

Sideways ROM authoring notes

Introduction

This guide is a collection of notes from numerous sources (see References) on the format and authoring of sideways ROMs, intended to be read primarily by software authors. It covers how the machine operating system will interact with the software through its two entry points, the language entry point and service entry point.

What is a sideways ROM?

The 6502 processor (and derivatives) used in the BBC Microcomputer series are restricted by a 16 bit address bus, giving the possibility of 65536 uniquely addressable locations.

Usually 32k is set aside for RAM, and the top 32k set aside for ROM, being arranged as 16k for applications code such as the *View* wordprocessor and 16k for the operating system.

Such an arrangement has one problem, the machine can only hold 16k of application code (*View* in this example) so every time a different application is needed the lid would need to be removed and the ROM chip exchanged.

To avoid this a number of application ROMs (*View*, *Viewsheet*, *Disc Filing System*, etc...) can occupy sockets on the motherboard and only one is activated at a time using chip select logic. As only one can be active at a time they always occupy the same address in memory, and can be thought of as lined up side by side waiting to be activated - they are sideways ROMs.

Conventions used in this manual

The following typographical conventions are used throughout this guide:

Hexadecimal numbers are prefixed with ampersand.

Decimal numbers have no prefix.

Binary numbers may be denoted with a leading percent and given in decending bit significant order (ie. for an eight bit number they will be written in the order %76543210).

Multibyte data is stored in memory in little endian form.

Copyright

Econet is a registered trademark of Acorn Computers Ltd.

The term 'BBC' refers to the computer made for the BBC literacy project.

History

- V1.00 Created
- V2.00 Added details on the Electron calls
- V2.10 Corrected some typos, reformatted for web pages
- V2.15 Corrections from JG Harston. Added ROM type 13
- V2.20 Reformatted into more logical sections, added introduction
- V2.21 Added service call &2B detail
- V2.22 More unrecognised OSWord information for the Master

ROM header format

The sideways ROM must contain at least a minimal header in order for it to be recognised by the operating system.

- &8000 JMP language (usually set to 0 if not a language)
- &8003 JMP service (always present, except 6502 BASIC)
- &8006 ROM type byte
 - BIT#7 Service entry present bit - 6502 BASIC doesn't have one
 - BIT#6 Language entry point present bit - languages only
 - BIT#5 Set if got a 2nd processor relocation address
 - BIT#4 Set if ROM supports Electron firmkeys (KEY+FUNC/KEY+CAPS LK)
 - BIT#3210
 - 0000 6502 (or 65C12) 6502 BASIC ROM
 - 0001 Reserved
 - 0010 6502 (or 65C12) code but not the 6502 BASIC ROM
 - 0011 68000 code
 - 1000 Z80 code
 - 1001 32016 code
 - 1010 Reserved
 - 1011 80186 code
 - 1100 80286 code
 - 1101 ARM code
- &8007 Copyright offset pointer (no. bytes from start of the ROM to the 0 byte just before the text).
Use &8yyy MOD 256 to find this.
- &8008 Binary version number. Usually the whole number part of the version string - this location is not used by the MOS, so could be any value of significance to you.
- &8009 Title string followed by zero terminator.
If the ROM is a language then this text will be printed by the MOS when the language is started.
- &8xxx Version string followed by zero terminator.
This is optional, taking the form "1.23" or "1.23 (01 Jan 1999)".
If it does exist and the ROM is a language then when the language is started the error message pointer at &FD and &FE points to it, or if missing they will point to the copyright message instead.
- &8yyy Zero byte followed by copyright string followed by zero terminator.
Mandatory, taking the form "(C)1999 Author", the MOS uses as one of the means to determine if a valid ROM is present. The preceding zero is the one pointed to by the byte at &8007.
- &8zzz Tube relocation address (4 bytes, high byte first).
Required only if bit 5 of the ROM type is set.
Service ROMs aren't copied across the Tube at all. ROMs which contain both a service entry and language entry will be, though the service code is only run on the host so should remain assembled starting at &8000.
If the ROM type byte denotes this ROM contains 6502 code, and the high 2 bytes aren't zero this denotes the ROM normally runs at &8000 but includes a relocation table which MOS 3.50 will handle.
The format of this table is given in the MOS 3.50 documentation.

Note: Addresses after &8009 are dependant on the length of the strings which precede them.

The language entry point

Languages

Need not be programming languages, but simply those with language entry points (eg. the *View* wordprocessor poses as one).

1. Languages should be able to be started by

A star command, which if recognised issues *FX142,skt# to select itself.

Enters via the service entry point call 4.

The user typing in *FX142,skt# manually.

Enters via the language entry point.

Being the configured language (or the highest socket number for machines without non volatile configuration memory) at reset.

Enters via the language entry point.

2. First re-enable interrupts

The language is started with interrupts disabled, use CLI to reenable them.

