ARM coprocessor supported OS calls

Introduction

This document describes the SWI calls available from the system software which is supplied with the ARM7 coprocessor board.

The available SWI calls can generally be put into three classes

> 1 - Calls translated and passed to the host micro
>> By example, OS_Plot is expanded into VDU25.
> 2 - Calls which are handled by the coprocessor alone
>> By example, OS_SynchroniseCodeAreas operates on the cache on the ARM.
> 3 - Calls which are intercepted by the coprocessor and a decision made as to how it is handled
>> By example, OS_Byte 130 is handled by the coprocessor, OS_Byte 129 is not.

On the whole, most SWIs fall into catagory 1, catagories 2 and 3 tend to be implemented so that BASIC will run.

It is strongly recommended that the reader has a copy of the "Advanced User Guide for the BBC Micro" (Bray, Dickens, Holmes ISBN 0946827001) and the "RISC OS 3 Programmer's Reference Manual" (Acorn Computers ISBN 1852501103) as this document does not attempt to detail in full all of the inner workings of a call - more a concise overview of the availability of a call and what to expect back as the answer.

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

V0.10	First draft
V0.15	Added OS_Byte/OS_Word and filing system call details
V0.16	Filled in the remaining supported SWIs
V0.24	Additions to reflect ARM Tube OS version 0.24
V0.25	Note for OS_GBPB reason code 8 for reading the cycle id
V0.29	Added OS_Word 14 subreason code 4 detail to match ARM Tube OS version 0.29
V0.33	Updated OS_PlatformFeatures and OS_Word to match implementation
V0.40	Added detail of new SWIs implemented in ARM Tube OS version 0.40
V0.45	Debugger SWI detailed

General notes

SWI documentation

To avoid duplication of large sections of material, where a SWI call is implemented fully the reader will simply be referred to alternative documentation, only where a SWI differs will the differences be covered here.

Vectors

There are two classes of vectors on the coprocessor, the hardware exception vectors required by the ARM itself (see OS_ClaimProcessorVector) and software vectors (see OS_Claim).

At various points during normal operation the software vectors will be called to give 3rd parties an opportunity to reimplement the call in a different way. The vectors are arranged as a simple list and the most recent use of OS_Claim will be called first, then down the list, until eventually the default handler in ROM will be called.

By example: when a SWI is called the ROM tables are checked first and if no match is found it is passed to UKSWIV - the unknown SWI vector. The default routine for UKSWIV calls the unknown SWI environment handler in the currently running application, and the default ROM based environment handler just causes an error "SWI not known".

To add a whole new SWI simply register with UKSWIV and check the value of R11 for the desired SWI number

> If it does not match then return using the address given in the link register

> If it does match the call can be claimed by pulling a return address off the stack

when claiming no other registrants for that vector will be called.

While it is true that all software vectors are present and can be claimed, currently only CliV/TickerV/ EventV/ErrorV/UKSWIV/MouseV/ChangeEnvironmentV are offered as the remainder are already present in the host.

Memory usage

Do not make assumptions about the layout of memory or specific locations within the coprocessor as these are likely to change. Use documented operating system calls to access the values you require, or to find out where specific regions of memory are placed.

Interrupts

All interrupts (IRQ and FIQ) are handled by the coprocessor for use with the Tube. They should not be claimed or disabled for extended periods as this may adversely affect the operation of the Tube.

Events

Events from the host are passed across the Tube where they will be offered to the software vector EventV. Higher event numbers that the host does not support can be emulated with OS_GenerateEvent if required.

Processor mode

The processor always runs in 32 bit mode, do not attempt to switch to a 26 bit mode or call SWIs while in Thumb mode. These details can be determined at run time by reading OS_PlatformFeatures.

Paged ROMs

ARM code can be stored in paged ROM in the host and loaded automatically at reset, this operation is detailed separately.

Calling X SWIs

Where the SWI number is given in brackets, thus:

OS_Write0 (&02)

there is an alternative version of the SWI with bit 17 of the SWI number set, thus:

XOS_Write0 (&20002)

which performs the same task except that if an error occurs during the processing the error handler will not be called, so for example your BASIC statements after ON ERROR will not be used, and instead the SWI will return with the processor's overflow (V) flag set and register R0 pointing to an error block.

