# Rom PODULE Guide

Acorn

The product described in this guide and products for use with it are subject to continuous development and improvement. All information of a technical nature and particulars of the product and its use (including the information and particulars in this guide) are given by Acorn Computers Limited in good faith. However, it is acknowledged that there may be errors or omissions in this guide or in the products it describes. Acorn Computers welcomes comments and suggestions relating to the product and this guide.

All correspondence should be addressed to:

Customer Support and Services,
Acorn Computers Limited,
Cambridge Technopark,
645 Newmarket Road,
Cambridge CB5 8PB.

All maintenance and service on the product must be carried out by Acorn Computers authorised dealers. Acorn Computers can accept no liability whatsoever for any loss or damage caused by service, maintenance or repair by unauthorised personnel. This guide is intended only to assist the reader in the use of this 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 guide, or any incorrect use of the product.

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



First published 1987
Issue 1 1987
Published by Acorn Computers Limited
Part number 0476,220

# INTRODUCTION

The ROM Podule provides a ROM filing system for use on the Archimedes. It contains five vacant sockets into which 16kbyte, 32kbyte, 64kbyte ROMs/EPROMs or 128kbyte JEDEC EPROMs may be plugged. Each EPROM/ROM can contain several files of varying size. The ROM Manager, which is fitted on the board, provides a series of both high- and low-level commands for accessing the data in these files.

In conjunction with the 6502 Emulator, the ROM Podule allows certain ROM based programs which are available for the BBC Microcomputer Models B and B+ and the BBC Master Series Microcomputers to be used. For a ROM to be able to be read in this way it must be unprotected and contain only `legal' 6502 machine code.

Two additional sockets are provided for static RAM. Optional components may be added to provide battery backup to these RAM sockets. This upgrade is described in detail in this guide. It allows the data in the RAM to be retained when the power is switched off in a similar manner to the battery backed CMOS RAM in the Archimedes microcomputer.

Instructions for fitting the ROM Podule are given in a separate *Podule Installation Leaflet* provided as part of this package. In order to install a Podule into the Archimedes, it is necessary to have a backplane fitted. The backplane is available separately and full instructions for installation are supplied with it. If you do not feel confident about fitting the Podule or backplane, then take your Archimedes and the upgrade to your dealer who will fit it for you.

# INSTALLING EPROMS AND ROMS

## THE ROM SOCKETS

The ROM Podule contains six sockets for EPROMs/ROMs, ICI, IC2, IC3, IC4, IC5 and IC6. ICI contains the ROM Manager program which must always be present. IC2 - IC6 are free for your use. Each of the sockets can hold an EPROM or ROM of any of the following sizes:

| Device | Size | Number of pins |
|---|---|---|
| EPROM/ROM | 128kbit (16kbyte) | 28 |
| EPROM/ROM | 256kbit (32kbyte) | 28 |
| EPROM/ROM | 512kbit (64kbyte) | 28 |
| JEDEC EPROM | 1024kbit (128kbyte) | 32 |

WARNING: Most electronic devices can be damaged by static electricity. Keep the EPROM/ROM in anti-static foam until fitted and avoid touching the pins of the EPROM/ROM during fitting.

An EPROM/ROM should be inserted as follows:

1 Remove the Podule and place it on a flat surface.

2 Check that all the pins on the EPROM/ROM are straight.

3 Hold the EPROM/ROM so that its notch points towards the notch in the socket.

4 If the EPROM/ROM contains only 28 pins, line up these pins so that they coincide with the pin positions in the socket as far away from the notch as possible.

5 Carefully insert the EPROM/ROM in the socket.

The EPROM containing the ROM Manager program can be used as a guideline as to how EPROMs/ROMs should be fitted.

## SETTING THE LINKS

Each socket has two links associated with it, LK1 and LK2. Each link consists of three pins and a plug which can be moved so that it links the middle pin with either of the outer pins, pin A or pin C. These plugs must be placed correctly for the size of the EPROM/ROM in the socket. The choice of which outer pin to connect to depends on the device size as follows:

| Device Size | LK1 | LK2 |
|---|---|---|
| 128kbit (16kbyte) | A | C |
| 256kbit (32kbyte) | C | C |
| 512kbit (64kbyte) | C | A |
| 1024kbit(128kbyte) | C | A |

## INSTALLING THE PODULE

Instructions on how to install the ROM Podule into your Archimedes are contained in a separate *Podule Installation Leaflet* supplied with this package.

# CONFIGURING THE SYSTEM

When you have fitted your ROM Podule and turned the power back on, the next step is to configure the system so that it knows the size and type of each EPROM/ROM on the ROM Podule. To do this, exit the Desktop if you are in it and use the following command:

```
*CONFIGURE ROMBOARD <p> <d> [<t>l
```

where <p> is the Podule socket (0 - 3)
<d> is the ROM socket number (1 - 6)
and <t> is the device type (none, if not present)

To find the number of the Podule socket either look at the number printed on the backplane or use the *PODULES command.

