *CONFIG BOOT setting)
b4-7 No effect

Notes concerning unchanged calls

1. On the Master Series OSBYTE &09 and &0A are not the only way of selecting flash rates, for example VDU 23,9,m,0,0,0,0,0,0,0 and VDU 23,10,n,0,0,0,0,0,0,0 are identical to *FX9,m and *FX10,n.

2. OSBYTEs &10 and &11 are only emulated by the Compact as it has no ADC hardware. See Appendix I.

3. OSBYTE &12. The size and place of the function key buffer has changed. It is now in private RAM (ANDY) and occupies 1024 bytes.

4. OSBYTE &15/&0F/&80. There is no speech buffer on Master Series machines.

5. OSBYTE &77 closes *SPOOLON as well as normal *SPOOL files on M/C

6. OSBYTE &78. There are more keys on a Master. See Appendix F for tables of Master keys.

7. OSBYTE &9E and &9F (Read and Write speech processor) are offered to Paged ROMs on the Master Series.

OSWORD calls

OSWORD is the name of a major Operating System entry point at &FFF1. Like OSBYTE, the function invoked is dependent on the value of the accumulator. Unlike OSBYTE, only a few accumulator values are used by the Operating System, so many other firmware programs use OSWORD as an entry point for their low level routines rather than OSBYTE for which nearly all possible accumulator values are already used. Some of this firmware is from Acorn, and others are third-party products.

OSWORD calls always pass and return their parameters through a block of memory, which you can define anywhere in the system (Some calls require the block to be on the I/O processor). The address of this block of memory is placed in the X (low) and Y (high) registers before calling OSWORD. The size of this block is dependent on the function, but should not normally exceed 16 bytes. If more than 16 bytes worth of parameters are required, you should use the OSWORD block to point to another area of memory. Like all system calls, OSWORD may enable interrupts during execution.

OSWORD calls with an accumulator value of &E0 (224) or greater are not passed to the Operating System but to the address in held in USERV (&200/&201 in the I/O processor). This is a useful way of passing data across the Tube. Below is a list of all known OSWORD calls implemented by Acorn and others. As with OSBYTE calls whose function has not changed since the Model B are, for reasons of space, listed but not documented.

When using the Tube, the number of bytes transferred in each direction depends on the OSWORD call number. The first 16 OSWORD calls (&00-&0F) use a parameter block whose size is determined by a table in the Tube OS. Calls from &10-&7F always use a 16-byte block. Calls from &80-&FF take the contents of the first byte of the block as the length of the input parameter block, and the contents of the second byte of the block as the number of parameters to return.

OSWORD Calls in Numeric Order

In the list below, the identification 3P ROM is used to indicate those calls supplied by third party firmware.

OSWORD &00 (0) **Read an input line to memory** **All**
OSWORD &01 (1) **Read system clock** **All**
OSWORD &02 (2) **Write system clock** **All**
OSWORD &03 (3) **Read interval timer** **All**
OSWORD &04 (4) **Write interval timer** **All**
These have not changed in function on the B+ and Master Series. See Model B User Guides.

OSWORD &05 (5) **Read byte of I/O memory** **All**
OSWORD &06 (6) **Write byte to I/O memory** **All**
These have not changed in function, but additionally on the BBC B+ support access to the shadow screen and 12k sideways RAM through the use of I/O address &FFFExxxx. See Chapter 5 for more details.

OSWORD &07 (7) **Generate a sound** **All**
OSWORD &08 (8) **Define an envelope** **All**
OSWORD &09 (9) **Read pixel colour** **All**
OSWORD &0A (10) **Read a character definition** **All**
OSWORD &0B (11) **Read the VDU palette** **All**
OSWORD &0C (12) **Write the VDU palette** **All**
OSWORD &0D (13) **Read graphics cursor pos** **All**
These have not changed in function on the B+ and Master Series. See Model B Guides.

OSWORD &0E (14) **Read CMOS clock** **M/C**
The CMOS clock may be read in three different ways, the manner being determined my a value placed in the parameter block.

1. Return time/date

Entry : XY+0=0

This call returns the time and date as a 24 byte string (starting at XY+0) in the form "Sun,11 Jan 1987.14:02:55" (without the quotes) followed by a carriage return.

2. Return time/date as BCD data.

Entry : XY+0=1
Exit : XY+0 Year (00-99)
XY+1 Month (01-12)
XY+2 Date (01-31)
XY+3 Day (01-07) 01=Sun 02=Mon..
XY+4 Hours (00-23)
XY+5 Minutes (00-59)
XY+6 Seconds (00-59)

3. Convert a BCD value to a text string

Entry : XY+0=2
This call expects the data to be set up as returned by call 1 above, (but from XY+1, not XY+0) and will return the data in the format in call 0. You would use this call to log certain times, perhaps in a control application, using the more compact binary format (which is also more suitable for processing), and then later, to print those times, the call is used to create strings. Note that there are problems using this call with some early 6502 second processors, in that only one parameter is passed across by OSWORD instead of the correct number,

OSWORD &0F (15) Write CMOS clock M/C
The CMOS clock may be written to in three different ways, the manner being determined my a value placed in the parameter block.

1. Write time only

Entry : XY+0=8
Exit : XY+1 to XY+8 are set to a string representing the current time in the format "13:53:00" (without the quotes). No carriage return is needed.

2. Write date only

Entry : XY+0=16

Exit : XY+1 to XY+15 are set to a string representing the current date in the format “Sun,06 Feb 1987” (without the quotes). No carriage return is needed.

3. Write date and time

Entry : XY+0=24
Exit : XY+01 to XY+15 contains the date string ie “Wed,09 Nov 1988” (without the quotes).
XY+16 contains &2E (ASCII code for a full stop).
XY+17 to XY+24 contains the time string ie “13:10:55” (without the quotes).

OSWORD &10 (16) Econet transmit NFS/ANFS
This call is used to transmit data co-operatively, ie, there must be a receive station also involved (using OSWORD &11). It can also be used for immediate operations as follows:

XY=&81	Peek a remote station
XY=&82	Poke a remote station
XY=&83	JSR to remote station routine
XY=&84	Cause an event at remote station
XY=&86	Halt program running on remote station
XY=&87	Continue a halted program
XY=&88	Read machine type of remote station

For more details, see the Econet Advanced User Guide.

OSWORD &11 (17) Econet receive NFS/ANFS
This call also applies to co-operative data transfers. See OSWORD &10, and OSBYTE &32 to &34. There are so many variants of this call that it would be impossible to document them here. For more information see the Econet Advanced User Guide.

OSWORD &12 (18) Econet read args NFS/ANFS
This call reads an Econet argument block. No parameters are required on entry. On exit, XY to XY+1 hold the station number. XY+2 onwards holds an argument block buffer. OSWORD &13 (XY=&09) will find the number of arguments received. For more information see the Econet Advanced User Guide.

OSWORD &13 (19) R/W station info NFS/ANFS
This controls various information concerning your own station. The function is determined by the byte at XY. Relevant data is placed at XY+1 onwards. There are too many functions to describe in detail here. See the Econet Advanced User Guide.

OSWORD &14 (20) Talk to fileserver etc. NFS/ANFS
Again, the value of XY determines the function of this call

XY=&00	Communicate with the fileserver (26 functions)
XY=&01	Send text message to another station
XY=&02	Cause a fatal error at a remote machine

OSWORD &28 (40) Graphics processor 3P ROM
OSWORD calls &28 to &31 have been provisionally allocated to the Millipede Prisma 2 graphics processor.

OSWORD &40 (64) Mouse/Trackerball 3P ROM
OSWORD &41 (65) Mouse/Trackerball 3P ROM
Used by AMS Ltd — Full details of this OSWORD call are supplied in the appropriate Advanced Memory Systems manual. As the Marconi RB2 trackerball is mutually exclusive with the AMX mouse, the call may also be used to drive the former. Such code exists, for example, in the Master AIV System, provided to run the BBC Domesday videodiscs. If writing code for a mouse it is also permissible to use the trackerball reserved space in page &D — see Appendix D.

OSWORD &42 (66) Transfer to/from SRAM B+/M/C
This call is implemented in the 1770 DFS in the BBC B+, and in the MOS on the Master series. It is the low-level equivalent of *SRREAD and *SRWRITE. The parameter block is preserved after a call, which is useful for multiple accesses. The call copies a block of memory from main RAM to Sideways RAM (write) or vice versa (read).

XY+0	=&00	Read from absolute address
	=&40	Read from pseudo address
	=&80	Write to absolute address
	=&C0	Write to pseudo address

XY+1..4	32-bit main memory address
XY+5..6	16-bit number of bytes to transfer
XY+7	ROM id for absolute addressing (use ROM numbers or &10...&13 for W...Z)
XY+8..9	16-bit sideways address (absolute or relative)

OSWORD &43 (67) Load/save to SRAM B+/M/C

This call is also implemented in the 1770 DFS in the BBC B+, and in the MOS in the Master series. It is the low-level equivalent of *SRLOAD and *SRSAVE. The parameter block is preserved after a call - unless it is overwritten during the transfer. After the call a block of Sideways RAM will have been written to or read from disc/tape. An optional buffer can be used which will speed up transfer into Sideways RAM.

XY+0	=&00	Read from absolute address
	=&40	Read from pseudo address
	=&80	Write to absolute address
	=&C0	Write to pseudo address
XY+1..2		Address of filename (must be on I/O processor)
XY+3		ROM id for absolute addressing (Numbers or &10...&13 as above)
XY+4..5		16-bit sideways start address (absolute or relative)
XY+6..7		16-bit number of bytes to save (ignored on load)
XY+8..9		16-bit buffer (I/O) start address
XY+10..11		16-bit size of buffer

If XY+10 and XY+11 are zero, a default buffer in HAZEL is used, and user RAM remains uncorrupted. This is equivalent to not using the 'Q' option in *SRLOAD/*SRSAVE. Note that the buffer used is ROM workspace, not the 'safe' &DD00-&DEFF area. If XY+11 is greater than &7F then the area from OSHWM to HIMEM is used as a buffer. Otherwise the buffer used is as defined above.

Note that *SRROM and *SRDATA have no assembly language equivalent. For further information on OSWORD &42 and &43 see Chapter 5.

OSWORD &44 (68) AMX Mouse support 3P ROM
OSWORD &45 (69) Aries B32 Move/swap 3P ROM
OSWORD &46 (70) Allocated to BBC Soft 3P ROM
For further details of these calls contact the appropriate companies.

OSWORD &47 (71) to &5E (94) 3P ROM
OSWORD calls &47 to &5E are available for third-party use. If you are designing a piece of software for general distribution, and you require an allocated number, contact The Third Party Co-ordinator, Acorn Computers Ltd, 645 Newmarket Road, Cambridge CB5 8PD. By doing this, you can at least hope that your software won't clash with that of others.

OSWORD &5F (95) BBC Soft 'Monitor' 3P ROM
This third party call will only work if you have Monitor by BBC Soft. The call is used for reading memory, including sideways memory, across the Tube.

OSWORD &60 (96) Read MSN and status byte VFS
This call reads the Master Sequence Number (ie the number of times a disc/directory has been written to) and status byte of the currently mounted videodisc. The call is implemented in the Videodisc Filing System, VFS. There are no parameters for entry. On exit, two bytes of information are placed at XY. The first byte (XY) contains the Master Sequence Number for the current directory (in BCD) — ie how many times that directory was written to prior to mastering onto videodisc. XY+1 contains a status byte, in which each bit has the following meaning:

Bit	Meaning (when bit is set)
b0	File 'ensuring' in progress - IRQ pending (will never happen)
b1	Bad Free Space Map
b2	*OPT 1 setting
b3	Not used

b4	Not used	
b5	LVROM controller present	
b6	Tube in use by VFS	
b7	Tube present (there are other ways of reading this)	

OSWORD &62 (98) Access LVROM controller VFS

This is a command to transfer videodisc sectors directly into memory. On entry the parameter block must be set up as follows:

XY+0	0	
XY+1..4	Start memory address	
XY+5	Videodisc controller function code	
	&00 (0)	Test drive ready (XY!11=0)
	&01 (1)	Seek track 0 (XY!11=0)
	&03 (3)	Request status (OSWORD &63 is better)
	&08 (8)	Read data (compatible with ADFS)
	&1B (27)	Start/stop the unit (XY!11=0) XY+9=0 Stop XY+9=1 Start
	&C8 (200)	Read F-code result from LVDOS
	&CA (202)	Transmit F-code to LVDOS (For F-codes see the AIV manual)
XY+6	b5-7	Drive number
	b0-4	Sector Number (bits 16-20)
XY+7	Sector Number (bits 8-15)	
XY+8	Sector Number (bits 0-7)	
XY+9	No of sectors	
XY+10	0	
XY+11	Length of data in bytes (ignored if XY+9 is non-zero). This is a 32-bit value.	

The drive number in XY+6 is ORed with the current drive, to give the drive number to be accessed. Thus if Drive 1 is currently *MOUNTed, it is not possible to access Drive 0, unless it is first *MOUNTed. On exit the byte at XY contains 0 if the operation occurred satisfactorily. If it did not, the error codes are:

&00 (0) Controller error - will never happen
&02 (2) Drive door open
&03 (3) Media error, ie disc dirty
&05 (5) Bad SCSI command

OSWORD &63 (99) Read last VFS error info **VFS**

This call takes no entry parameters and returns information about the last VFS error that occurred. It should only be used when VFS is selected as the current filing system, and should be called immediately after the error happens.

Entry : No parameter setting is required
Exit : XY to XY+2 (b0-4) contain the sector number where error occurred.
XY+2 (b5-7) contains the current drive number
XY+3 (b0-6) contains the error number (see OSWORD &62)
XY+3 (b7) if set, means ignore channel number (XY+4) of error, as it isn't valid, use sector number only.
XY+4 Channel number of file where error happened

OSWORD &64 (100) Read current F-code **VFS**

This will place the current F-code command string at XY, which could then be printed out using BASIC's $variable feature. There are no entry parameters. This only works with VFS selected.

OSWORD &70 (112) Read MSN and status byte **ADFS**

This call is implemented in ADFS and has an identical form to OSWORD &60. XY contains the number of times the current directory has been written to, and XY+1 contains:

Bit	Meaning (when bit is set)
b0	File 'ensuring' in progress
b1	Bad Free Space Map
b2	*OPT 1 setting
b3	Not used
b4	Not used
b5	Winchester disc controller present
b6	Tube in use by ADFS

b7 Tube present (there are other ways of reading this)

OSWORD &71 (113) Read free space ADFS

This call is implemented in ADFS. There are no parameters for entry. On exit XY to XY+3 contain a 32-bit number which is the free space available on the disc. This is the value used by *FREE (which of course also converts the number to decimal). Listing 4.3 is an example of this call.

Listing 4.3

```
10 REM Read ADFS Free Space
20 REM (c) Dave Atherton 1987
30 REM MOS : A Dabhand Guide
40 :
50 *MOUNT 0
60 A%=&71:X%=&80:Y%=0
70 CALL &FFF1
80 PRINT "There are ";!&80;" bytes free"
```

OSWORD &72 (114) Read/write sectors ADFS

This is a command to transfer ADFS disc sectors directly to or from memory. On entry the parameter block must be set up as follows:

XY+0	0
XY+1..4	Start memory address
XY+5	&08 (8) Read Data &0A (10) Write Data &0B (11) Seek Track (undocumented by Acorn) — Other values ignored.
XY+6	b5-7 Drive number b0-4 Sector Number (bits 16-20)
XY+7	Sector Number (bits 8-15, ie high byte)
XY+8	Sector Number (bits 0-7, ie low byte)
XY+9	No of sectors (read operations only)
XY+10	Unused
XY+11	Length of data in bytes (write operations only). This is a 32-bit value.

The drive number in XY+6 is ORed with the current drive, in the same way as explained in OSWORD &62 above. For read operations, byte XY+10..14 are ignored. For write operations, byte XY+9 is ignored. On exit the byte at XY contains 0 if the operation took place without error, if not, the error codes are:

&48 (72)	CRC error
&50 (80)	Sector not found
&60 (96)	Bad command
&61 (97)	Bad address
&63 (99)	Volume error
&65 (101)	Bad drive

NB: This call actually passes a command to a SCSI (Small Computer Systems Interface), also known as SASI (Shugart Associates Systems Interface) controller, and if a Winchester drive is fitted, this device will be present and receive the command directly. For floppy discs, ADFS contains a SCSI emulation subsystem, which takes SCSI commands (&08, &0A and &0B only) and translates them into commands to drive the 1770 chip. Theoretically this subsystem could be replaced with another, but in existing releases of ADFS, the subsystem entry points are not documented.

OSWORD &73 (115) Read last error info ADFS

This call returns information about the last ADFS error that occurred, and should only be used when ADFS is selected as the current filing system, and should be called immediately after the error happens.

Entry : No parameter setting is required

Exit : XY to XY+2 (b0-4) contain the sector number where error occurred.
XY+2 (b5-7) contains the current drive number.
XY+3 (b0-6) contains the error number (see OSWORD &72)
XY+3 (b7) if set, means ignore byte at XY+4, use sector number only
XY+4 Channel number of file where error happened

OSWORD &7A (122) Teletext commands 3P ROM

This call controls many functions applicable to Teletext. The call is serviced by the Acorn TFS (Teletext Filing System), the BBC Soft Advanced Teletext System, and the proprietary systems produced by

other teletext decoder manufacturers, and will not work if these are not present. The functions are determined by a code byte in XY. If you are writing your own software to support the teletext hardware, you may wish to support this call. See the relevant manuals for details.

OSWORD &7B (123) Modem commands 3P ROM
This call controls many functions applicable to telephone communications. It is serviced by the Acorn Prestel ROM, and also CommSoft, by SoftMachinery (authors of the former). If you are writing your own communications software, then you may wish to support this call. See the relevant manual for details.

OSWORD &7D (125) Read Master Sequence No. DFS
This call returns the number of times a DFS catalogue has been written to since formatting, MOD 100. The number is returned as a binary quantity, but any display should use hex display routines, as it is internally stored (Sector 001, byte 4) in binary coded decimal.

No entry conditions. On exit XY+0 contains the number.

OSWORD &7E (126) Read disc size DFS
This command applies to DFS discs only. There are no entry parameters. On exit XY to XY+2 contain the 24-bit number representing the number of bytes on this drive (&19006 for 40 track discs and &32000 for 80 track discs). Many protection schemes alter these bytes, and they are not a reliable guide to whether is disc is formatted in 40 or 80 tracks. To do that, a much more reliable guide would be to try and read track IDs for track 2 using OSWORD &7F and see if it contains the same IDs as track 1. If it does you are probably trying to read a 40 track disc in an 80 track drive and a suitable message can be issued - however track IDs can also be altered in the quest for uncopyable software.

This command will still work if DFS is not selected as the current filing system. This could cause problems if ADFS is selected and there is an ADFS disc present in the drive.

OSWORD &7F (127) Send Command to 8271 DFS
This is a multi-purpose command to send commands to the disc controller on the 8271 DFS. The 1770 DFS contains emulation of many of

the commands, which it then translates into its own data format. Therefore under 1770 DFS, all commands must be issued in 8271 format. See the section on the 1770 controller in Chapter 8 for more details.

The entry format is:

XY+0	Drive number (If &FF, then use current drive)
XY+1..4	32-bit memory address
XY+5	Number of parameters
XY+6	8271 command (See table in Chapter 8)
XY+7..XY+n	Parameters

After the call, XY+n+1 contains 0 if the operation took place without error, or an error code:

&08 (8)	Clock error
&0A (10)	Late DMA (will never occur on a BBC Micro)
&0C (12)	Error in CRC of ID field
&0E (14)	Error in CRC of data field
&10 (16)	Drive not ready (cannot occur)
&14 (20)	Track 0 not found
&16 (22)	Write fault (ie disc is write-protected)
&18 (24)	Sector not found
&FE (254)	Emulation of command not supported (1770)

NB: It is possible to use OSWORD &7F when DFS is not the currently selected filing system, but this isn't wise as the call uses DFS filing system workspace, which may be used for other purposes when another filing system is active.

Listing 4.4 shows how to use OSWORD &7F, in this case to read the sector IDs of a disc. Command &5B reads the sector IDs from a disc, which is useful when you need to retrieve data from a disc. Note how the error code is tested for, this is the byte following the last parameter byte. The program prints out the track number on the disc which should match the physical track number given after 'Track', but may not; the head number, which should always be 0, the logical sector number, which usually doesn't match the physical sector number due to skewing, and finally the sector size. A code is used where

0=128 bytes, 1=256 bytes etc. This can be expressed by the formula in 'FNconv'.

Listing 4.4

```
 10 REM Read sector IDs (OSWORD &7F example)
 20 REM (c) Dave Atherton 1987
 30 REM for B/B+/E/M/C with DFS
 40 REM MOS: A Dabhand Guide
 50 :
 60 MODE 7
 70 @%=&A07
 80 DIM b &20,B% 100
 90 osword=&FFF1
100 PRINT "Sector ID Reader"
110 INPUT"How many tracks " T%
120 IF T%=0 T%=40
130 FOR I%=0 TO T%-1
140 PRINT TAB(14,4);"Track ";I%'
150 PRINT TAB(5);"Trk", "Head", "Sec No","Size"'
160 PROCosword7F (0, B%,3,&5B,I%,0,10)
170 FORJ%=0 TO 39 STEP4
180 FOR K%=J% TO J%+2
190 PRINT B%?K%;" ";
200 NEXT
210 PRINTFNconv (B%?K%)
220 NEXT
230 VDU30
240 NEXT
250 END
260 :
270 DEF PROCosword7F (drv,buf,pars,cmd,par1,par2,par3)
280 ?b=drv
290 b!1=buf
300 b?5=pars
310 b?6=cmd
320 b?7=par1
330 b?8=par2
340 b?9=par3
350 n=b?5+7
360 X%=b:Y%=b DIV 256:A%=&7F
370 CALL osword
380 IF b?n=0 THEN ENDPROC
390 PRINT"Disc error &";~b?n : END
400 :
410 DEFFNconv(X) =2^(X+7)
```

OSWORD &80 (128) Misc. IEEE commands IEEEFS
This OSWORD command, implemented in the IEEEFS, provides access to the various IEEE controls. For further information, see the IEEEFS manual supplied with the Acorn IEEE488 interface.

OSWORD &82 (130) R/W params Cambridge Ring
OSWORD &83 (131) Data transmission Cambs. Ring
OSWORD &84 (132) Ring polling Cambs. Ring
OSWORD &90 (144) Service Interface Network
OSWORD &A0 (160) Isolated word recogniser
These commands are implemented by specialist hardware, and you are referred to the appropriate product manuals. Note however that it is not a good idea to use these numbers in your own programs. If you require an OSWORD call for your own use, contact Acorn.

OSWORD &FE Z80 SP Disc Read Z80 Host
This call, serviced by the Z80 Tube OS, is used by the firmware to read single density (FM) discs in the double density (MFM) CP/M system.

OSWORD &FF Z80 Data Transfer Z80 Host
This call, serviced by the Z80 Tube OS, is used by the firmware to transfer blocks of data across the tube. See also OSWORDs 5 and 6.

5 : Shadow & SRAM

Introduction

As 8-bit microprocessors, the 6502 family can only directly access 64k of memory. Yet, this is not enough for many applications and so Acorn increased the RAM available from 32k on the standard Model B, to 64k on the B+, (upgradeable to 128k) and then to 128k as standard on the Master and Compact. This extra RAM cannot all appear in the memory map of the computer at the same time, and so it is 'paged', ie, brought into the map when needed. Essentially there are two areas of memory where paging occurs — the screen area, known as shadow memory, and the ROMs area, known as sideways memory or SRAM.

Shadow Memory

The 32k of RAM on a BBC B Micro and Electron appears to the processor as a contiguous block addressed from &0000 to &7FFF. The system takes up about 3k of that for essentials (from the bottom), and a further amount for the screen (from the top), leaving a block in the middle, for programs and data.

The BBC B+ 64k of RAM is laid out in way which allows a lot of software to take advantage of it. There is 32k present as on the Model B, and a further 20k is provided for the screen, so nothing is needed from the top of the main 32k. This provides more program space. The snag is that if an 8k or even a 1k screen mode is used, the other 12 or 19k is difficult (although not impossible) to use for other tasks. The BBC B+ can be expanded to 128k but the additional 64k is Sideways RAM, not additional program memory.

The Master 128 and Compact also have Shadow RAM which is implemented in a similar way to the BBC B+. The important difference is that Master and Compact Shadow screens can be written to whilst they are not displayed, as the controls to determine which bank to read from or write to and which bank to display are separate.

Shadow memory is clearly defined by Acorn as having one purpose — to be used as screen memory so that more of the main RAM can be used as program memory. As a programmer you don't have to worry about this — the MOS takes care of it all for you: There are no high-level calls provided to do anything unusual but should you wish to use shadow memory in a different way, it is perfectly possible. Say for example you are writing a Mode 7 only program such as a teletext editor, which will always run on a Master. Then clearly shadow memory from &3000 to &7BFF will be wasted unless you make some special use of it. Listing 5.1 is an example. This program will save 19 MODE 135 screens to shadow RAM, and then restore them, without affecting the contents of main RAM.

On the BBC B+ there is an additional 12k of RAM mapped between &8000 and &AFFF. This is not used by the MOS, and can be used as a data area. See below for details of how to access this memory.

Sideways RAM

In addition to the 64k of RAM used as main memory and Shadow RAM -ANDY, HAZEL, LYNNE etc, the Master and B+128 are supplied with another 64k of 'Sideways' RAM, so called because it appears as four banks of 16k each mapped at the same locations (&8000-&BFFF), which is also the locations that BASIC, ADFS and the other 'Sideways' ROMs occupy. This RAM can be used in a number of ways, which are listed below.

The RAM in the Master 128 and Compact occupies slots 4, 5, 6 and 7. On the B+ it occupies slots 0, 1, 12 and 13 unless link S13 is moved (see below).

Many BBC Model B owners have Sideways RAM fitted to their machine usually in the form of a dedicated RAM card, but sometimes as part of a ROM extension board. The latter method tends to offer only 16k (one bank), whereas the dedicated cards offer 2, 4, 8 or even 16 banks.

There are some commands associated with Sideways RAM which are listed here along with their derivative at machine level. Full details of command syntax can be found in Chapters 2 and 4.

*SRLOAD	OSBYTE &44	OSWRSC
*SRSAVE	OSBYTE &45	
*SRREAD	OSWORD &42	
*SRWRITE	OSWORD &43	
*SRROM	OSRDRM	
*SRDATA	OSRDSC	

Using SRAM for ROMs

Many serious applications packages on the BBC Micro are written in ROM format. This is not because ROM format is any better intrinsically — indeed it imposes the serious constraint of having to write your entire program in assembler — but because the whole of user RAM is available for data. With only 32k of user RAM on the original Model B this is important, and the ROM format soon became popular for much serious software.

Nowadays, there is so much good software written in this format, that there are often insufficient sockets in the machine to hold all the ROM programs you are likely to want. With Sideways RAM this is not a problem. If a ROM program is copied onto disc, it can be re-run in Sideways RAM, and the ROM need never again be plugged in. A further advantage is that ROM-based software will often co-reside with RAM based programs, and they are not wiped out (except on the Compact when *FX200,3 is used) on reset.

Finally, they can always be programmed or 'blown' into an actual ROM or EPROM, and then the program will be in the machine permanently.

Making a Disc Image

To make a 'disc image' of a ROM, you must first plug the ROM into the computer (whilst switched off). See your User Guide for details of how to do this. Then, by typing *ROMS, you can see in which socket (0-15 or 0-F) you have placed the ROM. Listing 5.2 will save plugged-in ROMs to disc.

Copying published ROM software onto disc must only be done if the publisher permits personal backups. Some ROM software is copy-protected

to discourage piracy. It tries to write to itself which will fail in a ROM, but will succeed in Sideways RAM. The software then crashes if a successful write occurred. Another protection method is for the code to check itself to see if it was there at power-on. Acorn Sideways RAM is not write-protected or battery backed, and so some protected ROM software may not run.

To reload a saved ROM image into Sideways RAM the *SRLOAD command is used. The syntax is:

```
*SRLOAD <filename> <load addr> <bank id> (<Q>) (<I>)
```

The filename is that of the ROM image on disc. The load address is always &8000 for ROM software (the & is not used in the command). The bank address denotes which bank of Sideways RAM is to be used. On the Master and Compact the available banks are 4,5,6 and 7. On the BBC B+, the banks are 0, 1, 11 and 12. To provide compatibility across the two machines the letters W, X, Y and Z can be used instead. Get into the habit of using the letters rather than numbers. On the Master and Compact W=4, X=5, Y=6 and Z=7. On the BBC B+, W=11, X=12, Y=0, and Z=1. Link S13 on the BBC B+ may be moved so that Y=14 and Z=15, and the system ROMs which normally occupy those slots subsequently appear as ROMs 0 and 1. Note that on the BBC B+, the *SRLOAD command takes a hex parameter for the banks &A-&F, without the &.

On the Master Compact only, an additional option 'I' (for initialise) can be specified with the *SRLOAD (and *SRWRITE) commands. This loads the image in, and then updates the ROM table, which is located at &02A1. The contents of the ROM type byte (the seventh byte of the ROM you have just loaded, location &8006) is copied into &02A1+n where n is the bank number. This is necessary for the MOS to recognise that the ROM is present. One way of discovering which ROMs are present in the machine is to read the 16 bytes from &02A1 to &02B0 on the I/O processor. This feature is supplied on the Master 128 in MOS 3.21 onwards.

The 'I' suffix should not be used when loading a ROM which claims workspace in memory (see Chapter 6, and below). This is because the initialisation procedure does not cover claiming workspace. You must press CTRL-BREAK before using such ROMs.

Workspace claims and Frugalising

When a ROM claims workspace in memory (see Chapter 6) it does this by placing a byte in a table starting a &0DF0 on the I/O processor. The byte used is &0DF0+n, where n is the slot number of the ROM. The byte placed is the highest page number claimed — if claiming two pages from &1900, the byte used will be &1B. On the BBC B+ B/E, this number can never exceed &807F, as this is where RAM stops, and thus the top bit of the byte is free for another purpose. This purpose is frugalising. If the top bit of the workspace byte is set, this is a signal that the ROM should not initialise at power-up. It should regard itself as *UNPLUGged. On the Master and Compact, workspace can be claimed in HAZEL (&C000-&DFFF), so this system is unsuitable, and instead a separate *UNPLUG system is contained in the MOS.

Using SRAM for Data Storage

The Sideways RAM in the BBC B+ and Master Series can be used in two modes, known as pseudo and absolute addressing. In absolute mode each slot is identified by its normal memory map address, and slot or ROM number ie to save a Mode 0 screen image into Sideways RAM slot Z the command would be:

```
*SRWRITE FFFF3000+5000 8100 Z
```

Absolute mode is necessary when you need to specify which socket is needed, or it is convenient to refer to memory by the map address. Pseudo mode has the advantage of offering your program one contiguous block starting from &0000, and ROM slot names (to &FFBF - 64 bytes are used for identification), and ROM slots can be ignored.

Listing 5.3 is a program which uses the pseudo-addressing system to save three Mode 0 screens to sideways RAM, and then recall them very quickly.

Writing your own ROMs

Sideways RAM offers the perfect environment to develop Model B software. Programs can be assembled and tested in Sideways RAM before being placed permanently in ROM or EPROM. This avoids having to program many EPROMS (a slow process) before getting it right. Full details of the protocols required for ROM software are given in Chapter 6.

Sideways RAM on the Model B is often fitted in such a way that any write access to addresses &8000-&BFFF will go to the Sideways RAM.

This means that you can start your assembly code with:

```
P%=&8000
```

and the code will be assembled straight into Sideways RAM. On the B+/M/C things are not so straightforward. To move code into Sideways RAM, the *SRWRITE command is used, and the object code must first be assembled in ordinary RAM. However the code must be assembled so that it will run in Sideways RAM, ie the start address must be &8000. BASIC 2 allows you to specify a start address for the code, which is not the same as the place the object code is stored. P% is set to the normal start address, and O% is set to the address where code is to be stored. Listing 5.4 is an example of this technique. It assembles a simple ROM program to alter the start-up message on BREAK and CTRL-BREAK

Low Level Paging Control

Paging of RAM banks is done by certain switches which appear on the memory map, and can be written to. These switches are located at two addresses. The first, known as ROMSEL, standing for ROM SELect, is located at &FE30 (I/O processor) on all machines. This controls the Sideways RAM banks only. The second, ACCCON, standing for ACCess CONtrol, is located at &FE34. This is present on the Master and Compact, and in a limited form, on the BBC B+.

Model B and Electron

On the Model B and Electron there is no shadow RAM. ROMSEL is only four bits wide, and therefore only values of &00 to &0F are valid — each selects the page of ROM/RAM which matches the number given. This appears at &8000-&BFFF. The rest of the memory is single-page only. Note that there is a 768 byte section (filled with ROM) from &FC00 to &FEFF which is switched out to I/O ports, and cannot be switched in.

If any third party Sideways RAM is fitted to a Model B or Electron, it is usually accessed by putting the bank number into ROMSEL. For writing to memory however, the practice varies. Some Sideways RAM systems, including most of the 16k only ones, are wired so that

any write operations to addresses between &8000 and &BFFF are directed to the sideways RAM. Multiple bank systems tend to use a control port mapped in a similar manner to ROMSEL, but often at a different address.

ACCCON is not present on the BBC B or Electron, and any accesses to &FE34 will have unpredictable effects. Note that the permanent claiming of write lines to addresses between &8000 and &BFFF can cause problems if you then try to fit a second Sideways RAM module, or more subtly, you try to develop code in one of the new bank-switching EPROMs such as the 27513 and 27011 devices.

Model B+

In addition to Model B and Electron type ROM selection, there are two controls which toggle the 20k plane of Shadow memory (&3000-&7FFF) and the 12k of Sideways memory.

The top bit (bit 7) of ROMSEL controls the paging of the 12k of Sideways RAM. If set, then accesses to addresses &8000 to &AFFF are directed to the 12k special Sideways RAM. If clear, then the ROM bank as determined by the lower four bits of ROMSEL is accessed. Some software does not intelligently mask out the top four bits of ROMSEL, (including MOS 2.00) and thus attempts to access ROMs with numbers between 128 and 135, will access this special Sideways RAM. For example an 8k language ROM can be loaded into this RAM, and started up with *FX142,128. Doing this, however, will not re-select the language on BREAK.

