A brief introduction (revision) of

interrupt/exception

What is interrupt/exception?

- Main ()
- {
- :
- Doing something
- (e.g.
- browsing)
- :
- } ring

Can happen anytime
Depends on types of interrupts

Phone rings

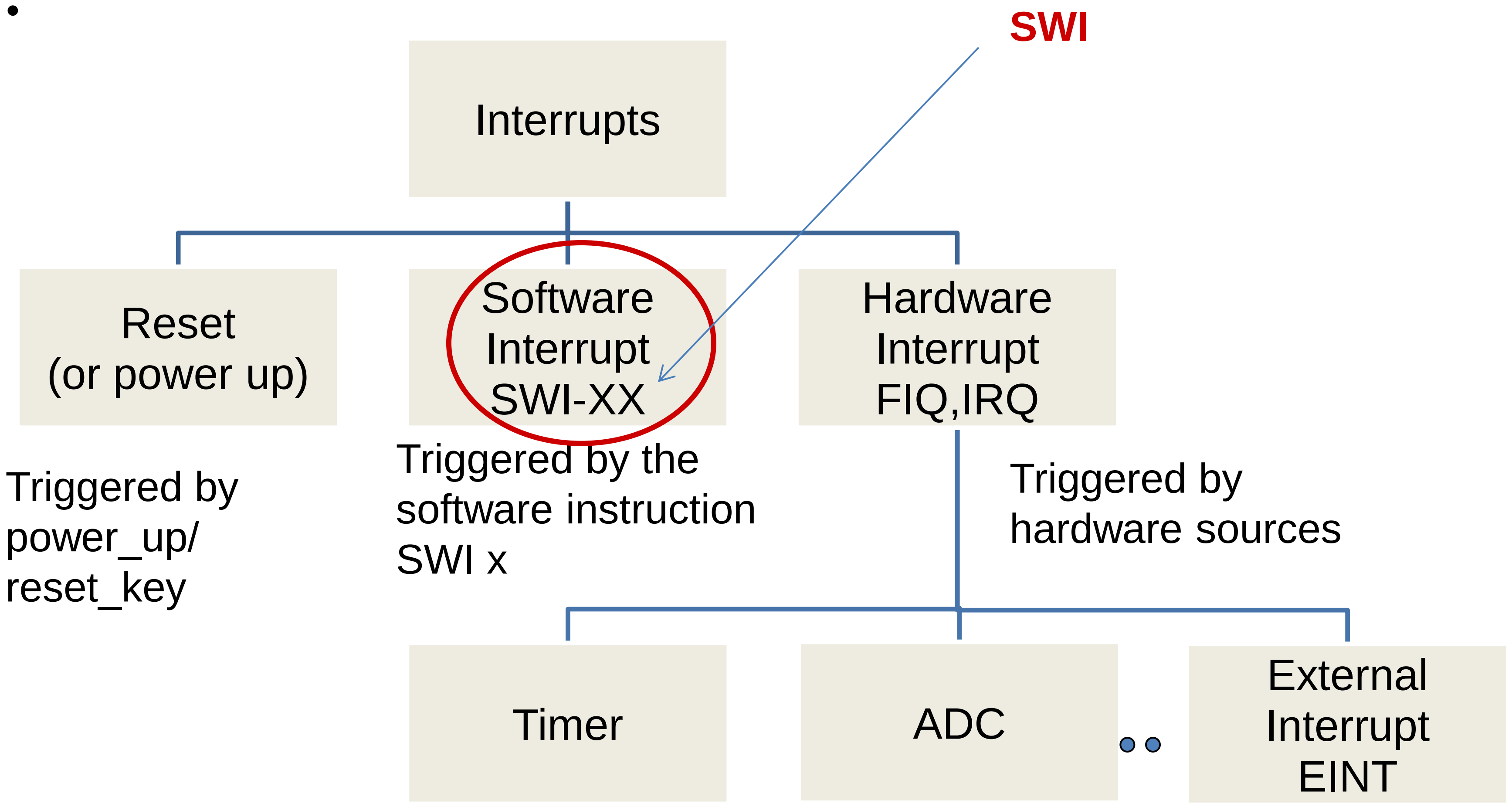
```
_isr() //Interrupt service routine
{

  some tasks (e.g. answer
                    telephone)

}//when finished,
//goes back to main
```

Examples

- When your computer is running, a key press will trigger an interrupt to input a character to your system
- The dispatcher in the operating system is implemented by timer interrupt.
  - Timer interrupts the CPU at a rate of 1KHz
  - At each interrupt the system determines which task to run next.

Important interrupts

-

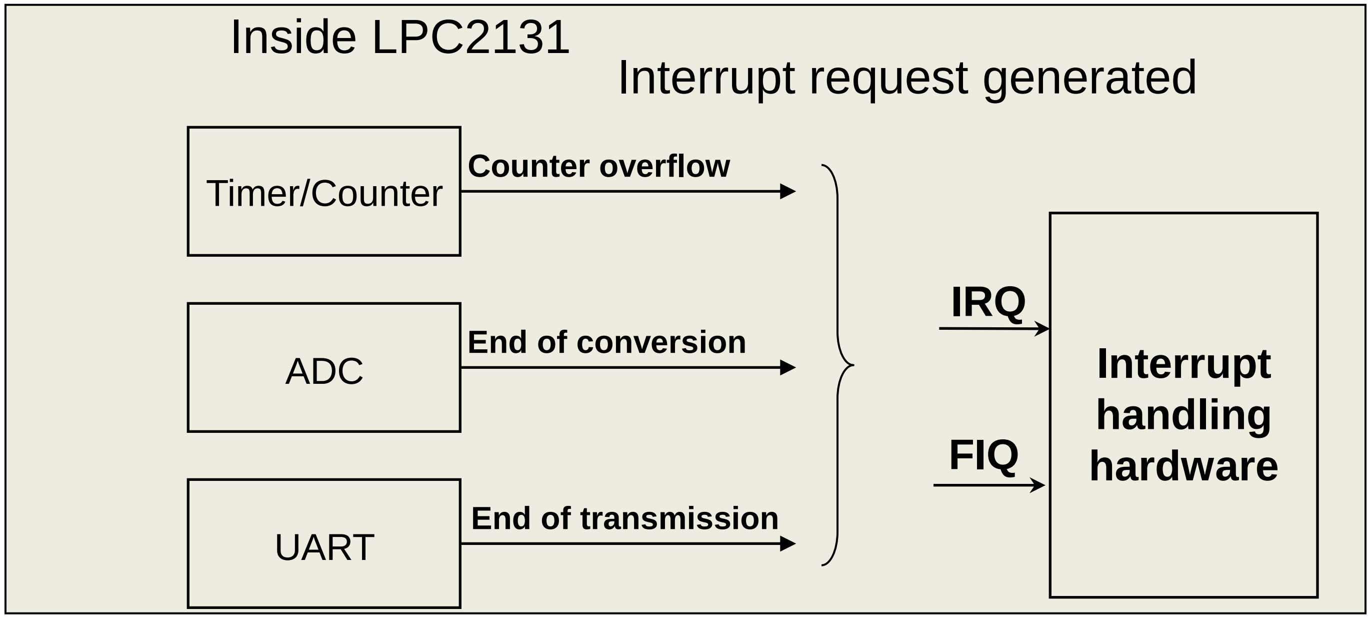


Interrupt and exception

- The terms are used differently by various manufacturers
- Traditionally exception means
  - The normal operation of a program is interrupted and the processor will execute another piece of software (exception handling) somewhere.
    - Interrupt (hardware interrupt) is an exception caused by some hardware condition happening outside the processor (e.g. external hard interrupt, IRQ FIQ).
    - Software interrupt (SWI) is an exception caused by an assembly software instruction (SWI 0x?? exception call instruction) written in the software code.
    - Trap is an exception caused by a failure condition of the processor (e.g. abort “pre-fetch , data” , undefined instruction, divided_by_zero, or stack overflow etc)