3. Reset the stack pointer to &FF

The language should flatten the stack, use LDX #255 : TXS to do this.

Conditions on entry

On jumping to the language entry point A will contain one of the following

0 No language found, calling the Tube ROM instead

1 Normal startup

Types 2 and 3 are for the FUNC keys:

2 Request next byte of softkey expansion. Y=byte. [Electron MOS only]

3 Request length of softkey expansion. Y=key number on entry and should be set to the length. [Electron MOS only]

Next, the MOS will set the error point at &FD/&FE to point at the version string (or copyright message if no version string is present).

This is the reason why REPORT in 6502 BASIC shows the copyright message assuming no errors have occurred since it started.

The language must change BRKV to point to an error handler, usually in ROM.

The MOS copies your socket number so that if Break is pressed the last active language is re-entered. The MOS also automatically prints the ROM's title string (&8009) so that the user is acknowledged.

The language has access to memory &400 to &7FF (1024B) as workspace and zero page from &00 to &8F along with the block OSHWM to screen memory.

The language should not perform direct memory accessing, as languages are copied across the Tube interface. Note however that service entry points are assembled as per the usual to run in the host BBC. The rest however may be assembled so that absolute addressing is higher (as with Hi-BASIC) paired with of course a higher Tube relocation address. If the language is to work in BOTH a BBC and a second processor then it MUST not have been assembled to a 'high' version.

```
FOR x=4 TO 7 STEP 3      (offset assembly)
O%=&5000:P%=&8000      (assemble at &5000 as though at &8000)
```

```

[OPT x
JMP language
JMP service                (and rest of header)
.service
etc... ]
P%=&B800                    (higher address for Tube)
[OPT x
.language
etc... ]
NEXT

```

BRK errors

The BBC micro uses the BRK instruction to change program flow and cause an error. These are effectively software interrupts, which are handled by the generic IRQ handler. They take the format `0+ERR+"Error"+0`.

The microprocessor saves the `addr+2` of the first BRK on the stack high byte first, and the processor's status register is copied onto the stack too. Interrupts are disabled (so no other errors can occur) and the BRK flag is set. Execution jumps to the value held at `&FFFF &FFFE` - points to the OS.

Next, the OS saves the accumulator at `&FC`, pulls the stack (the processor's status register) then pushes the processor's status register back onto the stack leaving a copy in the accumulator.

This is ANDed with 16 to isolate the BRK bit. If it was not a BRK interrupt then the OS jumps through IRQ1V. If bit 4 was set then the `addr+2` is read in, has one subtracted from it and then this is stored at `&FD` and `&FE`. Provided the error was set up in the above format this address now points to the error number.

Location `&F4` (RAM copy of ROM select latch) is read and copied. Service call six is then issued, which after the service ROMs have done as they will the currently active language is re-enabled.

Interrupts are re-enabled, and the operating system jumps through BRKV (which the language has previously set up to point to its error handler). The language MUST reset the stack pointer (to `&FF`) to avoid all the imbalances that the microprocessor has created.

Error messages should not be executed from within service ROMs, as not all language ROMs (inc.6502 BASIC) bother to check which ROM caused the error which renders REPORT useless. See *FX186 to find the ROM active at last BRK opcode.

A RAM copy should be made of error messages at the bottom of the hardware stack (`&100` up), allowing the error message to be printed without needing to use OSRDRM to get the bytes of the error message from within your ROM.

Handling escape

Testing whether the user has pressed Escape every time you read the keyboard is required to prevent the system locking up - when reading the keyboard via OSRDCH the Escape status is returned in the carry flag.

While not explicitly reading the keyboard (eg. waiting for the disc controller to respond) the Escape state is available through

- checking for bit 7 set at `&FF`
- leaving it up to the operating system to warn you via EVENTV having first enabled the Escape event

The Escape flag must be reset using *FX126. Note however that *FX126 also purges buffers and closes any open *EXEC files. On return Y=&FF means Escape condition cleared, Y=0 means still set.

Changing ROMs

The currently selected ROM number set with the write only ROM select latch at address &FE30. As &FE30 is write only a RAM copy of the currently selected ROM is held at &F4. To manually alter the current ROM perform

```
LDX#romnum
STX&F4
STX&FE30
```

Note that &F4 is written before the hardware register incase an interrupt occurs between the two stores. The MOS preserves &F4 during interrupt processing so on exit from the interrupt handler the correct ROM will be reselected.

Unless the desired target address in "romnum" is the same as that at which this code resides, the changeover of ROMs will need to be performed from RAM.

The service entry point

All ROMs (except 6502 BASIC) have service entry points so must be able to process service calls from the MOS. ROMs can issue service calls themselves through the use of *FX143.

The MOS recognises BASIC by its lack of language entry, and in fact the command *BASIC is handled by the MOS and not by BASIC itself.