The other two sockets on the board, 7 and 8, are for RAM only. See the chapter entitled *Static RAM Upgrade* for details on how to configure these.

The device type may take the following values:

| EPROM/ROM type | <t> |
|---|---|
| 128kbit (16kbyte) | 10128 |
| 256kbit (32kbyte) | 10 256 |
| 512kbit (64kbyte) | 10 512 |
| 1024kbit(128kbyte) | 101024 |

The '10_' is used to denote an EPROM/ROM as opposed to RAM and is optional.

Each ROM socket is assumed to be empty. Hence, this command must be issued for every EPROM/ROM on the Podule, including the ROM Manager EPROM. This is a 128kbit EPROM which is fitted in ROM socket one, so if the ROM Podule is fitted in Podule socket zero, type:

```
*CONFIGURE ROMBOARD 0 1 10 128
```

to configure this ROM. Failure to do so will result in the commands provided by the ROM filing system not being available.

Repeat this procedure for each EPROM/ROM in turn and then press Ctrl Break to make the new configuration take effect. The ROM Podule configuration is stored in the battery-backed CMOS RAM in the Archimedes so it is only necessary to reconfigure it if:

• the ROM Podule is moved to a different Podule slot

• an extra device is added to the board

• an existing device is replaced by one of a different size

• an existing device is removed completely

• the configuration settings are lost, eg if the batteries are removed or an R-power on is performed.

To check that the board has been fitted correctly type:

```
*PODULES
```

A list of the Podule slots, zero to three, is displayed, together with the name of any Podule held in each slot. The ROM Podule should be declared as being present in one of these slots. For example, if you only have the ROM Podule fitted and you have placed it correctly in slot zero, the following is displayed:

```
Podule 0 : ROM podule
Podule 1 : No installed podule
Podule 2 : No installed podule
Podule 3 : No installed podule
```

To check which files you have on the EPROMs/ROMs plugged into the ROM Podule, type:

```
*ROM
```

to select the ROM filing system (ROMFS). Then type:

```
*CAT
```

to obtain a list of the names of the ROM files in order. For example, if you have fitted VIEW 2.1 only, the following is displayed:

```
ROMIT
VIEW
```

To check the configuration being held in CMOS RAM for each of the sockets type:

```
*STATUS ROMBOARD
```

The following type of display should be obtained:

```
Podule 0
   Device 1 = 64K byte EPROM
   Device 2 = 16K byte EPROM
   Device 3 = Empty socket
   Device 4 = Empty socket
   Device 5 = Empty socket
   Device 6 = Empty socket
   Device 7 = Empty socket
   Device 8 = Empty socket
```

# USE WITH THE 6502 EMULATOR

Some of the ROM-based software available for the 6502-based BBC Microcomputers can be run using the ROM Podule and the 6502 Emulator. To run a particular program, for example VIEW 2.1, follow the steps given below:

1 Install the ROM Podule and the VIEW ROM as described in the chapter Installing *EPROMS* and *ROMS*.

2 Configure the system to recognise the ROM Manager and the VIEW ROM as described in the chapter Configuring *the System*.

3 Place the Welcome Disc in drive 0 and type:

```
*ADFS
```

to select the Advanced Disc Filing System.

4 Type:

```
*:0.$.MODULES.65Arthur
```

This loads and enters the 6502 Emulator running BASIC IV, producing a screen display similar to the following:

```
6502 Emulator 0.82 (01 Jun 1987)

Acorn 6502 TOS 0.40 (11 Jun 1987)

BASIC
```

5 Type:

```
*ROM
*GO F800
```

This enters the ROM filing system and then enters the Emulator Command Line Interpreter (CLI). The CLI prompt is displayed as follows:

```
65*
```

6 Type:

```
LOAD VIEW 8000
GO 8000
```

This loads the VIEW code from the ROM into memory at address &8000 and then executes the code starting at this address. The usual VIEW message is given, for example:

```
VIEW A2.1

No text
Editing No File
Screen Mode 0
Printer default
```

You can now use the VIEW word processor in the normal way.

10

# THE FILING SYSTEM

## FILE ORGANISATION

### Filenames

The ROM filing system (ROMFS) supports filenames containing up to 64 characters. However, many commands within the rest of the operating system restrict the length to ten.

A typical filename is, for example:

```
ProgI
```

### Directories

ROMFS does not have a true hierarchical directory structure but the effect can be simulated through the use of filenames containing the fullstop character. Thus, a filename can consist of a number of filename components separated by fullstops, provided that the total length is less than 64 characters. For example:

```
graphics.circles
```

refers to a file called `circles' in the directory `graphics'.