Important interrupts in words

- Reset, a special interrupt to start the system– happens at power up , or reset button depressed)
- Software interrupt SWI: similar to subroutine – happens when “SWI 0x??” is written in the program
- Hardware interrupt
  - FIQ (fast interrupt) or IRQ (external interrupt), when
    - the external interrupt request pin is pulled low, or
    - an analogue to digital conversion is completed, or
    - A timer/counter has made a regular request

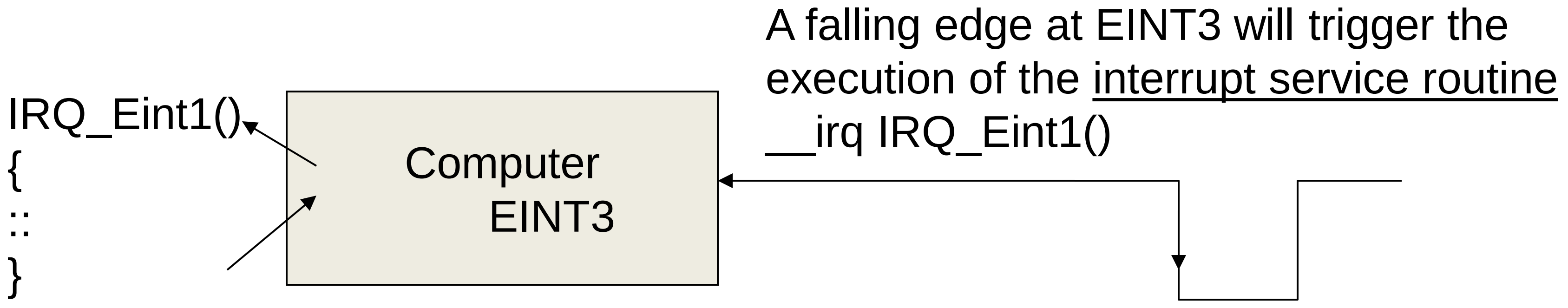


Introduction to

Software Interrupt (SWI)

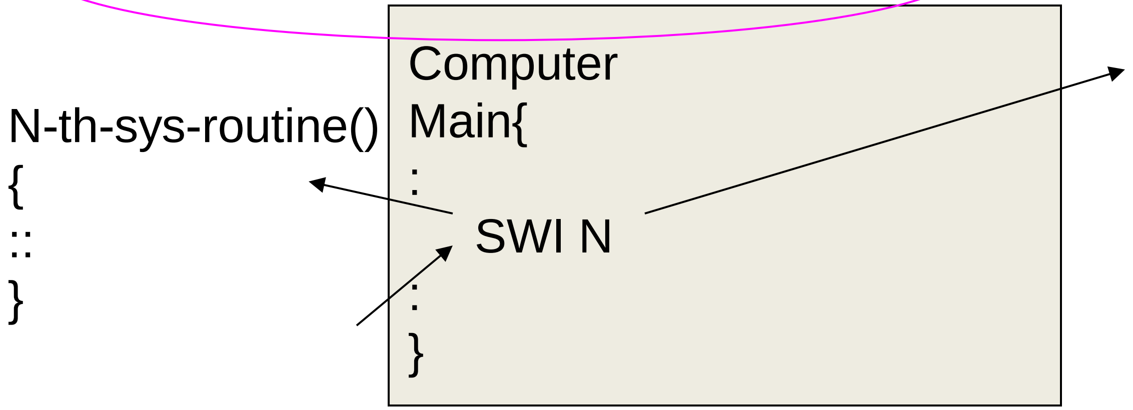
Compare hardware and software interrupt

– Hardware interrupt, e.g.



A falling edge at EINT3 will trigger the execution of the interrupt service routine __irq IRQ_Eint1()

– Software interrupt



An instruction "SWI N" in the program will trigger the execution of the N-th-sys-routine (system routine)

Exception (interrupt) Modes

- ARM supports 7 types of exceptions and has a privileged processor mode for each type of exception.
- ARM Exception (interrupt) vectors

	Address	Exception	Mode in Entry
1	0x00000000	Reset	Supervisor
2	0x00000004	Undefined instruction	Undefined
3	0x00000008	Software Interrupt	Supervisor
4	0x0000000C	Abort (prefetch)	Abort
5	0x00000010	Abort (data)	Abort
x	0x00000014	Reserved	Reserved
6	0x00000018	IRQ (external interrupt)	IRQ
7	0x0000001C	FIQ (fast interrupt)	FIQ

SWI

IRQ

Different types of exceptions

1) Reset (supervisor model, at power up , or reset button depressed)
2) Undefined Instruction (for co-processors *)
3) *Prefetch Abort for instruction fetch memory fault
4) *Data Abort : for data access memory fault
5) Software Interrupt (SWI) : supervisor mode, operating sys. calls
6) FIQ (Fast interrupt request)
7) IRQ (interrupt request)

- * not discussed here , refer to

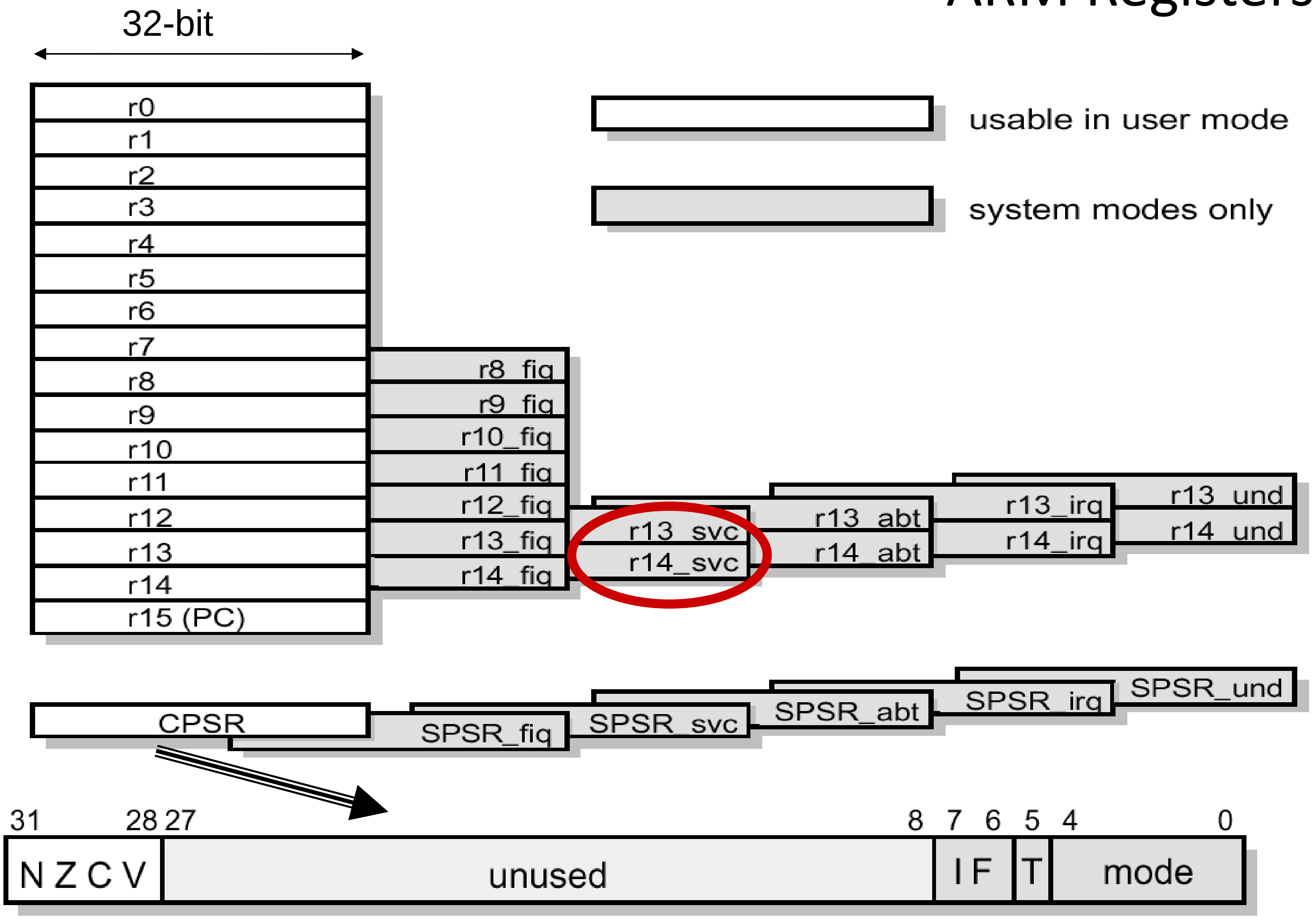
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0210c/DDI0210B.pdf