ROMs are selected in order (from &F (hi priority) to &0) with the following information in the registers

- A=Service call number
- X=Number of current ROM socket
- Y=Any other parameters (if applicable)

ROMs that do not want to trap a particular service call need only perform an RTS ensuring that the A, X, Y are all left unaltered. Otherwise, to tell the OS that the call has been tended to perform LDA#0:RTS generally.

CALL	MEANING
0 &00	Call already serviced by higher priority ROM. [All MOS versions]
1 &01	Claim absolute workspace in RAM This allows workspace to be claimed by a ROM that other ROMs may corrupt if you're currently paged out. Y contains the current upper limit of this absolute workspace (initially set to &0E because &0E00 is the default of OSHWM). It can be incremented if it is considered that the upper limit is not sufficient, depending on the number of pages required. Do not alter the accumulator or X register simply perform an RTS. [All MOS versions]
2 &02	Claim private workspace in RAM This allows workspace to be claimed by a ROM that no other ROMs may use even when you're paged out. The value of Y on entry is the next free page of memory (to which OSHWM would be set to if no ROMs claim any memory). If you DO want workspace then store the current value of Y at (&DF0+YOUR ROM SOCKET NUMBER), then increment Y depending on the number of pages you require. Do not alter the accumulator or X register. [All MOS versions]
3 &03	ROM auto boot After a break the service ROMs are called, primarily to allow filing system ROMs to be booted. eg. D-Break starts the Disc Filing System. If the Y register is zero on entry the filing system should look for a file called !Boot (usually) and RUN/LOAD/EXEC it as required. Perform OSBYTE &7A to read the keyboard, the internal key number is returned in X. The key value is determined by finding the key under INKEY in the User guide performing INTERNAL=(INKEY NUMBER+1)*(-1). If the key being pressed is yours flush the keyboard buffer, and perform any required boot up actions. LDA#0 to acknowledge and RTS, otherwise restore ALL registers. [All MOS versions]
4 &04	Star command not recognised The location ((?&F3*256)+?&F2)+Y points to the first character of the command. Compare the string using the method outlined in 'Interpreters' below. If the command's not yours then restore ALL registers and RTS If you've accepted and have run the command then LDA#0:RTS If no ROM claims the command it's passed to the filing system which should attempt to *RUN it. [All MOS versions]

5 &05 Interrupt not recognised

If your ROM makes use of the IRQ line, and the BBC generates an unexpected interrupt that the OS cannot process then it is passed to the ROMs. You then check whatever hardware is necessary and process the interrupt, LDA#0 and RTS (not RTI). If you were expecting an interrupt, but the one generated was not from the device you're checking for then restore ALL registers and RTS so that other ROMs can check their peripherals. If no ROM claims it then it is passed to the user through IRQ2 vector. **[All MOS versions]**

6 &06 BRK occurred

ROMs are polled to inform the software that opcode 0 has been executed. &FD & FE point to the error number generated and &F0 contains the stack pointer immediately after BRK was executed. Use OSBYTE &BA to find (if ((?&FE*256)+?&FD) > &7FFF) in which ROM the BRK was executed.

If you trap this call successfully LDA#0 and RTS otherwise restore ALL registers then RTS upon which it will be passed to the current language via BRKV which points to the language ROM's error handler. **[All MOS versions]**

7 &07 OSBYTE not recognised

If the MOS does not recognise an osbyte then &EF = A
&F0 = X
&F1 = Y

Then call 7 is issued. If it passes through all ROMs and is not claimed then Bad Command is printed.

If you do claim it then LDA#0:RTS otherwise restore ALL registers. **[All MOS versions]**

8 &08 OSWORD not recognised

This service call is performed in a similar manner to call 7.

Trapping OSWORDS > &E0 is silly as these all pass through USERV.

OSWORD 7 (make a sound) will also cause this service call if the channel number is out of the range 0 to 3 - allowing for future sound system upgrades across the 1MHz bus or cartridge slots. **[BBC and Electron MOS versions]**

OSWORD 7 (make a sound) will cause this service call if the continuation control bits aren't 0 or 1, effectively changing the lowest out of range channel to &2000. **[Master MOS only]**

OSWORD 14 (read real time clock) with subreason codes greater than 2 will be offered using this service call to allow extra subreason codes to be defined. **[Master MOS only]**

OSWORD 7 (make a sound) and OSWORD 8 (define envelope) are passed through this vector if the external sound flag is set with *FX219, effectively making all channel numbers out of range. **[Electron MOS only]**

9 &09 *HELP issued

The ROM may print out a string so that the user knows which version of firmware he is using. The ROM may also wish to perform extended help (another few lines of text with the available * commands) and should do so by taking the address at &F2 (LB) and &F3 (HB) and Y which point to the first non SPACE character after the *HELP command.