The 20k of shadow RAM is switched by the top bit of ACCCON (&FE34). When the bit is set, the shadow bank is selected. When clear the main bank is selected. By toggling this bit directly, it is possible to flick between the two planes for animation purposes. Note however that you can only write to the bank which is currently displayed, as the single toggle controls both display and read/write access.

Like the Model B/Electron, the BBC B+ has 768 bytes from &FC00 and &FEFF permanently banked out to I/O.

Master 128 and Compact

This offers a greatly expanded set of controls. Here is a summary:

ACCCON—&FE34 Read and Write

b7	IRR	1=Causes an IRQ to the processor
b6	TST	1=Selects &FC00-&FEFF read from ROM
b5	IFJ	1=Internal 1MHz bus 0=External 1MHz bus
b4	ITU	1=Internal Tube 0=External Tube
b3	Y	1=Read/write HAZEL &C000-&DFFF 0=Read/write ROM &C000-&DFFF
b2	X	1=Read/write LYNNE &3000-&7FFF 0=Read/write main memory &3000-&7FFF
b1	E	1=Causes shadow if VDU code 0=Main all the time
b0	D	1=Display LYNNE as screen 0=Display main RAM screen

ROMSEL—&FE30 Write Only (Copy in &F4)

b7	RAM	1=Page in ANDY &8000-&8FFF 0=Page in ROM &8000-&8FFF
b6		Not used
b5		Not used
b4		Not used
b3	PM3	Select ROM bank
b2	PM2	Select ROM bank
b1	PM1	Select ROM bank
b0	PM0	Select ROM bank

Names Explained

ACCCON is a read/write register, and it is permissable to write code that reads it directly, although you should use OSBYTE &96 and &97 if want your code to work on the 6502 Second Processor.

HAZEL is the name given by Acorn to the 8k of RAM used by the MOS, filing systems, and other ROMs, at addresses &C000—&DFFF.

ANDY is the name of the 4k of RAM used by the MOS at &8000—&8FFF.

LYNNE is the name of the 20k shadow screen at &3000—&7FFF.

Your code should always store the current ACCCON setting, alter it to your own requirements, and then reset it to the original value before you exit. Any setting of bits 6 and 7 should be done while interrupts are disabled. At bit level, ACCON is arranged as follows:

b7: This causes an IRQ to occur. If you set this bit, you must write a routine on IRQ2V to clear it.

b6: If set, this switches in the ROM at &FD00—&FEFF for reading, although writes to these addresses are still directed to I/O. The MOS will not work properly with this bit set. MOS 3.21/5.10 uses this feature to place some of the reset code in this area, which is run at power up. Previously it contained an ASCII text listing the people who developed the computer.

b5: If set, the internal 1MHz bus on the Master 128 is selected, and accesses to FRED and JIM appear on the cartridge port, which is where the internal 1MHz bus is wired. If clear, the normal external bus is used, IDC socketed under the keyboard.

b4: If set, the internal Tube connector is used. If clear, the external connector under the keyboard is used. This control could be used to run two Tube processors.

b3: If set, accesses to &C000—&DFFF are directed to HAZEL, an 8k bank of RAM. If clear, then operating system ROM at these addresses is used. The MOS uses both, and contains all the necessary switching code to ensure that the right bank is selected at the right time.

b2: If set, read/write accesses to &3000—&7FFF are directed to LYNNE, the shadow screen RAM. If clear such accesses are to the main RAM at these addresses. This control does not affect the screen display, which is determined by the setting of bit 0.

b1: When the shadow screen is active, the writing of characters and graphics are to one bank of RAM, but ordinary memory manipulation is in the other bank. To avoid constant switching by applications software, the memory controller has a special feature, activated by setting this bit. The feature is that if the processor program counter is between &C000 and &DFFF, read/write access is directed to the LYNNE shadow RAM. If the program counter is anywhere else main

RAM is accessed. The MOS VDU driver code is situated in this area. The system works by direct hardware links between the 65C12 address bus, and the memory controller. If this bit is clear, then all accesses are to normal RAM.

b0: If set, LYNNE memory is displayed as the current screen. If clear, then main RAM between &3000 (or higher) and &7FFF is used. Altering this bit directly will flick instantly between the two screens. The easiest way to do this is to EOR ACCCON with &01. Remember that unlike the BBC B+, the hidden screen can be updated by correct use of these two bits, which are controlled by MOS calls OSBYTE &70/&71. ACCCON on the B+ is a write only register, so its state must be read from the RAM copy, bit 4 of the VDU status byte.

ROMSEL on the Master Series offers similar control to the earlier machines. The bottom four bits control which of 16 banks of ROM/Sideways RAM appear in the memory map between &8000 and &BFFF. Bit 7 controls ANDY, a 4k bank of RAM located between &8000 and &8FFF, and used by the MOS as graphics workspace, and the function key string buffer. When set, the RAM in ANDY appears between &8000 and &8FFF. When clear, this area is occupied by the ROM bank selected by the bottom nibble of ROMSEL.

High Order OSWORD &05 and &06

There are some misconceptions about OSWORD &05 and &06 when used to access paged memory using an address in which the high bytes are &FFFE ie XY+2=&FE, XY+3=&FF. On the Model B+, this call will access the 12k special Sideways RAM, (addresses &FFFE8000 to &FFFEAFFF) and the 20k shadow screen (addresses &FFFE3000 to &FFFE7FFF).

The feature is however implemented in DFS, not the MOS, and did not appear on all versions of the BBC B+ 1770 DFS. It should therefore be avoided on applications intended for commercial release. On the Master 128 and Compact, the feature does not exist, despite documentation to the contrary. However in ADFS only, it is possible to *LOAD and *SAVE data from shadow RAM to disc. This is done by specifying addresses using the &FFFE convention. The shadow screen must be currently selected for display (ACCCON b0) although not necessarily for writing, and the load address must have &FFFE as its higher order address (either in the file attributes or explicitly in the OSFILE command).

Listings

Listing 5.1

```
 10 REM Save ROM slot to disc
 20 REM (c) Dave Atherton
 30 REM for B/B+/E/M/C
 40 REM MOS:A Dabhand Guide
 50 :
 60 MODE 135
 70 HIMEM=&3BFF
 80 INPUT "ROM number (0-15) : " R%
 90 INPUT "Filename for ROM  : " F$
100 FOR I%=0 TO &4000
110 I%?&3C00=FNpeek (I%+&8000, R%)
120 NEXT
130 OSCLI "SAVE "+F$+" 3C00+4000 0 FFFF8000"
140 END
150 :
160 DEF FNpeek (!&F6,Y%)=(USR &FFB9) AND &FF
```

Listing 5.1. Save ROM slot to Disc

Listing 5.2

```
 10 REM Sideways RAM addressing
 20 REM (c) Dave Atherton 1987
 30 REM for B+128/M/C
 40 REM MOS:A Dabhand Guide
 50 :
 60 MODE 0
 70 FOR R%=ASC"W" TO ASC"Z"
 80 OSCLI "SRDATA "+CHR$ R%
 90 NEXT
100 :
110 REM Some complex patterns
120 :
130 DIM T$(2):T$(0)="SIN":T$(1)="COS":T$(2)="COS"
140 VDU29,640;512;
150 PROCpic(300,300,20,24,2,&0000)
160 PROCpic(50,300,60,6,0,&5000)
170 PROCpic(600,10,100,8,2,&A000)
180 PRINTTAB(10,5)"All pictures now saved in sideways RAM"
190 PRINTTAB(16,7)"Press SPACE to recall them."
200 FOR I%=0 TO &A000 STEP &5000
210 REPEAT UNTIL GET=32
220 OSCLI "SRREAD FFFF3000 FFFF8000 "+STR$~I%
```

```
230 NEXT
240 GOTO 200
250 END
260 :
270 DEFPROCpic(A%,B%,C%,petals%,F%,addr%)
280 FOR B%=60 TO 800 STEP C%
290 A%=B%/6
300 MOVE0,A%+B%
310 FOR i=0 TO 2*PI STEP PI/100
320 IF F%<2 F%=1-F%
330 radius%=EVAL(T$(F%)+" "+STR$(i*petals%))*A%+B%
340 DRAW SINi*radius%,COSi*radius%
350 NEXT
360 NEXT
370 OSCLI "SRWRITE FFFF3000+5000 "+STR$~addr%
380 CLS
390 ENDPROC
```

Listing 5.2. Sideways RAM Addressing.

Listing 5.3

```
 10 REM ROM Assembly on B+/M/C
 20 REM (c) Dave Atherton 1987
 30 REM for B+/M/C
 40 REM MOS:Dabhand Guide
 50 :
 60 MODE 7
 70 osnewl=&FFE7
 80 osasci=&FFE3
 90 FOR pass=4 TO 7 STEP 3
100 P%=&8000 : O%=&5000
110 [OPT pass
120 EQUB 0
130 EQUW 0
140 JMP service
150 EQUB &82
160 EQUB copyr AND &FF
170 EQUB &BB
180 .title
190 EQUS "Dave's Master 128"
200 EQUB 0
210 .copyr
220 EQUB 0
230 EQUS "(C) Dabs Press"
240 EQUB 0
250 .service
260 PHP
```

```
270 PHA
280 PHX
290 PHY
300 CMP #&27
310 BEQ message
320 CMP#3
330 BNE out
340 .message
350 LDA #11
360 JSR osasci
370 JSR osasci
380 LDX #0
390 .loop
400 LDA title,X
410 BEQ cr
420 JSR osasci
430 INX
440 BNE loop
450 .cr
460 JSR osnewl
470 JSR osnewl
480 .out
490 PLY
500 PLX
510 PLA
520 PLP
530 RTS
540 ]
550 NEXT
560 *SRWRITE 5000+200 8000 Z
570 PRINT"Now press CTRL-BREAK"
```

Listing 5.3. ROM Assembly on B+/M/C.

Listing 5.4

```
 10 REM Store data in Shadow RAM
 20 REM (c) Dave Atherton 1987
 30 REM for B+/M/C
 40 REM MOS:Dabhand Guide
 50 :
 60 MODE 135
 70 VDU23,1;0;0;0;
 80 PROCassemble
 90 REM Create screens and store them
100 FOR I%=0 TO 18
```

```
110 CLS
120 FOR J%=0 TO 9
130 I$=CHR$(128+RND(9))+CHR$141+"Screen "+STR$I%
140 PRINTTAB(J%*3,J%*2)I$
150 PRINTTAB(J%*3,J%*2+1)I$
160 NEXT
170 !adr=&3000+I%*&400:?direct=0:CALL M%
180 NEXT
190 CLS
200 PRINTTAB(2,4)"18 screens saved in shadow RAM"
210 PRINTTAB(4,6)"Press a key to recall them"
220 PRINTTAB(0,20)"NB : Main RAM &3000-&7FFF unaffected"
230 *FX21
240 IFGET
250 FOR I%=0 TO 18
260 !adr=&3000+I%*&400:?direct=1:CALL M%
270 NEXT
280 VDU23,1,1;0;0;0,10,10
290 END
300 DEFPROCassemble
310 DIM M% &100
320 scr=&70:adr=&72:pages=&74:direct=&75
330 FOR pass=0 TO 2 STEP 2
340 P%=M%
350 [OPT pass
360 LDA #4
370 STA pages \ &400 bytes
380 LDA #0
390 STA scr
400 STA adr   \ read from &7C00
410 LDA #&7C
420 STA scr+1
430 LDA &FE34
440 PHA
450 ORA #4
460 STA &FE34 \ select LYNNE
470 .loop
480 LDA direct \ 0=Save 1=Load
490 BEQ jsave
500 JSR load
510 BRA inc
520 .jsave
530 JSR save
540 .inc
550 INC scr
560 INC adr
570 BNE loop
580 INC scr+1 \ 4 pages
590 INC adr+1
600 DEC pages
610 BNE loop
620 PLA
630 STA &FE34
640 RTS
650 .load
660 LDA (adr)
670 STA (scr)
680 RTS
```

```
690 .save
700 LDA (scr)
710 STA (adr)
720 RTS
730 ]NEXT
740 ENDPROC
```

Listing 5.4. Store data in Shadow RAM.

6 : Sideways ROMs

Sideways ROM Format

As outlined briefly in Chapter 5, one popular format for programs on the BBC and Master series micros is the ROM format. Programs written in this format have the following features:

They can use the whole of program memory (&E00 to &7FFF), especially attractive for programs which use data areas, such as wordprocessors and databases.

- They can be 'blown' into an EPROM or ROM, and fitted to the micro as a permanent fixture. For this reason software such as I/O drivers, filing systems etc., are best written in ROM format.

- They can co-reside with programs written in BASIC, machine code or any other language. This makes the format attractive for utility type software written to aid development, editing and debugging, ie, BASIC toolbox and machine code monitors.

- They must be written in assembly language—a drawback for novice programmers —and unless complex ROM switching is employed they are limited to a length of 16k (BBC B/B+/E) or 32k (Master).

- They adhere to a standard format employing a ROM header - which allows the MOS to know of its existance and for it to process the various service calls issued to it by the MOS.

- The standard supports automatic Tube transfer, and a number of other operating system housekeeping facilities, such as operating system calls, calls to read data and execute code in other Sideways ROMs.

All the BBC microcomputer range contain operating system commands to support software fitted in 'Sideways' or 'paged' ROM. This means that many of the mundane operations such as switching in and out the right ROM and correctly recognising ROMs at power-up are invisible to the user.

Writing a ROM is not unlike writing any other assembly language program, always remembering that (unless the program is purely for Sideways RAM use) data and other information stores must be in a RAM area of memory, as by definition, a ROM is a read only memory.

ROM Header

When writing ROM software - that is software intended to run from one of the paged ROM/RAM banks at addresses &8000 to &BFFF – the first few bytes must be assembled to a standard format to form the ROM header. The format is as follows and is explained below:

```
.romentrypoint
JMP lang
JMP serv
EQUB romtype
EQUB copyr-&8000
EQUB verno
EQUS titles$
.copyr EQUB &00
EQUS "(C)"
EQUS copyright$
EQUB &00
EQUS vernum$
EQUB &00
EQUD tubeaddr
```

JMP lang

A jump to the language entry point, if any. If the ROM is a service ROM, then this should be replaced by three zeros, ie,

```
EQUW &00
EQUB &00
```

Languages may be designed solely for use on a certain co-processor, in which case, this jump should be coded in the machine code of the target processor. This may cause difficulties with processors that cannot code an absolute jump in three bytes. The MOS will copy the code across the Tube, to the relocation address—more details on this can be found below.

JMP serv

This is assembled at the service entry point and performs a JMP direct to the code written to handle the service calls. With the exception of BASIC, all ROMs must have a service entry point.

EQUB romtype
See below for the make-up of this byte.

EQUB copyr-&8000
The low byte of the address of the label ‘copyr’

EQUB verno
Internal version number. This has no effect on the code. It is displayed (in hex) as part of the Master *ROMS command. It is never used or displayed on Models B/B+/E.

EQUS title$
The title of your ROM. There is no need to terminate this with a carriage return character. Use of more than one word is permitted, but it is not a good idea to use control characters in the title, as some utility programs will then not work properly—it also looks ugly if the micro is run in any mode other than Mode 7. The title string will be displayed by the *ROMS command, and if a language, on selection.

.copyr
EQUB &00. The title string is terminated by a zero. This location is pointed to by the byte at &8007.

EQUS "(C)"
Must be ASCII codes &28, &43, and &29 in order.

EQUS copyright$
Usually the year and authors name, ie, “1987 David Atherton”, but again, this has no effect on the working of the code. Some utility programs display this message as part of extended versions of *ROMS

EQUB &00
The copyright string termination byte - must be zero.

EQUS vernum$
Doesn’t need to be the same as ‘verno’. This might be “1.20”, or “Two”. The convention to use is “X.YY” where a new number in X represents

major changes, and changes in YY are minor adjustments. This entry is optional, but mutually exclusive with the Tube address.

EQUB &00

A terminator which must be present if you use the version number.

EQUD tubeaddr

A 32-bit address for Tube relocation. The MOS will copy any language across the Tube to this address. This entry is optional, but without it, languages may not run properly on co-processors. If a tube address is required, you must not use the version string. When a language starts across the Tube, the address following the first zero byte after the (C) string is taken as the Tube execution address.

Entering a Language ROM

There are three ways of executing the code in a Sideways ROM—as a language, by service calls, or by use of the extended vectors.

The language entry of a Sideways ROM is at &8000. There are only three bytes available here, so the language should immediately perform a JMP to code elsewhere in the ROM. It is possible to write ROM code for non-6502 processors, to be transferred across the Tube to another co-processor in which case, all the code, including the initial JMP should be written in the code of the target processor.

ROM Service Calls

First some terms of reference. Within this description, the term 'you' means the ROM code that you are writing. When the text says that you must do things, it means that if you are writing a ROM program, these are things you will have to write code to deal with.

ROMs must respond to a number of 'service' calls, all of which involve the MOS selecting your ROM, and performing a JSR to &8003, ie the service call entry point. Upon entry into your ROM, the accumulator contains the service call number, X the socket number of your ROM and Y may contain a parameter.

As a general rule, if you 'claim' a service call, then you exit the call with the accumulator set to 0, because you have recognised the call as one appropriate to your program and have performed some function.

You wouldn't want another ROM to pick it up and also deal with it, which is what will happen if the call isn't claimed. On the other hand some functions like *HELP are clearly intended for every ROM, and it is wrong to claim such calls. In this case you should save all registers, process the call, and then restore all registers before RTSing. If you don't want to service a particular code at all, you should exit with all registers and flags preserved.

Here is some sample code, which might be the core of your service routine. Each call that you will service is dealt with by a conditional branch. Only known calls are serviced, others are ignored, and passed on. Exit points for claimed calls, and unclaimed calls are also supplied. The main code in the ROM may JMP to these to exit.

```
.serv
PHA \ stack everything
TXA
PHA
TYA
PHA
TSX
LDA &103,X  \ Get accumulator
CMP #&04
BEQ starcmd \ * commands
CMP #&09
BEQ help    \ *HELP
CMP #&27
BEQ reset   \ Master BREAK
CMP #&03
BEQ reset   \ Model B BREAK
.exit       \ General no-claim exit point
PLA
TAY         \ By definition any call not
PLA         \ dealt with in this List
TAX         \ is passed through, with
PLA         \ all registers preserved.
RTS
.claimed    \ General 'claimed' exit point
LDA #0      \ Setup accumulator
TSX         \ for de-stacking
STA &103,X
JMP exit    \ and exit with A=0
```

The call issued by the MOS (or yourself as a service ROM, for instance service call &0A, or even from a program in main memory) is OSBYTE &8F, with X set to the service call number, and Y carrying any

parameters. By the time it reaches you (at &8003), the service number is in A; X contains your ROM number (which is also in &F4), and Y contains any parameter, as passed by the original OSBYTE &8F.

Service Calls

These are the service calls that could be issued. Not all are issued on all machines, and this is noted.

&00 (0) Nothing required All

Another ROM has claimed this call, and you should ignore it—ie, you should RTS with all registers intact. Debugging code may wish to print some message to say that this call has been received, but generally it should be ignored and certainly final release programs should never do so. This call cannot be claimed. Note that ROM scanning stops once this call is claimed, so any attempt to do so will deny call access to lower priority ROMs.

&01 (1) Claim Static Workspace in main RAM All

Static workspace often also called absolute workspace, should only be claimed by filing systems. It always starts at &E00 on the I/O processor (on all machines) and is shared by the filing systems. Only one filing system can use the workspace at once.

Entry: Y=current upper limit (hi-byte of address) of workspace.
Exit : Y set to upper limit of static workspace required by you, or entry value, whichever is the greater.

This call is passed to all ROMs, and when done, OSHWM is set accordingly. Note that ROMs cannot immediately use this static allocation—they must claim it for use by issuing service call &0A (which is different to responding to service call &0A—see below). This call must not be claimed. Only filing systems should respond to it, See also calls &21 and &0A.

&02 (2) Claim Dynamic Workspace in RAM All

Dynamic workspace, also called Private workspace (but not to be confused with Private RAM on the B+ and Master series micros) may be claimed by any ROM, language or service, filing system or not. This RAM is exclusive to you, and can be relied on to hold data until hard

reset, and in practice, until power down. Workspace is claimed in units of 256 bytes, ie, a page at a time.

Entry: Y set to hi-address of next available page of memory.
Exit : Y increased by you by the number of pages you require of private workspace. Your code will need to note the Y value to determine where to place variables etc.

A byte is reserved for you at &DF0+ROM number. It is practice to store the start page of your private workspace in the first six bits of this byte, the top two bits being used for 'frugalising', ie, if they are set, the ROM does not initialise, see Chapter 5 for more details. On the Model B+, several ROMs use this frugalising system, particularly Acorn filing system ROMs. However, on the Master, top-bit frugalising cannot be used, due to the fact that the top bit is used for workspace identification. However you can use values between &40 and &BF to signify that your ROM is frugalised on a Master, as these pages are never used as workspace. Whether any utility programs recognise this is another matter! This call must not be claimed. See also service call &22.

&03 (3) Offer auto-boot B/B+/E

This call is provided for filing systems during initialisation. If you are a filing system you should respond as follows:

1. Check if any key is pressed on the keyboard (use OSBYTE &81 with Y=&FF).
2. If not, move to step 5.
3. Is it the unique key appropriate to this filing system?
4. If not, exit preserved registers ie, no claim.
5. Insert all the vectors appropriate (FILEV, ARGSV, FINDV ...).
6. If you wish, print your name, and do any other startup things.
7. Was Y=0 on entry? If not, move to step 9.
8. Select your root directory, then execute a file called !BOOT in the CSD according to your own conventions, ie, read any option settings and *LOAD/*RUN/*EXEC accordingly.
9. Zero the accumulator then exit.

On the Master at BREAK, this call is issued to the previous filing system ROM only. At CTRL-BREAK, the call is only issued to the FS ROM determined by *CONFIGURE, if it is a valid service ROM, and no keys are being pressed. Otherwise the call is issued to all ROMs.

&04 (4) **Offer unrecognised 'star' command** **All**

When a star command is issued, it is processed as follows:

1. The language passes the string to OSCLI.
2. Leading '*'s and spaces are stripped.
3. The command is checked to see if it is a MOS command. If so, the MOS processes it directly, and returns to the language.
4. If not, locations &F2 and &F3 are set to the start of the complete string. If there are padding spaces and stars, Y is increased so, LDA (&F2),Y will load the first alphanumeric character, and the MOS issues service call &04. The text will finish with &0D. You should check this. If the command is a language or filing system selection call, you should re-issue the appropriate commands—for example a language selection should not be done by using JMP &8000 but by using:

```
LDA #&8E   \ Language selection call
LDX &F4    \ This ROM
JMP osbyte \ and call it
```

5. If you cannot respond to the command, and neither can other ROMs, it is passed to the current filing system, in the form of *RUN command. Filing system ROMs thus get a second chance to take the command, by executing a routine rather than looking on backing store for a file. This is how DFS executes *ACCESS, *BACKUP, etc. Picking up commands this way means that they won't work if you are not the current filing system, because such commands tend to alter static workspace memory in a way that would be meaningless and potentially disastrous if another filing system were active.
6. On the Master Series, if the current filing system cannot *RUN the file, the library filing system is offered the command.
7. Finally if no-one recognises the command, the MOS executes a BRK with the 'Bad Command' error.

ROMs often respond to a number of '*' commands, and the usual method of dealing with them involves reading a table. The Acorn convention is that a ROM should respond without case sensitivity, and that the command may be abbreviated if terminated by a full stop (ASCII &2E). You may, perhaps, insist that a minimum number of letters is used to avoid clashing with, for instance, DFS commands.

The convention for parameters after a command is that they can be terminated and separated by a space (ASCII &20) or a comma (ASCII &2C), and that additional spaces are disregarded.

The MOS gives you no help in writing a command table and decoder. You must do this yourself. If all the text of all your commands can be fitted into 256 bytes, the best method is to index the table by X, and use Y to check each letter of the command. After testing the first item in your table of commands, you would reset Y, but not X, to look at the next item. A common practice of ROM writers is to put the routine address in hi-lo order after the command string. This is because all routine addresses must have a high byte of &80 or greater and the text of the command will never have values exceeding &7E.

&05 (5) Offer an Unrecognised Interrupt All

When an interrupt is received, the MOS first vectors through IRQ1V, and tests the devices it knows about (the keyboard, the RS423 port, etc.) to see which caused the interrupt. If it cannot find any such device, and the interrupt is not masked (see OSBYTEs &E7-&E9), it is then signalled to Sideways ROMs as call &05. You must check that your device has indeed caused an interrupt, and if so, perform your processing and claim the call. If no ROM claims the call, it will be passed to IRQ2V. You are servicing a MOS call here, not an direct IRQ, so you should terminate with RTS, and not RTI. This routine must be serviced very quickly, so if you don't use the call, exit straight away.

&06 (6) Offer an Unrecognised BRK All

When this call is issued a BRK has occurred. You are being offered a chance to deal with it, before the MOS error handler prints out the appropriate error message. After you have dealt with this call, then as long as you don't claim it, it is then indirected through BRKV, usually to the error handler of the current language.

Upon entry &FD/&FE point to the error number, which is followed by the error text—this may be a ROM address, in which case OSBYTE &BA will tell you which ROM contains the error text. &F0 contains the stack pointer immediately after the BRK occurred.

When writing ROMs which themselves generate errors, it is a good idea, upon an error occurring, to copy the BRK, the error number and the error message into main RAM, and execute it there, to facilitate ROM switching as you return after the error to the current language.

The conventional place to use for this purpose is the bottom of the stack starting at &100. This call may be claimed. If it is not claimed, Y must be preserved.

&07 (7) Offer an Unrecognised OSBYTE call All

A call has been made to OSBYTE (&FFF4), or a *FX command has been issued which is unknown to the MOS. If you supply additional OSBYTE calls, you should respond to this by checking location &EF to see if the OSBYTE number is one that you respond to. If so, &F0 and &F1 will contain the X and Y values passed by the user.

If you wish to respond to OSBYTE calls, you should obtain a number from Acorn, and respond only to this number. Acorn are currently only offering sections of OSBYTE calls, usually A=163 X=a given number, and you may respond to any value of Y.

If you do respond, then claim the call. Otherwise preserve all registers and exit.

&08 (8) Offer an Unrecognised OSWORD call All

The user has issued an OSWORD call in the range &10 to &DF, which is unknown to the MOS or any other Sideways ROM. If you supply additional OSWORD calls, you should respond by checking &EF to see if the OSWORD number is one that you respond to. The X and Y values can then be found from &F0 and &F1 respectively.

This call is also issued if an OSWORD &07 (sound) command is issued with a channel number between &1FFF and &EFFF. Note that you can not respond through this call to OSWORD calls &E0-&FF, as these are passed to USERV instead.

&09 (9) Offer a *HELP Command All

A *HELP command has been issued. Each ROM responds with text specific to itself. The conventional text to display is the ROM name, version number, and then, on a separate line, indented by two spaces, any keywords which generate extra help text. Teletext and general control characters should be avoided to ensure compatibility and an acceptable display in all screen modes. An example is:

```
Advanced DFS 2.10
  ADFS
```

To respond to *HELP <text>, the <text> string is found at (&F2),Y, terminated by a carriage return (&0D). The ROM should check this text, possibly using a table structure as described in call &04 above, and if a keyword is matched, type out the relevant text.

&0A (10) Claim Static Workspace All

This call is only applicable to filing systems. You should issue the call to indicate that you require the static workspace, which means that you are about to select yourself. After making this call, you may safely proceed to put your data in the private workspace, and set a flag to indicate that the workspace is now yours.

If you receive this call, it means that another filing system is claiming the space, and thus you've been deselected. You should copy any vital data into your private workspace area, and flag the fact that the workspace is no longer yours. This is particularly important on the Master as you may only be deselected on a temporary basis.

&0B (11) Release ability to cause NMIs All

If you are a filing system which uses NMIs, you should issue this call when you no longer require the NMI. You should set Y to the ROM number of the previous owner—see call &0C.

If you receive this call, you should compare Y with your own ROM number (CPY &F4) and if you find that this test matches, you may use the NMI space again.

&0C (12) Request/Clear NMI ability All

This call should only be issued by filing systems. When you require the NMI, you should issue this call with Y=&FF. When the call returns, the previous owner's ROM number is in Y, and you should store this safely, to use when you receive call &0B. Otherwise, when this call returns you may use the NMI safely.

If you receive this call, and you have control of the NMI, you should set Y to your ROM number (LDY &F4 will do this), store any valuable data kept in the NMI area (&D00-&D9F BBC, &D00-&D5F Master), and claim the call (ie, return with A=0). If you don't use the NMI or if you are not a filing system, you must return this call with registers intact. The way to do this is to simply not respond to the call.

&0D (13) Return Data Pointer for RFS All

This call is made by the MOS to find out which ROMs contain ROM Filing System (RFS) files. If you don't contain RFS files, ignore this call. Also, if the register 15-Y is less than your ROM number, ignore the call. If 15-Y>=X, you should put 15 minus your ROM number in &F5, a pointer to the start of your RFS data in &F6 and &F7, and claim the call. The following code will do the job for you.

```
.service13
TYA
EOR &0F    \ Invert Y value
CMP &F4    \ Compare with ROM no
BCS ignore \ A < ROM no
LDA &F4    \ Get. our ROM no
EOR &0F    \ Invert it
STA &F5    \ Store it
TAY        \ in Y as well
LDA #data MOD 256 \ Set up data pointer
STA &F6
IDA #data DIV 256
STA &F7
JMP exit   \ Claimed exit (A=0)
.ignore
JMP out    \ Unclaimed exit
```

&0E (14) Read Byte for RFS All

This call is made by the MOS to retrieve a byte from the ROM filing system. You should check that &F5 contains your inverted ROM number, and only if so, return the byte pointed to by (&F5),Y in the Y register. If the ROM number doesn't match, or if you don't have any RFS data, you should ignore this call. The following code will do the job for you:

```
.service14
LDA &F4     \ Get our ROM number
EOR &0F     \ Invert it
CMP &F5     \ Compare with &F5
BNE ignore  \ If different, exit
LDA (&F5),Y \ Get the RFS byte
TAY         \ Put it in Y
LDA #0      \ Claim the call
JMP exit    \ Claimed exit (preserve Y)
.ignore
JMP out     \ Unclaimed exit
```

&0F (15) Warning - Filing System Change All

If you are a filing system, you should issue this call during initialisation, after you have changed the vectors. This is to inform other filing systems that a change has occured.

&10 (16) *SPOOL/*EXEC Files Closure Warning All

This call is only applicable to filing systems. If you are a filing system, then before you close files open with *SPOOL and *EXEC, and before you do a general *CLOSE, you should issue this call.

If you receive this call, and you are the currently selected filing system, it means that another filing system is activating, and you should close any *SPOOL or *EXEC files that you have open. If you don't want to close them, for example when the takeover is by a temporary filing system, then you should claim this call.

&11 (17) Font Implosion/Explosion Warning B/B+/E

This call will never be made by the Master or Compact MOS. Operating Systems prior to 3.00 issues the call when processing OSBYTE &14. It is a warning to ROMs to move any data out of the way before OSHWM changes.

Entry : Y=new page value of OSHWM
Exit : Do not alter, but if you have any data stored between the previous OSHWM and the address Y*&100, then move it before character set data destroys it.

&12 (18) Restart a Filing System All

This call starts a filing system. It may be issued at any point. If you are a filing system, and you receive your start-up '*' command via call &04, you should issue this call with Y=your filing system number. Some previous documents erroneously indicated that Y contained the ROM number of the filing system. You can start filing systems from user programs with *FX143,18,n where n is the filing system number. This ought to generate a full startup, but it doesn't on some filing systems, eg, DFS 0.90.

&15 (21) Polling Interrupt for ROMs E/M/C

This call is made 100 times a second if the ROM polling semaphore is non-zero. For an explanation of the ROM semaphore system, see the

entries for OSBYTE &16 and &17 in Chapter 4. The Clock ROM program at the end of this chapter is an example of the use of this call. There are no entry conditions, and what you do during this call is up to you. It is a good idea however to either make your routine reentrant or exit within 10ms!

&18 (24) Interactive *HELP M/C

This call was designed originally to run with an interactive text help system. This has not appeared as a stand-alone product, but does appear on ANFS, where typing *HELP ON <text> will *TYPE a file called <text> in directory $.Library.!Help. The call is made by the Master and Compact MOS after service call 9, and ANFS is the only firmware to date to recognise it. The call should not be claimed. If you wish to implement a help system, then respond to this call by looking for text at (&F2),Y in the same way as service call &04 . If it matches any of your keywords, then output the appropriate help text. Acorn merely refer to this call as 'reserved'.

&21 (33) Claim Static Workspace in HAZEL M/C

This call is similar in nature and operation to call &01, except that the workspace allocations start not from &E00, but &C000 in HAZEL. The call is only issued by the Master Series MOS, where it is issued before call &01. Note that the last available page in HAZEL is &DB00-&DBFF, therefore you should not increase Y to a higher figure than &DB. If you need more space, note what you took from call &21, and claim the rest in call &01, which is guaranteed to come afterwards. Like &01, call &0A is issued to actually claim the space.

The MOS starts issuing this call with Y minus the number of pages claimed by service call &24. Therefore, if you require more space than this offers, you should claim it when call &01 is issued. The call must not be claimed. Only filing systems should respond to it. See also call &01 and &0A

Entry : Y=Highest static page so far claimed in HAZEL, plus 1.
Exit : Y=As entry or higher if you need more.

&22 (34) Claim Dynamic Workspace in HAZEL M/C

This call is similar in nature and operation to call &02, except that the workspace allocations are like call &21, in HAZEL. Again the call only occurs on the Master Series. See call &21 concerning the last

page of HAZEL. If a call &22 claim will increase Y above &DB, the balance of workspace should be claimed when call &02 is issued. You must not claim more pages than you said you would when you replied to call &24. The call must not be claimed. See also call &02.
Entry : Y=Highest dynamic page so far claimed, plus 1.
Exit : Y=Entry plus the number of pages you require.

Note that ADFS and DFS cannot work with workspace between HAZEL and main RAM. This will not occur with the standard ROM versions but could occur if the filing systems were run in lower banks.