common usage of exceptions

- For building operating systems
  - Reset (supervisor model, at power up , or reset button depressed)
  - Undefined Instruction (for co-processors *)
  - *Prefetch Abort for instruction fetch memory fault
  - *Data Abort: for data access memory fault
  - Software Interrupt (SWI) : supervisor mode, operating sys. calls
- For embedded systems, hardware systems
  - FIQ (Fast interrupt request)
  - IRQ (interrupt request)
- * not discussed here , refer to

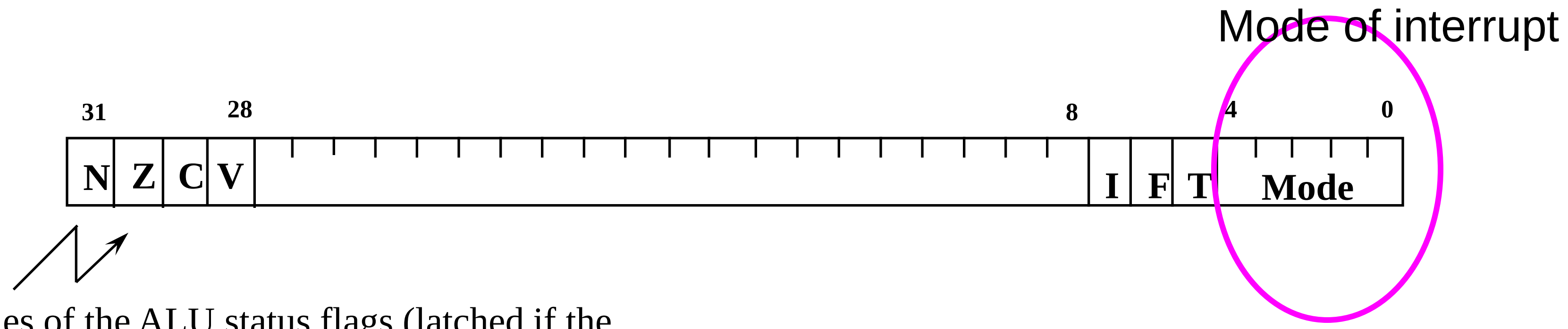
  http://infocenter.arm.com/help/topic/com.arm.doc.ddi0210c/DDI0210B.pdf

ARM Registers



Recall program status regs

-



Copies of the ALU status flags (latched if the instruction has the "S" bit set).

* **Condition Code Flags**

N = **N**egative result from ALU flag.
Z = **Z**ero result from ALU flag.
C = ALU operation **C**arried out
V = ALU operation o**V**erflowed

* **Interrupt Disable bits.**

I = 1, disables the IRQ.
F = 1, disables the FIQ.

* **T Bit (Architecture v4T only)**

T = 0, Processor in ARM state
T = 1, Processor in Thumb state

Recall: registers

General registers and Program Counter

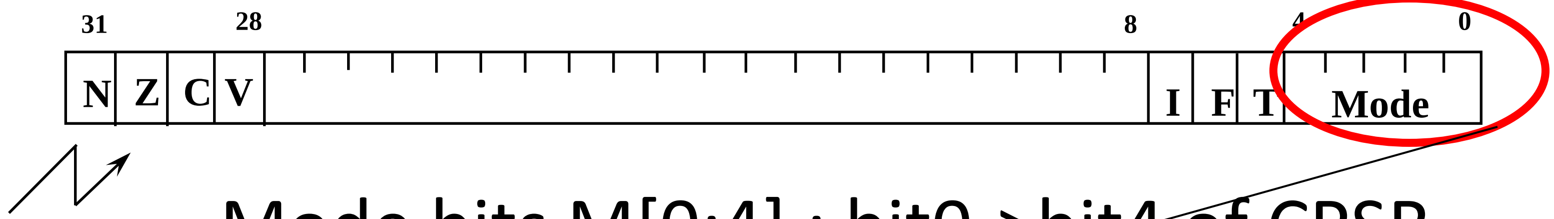
User32 / System	FIQ32	Supervisor32	Abort32	IRQ32	Undefined32
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12
r13 (sp)	r13_fiq	r13_svc	r13_abt	r13_irq	r13_undef
r14 (lr)	r14_fiq	r14_svc	r14_abt	r14_irq	r14_undef
r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)	r15 (pc)

Program Status Registers

cpsr	cpsr	cpsr	cpsr	cpsr	cpsr
	spsr_fiq	spsr_svc	spsr_abt	spsr_irq	spsr_undef

Shaded registers are extra Registers for different modes

SPSR= Saved Process Status Reg



Mode bits M[0:4] : bit0->bit4 of CPSR

- Table 2-2 PSR mode bit values

M[4:0]	Mode	Visible Thumb-state registers	Visible ARM-state registers
10000	User	r0–r7, SP, LR, PC, CPSR	r0–r14, PC, CPSR
10001	FIQ	r0–r7, SP_fiq, LR_fiq, PC, CPSR, SPSR_fiq	r0–r7, r8_fiq–r14_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	r0–r7, SP_irq, LR_irq, PC, CPSR, SPSR_irq	r0–r12, r13_irq, r14_irq, PC, CPSR, SPSR_irq
10011	Supervisor	r0–r7, SP_svc, LR_svc, PC, CPSR, SPSR_svc	r0–r12, r13_svc, r14_svc, PC, CPSR, SPSR_svc
10111	Abort	r0–r7, SP_abt, LR_abt, PC, CPSR, SPSR_abt	r0–r12, r13_abt, r14_abt, PC, CPSR, SPSR_abt
11011	Undefined	r0–r7, SP_und, LR_und, PC, CPSR, SPSR_und	r0–r12, r13_und, r14_und, PC, CPSR, SPSR_und
11111	System	r0–r7, SP, LR, PC, CPSR	r0–r14, PC, CPSR

- http://infocenter.arm.com/help/topic/com.arm.doc.ddi0210c/DDI0210B.pdf

We will study Software Interrupt

(SWI)

Why Software interrupt SWI?