If location ((?&F3*256)+?&F2)+Y contains CHR\$(13) then only *HELP was entered. Otherwise the letters that follow should be compared with a lookup table of the string you want to trap for. Force the string to upper case (AND #&DF) should caps lock have been off.

If it all matches up then print your extended help details (indent in the way that the DFS does for consistency)

- On exit restore ALL registers. **[All MOS versions]**
- 10 &0A Claim static workspace in RAM
When a ROM is paged in which is to make use of the static workspace set by call 1 then it should issue this call to inform all the other ROMs that may be using it to copy any relevant data into private workspace (if it claimed any). It may also be an idea to update some memory location somewhere that reminds yourself whether or not it is you currently using this absolute workspace. Claiming this workspace also frees up &B0 to &CF. **[All MOS versions]**
- 11 &0B Release of NMIs & NMI workspace (see also call 12)
SENT BY: The current NMI owner, with Y=previous owner id (usually the ROM number) and any NMI workspace should be tidied up or preserved if necessary.
RECEIVED BY: All ROMs, if Y=your owner id then the use of NMIs has been returned and your NMI code at &D00 should be installed. **[All MOS versions]**
- 12 &0C Request to claim use of NMI's & NMI workspace
SENT BY: Code wishing to take exclusive control of NMIs, eg. a filing system wishing to use the NMIs with the disc controller. On entry, Y=&FF when sent and on exit will contain the previous owner id (or remains &FF if there was no previous owner). ROMs are called in decreasing numerical order, this is why the filing system should be in the lowest priority socket - so that ALL possible higher priority ROMs have had a chance respond.
RECEIVED BY: All ROMs, if you are the current owner of NMIs then Y should be updated to be your owner id (usually the ROM number), you should relinquish control of the NMI workspace, and LDA#0:RTS. If you own NMIs then the NMI code should be placed at &D00 and the zero page locations &A0 to &A7 are free for your use. **[All MOS versions]**
- 13 &0D Initialise ROM filing system
On entry Y contains 15 minus the next ROM to be scanned. See RFS notes **[All MOS versions]**
- 14 &0E Return byte from ROM filing system
See also RFS. On entry Y contains 15 minus the next ROM to be scanned and on exit Y should contain the byte read. **[All MOS versions]**
- 15 &0F Vector changed
This call informs other ROMs which may be using intercepting vectors that one or more have been changed. This implies a change of filing system has TAKEN place because the FS repoints the filing vectors into the relevant new code. It is therefore likely that this call originated from the FS to courteously inform other ROMs that these vectors have been repointed. **[All MOS versions]**
- 16 &10 *EXEC/*SPOOL files about to close
A change of filing system is imminent. If the your ROM uses SPOOL or EXEC files then it must perform housekeeping functions and then close them before the filing system changes... You may however force the EXEC and SPOOL files to remain open by performing LDA#0 then RTS, with X and Y unchanged. Otherwise ALL registers must be restored before RTS. **[All MOS versions]**

- 17 &11 Character set about to explode/implode
This service call is induced by the user typing *FX20, n.
It means that OSHWM is on the move (Y=high byte of new OSHWM), and you should save any data that is in that area elsewhere before it is overwritten by font definitions. 6502 BASIC ignores this as it has no service entry point. **[All MOS versions]**
- 18 &12 Select filing system
On entry Y contains the filing system identity (see OSARGS) to change to. This provides a faster way than passing (for example) *DISC to OSCLI for programs which make use of more than one filing system:eg. files open on a NET and DISK or when copying from TAPE to DISC. Therefore this call must be accepted by at least the filing system ROM(s) but may be issued by any ROM. **[All MOS versions]**
- 19 &13 Character entered RS423 buffer
This call is made only under the Electron OS to allow expansion firmware to claim the call and take the byte in Y. If no claim is made, the OS purges the buffer. **[Electron MOS only]**
- 20 &14 Character entered printer buffer
This call is made only under the Electron OS to allow expansion firmware to claim the call and take the byte in Y. If no claim is made, the OS purges the buffer. **[Electron MOS only]**
- 21 &15 Software generated polling interrupt
After *FX22 is used this service call will be issued 100 times every second. It will stop after *FX23 has been entered. Trapping this call could for example allow a timer to be incremented somewhere in memory or allow you to periodically check some additional hardware that you've added. A, X, Y should exit unaltered incase other lower priority ROMs are also trapping this call. **[Electron and Master MOS versions]**
- 22 &16 BEL request - if the external sound flag is set (by *FX219) then this call is made when a VDU7/CTRL-G is printed **[Electron MOS only]**
- 23 &17 Sound buffer purged - informs the external sound system that an attempt has been made to purge the sound buffer(s) **[Electron MOS only]**
- 24 &18 Interactive *HELP
After all the ROMs have had service call 9 passed to them, this is issued to enable further help. eg. you print the question 'More details with MYROM? Y/N' and wait for a response. On the Econet the ANFS traps this call and trys to load in the file !HELP in the current directory in the fileserver. **[Master MOS versions]**
- 33 &21 Indicate static workspace in 'hidden' RAM
Static workspace may be used by any ROM, but only one at a time. This means that when paged out this memory will probably be corrupted by another ROM. On entry Y contains the current upper limit which should be incremented if it is felt that this is not enough although as hidden RAM runs from &C000 to &DBFF, Y < &DC, else part of the user's RAM will be bitten into. **[Master MOS versions]**