In cases such as this, each component of the filename should be restricted to ten characters.

Nested directories can be created and a filename such as `BASIC.graphics.circles' could be used.

In commands which take a directory name as an argument, the following characters have a special meaning:

| Character | Meaning |
|---|---|
| $ | top level |
| & | top level |
| @ | all levels |

## ENTERING THE ROM FILING SYSTEM (ROMFS)

*ROM

Syntax: *ROM

This command selects ROMFS as the current filing system.

## ROM FILING SYSTEM COMMANDS

The following commands are recognised by all filing systems. However, some of these commands attempt to write to files or perform operations on the current or library directory. Since ROMFS is a read-only system and does not have a true directory structure, use of these commands under ROMFS is inappropriate and either produces an error message or has no effect.

In addition, ROMFS does not generally support wildcarding in filenames. Unless it is specifically stated that wildcard characters may be used, the `*' and `#' characters will not be interpreted as wildcards.

Under ROMFS only one file may be open at a time.

All commands are listed for completeness. Where their use is not supported the effect of issuing them is stated. Otherwise, a full description of their syntax and the effect of the command under ROMFS is described.

*ACCESS

This command is accepted but ignored. All files under ROMFS have access attributes LR (locked and readable) and directories have attributes DL (directory and locked).

*APPEND

This command tries to write to ROMFS and therefore gives an error.

*BUILD

This command tries to write to ROMFS and therefore gives an error.

*CAT

Syntax: *CAT [<directory>]

*CAT displays a list of all objects in a directory. If no directory is specified all objects are listed.

*CDIR

This command tries to write to ROMFS and therefore gives an error.

**COUNT*

*Syntax:* *COUNT <file spec> [<options>]

*COUNT adds up the sizes of all the files which match the wildcarded specification given, provided that the file name contains no more than ten characters. The options available are:

| Option | Default | Description |
|---|---|---|
| C(onfirm) | off | prompt for confirmation of each count |
| R(ecurse) | on | count subdirectories and contents |
| V(erbose) | off | print information on each file counted |

To reverse the effect type '~' before the option.

*CLOSE

*Syntax:* *CLOSE

*CLOSE closes all files opened by ROMFS.

*COPY

*Syntax:* **COPY* <source> <destination> [<options>]

*COPY copies files matching the specification of the first (source) parameter, to the place specified by the second (destination) parameter. Options may be given as follows:

| Option | Default | **Description** |
|---|---|---|
| C(onfirm) | on | prompt for confirmation of each copy |
| D(elete) | off | delete the source after copy |
| F(orce) | off | force overwriting of existing objects |
| Q(uick) | off | use application workspace to speed transfer |
| V(erbose) | on | print information on each file copied |

To reverse the effect type '~' before the option.

For example:

```
*COPY ROM:COMAL ADFS::0.$.COMAL ~C
```

*CREATE

This command tries to write to ROMFS and therefore gives an error.

*DELETE

This command tries to write to ROMFS and therefore gives an error.

*DIR

ROMFS does not have a true directory structure therefore this command gives an error.

*DUMP

Syntax: *DUMP <filename> [<file-offset> [<start address>]]

*DUMP displays a hexadecimal and ASCII dump of the named file in the following format:

address hexadecimal bytes ASCII characters

`.' is used to represent any non-printable characters.

The number of bytes displayed per line depends on the current window size; for example, in an 80 column window, 16 bytes per line are displayed, in a 132 column window, 24 bytes per line are displayed.

The file offset specifies the point in the file at which the dump is to start; this defaults to zero. The start address is used to determine the address printed on each line of the display, and is the address which would be displayed if the file offset were zero. If this is not present, then it defaults to the load address of the file, unless the file is date/time stamped, in which case it is taken to be zero.

*ENUMDIR

Syntax: *ENUMDIR <directory name> <filename> [<wildcard pattern>]

*ENUMDIR creates a new file and places in it a list of filenames. The name of the new file is given in <filename>.

Any file in the directory given is only added to the list if it matches the wildcard pattern. `*' may be used to denote any number of any characters and `#' any single character. The default pattern is *.

Entries are separated by ASCII 10 (line feed) in the output file.

For example:

```
*ENUMDIR BASIC ADFS::4.$.Graphic DRAW*
```

Or

```
*ENUMDIR $ ADFS::4.$.ROMCat
```

*EX

Syntax: *EX [<directory>]

*EX displays file catalogue information for all objects in the directory given. If no directory is specified all objects are listed. The information supplied is as follows:

load address exec address length S.I.N. attributes filename

All values are given in hexadecimal notation. S.I.N. is the `System Internal Name' and is made up of the Podule number in the top byte and the start address of the file in the lower three bytes.