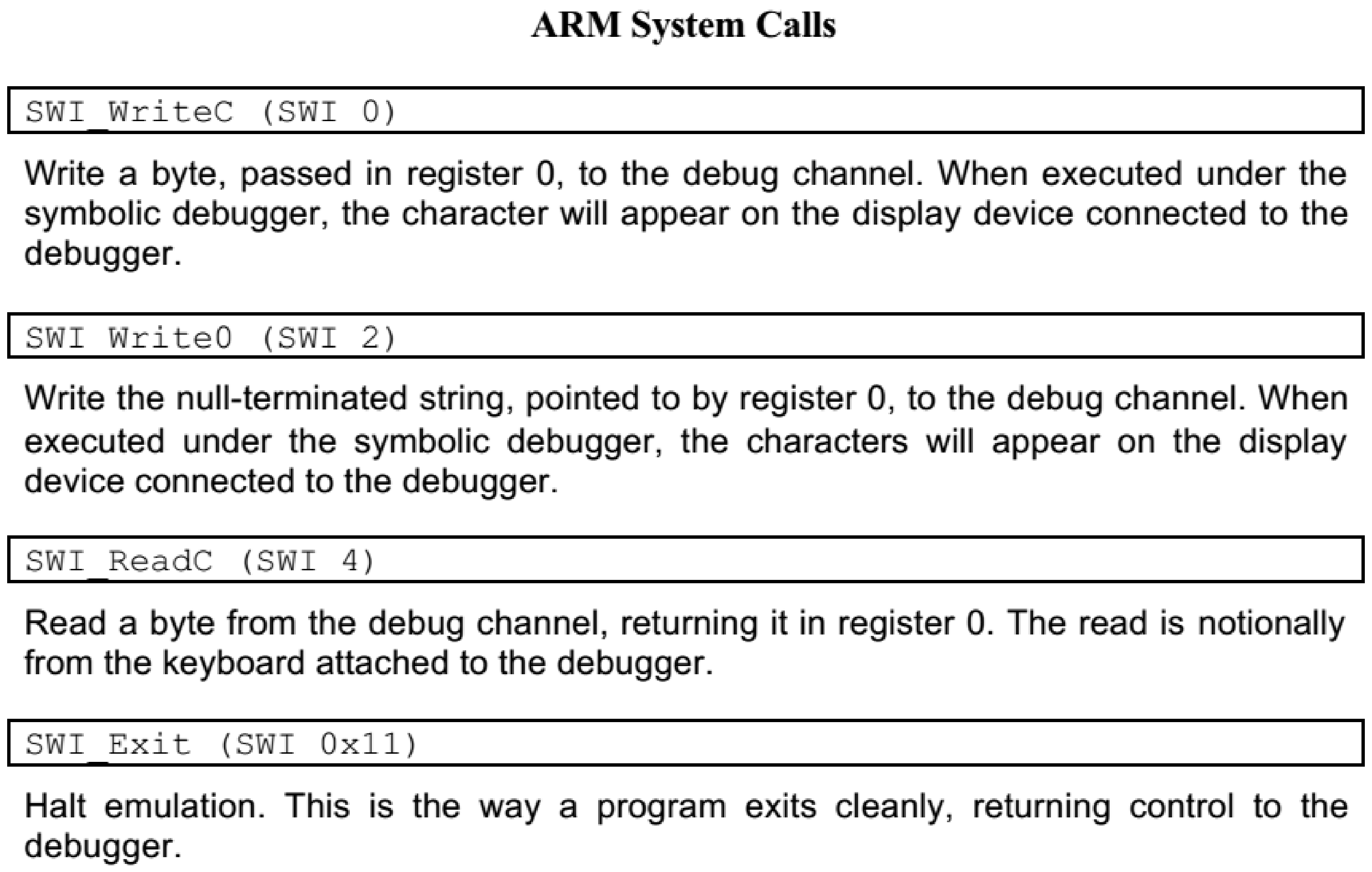
"Similar to a subroutine call but more efficient and organized"

- Make a list of often used routines
- To build system calls in Linux or Windows.
- E.g. print character , read keyboard etc..

SWI software interrupt

- For operating sys. (OS) developers to write often used routines
- E.g. SWI 0x12 is for “write a character to screen”
- So you may have a table of all routines and called by users or OS programs.
- <u>SWI table</u>
- 0x01= reset system
- 0x02= init timer
- :
- 0x12 = write a charter to screen
- 0x13= make a beep sound for 0.5 seconds…

SWI software interrupt



ARM System Calls

`SWI_WriteC (SWI 0)`

Write a byte, passed in register 0, to the debug channel. When executed under the symbolic debugger, the character will appear on the display device connected to the debugger.

`SWI_Write0 (SWI 2)`

Write the null-terminated string, pointed to by register 0, to the debug channel. When executed under the symbolic debugger, the characters will appear on the display device connected to the debugger.

`SWI_ReadC (SWI 4)`

Read a byte from the debug channel, returning it in register 0. The read is notionally from the keyboard attached to the debugger.

`SWI_Exit (SWI 0x11)`

Halt emulation. This is the way a program exits cleanly, returning control to the debugger.

SWI software interrupt

`SWI_Clock (SWI 0x61)`

Return, in r0, the number of centi-seconds since the support code began execution. In general, only the difference between successive calls to `SWI_Clock`, can be meaningful.