- 34 &22 Claim private workspace in hidden RAM
Having performed call &21 to all 16 ROMs the lowest address of private workspace is now known. This value is placed in Y and each ROM paged again to allow them to claim their own personal memory if required. This 'hidden' RAM is not available in the BBC-B but this call mimics that of service call 2. Private workspace is fixed to one ROM only and the ROM should note its lowest available page in the ROM workspace table at &DF0 upwards. The address of this table may alter but it comes straight after the extended vector table - the length of which is known & address of which is found with an OSBYTE. **[Master MOS versions]**
- 35 &23 Inform ROMs of top of static workspace. Y=max value+1.
Filing system software notes this value so that the rest of hidden RAM may be used as temporary buffers (eg. for command *COPY) **[Master MOS versions]**
- 36 &24 Indicate workspace requirements in hidden RAM
This 'hidden' RAM is not available in the BBC-B but this call mimics that of service call 1 **[Master MOS versions]**
- 37 &25 Inform MOS of filing system details
This is issued by the MOS and need only be trapped by the current filing system. It should return 11 bytes the start of which is pointed to by the vector at &F2 and &F3. These eleven consist of 8 bytes (space padded) of ASCII characters describing the filing system eg. "NET " then the lowest handle in use then the highest handle in use then the filing system number.
More than one block of 11 can be submitted to allow more than one synonym for a filing system to be used, eg. the tape filing system submits both "TAPE" and "CFS". Exit with Y incremented by (n*11) **[Master MOS versions]**
- 38 &26 Close all files
When *SHUT is entered this call is issued. **[Master MOS versions]**
- 39 &27 Hard-reset (Ctrl-Break or just turned on) has occurred **[Master MOS versions]**
- 40 &28 Unknown *CONFIGURE
This sets certain startup options, which are kept even after power is removed as the Master contains 50 bytes of backup CMOS RAM which is assigned as follows:
0 -19 *CONFIGURE system
20-29 Acorn's future use
30-38 Firmware ROM use (16 rom sockets present 7 of which represent the MegaROM software built which leaving 9 bytes ie. one for each of the remaining empty sockets such that ROM 0 uses byte 30 and ROM 9 uses byte 39.
39-49 User applications
On entry the vector at &F2 and &F3 plus the Y register point in the same way as *HELP.
Hence the *HELP and *HELP MYROM code can be used if modified.
This call is also issued if just *CONFIGURE is typed. If so then print COMMAND\$+STRING\$(9-LEN(COMMAND\$), " ")+"ON/OFF" to be consistent with the MOS.
Checking whether the ON and OFF option is present is easy as the O is common to both but N and F differentiate between the two options. **[Master MOS versions]**

41 &29 Unknown *STATUS

Having printed out the MOS's own startup options this call is issued. Test as with *HELP MYROM to see whether all the STATUS's were required or just one particular one - the vector at &F2 and &F3 + Y will either point to a command or a CHR\$(13). **[Master MOS versions]**

42 &2A ROM language starting up now

This can be trapped by the current language (and likewise service ROMs) to allow it to tidy up any language workspace or files and unhook any vectors it uses. **[Master MOS versions]**

43 &2B Read memory size

This service call is used to replace the default MOS startup banner to show the amount of sideways and shadow RAM fitted in the form "Acorn OS xxK". It is handled by the SRAM utils inside the DFS ROM. If the Tube is active, or the startup banner has been suppressed, or there was no sideways, or no shadow/sideways RAM found the banner is not replaced. The Y register contains the number of the calling ROM which will be compared with &F4 - this ensures that the banner is only printed once. **[BBC B+ MOS only]**

44 &2C Compact joystick call

This call allows firmware to respond to changes on the user port of the Compact. The Y register points to an offset from &200 of a 7 byte block: &200, Y+0=LSB of ADVAL

- Y+1=X coordinate (LSB first)
- Y+3=Y coordinate (LSB first)
- Y+5=space
- Y+6=space

This could also be used for mouse pointer type devices. **[Compact MOS versions]**

254 &FE Pre Tube initialisation

After OSHWM is defined this call is issued to allow any actions before the second processor starts to be performed, such as explode the font memory. On entry Y=0 denotes no Tube ULA was detected, else the Tube ULA is present - therefore it is issued every time Break is pressed regardless of whether the Tube is present, which may be useful. Exit with A=0 to claim, which suppresses the normal startup beep and banner and triggers call 255 to occur. **[All MOS versions]**