From BASIC this would be used as

:

SYS "XOS_Write0", "My string"+CHR$0 TO block;flags

IF (flags AND 1) = 1 THEN PROCdisplayerror(block)

PROCdosomethingelse

:

the error block consists of a 4 byte error number followed by a zero terminated error string. Calling the non-X version of the SWI will leave it up to BASIC to alter the program flow, such as

:

ON ERROR PROCdisplayerror(ERR, REPORT$) : END

SYS "OS_Write0", "My string"+CHR$0

PROCdosomethingelse

:

so in this example PROCdosomethingelse will never be reached.

OS_WriteC (&00)

Print the character in R0.

This call is passed to the host operating system VDU driver, hence it will be limited only by the capabilities of the host OS. For example, circle plotting (via VDU25) is not available on an unexpanded BBC Model B Microcomputer whereas it is on a BBC Master 128 Microcomputer.

On entry	R0=character to output
On exit	R0 preserved
Interrupts	IRQs enabled, FIQs enabled
Processor mode	SVC32
Reentrancy	Not reentrant

See PRM1 page 501.

OS_WriteS (&01)

Write the zero terminated string which follows the instruction in memory.
See PRM1 page 503.

OS_Write0 (&02)

Write the zero terminated string at the address in R0.
See PRM1 page 504.

OS_NewLine (&03)

Print a line feed and carriage return.
See PRM1 page 505.

OS_ReadC (&04)

Wait for an incoming character from the current input stream.
See PRM1 page 852.

OS_CLI (&05)

Process an operating system "*" command.
See PRM1 page 929.

OS_Byte (&06)

Control over a wide assortment of system effects.
The coprocessor handles OS_Bytes slightly differently depending on the reason code given:

	Reason codes &00 ⩽ R0 < &80
On entry	R0=reason code
	R1=parameter
	R2=ignored
On exit	R0=preserved
	R1=result (if any)
	R2=corrupted

	Read high order address
On entry	R0=&82
	R1,R2=ignored
On exit	R0=preserved
	R1=0
	R2=0
	Note that this returns the address of the parasite, not the host.

	Read top of operating system workspace
On entry	R0=&83
	R1,R2=ignored
On exit	R0=preserved
	R1=low byte
	R2=high byte
	Note that this returns the address of the parasite, not the host.

	Read bottom of screen memory
On entry	R0=&84
	R1,R2=ignored
On exit	R0=preserved
	R1=&FF
	R2=&FF
	Note that this call simply returns the largest number it can (&FFFF), infact the value of HIMEM is much higher than this, use OS_ChangeEnvironment to read the true value.

	Reason codes &80, &81, &84 < R0 ≤ &FF
On entry	R0=reason code
	R1=parameter
	R2=parameter
On exit	R0=preserved
	R1=result
	R2=result

Interrupts	IRQs enabled, FIQs unaltered
Processor mode	SVC32
Reentrancy	Not reentrant

See PRM1 page 54.

OS_Word (&07)

Control over a wide assortment of system effects which can't be expressed with 2 parameters.

The coprocessor handles OS_Words slightly differently depending on the reason code given:

	Read a line of input from the current input stream
On entry	R0=0
	R1=pointer to parameter block
On exit	R0,R1=preserved

This reason code is translated into a call to OS_ReadLine32, and handled on the coprocessor rather than being passed to the host.

Reason codes &01 ≤ R0 < &17

On entry   R0=reason code

R1=pointer to parameter block

On exit   R0=preserved

R1=preserved and parameter block updated if required

These calls are passed to the host with known size parameter blocks based on a table kept in the coprocessor.

One small exception are the enhanced capabilities of the real time clock reason codes which have been extended to allow read/write/conversions of the time as an integer. When the reason code is &0E, two new subreasons are added:

R1+0 =   3: *Read time as centiseconds since 1900 as a 5 byte integer*

The coprocessor will ask the host for the time as BCD then convert it back locally since this time format is not normally available.

R1+0 =   4: *Convert 5 byte centisecond value to string*

The coprocessor will translate the time into BCD locally then ask the host to convert it to a string.

and when the reason code is &0F, one new subreason code is added:

R1+0 =   5: *Write time as centiseconds since 1900 as a 5 byte integer*

The coprocessor will translate this into a string using OS_Word &0E, then pass it to the host as a string to set the clock using OS_Word &0F.

Reason codes &17 ≤ R0 < &80

On entry   R0=reason code

R1=pointer to parameter block

On exit   R0=preserved

R1=preserved and parameter block updated if required

These calls are passed to the host using a fixed 16 byte block sent and received.