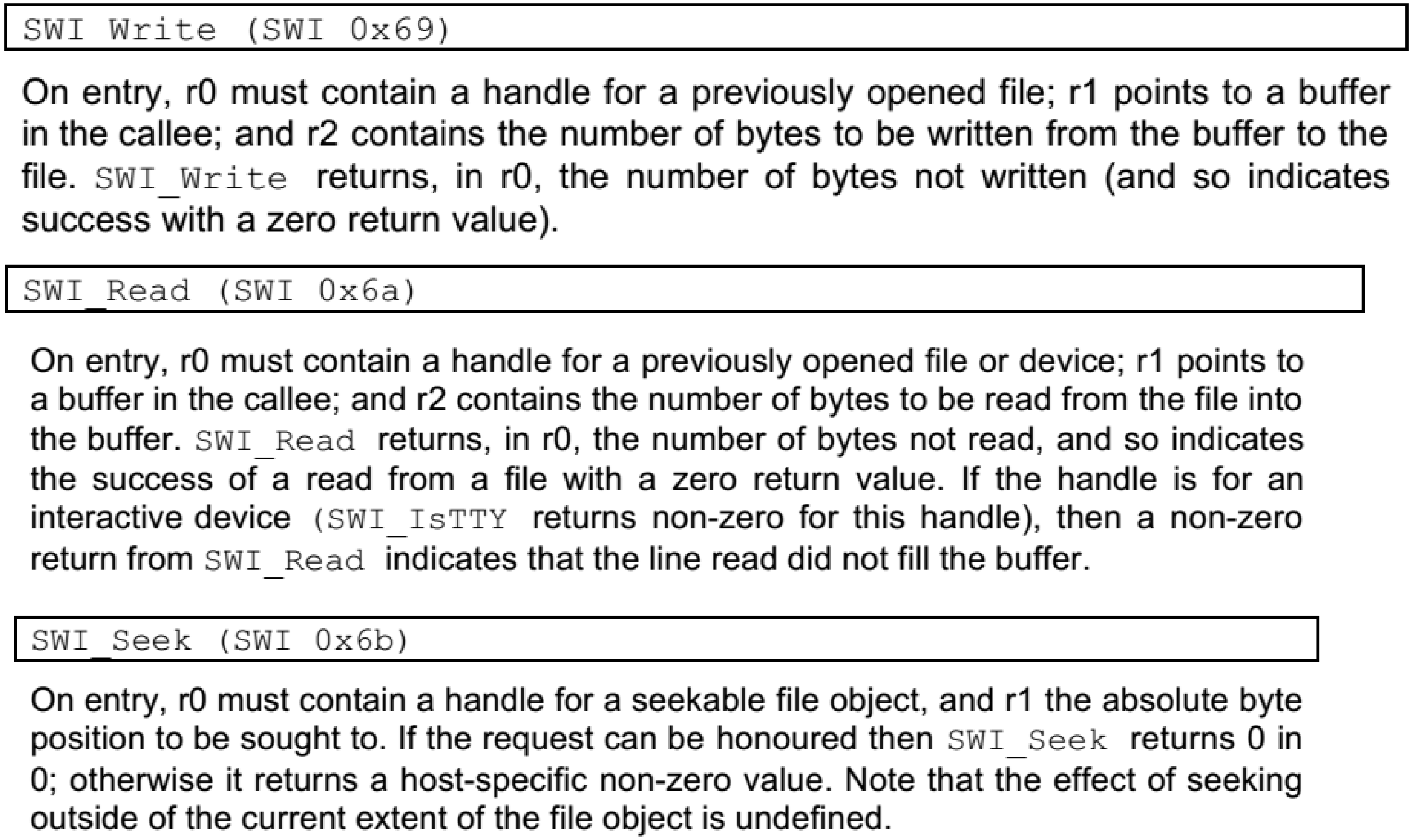
`SWI Open (SWI 0x66)`

r0 addresses a NUL-terminated string containing a file or device name; r1 is a small integer specifying the file-opening mode: `0` - read mode, `4` - write mode, `8` - apend mode. If the open succeeds, a non-zero handle is returned in r0, which can be quoted to `SWI_Close`, `SWI_Read`, `SWI_Write`, `SWI_Seek`, `SWI_Flen` and `SWI_IsTTY`. Nothing else may be asserted about the value of the handle. If the open fails, the value 0 is returned in r0.

`SWI_Close (SWI 0x68)`

On entry, r0 must be a handle for an open file, previously returned by `SWI_Open`. If the close succeeds, zero is returned in r0; otherwise, a non-zero value is returned.

SWI software interrupt



`SWI Write (SWI 0x69)`

On entry, r0 must contain a handle for a previously opened file; r1 points to a buffer in the callee; and r2 contains the number of bytes to be written from the buffer to the file. `SWI_Write` returns, in r0, the number of bytes not written (and so indicates success with a zero return value).

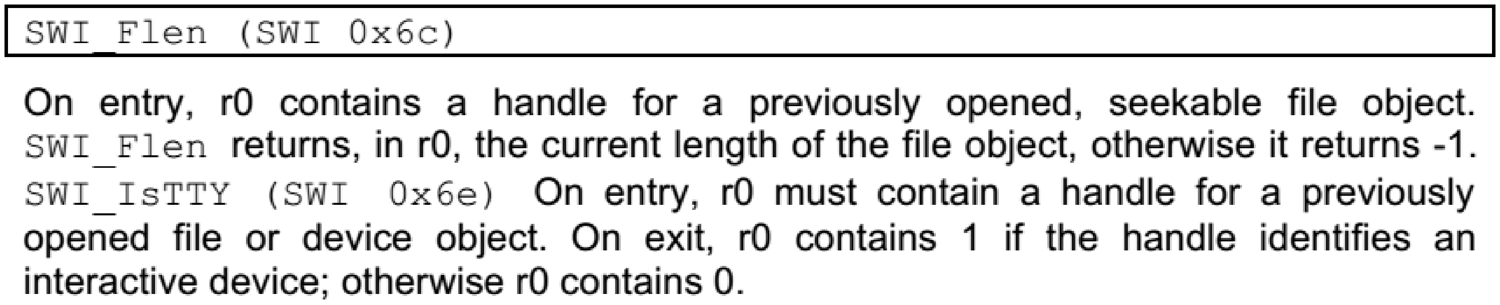
`SWI_Read (SWI 0x6a)`

On entry, r0 must contain a handle for a previously opened file or device; r1 points to a buffer in the callee; and r2 contains the number of bytes to be read from the file into the buffer. `SWI_Read` returns, in r0, the number of bytes not read, and so indicates the success of a read from a file with a zero return value. If the handle is for an interactive device (`SWI_IsTTY` returns non-zero for this handle), then a non-zero return from `SWI_Read` indicates that the line read did not fill the buffer.

`SWI_Seek (SWI 0x6b)`

On entry, r0 must contain a handle for a seekable file object, and r1 the absolute byte position to be sought to. If the request can be honoured then `SWI_Seek` returns 0 in 0; otherwise it returns a host-specific non-zero value. Note that the effect of seeking outside of the current extent of the file object is undefined.

SWI software interrupt



`SWI_Flen (SWI 0x6c)`

On entry, r0 contains a handle for a previously opened, seekable file object. `SWI_Flen` returns, in r0, the current length of the file object, otherwise it returns -1.
`SWI_IsTTY (SWI 0x6e)` On entry, r0 must contain a handle for a previously opened file or device object. On exit, r0 contains 1 if the handle identifies an interactive device; otherwise r0 contains 0.