255 &FF Post Tube initialisation

Just before the Tube's OSHWM is defined this call is issued. It comes BEFORE the host's filing system is set up. **[All MOS versions]**

Filing systems

Memory use

These can gain exclusive rights to the following:

&A0 to &A7	= NMI workspace (after having claimed them with call 11/12)
&A8 to &AF	= Command workspace when a *command has been entered
&B0 to &BF	= Temporary (if you control the absolute workspace)
&C0 to &CF	= Private workspace which are unaltered provided the filing system is unaltered
&D00 to &D5F	= NMI service code space. Used for tracing NMI's. The end location (&D5F) would be a good place to save old Y from call 12 for example

Extended vector entry

The filing system must almost certainly claim one or more of the filing system vectors, however the normal method of overwriting the RAM vector with the address of the user supplied routine does not work with sideways ROMs, since the ROM may be paged out at the time the vector is called - jumping to the wrong place in the wrong ROM.

Extended vector entry into paged ROM is a method to work round this, by putting the address and rom number of the routine to call into the extended vector area, the operating system will first page in the sideways ROM before calling it and also handle restoring the original sideways ROM when done.

If for example the user types *CODE this indirects via USERV (which by default points to Bad Command) but this can instead be changed to jump to code in a ROM that is currently paged out.

First read the address of the ROM pointer table with *FX168. This call returns $((Y*256)+X)$ which points to the start of the table containing the extended vector entry address, VTABLEaddr in this text.

By pointing the first vector ($@ \&0200+(2*VECTORnum)$) into the extended vector entry point into the operating system ($@\&FF00+(3*VECTORnum)$). In future if USERV is called the operating system is forced to read the ROM pointer table.

In that table ($@VTABLEaddr+(3*VECTORnum)$) there's a JMP addr and a ROM socket number LB, HB, ROM# whereupon the OS pages in the respective ROM and jumps.

eg. As USERV is VECTORnum 0
 When changing vectors use PHP:SEI
 The main vector is at &200 &201 so save old vector somewhere either for restoration later or to jump through.
 Poke &200 with &00 and &201 with &FF (LB then HB as usual)
 Perform *FX168 to find where VTABLEaddr for the current MOS
 Save NEWROMvec MOD 256 at VTABLEaddr+0
 Save NEWROMvec DIV 256 at VTABLEaddr+1
 Read your ROM socket number and store at VTABLEaddr+2
 Vectors settled now so CLI:PLP

Hence paged sideways ROMs may claim the normal OS vectors.

The ROM filing system

All machines come with the ROM filing system available, which is virtually identical in structure to the cassette filing system, but with the advantage of near instantaneous access, but with the disadvantage of

being read only.

Paged ROMs contain specially formatted data and a minimalist service call handler (to process call 13 and 14) to return the bytes when requested, and may also include more complex service call handlers to offer help for example. The OS will also check for ROM filing system images in the speech ROM if fitted.

Typing *ROM initialises the system, and the following are available:

```
*LOAD ; *CAT ; *EXEC ; *LOAD ; *RUN
OSARGS (current filing system identification only applies)
OSBGET (no OSBPUT)
OSFILE (except saving)
OSFIND (opening for output is not possible)
```

Call 13 is issued whenever the RFS is active AND an OS command is in use:

```
.service_13
CMP #13
BNE service_14      \Not a rom filing system initialisation call
PHA                 \Save A
TYA                 \Y contains the ROM number
EOR #15             \Evaluate 15-Y
CMP &F4             \Is it this ROM?
BCC service_passon
LDA #filename MOD 256
STA &F6             \Yes so poke MOS with start of data
LDA #filename DIV 256
STA &F7
LDA &F4
EOR #15             \Evaluate 15-myrom#
STA &F5             \And my ROM number
JMP service_claim
```

Call 14 is issued whenever the RFS is active AND an a byte is requested:

```
.service_14
CMP#14
BNE service_other   \Other service call handlers if implemented
PHA                 \Save A for later
LDA &F5
EOR #15             \Perform 15-&F5
CMP &F4             \Yes it's my ROM that is being asked
BNE service_passon
LDY #0
LDA (&F6), Y       \Load A with the value pointed to by (?&F7*256)+?&F6+Y
TAY
INC &F6             \Increase F6 by one as 1 byte has been got
BNE service_claim   \If passed from 255 -> 0 then increase the high byte too
INC &F7
.service_claim
PLA                 \Balance push
LDA#0               \Tell MOS the byte has been read
RTS
```

```
.service_passon
PLA
.service_other
RTS          \Couldn't find - pass to another ROM
```

