
ARM utilities

reference manual

ARM Evaluation System

Acorn OEM Products



ARM utilities

**Part No 0448,005
Issue No 1.0
1 August 1986**

© Copyright Acorn Computers Limited 1986

Neither the whole nor any part of the information contained in, or the product described in, this manual may be adapted or reproduced in any material form except with the prior written permission of the copyright holder. The only exceptions are as provided for by the Copyright (photocopying) Act, or for the purpose of review, or in order for the software herein to be entered into a computer for the sole use of the owner of this book.

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

- The manual is provided on an 'as is' basis except for warranties described in the software licence agreement if provided.
- The software and this manual are protected by Trade secret and Copyright laws.

The product described in this manual is subject to continuous developments and improvements. All particulars of the product and its use (including the information in this manual) are given by Acorn Computers in good faith.

There are no warranties implied or expressed including but not limited to implied warranties or merchantability or fitness for purpose and all such warranties are expressly and specifically disclaimed.

In case of difficulty please contact your supplier. Every step is taken to ensure that the quality of software and documentation is as high as possible. However, it should be noted that software cannot be written to be completely free of errors. To help Acorn rectify future versions, suspected deficiencies in software and documentation, unless notified otherwise, should be notified in writing to the following address:

Customer Services Department,
Acorn Computers Limited,
645 Newmarket Road,
Cambridge
CB5 8PD

All maintenance and service on the product must be carried out by Acorn Computers. Acorn Computers can accept no liability whatsoever for any loss, indirect or consequential damages, even if Acorn has been advised of the possibility of such damage or even if caused by service or maintenance by unauthorised personnel. This manual is intended only to assist the reader in the use of the product, and therefore Acorn Computers shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this manual, or any incorrect use of the product.

Econet® and The Tube® are registered trademarks of Acorn Computers Limited.

ISBN 1 85250 010

Published by:

Acorn Computers Limited, Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN, UK

Contents

1. Introduction	1
1.1 Conventions used in this manual	1
2. Operating system firmware	3
2.1 Introduction to the Executive ROM	3
2.2 The supervisor	4
2.3 ARM Second Processor memory map	7
2.4 Executive kernel	7
2.4.1 WriteC &00 (Acorn MOS OSWRCH)	8
2.4.2 WriteS &01	8
2.4.3 Write0 &02	8
2.4.4 NewLine &03 (Acorn MOS OSNEWL)	9
2.4.5 ReadC &04 (Acorn MOS OSRDCH)	9
2.4.6 CLI &05 (Acorn MOS OSCLI)	9
2.4.7 Byte &06 (Acorn MOS OSBYTE)	9
2.4.8 Word &07 (Acorn MOS OSWORD)	10
2.4.9 File &08 (Acorn MOS OSFILE)	10
2.4.10 Args &09 (Acorn MOS OSARGS)	11
2.4.11 BGet &0A (Acorn MOS OSBGET)	11
2.4.12 BPut &0B (Acorn MOS OSBPUT)	11
2.4.13 Multiple &0C (Acorn MOS OSGBP)	12
2.4.14 Open &0D (Acorn MOS OSFIND)	12
2.4.15 ReadLine &0E	13
2.4.16 Control &0F	13
2.4.17 GetEnv &10	14
2.4.18 Exit &11	15
2.4.19 SetEnv &12	15
2.4.20 IntOn &13	16
2.4.21 IntOff &14	16
2.4.22 CallBack &15	16
2.4.23 EnterSVC &16	17
2.4.24 BreakPt &17	17
2.4.25 BreakCtrl &18	17
2.4.26 UnusedSWI &19	17
2.4.27 SetCallBack &1B	18
2.4.28 WriteI &100	19
3. Acorn Object Format	20