For example:

```
FFFFFB40 899EC4A9 000590 00000000 LR ROMIT
FFFBBCOB BCBBCBBC 004000 00002000 LR VIEW
```

Note that for 6502 ROMs the load and execution addresses are presented as shown in the example above. The file type component is thus &BBC and this is recognised by the 6502 Emulator.

*EXEC

Syntax: *EXEC [<filename>]

*EXEC uses data from a named file, byte by byte, as though it were being typed from the keyboard. Data from this source is taken in preference to that from the current input stream, until the file is exhausted, when it is closed. If no parameter is given to *EXEC then the current exec file is closed. If another EXEC command is issued with a filename parameter, then the current exec file is closed and the exec action continues with data from the newly opened file.

*INFO

Syntax: *INFO <filename>

*INFO displays the same filing system information as *EX but for a single file.

*LCAT

ROMFS does not support libraries, therefore this command gives an error.

*LEX

ROMFS does not support libraries, therefore this command gives an error.

*LIB

ROMFS does not support libraries, therefore this command gives an error.

*LIST

Syntax: *LIST <filename>

*LIST displays the content of the named file in GSREAD format, in which:

- ASCII codes 32 to 126 (except solidus) are sent directly to the screen as the corresponding character in the current mode.

- all ASCII codes other than newline or carriage return are displayed as control code expansions, controlled by the current DumpFormat.

Each line (ie sequence of ASCII codes terminated by a carriage return, line feed or pairs of the above) is preceded by a line number, starting from one.

*LOAD

Syntax: *LOAD <filename> [<load address>]

*LOAD loads the specified file into memory. The address it is loaded at can be specified as a hexadecimal (or other given base) value after the filename. Otherwise, the load address supplied by the filing system will be used.

*OPT

Syntax: *OPT <option number [<option value>]

*OPT sets up various filing system options:

*OPT 1 control the display of file information during load as follows:

| Value | Meaning |
|---|---|
| 0 | suppress file information |
| 1 | display filename |
| 2 | display filename, load and execution addresses and length |
| 3 | as 2 with load and execution addresses interpreted as file type and date/time stamp if possible |

*OPT 4 Control the auto-start option for ROMFS as follows:

| Value | Meaning |
|---|---|
| 0 | disable the auto-start facility |
| 1 | *LOAD the !BOOT file |
| 2 | *RUN the! BOOT file |
| 3 | *EXEC the !BOOT file |

This option is held in the CMOS RAM of the Archimedes.

*PRINT

Syntax: *PRINT <filename>

*PRINT displays the contents of the named file in ASCII format. Each byte is sent to the VDU driver regardless of whether it is a printable character or a control character. Hence, unless the file is a simple text file, this command can produce undesirable results. It may be used to `replay' spooled graphics output.

*REMOVE

This command tries to write to ROMFS and therefore gives an error.

*RENAME

This command tries to write to ROMFS and therefore gives an error.

*RUN

Syntax: *RUN <filename> [<parameters>]

*RUN loads the named file into memory and starts execution. It uses the load and execution addresses stored by the filing system. The optional parameters may be accessed by the program itself.

*SAVE

This command tries to write to ROMFS and therefore gives an error.

*SETTYPE

This command tries to write to ROMFS and therefore gives an error.

*SHUT

Syntax: *SHUT

*SHUT closes all open files. It is similar to *CLOSE except that it affects all filing systems, rather than just the current one.

*SHUTDOWN

Syntax: *SHUTDOWN

*SHUTDOWN closes all open files on all filing systems. It also logs off file servers (on NetFS) and causes hard discs to be parked (on ADFS).

*SPOOL

This command tries to write to ROMFS and therefore gives an error.

*SPOOLON

This command tries to write to ROMFS and therefore gives an error.

*STAMP

This command tries to write to ROMFS and therefore gives an error.

*TYPE

Syntax: *TYPE <filename>

*TYPE is similar to *LIST in that it displays the content of the named file in GSREAD format. However, it does not precede each line with a line number.

*UP

ROMFS does not have a true directory structure therefore this command gives an error.

*WIPE

This command tries to write to ROMFS and therefore gives an error.

## LOW LEVEL CALLS

In addition to the * commands documented above, lower level calls to the operating system routines can be made to access the files. These calls can be made from BASIC by using the SYS statement or from assembler by calling the appropriate SWI. The calls which are supported by ROMFS are listed below. More details about these and other operating system routines can be found in the *Programmer's Reference Manual* which is available separately.

### OS_Find - OPEN OR CLOSE A FILE FOR BYTE ACCESS

OS_Find is used to open and close files. Opening a file declares that byte access to the filing system is required. Closing a file declares that byte access is complete. Under ROMFS only one file may be open at a time. When used to open a file, OS_Find returns a `handle', which is a unique identifier by which the file contents are made available to applications. This handle is used by the other routines to reference the file.

| | |
|---|---|
| *On entry :* | RO = action |
| | RI = file handle to close (if RO = 0) |
| | RI = pointer to filename (if RO <> 0) |
| *On exit :* | RO = file handle, or is preserved if RO=0 on entry |

The particular action is determined by RO as follows:

RO = 0 (&00) indicates that an open file is to be closed.