Reason codes &80 ≤ R0 ≤ &FF

On entry   R0=reason code

R1=pointer to parameter block

On exit   R0=preserved

R1=preserved and parameter block updated if required

These calls are passed to the host using the first byte in the block to determine the number of bytes sent, and the second byte in the block used to determine the number of returned bytes (to a maximum of 128 each way).

Interrupts   IRQs enabled, FIQs unaltered

Processor mode   SVC32

Reentrancy   Not reentrant

See PRM1 page 61.

OS_File (&08)

Operations on entire files.

This call is passed to the host filing system, hence it will be limited by the facilities offered by that filing system.

See PRM2 page 30.

OS_Args (&09)

Miscellaneous housekeeping operations on an open file.

This call is passed to the host filing system, hence it will be limited by the facilities offered by that filing system.

See PRM2 page 46.

OS_BGet (&0A)

Get a byte from an open file.
See PRM2 page 60.

OS_BPut (&0B)

Put a byte to an open file.
See PRM2 page 62.

OS_GBPB (&0C)

Block operations on an open file.

This call is passed to the host filing system, hence it will be limited by the facilities offered by that filing system.

Note that unlike its equivalent on the host it is not possible to read the cycle id through this interface as the file handle in R1 is defined as always being preserved, though ADFS and DFS have a corresponding OS_Word to read the cycle id.

See PRM2 page 63.

OS_Find (&0D)

Open (or close) a file.

This call is passed to the host filing system, hence it will be limited by the facilities offered by that filing system.

See PRM2 page 72.

OS_ReadLine (&0E)

Read a line of input.

This call is translated into OS_ReadLine32 and reissued, you should avoid its use or explicitly test that the buffer being provided is below &40000000 to ensure that its address is not accidentally interpreted as one of the two flags in R0 bits 30 and 31.

See PRM1 page 910.

OS_Control (&0F)

Obsolete, use OS_ChangeEnvironment instead.

This call returns the error "SWI not known".

OS_GetEnv (&10)

Read values about the enviroment of the last program started.
See PRM1 page 298.

OS_Exit (&11)

Exits the last program started and returns to the most recent exit handler.

The coprocessor does not maintain any system variables, therefore the return code/error block/magic word are all ignored at present. For compatibility they should still be set as appropriate.

On entry	R0=pointer to error block
	R1="ABEX" if the return code is to be tested
	R2=return code
On exit	Does not exit
Interrupts	IRQs unaltered, FIQs unaltered
Processor mode	USR32
Reentrancy	Not reentrant

See PRM1 page 300.

OS_SetEnv (&12)

Obsolete, use OS_ChangeEnvironment instead.
This call returns the error "SWI not known".

OS_IntOn (&13)

Enable interrupts and return.
See PRM1 page 138.

OS_IntOff (&14)

Disable interrupts and return.
See PRM1 page 138.

OS_CallBack (&15)

Obsolete, use OS_ChangeEnvironment instead.
This call returns the error "SWI not known".

OS_EnterOS (&16)

Switch the processor to SVC32 mode and return.
See PRM1 page 140.

OS_BreakPt (&17)

Breakpoints are not supported by the coprocessor software.
This call returns the error "SWI not known".
See PRM1 page 366.

OS_BreakCtrl (&18)

Obsolete, use OS_ChangeEnvironment instead.

This call returns the error "SWI not known".

OS_UnusedSWI (&19)

Obsolete, use OS_ChangeEnvironment instead.

This call returns the error "SWI not known".

OS_UpdateMEMC (&1A)

As there is no MEMC device on the circuit board this call is not available.

This call returns the error "SWI not known".

See PRM1 page 366.

OS_SetCallBack (&1B)

The coprocessor does not issue call backs, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 308.

OS_Mouse (&1C)

Read the mouse position and button state. The coprocessor attempts to read the mouse position using ADVAL(7) and ADVAL(8) and the buttons with INKEY(-10) INKEY(-11) INKEY(-12) should the host machine support this.

See PRM1 page 699.

OS_Heap (&1D)

No heap management facilities are offered on the coprocessor, so this call is not available.

This call returns the error "SWI not known".

See PRM1 page 368.

OS_Module (&1E)

Perform miscellaneous operations on relocatable modules in memory.

The coprocessor only supports a single application module, by default this is BASIC, which itself only ever calls OS_Module once which is the reason code which is supported.

	Enter module
On entry	R0=reason code 2
	R1=pointer to module name
	R2=pointer to command line parameters
On exit	Does not exit unless an error occurs
Interrupts	IRQs undefined, FIQs enabled
Processor mode	SVC32
Reentrancy	Not defined

See PRM1 page 224.