SWI software interrupt

```
;
; An example to write a short file on disk
;
          AREA  Example, CODE, READONLY          ; name this block of code
SWI_Exit        EQU    0x11              ; tidy finish
SWI_Clock       EQU    0x61
SWI_Open        EQU    0x66
SWI_Close       EQU    0x68
SWI_Write       EQU    0x69
write_only      EQU    4                 ; mode 4 = open to write
          ENTRY                                    ; mark first instruction
                                         ; to execute
start     ADR   r0, filename             ; r0 points to string
          MOV   r1, #write_only
          SWI   SWI_Open                 ; open a file for writing
          MOV   r5, r0                   ; save file-handler in r5
          ADR   r1, String               ; point to a string
          MOV   r2, #14                  ;   ..... 14 characters long
          SWI   SWI_Write                ; write to file
          MOV   r0, r5
          SWI   SWI_Close                ; close the file
          SWI   SWI_Exit

filename = "test.txt",0
String = "Hello World!",&0a,&0d

          END
```

```
; 16bit data transfer

        TTL     move16 – 16-bit data transfer
        AREA    Program, CODE, READONLY
        ENTRY

Main
        LDRB    R1, Value       ; Load value
        STR     R1, Result      ; Sore it again
        SWI     &11             ; exit()

Value   DCW     &C123           ; Source value to be moved
        ALIGN                   ; Alling next word
Result  DCW     0               ; Reserve space for result

        END
```

```
; Find the length of a null terminated string

Main
        LDR     R0, =Data1        ; Load the address of the lookup table
        MOV     R1, #-1           ; Start count at -1
Loop
        ADD     R1, R1, #1        ; Increment count
        LDRB    R2, [R0], #1      ; Load the first byte into R2
        CMP     R2, #0            ; Is it the terminator ?
        BNE     Loop              ; No => Next char

        STR     R1, CharCount ; Store result
        SWI     &11

        AREA    Data1, DATA
        DCB     "Hello, World", 0
```

```
Blank EQU     " "

Main
        LDR     R0, =Data1    ; load the address of the lookup table
        MOV     R1, #Blank    ; store the blank char in R1
Loop
        LDRB    R2, [R0], #1  ; load the first byte into R2
        CMP     R2, R1        ; is it a blank
        BEQ     Loop          ; if so loop

        SUB     R0, R0, #1    ; otherwise done - adjust pointer
        STR     R0, Pointer   ; and store it
        SWI     &11

        AREA    Data1, DATA
        DCB     "      7   "
```

```
Blank EQU     ' '
Zero  EQU     '0'
Main
      LDR     R0, =Data1     ; load the address of the lookup table
      MOV     R1, #Zero      ; store the zero char in R1
      MOV     R3, #Blank     ; and the blank char in R3
Loop
      LDRB    R2, [R0], #1   ; load the first byte into R2
      CMP     R2, R1         ; is it a zero
      BNE     Done           ; if not, done
      SUB     R0, R0, #1     ; otherwise adjust the pointer
      STRB    R3, [R0]       ; and store it blank char there
      ADD     R0, R0, #1     ; otherwise adjust the pointer
      BAL     Loop           ; and loop

      DCB     "000007000"
```

```
Main
        LDR     R0, =Data1     ;load the address of the lookup table
        EOR     R1, R1, R1     ;clear R1 to store sum
        LDR     R2, Length     ;init element count
        CMP     R2, #0         ;zero length table ?
        BEQ     Done           ;yes => skip over sum loop
Loop
        LDR     R3, [R0]       ;get the data that R0 points to
        ADD     R1, R1, R3     ;add it to R1
        ADD     R0, R0, #+4    ;increment pointer
        SUBS    R2, R2, #0x1   ;decrement count with zero set
        BNE     Loop           ;if zero flag is not set, loop
Done
        STR     R1, Result     ;otherwise done - store result
        SWI     &11
```

```
; normalize a binary number

Main
        LDR     R0, =Data1          ;load the address of the lookup table
        EOR     R1, R1, R1          ;clear R1 to store shift count
        LDR     R3, [R0]            ;get the value to normalize
        CMP     R3, R1              ;is it a non-zero value
        BEQ     Done                ;yes => already normalised
Loop
        ADD     R1, R1, #1          ;increment shift counter
        MOVS    R3, R3, LSL #0x1 ;shift value by one bit
        BPL     Loop                ;loop until upper bit (sign bit) set
Done
        STR     R1, Shifted         ;otherwise done - store result
        STR     R3, Normal
        SWI     &11                 ;exit
```

```
  ; Disassemble a byte into its high and low order nibbles
Main
        LDR     R1, Value           ; Load value to be disassembled
        LDR     R2, Mask            ; Load the bitmask
        MOV     R3, R1, LSR #0x4    ; Copy high order nibble into R3
        MOV     R3, R3, LSL #0x8    ; Now left shift it one byte
        AND     R1, R1, R2          ; AND number with bitmask
        ADD     R1, R1, R3          ; Add the result of that to
                                    ; What we moved into R3
        STR     R1, Result          ; Store the result
        SWI     &11

Value   DCB     &5F                 ; Value to be shifted
        ALIGN
Mask    DCW     &000F               ; Bitmask = %...0001111
        ALIGN
Result  DCD     0                   ; Space to store result
```

```
; Find the larger of two numbers

Main
        LDR   R1, Value1  ; Load the first value to be compared
        LDR   R2, Value2  ; Load the second value to be compared
        CMP   R1, R2      ; Compare them
        BHI   Done        ; If R1 contains the highest
        MOV   R1, R2      ; otherwise overwrite R1
Done
        STR   R1, Result  ; Store the result
        SWI   &11

Value1 DCD    &12345678 ; Value to be compared
Value2 DCD    &87654321 ; Value to be compared
Result DCD    0         ; Space to store result
```

Program: add64.s

```
Main
        LDR     R0, =Value1 ; Pointer to first value
        LDR     R1, [R0]    ; Load first part of value1
        LDR     R2, [R0, #4] ; Load lower part of value1
        LDR     R0, =Value2 ; Pointer to second value
        LDR     R3, [R0]    ; Load upper part of value2
        LDR     R4, [R0, #4] ; Load lower part of value2
        ADDS    R6, R2, R4  ; Add lower 4 bytes and set carry flag
        ADC     R5, R1, R3  ; Add upper 4 bytes including carry
        LDR     R0, =Result ; Pointer to Result
        STR     R5, [R0]    ; Store upper part of result
        STR     R6, [R0, #4] ; Store lower part of result
        SWI     &11

Value1 DCD      &12A2E640, &F2100123 ; Value to be added
Value2 DCD      &001019BF, &40023F51 ; Value to be added
Result DCD      0                   ; Space to store result
```

Program: factorial.s

```
        AREA    Program, CODE, READONLY
Main
        LDR     R0, =DataTable         ; Load address of lookup table
        LDR     R1, Value              ; Offset of value to be looked up
        MOV     R1, R1, LSL #0x2       ; Read value from table
        ADD     R0, R0, R1             ; index into table at R0
        LDR     R2, [R0]               ; Read value from table entry R1
        LDR     R3, =Result            ; Load address of result
        STR     R2, [R3]               ; Store the answer
        SWI     &11

        AREA    DataTable, DATA
        DCD     1 ;0! = 1              ; Table containing factorials
        DCD     1 ;1! = 1
        DCD     2 ;2! = 2
Value   DCB     5
Result  DCW     0
```

Program: move16.s

```
; 16bit data transfer

          TTL       move16 – 16-bit data transfer
          AREA      Program, CODE, READONLY
          ENTRY

Main
          LDRB      R1, Value      ; Load value
          STR       R1, Result     ; Sore it again
          SWI       &11            ; exit()

Value     DCW       &C123          ; Source value to be moved
          ALIGN                    ; Alling next word
Result    DCW       0              ; Reserve space for result

          END
```

Program: invert.s

```
; Find the one's compliment (inverse) of a number

          TTL       invert.s – one's complement
          AREA      Program, CODE, READONLY
          ENTRY

Main
          LDR       R1, Value       ; Load number to be processed
          MVN       R1, R1          ; Invert (not) the value
          STR       R1, Result      ; Store the result
          SWI       &11             ; exit()

Value     DCD       &C123           ; Value to be complemented
Result    DCD       0               ; Reserve space for result

          END
```

Program: add2.s

```
; Add two numbers and store the result
. . .
Main
            LDR    R0, =Value1      ; R0 = &Value1
            LDR    R1, [R0]         ; R1 = *R0
            ADD    R0, R0, #0x4     ; R0++
            LDR    R2, [R0]         ; R2 = *R0
            ADD    R1, R1, R2       ; R1 = R1 + R2
            LDR    R0, =Result      ; R0 = &Result
            STR    R1, [R0]         ; *R0 = R1
            SWI    &11              ; exit(0)

Value1      DCD    &37E3C123
Value2      DCD    &367402AA
Result      DCD    0
. . .
```