If R1 is zero then all currently open sequential files associated with the current filing system are closed.

If R1 is non-zero, if is taken to be a file handle, and the corresponding open file is closed.

RO = 64 (&40) indicates that a file is to be opened for input.

The location in memory containing the first character of the filename is pointed to by Rl. This name must be terminated by a carriage return, line feed or null byte.

If the file does not exist, then a file handle of zero is returned in R0; this is not an error. Otherwise the file is opened for reading, and a unique file handle passed back to the caller in R0.

RO = 192 (&CO) indicates that a file is to be opened for update.

Note that ROMF S does not support write operations. Therefore this command can only be used for reading files.

The location in memory containing the first character of the filename is pointed to by Rl. This name must be terminated by a carriage return, line feed or null byte.

If the file does not exist, then a file handle of zero is returned in R0. Otherwise the file is opened and a unique file handle passed back to the caller in R0.

OS_GBPB - READ BYTES AND FILE INFORMATION

OS_GBPB has many actions, including transferring bytes from an open file and reading file information.

| | |
|---|---|
| *On entry :* | RO = action |
| | R 1 - R6 depend on RO |
| *On exit :* | RO is preserved |
| | R 1 is preserved |
| | R2 - R4 may be updated, depending on action |
| | R5 is preserved |
| | R6 is preserved |

The particular action of OS_GBPB is determined by RO as follows:

RO = 3 read bytes from a specified position in a file

| | |
|---|---|
| *On entry :* | R 1 = file handle |
| | R2 = memory address to put data |
| | R3 = number of bytes to read from file |
| | R4 = sequential file pointer to use for start of block |
| *On exit :* | R1 is preserved |
| | R2 = memory address of first byte not written to |
| | R3 = number of bytes not read |
| | R4 = sequential pointer plus number of bytes transferred |
| | C flag depends on R3 |

Data is written to memory from the given file at the specified file address. If the file address is greater than the current file extent then no bytes are read, and the sequential file pointer is not updated. Otherwise the sequential file pointer is set to the specified file address. The memory pointer is incremented for each byte read, and the final value is returned in R2. The sequential pointer is incremented for each byte read, and the final value is returned in R4. The EOF-error-on-next-read flag is cleared.

If the number of bytes not read is not zero, then the carry flag is set on exit, otherwise it is clear.

RO = 4 read bytes from the current position in the file

| | |
|---|---|
| On entry : | R1 = file handle |
| | R2 = memory address to put data |
| | R3 = number of bytes to read from file |
| On exit : | R1 is preserved |
| | R2 = memory address of first byte not written to |
| | R3 = number of bytes not read |
| | R4 = original pointer plus number of bytes transferred |
| | C flag depends on R3 |

Data is written to memory from the given file at the current sequential file pointer. The memory pointer is incremented for each byte read, and the final value is returned in R2. The sequential pointer is incremented for each byte read, and the final value is returned in R4. The EOF-error-on-next-read flag is cleared.

RO = 5 read name and boot (OPT 4) option

| | |
|---|---|
| On entry : | R2 = memory address to put data |
| On exit : | R2 is preserved |
| | C flag is undefined |

This call obtains the name of the device and its boot option. This data is returned in the area of memory pointed to by R2, in the following format:

<name length byte><device name><boot option byte>

RO = 6 read current directory name and privilege byte

| | |
|---|---|
| On entry : | R2 = memory address to put data |
| On exit : | R2 is preserved |
| | C flag is undefined |

This call obtains the name of the currently selected directory. This is always `$' under ROMFS. It also returns the privilege status in relation to that directory. This data is returned in the area of memory pointed to by R2, in the following format:

<zero byte><name length byte><current directory name><privilege byte>

The privilege byte is &00 if the user has `owner' status (ie can create and delete objects in the directory) or &FF if the user has `public' status (ie is prevented from creating and deleting objects in the directory). On ROMFS the user always has owner status.