&23 (35) This is Top of Static Workspace M/C
This call tells you the value of the top page of static workspace in HAZEL, so that you can use more static workspace than you first asked for, for example as buffers.
Entry : Y=highest page of static workspace.
Exit : Do not alter or claim.

&24 (36) Dynamic Workspace Requirements M/C
This call asks you how much workspace you will require in HAZEL. Unlike call &22, we are not dealing in memory addresses at this stage, just numbers. Without bothering to read the value of Y, just decrement it by the number of pages you will require.
Entry : Y=&DC minus number of pages claimed by higher priority
Exit : Entry value minus pages required by you.

&25 (37) Advise Filing System Information M/C
Only filing systems should respond to this call. You should write information to the eleven bytes pointed to by (&F2),Y. The information is:

Bytes 1-8 : Filing system name (ie, ADFS, DISC, etc.) terminated by &20's, ie, ASCII spaces.
Byte 9 : Lowest file handle used by the filing system.
Byte 10 : Highest file handle used by the filing system.
Byte 11 : Filing system handle

On exit Y is increased by 11. If you don't respond to this call, the temporary FS mechanisms (see Chapter 8) won't work. Some 'official'

filing systems, such as VFS, and TELESOFT FS don't respond and thus won't work as temporary filing systems. This call must not be claimed. Here is a typical coding of the response to this call:

```
LDX #0
.nloop
LDA fsname, X                   \ Copy name to (&F2)
STA (&F2),Y
INY
INX
CPX #8
BNE nloop
LDA handlo
STA (&F2),Y
INY
LDA handhi
STA (&F2),Y
INY
LDA fsno
STA (&F2),Y
....        \ Restore any stack levels, A X but not Y
RTS
.fsname
EQUS "ANYFILE"                  \ Max 7 chars
EQUB &20
.handlo
EQUB &70 \ Lowest handle
.handhi
EQUB &74 \ Highest handle
.fsno EQUB &11 \ Filing system no (all imaginary!)
```

&26 (38) Close All Files M/C

The MOS issues this in response to a *SHUT command by the user. Only filing systems should respond to the call. A filing system should select itself, close all open files, and then exit. Don't worry about re-selecting the previous filing system - the MOS will sort all that out. The call should not be claimed.

&27 (39) Reset has Occurred M/C

The user has powered up, or pressed BREAK (including CTRL-BREAK). The call is used by ANFS to reclaim the NMIs. Otherwise it is a general initialisation call. You can display start-up messages, display initialise memory areas, check your environment, and do the usual startup things. Remember that when you receive this call, you haven't yet claimed any memory, and so you can't rightfully go

walking all over it. The call shouldn't be claimed. NB: The call is issued before filing system initialisation, so any code written to activate on this call should not access the filing system in any way.

&28 (40) Unknown CONFIGURE Option M/C

This call allows Sideways ROMs to use the non-volatile system settings in the normal manner. ROMs wishing to participate should intercept this call and first check if any text follows the command. Any such text is, as ever, pointed to by (&F2),Y.

If no text follows, ie, if (&F2),Y points to a carriage return (after disposing of surplus spaces), the user has issued *CONFIGURE alone, and the ROM should respond by typing a text message, consisting of each keyword to which it will accept a configuring command, together with some clues to the user of the parameters required. These will appear to the user after the end of the list generated by the MOS and if there is more than one, they should be in alphabetical order.

If some text is included, ie, if (&F2),Y points to a text string, then the ROM should compare the text against the CONFIGURE commands that it recognises. If the ROM in question has several star commands and/or *HELP responses, it may be economic to use the same routines to deal with this. As usual, if a match is made, the command should be claimed, otherwise, the ROM should provide and exit with all registers and the contents of (&F2) preserved.

Like star commands (service call &04), the MOS provides virtually no help in processing the command. The ROM must work out which parameter is being altered, and modify the CMOS RAM/EEPROM accordingly. Applications will normally use the bytes from 30-39. They should be used very sparingly, as there is very little non-volatile memory. If you are releasing a commercial product, it is a good idea to apply for a non-volatile byte allocation from Acorn.

&29 (41) Unknown STATUS Option M/C

This is very similar to call &28. The command *STATUS has been issued, and the text following the command starts at (&F2),Y. If there is no text, you should type out the currently configured setup (for your non-volatile items only), and exit. If text follows, you must try to recognise it, and if you do, print the status value of that item (only). The MOS prints out either the numeric or string setting, and you should follow this protocol.

&2A (42) Language startup M/C
This call is issued by the MOS when a language is about to start up. It can be used to activate language support ROMs.

&2B (43) Reserved All
This call is reserved for Acorn use. If you are writing a ROM and you need your own service calls, once again apply to Acorn for your own number, to avoid clashes with other producers.

&2C (44) Compact pointing devices C
This call is provided on the Compact only, to allow sideways ROMs to respond to pointing devices fitted to the User Port.

Entry: Y contains an offset from &200 to pointer information

The information is:

&200+Y+0	ADVAL low byte
&200+Y+1	X co-ordinate low byte
&200+Y+2	X co-ordinate high byte
&200+Y+3	Y co-ordinate low byte
&200+Y+4	Y co-ordinate high byte
&200+Y+5	Spare
&200+Y+6	Spare

If you are responsible for moving a pointer or whatever in response to this information, you should claim the call. Otherwise, ignore it.

&FE (254) Tube System has Initialised B/B+/M
This call is issued after I/O processor OSHWM has been determined. The call is used to advise the co-processor that it may use the memory between OSHWM and HIMEM for I/O related work. The call is always issued whether or not a co-processor is active, and on entry Y=&FF if Tube hardware present, Y=0 if Tube hardware absent.

&FF (255) Tube System about to Initialise B/B+/M
This call is issued after BREAK and CTRL-BREAK, if Tube hardware is fitted and active, immediately before generating the Tube sign-on message, and initialising the configured (Master) or default (B/B+/E) filing system. This call must be claimed to start the Tube.

Languages
These are the calls you would expect a language to respond to:

&02	Allocation of dynamic workspace in main RAM.
&04	for selection using *languagename.
&09	Standard although not extended *HELP.
&22	Allocation of dynamic workspace in HAZEL.
&24	Dynamic workspace claiming.

A language is entitled to use zero page &00-&8F on the language processor, although convention deems that &70-&8F is reserved for user code. It should perform LDX #&FF:TXS, and has exclusive ownership of &400-&7FF. The area between OSHWM and HIMEM can be used, the boundaries of which can be read with OSBYTEs &83 and &84 respectively. A language should redirect BRKV to itself, and deal with errors in a sensible way, or more pertinently, provide a well-defined point to return to after the error message has been printed. Interrupts must be enabled when a language starts.

Filing Systems
Filing Systems should respond to the following service calls:

&01	Allocation of static workspace in main RAM.
&02	Allocation of dynamic workspace in main RAM.
&03	Auto-boot and reading of explicit key and SHIFT key.
&04	For selection using *fsname and utilities (but not commands).
&09	Standard and possibly extended *HELP.
&0A	Workspace claim call.
&0B	NMI release.
&0C	NMI claim.
&0F	Vector claim.
&10	File closure warning.
&12	Initialise filing system.
&21	Allocation of static workspace in HAZEL.
&22	Allocation of dynamic workspace in HAZEL.
&23	Top of static workspace.
&24	Indicate dynamic workspace.
&25	Filing system information.
&26	Participate in *SHUT.
&27	Reset has occurred.

On start-up a filing system should:

1. Call OSFSC with A=6.
2. Set up the necessary extended vectors (see below).
3. Issue service call &0F once only (cf NFS which claims twice!)
4. Find and restore any files left open when this filing system was last shut down.
5. Exit, ready to respond to commands.

A filing system has ownership of :

&A0-&A7:	Whilst it owns the NMI. Data here will be corrupted if it is left while you wait to own the NMI again.
&A8-&AF:	Workspace for filing system utility * commands.
&B0-&BF:	Scratch space. Cannot be preserved between calls.
&C0-&CF:	Preserved until you are deselected.
&D00-&D5F:	NMI code area. Available on the same terms as &A0-&A7.

Extended Vectors

A third way of accessing code in Sideways ROM is through extended vectors. This involves claiming one of the standard system vectors, and directing it to your own ROM based code. It isn't as simple as just loading the vector with the address in ROM, as this doesn't give the MOS a chance to manage the paging in and out of the various banks. To point a standard Page 2 vector (see Appendix E for a list) to a Sideways ROM, this is what to do:

1. Calculate the vector number. This is: (Vector address-&200) DIV 2. Call this 'n'.
2. Set the vector to point to &FF00+3*n.
3. Read the extended vector address using OSBYTE &A8. It is returned in X(lo) and Y(hi). Call this 'e'.
4. Set e+3*n to your ROM routine address (lo).
5. Set e+3*n+1 to your ROM routine address (hi).
6. Set e+3*n+2 to your ROM number.

Your ROM would normally do this as part of initialising, or reacting to a 'switch on' star command. When this has been set up, you can safely assume that all calls which JMP(vector) will result in pointing

to your ROM. Remember that BREAK resets all vectors, and selecting a filing system tends to reset all vectors to do with files.

Note that extended vectors are not required when directing BRKV to the current language. Just put the ROM address into BRKV.

Clock ROM (Master 128 only)
Listing 6.1 is an example of the sort of utility type program that can be written in Sideways ROM format and then loaded into Sideways RAM for use. When installed it will display a real time clock in the corner of the screen. The time is generated from the CMOS real-time clock fitted to the Master 128. Several of the new system calls are needed to handle the display and update the clock. The code also contains some of the new 65C02 opcodes supported by BASIC 4 assembler. Enter the program, save it to disc, and then RUN it. To use the clock, press CTRL-BREAK, then type *CLOCK ON.

How it works
The main job of updating the clock, which would in the past have required some interception of the interrupt vectors, is now controlled by the new OSBYTE calls controlling 'ROM semaphore'. These new calls are OSBYTE &16 and &17. Acorn describe these calls as 'increment' and 'decrement' the ROM semaphore. The Electron and Master MOS supports a system known as ROM polling, where the ROMs are offered a service call (&15) every 10 ms.

This is very similar to an interrupt except that no vector claiming is necessary, and all that a ROM has to do to get the poll call is issue an OSBYTE &16. There are no parameters to the call. Conversely, an OSBYTE &17 will cancel the polling without further action.

The reason that the terms 'increment' and 'decrement' are used, rather than the more usual 'on' and 'off', is because more than one ROM may claim polling. Therefore the polling is only turned off when all ROMs relinquish their claims. OSBYTE &B3 tests the state of the semaphore. This is used in the Clock ROM to avoid double incrementing, ie, if *CLOCK ON is typed when the clock is already on.

Not only is it nice to be freed from all the usual vector claiming, but also, as anyone who has tried to put routines on interrupts will know, the usual problems of sorting out the ESCAPE key, errors, timing, etc.,

are removed because it's all handled by the MOS. Of course you still have to keep your routines quick unless you make them re-entrant.

The next new feature is the new service call to claim workspace in the 'hidden' RAM with service call &22 instead of the usual service call &02. This is for a page of buffer memory to put the time data from the OSWORD call. This buffer could have been at the end of the code for use in Sideways RAM, but that wouldn't work in a ROM.

The most important call for this application is the routine to read the clock itself. This is done with an OSWORD call (&0E). The ROM uses call 0 to read a string, and then a standard print routine to display the time only.

You will see a number of new 65C02 instructions in the program including PHX, PHY, and INA (INC A) and DEA (DEC A). Note the use of OSBYTE &A5 to read the cursor position rather than the more usual OSBYTE &86. This new call reads the output cursor, not the machine cursor, so returns to the right place when COPYing information.

Listings

Listing 6.1

```
 10 REM Clock ROM
 20 REM (c) Dave Atherton 1987
 30 REM Master 128 only
 40 REM MOS: A Dabhand Guide
 50 :
 60 REM Requires 2 bytes of zp (&9E/&9F)
 70 REM at all times. No other main memory
 80 REM workspace required. 1 page hidden
 90 REM RAM claimed.
100 :
110 zp=&9E
120 zp1=&9C
130 REM &9C is presered
140 osnewl=&FFE7
150 oswrch=&FFEE
160 osbyte=&FFF4
170 osword=&FFF1
180 gsinit=&FFC2
190 DIM M% 1000
200 FOR pass=4 TO 7 STEP 3
210 P%=&8000
220 O%=M%
230 [OPT pass
240 EQUW &00
250 EQUB &00
260 JMP serice
```

```
270 EQUB &82 \ Serice ROM
280 EQUB c-&8000
290 EQUB &11
300 .title
316 EQUS "CLOCK ROM"
320 EQUB 0
330 .ersion
340 EQUS "1.10"
350 .c
360 EQUB 0
370 EQUS "(C)1987 Dave Atherton"
380 EQUB 0
390 :
400 .serice
410 PHA
420 PHX
430 PHY \ Stack all registers
440 CMP #9
450 BEQ help
460 CMP #4
470 BEQ command
480 CMP #&15
490 BEQ poll
500 CMP #&22
510 BNE chkwsi
520 JMP wspace
530 .chkwsi
540 CMP &624
550 BNE out
560 DEY
570 PLA
580 PLX
590 PLA
600 RTS
610 .out \ Exit unclaimed
620 PLY
630 PLX
640 PLA
650 RTS
660 .zout \ Exit after action
670 PLY
680 PLX
690 PLA
700 LDA #0
710 RTS
720 .command
730 JMP command2
740 .help
750 JMP help2
760 .poll \ Execute clock routine
770 LDA #30
780 JSR setzp
790 LDA (zp) \ New 65CO2
800 DEA      \ ditto
810 STA (zp) \ ditto
820 BNE out  \ Only action every 99 polls (every sec)
830 LDA #99
840 STA (zp)
```

```
 850 LDA #&75
 860 JSR osbyte \ Get VDU status
 870 PHX \ Store it
 880 LDA #&10
 890 STA &D0 \ Put new status in
 900 LDA #165
 910 JSR osbyte \ Correctly return o/p cursor when split
 920 PHY
 930 PHX \ Store xy - new 65C02
 940 LDA #135
 950 JSR osbyte
 960 LDA #31
 970 JSR oswrch
 980 TYA
 990 AND #7
1000 TAY
1010 LDA mode, Y
1020 JSR oswrch
1030 LDA #0
1040 JSR oswrch
1050 JSR setzp
1060 LDA #0
1070 STA (zp)
1080 LDX zp
1090 LDY zp+1
1100 LDA #14 \ Read CMOS clock as string
1110 JSR osword
1120 LDA #16
1130 JSR setzp
1140 LDX zp
1150 LDY zp+1
1160 JSR print
1170 LDA #31
1180 JSR oswrch
1190 PLA
1200 JSR oswrch
1210 PLA
1220 JSR oswrch
1230 PLA
1240 STA &D0 \ Restore old VDU status
1250 JMP out
1260 .wspace
1270 \ entry A=22, X=? Y=first page aailable
1280 TYA
1290 STA &DF0,x
1300 \ Exit with incremented Y
1310 INY
1320 PLX
1330 PLX \ Throw away Y, pop X
1340 PLA
1350 RTS \ and get out
1360 \
1370 .setzp \ Set zp, zp+1 to point to buffer + A
1380 STA zp
1390 LDX &F4
1400 LDA &DF0,X \ Get workspace page
1410 STA zp+1
1420 RTS
```

```
1430 \
1440 .help2
1450 JSR gsinit
1460 LDA (&F2),Y
1470 CMP #13
1480 BNE helpend
1490 JSR osnewl
1500 LDX #title MOD 256
1510 LDY #title DIV 256
1520 JSR print
1530 LDA #32
1540 JSR oswrch
1550 LDX #ersion MOD 256
1560 LDY #ersion DIV 256
1570 JSR print
1580 JSR osnewl
1590 .helpend
1600 JMP out
1610 \
1620 .args \ If not followed by ON or OFF
1630 JSR osnewl
1640 LDX #argt MOD 256
1650 LDY #argt DIV 256
1660 JSR print
1670 JSR osnewl
1680 JMP zout
1690 \
1700 .command2
1710 LDX #0
1720 .mloop
1730 LDA (&F2),Y
1740 AND #&DF
1750 CMP table,X
1760 BNE nomatch
1770 INY
1780 INX
1790 BNE mloop
1800 .nomatch
1810 LDA (&F2),Y
1820 CMP #13
1830 BEQ args
1840 CMP #32
1850 BEQ ok
1860 .cout
1870 JMP out
1880 .loopc
1890 INY
1900 .ok
1910 LDA (&F2),Y
1920 CMP #32
1930 BEQ loopc
1940 CMP #13
1950 BEQ args
1960 AND #&DF
1970 CMP #ASC "O"
1980 BNE args
1990 INY
2000 LDA (&F2),Y
```

```
2010 AND #&DF
2020 CMP #ASC"F"
2030 BEQ off
2040 CMP #ASC"N"
2050 BNE args
2060 .on
2070 JSR state
2080 BNE on_now
2090 LDA #22
2100 JSR osbyte \ Inc semaphore
2110 LDA #31
2120 JSR setzp
2130 LDA #&FF
2140 STA (zp)
2150 .on_now
2160 JMP zout
2170 .off
2180 JSR state
2190 BEQ off_now
2200 LDA #23
2210 JSR osbyte \ Dec semaphore
2220 LDA #31
2230 JSR setzp
2240 LDA #0
2250 STA (zp) \ Clock off
2260 .off_now
2270 JMP zout
2280 .state
2290 LDA #179
2300 LDX #0
2310 LDY #255
2320 JSR osbyte \ read state of semaphore
2330 CPX #0
2340 RTS
2350 \
2360 .print \ print string XY to a zero
2370 LDA zp1
2380 PHA
2390 LDA zp1+1
2400 PHA
2410 STX zp1
2420 STY zp1+1
2430 LDY #0
2440 .prloop
2450 LDA (zp1),Y
2460 CMP #32
2470 BCC done
2480 JSR oswrch
2490 INY
2500 BNE prloop
2510 .done
2520 PLA
2530 STA zp1+1
2540 PLA
2550 STA zp1
2560 RTS
2570 .mode EQUB 72
2580 EQUB 32
```

```
2590 EQUB 12
2600 EQUB 72
2610 EQUB 32
2620 EQUB 12
2630 EQUB 32
2640 EQUB 32
2650 .table EQUS "CLOCK"
2660 EQUB 255
2670 .argt EQUS "Syntax : CLOCK <ON|OFF>"
2680 EQUB 0
2690 .end
2700 ]
2710 NEXT
2720 S$=STR$~M%
2730 E$=STR$~(end-&8000+M%)
2740 REM Save in sideways RAM
2750 OSCLI "SRWRITE "+S$+" "+E$+" 8000 Y"
```

Listing 6.1 CMOS Clock ROM.

7 : The Tube

Introduction.

The Tube is Acorn's proprietary name for the 2MHz interface fitted to Models B, B+ and Master 128. The principles referred to here also apply to the Electron Tube produced by Advanced Computer Products Ltd. Its purpose is to allow another CPU and memory to operate whilst the keyboard, screen and other Operating System input/output (I/O) facilities are being controlled by the CPU on the main micro (known as the I/O or host processor). This enables computers to be built which only require processor, memory and software, and not the usual I/O considerations. The processor attached to the Tube is known as the second-, language-, or co-processor. Co-processors developed to date include a fast 65C102, Z80, 32016, 80186, and ARM. Third parties have also produced 6502, Z80 and 68000 co-processors. Co-processors for the Master 128 only, do not need to be boxed or powered as they can be plugged directly into the main PCB. Adaptors have been produced to interface Master 128 co-processors to the Model B/B+.

In a two processor system, the 32-bit addressing used by the MOS becomes relevant, with &FFFF0000 to &FFFFFFFF being the I/O processor memory map, and &0 upwards being the co-processor memory map. 32 bits offers 4Gb (Gigabytes—4,000 million bytes) of address space, so it is unlikely that the co-processor would ever need the top 64k of address space. When a co-processor has control of the system, the important Operating System calls are still available. On a 6502 co-processor, these are between &0000FF00 and &0000FFFF. On non-6502 co-processors there is support for OSBYTE and OSWORD, but its implementation depends on the processor in question. For example, the Z80 uses HL to take the place of X and Y, but sticks to &FFF1 and
&FFF4 as entry points. In general, the support for OSRDCH, OSWRCH and the file commands will be there, but not offered to the user. This is because the other co-processors will usually have another operating system (CP/M, MS-DOS, DOS+, OS/9, Unix, CP/M 68K, Panos, etc.) which provide their own I/O entry points.

Although the principles apply to all co-processors, detailed information here is only supplied on the 6502 co-processor.

Tube Host Code and OS

The work of Tube data transfer is shared between two programs—the Tube Host Code and the Tube Operating System. The Host Code, which runs on the I/O processor, is the same irrespective of the co-processor in use. It is supplied in the DFS ROM from version 1.20, all versions of the 1770 DFS, NFS from version 3.60, all versions of ADFS, and the Master MOS ROM. It is also supplied under licence in the Watford Electronics 8271 DFS ROM from version 1.44.

When a co-processor is present, powered up, and software switched in, the Host Code is copied from the Sideways ROM to locations &400 to &6FF. This area is otherwise unused as it is language workspace, and any languages implemented will at this time be running on the other processor. The Host Code is entered through three documented entry points, and takes parameters through A, X, Y and P.

&400 Copy a language across the Tube.
&403 Copy the ESCAPE flag to Tube memory.
&406 Multi-purpose data transfer.

The Tube Operating System runs on the co-processor, and so a different one, written in the appropriate machine code, exists for each. On the 6502 system, this is in memory from &F800 to &FFFF, with locations &FEF8 to &FEFF acting as the Tube registers and copied out to I/O. This code will run from a reset and so locations &FFFC/D have &F800 written into them.

A program can find out which side of the Tube it is running on by issuing an OSBYTE &82 call to read the higher order address which is greater than &FF00 if it is running on the I/O side, or less if it is running on the Tube. The program should not try to find out which side of the Tube it is running on by issuing OSBYTE &EA.

&400 Copying a language across the Tube All

This call is used by the MOS, (as part of OSBYTE &8E) when the Tube is active. It is not of any use to the user. Note that although only the language part of a language/service ROM is needed on the co-processor (indeed the same ROM may contain say Z80 language code, and 6502 service code), and a language may only be 8k in length,

all 16k is nevertheless transferred across the Tube, to the relocation address specified. As the language jump must be coded in the target co-processor's machine code (see Chapter 6 for how it knows) problems may arise if a processor is used which cannot code an unconditional jump in three bytes!

On the Model B/B+ any event issued during a language transfer will cause the computer to crash, so events should be disabled before calling the transfer. This is not necessary on the Master Series.

&403 Copy the ESCAPE flag to Tube memory All

This call merely takes the contents of zero page location &FF on the I/O processor, and copies it to the co-processor, thus allowing the Tube OS to recognise the ESCAPE event.

&406 Writing data across the Tube All

On a co-processor system high level applications such as languages, and special disc operating systems such as MS-DOS will run on the co-processor side—provided they have been written to do so of course. The low-level I/O drivers such as ADFS sector read/write, and the MOS will be on the I/O processor. Also any utility programs written in ROM are on the I/O processor and in 6502 machine code.

Obviously if a program running on one side of the Tube needs to access memory on the same side, normal LDA/STA or equivalent is used. Code running on the I/O processor when a co-processor is active must be written in assembler. This code is often put in Sideways ROM for many reasons to do with space, presence at power on, and elegance.

Accessing I/O memory from the co-processor

Programs running on any co-processor may use OSWORD &05 and &06 to read data from and write data to the I/O processor. The call is slow. If an OSBYTE is available as an alternative, eg OSBYTE &A0, then use this instead. With OSWORD &05 and &06 a 32-bit I/O address is given at XY, and the byte is placed/read in XY+4. Note that XY+2 and XY+3 will always be &FF.

There are other ways to pass data to the I/O processor. The *CODE (OSBYTE &88) and *LINE commands, and OSWORD calls with A greater than &DF will all pass control to USERV, &200/&201 on the I/O processor. Service code should understand the format of *LINE, OSWORD etc. This code must be placed in the I/O processor, and

USERV must point to it. Of course OSBYTE, OSWORD, '*' and most other MOS routines will pass calls with unknown values, for example, OSWORD with A=&90, or *FX 55,2 to the Sideways ROMs.

Accessing co-processor memory

Access to the co-processor memory from the I/O processor can be very useful. For example, you couldn't implement a filing system without it. Service ROMs also need the feature, especially those that deal with languages such as BASIC toolkits.

A program resident on the I/O processor wishing to read data from the co-processor should procede as follows:

1. Issue a system call to claim the Tube.
2. Issue a system call advising read or write, number of bytes, and co-processor start memory.
3. Perform a succession of read from or write to Data Register 3 to get the data, loading/storing it to/from I/O memory.
4. On completion, issue a system call to release the Tube.

Claiming and releasing the Tube

If you claim the Tube, you are guaranteed that no other program (ie, an interrupt-driven background task) will try to use it until you have finished (unless they ignore protocols!) but the claim does not actually affect data transfer. To claim, you need a Tube identity. This is a number in the range &00 to &3F. Some numbers have already been allocated as follows:

0	Cassette Filing System
1	Disc Filing System
2	Econet low-level primitives
3	Network filing system
4	Advanced Disc Filing System
5	Teletext Filing System
6	Reserved
7	Videodisc Filing System
8	Sideways RAM utilities
9	Z80 second processor usage
10-14	Reserved
15	Allocated to a third party
63	MOS

It is best that you start at the other end using say, identity &3E. (&3F is used by the MOS) You should perform a JSR to &406, with A containing &C0 plus the identity number, in this example A would contain &FE as &C0+&3E= &FE. If you are releasing a commercial product, it would be wise to contact Acorn to be allocated a number.

```
.claim
LDA #&FE \ Claim tube with id=&3E (other end)
JSR &406
```

Interrupts must be enabled during the claim. If the Carry flag is clear on exit from the claim, it has failed and you should try to claim again. Every claim (with one exception) needs a corresponding release. To release the Tube so that it is free for another user, you should call &406, with A set to &80 plus the identity number.

```
.release
LDA #&BE  \ &80+&3E
JSR &0406 \ &3E no longer needs Tube
```

For both claim and release calls the values of X and Y are irrelevant.

Advising Tube OS of data transfer required

After claiming the Tube, the next step is to advise the Tube OS of the size and direction of the data transfer required. This is done by calling &406, with A containing a parameter between 0 and 7. This parameter defines the action to be taken as follows:

A=0 Read any number of bytes slowly (1 init. only required).
A=1 Write any number of bytes slowly (1 init. only required).
A=2 Read 2 bytes.
A=3 Write 2 bytes.
A=4 JMP to routine (no release call required).
A=5 Used by the system to inform the co-processor that the Tube has been released.
A=6 Read 256 bytes.
A=7 Write 256 bytes.

The X and Y registers on entry point to a four-byte block in the I/O processor, which contains a second processor address to be used as the first address for the data transfer. For multi-byte transfers, the Tube OS will automatically increment this. Note that this call does not actually transfer the bytes.

Immediately after calling this routine, you must wait for the Tube to do its bit, by executing some null instructions. The time to wait is 24

μs (48 cycles) for the 'slow' setup, 26 μs (52 cycles) for the 2 byte setup, and 19 μs (38 cycles) for the 256 byte setup. NOP will waste 1 μs. JSR to an RTS will waste 6 μs.

Writing data to/reading data from the Tube

Having setup the Tube to receive or send a known number of bytes the rest of the procedure is very simple. This procedure applies to any co-processor. The delays should still be observed even for very fast co-processors such as the 80186 and ARM.

To Read:

1. If necessary, wait for a few microseconds—see below or
2. Test bit 7 of Status register 3 (&FEE4) until it is set.
3. Read the byte from Data Register 3 (&FEE5).
4. Store the byte in main memory, or process it immediately.
5. If 2 or 256 byte transfer repeat all steps once, or 255 times.

To Write:

1. Get the byte to be transferred from main memory or
2. If necessary, wait for a few microseconds—see below.
3. Test bit 6 of Status register 3 (&FEE4) until it is set.
4. Write the byte to Data Register 3 (&FEE5).
5. If 2 or 256 byte transfer repeat all steps once, or 255 times.

Delays

As well as the delays required after each setup call, there are also delays required between each access to Tube memory. These do not necessarily mean that NOPs need to be performed you may do some useful processing as long as you don't access &FEE4 and &FEE5 again within the delay time specified. If you are polling the Status Register, you don't need to bother about these delays.

Delay required between each read of &FEE4/&FEE5 (in μs)

Read:	1 byte-24 μs	2 bytes-26 μs	256 bytes-19 μs
Write:	1 byte-24 μs	2 bytes-13 μs	256 bytes-10 μs

In practice these figures are maxima, and lesser values may be experimented with. This is because Tube Register 3 is a FIFO buffer, ie, First In First Out. If you access the Tube registers using OSBYTE &96 and &97, you can ignore all these delays - the OSBYTE processing, takes more than enough time!

Transferring control to the Co-Processor

A program running on the I/O processor can transfer control to a routine on the co-processor with a JMP &406 instruction. Note that this is a 'JMP' not a 'JSR'—there is no return to the point after the I/O program made the call. The call is made to &406 with A=4 and XY pointing to a four byte I/O memory block. The memory block contains the address of the routine on the co-processor. Should you wish to transfer control back again, the co-processor program must invoke an I/O routine. Current Tube Host Code and Tube Operating Systems do not provide for any realistic concurrent processing except by using interrupts. (The hardware is perfectly capable). Listing 7.1 provides an example program which will transfer data across the Tube.

Listing 7.1.

```
 10 REM Control Tube memory from I/O side
 20 REM (c) Dave Atherton 1987
 30 REM for B/B+/E/M
 40 REM MOS : A Dabhand Guide
 50 :
 60 M%=&2F00
 70 osrdch=&FFE0
 80 osasci=&FFE3
 90 statR3=&FEE4
100 dataR3=&FEE5
110 block=&70
120 buf2P=&C200
130 opt=3
140 FOR pass=0 TO opt STEP opt
150 P%=M%
160 [OPT pass
170 \ Get a key, put it over the tube
180 \ clear registers, get it back &
190 \ then print it on screen as ASC
200 \
210 LDA #buf2P MOD 256
220 STA block
230 LDA #buf2P DIV 256
240 STA block+1
250 LDA #0
260 STA block+2
270 STA block+3
280 JSR claim
290 .loop
300 JSR osrdch      \ get a key
310 STA block+12    \ save it (at &7C)
320 LDA #1
```

```
  330 JSR setup      \ initialises the write op.
  340 .delay
  350 BIT statR3
  360 BVC delay      \ wait for space in the FIFO Tube
  370 LDA block+12
  380 STA dataR3     \ Put byte into FIFO Tube
  390 :
  400 \ This generates an NMI on the 2P
  410 \ whose OS responds by storing the
  420 \ byte, and incrementing the address
  430 \ ('buf2P' as passed here on .setup)
  440 \ for the next transfer, using nasty
  450 \ self modifying code. (No line 460)
  470 LDA #0
  480 JSR setup
  490 .delay2
  500 BIT statR3
  510 BPL delay2     \ wait for byte in FIFO to read
  520 LDA dataR3     \ Get. that. byte
  530 JSR osasci     \ and display it.
  540 CMP #27
  550 BNE loop
  560 JSR release    \ tell Tube system bye-bye.
  570 RTS
  580 :
  590 .claim         \ Set carry if claim successful
  600 LDA #&FF
  610 JSR &406
  620 BCC claim
  630 RTS
  640 .setup
  650 LDX #block MOD 256
  660 LDY #block DIV 256
  670 JSR &406
  680 RTS
  690 .release
  700 LDA #&BF
  710 JSR &406
  720 RTS
  730 .end
  740 ] NEXT
  750 A$="SAVE TubeKey "+STR$~M%+"+"+STR$~(end-M%)+"
FFFF"+STR$~M%+" FFFF"+STR$~M%
  760 PRINT A$
  770 OSCLI A$
  780 *RUN TubeKey
```

Listing 7.1 - Controlling Tube memory from the I/O side.

Notes for 6502 Co-Processor programs

These are the areas to study if you want your programs to run on a 6502 second processor, or are trying to debug a program that won't!

The following vectors and un-vectored system calls are not supported by the 6502 co-processor, and should not be called directly, ie, by JMP OSRDRM or JMP (VECTOR). They are all directed to a BRK in the Tube OS, giving the error message 'Bad'.

Unsupported calls

OSWRSC, OSRDSC, OSRDRM : These calls, relevant to paged areas of the I/O memory map must only be issued by the I/O processor. The way to deal with this is to install suitable routines on the I/O processor which pass data back to the co-processor.

OSVDU : Not needed, as OSWRCH is supported.

OSEVEN : Not appropriate as events cannot be generated by co-processors. They can however be handled by co-processors.

GSINIT, GSREAD : These are really only of use to writers of ROM software using one or more '*' commands.

Vectors

USERV, VDUV, KEYV, FSCV, UPTV, INSV, REMV, NETV, CNPV
These are the vectors which don't have associated page &FF system calls, and so there is no need for co-processor support. All these vectors are concerned with I/O actions. They can be intercepted on the I/O processor to point to alternative I/O routines.

Other matters

1. Primary OSHWM is &800 not &E00, so BASIC supported by machine code at &900 should either reset PAGE, or put the machine code elsewhere, including the I/O processor.
2. The character font is fully exploded, and can only be accessed legally. Thus you can't rely on 224-255 and 128-159 being the same characters. Some programs start with one set of numbers and indiscriminately move to another.
3. If the screen is driven by directly writing to memory, this will not appear when the program is run on a co-processor. This is particularly so with arcade games, a pity because the extra speed would be nice.

4. Other direct access to system RAM and ROM is likely to flounder in a similar manner, ie, direct calls to the OS or BASIC ROMs, trying to read bytes of SRAM, Operating System variables or workspace, and so on—but not language workspace. This should be moved to the co-processor, eg,

```
PRINT ?&18 * 256
```

   typed on co-processor BASIC would still return the value of PAGE (not OSHWM).

5. OSBYTE calls below 128 will have lost the contents of Y before they pass through BYTEV on the I/O processor. It is bad practice anyway to use Y in these calls as the MOS disregards it anyway. The point is made because some third-party ROMs may use such an OSBYTE and require a Y parameter.