Service coding and header is as per normal sideways ROMs which must include the normal copyright message and service code, but the actual data which follows is presented in the following constant length block format:

```
1      Sync byte (AScii "*") makes the block header appear as *FILENAME
1-10   Filename (upto 10 bytes but at least 1)
1      Zero terminator byte of filename
4      File load address (4 bytes LOW first eg. &1900 in host BBC is &0019FFFF)
4      Exec address (4 bytes LOW first eg. &1900 in host BBC is &0019FFFF)
2      Two byte block number, low byte first
2      Two byte length of block of data, low byte first
1      File flag byte
        bit 7 set=last block of current file
        bit 6 set=no data (as created by F=OPENOUT"FILE":CLOSE#F)
        bit 5 to 1 not used
        bit 0 set=protection bit, the file can only be *RUN
4      Address of first byte after the end of file (4 bytes). This is implemented to allow extra fast
        cataloguing instead of having to read through all the data too, it's not used by the cassette filing
        system. Using *OPT1, 2 forces the OS to do it slowly verifying all data.
2      Two byte header (NOT the data) CRC, high byte first
0-256  Data (upto but not necessarily 256 bytes)
2      Two byte data CRC, high byte first
```

Which is repeated for EVERY block of data until the end of the source file. However to save space in the ROM you need not keep repeating this header block. Instead only the first and last blocks need be included and all the intermediate blocks replaced by a hash ('#') which the OS interprets as "same as last":

```
        First block as defined above
0-256  Data (upto but not necessarily 256 bytes)
2      Two byte data CRC, high byte first
:
1      A hash ('#') character telling the MOS the header is the same as the last
0-256  Data (upto but not necessarily 256 bytes)
2      Two byte data CRC, high byte first
:
        Final block as defined above with last block flag set
0-256  Data (upto but not necessarily 256 bytes)
2      Two byte data CRC, high byte first
```

The last block cannot be abbreviated to # as even if the source file was of length exactly divisible by 256 - bit 6 of the file flag needs setting.

After all the programs the ROM contains have been included, there should be a plus ('+') placed as an end of ROM marker. It may only be omitted if the image was too large to fit on just one ROM, in which case the

firmware must be installed such that the continuation of the file must be in the socket immediately below the first half and the software included in the sideways ROM service code to account for this cross chip gap.

The CRC can be found by the following and this calculation applies for both the header CRC and data CRC. On entry LOB contains the length of the data in memory and DAT points to where it starts in memory. On exit memory locations CRCHB and CRCLB contains the checksum:

```
.init
LDA #0
STA CRCHB
STA CRCLB
TAY

.nextbyte
LDA CRCHB
EOR DAT, Y
STA CRCHB
LDX #8

.loop
LDA CRCHB
ROL A
BCC over
LDA CRCHB
EOR #8
STA CRCHB
LDA CRCLB
EOR #16      \Perform the CRC
STA CRCLB

.over
ROL CRCLB
ROL CRCHB
DEX
BNE loop     \Continue until all 8 bits are done
INY
CPY #LOB     \End yet?
BNE nextbyte
RTS         \Finished
```

Example code

Help

Using a register for an index the string entered by a user may be compared against a lookup table of keywords one letter at a time. Take the help command, it may be that *HELP or *HELP MYROM was typed.

The MOS stores the last non space string after the *HELP command at the address pointed to by the vector at locations &F2 and &F3 plus the Y register, so, having checked it WAS service call 9:

```
.oncall9
Print out the ROM name and version only

.helponly?
LDA (&F2), Y \Get byte
CMP #13      \Was it just *HELP return?
BNE compare  \If not then see lookup table

.exit
Restore all registers
RTS

.morehelp
LDX #&FF    \Set up X so that the generic loop increments it to 0
DEY        \Shrink Y as the generic loop increments it

.compare
INY
INX
LDA (&F2), Y
AND #&DF    \Force to upper
CMP table, X \Check against lookup table
BEQ compare \Matches so next
LDA table, X
CMP #&FF    \Did it not match because it is the terminator byte?
BEQ routine \If A did contain 255 then the command matched up
JMP exit

.table
EQUUS "MYROM" (The table contains one string, the ROM name MYROM
EQUB &FF      only. So this code is less complex than
              'interpreters')

.routine
Print out the additional indented details for *HELP MYROM
```