Program: shiftleft.s

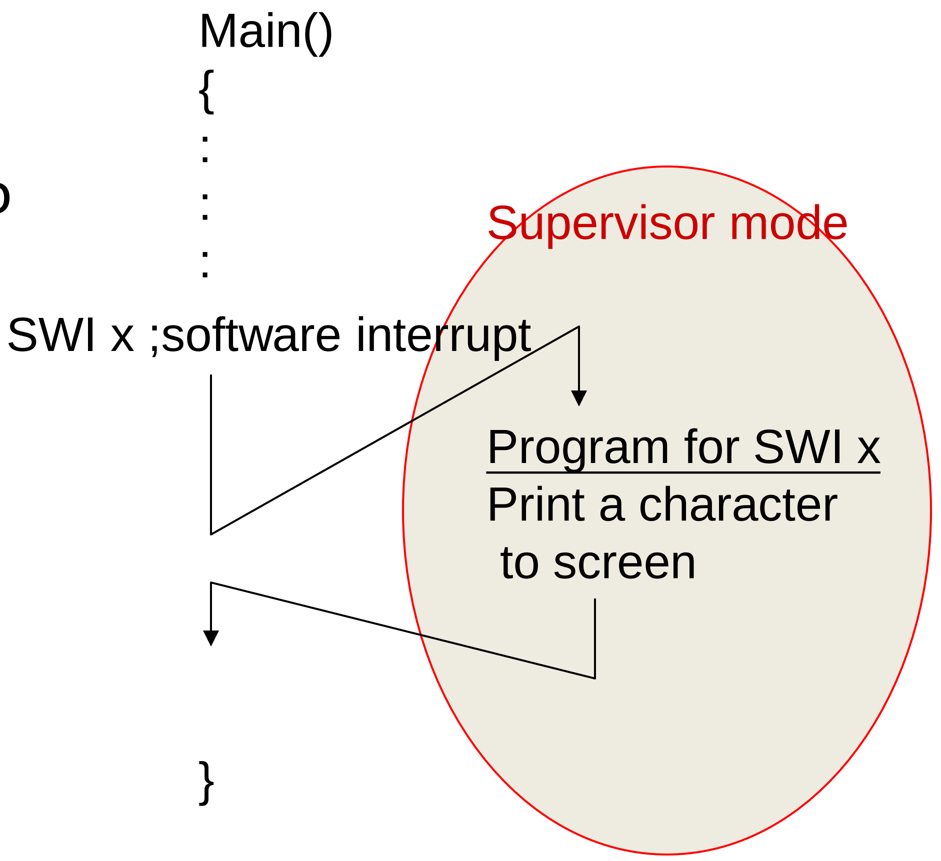
```
; Shift Left one bit

        TTL     shiftleft.s
        AREA    Program, CODE, READONLY
        ENTRY

Main
        LDR     R1, Value               ; Load the value to be shifted
        MOV     R1, R1, LSL #0x1        ; Shift Left one bit
        STR     R1, Result              ; Store the result
        SWI     &11                     ; exit

Value   DCD     &4242                   ; Value to be shifted
Result  DCD     0                       ; Space to store result

        END
```

Logical Shift Left by 1 bit

Example

E.g.1 For Building OS Operating system calls

- SWI software interrupt for writing OS calls
- An efficient way for user to make OS calls
- Examples, SWI table
  - SWI 20 = Print text on screen
  - SWI 23 =Read real time clock
  - SWI 35 =Keyboard read
  - -……



Code Example

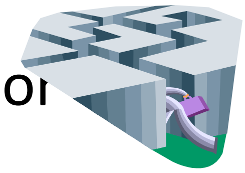
- When SWI is in your code:
- E.g. SWI vector=SWI 0x11, vector =0x11

```
        AREA Example, CODE, READONLY        ; name this block of code
        ENTRY                               ; mark first instruction
                                            ; to execute
start
        MOV     r0, #15                     ; Set up parameters
        MOV     r1, #20
        BL      firstfunc                   ; Call subroutine
        SWI     0x11                        ; terminate
firstfunc                                   ; Subroutine firstfunc
        ADD     r0, r0, r1                  ; r0 = r0 + r1
        MOV     pc, lr                      ; Return from subroutine
                                            ; with result in r0
        END                                 ; mark end of file
```

SWI interrupt procedures (enter the supervisor mode)

- SWI (software interrupt)
  - Caused by “SWI 0x??” in your program
  - Arm completes the current instruction.
  - Goto SWI exception address 0x08 (short form for 0x000 0008)
  - Exception entry, execution procedure (see next slide)
  - {Change to supervisor op. mode :CPSR (bit0-4)
  - :
  - :
  - Return from interrupt
    - `MOVS   pc, lr`
  - return to main}

Details of entering an interrupt (exception)

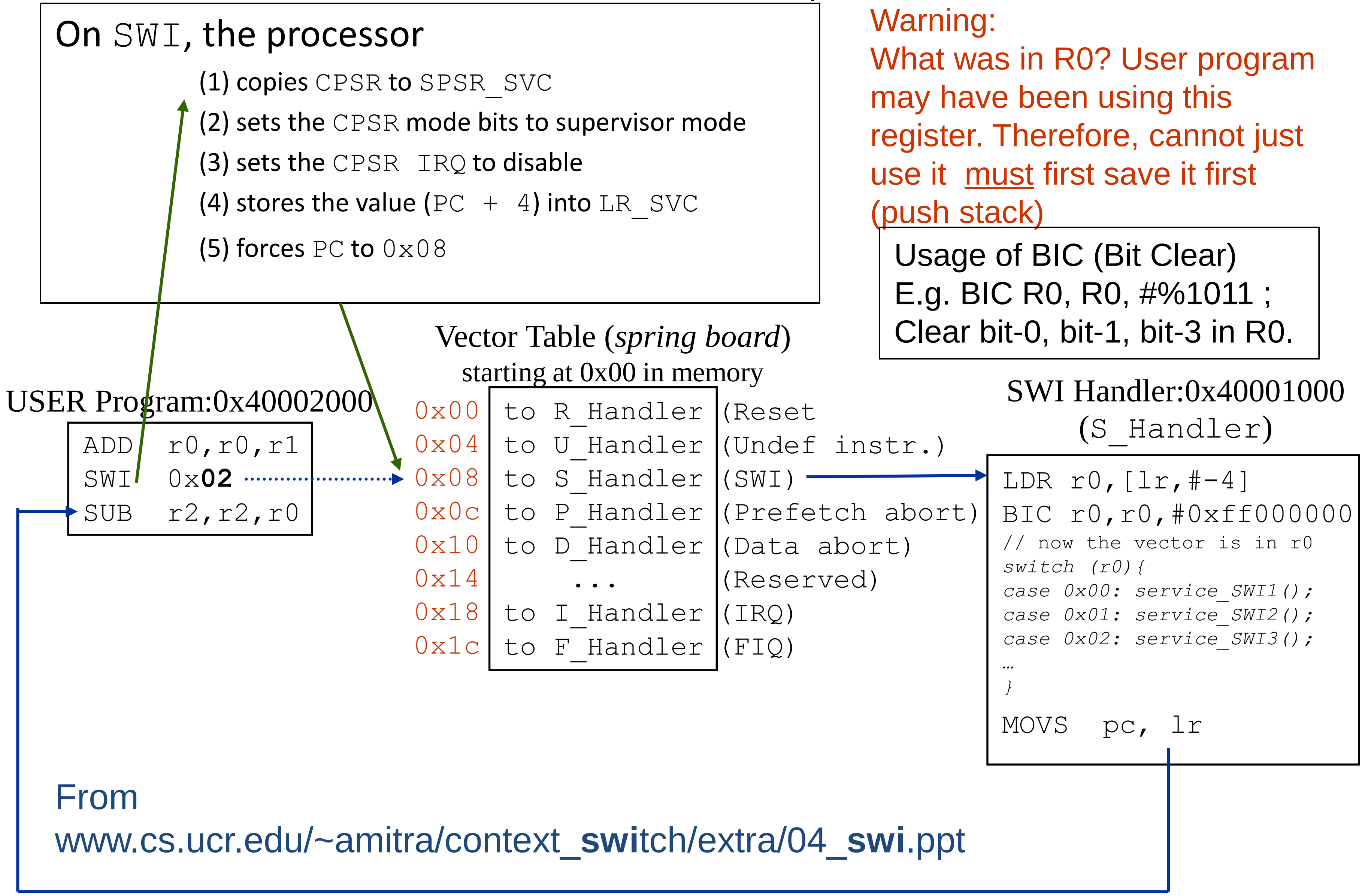


- Preserves the address of the next instruction in the appropriate Link Register (e.g. r14_svc → r14 of supervisor mode)
- Copies the CPSR (Current Program Status Register) into the appropriate SPSR (Saved Process Status Reg. e.g. SPSR_svc)
- Forces the CPSR mode bits to a value which depends on the exception (supervisor, interrupt etc)
- Forces the PC (program counter r15) to fetch the next instruction from the relevant exception vector
- It may also set the interrupt disable flags to prevent otherwise unmanageable nesting of exceptions.

http://infocenter.arm.com/help/topic/com.arm.doc.ddi0210c/DDI0210B.pdf

For your reference: SWI Software interrupt execution flow

SWI Software interrupt



On `SWI`, the processor

(1) copies `CPSR` to `SPSR_SVC`

(2) sets the `CPSR` mode bits to supervisor mode

(3) sets the `CPSR IRQ` to disable

(4) stores the value (`PC + 4`) into `LR_SVC`

(5) forces `PC` to `0x08`

Warning:
What was in R0? User program may have been using this register. Therefore, cannot just use it <u>must</u> first save it first (push stack)

Usage of BIC (Bit Clear)
E.g. BIC R0, R0, #%1011 ;
Clear bit-0, bit-1, bit-3 in R0.