3.1 Overall structure	20
3.2 Chunk file format	20
3.3 Object file format	22
3.4 The header chunk	22
3.4.1 Object file type	23
3.4.2 Version Id	23
3.4.3 Number of areas	23
3.4.4 Number of symbols	24
3.4.5 Entry Address Area/Entry Address Offset	24
3.5 Area declarations	24
3.5.1 Area name	24
3.5.2 AL (area alignment)	25
3.5.3 AT (area attributes)	25
3.5.4 Area size	26
3.5.5 Number of relocations	26
3.5.6 Base address	26
3.6 The areas chunk	26
3.6.1 Relocation	27
3.7 The symbol table chunk (OBJ_SYMT)	29
3.7.1 Name	29
3.7.2 AT	29
3.7.3 Value	30
3.7.4 Area name	30
3.8 The string table chunk (OBJ_STRT)	30
3.9 The identification chunk (OBJ_IDFN)	31
3.10 AOF-handling utilities	31
4. The linker	32
4.1 Link areas	32
4.2 Keywords	32
4.3 Examples	34
4.4 Diagnostics	35
4.5 Bugs	35
5. Machine code debugger	36
5.1 Starting the debugger	36
5.2 Debug input conventions	36
5.3 Debug commands	37
5.3.1 Named commands	37
6. Floating point emulation	41
6.1 Programmer's model	41
6.2 Floating register directives	42

6.3	Floating point constants	43
6.4	Load and store instructions	44
6.5	Conversion instructions	45
6.6	Move instructions	45
6.7	Compare instructions	46
6.8	Arithmetic instructions	47
6.9	The FPE and the Executive	49
6.10	Notes for advanced users	50
6.11	Floating point performance on ARM	51
7.	Miscellaneous utilities	52
7.1	Filing system utilities	52
7.1.1	List utility	52
7.1.2	Disc use utility	52
7.1.3	File/directory delete utility	53
7.1.4	Grope utility	54
7.2	Program development utilities	56
7.2.1	Comparing files	56
7.3	Software clock	56
7.4	Memory test	57
7.5	Column printing	57
7.6	The command command utility	59
7.7	AOF handling utilities	61
7.7.1	Chunk file decoder	61
7.7.2	Object file decoder	61
7.7.3	Merging two chunk files	62
7.7.4	m2run	63
8.	Appendix A	64

1. Introduction

This document is a reference guide to the ARM firmware (the Executive ROM) and to the various utility programs supplied on disc.

Chapter 2 deals with the Executive ROM, and includes sections on the routines contained within the supervisor, the use of the SWI calls, and the organisation of memory. Chapter 3 explains the structure of Acorn Object Format files. The remaining chapters deal with the utility software: the linker in chapter 4, the debug utility in chapter 5, the floating point emulator in chapter 6 and a group of miscellaneous utilities in chapter 7.

1.1 Conventions used in this manual

Most ARM source code has its own interpretations of the punctuation symbols and special symbols which are available from the keyboard. These are:

```
! " # $ % & ^ @  
( [ { } ] | : . , ;  
+ - / * = < > ? _
```

This often makes it difficult for the user to determine precisely which characters on the printed page are explanatory or descriptive, and which (if any) are the ones which AAsm will accept as having the correct syntax. A typewriter-style typeface has been used to indicate both text which appears on the screen and text which can be typed on the keyboard (for example, AAsm source code). This is so that the position of relevant spaces is clearly indicated.

Both general and specific examples of syntax and screen output is given – there are occasions where the full syntax of an instruction and its accompanying screen appearance would obscure the specific points being made. It follows therefore that not all the examples given in the text can be used directly since they are incomplete.

Curly brackets { } enclose optional items in the syntax. For example, AAsm accepts a three field source line which may be expressed in the form:

Function keys (such as f1) and control keys (such as tab) often need to be pressed by themselves or in combination with the shift and ctrl keys. To indicate this these keys are printed in boxes. For example:

Press the RETURN key **RETURN**

Press the ESCAPE key **ESCAPE**

Press the DELETE key **DELETE**

Press the COPY key **COPY**

2. Operating system firmware

2.1 Introduction to the Executive ROM

This chapter describes a set of supervisor calls linking ARM machine code to the Acorn 6502 MOS as used in the BBC Microcomputer. For further information on Acorn MOS refer to the User Guide for the BBC Microcomputer, the DFS or ADFS manual and the Advanced User Guide. Programs should assume the use of ADFS or ANFS if they wish to do something of file system specific nature. Refer to the *ARM CPU Software Manual* for details of the machine instruction set.