6. Programs may run faster due to the faster processor.

7. Programs that rely on undocumented 6502 instructions may fail on the tidier 65C02—see Chapter 3.

8. Stand-alone machine code programs can ignore the presence of a Tube by having *LOAD and *EXEC addresses of &FFFFxxxx. You must *RUN them directly from the FS.

Summary of Tube Registers

Status Register 1 - R1STAT - &FEE0

b7	DA1	Data in Data Register 1
b6	NF1	Not filled Data Register 1
b5	P	Set parasite reset active low
b4	V	Enable 2 byte FIFO operation of R3
b3	M	Enable parasite NMI from R3DATA
b2	J	Enable parasite IRQ from R4DATA
b1	I	Enable parasite IRQ from R1DATA
b0	Q	Enable host IRQ from R4DATA

Appears as &FEF8 on the 6502 Second Processor memory map.

Data Register 1 - R1DATA - &FEE1

This is used for OSWRCH, events and the ESCAPE flag. It should not be accessed by the user. Writing to this register causes an IRQ (or equivalent) on the co-processor. It appears at &FEF9 on the 6502 Second Processor memory map.

Status Register 2 - R2STAT - &FEE2

b7 DA2 Data in R2DATA
b6 NF2 Not filled R2DATA
b5-0 Don't care
Appears as &FEFA on the 6502 Second Processor memory map.

Data Register 2 - R2DATA - & FEE3

This is used by all other MOS calls, and again should not be accessed by the user. It appears at &FEFB on the 6502 Second Processor memory map. In the protocol table below, the Tube protocols used by each MOS function are listed.

Status Register 3 - R3STAT - &FEE4

b7 DA3 Data in R3DATA
b6 NF3 Not filled R3DATA
b5-0 Don't care
Appears as &FEFC on the 6502 Second Processor memory map.

Data Register 3 - R3DATA - &FEE5

User register for data transfer and I/O errors. Used by the calls documented in this chapter. Writing to or reading from this register causes an NMI (or equivalent) on the co-processor. Appears as &FEFD on the 6502 Second Processor memory map.

Status Register 4 - R4STAT - &FEE6

b7 DA4 Data in R4DATA
b6 NF4 Not filled R4DATA
b5-0 Don't care
Appears as &FEFE on the 6502 Second Processor memory map.

Data Register - R4DATA - &FEE7

This register is used for the control of data transfer. It should not be accessed by the user. Writing to this register causes an IRQ (or equivalent) on the co-processor. It appears as &FEFF on the 6502 Second Processor memory map.

Tube Protocols

Here are details of the transfers which take place between the I/O processor and co-processor, to service various system calls. Transfers

are through Data Register 2 unless otherwise specified. This is really a rundown of how the MOS and Tube OS do their jobs. It is useful for anyone trying to bypass the Tube OS. The letters C and H are used to represent Co-processor and Host processor respectively. For example, in the first call, OSRDCH, the user calls the routine with JSR &FFE0, which causes the Tube OS to send a code (&00) to the I/O processor. In response to this, the MOS on the I/O processor processes the call, which in this instance reads the keyboard until a key is pressed. The MOS, on the I/O side, then sends two bytes back to the Tube, the minimum needed to transfer all the parameters. The first byte contains the carry flag in bit 7 (the other bits are irrelevant), and the second byte contains the ASCII value of the key press.

OSRDCH

C to H &00 Reason code
H processes MOS call
H to C xx C flag in bit 7
H to C xx Result from RDCH

OSCLI

C to H &02 Reason code
C to H xx..xx.....&0D Command line terminated with &0D
H processes MOS call
H to C &7F Acknowledge

Short OSBYTE (<128)

C to H &04 Reason code
C to H xx X register contents
C to H xx A register contents
H processes MOS call
H to C xx X result

Long OSBYTE (>127)

C to H &06 Reason code
C to H xx X register
C to H xx Y register
C to H xx A register
H processes MOS call
H to C xx C in bit 7 (not used by OSBYTE &9D)
H to C xx Y result (not used by OSBYTE &9D)
H to C xx X result (not used by OSBYTE &9D)

OSWORD (except 0)

C to H	&08	Reason code
C to H	xx	A register
C to H	xx	Number of parameters
C to H	xx..xx.....	Parameters
H processes MOS call		
C to H	xx	Return parameter count
H to C	xx..xx.....	Return parameters

OSWORD 0

C to H	&0A	Reason code
C to H	xx	ASCII maximum
C to H	xx	ASCII minimum
C to H	xx	Maximum length
C to H	&07	I/O buffer most significant address
C to H	&00	I/O buffer least significant address (The MOS uses &FFFF0700)
H processes MOS call		
H to C	&FF	Escape flag or
H to C	&7F	Acknowledge
H to C	xx..xx.....&0D	Line terminated by &0D

OSARGS

C to H	&0C	Reason code
C to H	xx	Y register
C to H	xx..xx..xx..xx	4 bytes of zero page pointed to by X (high byte first, low byte last)
C to H	xx	A register
H processes MOS call		
H to C	xx	A result
H to C	xx..xx..xx..xx	4 byte block, high byte first as above

OSBGET

C to H	&0E	Reason code
C to H	xx	Y register
H processes MOS call		
H to C	xx	Carry flag in bit 7
H to C	xx	A result

OSBPUT

C to H	&10	Reason code
C to H	xx	Y register
C to H	xx	A register
H processes MOS call		
H to C	&7F	Acknowledge

OSFIND

C to H	&12	Reason code
C to H	xx	A register
If A=0:		
C to H	xx	Y register
H processes MOS call		
H to C	&7F	Acknowledge
If A<>0:		
C to H	xx..xx.....&0D	Filename terminated with &0D
H processes MOS call		
H to C	xx	A result (file handle or 0)

OSFILE

C to H	&14	Reason code
C to H	xx.....xx,&00	Filename terminated by zero
C to H	xx.....xx	16 bytes : Bytes 3-18 of OSFILE parameter block. High byte first, low byte last.
C to H	xx	A register
H processes MOS call		
H to C	xx	A result
H to C	xx.....xx	Bytes 3-18 of return block

OSGBPB

C to H	&16	Reason code
C to H	xx.....xx	13 bytes : OSGBPB parameter block. High byte first, low byte last.
C to H	xx	A register
H processes MOS call		
H to C	xx.....xx	Return parameter block
H to C	xx	Carry flag in bit 7
H to C	xx	A result

OSWRCH

C to H	xx	A register (uses Data Register 1)

I/O PROCESSOR BRK

H to C	&FF	Generate IRQ and reason code
H to C	&00	BRK
H to C	xx	Error code
H to C	xx..xx.....&00	Message terminated by zero

The first transfer of &FF is done through Data Register 4. The rest of the protocol uses Data Register 2.

EVENT

H to C	&00	Generate IRQ and reason code
H to C	xx	Y register
H to C	xx	X register
H to C	xx	A register

The transfers are done through Data Register 1.

ESCAPE FLAG CHANGE

H to C	xx	b7=1 : b6=New flag

The transfer is done through Data Register 1

TRANSFER START

H to C	xx	Operation : (A on entry to &0406)
H to C	xx	Number used to claim link
H to C	xx..xx..xx..xx	32 bit address (high byte first)
H to C	xx	Value used to reset Tube ULA

The transfers are done through Data Register 4.

8 : Filing Systems

There is no change to the MOS filing system interface on the Model B+, and not much on the Master 128 and Compact. This chapter lists only the extra filing system calls. If you are writing a filing system, then you must support these calls, or document the fact that you don't.

OSFIND

No change. Note that a new random access file on ADFS needs 64k of space, unlike the 16k needed under DFS.

OSGBPB/OSBPUT/OSBGET

No real change. Note that OSGBPB functions A=2 (Append bytes to file) and A=4 (Read bytes from PTR) are supported by the cassette FS on the Master 128 only. A=4 is also supported by the ROM filing system (RFS) on the Master 128 and Compact.

OSARGS

There are some new entries here, so the whole call is freshly documented. The four-byte zero page memory block, is pointed to by X, and may be on the I/O processor, or Second Processor.

Entry	**Result after call** X is preserved as is A except when specially set.
A=0:Y=0	Return current filing system number in A. There are now some new filing system numbers. See Appendix E for a list (B/B+/E/M/C).
A=0:Y=handle	Read PTR of file into zero page memory pointed to by X (B/B+/E/M/C).
A=1:Y=0	Return address of rest of '*' command. Used by disc/net based '*' commands to read their parameters (B/B+/E/M/C).

A=1:Y=handle	Write PTR of file from zero page memory pointed to by X (B/B+/E/M/C).
A=2:Y=0	Read filing system version number into A. On DFS/ADFS this will always return 1, as it will on ANFS, and NFS 3.60. It will however return 2 on NFS 3.34. It is not known entirely what use Acorn intend this call to be put to (B/B+ with NFS/M/C).
A=2:Y=handle	Read EXT of file into zero page memory pointed to by X. (B/B+/E/M/C)
A=3:Y=0	Return library filing system number in A. See the extended list in Appendix E. If no library FS is selected, then &FF is returned, which is not the same as having the current filing system equal to the library filing system (achieved by typing *LIBFS without any preceding filing system name), (M/C).
A=3:Y=handle	Write EXT of file from zero page memory pointed to by X. Files extended by this command are filled with zeros (M/C). Implemented on DFS after 2.25

OSFILE

Actions with A=0 to 6 and A=&FF are unchanged. There is a new call with A=7. This creates a file without transferring any data. The parameter block is set up in the same way as with A=0 (*SAVE).

Under DFS only the lock attribute was ever used. With ADFS (and NFS) there are further file attributes set up in the parameter block:

XY+14	b0	1=Can be read, 0=Cannot be read
	b1	1=Can be written to, 0=Cannot be written to
	b2	1=Has 'E' attribute, 0=File does not
	b3	1=Can't be deleted, 0=Can be deleted
	b4	1=Can be read by other users, 0=It can't
	b5	1=Can be written to by other users, 0=It can't
	b6-7	Not used
XY+15		Date file was last written to (01-31)
XY+16	b0-3	Month file was last written to (01-12)
	b4-7	Year file was last written to minus 81 (ie, 1987=6)
XY+17		Not used

Bit 2 of XY+14 only applies to ADFS, it is undefined in NFS. The date-stamping bytes XY+15 and XY+16 only apply to NFS, as do the attributes in bits 4-7 of XY+14. To maintain compatibility between disc, ADFS, and network, and future filing systems, you should set bits 4 to 7 of XY+14 to the same values as bits 0-3, and you should fill the date bytes with zero. The 'E' attribute is a special feature implemented only in ADFS which makes a file 'hidden' so that it can only be *RUN, and not *LOADed, or otherwise inspected. It is analagous to the 'lock' bit implemented on the cassette filing system.

OSFSC

The OSFSC calls have been extended, and are also for some reason omitted from the official Acorn Master Reference Guide. This call has therefore been documented here in full.

OSFSC is an entry point for some miscellaneous filing system control functions. It has no direct entry point, but is indirected through &21E, and is therefore called with JMP (&21E). You should place

```
JMP(&21E)
```

at the end of your code, say under label .osfsc, and then,

```
JSR osfsc
```

As with many other calls, the number loaded into the accumulator controls the function.

If you are writing a filing system, you must support all these:

A=0 An *OPT command. X and Y contain the parameters. The call passes through here last of all after going through OSCLI and OSBYTE &8B.

A=1 An end-of-file (EOF) check. X=handle. On exit X=&FF if EOF, X=0 if not EOF.

A=2 A */command (functionally equivalent to *RUN). XY point to the string following the */.

A=3 An unrecognised '*' command, XY points to the first letter of the command text.

A=4 A *RUN command, XY points to any following text.

A=5 A *CAT command. XY points to any following text.

A=6 New filing system start. Close all your files and preserve any data you need to preserve. Use OSBYTE &77 to close any *SPOOL and *EXEC files.
A=7 Filing system handles. You must return in X the lowest, and Y the highest handle possible.
A=8 The MOS has received a '*' command.

Commands 0 to 8 are applicable to all Acorn machines. The following FSCV commands are only applicable to the Master, following the integration of certain filing system specific commands into the MOS.

Master Series

A=9 A *EX command, XY points to any following text.
A=10 A *INFO command, XY points to any following text.
A=11 A *RUN command to the library filing system. You should only respond to this if you are the library FS. If so, respond as if the command were A=4.

Prohibited Filename Characters

Due to new features in ADFS, DFS and the MOS, it is recommended that none of the following characters are used in filenames.

.	full stop	:	colon
*	star	#	hash
$	dollar	&	ampersand
@	at	^	'hat', circumflex or up-arrow
"	quotes	-	minus
|	double bar		characters above ASCII 126

Using OSWORD &7F

OSWORD &7F is a general purpose command to access the 8271 disc controller. When the 1770 controller was substituted, Acorn wrote the new OSWORD &7F routine in such a way that it substituted the equivalent 8271 codes. For example the code for format on an 8271 controller is &63. The code for write track, the nearest equivalent on a 1770 is &20, but OSWORD &7F still accepts &63, and all other parameters as if the controller present were an 8271. Here is a list of the 8271 command codes, those emulated by the 1770 DFS are marked

with a *. A ‡ marks those calls deleted in DFS 2.40. The format of the command is given in Chapter 4.

Cmd	Description	No of Parameters
&35	Various setup commands	4
&40	Scan sector	5
&44	Scan deleted sector	5
&4A	Write 128-byte sectors	2
&4B	Write 256-byte or larger sectors *	3
&4E	Write 128-byte deleted sectors	2
&4F	Write 256-byte or larger deleted sectors *	3
&52	Read 128-byte sectors	2
&53	Read 256-byte or larger sectors *	3
&56	Read deleted sectors	2
&57	Read 256-byte or larger deleted sectors *	3
&5B	Read Track ID *	3
&5E	Verify 128-byte deleted sector	2
&5F	Verify 256-byte or larger deleted sector	3
&63	Format track *	5
&69	Seek track *	1
&6C	Read drive status	0
&7A	Write special registers *	2
&7D	Read special registers *	2
&E0	Read track *‡	1
&F0	Write track *‡	1

Temporary Filing Systems

A major addition to the Master Series MOS is the facility to use temporary and library filing systems. These additions have meant considerable changes to the system software and filing system protocols. The following notes detail the operation of the filing system control software on a Master Series micro. But firstly a brief glossary of terms.

Current filing system : This is the default filing system. It will be either the configured filing system, or a * command selected filing system, for example ADFS may be the configured filing system but DFS can become the current filing system by issuing the command, *DISC or *FX 143,18,4. The current filing system is used in the absence of a temporary filing system.

Active filing system : This is the filing system which is currently selected, and is the filing system which owns the absolute workspace, and has claimed the filing system vectors.

Library filing system : This is an optional filing system which is searched for transient, ie, disc based, commands not found on the current or temporary filing systems.

Filing system handler : This is the low level operating system code that sits between applications and the filing systems.

Transient command : A command located in secondary memory either as a executable machine code file, or an EXECable text file with the execution address suitably configured.

Page &DF

The RAM in HAZEL between &DF00 and &DFFF is used solely by the filing system handler and commands associated to files. The details are given below. Locations marked with a * are directly relevant to temporary/library filing systems.

&DF00	Current filing system number *
&DF01	Active filing system number *
&DF02	Library filing system number *
&DF03	Socket no. of ROM containing current filing system *
&DF04/5	Address of rest of command line for transient commands.
&DF06	Ending at &DFC1 with up to 17 filing system information blocks, terminated with a zero. Each block is eleven bytes long and consists of the filing system name padded with spaces to eight characters, the minimum and maximum handles supported, and the filing system number *
&DFC2	During *TYPE, LIST and PRINT if bit 7 is set, the GS format print routine is disabled, thus giving pure ASCII or *PRINT rather than *TYPE format), and setting bit 6 inhibits the printing of line numbers. For *BUILD and APPEND bit 7 is set for APPEND, clear during BUILD.
&DFC3/4	Line number (in BCD) for *BUILD, APPEND and LIST.

&DFC5	Last character printed by *LIST etc. It is used to prevent multiple CRLF.
&DFC6	Temporary filing system flag *
&DFC7	A block ending at &DFD3 forms the 13 byte OSGBPB block for the destination channel during *MOVE.
&DFD4	Source handle for *MOVE.
&DFD5	Destination handle for *MOVE.
&DFD6	High byte of *MOVE buffer location.
&DFD7	Length of *MOVE buffer in pages.
&DFD8/9	Pointer to destination in command line (*MOVE).
&DFDA/B	Copy of FSCV for active filing system *
&DFDC	Copy of ACCCON made by *MOVE.
&DFDD	If non-zero,*MOVE has changed ACCCON.
&DFDE	From here to &DFFF is currently unused.

The Filing System Handler

The purpose of the filing system handler is to intercept all relevant calls and select the required filing system. All these operations are transparent to the filing systems themselves. In order to detect the use of temporary filing systems the handler must intercept the seven filing system entry points. For all but FSC this is done by inserting code between the Page &FF call and the jump to the Page &2 vector. This means that, for example, JMP &FFDA is no longer the same as JMP (&214). As FSC has no mainstream entry point it is intercepted by changing FSCV to point the filing system handler, and copying the old FSCV to locations &DFDA and &DFDB. This process is performed each time a filing system changes the vectors. The filing system handler is also linked with the command line interpreter to allow the temporary filing system name to precede the command, eg,

```
*LOAD -ADFS-TEST
```

is the same as,

```
*-ADFS-LOAD TEST
```

The routines which intercept the FS vectors work as follows:

OSBPUT : The filing system appropriate to the file handle (in Y) is selected and the call is passed to the filing system through BPUTV.

If the handle does not match any filing systems, the current filing system is selected.

OSBGET : As for OSBPUT.

OSGBPB : If the call has an accumulator value of 1 to 4 (byte transfers) the filing system appropriate to the handle (contained in the first byte of the control block) is selected. For all other calls the current filing system is selected and the call is passed through GBPBV.

OSARGS : For calls in which Y<>0 (file related calls) the filing system is selected according to the handle (in Y), otherwise if invalid the current FS is selected. The call is then passed through ARGSV.

For calls with Y=0, and A<=3 the same procedure is followed:

Y=0,A=0:	The current filing system number (obtained from &DF00) is returned in A.
Y=0,A=1:	The address of the rest of the command line is copied from &DF04/5 to the zero page locations pointed to by X and X+1. This address is padded to 32 bits by inserting a high order address of &FFFF. The routine is then exited.
Y=0,A=2:	The call is exited with A=1 to signal network software is a version greater than NFS 3.60. In practice all Master Series computers will be fitted with a version of ANFS.
Y=0,A=3:	The routine exits with the Library filing system number (from &DF02) in A.

NB: None of the calls with Y=0, A<=3 are passed to the actual filing system routines.

OSFIND : If A=0 for 'close a file' the filing system is selected by the handle in Y (or the current filing system is selected if handle invalid), and the call is passed to FINDV. For A<>0 (open a file) the name pointed to by XY is searched for a temporary filing system name. Such a name must be preceded by a '-', which must be the first non space character in the name. If no temporary filing system name is found either the current filing system is selected, or the active filing system if one was selected in a *command line, eg,

```
*-DISC-DUMP TEST
```

will select DFS. If an unrecognized name is found an error is reported. Before passing the call to FINDV the pointer in XY is updated to point to after the temporary filing system name.

OSFILE : The 18 byte parameter block pointed to by XY is copied to &2ED to &2FE, this allows the original block to be resident in a Sideways ROM. The filing system is then switched in the same way as for OSFIND with A<>0, using the name pointed to by Block, Block+1. The updated name pointer is then written to the new block and FILEV called with XY = &2ED. On exit from the routine the parameter block copy with the exception of the name pointer is copied back to the original block. The routine is then exited.

FSCV : The action taken depends on the opcode passed in A, and is detailed below:

A=0 (*OPT X,Y): If a temporary filing system has already been set (eg, *-TAPE-OPT 1,2) the call is passed straight to the filing systems FSCV. Otherwise, the current filing system is selected and the call made.

A=1 (EOF #X): Before making the call the filing system is switched according to the handle in X. As is normal if the handle is illegal the current filing system is selected.

A=2 (*/): The address of the remainder of the command line is calculated and stored in &DF04/5. The filing system is then selected by name (as for OSFIND) and the call made to the filing system. There appears to be a bug in that if there are any spaces between the '/' and the filing system name, or the filing system name and the filename the pointer to the rest of the command line will point to after these, and not to after the filename.

A=3 (unrecognized command): See Library filing systems.

A=4 (*RUN): As for A=2.

A=5 (*CAT): The filing system is selected according to the text pointed to by XY, and then XY is updated to point to after the filing system name before the call to the filing system is made.

A=6 (filing system change imminent): No action, passed straight to filing system.

A=7 (Request for handle range): No action.

A=8 (OS Command about to be processed): No action.
A=9 (*EX): As for A=5.
A=10 (*INFO): As for A=5.
A=11 (Second unrecognised command): See below.

All other values for A result in a direct call to the old FSCV.

Library Filing Systems

On the Model B, when a '*' command was unclaimed by the MOS or any Service ROMs (ie, the command was not recognised by any ROM) it was passed through FSCV with A=3 and XY pointing to the rest of the command line. The filing system would then check this against its intrinsic command table, and if not found there would try to locate the file on the media and *RUN (or indirectly *EXEC) it. If the file could not be found the filing system responded with a 'Bad Command' error. For RFS and CFS which have no intrinsic commands and which are too slow to RUN a file the error was given as soon as the FSC call was received.

The system used on the Master series is extended to allow unfound commands to be passed to a second, Library, filing system. The new system works as follows.

On receiving a call with A=3 through FSCV the filing system handler sets up the address of the rest of the command line (in &DF04/5) and then calls the filing system's FSCV with A=3. The FS then behaves as before, checking its intrinsic commands followed by the current directory (CSD) and probably the library directory (CSL). However, if the command has still not been found, instead of giving an error the filing system jumps back through FSCV with A=11 with the contents of Y and X unchanged. The filing system handler on receiving this call checks to see if a Library FS is selected. If not (&DF02 < 0) then a 'Bad command' error is given. If however a Library FS has been set then this is selected and the call passed to this new filing system through its FSCV. On receiving an FSC call with A=11 a filing system should attempt to RUN the file from its CSD and CSL, and if it cannot find it should give a 'Bad Command' error.

After a hard reset or power on no library filing system is selected until a command of the form *-fs name-LIBFS is executed. All the LIBFS routine does is to copy the active filing system number from &DF01 (which will have already been set up by the CLI) to &DF02. *LIBFS

with no name will select the current filing system as the Library filing system. There is no legal way other than CTRL-BREAK of returning to the condition of no Library filing system.

Action on Service Calls

In order to keep track of what is going on the filing system handler must look at certain ROM service calls. It does this by looking at the conditions after the call has been issued. The actions are as follows:

A=&4 Unrecognized Command

If the call was not claimed no action is taken, otherwise the active filing system number is read by an OSARGS call with A=0, Y=0 direct to the filing system. This is stored in &DF01 as the active filing system. Then the active filing system is made the current filing system by copying &DF01 to &DF00. Finally &DF03 is set to the ROM number of the filing system by reading the ROM number of the extended FILEV. All this is necessary because the command may have brought about a permanent filing system change (eg, *DISC).

A=&F Indirections Changed

The value of FSCV is copied to &DFDA/B and FSCV is redirected into the filing system handler. The active filing system number is then setup as above. Finally, the active filing system is made current if no current filing system exists. This can only occur during a reset sequence and is detailed later. These actions are performed irrespective of whether the service was claimed.

A=&12 Start Filing System

The action is exactly as for A=4. NB: When a temporary filing system is started by the filing system handler a service call with A=&12 is used. However, this call is made at a lower level so that the system doesn't think a permanent change has occurred and take the actions given above.

Action on Reset

When a reset is issued (either power-on or BREAK), the following actions relevant to filing system are carried out:

- The rest of command line pointer (&DF05/6) is pointed to a carriage return in the MOS ROM.
- The filing system information table is set up by first writing in values for TAPE, CFS and ROM, and then issuing a service with A=&25 to pick up the rest.
- The top bit of &DF00 is set to signal no current filing system.
- If it is a hard reset the top bit of &DF02 is set to signal no LIBFS.
- An attempt is made to start up either the default filing system (hard reset), or the previous filing system (soft break). This is done by calling the ROM's service entry directly with A=3. If the ROM has no service entry, or does not claim the service all the ROMs are scanned. If nobody wants the call then the 1200 baud tape system is started.
- When the filing system starts up it will issue service &0F at some point which combined with the 'no current filing system' state will correctly set up active and current filing systems.

OSBYTE &8C and &8D

Because these two commands don't issue a service call with A=&12 the filing system handler can't alter the setting of the current filing system. Therefore, once these commands have started up the filing system they copy the active filing system number to the current filing system, and set the current filing system ROM to 15.

OSBYTE &6D

This call is used when it is necessary to select the active filing system as the current filing system. This might for example be necessary if an application that needs to read further files is started on a temporary filing system (eg, if a disc contains a database program which must read a data file off the same disc, and the program is started by *-DISC-DBASE from ADFS). The only action actually taken by this call is to copy the value from &DF01 to &DF00 (active filing system to current filing system) and to set &DF03 to the filing system ROM number.

Deselecting Temporary Filing Systems

Obviously when a temporary filing system has been used it is necessary to switch back to the current filing system. This is done in two ways: firstly, when a '*' command is issued the current filing system is reselected, and secondly, the filing system handler will always select the current filing system if no temporary filing system can be found.

Master Series Programming Considerations

The new features have meant that the necessary protocols for legal programming have changed. From an applications point of view it is important that filing system calls are made via page &FF, and not straight through the vectors. For a filing system to work correctly on a Master series computer the following points should be adhered to:

- The filing system must be in ROM, and correctly extend the appropriate vectors.
- Service calls 3, &12, &25 and &26 should be supported.
- The filing system should issue service call &0F once when starting up.
- The FSC entries should be properly handled, especially when A=3, 9, 10 or 11.
- The filing system must keep files open when not selected, and if possible should keep files open across break.
- The filing system must not directly alter or read any MOS variables.

These considerations are in addition to those for filing systems on a Model B.

Filing System information in this book

For further information about filing systems, see your computer User Guide, and also:

Chapter 2 :	*APPEND, *BUILD, *CAT, *CREATE, *CLOSE, *DELETE, *DUMP, *EX, *EXEC, *INFO, *LIST, *LIBFS, *LOAD, *MOVE, *OPT, *PRINT, *REMOVE, *ROM, *RUN, *SAVE, *SHUT, *SPOOL, *SPOOLON, *TAPE, *TYPE

Chapter 4 :	OSBYTE and OSWORD calls connected with filing systems.
Chapter 5 :	*SRLOAD, *SRSAVE and equivalent OSBYTE/OSWORD calls.
Chapter 6 :	Service calls to which filing systems must respond.
Chapter 7 :	Transferring data across the Tube. Necessary for all filing systems.
Chapter 9 :	Filing system use of CMOS RAM/EEPROM.
Appendix E :	Filing system numbers.

9 : Non-Volatile RAM

Introduction

The Master 128 and Compact contain a special RAM area in which details of the start-up default values for things such as screen mode are stored. These settings can be seen by issuing the command *STATUS and can be changed at anytime using the *CONFIGURE command.

There are 50 bytes allocated to the Master 128 and 128 bytes (sometimes 256 bytes) on the Compact—neither appear on the memory map and cannot be accessed using the indirection operators but can be read by using OSBYTE &A1 and changed with OSBYTE &A2 (except location 0), Chapter 4 contains full details.

On a Master 128 these bytes occupy volatile RAM and so a battery is fitted to preserve the contents of the CMOS RAM when the machine is not switched on. In the case of the Compact a special chip called an EEPROM is used. This is written and erased by using an electrical signal to semi-permanently change the chip contents. The EEPROM does have a limited life though it is not likely to expire in the machine's lifetime.

CMOS and EEPROM Memory Map

The first 40 bytes of both chips are used by the MOS and ROM firmware. Bytes 41 to 50 are availble for user applications. The memory map/byte usage by the MOS of the CMOS RAM (Master 128) and EEPROM (Compact) is detailed below:

Byte	Function
0	Econet station number
1	File server station number
2	File server network number
3	Printer server station number
4	Printer server network number

Byte	Function
5	b0-3 Default filing system ROM number
	b4-7 Default language ROM number
6	ROM frugal bits ROMs 0-7 (b0 = ROM 0), 1=Inserted
7	ROM frugal bits ROMs 8-15 (b0 = ROM 8), 0=Unplugged
8	All allocated to EDIT.
	b0-2 Screen mode (always shadow)
	000=Mode 128
	111=Mode 135
	Modes 130 and 133 not supported instead:
	010=Mode 'K'
	101=Mode 'D'
	b3 0=TAB to columns, 1=TAB below word
	b4 0=Overwrite mode, 1=Insert mode
	b5 0=Don't show returns, 1=Show carriage returns
	b6-7 Spare (for EDIT)
9	Reserved for Modem Driver
10	b0-2 Default screen mode (0-7) (111 = Mode 7)
	b3 Default shadow stat 0=Main, 1=Shadow screen
	b4 0=Interlace on, 1=Interlace off
	b5-7 *TV setting
	100=+4 to 111=-1
	000=0 to 011=3
11	b0-2 FDRIVE setting
	b3 1=Shift Caps set (Only one of these three)
	b4 1=No Caps set (should be set at any)
	b5 1=Caps Lock set (one time.)
	b6 0=No DIR loaded, 1=ADFS loads a DIR to start
	b7 0=Floppy disc default, 1=Hard disc default
12	Keyboard auto-repeat delay
13	Keyboard auto-repeat rate
14	Printer ignore character (see also 15 bit 1)
15	b0 0=No Tube, 1=Tube
	b1 0=Use ignore character, 1=No ignore character
	b2-4 Default serial baud rate (000=75 bd 111=19200 bd)
	b5-7 Default printer type (000=*FX5,0 100=*FX5,4)

Byte	Function	
16	b0	Not used
	b1	0=Loud Beep, 1=Quiet beep
	b2	0=Internal Tube, 1=External Tube
	b3	0=Scroll enabled, 1=Scroll protected
	b4	0=No Boot, 1=Auto Boot
	b5-7	Default serial format
17	b0	0=No Space, 1=Space allocated for Econet
	b1-7	Spare (for ANFS)
18	Reserved for ANFS	
19	Reserved for ANFS	
20-29	Reserved by Acorn for new firmware/filing systems	
30	Allocated to 3rd party suppliers - ROM 0	
31	Allocated to 3rd party suppliers - ROM 1	
32	Allocated to 3rd party suppliers - ROM 2	
33	Allocated to 3rd party suppliers - ROM 3	
34	Allocated to 3rd party suppliers - ROM 4	
35	Allocated to 3rd party suppliers - ROM 5	
36	Allocated to 3rd party suppliers - ROM 6	
37	Allocated to 3rd party suppliers - ROM 7 - see below	
38	Allocated to 3rd party suppliers - ROM 8	
39	Allocated to 3rd party suppliers - ROM 9	
3x	VIEW B3.0 Compact version - setup configuration	
	b0	0=No formatting, 1=Formatting
	bl	0=No justification, 1=Justification
	b2	0=Overwrite mode 1=Insert mode
	b3-7	Spare (for VIEW)
	(Byte used depends on RAM slot used—Compact default 7)	
40-49	Allocated to the user	

Compact EEPROM

Bytes 50-127 do not exist on the Master CMOS RAM. On the Compact EEPROM, they are spare bytes. Some Compact EEPROMs are 128-byte

units, others are 256 bytes in which case bytes 128-255 are available. OSBYTE &A1 with X=&FF determines the size of the EEPROM. On 256-byte EPROMs OSBYTE &A2 will not write to location 128. Locations 127 and 255 contain a signature byte to show that the EEPROM is active.

OSBYTE Revisited

For ease of reference details of OSBYTE &A1 and OSBYTE &A2 are reproduced here.

OSBYTE &A1 (161) Read CMOS RAM/EEPROM

This reads bytes from the 146818 CMOS chip on the Master 128, or the EEPROM chip on the Compact. Fifty locations are readable on the Master 128 (X=0 to X=49), and either 127 or 255 on the Compact. A special call (X=255) for the Compact only determines which if any type of EEPROM is present. Note that this call will not read the clock registers of the 146818.

Entry: X=byte to be read
0-49 on Master 128
0-127/254 on Compact
X=255 (Compact only)—Is EEPROM 128 or 256 byte?
Exit X=corrupt:Y=contents of RAM/EEPROM location
If X=255 on Compact:
Y=0 No EEPROM present
Y=&7F 128 byte EEPROM present
Y=&FF 256 byte EEPROM present

OSBYTE &A2 (162) Write CMOS RAM/EEPROM

This writes bytes to CMOS RAM or EEPROM. The address (see OSBYTE &A1 for range) is placed in X and the data in Y. Note that this call will not allow you to change byte 0 on either machine, or bytes 127, 128 or 255 on the Compact with large EEPROM. This is for Econet security reasons.

Entry: X=address on CMOS RAM/EEPROM
Y=Byte to be written.
Exit: X=preserved:Y=corrupt.

CMOS RAM Editor

Master 128 Only

Listing 9.1 provides a utility program that will allow you to alter the data in the Master CMOS RAM chip. The program accesses not only the 50 bytes of CMOS RAM data, but also the other 14 bytes of clock data as well. This is because instead of using the MOS, direct access is made to the 6522 slow data bus, to read all 64 bytes from the chip. If you change the clock data you will alter the time.

The core of the program is the subroutine 'cmos' which reads or writes data from the chip, With C=0, the routine will return in A the contents of register X. With C=1, the routine will write Y to register X. The program will not work with the Compact EEPROM which uses a different system of access. (A Compact version is provided on the Programs Disc. See Appendix K). Note also how OSBYTE &9A is used to generate a double-width cursor.

To use the program, run it, and use the cursor keys to move round the displayed data. To alter a byte, type hex digits (0-F) and the RAM will be updated. Be careful, though, there is no way of returning to the previous setting!