RO = 7 read library directory name and privilege byte

| | |
|---|---|
| *On entry :* | R2 = memory address to put data |
| *On exit :* | R2 is preserved |
| | C flag is undefined |

This call obtains the name of the library directory. This is always `$' under ROMFS. It also returns the privilege status in relation to that directory. This data is returned in the area of memory pointed to by R2, in the following format:

<zero byte><name length byte><library directory name><privilege byte>

RO = 8 read files

| | |
|---|---|
| *On entry :* | R2 = memory address to put data |
| | R3 = number of filenames to read |
| | R4 = start offset |
| *On exit :* | R2 is preserved |
| | R3 = number of filenames not read |
| | R4 = offset of next item to read (-1 if finished) |

R3 is treated as the number of filenames to read. R4 is the offset at which to start reading (ie if it is zero, the first item read will be the first file). Filenames are returned in the area of memory specified in R2. The format of the returned data is:

| | |
|---|---|
| length of first filename | (one byte) |
| first filename in ASCII | (length as specified) |
| . . | repeated as specified by R3 |
| length of last filename | (one byte) |
| last filename in ASCII | (length as specified) |

If all the filenames are read, then the carry flag is clear on exist, otherwise it is set.

RO = 9 read entries from specified directory

| | |
|---|---|
| *On entry :* | R1 = pointer to directory name (null terminated) |
| | R2 = memory address to put data |
| | R3 = number of object names to read |
| | R4 = offset of first item to read in directory |
| | R5 = buffer length |
| | R6 = pointer to special field if present, otherwise zero |
| *On exit :* | R3 = number of names read |
| | R4 = offset of next item to read (-I if finished) |
| | C flag is undefined |

This call reads the names of entries in a directory into an area of memory pointed to by R2. If the directory name is null, then all entries are used. The names are returned in the buffer as a list of null terminated strings.

RO = 10 read entries and information from specified directory

| | |
|---|---|
| On entry : | R1 = pointer to directory name (null terminated) |
| | R2 = memory address to put data |
| | R3 = number of object names to read |
| | R4 = offset of first item to read in directory |
| | R5 = buffer length |
| | R6 = pointer to special field if present, otherwise zero |
| On exit : | R3 = number of records read |
| | R4 = offset of next item to read (-1 if finished) |
| | C flag is undefined |

This call reads information about entries in the given directory into the area of memory pointed to by R2. If the directory name is null, then all files are read. The names and information are returned in records, with the following format:

| Offset | Contents |
|---|---|
| &00 | load address |
| &04 | execution address |
| &O8 | length |
| &OC | attributes |
| &10 | object type |
| &14 | object name (null terminated) |

Each record is word aligned.

## OS_BGet - READ SINGLE BYTE FROM AN OPEN FILE

OSBGET returns the byte at the current sequential file pointer position. If the EOF-error-on-next-read flag is set on entry, then an `End of file' error is given. If the sequential pointer is equal to the file extent (ie trying

to read at end-of-file) then the EOF-error-on-next read flag is set, and the call returns with the carry flag set, RO being undefined. Otherwise, the sequential file pointer is incremented and the call returns with the carry flag clear.

| | |
|---|---|
| On entry : | R1 = file handle |
| *On* exit : | RO = byte read if C clear, undefined if C set |
| | R1 is preserved |

OS_Args - READ AND SET ARGUMENTS FOR AN OPEN FILE

OSARGS reads an open file's arguments or returns the filing system type in use.

| | |
|---|---|
| *On* entry : | RO = action |
| | R1 = file handle or 0 |
| | R2 = data |
| On entry : | RO = filing system number or is preserved |
| | R1 is preserved |
| | R2 = data |

The particular action of OS Args is specified by RO as follows:

RO = 0

| | |
|---|---|
| R1 = 0 | return currently selected filing system number in RO |
| R1 = file handle | return sequential pointer for file in R2 |

RO=1

| | |
|---|---|
| R1 = file handle | set sequential pointer to R2 |
| R2 = new value | |

RO=2

| | |
|---|---|
| R1 = file handle | return extent (ie length) of file in R2 |

RO=4

| | |
|---|---|
| R1 = file handle | return size of (ie space occupied by) file in R2 |

RO=5

| | |
|---|---|
| R1 = file handle | return end-of-file indication |

If the sequential pointer is equal to the extent of the given file, then an end-of-file indication is given, with R2 set non-zero on exit. Otherwise R2 is set to zero on exit.

OS_File - PERFORM ACTION ON WHOLE FILE

The particular action of OS File is specified by RO as follows:

RO = 5 read catalogue information for the named object

| | |
|---|---|
| On entry : | R1 = pointer to filename |
| On exit: | RO = file type |
| | R 1 is preserved |
| | R2 = load address |
| | R3 = execution address |
| | R4 = file length |
| | R5 = file attributes |

The load address, execution address, length and file attributes from the named file's catalogue entry are read into registers R2, R3, R4 and R5. On exit, RO contains either:

| Value | Meaning |
|---|---|
| 0 | notfound |
| 1 | file found |
| 2 | directory found |

Files in ROM have the bits of the file attributes set as follows:

| Bit | Value | Meaning |
|---|---|---|
| 0 | 1 | object has read access for you |
| 1 | 0 | object does not have write access for you |
| 3 | 1 | object is locked against deletion |
| 4 | 1 | object has read access for others |
| 5 | 0 | object does not have write access for others |

RO = 255 load the named file into memory

| | |
|---|---|
| On entry : | R1 = pointer to filename |
| | R2 = load address of file (if R3 = 0) |
| | R3 = load at own / load at given flag |
| On exit : | RO = 1 (object is a file) |
| | R1 is preserved |
| | R2 = load address |
| | R3 = execution address |
| | R4 = file length |
| | R5 = file attributes |

The named file is loaded into memory at a location determined by the contents of R3 as follows:

- if the least significant byte of R3 is zero, the file is loaded into memory at the address specified in R2.
- if the least significant byte of R3 is non-zero, the file is loaded into memory using the file's own load address.