The ARM Second Processor consists of an ARM CPU, memory, timing circuitry and Acorn's Tube interface to the BBC Microcomputer. The BBC Microcomputer serves as an IO processor; it handles all the input and output devices (keyboard, rs423, text and graphics displays, printer, disc drives and so on). The BBC Microcomputer requires additional software from the DNFS ROM to deal with the Tube. The ARM Second Processor contains a ROM called the Executive kernal which deals with the ARM end of the Tube. The ROM also contains a set of useful utilities in the supervisor.

To start the system, switch on both the BBC Microcomputer and the ARM Second Processor. For the BBC Microcomputer to recognise the Second Processor it must be reset or switched on after the Second Processor has been switched on. The following message, or something similar, should appear on your screen:

```
Acorn ARM Second Processor 4096K
```

```
Acorn ADFS
```

```
A*
```

The 'A*' will appear either as A* on a blue background in mode 7 or as a large legend in any of the other modes: it is the prompt from the supervisor.

2.2 The supervisor

When nothing else is using the system, the supervisor gives its A* prompt and its built-in commands can be used: anything it does not understand will be passed to the Acorn MOS CLI. The built in commands are:

- **BreakClr**
Removes all breakpoints or just the one at the specified address. Puts the original contents back into the location.
- **BreakList**
Lists the currently set breakpoints.
- **BreakSet**
Set a breakpoint at an address.
- **Buff**
Turn file buffering on
- **Continue**
Start execution from breakpoint saved state. If there is a breakpoint at the continuation position, then a prompt will be given: reply Y if it is permissible to execute the instruction at a different address (that is, it does not refer to the pc).
- **Help**
Generates reassuring message from supervisor: gives version number of the kernal (this manual refers to kernel version -.008). `help supervisor` gives the list of commands that the supervisor accepts.
- **InitStore**
Fills all memory with &EE000000 or the specified data.
- **Memory**
Displays memory in ARM words from the address or register given to the next address or register given, a + meaning added to the first address. Default second address means 256 bytes displayed.
- **MemoryA**
Display and alter memory in bytes or words, signalled by an optional B or b. Given just an address/register it enters an interactive mode: **(RETURN)** to go to next location, - to go back one location, hex digits to alter a location and proceed, anything else to exit. Alternatively give the data required on the command line as well.

- **MemoryI**
Disassemble ARM instructions. Syntax as for memory. Given a limit it proceeds to the limit, otherwise it disassembles 24 instructions and waits for a key.
- **NoBuff**
Turns file buffering off
- **Quit**
Leave the supervisor by performing an SWI Exit. Any other supervisor will be returned to - or, of course, the supervisor itself.
- **ShowRegs**
Displays the registers caught on one of the four traps (unknown instruction, address exception, data abort, address abort)
- **Transfer**
Copies files from one file area to another. The syntax is `transfer argument`. Examples of arguments are:

```

disc adfs fred          copy fred from disc to adfs
net adfs fred jim      copy fred from net to adfs as jim
disc adfs *            all files in current disc directory
disc adfs              prompt for file names to be typed
dir@$.1 dir@$.2 *     all files in $.1 to $.2
net 'dir@&.1 adfs *    all files in net directory &.1 to adfs
    
```

The first two fields of the argument are simply interpreted by the command line interpreter or CLI, so they can cause quite a wide range of effects. The character @ in the first two fields of the argument will be replaced by a space character, the character ` in the first two fields of the argument represents a newline (multiple CLI commands) thus one can use transfer between net file servers, `transfer fs@1.254 fs@0.126 *`. Transfer has special code in it, so that, although it changes file systems, this does not cause the exec file to be lost.

- **Go**
Enter program at the address given. No address corresponds to `€1000`; R0 to R15 corresponds to the contents of the registers dumped on a trap. After the address a ; should precede the environment string. GO is, in fact, implemented at the Executive kernel level, so it still works from inside other systems. GOS enters the supervisor: the caller can be returned to if an Exit handler has been set using the Quit command.

Star commands, for example *CAT may be used by typing them directly to the A* prompt. It is not necessary to type the *. Refer to the MOS, DFS, ADFS documentation for details of a particular ROM facility.