Listings 9.1.

```
 10 REM CMOS RAM Editor
 20 REM by D Atherton and D Spencer
 30 REM (c) Dave Atherton 1987
 40 REM for Master 128 only
 50 REM Mos : A Dabhand Guide
 60 :
 70 osbyte=&FFF4
 80 osword=&FFF1
 90 osrdch=&FFE0
100 oswrch=&FFEE
110 osnewl=&FFE7
120 osfind=&FFCE
130 osbput=&FFD4
140 control=&FE40
150 direction=&FE43
160 slowdatabus=&FE4F
170 DIM M% &300
180 ptr=&70
190 temp=&71
200 handle=&72
210 :
220 opt=2
230 FOR pass=0 TO opt STEP opt
240 P%=M%
```

```
250 :
260 [OPT pass
270 LDY #0
280 .text1
290 LDA text,Y
300 JSR oswrch
310 INY
320 CPY #(textend-text)
330 BNE text1
340 LDA #154
350 LDX #203
360 JSR osbyte
370 JSR display
380 LDA #0
390 STA ptr
400 JSR curset
410 LDA #4
420 LDX #1
430 JSR osbyte
440 .key
450 JSR osrdch
460 BCS escape
470 AND #&7F
480 CMP #12
490 BCS key1
500 CMP #8
510 BCC key1
520 \ A cursor key
530 TAY
540 LDA keystep-8,Y
550 CLC
560 ADC ptr
570 AND #&3F
580 STA ptr
590 TAX
600 JSR curset
610 .key1
620 CMP #32
630 BNE key4
640 JSR display
650 .key4
660 CMP #ASC"0"
670 BCC key3 \ Under "0"
680 CMP #ASC "9"+1
690 BCS key3 \ Over "9"
700 AND #&0F
710 JSR rotate
720 .key3
730 AND #&DF
740 CMP #ASC"A"
```

```
 750 BCC key2
 760 CMP #ASC"F"+1
 770 BCS key2
 780 SEC
 790 SBC #55
 800 JSR rotate
 810 .key2
 820 BRA key
 830 :
 840 .escape
 850 LDA #&7E
 860 JSR osbyte
 870 LDA #4
 880 LDX #0
 890 JSR osbyte
 900 LDX #0
 910 LDY #23
 920 JMP tab
 930 :
 940 .newline
 950 JSR osnewl
 960 LDA #134
 970 JSR oswrch
 980 JSR oswrch
 990 JSR oswrch
1000 JSR oswrch
1010 PHA
1020 TXA
1030 JSR hexout
1040 PLA
1050 LDA #135
1060 JSR oswrch
1070 JMP oswrch
1080 :
1090 .hexout
1100 PHA
1110 LSR A
1120 LSR A
1130 LSR A
1140 LSR A
1150 JSR hex1
1160 PLA
1170 .hex1
1180 AND #&0F
1190 CLC
1200 ADC #ASC"0"
1210 CMP #ASC"9"+1
1220 BCC digit
1230 ADC #6
1240 .digit
1250 JMP oswrch
```

```
1260 :
1270 .curset
1280 \ Value in X, Set cursor
1290 TXA \ Calculate Xpos
1300 TAY
1310 AND #7
1320 STA temp
1330 ASL A
1340 CLC
1350 ADC temp
1360 CLC
1370 ADC #8
1380 TAX \ Store Xpos in X
1390 TYA \ Original value
1400 LSR A \ DIV 8
1410 LSR A
1420 LSR A
1430 AND #7
1440 CLC
1450 ADC #6
1460 TAY \ Store Ypos in Y
1470 INX \ Coz of funny cursor
1480 JMP tab
1490 :
1500 .rotate \ enters with low nibble
1510 STA temp
1520 CLC
1530 LDX ptr
1540 JSR cmos
1550 ASL A
1560 ASL A
1570 ASL A
1580 ASL A
1590 AND #&F0
1600 CLC
1610 ADC temp
1620 PHA
1630 LDA #8
1640 JSR oswrch
1650 PLA
1660 TAY
1670 JSR hexout
1680 LDX ptr
1690 SEC
1700 JSR cmos
1710 LDA #8
1720 JMP oswrch
1730 :
1740 .tab
1750 LDA #31
1760 JSR oswrch
```

```
1770 TXA
1780 JSR oswrch
1790 TYA
1800 JMP oswrch
1810 :
1820 .display
1830 LDX #0
1840 LDY #5
1850 JSR tab
1860 LDX #0
1870 .ploop
1880 TXA
1890 AND #7
1900 BNE notst
1910 JSR newline
1920 .notst
1930 CLC
1940 JSR cmos
1950 JSR hexout
1960 LDA #32
1970 JSR oswrch
1980 INX
1990 CPX #64
2000 BNE ploop
2010 RTS
2020 :
2030 .cmos
2040 \ This routine will read (C=0) or write (C=1) to any
2050 \ register, clock and RAM in the 146818 clock chip.
2060 \ Enter with X-register number, and value returned
2070 \ in A, or for write X=reg, Y=value, C=1
2080 \
2090 \Note that this will NOT work with the Compact EEPROM
2100 \
2110 PHP
2120 SEI         \ interrupts off
2130 PHP
2140 JSR select  \ Select correct register
2150 PLP
2160 BCC read
2170 :
2180 LDA #&41    \ Take chip enable active and select write
2190 STA control
2200 LDA #&FF    \ Set. slow data bus for output 1
2210 STA direction
2220 LDA #&4A    \ Take DS active
2230 STA control
2240 STY slowdatabus     \ Write out byte :
2250 BRA exit
2260 :
2270 .read
```

```
2280 LDA #&49    \ Take chip enable active and select read
2290 STA control
2300 STZ direction \ Set slow data bus for input
2310 LDA #&4A
2320 STA control \ Take DS active
2330 LDY slowdatabus     \ Read data byte
2340 :
2350 .exit
2360 LDA #&42    \ Put DS inactive again
2370 STA control
2380 LDA #2      \ Deactivate CE
2390 STA control
2400 STZ direction \ Make sure slowdatabus is all input
2410 PLP
2420 TYA         \ Byte read in A (as well!)
2430 RTS
2440 :
2450 .select
2460 LDA #2     \ Deactivate CE and DS
2470 STA control
2480 LDA #&82    \ Take AS active
2490 STA control
2500 LDA #&FF    \ Make slow data bus output
2510 STA direction
2520 STX slowdatabus     \ Write out register address
2530 LDA #&C2    \ Take CE active
2540 STA control
2550 LDA #&42    \ Deactivate AS leaving chip selected
2560 STA control
2570 RTS
2580 :
2590 .keystep
2600 EQUB -1 \ Left
2610 EQUB  1 \ Right
2620 EQUB  8 \ Down
2630 EQUB -8 \ Up
2640 :
2650 .text
2660 EQUB 22
2670 EQUB 135
2680 EQUW &117 \ 23,1
2690 EQUD 10
2700 EQUD 0 \ Cursor off
2710 EQUB 10
2720 EQUS "  146818 RTC and CMOS RAM Editor"
2730 EQUB 31
2740 EQUB 0
2750 EQUB 5
2760 .textend
2770 .fname
2780 EQUD 0
```

```
2790 EQUD 0
2800 .osblk
2810 EQUD 0
2820 EQUD 0
2830 ]
2840 NEXT
2850 CALL M%
```

Listing 9.1. The CMOS RAM Editor.

10 : Differences

Introduction

In this chapter, the differences between the various models of BBC Microcomputer are detailed. These are all researched from existing documentation, but this has never before have been brought together in this way. This list will help you make your software run on all machines, and also, to see why existing commercial software will not run on later models. The items covered include software and hardware as follows:

BBC BASIC 1, 2, 3, 4, 40, and 5.
Model B to Model B+.
Model B/B+ to Master 128.
Master 128 to Master Compact.
Master 128 MOS 3.20 to MOS 3.21.
ADFS 1.50 to ADFS 2.10.
DFS 0.90 to DFS 2.42 inclusive.
Master EDIT 1.00 to Master EDIT 1.16.

BBC BASIC

This is obviously a major feature of all BBC Micros. When the language starts up, the error printing routine points to the copyright string rather than an error message, this contains the year of release. You can check this by typing *BASIC, then REPORT. This is how versions are identified. This section is concerned with the differences between the various versions of BASIC.

BBC BASIC 1 - (C) 1981 Acorn

This is the original Model B BASIC and all published software should work on this, the oldest BASIC. The rest of this section covers the changes from BASIC 1.

For a full definition of BASIC as a language, see the original Model B User Guide.

BBC BASIC 2—(C) 1982 Acorn

This was released as an optional replacement for Model B owners, and installed in new Model B's from about 1983. This is the standard and only version of BASIC in Model B+ and Electron machines. Here are the differences from BASIC 1:

1. ELSE no longer leaves a byte on the hardware stack with ON..GOTO.
2. INSTR no longer leaves the shorter string on the software stack.
3. EVAL now works completely, ie, EVAL("TIME").
4. PRINT -ABS 1 now works correctly.
5. New keyword, OSCLI, introduced using token &FF.
6. OPENIN changes function, a file is opened for read only. It is assigned a new token, &8E.
7. SIN, COS, LN and LOG re-coded for accuracy. LN(2E-39) bug fixed.
8. A new keyword OPENUP takes old OPENIN function and token.
9. Number printing now has 10 figures of precision.
10. @% now defaults to &90A instead of &A0A.
11. Changing MODE now sets COUNT to zero.
12. INPUT "Prompt";X now accepted. ';' is new and works like ','
13. Fatal errors introduced. If an errors ERR is 0, then ON ERROR is ignored. STOP and 'No room' are fatal errors.
14. New error (ERR=45) 'Missing #' arises when BGET and BPUT are used without a #.
15. DIM space X where X < -1 gives an error 'Bad DIM'.
16. The assembler now handles ASC":" correctly.
17. The assembler now does displaced assembly controlled by O% and OPT 4-7.
18. The assembler now contains pseudo-ops EQUB, EQUS, EQUD, and EQUW.
19. There is now some garbage collection. This can be demonstrated by:

```
A$="":FOR I%=0 TO 250:A$=A$+"X":NEXT
```

    which is much more efficient. The situation is still imperfect if you alternate two strings in this process.

BBC BASIC 3 - (C) 1983 Acorn

Not often seen, this is the version of BASIC for the US version of the BBC Micro. A few minor changes were introduced over BASIC 2. The

HI-BASIC distributed with the 6502 second processor is a relocated version of BASIC 3. The Changes:

1. The interpreter accepts COLOR for COLOUR. The detokeniser (LIST) spells the word as COLOUR. An American version of the BASIC is available which spells the word as COLOR. This is the only difference between the two versions.
2. SAVE N$+X$ now works.
3. The use of ? and ! as formal parameters works correctly.
4. Random number generator re-coded for speed.

BBC BASIC 4 - (C) 1984 Acorn

This is the version of BASIC supplied on the Master 128. Differences from the previous versions are:

1. Trailing spaces are stripped from lines entered in BASIC.
2. Leading spaces between line number and line text are now discarded, if the LISTO option is 1 or greater.
3. LISTO now indents loops correctly
4. LIST IF function now available. The syntax is:

   ```
   LIST <<linenr,linenr>> IF <text>
   ```

   for example,

   ```
   LISTIF label
   LIST 200,300 IF label
   ```

   Anything may be searched for. Tokens are always treated, so with the program:

   ```
   10 PRINT "TEXT"
   20 TEXT$="PRINT"
   ```

   typing LISTIF PRINT or LIST IF P. will list line 10 only. The way to find strings which are also tokens is to search for less than the token string. In the example LISTIF PRIN only line 20 will be listed. LISTIF cannot find TIME, HIMEM, PAGE, LOMEM and PTR# if they are operators, eg, LOMEM=&4000, (EXT#n=will be found). It can however find them as operands, eg, X=LOMEM.
5. RENUMBER and LIST are not now confused by byte &8D in comments and listings.
6. LIST will now print colour codes in REM statements which are not surrounded by quotes.

7. EXT#n=x now works to alter the size of a file. It uses OSARGS A=3:Y=handle, thus the command will not work with filing systems that don't support this.
8. A pseudo-variable, TIME$, returns a 24-byte string identical to that returned by *TIME. It can be assigned with either date or time strings or both. See OSWORD &0F for details.
9. AUTO no longer prints a space after the line number.
10. General recursion is now allowed in FOR loops.
11. A new command EDIT, which has the same syntax as LIST, including the IF options, passes sections or all of the program to the 'Edit' language. BASIC issues a command *EDIT mm,nn where mm and nn are hex numbers. The numbers are zero page hex addresses which contain the start addresses and end addresses of the text.
12. A VDU list can be terminated with '|' which translates into nine VDU 0's, are enough to complete any VDU command.
    For example:

    ```
    VDU 19,0,7;0;0;0;19,1;0;0;0;
    ```

    can now be expressed as:

    ```
    VDU 19,0,7| 19,1|
    ```

13. Random number generator changed again, for RND(1) and RND(>1).
14. ON n PROCa,PROCb,PROCc,PROCd ELSE PROCe:PROCf now works.
15. RESTORE <lnr> where <lnr> is a line without DATA on it, but with a comma is now not treated as a line of data.
16. Some formatting of assembly listings is now provided.
17. The assembler incorporates all 65C02/65C12 opcodes. INC A can be represented as INA. DEC A can be represented as DEA. STZ can be represented as CLR. This is for compatibility with MASM, Acorn's assembler in their 6502 Development System.
18. ASL ADDREG is now coded correctly, formerly coded as ASL A \DDREG. This applies to other mnemonics which work with the accumulator, ie, ASL, LSR, ROL, ROR, DEC, INC.
19. X, Y and A register references in assembler may now be in lower case, ie, x, y and a.
20. EQUB, EQUS, EQUD, EQUW may also be in lower case.

BBC BASIC 40 — (C) 1986 Acorn

This version of BASIC is fitted in the Master Compact.

1. The floating point code is totally recoded for improved speed and accuracy.
2. MOD bug now fixed in BASIC 41, the 2nd version of BASIC 40

BBC BASIC 5 - (C) 1987 Acorn

This BASIC has only been coded so far on the ARM computer, Acorn's house-designed processor unit, as used in the Acorn Archimedes. This is only a very brief run down of changes.

1. Introduction of WHILE .. ENDWHILE structure.
2. Introduction of CASE, WHEN ... OTHERWISE, ENDCASE structure.
3. Block-structured IF, like PASCAL.
4. Local errors, where ON ERROR does not wipe BASIC stack out.
5. Single step and enhanced tracing.
6. Improved PRINT accuracy, including non-scientific printing of small numbers.
7. Function to return size of arrays PRINT DIM(A())
8. Passing arrays to procedures and functions (long overdue!).
9. Many new statements and extensions.
10. New operators <<n (32 bit ASL n times), >>n (32 bit ASR n times), >>>n (32 bit LSR n times).
11. % is binary constant ie, Q%=%10101010.
12. | is address of floating point variable ie, z=|x.
13. "+=" and "-=" offer arithmetic at assignment. X-=1 is the same as X=X-1.
14. Many new commands such as HELP, LVAR etc.
15. LISTO greatly enhanced.
16. Integral ARM assembler.
17. Assembler has different convention for labels.
18. String storage completely rewritten.
19. Seven new error codes.
20. Ignores @% for reporting line numbers.
21. FOR statement recoded to avoid overflow.
22. Entry-time syntax checking.

BBC Machines

The changes between successive releases of BBC micros in hardware and software terms are detailed below. Details of changes from BASIC versions can be found above.

Changes: Model B to Model B+

The BBC Micro Model B+, also known as the 'Issue 10 board' was a small modification, even though the main PCB underwent a major redesign mainly to take advantage of new cheaper components. The end result is a machine that is broadly similar, however it did start the issue of compatibility across a range of machines. These are the differences:

Hardware

1. Additional 20k of 'shadow' screen memory, mapped at &3000-&7FFF.
2. Additional 12k of 'sideways' memory mapped at &8000-&AFFF.
3. 1770 disc interface replaces 8271 interface.
4. Bit 7 &FE30 controls 12k section (1=section active).
5. Bit 7 &FE34 controls 20k screen (1=shadow active).

Software

1. 1770 DFS is standard, replacing optional 8271 DFS.
2. Extra commands for DFS in 1770 version *FORMAT, *VERIFY, *FREE, *MAP.
3. Extension to VDU 22 to select Modes 128-135 (Shadow modes).
4. *SHADOW command. See Chapter 2 for details.
5. New OSBYTE &72, identical to *SHADOW.
6. VDU status byte (&D0) bit 4 contains shadow state (1=shadow).
7. OSBYTE &84, &85 always return &8000 if shadow active.
8. OSBYTE &87 does NOT return shadow mode number, just 0-7.
9. New OSBYTE &EF reads shadow state.
10. New system call OSWRSC at &FFB3.
11. OSWORD 5 and 6 can now be used to read shadow memory.

Changes: Models B/B+ to Master 128

The differences here are huge. Most of the book describes these, but here is a brief, but nevertheless, complete list.

Hardware

1. Shadow screen as on B+.
2. Real time clock/CMOS RAM.
3. Numeric keypad fitted.
4. Extra 12k workspace RAM for MOS.
5. Four Sideways RAM banks fitted as standard.
6. Extra latches at b7 &FE30 and &FE34. See Chapter 5.
7. Cartridge sockets fitted, with internal 1MHz bus which is actually 2MHz.
8. Internal Modem port.
9. Internal Tube connector.
10. 1770 disc interface replaces 8271.

Software

1. CLI environment supported—accessed by *GO.
2. Permanent machine settings, MODE, printer type etc. in CMOS RAM. *CONFIGURE and *STATUS commands control this.
3. BASIC 4 supplied as standard.
4. Cassette Filing System now performs OSGBPB calls 1 & 3, OSFSC 7, *EX (but not *INFO), and this responds to service call &12.
5. ROM filing system responds to service call &12.
6. Temporary and library filing systems implemented.
7. Greatly extended graphics commands, similar to Acornsoft GXR - Graphics eXtension ROM.
8. Extended text printing facilities.
9. Characters 128-255 defined in ROM.
10. Many new '*' commands, see Chapter 2 for details.
11. Some new and extended MOS calls especially in the filing system area.
12. Extended Sideways ROM service calls.

Changes: Master 128 to Master Compact

The Master Compact is very similar to the Master 128. In general most software written for the 128 will work on the Compact, and most modifications to Model B software needed for the 128 are also needed for the Compact. The main differences are improvements made to the MOS and ADFS in the nine months between release of the Master 128 (January 1986) and the Master Compact (September 1986).

Hardware

1. Standard issue 3.5" double-sided 80 track discs.
2. Optional supplied colour and mono monitors.
3. Missing items: Real Time Clock, Cassette filing system, 1MHz bus, Cartridge Ports, Tube interface, A-D converter, Colour Video, RS423 port, Internal Modem Slot.
4. EEPROM replaces CMOS RAM, bytes increased from 50 to 128 or 256
5. @ character now obtained on shift-0, @ key now becomes a 'code' key.

Software

1. OSWORDs &0E and &0F and *TIME and BASIC's TIME$ return a fixed date/time "Fri,31 Dec 1999.23:59:59", unless ANFS fitted when OSWORD &14 is issued, see Chapter 4.
2. Configuration system lists items in alphabetical order.
3. New configure options SWITCHED, PROPORTIONAL, STICK for joystick.
4. New OSBYTE call (&A1) to read EEPROM size.
5. New A-D code emulation, see Appendix H.
6. Extra cursor key option set by *FX4,3.
7. Code to support non-existent hardware such as Tube, 1MHz bus and Cassette removed.
8. SRAM code and graphics ellipse code now in the MOS area.
9. Bug in long thin ellipses fixed.
10. MOS sets sensible defaults in the absence of an EEPROM.
11. 'I' option added to the *SRLOAD command.
12. *BUILD and *APPEND allow entry of top bit set characters.
13. Code key system. Press CTRL/SHIFT/CODE all together, then release. The next key pressed will have 128 added to its ASCII value.
14. The first JSR BREAK call in the MOS to allow break indirection has been changed to preserve ROMID (location &F4).

15. INKEY-256 now returns &F5. *FX0,1 now returns 5. *FX0,0 now prints MOS instead of OS.
16. Key interpretations of OSBYTEs &DD to &E4 inclusive have changed for X=2. This value now means return a NUL prior to a predefined code. See Chapter 4 for full details.
17. User printer routines and extension vectored routines may now page in HAZEL safely.
18. *ROMS now gives the message RAM instead of ROM for RAM slots.
19. *TAPE and *MOTOR commands have no effect.
20. -CFS- and -TAPE- filing system names are not supported.
21. DFS is not present in ROM, and thus neither are its commands, it is however supplied on disc, in later releases of the Master Compact *Welcome Disc*.
22. BASIC is internally altered, and improved. The new version is known as BASIC &40.
23. *ROMS now ignore duplicate ROMs. Second image (counting from &F) is displayed as '?' and &01 is placed in the table at &2A1. Unplugged ROMs are still tested for duplication.
24. *REMOVE now fails if given two filenames, to avoid confusion with *RENAME)

MOS, DFS and ADFS

This section details the changes made in various releases of the MOS, DFS and ADFS.

Changes: Master 128 MOS 3.20 to MOS 3.21

1. *CONFIGURE and *STATUS recoded. They now list in alphabetical order.
2. Reset (ie, startup) code has now been placed in the three pages of ROM mapped at &FC00 to &FEFF. Reset now includes reads to the 1770 data and status registers (to clear spurious NMIs) and the Sideways RAM is cleared.
3. *BASIC routes through OSBYTE &8E.
4. Bugs concerned with CFS OSGBPB over the Tube, and long thin ellipses fixed.
5. All the appropriate Compact features are included.

Changes: ADFS 1.50 (M128) to ADFS 2.10 (Compact)

1. *DRIVE is implemented for software compatibility.
2. *COPY and *COMPACT use shadow screen if available, and will not then corrupt user workspace. If possible select a non-shadow mode before using these commands. If shadow screen is not available *COMPACT uses two pages of utilty workspace, and one page of CLI buffer, and *COPY forces Mode 135, both to avoid corrupting user data.
3. *FORMAT, *VERIFY, and *BACKUP are implemented.
4. OSGBPB calls A=6 and A=7 return a zero byte after the CSD or CSL name to be compatible with the ownership byte returned by NFS/ANFS.
5. CLOSE#0 had a bug causing a 'Channel' error - now fixed.
6. General speed improvements.
7. Winchester driver code removed.
8. Skew value is now 4 (from 7 on old formatter).

Changes: DFS 0.90 to DFS 1.20 on 8271 (BBC B)

1. Many bugs removed, most notably when files were open on two surfaces, it was easy to write back the catalogue to the wrong surface
2. Network Filing System included in the ROM.
3. *BACKUP, *DESTROY and *ENABLE work differently. If *ENABLE is used, then the commands work as before. If not, then the command does not terminate, but offers a "Go (Y/N)?" prompt.

Changes: DFS 1.20 to DFS 2.10 on 1770 (B+/M)

1. Controller code altered to drive BBC B+ 1770 controller.
2. DRIVE now takes a second parameter to allow, or cancel, double-stepping, which allows you to read (but not write) 40-track discs on 80-track drive. *DRIVE n 40 will set Drive n to read 40-track discs - *DRIVE n 80 will cancel the facility, and allow reading of 80-track discs again. There is no restriction on mixing tracks on different sides of the same disc.
3. The keyboard link settings are now:

Step Time	6ms	12ms	20ms	30ms(default)
Link3	0	1	0	1
Link4	0	0	1	1

These can be set from the keyboard, if a DIL switch is fitted, or by using OSBYTE &FF from the keyboard or in software.

4. The 'Drive fault' error (&C5) has been deleted.
5. The following new commands are included in the ROM: *CLOSE, *EX, *FORM, *FREE, *MAP, *ROMS, *VERIFY. See Chapter 2 for details of these commands.

1770 DFS Versions

Several versions of this DFS have been released each with minor changes. Version number and changes to it from the previous list are as follows:

2.00 Early Model B/B+ versions (FDC at &FE84-&87)
2.10 First official B/B+ release.
Includes SRAM commands, which cause *CLOSE bug.
Only version with double density (at OSWORD level).
2.20 First Master version (FDC at &FE28-&FE2B).
2.24 *HELP SRAM sends only one linefeed after each line.
OSFILE &FF (*LOAD) returns A=1 if found.
*SAVE longer than 64k works.
2.25 *CLOSE bug fixed.
EXT#X=number now works.
LIBFS deals with '*' commands correctly.
Now responds to ROM service calls &25 and &26.
2.26 All four head speeds selectable (FDRIVE).
OSGBPB recoded for speed.
Last BBC B/B+ version.
2.28 Improvement in reset characteristics.
Pressing BREAK, after changing discs while files open, now correctly causes 'Disc changed'.
2.29 OSGBPB Tube problem fixed (only appeared in 2.26).
2.40 SRAM code is now in the MOS.
OSWORD &7F with A=&E0/&F0 (1770 read/write track) deleted.
Only one private page claimed on service &22.
Some ADFS code now in end of ROM 9.
Extra 1770 reads at reset to clear spurious NMIs.

EDIT

EDIT is supplied as part of the Mega-bit ROM on the Master 128.

Changes: Master EDIT 1.00 to HI-EDIT 1.16

1. Translated enhanced characters in formatter work correctly.
2. Display update works correctly.
3. Status display shows TAB state.
4. New mode GETs instead of INPUTs.
5. Shift-f5 'D' message amended.
6. New file cannot be loaded without confirmation, unless old file is unmodified.
7. Carriage returns now displayed as inverse $, not inverse M.
8. Wipe to end-of-line uses clear text block instead of clear window. This no longer affects the cursor position in cursor editing mode.
9. Insert file no longer changes the current filename.
10. "*EDI.FRED" & "*EDIT FRED" now allowed. Previously exactly one space was needed before filename.
11. Version number displayed on descriptive page is now the same as that in the ROM header.
12. Binary version number is now 2 and copyright year is 1986.

Appendices

A:	Complete OSBYTE List	185
B:	Complete OSWORD List	191
C:	Complete VDU List	193
D:	Memory Map	198
E:	OS Call Item List	213
F:	Key Numbers	217
G:	PCB Links	223
H:	Cartridge Ports	230
I:	Compact Analogue Emulator	234
J:	Connector Pinouts	241
K:	The Programs Disc	245
L:	Guide to Dabhand Guides	247

A : Complete OSBYTE List

These calls perform a variety of MOS functions. They are used by setting the accumulator to the relevant number, and then calling &FFF4. X and Y are loaded with any parameters required. Calls are marked to show which machines they are implemented on. The numbers in brackets refer to their *FX equivalent. Chapter 4 contains full details on all the new commands.

Number		Action	Machines
&00	(0)	Print Operating System version	All
&01	(1)	User OSBYTE call, read/write &281	All
&02	(2)	Select input stream	All
&03	(3)	Select output stream	All
&04	(4)	Select cursor state	All
&05	(5)	Select printer destination	All
&06	(6)	Set character ignored by printer	All
&07	(7)	Set RS423 receive baud rate	All
&08	(8)	Set RS423 transmit baud rate	All
&09	(9)	Set flash duration mark state	All
&0A	(10)	Set flash duration space state	All
&0B	(11)	Set keyboard auto-repeat delay	All
&0C	(12)	Set keyboard auto-repeat rate	All
&0D	(13)	Disable events	All
&0E	(14)	Enable events	All
&0F	(15)	Flush buffers	All
&10	(16)	Select ADC channels to be sampled	All
&11	(17)	Force an ADC conversion	All
&12	(18)	Clear function key definitions	All
&13	(19)	Wait for vertical sync	All
&14	(20)	Explode soft character RAM allocation	B/B+/E
		Restore default character set	M/C
&15	(21)	Flush specific buffer	All
&16	(22)	Increment polling flag	E/M/C
&17	(23)	Decrement polling flag	E/M/C
&18	(24)	Select external sound	E

&19	(25)	Restore a group of font definitions	M/C
&1A	(26)	26-67 unused	
&44	(68)	Test sideways RAM presence	B+/M/C
&45	(69)	Test pseudo/absolute SWR usage	B+/M/C
&46	(70)	70-106 unused	
&6B	(107)	Select internal/external 1MHz bus	M/C
&6C	(108)	Write usage of main/shadow memory	M/C
&6D	(109)	Make temporary filing system permanent	M/C
&6E	(110)	Unused	
&6F	(111)	Unused - Aries B20 board	B
&70	(112)	Select memory for VDU code	M/C
&71	(113)	Select memory for display	M/C
&72	(114)	Write usage of shadow memory	B+/M/C
&73	(115)	Blank or restore palette	E
&74	(116)	Reset internal sound system	E
&75	(117)	Read VDU status	All
&76	(118)	Reflect keyboard status in LEDs	All
&77	(119)	Close any SPOOL/EXEC files	All
&78	(120)	Write current keys pressed information	All
&79	(121)	Perform keyboard scan	All
&7A	(122)	Perform keyboard scan from &10	All
&7B	(123)	Printer driver going dormant	All
&7C	(124)	Clear ESCAPE condition	All
&7D	(125)	Set ESCAPE condition	All
&7E	(126)	Acknowledge detection of ESCAPE	All
&7F	(127)	Check for EOF on open file	All
&80	(128)	Read ADC channel/Get buffer status	All
&81	(129)	Read key with time limit	All
&82	(130)	Read machine high order address	All
&83	(131)	Read OSHWM (start of user RAM)	All
&84	(132)	Read HIMEM (end of user RAM)	All
&85	(133)	Read HIMEM for a given mode	All
&86	(134)	Read cursor position (POS and VPOS)	All
&87	(135)	Read character at input cursor position	All
&88	(136)	Perform *CODE	All

&89	(137)	Perform *MOTOR	B/B+/E/M
&8A	(138)	Insert value into buffer	All
&8B	(139)	Perform *OPT	All
&8C	(140)	Perform *TAPE	B/B+/E/M
&8D	(141)	Perform *ROM	All
&8E	(142)	Enter language ROM	All
&8F	(143)	Issue paged ROM service request	All
&90	(144)	Perform *TV	All
&91	(145)	Get character from buffer	All
&92	(146)	Read from FRED, 1MHz bus	All
&93	(147)	Write to FRED, 1MHz bus	All
&94	(148)	Read from JIM, 1MHz bus	All
&95	(149)	Write to JIM, 1MHz bus	All
&96	(150)	Read from SHEILA, mapped I/O	All
&97	(151)	Write to SHEILA, mapped I/O	All
&98	(152)	Examine buffer status	All
&99	(153)	Insert character into input buffer	All
&9A	(154)	Write to video ULA control register and copy	All
&9B	(155)	Write to video ULA palette register and copy	All
&9C	(156)	Read/Write (R/W) 6850 control reg. and copy	All
&9D	(157)	“Fast” Tube BPUT	All
&9E	(158)	Read from speech processor	B/B+
&9F	(159)	Write to speech processor	B/B+
&A0	(160)	Read VDU variable base address	All
&A1	(161)	Read CMOS RAM/EEPROM	M/C
&A2	(162)	Write CMOS RAM/EEPROM	M/C
&A3	(163)	Reserved for third parties	
&A4	(164)	Check if data is suitable for I/O proc	M/C
&A5	(165)	Read output cursor position	M/C
&A6	(166)	Get start address of OS variables (lo)	All
&A7	(167)	Get start address of OS variables (hi)	All
&A8	(168)	Get address of ROM pointer table (lo)	All
&A9	(169)	Get address of ROM pointer table (hi)	All
&AA	(170)	Get address of ROM information table (lo)	All
&AB	(171)	Get address of ROM information table (hi)	All

&AC (172)	Get address of key translation table (lo)	All
&AD (173)	Get address of key translation table (hi)	All
&AE (174)	Get start address of VDU variables (lo)	All
&AF (175)	Get start address of VDU variables (hi)	All
&B0 (176)	R/W CFS timeout counter	B/B+/E/M
&B1 (177)	R/W input source	All
&B2 (178)	R/W keyboard semaphore	All
&B3 (179)	R/W primary OSHWM	B/B+/E
	Read keyboard semaphore	M/C
&B4 (180)	R/W current OSHWM	All
&B5 (181)	R/W RS423 mode	All
&B6 (182)	Read character definition explosion state	All
&B7 (183)	R/W cassette/ROM filing system switch	All
&B8 (184)	Read RAM copy of video ULA control register	All
&B9 (185)	Read RAM copy of video ULA palette register	All
&BA (186)	R/W ROM number active at last BRK (error)	All
&BB (187)	R/W number of ROM socket containing BASIC	All
&BC (188)	Read current ADC channel	All
&BD (189)	R/W maximum ADC channel number	All
&BE (190)	Read ADC conversion type	B/B+/E/M
	Write stick sensitivity	C
&BF (191)	R/W RS423 use flag	All
&C0 (192)	Read RS423 control flag	All
&C1 (193)	R/W flash counter	All
&C2 (194)	R/W mark period count	All
&C3 (195)	R/W space period count	All
&C4 (196)	R/W keyboard auto-repeat delay	All
&C5 (197)	R/W keyboard auto-repeat period	All
&C6 (198)	R/W *EXEC file handle	All
&C7 (199)	R/W *SPOOL file handle	All
&C8 (200)	R/W ESCAPE, BREAK effect	All
&C9 (201)	R/W Econet keyboard disable	All
&CA (202)	R/W keyboard status byte	All
&CB (203)	R/W RS423 handshake extent	All
&CC (204)	R/W RS423 input suppression flag	All

&CD (205)	R/W cassette/RS423 selection flag	B/B+/E/M
&CE (206)	R/W Econet OS call interception status	All
&CF (207)	R/W Econet OSRDCH interception status	All
&D0 (208)	R/W Econet OSWRCH interception status	All
&D1 (209)	R/W speech suppression status	B/B+
&D2 (210)	R/W sound suppression status	All
&D3 (211)	R/W BELL channel	All
&D4 (212)	R/W BELL envelope number/amplitude	All
&D5 (213)	R/W BELL frequency	All
&D6 (214)	R/W BELL duration	All
&D7 (215)	R/W startup message and !BOOT options	All
&D8 (216)	R/W length of soft key string	All
&D9 (217)	R/W number of lines printed since last page	All
&DA (218)	R/W number of items in VDU queue	All
&DB (219)	R/W TAB character value	All
&DC (220)	R/W ESCAPE character value	All
&DD (221)	R/W character &C0 to &CF status	All
&DE (222)	R/W character &D0 to &DF status	All
&DF (223)	R/W character &E0 to &EF status	All
&E0 (224)	R/W character &F0 to &FF status	All
&E1 (225)	R/W function key status	All
&E2 (226)	R/W SHIFT+function key status	All
&E3 (227)	R/W CTRL+function key status	All
&E4 (228)	R/W CTRL+SHIFT+function key status	All
&E5 (229)	R/W ESCAPE key status	All
&E6 (230)	R/W flags determining ESCAPE effects	All
&E7 (231)	R/W IRQ bit mask for user 6522	All
&E8 (232)	R/W IRQ bit mask for 6850	All
&E9 (233)	R/W IRQ bit mask for system 6522	All
&EA (234)	Read Tube flag	All
&EB (235)	Test for speech processor	B/B+
&EC (236)	R/W write output status as set by *FX3	All
&ED (237)	R/W cursor status as set by *FX4	All
&EE (238)	Set base for numeric keypad	M/C
&EF (239)	R/W shadow state (*FX114 setting)	B+/M/C

&F0	(240)	Read country flag (UK or USA MOS)	E/M/C
&F1	(241)	R/W location &281 (*FX1 setting)	All
&F2	(242)	Read RAM copy of serial processor ULA	All
&F3	(243)	R/W timer switch state	All
&F4	(244)	R/W soft key consistency flag	All
&F5	(245)	R/W *FX5 (printer) setting	All
&F6	(246)	R/W *FX6 (ignore) setting	All
&F7	(247)	R/W 1st byte of BREAK intercept code	All
&F8	(248)	R/W 2nd byte of BREAK intercept code	All
&F9	(249)	R/W 3rd byte of BREAK intercept code	All
&FA	(250)	Read *FX112 setting	M/C
&FB	(251)	Read *FX113 setting	M/C
&FC	(252)	R/W current language ROM number	All
&FD	(253)	R/W last BREAK type	All
&FE	(254)	R/W available RAM (16 or 32K)	B/B+/E
		Enable/disable SHIFT on numeric pad	M/C
&FF	(255)	R/W start up options	All

B : Complete OSWORD List

OSWORD is a MOS entry point at &FFF1. All calls are made by setting X (lo) and Y (hi) to point to a parameter block in memory, and setting the accumulator to the call number. The parameter block is set up first, and then a call to &FFF1 is made. Chapter 4 contains more details.

