SWI Codes for the ARMSim Simulator

1. Basic SWI Operations

The SWI codes numbered in the range 0 to 255 inclusive are reserved for basic instructions that ARMSim needs and should not be altered. Their list is shown in Table 1 and examples of their use follow. The use of "EQU" is strongly advised to substitute the actual numerical code values.

Table 1: SWI operations (0x00 - 0xFF)

Opcode	Description and Action	Inputs	Outputs
`swi 0x00`	Display Character on Console	r0: the character	
`swi 0x02`	Display String on Console	r0: address of a null terminated ASCII string	
`swi 0x07`	Prompt User for an Integer	r0: address of a null terminated ASCII string	r0: the integer
`swi 0x11`	Halt Execution		
`swi 0x12`	Allocate Block of Memory on Heap	r0: block size in bytes	r0: address of block
`swi 0x13`	Deallocate All Heap Blocks		
`swi 0x66`	Open File (mode values are: 0 for input, 1 for output, 2 for appending)	r0: file name, i.e. address of a null terminated ASCII string containing the name r1: mode	r0: file handle If the file does not open, a result of -1 is returned
`swi 0x68`	Close File	r0: file handle	
`swi 0x69`	Write String to a File	r0: file handle r1: address of a null terminated ASCII string	
`swi 0x6a`	Read String from a File	r0: file handle r1: destination address r2: max bytes to store	r0: number of bytes stored
`swi 0x6b`	Write Integer to a File	r0: file handle r1: integer	
`swi 0x6c`	Read Integer from a File	r0: file handle	r0: the integer
`swi 0x6d`	Get the current time (ticks)		r0: the number of ticks (milliseconds)

2. Detailed Descriptions and Examples for Codes in Table 1

Display Character on Screen: swi 0x00

Displays one character in the output window.

```
    mov    r0,#'X
    swi    0x00
```

Display String on Screen: swi 0x02

Displays a string in the output window.

```
    ldr    r0,=MyString
    swi    0x02
    ...
MyString: .asciz "Hello There\n"
```

Prompt User for an Integer: swi 0x07

Creates a pop-up window which displays the supplied message and waits until the user types a number into that window.

```
    ldr    r0,=Prompt
    swi    0x07
    ldr    r1,=Number
    str    r0,[r1]
    ...
Number:    .word  0
Prompt:    .asciz "Enter a number"
```

Halt Execution: swi 0x11

Stops the program.

```
    swi    0x11
```

Allocate Block of Memory on Heap: swi 0x12

Obtain a new block of memory from the heap area of the program space.

If no more memory is available, the special result -1 is returned and the C bit is set in the CPSR.

```
    mov    r0,#28    ; get 28 bytes
    swi    0x12
    ldr    r1,=Address
    str    r0,[r1]
    ...
Address:   .word  0
```

Deallocate All Heap Blocks: swi 0x13

Causes all previously allocated blocks of memory in the heap area to be considered as deallocated (thus allowing the memory to be reused by future requests for memory blocks).

```
    swi    0x13
```

Open File: swi 0x66

Opens a text file for input or output. The file name is passed via r0. Register r1 specifies the file access mode. If r1=0, an existing text file is to be opened for input. If r1=1, a file is opened for output (if that file exists already, it will be overwritten, otherwise a new file is created). If r1=2, an existing text file is opened in append mode, so that any new text written to the file will be added at the end.
If the file is opened successfully, a positive number (the file handle) is returned in r0. Otherwise, a result of -1 is returned and the C bit is set.

```
    ldr    r0,=FileName
    mov    r1,#0  ; input mode
    swi    0x66
    bcs    NoFileFound
    ldr    r1,=InputFileHandle
    str    r0,[r1]
    ...
NoFileFound:
    ldr    r0,=ErrMsg
    swi    0x06
    ...
InputFileHandle: .word 0
FileName: .asciz "mydata.txt"
ErrMsg: .asciz "File not found"
```

Note: The default location for the file is the same folder as the assembler source code file. If another location is desired, a full path to the file location can be used. For example, the code shown on the right opens (or creates) a text file in the Windows Temporary directory.

```
    ldr    r0,PathName
    mov    r1,#1  ; output mode
    swi    0x66
    ...
PathName:
     .asciz "C:\\TEMP\\MyFile.txt"
```

Close File: swi 0x68

Closes a previously opened file.

(Unless a file is closed, it cannot be inspected or edited by another program such as TextPad.)

```
    ldr    r0,=InputFileHandle
    ldr    r0,[r0]
    swi    0x68
```

Write String to a File: swi 0x69

Writes the supplied string to the current position in the open output file.