Programs from the current filing system are executed simply by typing their names. Refer to documentation on the particular program for the parameters it may require. Usually programs will respond to the parameter `-help`.

The A* prompt is created by redefining character 255. Spooling is disabled while this prompt is output using `fx3` calls. The character 255 is left as a solid block. With the ARM Second Processor connected the entire character set has been exploded, see description of `fx20` in the User Guide.

With the Second Processor connected addresses are used to distinguish between the memories of the two machines. The ARM Second Processor can use addresses up to `Ramtop` (see section 3). The BBC Microcomputer uses addresses `FFFxxxx`. The DFS filing system only has 18-bit addressing; any address greater than `2FFFF` is assumed to be in the BBC Microcomputer: use of ADFS or the net filing systems avoids any problems of restricted addressing.

2.3 ARM Second Processor memory map

Executive ROM (16K)		
-----	-----	&3000000
nothing: aborts are caused		
-----	-----	&2000000
tube IO chip		
-----	-----	&1000000
repetitions of the RAM area		
-----	-----	Ramtop
file buffer RAM		
-----	-----	Ramtop-20K
user RAM		
-----	-----	&1000
system RAM: stacks etc.		
-----	-----	&0D00
system RAM: Executive kernel itself		
-----	-----	&0000

Programs should load in memory at &1000 or higher. If a program is to be run at an address outside available memory, the Executive will try wrapping it around in available memory: the TWIN program is supplied to run at &1D0000: in a 1/4 Mbyte machine it thinks it has been run at &10000, in a 1 Mbyte machine it thinks it has been run at &D0000. For a 4 Mbyte machine it could be moved to &3D0000, however, due to the hardware design of the 2 and 4 Mbyte machines this would cause it not to work on the 2 Mbyte machine, although it would be fine with the full 4 Mbytes. Programs should only use memory up to the Ramtop limit returned by the GetEnv call, on an otherwise empty machine the Excutive kernel uses the top 20K for file buffers.

2.4 Executive kernel

All BBC Microcomputer IO is accessed through SWI calls.

The following information documents all the Executive kernel calls that a program can make using the SWI instruction. A, X, Y represent the 8 bit 6502 registers referred to in the Acorn MOS documentation, c the 6502 carry flag and the ARM carry flag. R0 and so on are ARM registers. (string) is an indirect pointer to a string terminated by hex &00, &0A or &0D. R0b means lsb of R0 (only bottom 8 bits used as an input parameter, top 24 bits zeroed on result). R0 means all 32 bits of R0. All successful SWIS will clear the ARM v flag.

2.4.1 WriteC &00 (Acorn MOS OSWRCH)

Write R0b to the terminal output. For example:

```
MOV R0,#"H"  
SWI WriteC
```

```
In: R0b  
Out:
```

2.4.2 WriteS &01

Write the bytes following the call to the terminal output. Terminates at first zero and starts execution at the next 32bit word. For example:

```
SWI WriteS  
= "Hello World",10,13,0  
ALIGN  
SWI WriteI+"K"
```

```
In:  
Out:
```

2.4.3 Write0 &02

Write the bytes pointed to by register 0 to the terminal output. Terminates at first zero. R0 points to the byte after the zero on exit. For example:

```
ADD R0,PC,#data--8
    SWI Write0
```

In: R0

Out: R0 updated

2.4.4 NewLine &03 (Acorn MOS OSNEWL)

Write a line feed (&0A) followed by a carriage return (&0D) to the terminal output. For example:

```
SWI NewLine
```

In:

Out:

2.4.5 ReadC &04 (Acorn MOS OSRDCH)

Read a character and validity from the terminal input. C set if an unusual character (for example Escape) is read. For example:

```
SWI ReadC
    BCS Escape
```

In:

Out: R0 contains character; C contains validity

2.4.6 CLI &05 (Acorn MOS OSCLI)

Interpret a string as a command. The string is checked for HELP (or valid abbreviations) and an appropriate message issued. An additional command over those in the IO processor MOS is the GO command, see the data about the supervisor. For example:

```
SWI CLI
```

In: R0 pointing to string

Out: may not return if another
program has been executed

2.4.7 Byte &06 (Acorn MOS OSBYTE)

Do an Acorn MOS OSBYTE call with A=R0b, X=R1b, Y=R2b, C=C. For calls with R0b less than 128 the R2 register is not required or altered. For example:

```
MOV R0,#5
      MOV R1,#4
      SWI Byte ;select network printer
```

In: R0, R1, R2
Out: R0, R1, R2, C

Note: because of the read only file buffering in the Executive the OSBYTE &7F call checks R1 WORD for being greater than 256 (range of fast handles is 257 to 511).