USER Program:0x40002000

```
ADD   r0,r0,r1
SWI   0x02
SUB   r2,r2,r0
```

Vector Table (*spring board*)
starting at 0x00 in memory

```
0x00  to R_Handler  (Reset
0x04  to U_Handler  (Undef instr.)
0x08  to S_Handler  (SWI)
0x0c  to P_Handler  (Prefetch abort)
0x10  to D_Handler  (Data abort)
0x14      ...       (Reserved)
0x18  to I_Handler  (IRQ)
0x1c  to F_Handler  (FIQ)
```

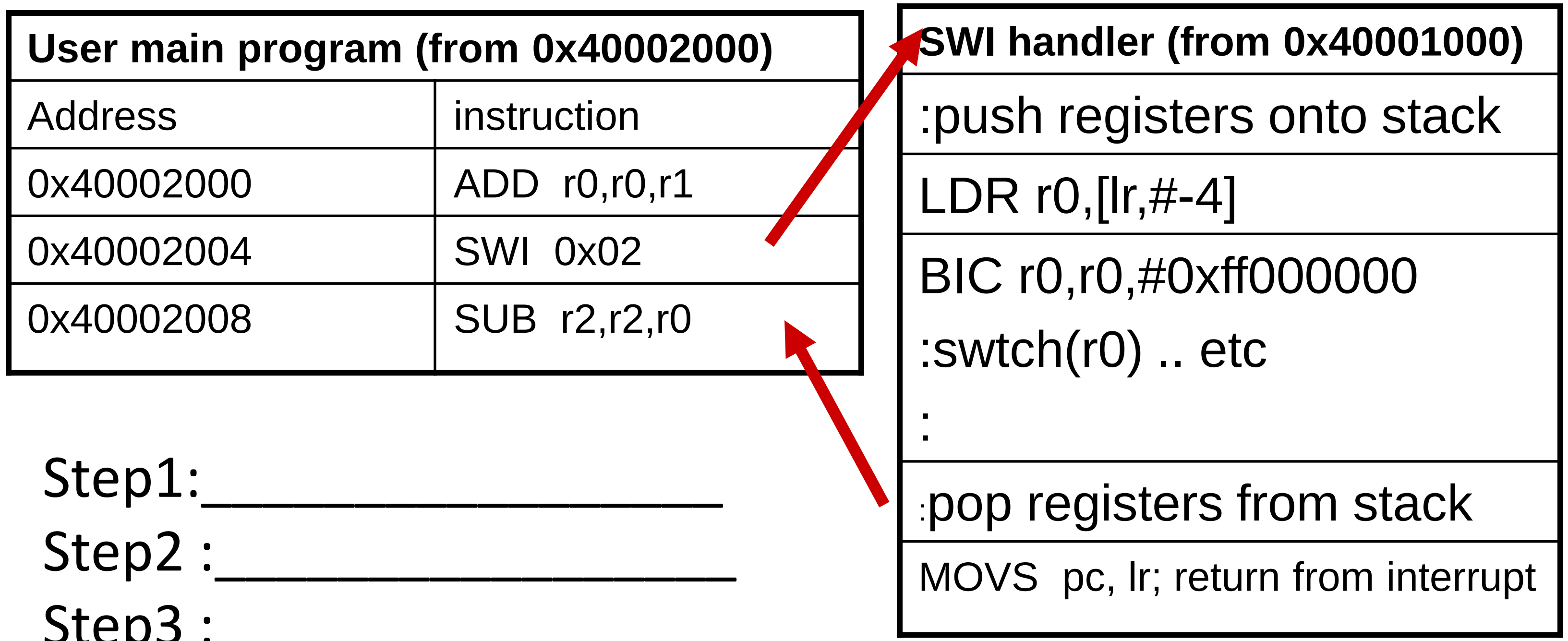
SWI Handler:0x40001000
(`S_Handler`)

```
LDR r0,[lr,#-4]
BIC r0,r0,#0xff000000
// now the vector is in r0
switch (r0){
case 0x00: service_SWI1();
case 0x01: service_SWI2();
case 0x02: service_SWI3();
…
}
MOVS  pc, lr
```

From
www.cs.ucr.edu/~amitra/context_**sw**itch/extra/04_**swi**.ppt

Exercise 12.2: SWI handler : Assume the SWI handler is at 0x40001000

- i) What is the content of address 0x08? Why?
- Answer:?____
- ii) What will the Processor do when it runs SWI 0x02
- Fill in steps that the precessor will do when entering SWI



User main program (from 0x40002000)	
Address	instruction
0x40002000	ADD r0,r0,r1
0x40002004	SWI 0x02
0x40002008	SUB r2,r2,r0

SWI handler (from 0x40001000)
:push registers onto stack
LDR r0,[lr,#-4]
BIC r0,r0,#0xff000000 :swtch(r0) .. etc :
:pop registers from stack
MOVS pc, lr; return from interrupt

- Step1:__________________
- Step2 :__________________
- Step3 :__________________
- Step4 :__________________
- Step5:____________________

Exercise 12.3 SWI handler

- Inside the SWI handler
  - i) What are the mode bits M[0:4] : bit0->bit4 of CPSR?
    Answer: ?_______________
  - Ii) What is the running mode inside the SWI handler: supervisor or user32?
    - ANSWER: ?__________________________________
  - III) When the link register lr is used, which lr the processor is using :r14(lr) or r14_svc or r14_irq?
    - Answer?__________________________________
- The Machine code of SWI 0x02 is 0xea000002 . List the values of r0 after the first and second instruction of the SWI handler.
  1. LDR r0,[lr,#-4]   ; fill in the blank, r0 = ?_______
  2. BIC r0,r0,#0xff000000; clear most significant 2 bytes of r0; r0=?_____
  3. // now the vector is in r0
  4. *switch (r0){*
  5. *case 0x00: service_SWI1();*
  6. *case 0x01: service_SWI2();*
  7. *case 0x02: service_SWI3();*
  8. *…*
  9. *}*

Details of leaving an interrupt

ARM7TDMI tech. ref. (section 2.8 exception)[1]

Bye Bye!

- At the end of the SWI handler: Movs pc,lr
- Move the r14, minus an offset to the PC. The offset varies according to the type
- (auto) SPSR_svc →CPSR.
- (auto) Clear the interrupt disable flags that were set on entry

Summary

- Learned the basic concept of exceptions and interrupts
  - SWI (Software interrupt)

Appendix

- Alternative set bit method in “C”
- Y=0x1<<21;//left shift 21 bits, this sets bit21=1 and other bits= 0
- Before shift
  - Y=0x1=0000 0000 0000 0000 0000 0000 0000 0001 (Binary)
- After shift
  - Y= 0000 0000 0010 0000 0000 0000 0000 0000 (Binary)
  -
    - bit 31 bit 21 bit0

  - Exercise: set bit 9 of register R to be 1, other bits to be 0.
  - Answer=0x1<<9;
  - So R=0000 0000 0000 0000 0000 0010 0000 0000 (Binary)
  - =0x200

Bit9 =1