The file handle, passed in r0, must have been obtained by an earlier call to the Open File `swi` operation.

```
    ldr    r0,=OutputFileHandle
    ldr    r0,[r0]
    ldr    r1,=TextString
    swi    0x69
    bcs    WriteError
    ...
TextString: .asciz "Answer = "
```

Note: The special file handle value of 1 can be used to write a string to the output window of the ARM simulator (giving the same behaviour as `swi 0x02`). A brief example appears on the right.

```
    mov    r0,#1
    ldr    r1,=Message
    swi    0x69   ; display message
    ...
Message: .asciz "Hello There\n"
```

Read String from a File: swi 0x6a

Reads a string from a file. The input file is identified by a file handle passed in r0; r1 is the address of an area which the string is to be copied into; r2 is the maximum number of bytes to store into memory.

One line of text is read from the file and copied into memory and a null byte terminator is stored at the end. The line is truncated if it is too long to store in memory. The result returned in r0 is the number of bytes (including the null terminator) stored in memory.

```
    ldr    r0,=InputFileHandle
    ldr    r0,[r0]
    ldr    r1,=CharArray
    mov    r2,#80
    swi    0x6a
    bcs    ReadError
    ...
InputFileHandle: .word 0
CharArray:       .skip 80
```

Write Integer to a File: swi 0x6b

Converts the signed integer value passed in r1 to a string and writes that string to the file identified by the file handle passed in r0.

```
    ldr    r0,=OutputFileHandle
    ldr    r0,[r0]
    mov    r1,#99
    swi    0x6b  ; write 99 to file
    bcs    WriteError
    ...
OutputFileHandle: .word 0
```

Note: The special file handle value of 1 can be used to write the integer to the output window. An example appears on the right.

```
    mov    r0,#1
    mov    r1,#99
    swi    0x6b ; display 99
```

Read Integer from a File: swi 0x6c

Reads a signed integer from a file. The file is identified by the file handle passed in r0. The result is returned in r0.

If a properly formatted number is not found in the input, the C bit is set and r0 is unchanged.

```
    ldr    r0,=InputFileHandle
    ldr    r0,[r0]
    swi    0x6c
    bcs    ReadError
    ldr    r1,=Number
    str    r0,[r1]
    ...
InputFileHandle: .word 0
Number:          .word 0
```

3. SWI Operations for Plug-Ins

The SWI codes numbered above 255 have special purposes; they are mainly used for interaction with plug-in modules which can be loaded with the ARMSim simulator. Table 2 provides a current list of these codes. Examples of their use follow with illustrations of the corresponding component. However, further details for related plug-in modules are not included here. The use of "EQU" is strongly advised to substitute the actual numerical code values.

Table 2: SWI operations > 0xFF

Opcode	Description and Action	Inputs	Outputs
`swi 0x200`	Set the 8-Segment Display to light up.	r0: the 8-segment Pattern (see below in Figure 1)	The appropriate segments light up to display a number or a character
`swi 0x201`	Set the two LEDs to light up.	r0: the LED Pattern, where: Left LED on = `0x02` Right LED on = `0x01` Both LEDs on = `0x03` (i.e. the bits in position 0 and 1 of r0 must each be set to 1 appropriately)	Either the left LED is on, or the right, or both
`swi 0x202`	Check if one of the Black Buttons has been pressed.	None	r0 = the Black Button Pattern, where: Left black button pressed returns r0 = `0x02;` Right black button pressed returns r0 = `0x01;` (i.e. the bits in position 0 and 1 of r0 get assigned the appropriate values).
`swi 0x203`	Check if one of the Blue Buttons has been pressed.	None (see below in Figure 2)	r0 = the Blue Button Pattern (see below in Figure 2).

Table 2: SWI operations > 0xFF (Continued)

Opcode	Description and Action	Inputs	Outputs
`swi 0x204`	Display a string on the LCD screen	r0: x position coordinate on the LCD screen (0-39); r1: y position coordinate on the LCD screen (0-14); r2: Address of a null terminated ASCII string. Note: (x,y) = (0,0) is the top left and (0,14) is the bottom left. The display is limited to 40 characters per line.	The string is displayed starting at the given position of the LCD screen.
`swi 0x205`	Display an integer on the LCD screen	r0: x position coordinate on the LCD screen (0-39); r1: y position coordinate on the LCD screen (0-14); r2: integer to print. Note: (x,y) = (0,0) is the top left and (0,14) is the bottom left. The display is limited to 40 characters per line	The string is displayed starting at the given position of the LCD screen.
`swi 0x206`	Clear the display on the LCD screen	None	Blank LCD screen.

Table 2: SWI operations > 0xFF (Continued)

Opcode	Description and Action	Inputs	Outputs
`swi 0x207`	Display a character on the LCD screen	r0: x position coordinate on the LCD screen (0-39); r1: y position coordinate on the LCD screen (0-14); r2: the character. Note: (x,y) = (0,0) is the top left and (0,14) is the bottom left. The display is limited to 40 characters per line	The string is displayed starting at the given position of the LCD screen.
`swi 0x208`	Clear one line in the display on the LCD screen	r0: line number (y coordinate) on the LCD screen	Blank line on the LCD screen.