OS_Claim (&1F)

Registers a routine at the start of the list for the given software vector chain, eliminating any duplicate claimants. A duplicate claimant is defined as one previously registered with the same parameters.

See PRM1 page 66.

OS_Release (&20)

Removes the first routine from a software vector chain with matching parameters to those used at registration. If there are duplicate claimants, such as those added with OS_AddToVector, this will need to be called multiple times to remove them all.

See PRM1 page 68.

OS_ReadUnsigned (&21)

Read a number string in, optionally in base 16 if prefixed by '&' or base n if prefixed by 'n_'.

See PRM1 page 448.

OS_GenerateEvent (&22)

Simulates an event occurring. For events that are supported by the host these must first have been enabled with OS_Byte 14, all other events call EventV directly with no checking.

See PRM1 page 152.

OS_ReadVarVal (&23)

The coprocessor does not maintain any system variables, therefore this call always returns an error denoting that the variable was not found.

See PRM1, page 309.

OS_SetVarVal (&24)

The coprocessor does not maintain any system variables, therefore this call is not available.

This call returns the error "SWI not known".

See PRM1, page 311.

OS_GSInit (&25)

General string processing is not included in the coprocessor, therefore this call is not available.

This call returns the error "SWI not known".

See PRM1, page 450.

OS_GSRead (&26)

General string processing is not included in the coprocessor, therefore this call is not available.

This call returns the error "SWI not known".

See PRM1, page 452.

OS_GSTrans (&27)

General string processing is not included in the coprocessor, therefore this call is not available.

This call returns the error "SWI not known".

See PRM1, page 454.

OS_BinaryToDecimal (&28)

Converts a signed 4 byte integer into text.

Calls OS_ConvertInteger4 internally to perform the conversion, then corrects the exit parameters.

See PRM1 page 456.

OS_FSControl (&29)

The host runs all filing system code, therefore this call is not available.

This call returns the error "SWI not known".

See PRM2 page 77 and PRM5a.

OS_ChangeDynamicArea (&2A)

The coprocessor does not support dynamic areas, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 377.

OS_GenerateError (&2B)

Generates an error with the block given in R0.

See PRM1 page 45.

OS_ReadEscapeState (&2C)

Returns in the C flag whether there is a pending Escape.

See PRM1 page 912.

OS_EvaluateExpression (&2D)

The coprocessor does not maintain any system variables, therefore this call is not available.

This call returns the error "SWI not known".

See PRM1, page 457.

OS_SpriteOp (&2E)

The host does not directly support most sprite operations, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 761.

OS_ReadPalette (&2F)

This call is not available. OS_Word reason code &0B offers equivalent functionality for the BBC micro palette.

This call returns the error "SWI not known".

See PRM1 page 701.

OS_ServiceCall (&30)

The coprocessor does not support service calls, however this SWI will silently return with all registers preserved denoting that the service was not claimed.

See PRM1 page 250.

OS_ReadVduVariables (&31)

This call is not available. OS_Byte reason code &A0 offers equivalent functionality for the BBC micro VDU variables.

This call returns the error "SWI not known".

See PRM1 page 703.

OS_ReadPoint (&32)

Read the logical colour of the given coordinate. On the coprocessor the X and Y coordinates are truncated to 16 bits then is translated internally into OS_Word reason code &09.

See PRM1 page 707.

OS_UpCall (&33)

Issues an upcall warning to the UpCall handler set by OS_ChangeEnvironment.

See PRM1 page 177.

OS_CallAVector (&34)

Calls a software vector chain with entry conditions required by the vector in question.

See PRM1 page 70.

OS_ReadModeVariable (&35)

This call is not available. OS_Byte reason code &A0 offers equivalent functionality for the BBC micro mode variables.

This call returns the error "SWI not known".

See PRM1 page 709.

OS_RemoveCursors (&36)

A cursor stack is not implemented on the coprocessor, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 712.

OS_RestoreCursors (&37)

A cursor stack is not implemented on the coprocessor, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 714.

OS_SWINumberToString (&38)

Convert the number of a SWI into its corresponding SWI name.

See PRM1, page 459.

OS_SWINumberFromString (&39)

Convert the name of a SWI into its corresponding SWI number.

See PRM1, page 461.

OS_ValidateAddress (&3A)

Confirms that the region specified lies within logical RAM.
See PRM1 page 379.

OS_CallAfter (&3B)

The coprocessor does not issue call backs, therefore this is not available.
This call returns the error "SWI not known".
See PRM1 page 429.