2.4.8 Word &07 (Acorn MOS OSWORD)

Do an Acorn MOS OSWORD call with A=R0b and a parameter block pointed to by R1. Note that call with R0b=0 (Acorn MOS RDLN) does nothing, the ReadLine call should be used instead. For example:

```
MOV R0,#1
      SUB R1,SP,#5
      SWI Word ;read time
```

In: R0, R1 pointing to parameter block
Out: parameter block updated.

2.4.9 File &08 (Acorn MOS OSFILE)

Do an Acorn MOS OSFILE call with A=R0b. Instead of a parameter block R1 to R5 contain the data (string pointer, load address, exec address, start address {*length*}, end address {*attributes*}). For example:

```
MOV R0,#5
      MOV R1,#ptr
      SWI File ;file info
```

In: R0, R1 pointing to string, R2, R3, R4, R5
Out: R2, R3, R4, R5 updated.

2.4.10 Args &09 (Acorn MOS OSARGS)

Do an Acorn MOS OSARGS call with A=R0b, file handle in R1, data value in R2. For example:

```
MOV R0,#0
      LDR R1,handle
      SWI Args ;read ptr to R2
```

In: R0, R1 (handle), R2
Out: R2 updated.

2.4.11 BGet &0A (Acorn MOS OSBGET)

Read the next byte from the file whose handle is in R1. Byte returned in R0, validity in C (set if end of file). The Executive does local buffering for Input Only and Output Only files by using handles in the range 256 to 511 - see section 2.4.14. For example:

```
LDR R1,handle
      SWI BGet
      BCS EndOfFile
```

In: R1 (handle)
Out: R0, C.

2.4.12 BPut &0B (Acorn MOS OSBPUT)

Write R0b to the file whose handle is in R1. The Executive does local buffering for Input Only and Output Only files by using handles in the range 256 to 511 - see section 2.4.14. For example:

```
MOV R0, #data
      LDR R1, handle
      SWI BPut
```

In: R0b, R1 (handle)
Out:

2.4.13 Multiple &0C (Acorn MOS OSGBP)

Read and write multiple bytes from file whose handle is in R1. Control in R0b; addresses in R2 (data pointer), R3 (number of bytes), R4 (pointer in file, if required). C is set if the transfer could not be completed. Executive local buffering does not apply to the multiple call. For example:

```
MOV R0, #1
      LDR R1, handle
      MOV R2, #data
      MOV R3, #56
      MOV R4, #100
      SWI Multiple ; put 56 bytes to
                  ; file from 'data' at offset 100
```

In: R0b, R1 handle,
R2 points to data,
R3 number of bytes,
R4 position

Out: R2, R3, R4 updated.
C set if transfer past end of file.

2.4.14 Open &0D (Acorn MOS OSFIND)

Open and close a file. If R0b=0 then the file whose handle is in R1 is to be closed; if R1b=0 then all files on the current filing system are closed. If R0 is not zero a file, whose name R1 is pointing at, is to be opened, (&40 for Input only, &80 for Output, &C0 for Update); the file handle being returned in R0. The Executive does local buffering for BGet for Input Only and for BPut for Output Only files by using handles in the range 256 to 511: this feature can be ignored by masking out the extra bit in the handle, but you must not mix use of the masked and unmasked handles. Alternatively the command NOBUFF can be used at the supervisor prompt (NOT as a CLI call). The buffering produces the following times for BGet-ing a 64K file:

ANFS	ADFS	Executive+ANFS	Executive+ADFS
40sec	32sec	10sec	4sec

For example:

```
MOV R0, #&40
      MOV R1, NameAddress
      SWI Open
```

In: R0, R1 (handle/ pointer to name)

Out: R0 (handle if opened)