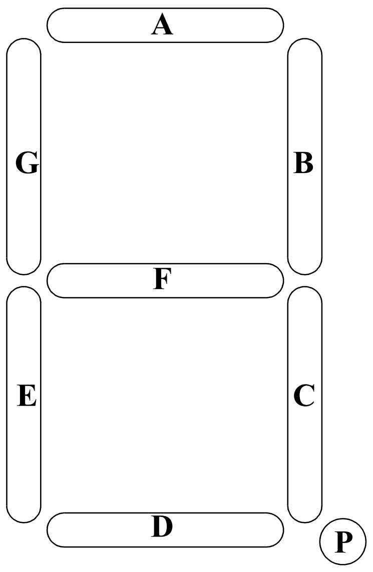
Detailed Descriptions and Examples for Codes in Table 2

Set the 8-Segment Display to light up: swi 0x200

The appropriate segments light up to display a number or a character. The pattern of segments to be lit up is assigned to register `r0` before the call to `swi 0x200`. Figure 1 shows the arrangements of segments, and an example follows. Each segment is logically labelled and its byte code is shown in the list alongside. In order to display the number "3", segments "*A*", "*B*", "*C*", "*D*" and "*F*" must be illuminated. The code to be assigned to `r0` is computed by the logical `OR` of the individual byte codes.

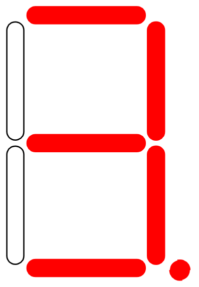
Below some segments of code are shown as examples for the 8-segment Display. The "`.equ`" statements are useful for accessing the byte values associated with the labels of each segment. An example of a possible declaration of data is also given for the display of integers, where the

Figure 1: The Pattern for the 8-Segment Display



Display byte values	
A	`0x80`
B	`0x40`
C	`0x20`
P	`0x10`
D	`0x08`
E	`0x04`
F	`0x02`
G	`0x01`

Example: number "3"



For example, the number 3 plus the right hand dot would have a pattern value computed as the logical OR of the values of the segments "A,B,C,D,F,P" to form the integer: `0x80 | 0x40 | 0x20 | 0x08 | 0x02 | 0x10 = 0xFA`, to be assigned to r0.

byte values representing a particular number are already "`OR`ed" together within the array data structure and can be indexed appropriately.

Use "`.equ`" statements to set up the byte value of each segment of the Display.

```
.equ   SEG_A,0x80
.equ   SEG_B,0x40
.equ   SEG_C,0x20
.equ   SEG_D,0x08
.equ   SEG_E,0x04
.equ   SEG_F,0x02
.equ   SEG_G,0x01
.equ   SEG_P,0x10
```

In the example of Figure 1, the number 3 has a pattern value computed as the logical OR of the values of the segments "`A,B,C,D,F`" to form the integer:
`0x80 | 0x40 | 0x20 | 0x08 | 0x02 | = 0xFA,` to be assigned to `r0`.

It may be easier to use a data declaration for an array of words and then index into it. Each element can be initialized to contain the value representing a number by having the appropriate byte values "`OR`ed" together.

A possible data declaration for an array of words which can be indexed to obtain the appropriate value for the number to be displayed.

```
Digits:
    .word  SEG_A|SEG_B|SEG_C|SEG_D|SEG_E|SEG_G    @0
    .word  SEG_B|SEG_C                            @1
    .word  SEG_A|SEG_B|SEG_F|SEG_E|SEG_D          @2
    .word  SEG_A|SEG_B|SEG_F|SEG_C|SEG_D          @3
    .word  SEG_G|SEG_F|SEG_B|SEG_C                @4
    .word  SEG_A|SEG_G|SEG_F|SEG_C|SEG_D          @5
    .word  SEG_A|SEG_G|SEG_F|SEG_E|SEG_D|SEG_C    @6
    .word  SEG_A|SEG_B|SEG_C                      @7
    .word  SEG_A|SEG_B|SEG_C|SEG_D|SEG_E|SEG_F|SEG_G@8
    .word  SEG_A|SEG_B|SEG_F|SEG_G|SEG_C          @9
    .word  0                            @Blank display
```

An example of a possible routine to display a number in the 8-segment Display using the declarations given above is shown here.