OS_CallEvery (&3C)

The coprocessor does not issue call backs, therefore this is not available.
This call returns the error "SWI not known".
See PRM1 page 431.

OS_RemoveTickerEvent (&3D)

The coprocessor does not issue call backs, therefore this is not available.
This call returns the error "SWI not known".
See PRM1 page 433.

OS_InstallKeyHandler (&3E)

As there is no keyboard plugged into the circuit board this call is not available.
This call returns the error "SWI not known".
See PRM1 page 914.

OS_CheckModeValid (&3F)

This call is not available. OS_Byte reason code &85 offers equivalent functionality for the BBC micro mode validation.
This call returns the error "SWI not known".
See PRM1 page 715.

OS_ChangeEnvironment (&40)

Read or write the enviroment handler (or value) given in R0.
See PRM1 page 315.

OS_ClaimScreenMemory (&41)

The coprocessor does not support dynamic areas, therefore this is not available.
This call returns the error "SWI not known".
See PRM1 page 380.

OS_ReadMonotonicTime (&42)

Reads the number of centiseconds elapsed since power up.
See PRM1, page 434.

OS_SubstituteArgs (&43)

The coprocessor does not maintain any system variables, therefore this call is not available.

This call returns the error "SWI not known".

See PRM1, page 463.

OS_PrettyPrintCode (&44)

The compressed text dictionary is not included in the coprocessor, therefore this call is not available.

This call returns the error "SWI not known".

See PRM1, page 518.

OS_Plot (&45)

Plot operation through the VDU driver.

This call is translated into a VDU25 sequence by truncating the X and Y coordinates given to 16 bits.

On entry	R0=plot operation
	R1=X coordinate
	R2=Y coordinate
On exit	R0-R2 corrupt

Interrupts	IRQs undefined, FIQs enabled
Processor mode	SVC32
Reentrancy	Not reentrant

See PRM1 page 717.

OS_WriteN (&46)

Write the first N characters of the string pointed to by R0.

See PRM1 page 522.

OS_AddToVector (&47)

Similar to OS_Claim, except that no check is made for duplicates. Therefore, this allows the same routine to be added multiple times if desired.

See PRM1 page 72.

OS_WriteEnv (&48)

Set the values returned by OS_GetEnv.

See PRM1 page 317.

OS_ReadArgs (&49)

General string processing is not included in the coprocessor, therefore this call is not available.

This call returns the error "SWI not known".

See PRM1, page 465.

OS_ReadRAMFsLimits (&4A)

The coprocessor does not run any filing systems itself, therefore this call is not available.

This call returns the error "SWI not known".

See PRM1, page 382.

OS_ClaimDeviceVector (&4B)

No calls pass through software vectors inside the coprocessor, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 121.

OS_ReleaseDeviceVector (&4C)

No calls pass through software vectors inside the coprocessor, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 123.

OS_DelinkApplication (&4D)

The coprocessor only supports a single application, therefore this call is not available.

This call returns the error "SWI not known".

See PRM1, page 74.

OS_RelinkApplication (&4E)

The coprocessor only supports a single application, therefore this call is not available.

This call returns the error "SWI not known".

See PRM1, page 76.

OS_HeapSort (&4F)

This call is not available.

This call returns the error "SWI not known".

See PRM1, page 937.

OS_ExitAndDie (&50)

Exits the last program started and returns to the most recent exit handler, killing the module named in R3.

The coprocessor does not maintain any system variables, therefore the return code/error block/magic word are all ignored at present. For compatibility they should still be set as appropriate.

On entry	R0=pointer to error block
	R1="ABEX" if the return code is to be tested
	R2=return code
	R3=pointer to module name
On exit	Does not exit

Interrupts	IRQs unaltered, FIQs enabled
Processor mode	USR32
Reentrancy	Not reentrant

See PRM1 page 318.

OS_ReadMemMapInfo (&51)

Returns the size and number of pages in logical RAM.

See PRM1 page 383.

OS_ReadMemMapEntries (&52)

As there is no MEMC device on the circuit board this call is not available.

This call returns the error "SWI not known".

See PRM1 page 384.

OS_SetMemMapEntries (&53)

As there is no MEMC device on the circuit board this call is not available.

This call returns the error "SWI not known".

See PRM1 page 386.

OS_AddCallBack (&54)

The coprocessor does not issue call backs, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 319.

OS_ReadDefaultHandler (&55)

Reads the default values used for OS_ChangeEnvironment when no other handler exists.

See PRM1 page 321.