&00	(0)	Read an input line to memory	All
&01	(1)	Read system clock (as in BASIC TIME)	All
&02	(2)	Write system clock	All
&03	(3)	Read interval timer	All
&04	(4)	Write interval timer	All
&05	(5)	Read byte of I/O processor memory	All
&06	(6)	Write byte to I/O processor memory	All
&07	(7)	Generate a sound (as in BASIC SOUND)	All
&08	(8)	Define an envelope (BASIC ENVELOPE)	All
&09	(9)	Read pixel colour (as in BASIC POINT)	All
&0A	(10)	Read a character definition	All
&0B	(11)	Read the VDU palette	All
&0C	(12)	Write the VDU palette	All
&0D	(13)	Read current/previous graphics cursor position	All
&0E	(14)	Read CMOS clock (3 possible calls)	M/C
&0F	(15)	Write CMOS clock (3 possible calls)	M/C
&10	(16)	Econet transmit	NFS/ANFS
&11	(17)	Econet receive	NFS/ANFS
&12	(18)	Econet reading of argument block	NFS/ANFS
&13	(19)	Read/write station information	NFS/ANFS
&14	(20)	Communicate with fileserver etc.	NFS/ANFS
&28	(40)	Prisma Graphics processor	3P ROM
&29	(41)	Prisma Graphics processor	3P ROM
&2A	(42)	Prisma Graphics processor	3P ROM
&2B	(43)	Prisma Graphics processor	3P ROM
&2C	(44)	Prisma Graphics processor	3P ROM
&2D	(45)	Prisma Graphics processor	3P ROM
&2E	(46)	Prisma Graphics processor	3P ROM

&2F	(47)	Prisma Graphics processor	3P ROM
&30	(48)	Prisma Graphics processor	3P ROM
&31	(49)	Prisma Graphics processor	3P ROM
&40	(64)	AMX Mouse/Master Trackerball	3P ROM
&41	(65)	AMX Mouse/Master Trackerball	3P ROM
&42	(66)	Memory transfer to/from SRAM	B+/M/C
&43	(67)	Load/save to Sideways RAM	B+/M/C
&44	(68)	AMX Mouse support	3P ROM
&45	(69)	Aries B32 move/swap memory	3P ROM
&46	(70)	Allocated to BBC Soft	3P ROM
&47	(71)	Available for third parties, to …	3P ROM
&5E	(94)	… Available for third parties	3P ROM
&5F	(95)	BBC Soft 'Monitor'	3P ROM
&60	(96)	Read Master Sequence number and status byte	VFS
&62	(98)	Access the LVROM disc controller	VFS
&63	(99)	Read last VFS error information	VFS
&64	(100)	Read current F-code	VFS
&70	(112)	Read Master sequence no and status byte	ADFS
&71	(113)	Read free space	ADFS
&72	(114)	Read/write sectors	ADFS
&73	(115)	Read last error information	ADFS
&7A	(122)	Teletext commands	3P ROM
&7B	(123)	Modem commands	3P ROM
&7D	(125)	Read Master Sequence number	DFS
&7E	(126)	Read disc size	DFS
&7F	(127)	Issue command to 8271 controller	DFS
&80	(128)	Miscellaneous IEEE commands	IEEEFS
&82	(130)	Cambridge Ring r/w parameters	C.Ring
&83	(131)	Cambridge Ring data transmission	C.Ring
&84	(132)	Cambridge Ring ring polling	C.Ring
&90	(144)	Network Service Interface	Network
&A0	(160)	Isolated word recogniser	
&FE	(254)	Z80 SP Disc Read	Z80 Host
&FF	(255)	Z80 Data Transfer	Z80 Host

C : Complete VDU Command List

The BBC Micro VDU drivers control not only text and graphics output, but also the alteration of all text and graphics attributes (except video ULA direct reprogramming). For example, a screen mode may be selected, colours defined, a fill pattern determined, and then text and flood filled graphics may be produced without any other system calls. To drive the VDU all you do is load the accumulator with a value, and perform a JSR to &FFEE. The control codes have accumulator values between 0 and 31, and some take additional parameters so if a call with A=129 followed a call with A=22 it would mean that screen mode 129 was required, not that ASCII character 129 is to be printed. The number of parameters required is indicated, as are Master-only commands. Note that all graphics plotting is done by adding parameters to VDU 25, and that many of the more esoteric commands are done with VDU 23.

VDU	Action performed	No of Parameters	Machine
0	Do nothing	0	All
1	Send next character to printer	1	All
2	Send subsequent codes to printer also (8-13 and 32-126 only, unless done with VDU 1)	0	All
3	Stop all printer output	0	All
4	Disable 'text at graphics cursor'	0	All
5	Enable 'text at graphics cursor'	0	All
6	Cancel any driver disability from VDU 21	0	All
7	Sound a beep	0	All
8	Move cursor back one space	0	All
9	Move cursor forward one space	0	All
10	Move cursor down one line	0	All
11	Move cursor up one line	0	All
12	Clear screen	0	All
13	Carriage return	0	All
14	Enable 'paged mode' output	0	All
15	Disable 'paged mode' output	0	All
16	Clear graphics window (CLG)	0	All
17	Define text colour (COLOUR)	1	All
18	Define graphics colour (GCOL)	2	All

19	Define logical colour	5	All
20	Restore default settings of VDU 17 / 19	0	All
21	Disable VDU drivers	0	All
22	Select screen mode	1	All
23	Various functions - see below	9	All
24	Set graphics window	8	All
25	Various graphics plotting commands - see below	5	All
26	Restore default text and graphic windows	0	All
27	Do nothing	0	All
28	Define text window	4	All
29	Set graphics origin	4	All
30	Home cursor	0	All
31	Move cursor to X,Y	2	All
32	ASCII printable characters SPACE to ...	0	All
126	ASCII tilde ~	0	All
127	Delete last character	0	All

VDU 23

This performs several functions depending on the second parameter. If the second parameter is above 31, the command simply defines the bit pattern for the ASCII character denoted by the second parameter. Otherwise, the command controls aspects of text display. The table uses the BBC BASIC 4 convention of terminating a VDU sequence which needs a number of zeros with '|' which represents '9 zeros'. The command always requires 9 parameters, so if programming in assembler, or in earlier versions of BASIC, follow the command with enough zeros to make 9 altogether. Note that BBC BASIC 4 actually sends 9 ASCII zeros in all events, as the extra ones have no effect, and you might like to write a subroutine to do just this, to avoid having to count the required number.

VDU	**Action performed**	**Machine**
23,0	Write to 6845 VDU 23,0,reg,value|	All
23,1	Cursor state VDU 23,1| = off, VDU 23,1,1| = on VDU 23,2| = steady, VDU 23,3| = slow	All
23,2	Set Extended Colour Fill pattern 1	M/C
23,3	Set Extended Colour Fill pattern 2	M/C
23,4	Set Extended Colour Fill pattern 3	M/C

23,5	Set Extended Colour Fill pattern 4	M/C
23,6	Set dotted line mark/space ratio	M/C
23,7	Scroll a text window	M/C
23,8	Clear a text window	M/C
23,9	Set n $^{1}/_{50}$s flash time colour 1: VDU 23,9,n|	M/C
23,10	Set n $^{1}/_{50}$s flash time colour 2: VDU 23,10,n|	M/C
23,11	Set default ECF patterns: VDU 23,11|	M/C
23,12	Set simple ECF pattern 1	M/C
23,13	Set simple ECF pattern 2	M/C
23,14	Set simple ECF pattern 3	M/C
23,15	Set simple ECF pattern 4	M/C
23,16	Control cursor directional movement	M/C
23,17	Reserved	
23,18	Reserved	
23,19	Reserved	
23,20	Reserved	
23,21	Reserved	
23,22	Reserved	
23,23	Reserved	
23,24	Reserved	
23,25	Reserved	
23,26	Reserved : Acorn Communicator font changes	Comm.
23,27	Master Compact/GXR Sprites	C
23,28	User call	All
23,29	User call	All
23,30	User call	All
23,31	User call	All
23,32	Define matrix of ASCII character 32	All
23,33	Define matrix of ASCII character 33 to …	All
23,255	… Define matrix of ASCII character 255	All

Parameters from 33 to 255 are also for redefining characters' matrices. The matrix for character 127 can be defined, but cannot be used, as this is a control code (DELETE).

VDU 25

This command, equivalent to PLOT in BBC BASIC, is for drawing high resolution graphics on screen. The commands come in groups of eight. The groups are formed as follows.

0	Move old graphics cursor to current graphics cursor.
1	Plot in current graphics foreground colour.
2	Plot in inverse of current pixel colour.
3	Plot in current background colour.

These numbers apply to relative plotting. For absolute plotting add 4 to them. Therefore to plot an inverse line, minus the final point, using absolute co-ordinate, the plot number would be 25+4+2 = 31. All VDU 25 commands take five parameters—the plot number, the X co-ordinate, low byte then high byte, followed by the Y co-ordinate in the same manner.

VDU	Action performed	Machine
25,0	Plot line	All
25,8	Plot line minus final point	All
25,16	Plot dotted line	All
25,24	Plot dotted line minus final point	All
25,32	Plot line minus initial point	M/C
25,40	Plot line minus initial and final point	M/C
25,48	Plot dotted line minus initial point	M/C
25,56	Plot dotted line minus initial and final point	M/C
25,64	Plot dot	All
25,72	Fill line to left and right non-background	All
25,80	Plot triangle	All
25,88	Fill line to right to background	All
25,96	Plot rectangle	M/C
25,104	Fill line to left and right to foreground	M/C
25,112	Plot parallelogram	M/C
25,120	Fill line to right to non-foreground	M/C
25,128	Flood fill to non-background	M/C
25,136	Flood fill to foreground	M/C
25,144	Plot circle	M/C
25,152	Plot filled circle	M/C

25,160	Plot circular arc	M/C
25,168	Plot filled chord segment	M/C
25,176	Plot filled sector	M/C
25,184	Move/copy rectangle	M/C
25,192	Plot ellipse outline	M/C
25,200	Plot solid ellipse	M/C
25,208	Reserved	M/C
25,232	Sprites	C
25,240	Reserved for user calls	M/C

D : Memory Map

The memory maps are listed below. Most of the usage is similar across the range of Acorn machines, so we have produced only one memory map, and noted the differences, instead of producing separate maps.

Your machine code may use any workspace designated as 'scratch' provided that it does not expect it to be preserved after any system call. In general you may safely use areas reserved for certain functions if you can guarantee that those functions will not be invoked. ie if there is no sound in your program, you may use the sound buffers for other purposes. Note however that hard or soft reset may wipe all memory designated as buffer space, and load defaults for all MOS variables, even those you may not be using.

Page &00 (0)

Models B/B+/E

&00-&8F	Language workspace
&90-&9F	Econet system workspace
&A0-&A7	NMI owner's workspace
&A8-&AF	Operating system scratch space
&B0-&BF	Filing system scratch space
&C0-&CF	Filing system workspace
&D0	VDU status byte (STATS)
&D1	Byte mask for current graphics point (ZMASK)
&D2	Text colour OR mask (ZORA)
&D3	Text colour EOR mask (ZEOR)
&D4	Graphics colour OR mask
&D5	Graphics colour EOR mask
&D6-&D7	Addr of top line of curr graphics char cell
&D8-&D9	Address of top scan line of curr text char
&DA-&DB	Graphics scratch space (ZTEMP)
&DC-&DD	Graphics scratch space (ZTEMPA)
&DE-&DF	Graphics scratch space (ZTEMPB)
&E0-&E1	Pointer to multiplication tables
&E2	CFS/RFS status byte
&E3	CFS/RFS options byte
&E4-&E6	General OS workspace
&E7	Auto-repeat countdown timer byte
&E8-&E9	Pointer to input buffer for OSWORD &00
&EA	RS423 timeout counter

&EB	Cassette/RFS 'critical' flag
&EC	Internal key number of most recently pressed key
&ED	Internal key number of most recent key but one
&EE	Internal key number of character to be ignored when using OSBYTE &79. Also allocated as 1MHz bus paging register.
&EF	A value for last OSBYTE/OSWORD
&F0	X value for last OSBYTE/OSWORD
&F1	Y value for last OSBYTE/OSWORD
&F2-&F3	Used (with Y) as a text pointer for command input
&F4	RAM copy of ROMSEL (&FE30)
&F5	Logical speech PHROM or ROM filing system ROM number.
&F6-&F7	Address pointer into paged ROM/speech PHROM
&F8-&F9	Spare
&FA-&FB	General OS workspace
&FC	Interrupt accumulator save register
&FD-&FE	Pointer to last error number/message
&FF	Escape flag (bit 7)

Master 128/Compact

Identical except :-

&E0-&E1	General workspace. There is no access to the multiplication table on the Master and Compact.
&F8-&F9	Now soft key expansion pointer. Points to next free byte in soft key buffer. These bytes used to be spare, so beware Model B programs that used them for their own purposes.

On Compact, CFS locations apply to RFS only. On Master/Compact &F5-&F7 used as RFS pointers only.

Page &01 (1)

All Machines

&100-&1FF	6502 hardware stack.
&100-	Error message buffer - used by most sideways ROMs to store their error messages.
&128-	Used by Tube Host Code to store parameter blocks.

Page &02 (2)

Models B/B+/E

&200-&235	MOS vectors
&236-&28F	System variables accessed by OSBYTEs &A6-&FF
&290	VDU vertical adjust (as set by OSBYTE &90)
&291	Interlace toggle flag (as set by OSBYTE &90)
&292-&296	First value of TIME
&297-&29B	Second value of TIME
&29C-&2A0	Countdown interval timer value
&2A1-&2B0	Paged ROM type byte table (pointed to by OSBYTEs &AA and &AB)
&2B1-&2B2	INKEY countdown timer
&2B3-&2B5	Work locations OSWORD 0
&2B6-&2B9	Low bytes of most recent ADC conversions
&2BA-&2BD	High bytes of most recent ADC conversions
&2BE	Last channel to finish conversion
&2BF-&2C8	Event enable flags
&2C9	Soft key expansion pointer
&2CA	First auto-repeat count
&2CB-&2CD	Workspace for two-key rollover
&2CE	Sound semaphore
&2CF-&2D7	Buffer busy flags
&2D8-&2E0	Buffer start indices (next byte to be removed)
&2E1-&2E9	Buffer end indices (last byte to be inserted)
&2EA-&2EB	Block size of current block of open CFS input file
&2EC	Block flag of current block of open CFS input file
&2ED	Last character of current block as above
&2EE-&2FF	Control blocks for OSFILE during *LOAD and *SAVE

Master 128/Compact
As the soft key expansion pointer now requires two bytes, it has been moved to &F8/&F9. &2C9 is thus free, and all the following bytes move down one:

&2C9	First auto-repeat count
&2CA-&2CC	Workspace for two-key rollover
&2CD	Sound semaphore
&2CE-&2D6	Buffer busy flags
&2D7-&2DF	Buffer start indices (next byte to be removed)
&2E0-&2E8	Buffer end indices (last byte to be inserted)
&2E9-&2EA	Block size of current block of open CFS input file
&2EB	Block flag of current block of open CFS input file
&2EC	Last character of current block as above
&2ED	Spare

Page &03 (3)

Models B/B+/E

&300-&307	Current graphics window in internal co-ordinates
&308-&30B	Current text window in same order as VDU 28 command
&30C-&30F	Current graphics origin in same order as VDU 29 command
&310-&313	Current graphics cursor
&314-&317	Old graphics cursor (in internal)
&318	POS
&319	VPOS
&31A	Line within current graphics char of POINT
&31B-&31E	Graphics workspace or part of VDU queue
&31F-&323	VDU queue (always ends at &323)
&324-&327	Current graphics cursor (in internal)
&328-&349	General graphics co-ordinate workspace (Don't use &328 in Mode 7/135)
&34A-&34B	Text cursor address for 6845
&34C-&34D	Text window size in bytes (for scrolling)
&34E	High byte of HIMEM (ignoring shadow)
&34F	No of bytes taken by a char in this mode
&350-&351	Address of top left corner of screen, as sent to 6845
&352-&353	No of bytes taken per char row of the screen.
&354	Size of screen memory (in pages)

&355	Current screen mode (ignoring shadow)
&356	Memory map type
&357-&35A	Current colours being used by the MOS
&35B	Current graphics foreground plot mode (as set by VDU 18)
&35C	Current graphics background plot mode (as set by VDU 18)
&35D-&35E	General jump vector
&35F	Last setting of 6845 cursor start register
&360	No of logical colours in this mode -1
&361	No of pixels per byte in this mode -1 (0=text)
&362-&363	Left and right colour masks
&364-&365	X and Y co-ordinates of text input cursor

B/B+/E

&366	Teletext output cursor character (Default=&FF)
&367	Font flag
&368-&36E	Hi-bytes of addresses for sections of font

M/C

&366	VDU 23,16 setting
&367	Dot pattern (as set by VDU 23,6)
&368	Current state of dot pattern
&369	Colour plotting. This is set to the ECF pattern if ECF in use, else zero
&36A	Graphics foreground. Set to ECF pattern if ECF in use, else zero
&36B	Graphics background. Set to ECF pattern if in use, else zero
&36C	Top bit set when cursor is in column 81
&36D	Current graphics foreground colour (as set by GCOL)
&36E	Current graphics background colour (as set by GCOL)

All machines

&36F-&37E	Colour pallette - contains physical colour for each logical one.
&37F	Spare

&380-&3DF are CFS (or RFS) workspace. This space is completely free if you are not using CFS or RFS. It is set to zero on power-up and is not altered by hard reset, except for &3D1 which is set to its default of &19 (25). The locations apply to all machines. Obviously the Compact uses them for RFS only, so &380-&3A6 (not &39E) are always free to the Compact or if you are only reading from CFS.

&380-&39C	Header block for CFS BPUT files
&39D	Offset for next byte to be BPUT by CFS
&39E	Offset for next byte to be BGET by CFS
&39F-&3A6	Spare
&3A7-&3B1	Filename of current CFS input file
&3B2-&3BC	Filename of most recent CFS block read
&3BD	&00
&3BE-&3C1	Load address of most recent CFS block read
&3C2-&3C5	Exec address of most recent CFS block read
&3C6-&3C7	Block number of most recent CFS block read
&3C8-&3C9	Length of most recent CFS block read
&3CA	Flag byte of most recent CFS block read
&3CB-&3CE	Spare bytes (will be overwritten by next CFS block)
&3CF-&3D0	Checksum bytes of most recent CFS block read
&3D1	CFS sequential block gap as set by *OPT 3
&3D2-&3DB	Filename of file being searched for
&3DC	&00
&3DD-&3DE	Number of next block expected by BGET
&3DF	Copy of flag byte of last block read
&3E0-&3FF	Keyboard input buffer

Pages &04-&07

All machines

&400-&7FF	Language workspace

Page &08 (8)

All machines

&800-&83F	Sound workspace
&840-&84F	Sound channel 0 buffer

&850-&85F	Sound channel 1 buffer
&860-&86F	Sound channel 2 buffer
&870-&87F	Sound channel 3 buffer
&880-&8BF	Printer buffer
&8C0-&8FF	Storage of ENVELOPEs 1-4 (first 13 bytes of each 16 used)

Page &09 (9)

All machines

&900-&9BF	Storage of ENVELOPEs 5-16 or RS423/Cassette buffer (RS423/Cassette activity will overwrite the envelope data)
&9C0-&9FF	Speech buffer or Cassette buffer. The RS423 buffer runs to &9BF, but the cassette buffer extends to a full 256 bytes. Note that CFS OSFILE does not corrupt the cassette buffer.

Page &0A (10)

All machines

&A00-&AFF	Cassette or RS423 input buffer

Page &0B (11)

The uses of pages &0B and &0C has totally changed on the Master Series. If you are sure that your program will not be used on Econet, and will only be run on Master 128s, you may use these pages for code or data.

Model B/B+/E

&B00-&B10	Soft key pointers
&B11-&BFF	Soft key definitions

Master 128/Compact

&B00-&BFF	Econet workspace
&B01	Station number
&B02	File server station number
&B03	File server network number
&B18	Printer server type string

Page &0C (12)

Model B/B+/E

&C00-&CFF	Character definitions - ASCII characters 128-159

Master 128/Compact

&C00-&CFF	Econet workspace

Page &0D (13)

All machines

&D00-&D5F	NMI handler code (on Model B, up to &D9E was reserved)
&D60-&D7F	Econet workspace
&D80-&D91	Spare
&D92-&D9E	Reserved for use by Trackerball / input device
&D9F-&DEF	Extended vector table for paged ROMs
&DF0-&DFF	Private workspace or frugalising table for paged ROMs

Pages &0E-&7F

Model B/B+/E
If a Filing system such as DFS and/or ADFS is fitted then these will claim workspace memory. The amount claimed will vary but as a rule of thumb expect PAGE to be set to &1900 through to &2000. Memory above this to &7FFF is free for applications.

Master 128/Compact
User RAM to &7FFF free for application programs.

Private RAM

Master 128/Compact
A 12k block of Private RAM exists on Master series computers divided into two blocks. The first, called ANDY, is 4k in length and the second called HAZEL, is 8k in length. This 12k block is present on the Model B+ but unlike Master series micros it is not used by the MOS or sideways ROMs as workspace.

ANDY - &8000 to &8FFF

Master 128/Compact

&8000-&800F	Start addresses of 16 soft key strings (low bytes)
&8010	End address+1 of last string (low byte)
&8011-&8020	Start addresses of 16 soft key strings (high bytes)
&8021	End address+1 of last string (high byte)
&8022-&83FF	Soft key data (no terminators, all pure data)
&8400-&87FF	VDU drivers
&8800-&88FF	VDU variables
&8900-&8FFF	Current character definitions 32-255

HAZEL - &C000 to & DFFF

Master 128/Compact

&C000-&DBFF	Workspace for paged ROMs
&DC00-&DCFF	MOS CLI buffer. You can use this but it will be corrupted by all '*' commands.
&DD00-&DEFF	Available for user routines. This area is used by *MOVE and other MOS routines.
&DF00-&DFFF	Various controls, mostly to do with filing systems. A complete list of memory usage in page &DF is given in Chapter 8.

Pages &80-&BF

All machines

These are occupied by 16 banks of sideways RAM or ROM. Details of the mapping is given in Chapter 5. Also in this area is ANDY, details of which are given above.

Pages &C0-&FB

All machines

This is occupied by Operating System ROM, when HAZEL is not switched in.

Page &FC (252) - FRED

All machines

&FC00-&FC0F	Test hardware

&FC10-&FC13	Teletext
&FC14-&FC17	Unallocated
&FC18-&FC1F	Reserved
&FC20-&FC27	IEEE 488
&FC28-&FC2F	Unallocated
&FC30-&FC3F	Reserved
&FC40-&FC47	Winchester
&FC48-&FC4F	Reserved
&FC50-&FC7F	Unallocated
&FC80-&FC8F	Test hardware
&FC90-&FCFE	Unallocated
&FCFF	Paged memory register

Page &FE (254) – SHEILA

Master 128/Compact
This is where all the I/O devices are mapped. Some I/O devices are mapped directly, others are accessed on the slow data bus of the system VIA. The Compact of course, has some devices missing. For more details on the I/O devices see the User Guide, Advanced User Guide, and the New Advanced User Guide, and Master Ref. Pt.1

	Read	**Write**
&FE00	6845 CRTC addr. Reg.	Do.
&FE01	6845 CRTC data reg.	Do.
&FE08	6850 ACIA status reg.	6850 control register
&FE09	6850 ACIA recve data reg.	6850 transmit register
&FE10	-	1MHz SerProc (ULA)
&FE18	7002 ADC status register	7002 ADC data latch
&FE19	7002 ADC data high byte	-
&FE1A	7002 ADC data low byte	-
&FE20	-	VIDPROC control register
&FE21	-	VIDPROC palette register
&FE24	Control latch for 1770	
&FE28	1770 FDC data register	
&FE30	-	ROMSEL
&FE34	ACCCON select register	Do.
&FE40	Input Slow Control Bus	Output Slow Control Bus

&FE41	Input Slow Data Bus	Output Slow Data Bus
&FE42	Data Direction SCB	
&FE43	Data Direction SDB	
&FE44	Timer 1 Low Order counter	Timer 1 Low Order latch
&FE45	Timer 1 High Order counter	
&FE46	-	Timer 1 Low Order latch
&FE47	-	Timer 1 High Order latch
&FE48	Timer 2 Low Order counter	Timer 2 Low Order latch
&FE49	Timer 2 Hi Order counter	
&FE4A	Shift Register	
&FE4B	Auxiliary Ctrl Register	
&FE4C	Peripheral Ctrl Register	
&FE4D	Interrupt Flag Register	
&FE4E	Interrupt Enable Register	
&FE4F	Input Slow Data Bus	Output Slow Data Bus
&FE60	Input User Port	Output User Port
&FE61	-	Output printer data
&FE62	Data Direction User Port	Do.
&FE63	Data Dir. Printer Data	Do.
&FE64	Timer 1 Low Order counter	Timer 1 Low Order latch
&FE65	Timer 1 High Order counter	
&FE66	Timer 1 Low Order latch	
&FE67	-	Timer 1 High Order latch
&FE80	Int. Modem Data	Int. Modem Data
&FE84	Int. Modem Control	Int. Modem Control
&FEA0	Net interface	
&FEE0	Tube R1STAT	Do.
&FEE1	Tube R1DATA	Do.
&FEE2	Tube R2STAT	Do.
&FEE3	Tube R2DATA	Do.
&FEE4	Tube R3STAT	Do.
&FEE5	Tube R3DATA	Do.
&FEE6	Tube R4STAT	Do.
&FEE7	Tube R4DATA	Do.
&FEE8	Reserved for Tube splitter	Do.

The missing addresses contain further images of the various ports, as full address decoding is not performed. The changes from the B/B+ are that the ADC and disc controllers have moved, they were previously mapped at &FEC0 and &FE80 respectively, the disc controller of course being a different device on the Model B (but the 1770 was mapped at &FE80-&FE84 on the BBC B+). Also the Econet station identity used to be hardwired at &FE18, but is now contained in byte 0 of non-volatile memory.

Page &FF (MOS)

See Appendix E for a list of the MOS calls, whose entry points are in page &FF.

Graphical maps

The next three pages (don't!) contain graphical maps to show you how the memory in the BBC B, B+, and Master Series is allocated.

FIGURE D.1 BBC B/Electron Memory Map.

FIGURE D.2 BBC B+ Memory Map.

FIGURE D.3 Master Series Memory Map.

E : OS Call Item List

Baud Rates

These are the baud rates supported by the serial system. These numbers are used by OSBYTE &07 and &08, and *CONFIGURE and *STATUS

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

Buffers

OSBYTE &15, &80, &8A, &91 etc. control the various buffered I/O channels, and make extensive use of buffer numbers. These are the various buffers and their numbers.

0	Keyboard buffer
1	RS423 input buffer
2	RS423 output buffer
3	Printer buffer
4	Sound channel 0 buffer
5	Sound channel 1 buffer
6	Sound channel 2 buffer
7	Sound channel 3 buffer
8	Speech buffer (Reserved on Master)

Events

The event vector (&220) is entered with A set to the 'event number', which is also the numbering system used by OSBYTE &0D and &0E. Note that event 254 is not affected by OSBYTE &0D/&0E, but instead by OSBYTE &34,

0	Output buffer has become empty
1	Input buffer has become full
2	Character has entered buffer
3	An ADC conversion is complete
4	Start of vertical sync signal
5	Interval timer has crossed zero
6	The active ESCAPE key has been pressed
7	An RS423 error has occurred
8	A network error has occurred
9	The user has generated an event
254	Network receive event

Filing System Information

No.	Name	BREAK	Selected by
0	No filing system	-	*NOTAPE (OS 0.1 only)
1	1200baudCFS	SPACE	*TAPE
2	300 baud CFS	-	*TAPE 3
3	RFS	SHIFT-SPACE	*ROM
4	DFS	D	*DISC / *DISK
5	NFS	-	*NET
6	Telesoftware FS	T	*TELESOFT
7	IEEE FS	I	*IEEE
8	ADFS	A/F	*ADFS / *FADFS
9	Host Filing System	-	-
10	Videodisc FS	Q/L	*VFS / *LVFS

System Calls and Vectors

These are all the system entry points, and the vectors through which they pass. Note that some vectors have no ROM entry point, so you call them by JMP (vector), and some MOS routines are not vectored (such as GSINIT)

Routine	Address	Vector	Addr	Ext Vector Address & action
		USERV	&200	&FF00 *CODE, *LINE etc.
		BRKV	&202	&FF03 All BRKs thru here
		IRQ1V	&204	&FF06 All interrupts
		IRQ2V	&206	&FF09 All unrec. interrupts
OSCLI	&FFF7	CLIV	&208	&FF0C '*' commands
OSBYTE	&FFF4	BYTEV	&20A	&FF0F *FX calls
OSWORD	&FFF1	WORDV	&20C	&FF12 OSWORD calls
OSWRCH	&FFEE	WRCHV	&20E	&FF15 Output character
OSRDCH	&FFE0	RDCHV	&210	&FF18 Get character
OSFILE	&FFDD	FILEV	&212	&FF1B *LOAD/*SAVE
OSARGS	&FFDA	ARGSV	&214	&FF1E Misc. file info
OSBGET	&FFD7	BGETV	&216	&FF21 Get 1 byte from file
OSBPUT	&FFD4	BPUTV	&218	&FF24 Put 1 byte to file
OSGBPB	&FFD1	GBPBV	&21A	&FF27 Misc. file transfers
OSFIND	&FFCE	FINDV	&21C	&FF2A Open/close files
		FSCV	&21E	&FF2D Filing system ctrls
OSEVEN	&FFBF	EVENTV	&220	&FF30 Cause an event
		UPTV	&222	&FF33 User print vector
		NETV	&224	&FF36 All Econet data
		VDUV	&226	&FF39 All VDU output
		KEYV	&228	&FF3C All key presses
		INSV	&22A	&FF3F Insertion to buffer
		REMV	&22C	&FF42 Removal from buffer
		CNPV	&22E	&FF45 Count/purge buffer

Routine	Address	Vector	Addr	Ext Vector Address & action
		IND1V	&230	&FF48 Spare 1
		IND2V	&232	&FF4B Spare 2
		IND3V	&234	&FF4E Spare 3
OSASCI	&FFE3	via OSWRCH		OSWRCH with CR=CRLF
OSNEWL	&FFE7	via OSWRCH		Print a CRLF
OSVDU	&FFBC	Not vectored		Raw VDU entry point
OSRDSC	&FFB9	Not vectored		Read from paged mem.
OSWRSC	&FFB3	Not vectored		Write to paged mem.
GSINIT	&FFC2	Not vectored		Initialise (&F2),Y
GSREAD	&FFC5	Not vectored		Read from (&F2),Y
NVRDCH	&FFC8	Not vectored		Unvectored OSRDCH
NVWRCH	&FFCB	Not vectored		Unvectored OSWRCH
	&FFB6/8	VECTAB		&FFB6 - Length
				&FFB7/8 - Address of default vector table
		NMIV		&FFFA 6502 NMI vector
		RESETV		&FFFC 6502 reset vector
		IRQV		&FFFE 6502 IRQ vector

F : Key Numbers

This is a table of the ASCII codes, and internal numbers generated by each key. The INKEY number equals the internal key number EOR &FFFF. So SPACE is (98 EOR &FFFF)=&FF9D=-99. The only MOS call using internal key numbers is OSBYTE &78. The INKEY numbers are used by OSYTE &81. The ASCII numbers are used by OSRDCH.