Register `R0` and `R1` are input parameters, where `R0` contains the integer to be displayed and `R1` contains "`1`" to display the "`P`" segment, or "`0`" otherwise.

```
@ *** Display8Segment (Number:R0; Point:R1) ***
@ Displays the number 0-9 in R0 on the
@ LED 8-segment display
@ If R1 = 1, the point is also shown
[1] Display8Segment:
      stmfd sp!,{r0-r2,lr}
[2]   ldr   r2,=Digits
[3]   ldr   r0,[r2,r0,lsl#2]
[4]   tst   r1,#0x01          @if r1=1,
[5]   orrne r0,r0,#SEG_P      @then show "P"
[6]   swi   0x200
[7]   ldmfd sp!,{r0-r2,pc}
```

In line [3], register `r0` is assigned the byte value corresponding to the indexed element of the array digits. For example, to display the number "3", after execution of line [2] the input register `r0` should contain the integer value 3 and register `r2` contains the address of the array "`Digits`". Then the computation implied by "`[r2,r0,lsl#2]`" adds 12 bytes to the address currently in `r2` (i.e. `r0` shifted left by 2 positions, which evaluates to "3" x 4 = 12) and loads the word in position 3 of the array, namely: `.word SEG_A|SEG_B|SEG_F|SEG_C|SEG_D`. In fact, this uses the segments "`A,B,C,D,F`" to display the correct number. In line [4] the content of `r1` is tested. If `r1` = 1 then the segment "`P`" is added to the display, with its value `OR`ed with the previous ones in `r0`.

Set the two LEDs to light up: swi 0x201

Light up the LEDs: the left or the right or both, according to the value supplied by `r0`.

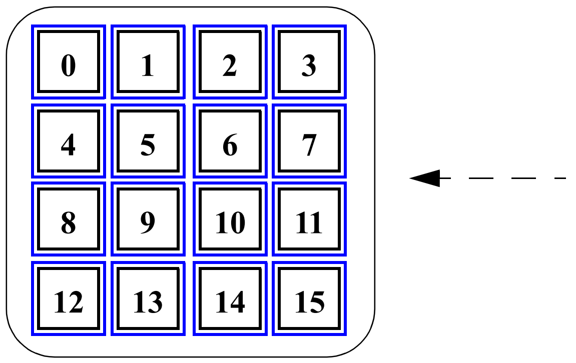
```
mov    r0,#0x02
swi    0x201      @ left LED on
mov    r0,#0x01
swi    0x201      @ right LED on
mov    r0,#0x03
swi    0x201      @ both LEDs on
```

Check if one of the Black Buttons has been pressed: swi 0x202

After the call with `swi 0x202`, test the content of `r0`: if `r0=2` then the left black button was pressed; if `r0=1`, the right black button was pressed.

```
swi    0x202
cmp    r0,#0x02
beq    ActOnLeftBlack
bal    ActOnRightBlack
```

Figure 2: The Blue Buttons (the Keypad)


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
The keypad with 16 blue buttons as depicted in the board view.

Each button has a corresponding bit position in the 16-bit lower portion of register 0. When a button is pressed, the corresponding bit is set. For example, when the button in position "7" is pressed, the `swi 0x203` instruction returns `r0 = 0x80`, that is,

`r0 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0`

in binary, where the bit in position "7" has been set.

Check if one of the Blue Buttons has been pressed: swi 0x203

After the call with `swi 0x203`, test the content of `r0`. The number in `r0` corresponds to the position of the blue button as depicted in Figure 2. For example, if `r0=128` (which is `1<<7`) then the blue button in position 7 was pressed.

```
swi    0x203
tst    r0,#(1<<7)
bne    Button7pressed
. . . . .
```

Display a string on the LCD screen: swi 0x204

Display the string whose address is supplied in `r2` on the LCD screen at position (x,y), where here `r0`=x=4 and `r1`=y=1 (that is, line 1 at column 4).

```
    mov    r0,#4
    mov    r1,#1
    ldr    r2,=Message
    swi    0x204  ; display message
    ...
Message: .asciz "Hello There\n"
```

Display an integer on the LCD screen: swi 0x205

Display an integer on the LCD screen. The integer is in `r2`, to be shown at position (x,y), where here `r0`=x=4 and `r1`=y=1, and `r2`=23 (that is, line 1 at column 4).

```
    mov    r0,#4
    mov    r1,#1
    mov    r2,#23
    swi    0x205  ; display integer
```

Clear the display on the LCD screen: swi 0x206

Clear the whole LCD screen.

```
    swi    0x206  @ clear screen
```

Display a character on the LCD screen: swi 0x207

Display a character on the LCD screen. The character is in `r2`, to be shown at position (x,y), where here `r0`=x=4 and `r1`=y=1, and `r2`="`Z`" (that is, line 1 at column 4).

```
    mov    r0,#4
    mov    r1,#1
    mov    r2,#'Z
    swi    0x207  @display char
```

Clear one line in the display on the LCD screen: swi 0x208

Clear the one line on the LCD screen, the line number is given in `r0`.

```
    ldr    r0,#5
    swi    0x208  @clear line 5
```