OS_SetECFOrigin (&56)

This call is not available. Sequence VDU23,17,6 offers equivalent functionality for the BBC micro ECF origin where supported by the VDU system.

This call returns the error "SWI not known".

See PRM1 page 718.

OS_SerialOp (&57)

The on board serial port this is intended for development use only, therefore this call is not available.

This call returns the error "SWI not known".

See PRM2 page 459.

OS_ReadSysInfo (&58)

Read information on the current system.

A number of subreason codes are implemented.

	Read configured screen size
On entry	R0=reason code 0
On exit	R0=number of bytes allocated to the configured screen mode

Read configured screen mode

On entry	R0=reason code 1
On exit	R0=mode
	R1=monitor type
	R2=sync

Read machine id and ASIC presence flags

On entry	R0=reason code 2
On exit	R0=IOEB ASIC presence flags
	R1=82C710 presence flags
	R2=LCD ASIC presence flags
	R3=low half of machine id (or zero if none available)
	R4=high half of machine id (or zero if none available)

Read 82C71x features

On entry	R0=reason code 3
On exit	R0=basic features
	R1=extra features mask
	R2=extra features 1
	R3=extra features 2
	R4=extra features 3

Read ethernet address

On entry	R0=reason code 4
On exit	R0=low 32 bits of MAC address (or zero if none available)
	R1=high 16 bits of MAC address (or zero if none available)

Read raw machine id

On entry	R0=reason code 5
On exit	R0=low 32 bits of unique number (or zero if none available)
	R1=high 32 bits of unique number (or zero if none available)

Read platform class

On entry	R0=reason code 8
On exit	R0=class
	R1=miscellaneous platform flags

Read version information

On entry	R0=reason code 9
	R1=value to read
	0: version number as string
	1: part number as string
	2: build date as string
	3: dealer name as string
	4: registered user name as string
	5: registered user address as string
On exit	R0=pointer to requested string (or zero if not available)
On exit	R0-R1 preserved

Interrupts	IRQs unaltered, FIQs enabled
Processor mode	SVC32
Reentrancy	Reentrant

See PRM1 page 719 and PRM5a.

OS_Confirm (&59)

Waits for some response from the user, and determines if the answer was "Yes" or "No".

See PRM1 page 940.

OS_ChangedBox (&5A)

Graphics clipping is performed by the host, who does not maintain the changed area, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 724.

OS_CRC (&5B)

This call is not available.

This call returns the error "SWI not known".

See PRM1, page 942.

OS_ReadDynamicArea (&5C)

The coprocessor does not support dynamic areas, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 388.

OS_PrintChar (&5D)

Send character to the printer.

The coprocessor translates this into an OS_Byte 138 operation to buffer id number 3.

See PRM1 page 521.

OS_ChangeRedirection (&5E)

Command line input and output cannot be redirected on the coprocessor, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 931.

OS_RemoveCallBack (&5F)

The coprocessor does not issue call backs, therefore this is not available.

This call returns the error "SWI not known".

See PRM1 page 322.

OS_FindMemMapEntries (&60)

As there is no MEMC device on the circuit board this call is not available.

This call returns the error "SWI not known".

See PRM1 page 390.

OS_SetColourCode (&61)

Define the foreground or background colour of VDU text and graphics. On the coprocessor some flag bits in R0 are not available, for example those which set the ECF patterns using 32 bit words, and will be faulted.

On entry   R0=flags

                         bits 3-0: GCOL plot action

                         bit 4: set to alter background colour (else foreground)

                         bit 5: must be clear

                         bit 6: set to change text colour (else graphics)

                         bit 7: must be clear

                         bits 31-8: reserved

                R1=colour number (flag bit 5 set) or pointer to ECF block (flag bit 5 clear)

On exit   R0-R1 preserved

Interrupts   IRQs undefined, FIQs enabled

Processor mode   SVC32

Reentrancy   Not defined

See PRM1 page 726 and PRM5a.

OS_ClaimSWI (&62)

Though assigned a name, this SWI is not implemented in RISC OS.

This call returns the error "SWI not known".

OS_ReleaseSWI (&63)

Though assigned a name, this SWI is not implemented in RISC OS.

This call returns the error "SWI not known".

OS_Pointer (&64)

The host does not directly support a graphical pointer, therefore this is not available.

This call returns the error "SWI not known".

OS_ScreenMode (&65)

Operations on the screen modes. The coprocessor relies on the host for screen display therefore only supports the ability to set and read the current mode using mode numbers supported by the host.