Key/	**ASCII**		**INKEY**		**Internal Key No**	
char	**dec**	**hex**	**dec**	**hex**	**dec**	**hex**
SPACE	32	20	-99	9D	98	62
!	33	21	See '1'			
"	34	22	See '2'			
#	35	23	See '3'			
$	36	24	See '4'			
%	37	25	See '5'			
&	38	26	See '6'			
'	39	27	See '7'			
(	40	28	See '8'			
)	41	29	See '9'			
*	42	2A	See ':'			
+	43	2B	See ';'			
,	44	2C	-103	99	102	66
-	45	2D	-24	E8	23	17
.	46	2E	-104	98	103	67
/	47	2F	-105	87	104	68
0	48	30	-40	D8	39	27
1	49	31	-49	CF	48	30
2	50	32	-50	CE	49	31
3	51	33	-18	EE	17	11

Key/	ASCII		INKEY		Internal Key No		
char	**dec**	**hex**	**dec**	**hex**	**dec**	**hex**	
4	52	34	-19	ED	18	12	
5	53	35	-20	EC	19	13	
6	54	36	-53	CB	52	34	
7	55	37	-37	DB	36	24	
8	56	38	-22	EA	21	15	
9	57	39	-39	D9	37	25	
:	58	3A	-73	B7	72	48	
;	59	3B	-88	A8	87	57	
<	60	3C	See ','				
=	61	3D	See '-'				
>	62	3E	See '.'				
?	63	3F	See '/'				
@	64	40	-72	B8	71	47	(Code key on C)
A	65	41	-66	BE	65	41	
B	66	42	-101	9B	100	64	
C	67	43	-83	AD	82	52	
D	68	44	-51	CD	50	32	
E	69	45	-35	DD	34	22	
F	70	46	-68	BC	67	43	
G	71	47	-84	AC	83	53	
H	72	48	-85	AB	84	54	
I	73	49	-38	DA	38	26	
J	74	4A	-70	BA	69	45	
K	75	4B	-71	B9	70	46	
L	76	4C	-87	A9	86	56	
M	77	4D	-102	9A	101	65	

Key/	ASCII		INKEY		Internal Key No	
char	**dec**	**hex**	**dec**	**hex**	**dec**	**hex**
N	78	4E	-86	AA	85	55
O	79	4F	-55	C9	54	36
P	80	50	-56	C8	55	37
Q	81	51	-17	EF	16	10
R	82	52	-52	CC	51	33
S	83	53	-82	AE	81	51
T	84	54	-36	DC	35	23
U	85	55	-54	CA	53	35
V	86	56	-100	9C	99	63
W	87	57	-34	DE	33	21
X	88	58	-67	BD	66	42
Y	89	59	-69	BB	68	44
Z	90	5A	-98	9E	97	61
[	91	5B	-57	C7	56	38
\	92	5C	-121	87	120	78
]	93	5D	-89	A7	88	58
^	94	5E	-25	E7	24	18
_	95	5F	-41	D7	40	28
£	96	60	See '_'			
a	97	61	See 'A'			
b	98	62				
c	99	63				
d	100	64				
e	101	65				
f	102	66				
g	103	67				

Key/ char	ASCII dec	hex	INKEY dec	hex	Internal Key No dec	hex
h	104	68				
i	105	69				
j	106	6A				
k	107	6B				
l	108	6C				
m	109	6D				
n	110	6E				
o	111	6F				
p	112	70				
q	113	71				
r	114	72				
s	115	73				
t	116	74				
u	117	75				
v	118	76				
w	119	77				
x	120	78				
y	121	79				
z	122	7A				
<	123	7B				
|	124	7C				
>	125	7D				
~	126	7E				

Key/	ASCII		INKEY		Internal Key No	
char	**dec**	**hex**	**dec**	**hex**	**dec**	**hex**
ESCAPE	27	1B	-113	8F	112	70
TAB	9	9	-97	9F	86	60
CAPS LOCK			-65	BF	64	40
CTRL			-2	FE	1	01
SHIFT LOCK			-81	AF	80	50
SHIFT			-1	FF	0	00
DELETE	127	7F	-90	A6	89	59
COPY	135	87	-106	96	105	69
RETURN	13	0D	-74	B6	73	49
UP CURSOR	139	8B	-58	C6	57	39
DN CURSOR	138	8A	-42	D6	41	29
LT CURSOR	136	88	-26	E6	25	19
RT CURSOR	137	89	-122	86	121	79
f0			-33	DF	32	20
f1			-114	8E	113	71
f2			-115	8D	114	72
f3			-116	8F	115	73
f4			-21	EB	20	14
f5			-117	8B	116	74
f6			-118	8C	117	75
f7			-23	E9	22	16
f8			-119	8D	118	76
f9			-120	8E	119	77

Start Up Option

Model B/B+ only (on front of keyboard)

Key/	ASCII		INKEY		Internal Key No	
char	**dec**	**hex**	**dec**	**hex**	**dec**	**hex**
bit 0					9	09
bit 1					8	08
bit 2					7	07
bit 3					6	06

Key/	ASCII		INKEY		Internal Key No	
char	dec	hex	dec	hex	dec	hex
bit 4					5	05
bit 5					4	04
bit 6					3	03
bit 7					2	02

Numeric Keypad

Master 128 and Compact only

Key/	ASCII		INKEY		Internal Key No	
char	dec	hex	dec	hex	dec	hex
Kpad 0	48	30	-107	95	106	6B
Kpad 1	49	31	-108	94	107	6C
Kpad 2	50	32	-125	83	124	7C
Kpad 3	51	33	-109	93	108	6C
Kpad 4	52	34	-123	85	122	7A
Kpad 5	53	35	-124	84	123	7B
Kpad 6	54	36	-27	E5	26	1A
Kpad 7	55	37	-28	E4	27	1B
Kpad 8	56	38	-43	D5	42	2A
Kpad 9	57	39	-44	D4	43	2B
Kpad #	35	23	-91	A5	90	5A
Kpad +	43	2B	-59	C5	58	3A
Kpad -	45	2D	-60	C4	59	3B
Kpad /	47	2F	-75	B5	74	4A
Kpad *	42	2A	-92	A4	91	5B
Kp DELETE	127	7F	-76	B4	75	4B
Kpad ,	44	2C	-93	A3	92	5C
Kpad .	46	2B	-77	B3	76	4C
Kp RETURN	13	0D	-61	C3	60	3C

G : PCB Links

The circuit board links on BBC Microcomputers allow various options to be selected, these are in the main to do with fixing of temporary problems, but also to configure the machine to work in a particular hardware setup. In the lists below, L=Link, ie, L24=Link 24. Link numbers are sometimes shown on the circuit board as S24, which also means Link 24. b is used for bit, ie, b4=bit 4. The four points of the compass, N, S, W and E, are used to signify the direction of the link being described. This assumes that you have the circuit board in front of you in the position in which it sits during use. North is away from you, South is near you, West is to your left hand, and East to your right. Otherwise O=Open and C=Closed links.

BBC Model B

L1	N=Selects printer strobe on printer connector. S=Selects 6522 CA2 connected to printer strobe.
L2	O=Enable Econet NMI. C=Disable Econet NMI. Don't do this when Econet interface fitted.
L3	Clock base frequency for Econet. Not fitted from Issue 4.
L4	E=5.25" /3.5" /3" disc interface. W=8" disc interface.
L5	N=Enable Econet clock. S=Disable Econet clock. Not fitted from Issue 4.
L6	N=Divide Econet clock by 2. S=Divide Econet clock by 4. Not fitted from Issue 4.
L7	W=Connect 8271 (pin 30) to +5v. E=Connect 8271 (pin 30) to 0v.
L8	O=Disconnects the disc head load from PL8. C=Connects the disc head load to PL8.
L9	O=Enable disc NMI. C=Disable disc NMI.
L10	W=Select 5.25" /3.5" /3" disc interface. E=Select 8" disc interface.

L11 This block of eight links determines the Econet station number, and appears at &FE18 in the memory map. Open links are read as 1, closed links as 0. b7 is at the SOUTH end.

L12 C=Connect ROM select line A to 0v - Model A.
O=ROM select line A driven by IC76 (mapped at &FE30)—Model B.

L13 C=Connect ROM select line B to 0v - Model A.
O=ROM select line B driven by IC76 (mapped at &FE30)—Model B.

L14 O=Enable I/O in page &FD.
C=Enable ROM in page &FD.

L15 O=Enable fast access to page &FD.
C=Disable fast access to page &FD.

L16 O=Enable fast access to page &FC.
C=Disable fast access to page &FC.

L17 O=Disable FRED, enable ROM output from page &FC.
C=Enable FRED, disable ROM output from page &FC.

L18 N=Fast access to sideways ROM IC100.
S=Slow access to sideways ROM IC100.

L19 W=Slow access to sideways ROMs IC52, IC88, IC101.
E=Fast access to sideways ROMs IC52, IC88, IC101.

L20 N=ROMSEL provides HIGH ROM select to IC20.
S=A13 provides HIGH ROM select to IC20.

L21 Two position links:
NS/NS=IC51 mapped at &8000-&BFFF, other ROMs at &C000-&FFFF.
EW/EW=IC51 mapped at &C000-&FFFF, other ROMs at &8000-&BFFF.

L22 N=ROMSEL provides LOW ROM select to IC20.
S=A12 provides LOW ROM select to IC20.

L23 N=RS423 data line not terminated.
S=RS423 data line terminated.

L24 O=RS423 CTS line not terminated.
C=RS423 CTS line terminated.

L25 N=32k RAM select - Model B
S=16k RAM select - Model A
L26 W=Normal video output.
E=Inverse video output.
L27 W=5.25″ /3.5″ /3″ disc select (8MHz clock).
E=8″ disc select (16MHz clock).
L28 W=Base baud rate select.
E=1200 baud rate select.
L29 W=1200 baud rate select.
E=Base baud rate select.
L30 Used to wire-OR ROM select signals. See L34-L38.
L31 W=positive RGB CSYNC.
E=negative RGB CSYNC.
L32 W=A13 pins IC52 and IC88 to processor A13.
E=A13 pins IC52 and IC88 to +5v.
L33 W=A13 pins IC100 and IC101 to processor A13.
E=A13 pins IC100 and IC101 to +5v.
L34 O=Allows OS ROM to be wire-OR'ed.
C=Uses ordinary OS ROM select (Issue 4 onwards).
L35 O=Allows IC52 ROM to be wire-OR'ed.
C=Uses ordinary IC52 ROM select (Issue 4 onwards).
L36 O=Allows IC88 ROM to be wire-OR'ed.
C=Uses ordinary IC88 ROM select (Issue 4 onwards).
L37 O=Allows IC100 ROM to be wire-OR'ed.
C=Uses ordinary IC100 ROM select (Issue 4 onwards).
L38 O=Allows IC101 ROM to be wire-OR'ed.
C=Uses ordinary IC101 ROM select (Issue 4 onwards).
L39 O=Mono output on BNC socket.
C=Colour output on BNC socket.

BBC Model B+

L1 8/16MHz clock select for 8271 data window.
L2 Select ready between internal and external.

L3 Disc controller side select line control.
L4 Switch disc index input.
L6 Select 5.25″ or 8″ disc drives.
L7 E=1770 Disc Interface fitted.
W=8271 Disc Interface fitted.
L8 O=8271 circuitry enabled.
C=1770 circuitry enabled.
L9 W=Slot 2/3 (IC35) accepts 16k (27128-compatible) ROMs.
E=IC35 accepts 32k (27256-compatible) ROMs (Top half appears in slot 2)
L10 W=Disable BREAK key.
E=Enable BREAK key.
L11 W=Slot 4/5 socket (IC44) accepts 16k ROMs.
E=IC44 accepts 32k (27256-compatible) ROMs (Top half appears in slot 4).
L12 W=IC57 accepts 16k (27128-compatible) ROMs
E=IC57 accepts 32k (27256-compatible) ROMs (Top half appears in slot 6).
L13 N=BASIC/MOS ROM appears in Sockets 0/1.
S=BASIC/MOS ROM appears in Sockets 14/15.
L14 N=True video.
S=Inverse video.
L15 W=Slot 8/9 socket (IC62) accepts 16k ROMs.
E=IC62 accepts 32k (27256-compatible) ROMs (Top half appears in slot 8).
L16 O=Deselect BASIC/OS ROM.
C=Select BASIC/OS ROM.
L17 Select high/low level analogue input (Default low).
L18 W=Slot 10/11 (IC68) accepts 16k (27128-compatible) ROMs.
E=IC68 accepts 32k (27256-compatible) ROMs (Top half appears in slot 10).
L19 W=MOS/BASIC slot (IC71) accepts 16k (27128-compatible) ROMs.

E=IC71 accepts 32k (27256-compatible) ROMs (Top half appears in lower slot).

L20 S=Enable internal speaker (default).
N=Disable internal speaker.

L21 External audio output, normally bypassed by a 1nF capacitor.

L23 Set of eight links. Econet station identity.

L24 O=RS423 control not terminated.
C=RS423 control terminated.

L25 O=RS423 data not terminated.
C=RS423 data terminated.

L26 O=Mono output on composite video.
C=Colour output on composite video (hardwired default).

L27 N=Negative RGB CSYNC (Default).
S=Positive RGB CSYNC.

L28 Enable PAL/Disable alternation.

L29 C=Econet collision detect circuitry disabled (Factory default).
O=Econet collision detect circuitry enabled if fitted.

L30 C=Grounded pin 5 of CPU. Always closed for 6512.

Master 128

L1 A=1MHz bus signal is input via amp to speaker (hardwired default).
B=1MHz bus signal is an output from audio circuit.

L2 C=No -5V decoupled supply to cartridge (hardwired default).
O=-5V decoupled supply to cartridge.

L3 Not present on production boards.

L4 O=Clock chip IRQ on alarm time being reached not active.
C=Above feature is active. No harm will be done if you make this link.

L5 C=Negative CSYNC polarity. (Hardwired default).
O=Positive CSYNC polarity.

L6 Four pin link:
A—B Computer 16MHz reference provided by onboard crystal.

B—D Computer 16MHz reference provided by A17 cartridge pin.
C—D (and A—B) Cartridges clocked by 8MHz onboard signal, synchronous with 2MHz cartridge signal.

NB: If B—D is made, and no clock source is fitted on the cartridge port, dynamic RAM will be damaged. Any such cartridges must therefore be fitted with a locking bar to the computer. GENLOCK is the main application of this feature.

L7	C=RGB positive polarity (hardwired default). O=RGB negative polarity.
L8	Not present on production machines.
L9	Switch between 64k and 128k main system ROM. Present but not usable on production machines.
L10	Not supplied on UK machines. Fitted to NTSC machines to select channels.
L11	Not present on production machines.
L12	E=Cartridge line A10 is logic low. W=Cartridge line A10 is CSYNC
L13	C=Analogue VRef supplied by onboard circuitry (hardwired default). O=Analogue VRef supplied by user either at PR1 or at VRef on connector.
L14	Selects whether 1.23MHz signal is supplied by chroma chip or IC35. Present but not usable on production machines.
L15	C=TV output is PAL (hardwired default). O=TV output is NTSC.
L16	O=Chrominance information not filtered from luminance. C=Chrominance information filtered from luminance.
L17	Not present on production machines.
L18	W=ROM slots 6 and 7 are occupied by Sideways RAM. E=ROM slots 6 and 7 are occupied by IC41, a 27256-compatible 32K ROM.
L19	W=ROM slots 4 and 5 are occupied by Sideways RAM. E=ROM slots 4 and 5 are occupied by IC37, a 27256-compatible 32K ROM.

L20 Not present on production machines.
L21 O=Cartridge line B10, linked to other cartridge line B10. (hardwired default).
C=Cartridge line B10 fitted to CRTC LPSTB (for GENLOCK, to avoid having to use the A—D connector).

Master Compact

L1 This is the audio loudspeaker connector.
L2 This is the Power In connector.
L3 C=Drive 2 (hardware drive, not surface) line of 1772 enabled.
0=Drive 2 line of 1772 disabled (hardwired default).
L4 E=CTS line from 6850 connected to RS232 interface (hardwired default).
W=CTS line connected to cassette circuitry of Serial ULA.
L5 E=Rx Data line from 6850 connected to RS232 interface (hardwired default).
W=Rx Data line connected to cassette circuitry of Serial ULA.
L6 This is the keyboard connector, a twin row metric pitch 26 way connector.
L7 O=LPSTB is not connected to the joystick socket. (Hardwired default).
C=LPSTB is connected to the joystick socket.
L8 This is the digital joystick socket.
L9 E=Normal video display (hardwired default).
W=lnverse video display.
L10 E=Negative RGB sync (hardwired default).
W=Positive RGB sync.
L11 N=ROM slots 0 and 1 read from expansion connector. (Factory default).
S=ROM slots 0 and 1 read from IC17, a 27256-compatible 32k ROM.
L12 N=OS ROM IC49 is regarded as a 64k byte ROM.
S=OS ROM IC49 is regarded as a 128k byte ROM.
L13 This is the 50-way edge connector.

H : Cartridge ports

Master 128

The Master 128 cartridge ports are two 44 way edge connectors behin a spring-loaded flap just above the keyboard. The edge connectors are arranged so that, for each socket, Side B is the side nearer to you (South), and Side A is the side nearer the back (North). The specification is a superset of the Electron Plus 1 system, and the differences between the two are noted. Accesses to the M128 cartridges are performed at 2MHz.

Pinout List

la	+5V : The main power supply. Maximum rating is 150mA, or 50mA on an Electron.
1b	Connected to 1a
2a	~OE : Output Enable. Active low during the PHI2 period of the system clock.
2b	A10
3a	~RST : System Reset. Active low during reset.
3b	D3
4a	CSRW : Chip Select, Read/Write. During access to memory mapped I/O and always on the Electron, this is the 6502 R/W line. On the Master at other times, it is a chip select (active high).
4b	A11
5a	A8
5b	A9
6a	Al3
6b	D7
7a	Al2
7b	D6
8a	PHI2 : PHI2 out from the system clock.
8b	D5
9a	-5V : Rate 20mA max on both Electron and M128.

9b D4

10a CSYNC/MADET : If M128 link PL12 is in position A, this is 0V. On an Electron it is NC and thus floats to logic 1. This allows you to test which machine you are on. If PL12 is in the B position, the pin receives CSYNC signals. Cartridges that use CSYNC cannot therefore test which machine they are on, but their control software can do this much more easily anyway. The pin is NC on an Electron.

10b ~OE 2 : LPSTB/OE2. On the M128, this line is the light pen strobe, and is also wired to 10b of the other cartridge. On the Electron it is Output Enable 2, an output enable which is low during the active low portion of PHI2.

11a RW/READY : If low cartridges are being written to. If high, they may drive the system bus. On the Electron, this is a CPU wait state control (open collector), a totally different function.

11b A7 : On the M128 only, this line is buffered for 125ns after PHI2 goes low. Accordingly it is known on the M128 as BA7

12a ~NMI : Active low 6502 non maskable interrupt line.

12b A6 : BA6 on the M128.

13a ~IRQ : Active low 6502 interrupt request line.

13b A5 : BA5 on the M128

14a ~INFC : If low this means that (a) the cartridges are selected as internal 1MHz bus, and (b) the processor is accessing JIM (&FC00-&FCFF).

14b A4 : BA4 on the M128

15a ~INFD : As 14a, but for FRED (&FD00-&FDFF).

15b A3 : BA3 on the M128

16a ROMQA : This is set to the logical value of b0 of ROMSEL.

16b A2 : BA2 on the M128

17a CLOCK : A 16MHz input, which on the M128 only, can be changed to 8MHz by moving PL6.

17b A1 : BA1 on the M128

18a ~ROMSTB/~CRTCRST : On the M128 only, this is active low signal compatible with the system CRTC reset input. On the Electron, it's an active low signal which selects &FC73, which can be a paging register.

18b A0 : BA0 on the M128
19a ADOUT : The filtered output of the sum of all audio inputs to the host computer. Only a nominal load should be placed on this line.
19b D0
20a AGND : Return for 19a, which is cleaner than system ground.
20b D2
21a ADIN : Audio output from cartridge to M128, which will offer 1kOhm impedance. This will go direct to M128 audio circuitry. NB: Do not fit two cartridges which generate audio output. On the Electron, this pin is merely wired to 21a on the other socket.
21b D1
22a 0V : System Ground.
22b Connected to 22a.
(~ is NOT, an active low line, written with an overhead bar in some texts).

Master Compact

The Master Compact has a 50-way edge connector, which is part of its PCB, and offers many of the signals found on the M128 cartridge port. However, ~CSYNC, CRTC, 16MHz, ~RST, -5V, ANOUT, AGND and SPEECH are missing, the last four being absent from the Compact hardware. Added to the connector are SCREEN (0V), and PB5, PB6 and PB7 from the User Port, the other bits of the User Port being available on the joystick port—see Appendix I. On the Compact edge connector, side A is the underside and Side B is the visible side.

Pinout List

Pin	Side A	Side B
1	0V Screen	0V Screen
2	+5V	+5V
3	AT13	Al0
4	~RST	CD3
5	AA15	A11

Pin	Side A	Side B
6	A8	A9
7	A13	CD7
8	A12	CD6
9	PHI2 OUT	CD5
10	NC	CD4
11	NC	LPSTB
12	BR/~W	BA7
13	~NMI	BA6
14	~IRQ	BA5
15	~INFC	BA4
16	~INFD	BA3
17	AA14	BA2
18	~8MHz	BA1
19	0V	BA0
20	PB7 (USER 6522)	CD0
21	PB6 (USER 6522)	CD2
22	PB5 (USER 6522)	CD1
23	Removed for polarisation cutout	
24	0V	0V
25	0V SCREEN	0V SCREEN

I : Compact Analogue Emulator

The Master Compact differs from the previous BBC Microcomputers in that there is no analogue to digital converter, and thus proportional or analogue joysticks won't work. Instead, a socket is provided for 'Atari-compatible' joysticks, which are switched. There are five switches, left, right, up, down, and fire. As the switches are separate, it is possible to switch more than one of them on at once. Many joystick units allow this, providing eight directions of movement. The connections on the 9-way D-type plug are given in Appendix J. The connections are actually made to the MOS using the User Port, mapped at &FE60.

To provide compatibility with previous analogue joysticks, MOS 5.00 onwards provides emulation of the original analogue code. Here are details of how each call relating to the analogue system is handled.

Pointing device service call
On powering up the Compact, the CB1 and CB2 interrupts from the User VIA are enabled, and a service call &2C is issued by the MOS every 20ms. For details of this call see Chapter 6. The call allows you to write your own code to control a pointer.

OSBYTE &04 (4) Cursor key state

A new option *FX4,3 is provided. Full details are given in Chapter 4.

OSBYTE &10 (16) Write number of channels.

This is a totally artificial system, but it pretends to let you select different channels. Unlike the real system, the default number of channels is 2, and amazingly, you can set any number up to 128 channels. OSBYTE &BC will happily return over 100 different numbers constantly changing!

OSBYTE &11 (17) Write next channel to be sampled.

Again this is artificial, but you can write a channel number between 1 and the maximum set to by OSBYTE &10, and OSBYTE &BD will return it.

OSBYTE &80 (128) Return analogue data.

This call, which is equivalent to ADVAL in BASIC, returns values compatible with analogue systems (including fixed values if b5 on OSBYTE &BE set), as follows:

ADVAL(0)	high byte : last channel to convert (artificial value for compatibility).
ADVAL(1)	X co-ordinate in the range 0-&FFFF. This value also returned by ADVAL(3) and ADVAL(5)
ADVAL(2)	Y co-ordinate in the range 0-&FFFF. This value also returned by ADVAL(4) and ADVAL(6).
ADVAL(7)	and ADVAL(8) also return the X and Y positions, but if the sprite pointer ROM is fitted, it takes over these two functions, and returns the position of the pointer.

OSBYTE &BC (188) Read currently converting channel.

This returns a number between 1 and the maximum set by OSBYTE &10. It displays a range of numbers decrementing from the maximum to 1, although this is purely set up by the MOS, and has no meaning.

OSBYTE &BD (189) Read max. ADC channel number.

This returns the number set by OSBYTE &11, however meaningless that number may be.

OSBYTE &BE (190)

On earlier machines this was used to switch between fast 8-bit conversion and accurate 12-bit conversion, and to read the current setting. This is irrelevant in an emulation, and the call is used for an entirely different purpose.

Entry: X=bit value (see below):Y=0
Exit : X=Old value

or

Entry: X=bit value (see below):Y=255
Exit : X=Current setting

b7 If set, do not update ADVAL values from digital joystick or cursor keys. This allows a sideways ROM etc. to handle it instead. If b7 clear, update as normal.

b6 1=Key value(s) are entered into the keyboard buffer according the ADVAL(0) (low-byte) setting (see OSBYTE &80 above). If bits are set, the values entered are:

bit	Device	ASCII	Key
b7	Right	&89	Cursor right
b6	Up	&8B	Cursor up
b5	Down	&8A	Cursor down
b4	Left	&88	Cursor left
b3	Right button	&7F	Delete
b2	Middle button	&0D	Return
b1	Left button	&87	Copy
b0	Fire (joystick)	&87	Copy

Note that the emulation does not extend to key numbers, so

```
PRINT INKEY-58
```

will return -1 if the cursor up key is pressed, but 0 if the joystick up switch is being held. b0 is only applicable to a joystick, b1-3 are only applicable to a mouse/trackerball, although of course you could build your own device which used both.

If set, ADVAL 1 and 2 return fixed numbers as follows:

ADVAL 1	Left switch on=&FFFF
	Neither horizontal switch on=&7FFF
	XRight switch on=0
ADVAL 2	Down switch on=&FFFF
	Neither vertical switch on=&7FFF
	Up switch on=0

Some games need the analogue emulation, but many respond better with the fixed numbers. An option has been provided:

```
*CONFIGURE PROPORTIONAL
```

will permanently set up the analogue emulation, and

```
*CONFIGURE SWITCHED
```

will set up these fixed results.

b4 Not used, but unused for compatibility ie so that *FX190,12 can be issued.

b3-0 Analogue speed feature. The Compact can emulate an analogue joystick. These bits set the sensitivity of the emulation - how quickly the MOS changes values in response to a stick movement. A value of 1 is a slow or sluggish emulation. A value of 7 is very fast . There is a permanent setting of this available with the new *CONFIGURE STICK command, which takes a parameter between 0 and 15. Values 0, and 8-15 actually assume the default value, which is that selected by *FX190,3.

Low Level Access - The User Port

You can bypass the operating system and read the joystick switches directly from the user port at &FE60. To do this, the data direction register (DDR) at &FE62 must be set so that the port is configured for input. This is done by loading it with 0. All the bits are set to 1, and change to 0 when the various switches are closed as follows:

b7 Not affected by joystick port.
b6 Not affected by joystick port.
b5 Not affected by joystick port.
b4 Joystick right/mouse y-axis movement.
b3 Joystick up/mouse x-axis movement.
b2 Joystick down/mouse right button.
b1 Joystick left /mouse middle button.
b0 Joystick fire button/mouse left button.

Listing I.1 is a simple program to print out the value of the bits on the User Port. Run this with a joystick or mouse plugged in. Listing I.2 allows you to use the joystick with games that normally only have keyboard input provided that they use the negative INKEY system of reading keys, as most do. Also, the code must be assembled somewhere that the game is not using. I have used &900, RS232 buffer workspace, but if your game uses this area, you'll have to try somewhere else.

Listing I.1

```
 10 REM Joystick/Mouse test
 20 REM (c) Dave Atherton 1987
 30 REM for Compact only
 40 REM (or B/B+/M with mouse/trackerball)
 50 REM MOS: A Dab Hand Guide
 60 :
 70 MODE 7
 80 VDU23,1;0;0;0;0;
 90 ?&FE62=0
100 PRINTTAB(6,8)"User Port :"
110 REPEAT
120 PRINTTAB(18,8)FNbinary(?&FE60)
130 UNTIL FALSE
140 END
150 :
160 DEF FNbinary(X%)
170 LOCAL I%, A$
180 A$=""
190 FOR I%=7 TO 0 STEP-1
200 A$=A$+CHR$(48+SGN(X% AND 2^I%))
210 NEXT
220 =A$
```

Listing I.1 - Joystick/Mouse test

Listing I.2

```
 10 REM Joystick Key Emulator
 20 REM (c) Dave Atherton 1987
 30 REM for Compact only
 40 REM MOS : A Dabhand Guide
 50 :
 60 bytev=&20A
 70 zp=&AF
 80 opt=2
 90 M%=&900
100 FOR pass=0 TO 3 STEP 3
110 P%=M%
120 [OPT pass
130 .patch
140 PHP
150 CMP #129 \ Is it INKEY
160 BNE notus
170 CPY #&FF \ Is it -ve INKEY
```

```
180 BNE notus
190 PHX
200 JSR bytecal \ Get key value
210 STX zp-1
220 PLX
230 LDA #0
240 STA zp
250 TXA
260 LDY #0
270 SEC
280 .loop
290 ROL zp
300 CMP keys,Y
310 BEQ found
320 INY
330 CPY #6
340 BNE loop
350 \ Not one of our keys
360 LDY #&FF
370 LDA #129
380 BRA notus \ and exit
390 :
400 .found
410 LDA #&1F
420 TRB &FE62
430 LDA &FE60
440 AND zp
450 PHP
460 LDA #0
470 PLP
480 BNE exit
490 DEA
500 .exit
510 ORA zp-1
520 TAX
530 TAY
540 LDA #129
550 PLP
560 RTS
570 .return
580 PLP
590 .notus
600 PLP
610 .bytecal
620 JMP (bytev2)
630 .keys
640 EQUB -74  \ RETURN
650 EQUB -98  \ Z
660 EQUB -105 \ ?
670 EQUB -73  \ *
680 EQUB -67  \ X
```

```
690 .bytev2
700 EQUD 0
710 ] NEXT
720 !bytev2=!bytev
730 ?bytev=patch MOD 256
740 bytev?1=patch DIV 256
750 PRINT "Testing for negative INKEY"
760 PRINT" (or joystick equivalent)"
770 REPEAT
780 FOR I%=1 TO 127
790 IF INKEY-I% PRINT"Detected INKEY -";I%
800 NEXT
810 UNTIL FALSE
```

Listing I.2 - Joystick key emulator

J : Connector Pinouts

This Appendix lists the pinouts of the new hardware ports on the Master Series micros. The Master Series cartridge ports are covered in Appendix H.

Master 128

The Master 128 has all the ports provided on the Model B fitted as standard. Additionally there are cartridge ports, and an internal Tube and internal modem connector.

Pinouts: Internal Tube Connector

There are two 12-pin sockets, SK1 and SK2, SK1 being the rightmost of the two on the PCB.

Pin	SK1	SK2
1	0V decoupled	0V decoupled
2	+5V decoupled	0V decoupled
3	CD7	0V decoupled
4	CD6	+5V decoupled
5	CD5	+5V decoupled
6	CD4	~INTUBE
7	CD3	~RST
8	CD2	02IN
9	CD1	A0
10	CD0	A1
11	8MHz IN	A2
12	~IRQ	BR/~W

Internal Modem Connector

There are two connectors, PL10, a 5 pin in-line socket and PL12, a 20 pin in-line socket. The pins are situated in a way that the connections physically support the modem board itself. The pins on the two associated plugs (PL10 and PL12) have the following functions:

PL10

Pin	Function
1	Speech out
2	MAN out
3	Analogue ground
4	Key way connector
5	Sound in

PL12

Pin	Function
1	~IRQ line from processor
2	~RST line from processor
3	Buffered D0
4	Buffered D1
5	Buffered D2
6	Buffered D3
7	Buffered D4
8	Buffered D5
9	Buffered D6
10	Buffered D7
11	~MODEM active low select line
12	A0 : Fixed hardware addresses in SHEILA
13	A1 : so only four address bits needed.
14	A2
15	A3
15	1MHZE
17	R/~W
18	+5V decoupled
19	-5V

Master Compact

The Master Compact has RGB Video and Composite Video (phono socket) as fitted on the BBC Micro. Econet and RS232 sockets are fitted although the hardware is an optional upgrade. There are no Tube, 1MHz bus, or Cartridge connections. There are four new connectors: the disc interface, printer connector, digital joystick, and expansion connector. The first two are electrically similar to earlier machines, but use different connectors. The other two more or less replace the User Port and Cartridge sockets, but again, the connectors are different. The edge connector is detailed in Appendix H.

New Disc Interface Connector: 25-way D-Type

Pin	Signal
1	~INDEX
2	~DRIVE 0
3	~DRIVE 1
4	~DRIVE 2 (NC if PL3 is open)
5	~MOTOR ON
6	~DIRECTION
7	~STEP
8	~WRITE DATA
9	~WRITE GATE
10	~TRACK 0
11	~WRITE PROTECT
12	~READ DATA
13	SIDE SELECT
14-25	0V

New Printer Connector: 24-way Amphenol

Pin	Signal
1	Strobe
2	D0
3	D1
4	D2
5	D3
6	D4
7	D5
8	D6
9	D7
10	ACK
11-12	NC
13-24	0V Ground

Digital Joystick Port: 9-way D-type socket

Pin	Signal
1	PB3
2	PB2
3	PB1
4	PB4
5	CB1
6	PB0
7	+5V
8	0V Ground
9	CB2/LPSTB

Notes: LPSTB is supplied if link PL7 is made. The logic lines are all from the User Port.

K : The Programs Disc

A disc of programs is available from Dabs Press containing all the programs listed in this book, plus several others, including a whole memory editor, a complete sideways utility ROM image (which you can run from sideways RAM or program into an EPROM) and much much more!

This is what you get:

* Shadow RAM teletext saver
* Sideways RAM screen saver
* Example programs for OSCLI, OSWORD etc.
* Disc sector ID reader (DFS)
* ROM copier and saver to disc
* On-screen Master clock
* Econet *SET utility - single program for Compact and Master
* Compact EEPROM version of the CMOS RAM editor
* All the listings in this book
* A sideways ROM utility pack
* Whole memory editor
* Useful manual of instructions
* Status display program
* 640 x 512 display program

Most of the programs are written in 6502 assembly language, and all of them are provided as readable source code, often with documentation in the listings.

The disc is available for the BBC B, B+ and Master 128 in 40 track 5.25" DFS format for just £7.95. A copy program will allow you to convert the disc for use with an 80 track drive. The software can also be transferred to Econet or a hard disc. It is also available in 3.5" ADFS format for Master 128 and Master Compact for £9.95. The disc is not copy-protected, and comes in a handsome illustrated plastic wallet with full instructions. The price is inclusive of VAT in the UK

and postage and packing. For orders abroad there is no VAT, but the price is the same to cover the higher postage.

To obtain your copy of the disc then use the order form below, or a copy of it or a letter, and send it with a cheque or postal order or sterling money order to Dabs Press at the address below. Government, education, and companies may send an official order. Dealer enquiries are welcome.

Please rush me a copy of the Programs Disc to accompany David Atherton's MOS Dabhand Guide. I require the following version (please delete as required):

5.25" DFS (£7.95) / 3.5" ADFS (£9.95)
Name
Address
Postcode
I enclose a cheque/PO for
Please charge my Access/Visa card no.
Expires.
Tick here if you require a VAT receipt
Send to : Dabs Press, 76 Gardner Road, Prestwich, Manchester. M25 7HU.