Command interpreter

If an unrecognised star command (*ONEAND) is entered then the above code need only be repeated, but with the execution address stored in a table form too. As ROMs appear with high byte >&80 this means bit 7 is set so that negative flag will also be set:

```
.oncall4
LDX #&FF    \Set up X so that the generic loop increments it to 0
DEY        \Shrink Y as the generic loop increments it
TYA        \A only contained the service call number anyway
```

```

PHA                \Use stack as temp for Y so that it can be restored

.compare
INY
INX
LDA (&F2), Y      \Pointer for star command unrecognised by MOS
AND #&DF          \Force to upper
CMP table, X      \Check against lookup table
BEQ compare       \Matches so next

.notequal
LDA table, X      \Get the byte from the table that didn't match
BMI runnit        \If the byte is >&80 (ie. in a ROM) then in 2's
                  \complement this is a negative number

.findnext
INX               \If it was simply a mismatched letter then the start of
                  \the next command must be found and the whole
                  \process repeated

LDA table, X
BPL findnext      \Provided it's not the address (>&80) then continue
INX               \If we've reached the address block this can be skipped
                  \past

PLA
PHA               \Save it again for the 3rd and 4th... commands
TAY               \Restore Y to it's original decremented value from above
JMP compare

.table
EQU "ONEAND"      (command name must contain chrs 0 to 127 only)
EQUB oneand DIV 256 (DIV comes first as when in a ROM the addr
EQUB oneand MOD 256 will be >&80 so bit 7 will be set so that in
EQU "ANOTHER"    2's complement it is a negative number which
EQUB another DIV 256 can be tested for on loading the accumulator
EQUB another MOD 256 by using the command BMI)
EQUB &FF          (End of table marker)

.runnit
CMP #&FF          \Was infact the number >&80 the end of table marker?
BEQ passon        \None of my commands matched so leave
STA vector+1      \A currently contains the high byte (DIV)
INX               \Make X point to the low byte
LDA table, X      \A now contains the low byte (MOD)
STA vector+0      \Vector can be any suitable unused memloc
PLA               \Balance the push at the start - so A will now
                  \contain old Y
JMP (vector)      \And jump

.passon
PLA               \Balance the push at the start - so A will now
                  \contain old Y
RTS               \Allow any other ROMs to process the command

.oneand
Implementation code here.
As A contains old Y this it points to (command line - 1) so this can

```

```
be used to pass parameters
LDA #0      \Inform the MOS that the call has been answered
RTS
```

```
.another
Implementation code here.
As A contains old Y this it points to (command line - 1) so this can
be used to pass parameters
LDA #0      \Inform the MOS that the call has been answered
RTS
```

Printing numbers in hexadecimal

If the accumulator contains a number between 0 and 255 then to convert to a printable, 0 padded, hex code use:

```
.hexprint
PHA          \Save original A
LSR A       \Shift right 4 times, padding top 4 bits with 0
LSR A       \ie. Demoting top 4 bits, or dividing by 16
LSR A
LSR A
JSR convert
PLA          \Restore A
AND #15     \Remove upper
JSR convert
RTS         \Exit. Add an extra PHA/PLA if you want A uncorrupted

.convert
SED          \Perform in binary coded decimal from now on
CMP #10
ADC #48     \Add on ASCII code for the number 0
CLD         \Work in pure binary again
JMP OSWRCH  \Using JMP means the OS does the RTS - saves a byte
```

Printing numbers in decimal

Printing decimal is a little more tricky, this routine prints the value stored at 'num' and requires a single byte of workspace 'high':

```
.decprint
LDA#48      \ASCII for '0'
STAhigh     \Save in workspace
LDAnum
BEQfirst    \If the number is 0 then just print 000
LDA#0       \Otherwise initialise A
SED : CLC   \Operations in BCD & carry ready for ADC

.back
ADC#1
BCCnocarry
INChigh
CLC

.nocarry
DECnum
BNEback
```

```
CLD                \Work in binary again

.first
PHA                \Save A
LDAhigh
JSRoswrch         \Print the character produced by the above

.second
PLA:PHA           \Copy back original A
LSR A:LSR A
LSR A:LSR A       \Demote 4 bites (divide by 16 in binary)
CLC               \Clear carry ready for ADC
ADC#48            \Add ASC("0")
JSRoswrch         \Print to screen

.third
PLA:AND#15        \Restore A again, masking off bottom 4 bits
ADC#48
JMPoswrch        \Print CHR$(48+A)
```

References

Books

Title: ADVANCED SIDEWAYS RAM USER GUIDE by BRUCE SMITH
Publisher: VICTORY PUBLISHING, PO BOX 19, LONDON, N11 1DS.
ISBN: 0 948938 00 5
Published: 1986 (revised 1987)

Title: THE BBC MICRO ROM BOOK by BRUCE SMITH
Publisher: COLLINS PROFESSIONAL BOOKS, 8 GRAFTON ST., LONDON, W1X 3LA.
ISBN: 0 003830 75 6
Published: 1985 (not revised)

Title: THE Acorn Electron ADVANCED USER GUIDE
Publisher: ADDER PUBLISHING, PO BOX 148, CAMBRIDGE, CB1 2EQ.
ISBN: 0 947929 03 7
Published: 1984 (not revised)

Title: THE NEW ADVENACED USER GUIDE
Publisher: ADDER PUBLISHING, PO BOX 148, CAMBRIDGE, CB1 2EQ.
ISBN: 0 947929 05 3
Published: 1987 (not revised)

Errata

This section is for noting errata in existing publications.