                Select a screen mode

On entry   R0=reason code 0

                R1=mode number

On exit   R0-R1 preserved

                Return the mode specifier for the current mode

On entry   R0=reason code 1

On exit   R0 preserved

                R1=mode number

Interrupts	IRQs undefined, FIQs enabled
Processor mode	SVC32
Reentrancy	Not defined

See PRM5a.

OS_DynamicArea (&66)

The coprocessor does not support dynamic areas, therefore this is not available.
This call returns the error "SWI not known".
See PRM5a.

OS_AbortTrap (&67)

Though assigned a name, this SWI is not implemented in RISC OS.
This call returns the error "SWI not known".

OS_Memory (&68)

General operations on memory blocks. A limited subset of this call is available.

Read amount of a type of memory installed

On entry   R0=reason code and flags

        bits 7-0: 8

        bits 11-8: type

            1 = DRAM

            2 = VRAM

            3 = ROM

            4 = I/O

            5 = Soft ROM

        bits 31-12: zero

On exit   R1=number of pages of the requested type of memory

        R2=size of a page

Read controller presence and base

On entry   R0=reason code and flags

        bits 7-0: 9

        bits 31-8: zero

        R1=controller ID

        bits 7-0: sequence number

        bits 31-8: type

            0 = ECTCR

            1 = EASI space

            2 = VIDC1

            3 = VIDC20

            4 = S space

            5 = Extension ROMs

            6 = Tube space

On exit   R1=controller base, or zero if not present

Interrupts	IRQs undefined, FIQs enabled
Processor mode	SVC32
Reentrancy	Not defined

See PRM5a.

OS_ClaimProcessorVector (&69)

Claims one of the hardware exception vectors by altering the hardware vector table.

See PRM5a.

OS_Reset (&6A)

Simulates a hard reset under software control.

See PRM5a.

OS_MMUControl (&6B)

Miscellaneous operations on the MMU.

The coprocessor does not have an MMU, but as the SWI doesn't normally return any results so it does nothing and is not faulted.

OS_ResyncTime (&6C)

Reread the hardware real time clock and refresh the software copy.

The coprocessor does not have an RTC, but as the SWI doesn't normally return any results it does nothing and is not faulted.

OS_PlatformFeatures (&6D)

Return various details about platform capabilities. Only the code feature flags reason code is supported.

On entry	R0=reason code and flags
	0: read code features
On exit	R0=code feature flags
	bit 0 set: must use OS_SynchroniseCodeAreas when modifying code areas
	bit 1 set: enabling then disabling IRQs doesn't give time for one to occur
	bit 2 set: exception vectors can only be read in 32 bit mode
	bit 3 set: when storing PC, PC+8 is stored rather than PC+12
	bit 4 set: data aborts occur with "full early" timing
	bit 5 set: CPU has separate I and D caches
	bit 6 set: operating system executes in 32 bit mode
	bit 7 set: 26 bit mode is not available
	bit 8 set: processor is M class (supports UMULL etc)
	bit 9 set: processor supports Thumb
	bit 10 set: processor is E class (supports QADD etc)
	R1=pointer to IRQ routine if R0 bit 1 was set

Interrupts	IRQs unaltered, FIQs enabled
Processor mode	SVC32
Reentrancy	Reentrant

OS_SynchroniseCodeAreas (&6E)

Operations on the code cache.

The coprocessor has a unified cache, but this call is still implemented as a simple way to forcefully flush the cache.

On entry	R0=flags
	bits 31-1: reserved, must be 0
	bit 0 set: only synchronise a range
	R1=low address to synchronise if bit 0 of flags set
	R2=high address (inclusive)
On exit	R0-R2 preserved
Interrupts	IRQs unaltered, FIQs enabled
Processor mode	SVC32
Reentrancy	Not reentrant

OS_CallASWI (&6F)

Call a SWI whose number is determined at run time.

On entry	R0-R9=as required by the called SWI
	R10=SWI number to call
On exit	R0-R9=as defined by the called SWI
	R10=preserved
Interrupts	As defined by the called SWI
Processor mode	As defined by the called SWI
Reentrancy	As defined by the called SWI

OS_AMBControl (&70)

Miscellaneous operations on the AMB.

The coprocessor does not have an AMB, but as the SWI doesn't normally return any results so it does nothing and is not faulted

OS_CallASWIR12 (&71)

Call a SWI whose number is determined at run time.