L : Guide to Dabhand Guides

The following Dabhand Guides and software packs covering the BBC and Master series micros are published or planned for 1988. Leaflets are available on all these products which go into considerably more detail than space here permits. Publication dates and contents may be subject to change. All quoted prices are inclusive of VAT (on software -books are zero-rated), and postage and packing.

VIEW: A Dabhand Guide by Bruce Smith

ISBN 1-870336-00-3
Publication: Available Now. 248 pages.
Book: £12.95. Disc: DFS 5.25" £7.95 ADFS 3.5" £9.95. Book and disc together £17.95 (ADFS £19.95)
This is the most comprehensive tutorial and reference guide written about using the VIEW wordprocessor. Both the beginner and the more advanced user will find it to be an invaluable companion whether writing a simple letter or undertaking a thesis. In addition a suite of VIEW utility programs are provided including, VIEW Manager, an easily extendable front end. Thorny subjects such as macros, page layout and printer drivers are revealed.

ViewSheet and Viewstore: A Dabhand Guide

by Graham Bell
ISBN 1-870336-04-6
Publication : January 1988. 280 pages
Book: £12.95. Disc: DFS 5.25" £7.95 ADFS 3.5" £9.95. Book and disc together £17.95 (ADFS £19.95)
Written by Acorn User's Graham Bell this book is a complete tutorial and reference guide for the Acornsoft ViewSheet spreadsheet and the ViewStore database manager. It is specifically written to appeal both to the beginner and to the more knowledgeable user, whether you wish to double-check your bank statement or run a million pound business. Every aspect of setting up and using a database or spreadsheet is described in detail, and numerous examples are provided to guide you. There are also a number of utility programs to help you get more out of the VIEW family, including programs that join two databases together and help transfer spreadsheets into a word processor.

Bumper Assembler Bundle by Bruce Smith

Publication : Available Now
Two books, two discs and booklet: Just £9.95
If programming in machine code is just wishful thinking, here is the chance for you to explore this fascinating subject with our Bumper Assembler Bundle. We are able to bring to you a five-part package of assembly language materials at less than a third of their normal price - all from the prolific keyboard of BBC expert, and former Acorn User Technical Editor Bruce Smith. Here's a summary:

1. BBC Micro Assembly Language is an excellent 200-page introduction and tutorial guide to 6502 assembly language programming. A separate edition is also available for the Acorn Electron. Normal price is £7.95.
2. A DFS format disc packed with 60 (yes sixty!) programs from the book BBC Micro Assembly Language.
3. BBC Micro Assembler Workshop is the companion volume and follows on from where BBC Micro Assembly Language leaves off. Its 160 pages contain programs such as a super fast machine code sort, and a machine code disassembler. Normal Price of this book is £6.95.
4. A DFS format disc packed with 30 programs from the book BBC Micro Assembler Workshop.
5. A booklet detailing all the new chip commands and innovations supplied with the Master 128 and Master Compact that have come to life since the above books have been published.

All together this bundle would cost you over £30. We can offer it to you at an exclusive price of just £9.95 including P&P and VAT. An ADFS version of the two discs is also available at the same price.

FingerPrint by David Spencer

Publication : Available Now
Disc and manual - DFS version £9.95, ADFS version £11.95
FingerPrint is a single-step machine code tracing program. It allows you to step through machine code written by yourself, part of a commercial program, or even the operating system, examining and executing each instruction, and displaying registers and flags at every stage. FingerPrint will even trace code situated in Sideways RAM/ROM— watch and learn how BASIC works for instance!

MOS Plus by David Spencer

Publication: Available now
ROM £12.95, Disc for Sideways RAM £7.95 (3.5" £9.95)
A useful ROM for all Master 128 users providing ADFS *FORMAT, *VERIFY, *BACKUP, *CATALL and *EXALL in ROM (you can put away that Welcome disc), and some new '*' commands such as *FIND which finds a file anywhere on an ADFS disc. A complete alarm system is present using the Master 128 alarm facility, as is an AMX mouse driver. The ROM also fixes the infamous DFS *CLOSE bug.

SideWriter by Mike Ginns

Publication: Available now
5.25" disc £7.95, 3.5" £9.95
For Master Series owners, and BBC owners with Sideways RAM, a pop-up notepad which can be used from within any application. Simply press SHIFT-CTRL-TAB and your program is suspended and you're in SideWriter ready to make a note. Press TAB and you're back with your application screen exactly as you left it. Notes taken in SideWriter can be saved to disc, transferred to a word processor, or printed out.

Archimedes Assembly Language: A Dabhand Guide

By Mike Ginns
ISBN 1-870336-20-8
Publication: March 1988 300pp approx.
Book: £14.95, 3.5" disc: £9.95 Book and disc together: £21.95
This book shows you how to get the most from the remarkable new Archimedes micro by programming directly in the machine's own language - machine code. This book covers all aspects of machine code/assembler programming for all Archimedes machines.
For those who are totally new to assembler programming, the book contains beginners' sections which take the reader step by step through topics such as binary numbers, and logic operations.
To make the transition from BASIC to machine code as painless as possible, the book contains a section on implementing BASIC commands in machine code. All of the most useful BASIC statements are covered.

Conversion Kit by Bruce Smith

ISBN 1-870336-06-2
Available now. Software pack with 16pp manual
DFS version £7.95 ADFS version £9.95
One of the most difficult aspects of machine code programming is writing routines to convert from one base to the other. How many times have you wished you could take a two byte binary number and print it to the screen as a five digit decimal number? Or perhaps you have longed for a routine that will do the reverse. All these problems, and many more are solved by the Conversion Kit, a disc containing 24 expert routines that you can use in your own programs. Each routine is held within a BASIC and assembler program as well as a procedure that demonstrates how to use it.

Master Emulation ROM by David Spencer

ISBN 1-870336-23-2 Available now
Software pack in ROM £19.95 (Disc for Sideways RAM £14.95)
This new ROM software is especially for Model B and B+ owners, and provides you with most of the features of the Master 128, such as the new '*' commands, the extended filing system operations including the temporary filing system, the *CONFIGURE system (using battery-backed Sideways RAM and/or a disc file), and if you have the hardware, Sideways or Shadow RAM. The only Master Operating System software not covered in this ROM is the extended graphics software. With this ROM you'll be able to use the new features documented in the Master Operating System, and for third-party shadow and Sideways RAM users, run some programs which are normally only intended for Masters.

The ROM recognises most popular types of Sideways RAM including Solidisk 16 /32/128k boards, Solidisk 2 Meg/4 Meg 256k boards, all single-chip plug in types, and BBC B+128k. Watford and Aries boards are recognised by the Shadow RAM control software.

Hyperdriver by Robin Burton

Available: Jan 88
Software pack in ROM £29.95 (Disc for Sideways RAM £24.95)
Hyperdrive is the ultimate printer utility ROM for word processor users. Unlike other ROMs, which are mostly graphical in nature,

Hyperdrive concentrates on text features, using simple '*' commands to control page length, margins, tabs and of course, text effects such as bold, italic, underlining and so on. These commands can be issued directly from languages such as BASIC, or embedded into wordprocessor files such as those created by Wordwise and InterWord.

When used with VIEW, the ROM really comes into its own. All the '*' commands are available in the form of a special VIEW Printer Driver. Microspacing is supported. Also, additional macro sequences are available, where a complex printer command may be defined in terms of a numbered definition, which can then be invoked anywhere in the document.

The ROM includes an on-screen effects generator, an inbuilt NLQ font and printing using screen defined characters.

C: A Dabhand Guide by Mark Burgess

ISBN 1-870336-16-X (Programs Disc 1-870336-22-4)
Available March 1988 300pp approx.
Price: £14.95

This is the most comprehensive introductory guide to the C programming language so far written, giving clear and comprehensive explanations of this important programming language. The book is packed with example programs, making use of all of C's facilities. Unique diagrams and illustrations help you to visualise programs and to think in C. Assuming only a rudimentary knowledge of computing in a language such as BASIC or Pascal you are provided with a grounding in how to build up programs in a clear and efficient way.

To help new users, a complete chapter on fault finding and debugging helps you to trace and correct both simple and complex errors.

Please note: All future publications are in an advanced state of preparation. Content lists serve as a guide but we reserve the right to alter and adapt them without notification. If you would like more information about Dabs Press books and software then drop us a line at 76 Gardner Road, Prestwich, Manchester, M25 7HU, and we'll despatch our latest Dabhand Guides Guide.

Glossary

Here is a list of the computer jargon words used in the book, plus a few other words commonly met in BBC Micro literature.

ADC Analogue-to-Digital converter. A chip (number uPD7002) fitted as standard to the Model B/B+ and Master 128 to convert variable voltages into digital values. There is no ADC on the Compact.

ADLC Advanced Data Link Controller. A chip (number 68B54) fitted as part of the Econet upgrade, which controls Econet signals to and from the machine.

AIV Advanced Interactive Video. The important difference between AIV and ordinary Interactive Video, is that the former can deal with analogue television pictures, and digital data at the same time.

ANDY The name given by Acorn to the 4K of RAM, mapped from &8000 to &8FFF, used by the MOS to store key definitions, etc.

Assembler A program contained within BBC BASIC which converts assembly language into machine code.

Assembly language A computer language vaguely like English that has a direct correspondence with machine code. One assembly language instruction will translate to one machine code instruction.

bank A section of memory in a paged memory system. Sometimes also called a slot or page.

BASIC 4 The Master Series version of BBC BASIC. In addition to all the features found in BBC BASIC versions 1, 2 and 3 (US BASIC), there are many new features such as EDITOR entry and TIME$.

BASIC 40 The standard Master Compact BASIC, also supplied with later Master 128 models contain MOS 3.22 or later. This is similar to BASIC 4 but contains vastly improved function evaluation resulting in higher floating point speeds.

baud This is a communications term meaning bits per second. A 1200 baud modem will transmit data at 1200 bits per second, which is about 120 characters (An 8-bit character transmitted down a telephone line requires two or three extra bits to cope with transmission errors).

BCD Binary coded decimal. A system whereby the processor works in decimal, ie, adding one to &89 will produce &90 and not &8A. The system is used to facilitate control and display of numbers up to 99.
bit A binary digit. A group of 8 bits is known as a byte.
buffer In software, an area of memory which temporarily holds data which is being produced by one source, but cannot yet be handled by the eventual destination - for example a printer.
Bus A set of wires containing a number of computer digital signals which are usually useful together, for example an address bus would contain 16 wires, each with one bit of an address.
CMOS Complementary Metal Oxide Semiconductor. A chip manufacture technology, which usually produces integrated circuits which run at very low power, but not as quickly as other technologies. In this book CMOS is only referred to as (a) to identify non-volatile RAM as opposed to normal RAM, and (b) to identify the advanced 65C02 CPU as opposed to the older 6502.
contiguous This word appears a lot in computer literature. It means 'next to each other'. For example, if 1024 bytes of memory are reserved for a buffer, it will usually necessary for this to be contiguous, ie all 1024 bytes follow each other in memory. DFS and ADFS files are said to occupy 'contiguous' disc space, whereas network files do not. This means that if the first 256 bytes of a file are on Sector 5/1 of a disc, you can guarantee that the next 256 bytes will be on Sector 5/2 and so on. On the network, and on disc filing systems such as CP/M, you cannot guarantee this.
CSD The currently selected directory.
CSL The currently selected library.
decrement This means 'subtract 1 from'.
directory A special file (on DFS a fixed place on disc) containing information about other files, specifically their name, length, load and execution addresses, and the address of the sectors where the file data is stored on the disc.
Econet The proprietary name of Acorn's local area network. This is a system where several computers are cabled together and data can be transferred between them. One disc drive and one printer can be

shared by every machine on the network, and data can be transferred between computers (known as stations).
EEPROM Electrically Erasable Programmable Read Only Memory. This is like an EPROM in that it retains data while the computer is off. Unlike an EPROM it can be altered while fitted to the computer.
EPROM Erasable Programmable Read Only Memory. A device which acts like a ROM but can be wiped and filled with different data using low-cost equipment known as EPROM erasers and EPROM programmers. Erasure involves exposing the EPROM to ultraviolet light. Programming involves applying a higher voltage than normal (typically 21 volts) to the EPROM. So while the EPROM is fitted to the computer it cannot be erased or programmed but will retain data when the computer is off.
F-Codes Control codes to drive a Laservision player.
FDC Floppy Disc Controller. This is the complex chip (Intel 8271 or WD1770) provided to handle disc drive operations.
FIFO First-in First-out. A buffering system where, as the name implies, the earlier items buffered reappear before the later ones. Most normal I/O device buffers are FIFO.
HAZEL The name given by Acorn to the 8K of RAM, mapped from &C000 to &DFFF, used by the MOS, ROMS and filing systems as private workspace.
IEEE This usually means IEEE488, which is a standards paper (No.488!) produced by the Institute of Electrical and Electronic Engineers. The interface standard is very popular in the laboratory although its use in business computers is declining.
I/O Input / Output. A computer does its work in the CPU, but only through the I/O section can we humans know what is happening. For example screen, printer, sound and disc are output. Keyboard, joystick and disc are input.
increment This means 'add 1 to'.
JIM The name given by Acorn to locations &FD00 to &FDFF on the I/O processor. These locations are not RAM but in fact 1MHz bus I/O lines.

Joystick An input device, principally designed for moving a cursor or other object round the screen. The device usually consists of a lever which can be moved forwards and backwards and left and right, with a button (called 'fire' from games usage). On machines other than the Compact, the device is analogue ie. the stick movement is measured by potentiometers connected to the A-D port. On the Compact, the device is digital and stick movement is measured by making switches.
Language processor This term means the co-processor in a two processor BBC Micro environment.
library A special type of directory, for machine code programs. *RUN "filename" will look in the CSL before the CSD, to find the required file.
LIFO Last-in First-out. This is a buffering system where the later items are retrieved first. This is not very useful for a I/O buffer, but ideal for a program stack, as used in general subroutines and particularaly in recursive work.
LVROM This is the name given to a digital 12" videodisc, such as that supplied with the BBC Advanced Interactive Video System.
LYNNE The name given by Acorn to the 20k of screen RAM from &3000 to &7FFF, also referred to as the shadow screen.
Master Sequence Number A number stored on a disc which is incremented by one every time the disc or directory is written to.
Modem A device which converts computer signals into audio tones for transmission along speech telephone networks. The device also converts the tones into signals ie it can transmit and receive. If two users have compatible modems they can transfer data between their computers via the telephone.
MOS Machine Operating System. See OS
Mouse A small box containing a rubber or steel ball which is rolled along a flat surface. A wire is connected to a port on the computer. Movement of the mouse causes signals to be sent to the computer which can be interpreted, say, as cursor movement.
NMOS Negative Metal Oxide Semiconductor. Another chip technology offering high density (ie a lot on a small chip). Most major components on the BBC Micro use this technology.

object In the context of this book, this term means 'a file or a directory'. When you say that ADFS can hold 47 objects in a directory, this means a total of 47 files OR sub-directories.
Operating System A computer program which deals with all the fundamentals of operating the computer - detecting keypresses, putting characters on the screen, loading and saving files (shared with the Filing System), generating sound, colour and graphics, and other useful features. Access to the operating system on Acorn machines is provided by a group of documented calls with entry points in page &FF.
OSHWM Operating System High Water Mark. The first free location in memory after the Operating System, and any other firmware such as filing systems have claimed workspace memory. The operating system normally leaves OSHWM at &0E00. On machines other than the Master series, the disc and network filing systems increase this figure.
polling Most computers will access their various I/O devices from time to time to see if anything needs to be done. Input devices may have some new data. Output devices may be ready for more. The process of checking the devices is known as polling.
Port A place on the computer circuit where outside connections can be made. This will usually, although not necessarily, mean that a socket is fitted also.
Printer A device which takes computer data and prints it onto paper (or other flat material). The type of data generally printed is text. This is the basis of word processing.
RAM Random access memory. Computer memory which can be read from and written to.
relocatable If a machine code program is described as this, it means that the program can be loaded into any memory area, and it will run regardless.
reset To reset a bit means to load it with 0. See also set.
Rockwell The company that manufacture and market the R65C02 microprocessor. This chip contains the BBR, BBS, RMB and SMB instructions.

ROM Read Only Memory. An integrated circuit (chip) which contains data or programs. The data or programs cannot be altered. They are retained when the computer is off. The Acorn MOS and BBC BASIC are contained in ROM in all Acorn computers.
RS232 A specification number. The specification in question was a (successful!) attempt to define a standard for data transmission using only 2 wires. RS423 is a later and more stringent specification of which RS232 is a subset. The system essentially is that data is transferred at a known and fixed rate (9 speeds are available), by changing the potential on one wire from +12v to -12v with reference to the other.
RTC Real Time Clock.
R/W stands for Read/Write. It usually means that a port can be read from and written to, unlike some which are read-only, or write-only.
SCSI Small Computer Systems Interface. A general interface standard much used for interfacing computers to large capacity storage devices such as hard discs and videodiscs. Also known as SASI.
SERPROC A name given by Acorn to their serial ULA. This controls the RS232 and cassette ports.
set To set a bit means to load it with a 1. See also reset.
SHEILA The name given by Acorn to locations &FE00 to &FEFF of I/O processor memory. These locations are not RAM but in fact I/O ports.
skewing When discs are formatted, a technique can be used where logical sector zero of a track is not exactly aligned with logical sector zero of the previous track, but is in fact two or three sectors adrift. This means that after a disc head has read, say Track 0 Sector 9, and wishes next to read Track 1 Sector 0, it can move its head across, and immediately catch that sector—unlike exact alignment where the head would usually miss the next sector, and thus have to wait for the disc to spin one more revolution. This method of formatting discs is known as skewing.
Teletext This is a term covering the broadcast videotex services of the BBC (Ceefax) and IBA (Oracle). It is also common parlance for the Mode 7/135 display mode on Acorn machines, as this is compatible with the broadcast display (and Level 0 Videotex generally).

Trackerball A box containing a ball which can be rotated in all directions by the user. Rotation sends signals to the computer which are usually interpreted as cursor movement instructions.
TTL Transistor Transistor Logic. A chip technology used for relatively simple chips such as decoders, logic gates, counters etc. TTL chips are much faster than CMOS and NMOS.
ULA Uncommitted Logic Array. A group of logic circuits which are connected together in a way chosen by the computer designer. A 'plank' ULA is made, and then the particular design is laid onto it. This is done for reasons of cost. There are two ULAs in BBC B/B+, eight in the Master, and seven in the Compact.
VIA Versatile Interface Adaptor. Usually refers to the 6522 chips in the BBC Micro. One is used to control much system I/O, and the other is used for the parallel printer port, and the user port.
VIDPROC A name given by Acorn to their video processor ULA.
Winchester A Winchester disc drive is sometimes also known as a 'fixed' or 'hard' disc, because the media is not removable from the drive casing, and the disc itself is rigid. Although you cannot change the media in a Winchester unit, the drives are of very high capacity. Acorn produce 10 and 30 Megabyte models.
The name Winchester comes from the project codename used by IBM when they worked on this technology.

Bibliography

Advanced Disc Investigator. R. Northen, Advanced Computer Products 1986.
Advanced Disc Toolkit. Rob Northen, Advanced Computer Products 1986.
Advanced Sideways RAM User Guide. Bruce Smith, Victory Publishing 1986.
Advanced User Guide for the Acorn Electron, Adder Publishing/Acornsoft
Advanced User Guide for the BBC Micro. Bray Dickens and Holmes, Cambridge Microcomputer Centre 1983.
BASIC ROM User Guide. Mark Plumbley, Adder Publishing 1984.
The BBC Micro ROM Book. Bruce Smith, Collins 1985.
BBC Microcomputer System User Guide. John Coll, BBC Publications 1981, Revised by Acorn Computers 1985.
Econet Advanced User Guide, Acorn Computers Ltd, Acorn 1983.
Master Reference Manual Part 1. Acorn Computers Ltd, Acorn 1986.
Master Reference Manual Part 2. Acorn Computers Ltd, Acorn 1986.
Mastering Assembly Code. Richard Vialls, BBC Publications 1986.
Mastering the Disc Drive. Chris Snee, BBC Publications 1986.
Monitor (Software). Graham Bartram, BBC Publications 1985.
R65C02 Data Sheet. Rockwell Instruments 1986.
HD146818 Data Sheet. Hitachi 1986.

absolute addressing … 28,29,79,94
ACCCON … 42,95,96,99,153
access … 13,14,112
accessing co-processor memory … 135
action on reset … 157,158
action on service calls … 157
active system … 152
ADC … 34,44,67
address - absolute … 79,94
- pseudo … 79,94
addressing - absolute … 29
- indexed … 33
- indexed absolute indirect … 43
- indirect zero page … 43
- modes new … 43
- pseudo … 28,29,54
ADFS … 7,14,15,16,17,18,19,20,22,24,30,61,83,85,91,99,120,147,149
ADFS changes … 180
ADVAL … 47,67
advising Tube OS of data transfer required … 136
analogue emulation … 67
ANDY … 91,97,134
ANFS … 52,118,121,148
APPEND … 14,153
ARGSV … 111,154
ARM … 132,137,176
ASCII … 11,15,19,26,31,47,69,70,98,120
ATPL board … 52
attributes … 14
auto boot … 111

back … 14
BACKUP… 15,112
Bad command … 24,28,54,112
BASIC … 15,34,91,95,105,135,141,172
- version 1 … 172
- version 2 … 173
- version 3 … 173
- version 4 … 38,174
- version 40 … 176
- version 5 … 176
baud … 30
BBR … 33,36,40

BBS … 33,36,40
BCD … 77,81
BRA … 37
BREAK … 24,66,70,72,73,93,95,96,112,121,122,124,125
BRK … 33,35,113,114
BRKV … 123,124
BUILD … 14,15,153

cambridge ring … 88
cartridge … 55
CAT … 15,22,150,156
CDIR … 16
CFS … 19,22,23,61,62,66,147,156
changes … 177
- ADFS … 180
- DFS … 180
- from original 6502 … 34
- to edit … 180
chip family … 33
claiming and releasing the Tube … 135
claiming the Tube … 135
CLD … 34
CLI … 11,21
Clock ROM … 125
CLOSE … 16,27,121
CMOS … 8,17,30,33,34,63,76,77,125,161
CMOS and EEPROM Memory Map … 161
CMOS RAM Editor … 165
CNPV … 140
co-processor … 32,33
- memory accessing … 135
CODE … 16,23,135
COMMANDS … 13
Compact Analogue Emulator … AppI
compatibility … 7
CONFIGURE … 17,31,73,112,161
connector pinouts … AppJ
COPY … 17
Country Flag … 71
CP/M … 25,88,132
CRC … 32,84,87
CREATE … 17
CRLF … 25
CSD … 14,18,22,23,24,111,156
CSL … 22,23,24,156
current filing system … 152

Dabhand Guides … AppL
date … 76
DEA … 37,126
DEC A … 37
Decimal Flag … 33
DELETE … 17,18,25
deselecting temporary filing systems … 159
DESTROY … 13,18
DFS … 7,12,15,16,17,18,20,23,30,31,32,54,86,99,132,147,151
- changes … 180
differences … 172
Directory … 17,30
DISC … 13,18,22,120
- changed … 18
DISK see DISC
DISMOUNT … 18
DRIVE … 19
DUMP … 19
dynamic workspace
- in HAZEL … 119
- requirements … 119

Econet see Net
EDIT changes … 180
EEPROM … 8,17,30,63,121,163,164
Electron … 20,22,50,56,57,58,67,90,95,96,132
ENABLE … 19
EOF … 149
EPROM … 92,105
EQUB … 36
ESCAPE … 14,15,19,59,134
EX … 19,22,150,156
EXEC … 19,23,117,150
execution address … 26
extended vector … 124

F-code … 82,83
FADFS … 20,24
FILEV … 111
Filing System … 123,147
- active… 152
- change … 117
- current … 152
- handler … 152
- information … 120, … 160

- library … 152,156
- restart … 118
- ROMS … 123
- temporary … 151

FINDV … 111,154
Flag - Decimal … 33
font implosion … 117
FORM … 20
FORMAT … 20
FRED … 98
FREE … 20,84
frugalising … 93
FSCV … 140,150,153,155,156
FX … 12,20,21
FX … 200,3 … 24

GO … 21,60
GOIO … 16,21
GSINIT … 140
GSREAD … 25,27,31,140

handle … 150,156
HAZEL … 24,80,91,93,97,118,119,123,152
HELP … 21,115,123
- interactive … 118

hex … 19,24
hierarchical directories … 18
high order addresses … 99
HIMEM … 16,60,80

I option … 28,93
I/O - memory from the co processor … 134
- port … 35
- processor … 135

ID marks … 32
IEEE … 88
IGNORE … 21
INA … 38,126
INC … 34
INC A … 38
indexed absolute indirect addressing … 43
indexed addressing … 33
indirect zero page addressing … 43
INFO … 13,19,22,150,156
initialise ROM … 28

INKEY … 58
INSERT … 22,31
INSV … 54,140
interactive HELP … 118
interrupts … 113,136
IRQ … 142

JIM … 98
JMP … 34,37,43,149
joystick … 47

language … 123
- entry point … 106
- ROM … 108,123
- startup … 122
LCAT … 22
LDA … 22,23,35,43,56
LEX … 22
LIB … 23
LIBFS … 22,148
library … 22,23,118
- filing system … 152,156
LINE … 23,135
LIST … 23,31,153
LOAD … 21,23,26,149
- address … 23,26
low level paging control … 95
LVDOS - see videodisc
LYNNE … 55,91,97,98,99

Machine Operating System see MOS
Macintosh … 10
making a disc image … 92
MAP … 23
MASM … 38
Master ET … 8
Master Series cartridges … AppH
Megabit ROM … 54
Memory Map … AppD
MHz bus … 55,97
Mode … 56
Model A … 72
modem … 85

MOS … 7,12,13,19,25,27,30,31,47,55,71,91,114

- calls … AppE
- commands … 11
MOTOR … 24,61
MOUNT … 18,19,24,79,82
mouse … 79
MOVE … 24
MS-DOS … 132
MSN … 81,83,85

names explained … 97
NET … 16,18,22,30,51,52,68,78,79
NETV … 140
network … 15
new MOS calls … 45
NFS … 132,148,149, also see NET
NLE … 11
NMI … 115,124,142
non maskable interrupt see NMI
Non-volatile RAM … 161
NOP … 137
Note - on unchanged OSBYTE calls … 74
- for 6502 co-processor programs … 140
numeric keypad … 73

opcodes … 33
Operating System see Machine Operating System
OPT … 25,149
OS see MOS
OSARGS … 154
OSBGET … 28,147,154
OSBPUT … 147,154
OSBYTE … 8,20,21,45,135,163
OSBYTE Calls
- &00 … 47,71
- &04 …47
- &09… 74
- &0A … 74
- &0F … 74
- &10 … 67,74
- &11 … 74
- &12 … 74
- &13 … 48,66
- &14 … 49,66
- &15 … 74
- &16 … 49,65
- &17 … 50,65

- &18 … 50,57
- &19 … 50
- &32 … 51
- &33 … 51
- &34 … 52
- &35 … 52
- &44 … 52,54
- &45 … 54
- &60 … 54
- &6B … 55
- &6D … 158
- &70 … 55,56,72,99
- &71 … 55,56,72,99
- &72 … 56,71
- &73 … 57
- &74 … 50
- &75 … 57
- &77 … 74,150
- &78 … 74
- &80 … 58,74
- &81 … 47,58,71
- &83 … 123
- &84 … 60,123
- &86 … 61,64,126
- &87 … 61
- &88 … 16
- &89 … 61
- &8B … 25,149
- &8C … 24,30,62,158
- &8F … 21,62
- &90 … 31
- &9A … 165
- &A1 … 161,163
- &A2 … 64,161
- &A3 … 64
- &A4 … 64
- &A5 … 64,126
- &A8 … 124
- &B3 … 65,125
- &B4 … 49
- &B6 … 66
- &BC … 67
- &BD … 67
- &BE … 67
- &C2 … 45
- &C3 … 45

- &C8 … 72
- &CD … 68
- &E1 … 22
- &E4 … 22
- &EA … 70,133
- &EB … 70
- &EE … 70
- &EF … 61,71
- &F0 … 71
- &F6 … 66
- &FA … 72
- &FB … 72
- &FE … 72
- &FF … 73

OSCLI … 11,112,149
OSEVEN … 140
OSFILE … 17,28,99,148,154
OSFIND … 15,16,27,28,147,154
OSFSC … 19,149
OSGBPB … 147,153
OSHWM … 15,16,28,49,59,65,80,117,122,140
OSRDCH … 69
OSRDRM … 140
OSRDSC … 140
OSVARS … 45
OSVDU … 140
OSWORD 8,75,114,134,135
OSWORD Calls
- &05 … 76,99,147
- &06 … 99,147
- &0E … 30,76
- &10 … 78
- &11 … 78
- &12 … 78
- &13 … 78
- &14 … 30,79
- &40 … 79
- &41 … 79
- &42 … 79
- &43 … 80
- &5F … 81
- &60 … 81
- &62 … 82,84
- &63 … 83
- &72 … 84

- &73 … 85
- &7A … 85
- &7B … 85
- &7D … 85
- &7E … 86
- &7F … 86,150
- &80 … 88
- &FE … 88

OSWRSC … 140

PAGE … 15,140
palette … 57
PANOS … 132
parameters … 12
PCB links … AppG
PHX … 38,126
PHY … 39,126
piracy … 92
PLX … 39
PLY … 40
PRINT … 25,31
programming considerations … 159
Programs Disc … AppK
prohibited filename characters … 150
pseudo address … 28,29,54,79,94
PTR … 147

Q parameter … 28,29

R65c02 … 33
R65c102 … 33
RAM … 21,26,28,35,98
reading data from the Tube … 137
registers … 137
releasing the Tube … 135
reload address … 27
REMOVE … 25
REMV … 54,140
RENAME … 25
reset … 121
RFS … 19,22,23,25,66,147,156
- data pointer … 116
- read Byte … 116

RMB … 33,40,41,42
Rockwell … 33

ROM … 8,13,22,25,30,49,85,92,95,105,121,141
- copyright string … 107
- header … 105,106
- header … 106,118
- polling … 118
- service calls … 108
- title … 107
- type byte … 107
- version number … 107
ROMSEL … 52,95,96,99
RUN … 26,60,149,150

safe commands … 32
SAVE … 17,26,148
SBC … 34,44
Second Processor … 7,147
sector … 23,32,87
Service Calls
- &01 … 110,118
- &02 … 111
- &03 … 111
- &04 … 112,157
- &05 … 113
- &06 … 113
- &07… 114
- &08 … 114
- &09 … 115
- &0A … 115
- &0B … 115
- &0C … 115
- &0D … 116
- &0E … 116
- &0F … 117,157
- &10 … 117
- &11 … 117
- &12 … 157
- &15 … 49
- &21 … 118
- &22 … 119
- &23 … 119
- &24 … 119
- &25 … 120
- &26 … 121
- &27 … 121
- &28 … 121
- &29 … 122

- &2A … 122
- &2B … 122
- &2C … 122
- &41 … 30
- &FE … 122
- &FF … 122

service entry point … 107
shadow … 8,27,55,56,90,95
shadow RAM … 90
SHOW … 22,27
SHUT … 16,27,121
Sideways RAM … 8,26,28,29,31,52,53,54,79,80,90,91,93,94,105,133,141
Sideways ROM … 105,113
SMB … 33,40,41,42
sound … 57
splitter … 32
SPOOL … 27,74,117,150
SPOOLON … 27,28,74
SRAM see Sideways RAM
SRDATA … 28,29,80
SRLOAD … 28,80,93
SRREAD … 29
SRROM … 29,80
SRSAVE … 29,80
SRWRITE … 29,93,95
STA … 56
star commands … 112, see also command name, ie, CAT
static workspace … 115,118
STATUS … 30,122,161
STY … 41
STZ … 41
summary of Tube registers … 141

TAPE … 30,62,66
Teletext … 17,30,85,91,120
temporary filing systems … 55,151
temporary filing systems deselecting … 159
Terminal … 54
Third Party … 64,79
TIME … 30,76
title … 30
top of static workspace … 119
track … 20,32,87
transferring control to the co-processor … 138
transient command … 152
TRB … 41,42

TSB … 42
Tube … 29,64,75,97,98,122,123,132
- claiming … 135
- data transfer … 134
- ESCAPE flag … 134
- Host Code … 132,133
- program notes … 140
- protocols … 143
- read data … 137
- releasing … 135
- write data … 137
Turbo … 33
TV … 31,62,90
TYPE … 23,25,31
type styles … 9

unchanged calls … 74
UNIX … 132
UNPLUG … 22,31,93
unsafe commands … 32
UPTV … 140
USA machine … 71
USERV … 16,23,75,114,135,140
using - Sideways RAM for data storage … 94
- Sideways RAM for ROMS … 92
VDU … 8,25,27,30,31,56,57,App C
VDUV … 140
vector … 124
- extended … 124
VERIFY … 13,32
VFS see Videodisc
Videodisc … 79,81,82,136
VIEW … 10

WIPE … 13
workspace … 28,29,93
- frugalising … 93
- dynamic … 119
- dynamic requirements … 119
- static … 115,118
- top of static … 119
write protect … 93
writing your own ROMS … 94

X … 32

Z80 … 80,133

65C02 … 33
- new addressing modes … 43
- new instructions … 36
65C12 see 65C02

A Dabhand Guide

This book is the definitive reference work for programmers of the BBC Model B+, Master 128 and Master Compact computers. It also contains much material of interest to BBC Model B and Electron users. The book covers all features of the Acorn machine operating system (MOS) including:

- All 'star' commands on all models
- 65C12 opcodes (including Rockwell additions)
- All new OSBYTE/OSWORD and other system calls
- Sideways and Shadow RAM programming
- ROM service calls (complete) and header code
- Driving the Tube in both directions

Also included is a complete list of differences between the various Acorn computers, and in one convenient place, all those vital tables that you need when programming your BBC computer. The Shadow and Sideways RAM and Tube chapters are expanded to provide application ideas, and the book is liberally sprinkled with program listings.

David Atherton was manager of BBC Soft for three years and is a regular contributor to Acorn User magazine and is widely respected as an authority on the BBC Micro.

'Serious users shouldn't be without their copy of this invaluable book'
David Somers, A&B Computing November 1987

£12·95
ISBN 1-870336-01-1