An error is given if the file does not exist.

## FILE HEADER FORMATS

Each file in a ROM must have a special format header at the start of it so that ROMFS can recognise it and obtain the details it needs. This format is an extension of the format used on 6502 based BBC microcomputers. Two classes of ROMs are allowed:

- 6502 code ROMs

- Archimedes format ROMs

In both cases, the format of the header is such that there must always be a copyright message beginning with the three bytes &28 &43 &29 making up the string `(C)'. This string must be preceded by the byte &00. The preceding zero byte is to be found at an offset from the start of the file indicated by the value found in the byte at offset 7. The byte at offset 7 is thus known as the copyright pointer and the string to which it points is called the copyright string. When checking for a valid file, ROMFS checks that the copyright pointer points to a zero followed by the string `(C)'.

The two classes of ROM are distinguished by the value found in the bottom four bits of the ROM type byte at offset 6 from the start of the file. The following values are valid:

| Value | ROM contents |
|---|---|
| 0-3 | 6502 code |
| 13 | Archimedes code |

Therefore, if this value is 0-3, the file is assumed to be a 6502 ROM image, if it is 13 (&OD) then an Archimedes format file is assumed.

6502 ROMs

The format of a 6502 code ROM header is as follows:

| | |
|---|---|
| 4C xx xx | JMP Lang |
| 4C xx xx | JMP Serv |
| rt | ROM type |
| cp | copyright pointer |
| vn | binary version number |
| xx xx xx ... | title string 00 |
| xx xx xx ... | version String 00 |
| 284329 | (C) |
| xx xx xx ... | copyright string 00 |
| rr rr rr rr | Tube relocation address |

The bottom four bits of the ROM type byte must contain the value 0-3 if the file is to be taken as a 6502 ROM image.

The title string is used as the filename.

The rest of the header is ignored and the ROM is assumed to contain one file of length 16kbytes.

The load and execution addresses of the file are assigned by ROMFS. The values given are:

| Load | Execution |
|---|---|
| FFFBBCOB | BCBBCBBC |

Thus the file type is &BBC and the file can be recognised.

ARCHIMEDES ROMS

If, however, the bottom four bits of the ROM type byte contain the value 13 (&OD), then the header is assumed to be in the extended Archimedes ROM file format. In this case, further items from the header block are required to determine the length of the file and its load and execution addresses:

Filename

The title string contains the name to be used when referencing the file. The string is terminated by the first control character (ASCII 0-31). The space character (ASCII 32) is translated to

Execution address

The first word of the ROM contains the address to call in order to execute the code.

Load address

The `Tube relocation address' is used as the load address.

Length

The word after the `Tube relocation address' contains the length of the file. ROMs produced specifically for the Archimedes series can contain several files of varying lengths. In fact, one file can be split over several devices, provided that these are inserted in the correct order, giving a maximum file size of almost 640kbytes (5 x 128kbytes).

The length of one file is used to locate the start of the next. Hence, if any length is incorrect, the ROM manager will then jump to the wrong location and will fail to find a valid ROM header at this address. In this situation, the ROM Manager moves on to the next 8kbyte boundary and recommences its action from there.

The algorithm used to locate the start of the next file is as follows:

1 Look at current location.

2 If the header is correct, read the length and move on by that number of bytes.

3 Otherwise jump to the next 8kbyte boundary.

Directory entries

In order to simulate a directory structure within ROMFS, it is necessary to create dummy entries giving the directory names. These must consist of a ROM header with the value &8D in their ROM type byte. This distinguishes them from files which have the value &4D in the ROM type byte.

## ROM MAKER PROGRAM

A program to create a ROM image is provided in the ROM Manager ROM. To use this program type:

```
*ADFS
```

to enter the Advanced Disc Filing System and then:

```
*RUN ROM:ROMIT
```

to load and execute the program. Respond to the prompts as follows:

| Prompt for | Response |
|---|---|
| Output file name | name you wish to give to the image file. |
| Input object name | name of the file you wish to insert in the image ( **Return** if no more) |
| Name to give file | name this file is to have in the ROM |

The image produced can either be blown into an EPROM, if you have a suitable EPROM programmer, or can be loaded into RAM as described later in this guide.