On entry	R0-R9=as required by the called SWI
	R12=SWI number to call
On exit	R0-R9=as defined by the called SWI
	R12=preserved
Interrupts	As defined by the called SWI
Processor mode	As defined by the called SWI
Reentrancy	As defined by the called SWI

OS_SpecialControl (&72)

This call is not available.
This call returns the error "SWI not known".

OS_EnterUSR32 (&73)

This call is not available.
This call returns the error "SWI not known".

OS_EnterUSR26 (&74)

This call is not available.
This call returns the error "SWI not known".

OS_VIDCDivider (&75)

This call is not available.
This call returns the error "SWI not known".

OS_NVMemory (&76)

This call is not available.
This call returns the error "SWI not known".

OS_ClaimOSSWI (&77)

This call is not available.
This call returns the error "SWI not known".

OS_TaskControl (&78)

This call is not available.
This call returns the error "SWI not known".

OS_DeviceDriver (&79)

This call is not available.
This call returns the error "SWI not known".

OS_Hardware (&7A)

This call is not available.
This call returns the error "SWI not known".

OS_IICOp (&7B)

This call is not available.
This call returns the error "SWI not known".

OS_LeaveOS (&7C)

Switch the processor to USR32 mode and return.

OS_ReadLine32 (&7D)

Read a line of input.

This call is implemented locally on the coprocessor using OS_ReadC, and should be used in preference to OS_ReadLine.

On entry	R0=pointer to buffer
	R1=size of buffer
	R2=lowest permitted input character
	R3=highest permitted input character (inclusive)
	R4=flags
	bit 31 set: echo only characters being stored in the buffer
	bit 30 set: echo using password style entry
	bit 7-0 set: the password character to use
On exit	R1=nunber of characters stored excluding terminator, R0 corrupted, R2-R3 corrupted
Interrupts	IRQs enabled, FIQs enabled
Processor mode	SVC32
Reentrancy	Not reentrant

See also PRM1 page 910.

OS_SubstituteArgs32 (&7E)

The coprocessor does not maintain any system variables, therefore this call is not available.

This call returns the error "SWI not known".

OS_HeapSort32 (&7F)

This call is not available.

This call returns the error "SWI not known".

OS_Convert<various> (&C0-&FF)

This groups of calls converts between various input forms to text.

The following conversions are supported:

    OS_ConvertHex1 (&D0)
    OS_ConvertHex2 (&D1)
    OS_ConvertHex4 (&D2)
    OS_ConvertHex6 (&D3)
    OS_ConvertHex8 (&D4)
    OS_ConvertCardinal1 (&D5)
    OS_ConvertCardinal2 (&D6)
    OS_ConvertCardinal3 (&D7)
    OS_ConvertCardinal4 (&D8)
    OS_ConvertInteger1 (&D9)
    OS_ConvertInteger2 (&DA)
    OS_ConvertInteger3 (&DB)
    OS_ConvertInteger4 (&DC)
    OS_ConvertBinary1 (&DD)
    OS_ConvertBinary2 (&DE)

      OS_ConvertBinary3 (&DF)
      OS_ConvertBinary4 (&E0)
      OS_ConvertSpacedCardinal1 (&E1)
      OS_ConvertSpacedCardinal2 (&E2)
      OS_ConvertSpacedCardinal3 (&E3)
      OS_ConvertSpacedCardinal4 (&E4)
      OS_ConvertSpacedInteger1 (&E5)
      OS_ConvertSpacedInteger2 (&E6)
      OS_ConvertSpacedInteger3 (&E7)
      OS_ConvertSpacedInteger4 (&E8)
      OS_ConvertFixedNetStation (&E9)
      OS_ConvertNetStation (&EA)
      OS_ConvertFixedFileSize (&EB)
      OS_ConvertFileSize (&EC)

The following are not supported:
      OS_ConvertStandardDateAndTime (&C0)
      OS_ConvertDateAndTime (&C1)
it is suggested that OS_Word reason code &0E is used to read the time as a string directly.

OS_WriteI (&100-1FF)

Output the character encoded in the bottom byte of the SWI number to the VDU.
See PRM1 page 523.

Debugger_Disassemble (&40380)

Optionally, the ARM Tube OS includes a number of debugging aids to help developers debug programs being developed on the coprocessor.
The command *MEMORYI uses this SWI to disassemble opcodes in memory.
See PRM4 page 215.

Debugger_DisassembleThumb (&40381)

Optionally, the ARM Tube OS includes a number of debugging aids to help developers debug programs being developed on the coprocessor.
This call is not available.
This call returns the error "SWI not known".