Directories

To create a directory structure in the ROM you should enter the name of a directory (which must exist on the current filing system) in response to the `Input object name' prompt. The ROMIT program will notice that this is a directory instead of a file and prepare a dummy header for inclusion in the final ROM-image. You should give the name of the directory as you wish it to appear in the ROM in response to the `Name to give directory' prompt. Note that this name can be different to the input object name. For example:

| | |
|---|---|
| Output file name | MyROM |
| Input Object name | $ |
| Name to give directory | BASIC.graphics |

This will create a directory header indicating the presence of a directory `BASIC.graphics' in the ROM-image.

# STATIC RAM UPGRADE

## FITTING RAM

In addition to the six ROM sockets described previously, the ROM Podule contains two further sockets, IC7 and IC8. Each of these can hold 32kbytes of static RAM.

These are fitted in the same way as ROMs, however they have no links which need to be set.

Each RAM IC must be configured in a similar way to EPROMs/ROMs using the command

```
*CONFIGURE ROMBOARD <p> <d> 20 256
```

| | |
|---|---|
| where | <p> is the Podule socket(0-3) as before |
| | <d> is the RAM socket (7 or 8) |
| and | 20 256 indicates that the device is 32kbytes (256kbits) of static RAM |

Suggestions for the RAM which may be used are as follows:

- Hitachi HM62256P-12
- Toshiba TC55257P-12

## BATTERY BACKUP

It is possible to add battery backup so that the data in the RAM is retained when the power is switched off. The extra parts required for this upgrade are listed below. Your dealer should be able to assist you in obtaining these items.

| Component | Example |
|---|---|
| Battery, 3.6V 100mAhr NiCd rechargeable | VARTA DB 826/3 |
| Silicon diode, 1N4005 IA, 600V (min. working) | |
| Schottky diode, 100mA, 30V(max. working) | Mullard BAT85 |
| Resistor, 220 ohms 5% 0.25W | |
| Resistor, 39 ohms 5% 0.25W | |

To fit the battery backup it is necessary to cut a PCB link and to solder two resistors, two diodes and a battery to the Podule board. If you are not confident of doing this yourself then please ask your dealer to do it for you. Note that these additional components are not covered by your original ROM Podule guarantee.

The tools required are a small (15W) soldering iron, a pair of side-cutting pliers and a sharp knife.

Note: when you are soldering electronic components, the heat must be applied to the joint for as little time as possible. Heat conducts quickly along the copper connecting wires and PCB tracks and can damage the devices.

The steps required to fit the battery backup are given below. Please read them through carefully before commencing.

1 Remove the ROM Podule from the Archimedes and remove any ROMs fitted in it.

2 Locate the PCB link position LK3. This link is permanently connected both to VCC and VBB. Using the sharp knife, cut the track between the central hole and the hole labelled VCC. Note that the link connecting the central hole to VBB must be left intact.

3 Take the 220 ohm resistor, colour code red red brown. Bend its leads and insert it into the board in the space labelled R4 (nearest the connector). Solder it into the board and then trim the leads to length using the side-cutters.

4 Take the 39 ohm resistor, colour code orange white black and repeat step 3 in the space labelled R5.

5 Take the IN4005 diode and locate it in the space on the board labelled D1. Diodes must be fitted the correct way round. The line on the end of the diode case must be at the same end as the cross-line on the PCB, that is, pointing towards the battery side of the board. Solder the diode to the board and trim the leads to length.

6 Take the Schottky diode and locate it in the space on the board labelled D2. The line on the end of the diode case must be at the same end as the cross-line on the PCB, pointing towards the space for the battery. Solder it in place and trim the leads.

7 Locate the three pins of the battery in the holes on the PCB in the box labelled CUT LK3 BEFORE FITTING BATTERY and solder it in place.

Before plugging the Podule back into your Archimedes and switching on, check your work carefully. Incorrect installation of this battery upgrade can cause damage to the Podule or the Archimedes. Such damage is outside the scope of your guarantee and any subsequent repairs would only be carried out at your expense.

# ACCESSING RAM

## LOADING A ROM IMAGE INTO RAM

*ROMLOAD

Syntax: *ROMLOAD <Podule number <device> <filename> [<offset>]

This command loads the file <filename> into RAM. The Podule socket is given in <Podule number and the RAM socket number in <device>. The file is loaded at <offset> bytes from lowest address in the RAM; this defaults to zero. If the file is loaded into RAM socket 7 and is larger than the memory available in this socket, then, if possible, it overflows into the RAM in socket 8.

## READING FROM RAM

When an image has been loaded into RAM, it appears as a ROM in the ROMF S and can be treated in the normal way.

# APPENDIX A - CIRCUIT DIAGRAM