Practice Raspberry Pi2

PART II:

INTERRUPS

Configuration.inc

```
/* Configuration of all the I0 of the expansion board */
      .set   GPBASE,   0x3f200000
      .set   GPFSEL0,      0x00
      .set   GPFSEL1,      0x04
      .set   GPFSEL2,      0x08
.text
   ldr   r0, =GPBASE
                  ldr   r1, [r0,  #GPFSEL0]
                  ldr r4, =0b11001111111111111001000000111111 @ Mask for forcing 0
                  ldr r5, =0b00001000000000000001000000000000 @ Mask for forcing 1
                  and r1,r1,r4
                  orr    r1,r1,r5
      str   r1, [r0, #GPFSEL0]                    @GPIO4&9 as output, GPIO2&3 as input
@ Configure of GPSEL1 (address 0x3F200004) for  GPIO 10,11,17
      ldr   r1, [r0, #GPFSEL1]
                  ldr r4, =0b11111111001111111111111111001001  @ Mask for forcing 0
                  ldr r5, =0b00000000001000000000000000001001  @ Mask for forcing 1
                  and r1,r1,r4
                  orr    r1,r1,r5
      str   r1, [r0, #GPFSEL1]                    @GPIO10&11&17 as output
@ Configure of GPSEL2 (address 0x3F200008) for  GPIO 22,27
      ldr   r1, [r0, #GPFSEL2]
                  ldr r4, =0b11111111001111111111111001111111  @ Mask for forcing 0
                  ldr r5, =0b00000000001000000000000001000000  @ Mask for forcing 1
                  and r1,r1,r4
                  orr    r1,r1,r5
      str   r1, [r0, #GPFSEL2]                    @GPIO22&27 as output
```

inter.inc: using symbolic names for addresses

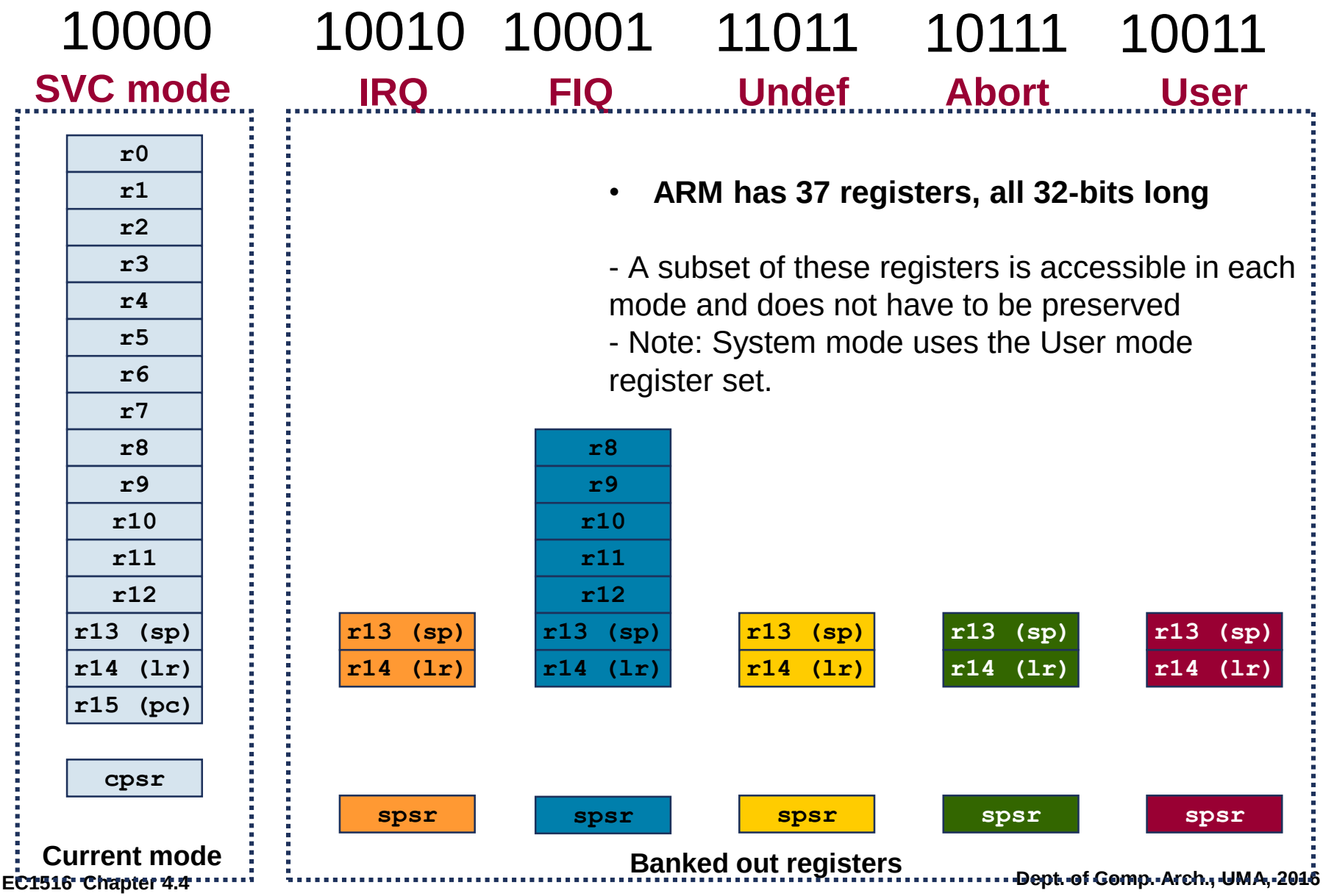
```
.macro    ADDEXC  vector, dirRTI
  ldr     r1, =(\dirRTI-\vector+0xa7ffffb)
  ror     r1, #2
  str     r1, [r0, #\vector]
.endm
  .set    GPBASE,   0x3F200000
  .set    GPFSEL0,      0x00
  .set    GPFSEL1,      0x04
  .set    GPFSEL2,      0x08
  .set    GPFSEL3,      0x0c
  .set    GPFSEL4,      0x10
  .set    GPFSEL5,      0x14
  .set    GPFSEL6,      0x18
  .set    GPSET0,       0x1c
  .set    GPSET1,       0x20
  .set    GPCLR0,       0x28
  .set    GPCLR1,       0x2c
  .set    GPLEV0,       0x34
  .set    GPLEV1,       0x38
  .set    GPEDS0,       0x40
  .set    GPEDS1,       0x44
  .set    GPFEN0,       0x58
  .set    GPFEN1,       0x5c
  .set    GPPUD,        0x94
  .set    GPPUDCLK0,    0x98
  .set    STBASE,   0x3F003000
  .set    STCS,         0x00
  .set    STCLO,        0x04
  .set    STC1,         0x10
  .set    STC3,         0x18
  .set    INTBASE,  0x3F00b000
  .set    INTFIQCON,    0x20c
  .set    INTENIRQ1,    0x210
  .set    INTENIRQ2,    0x214
```

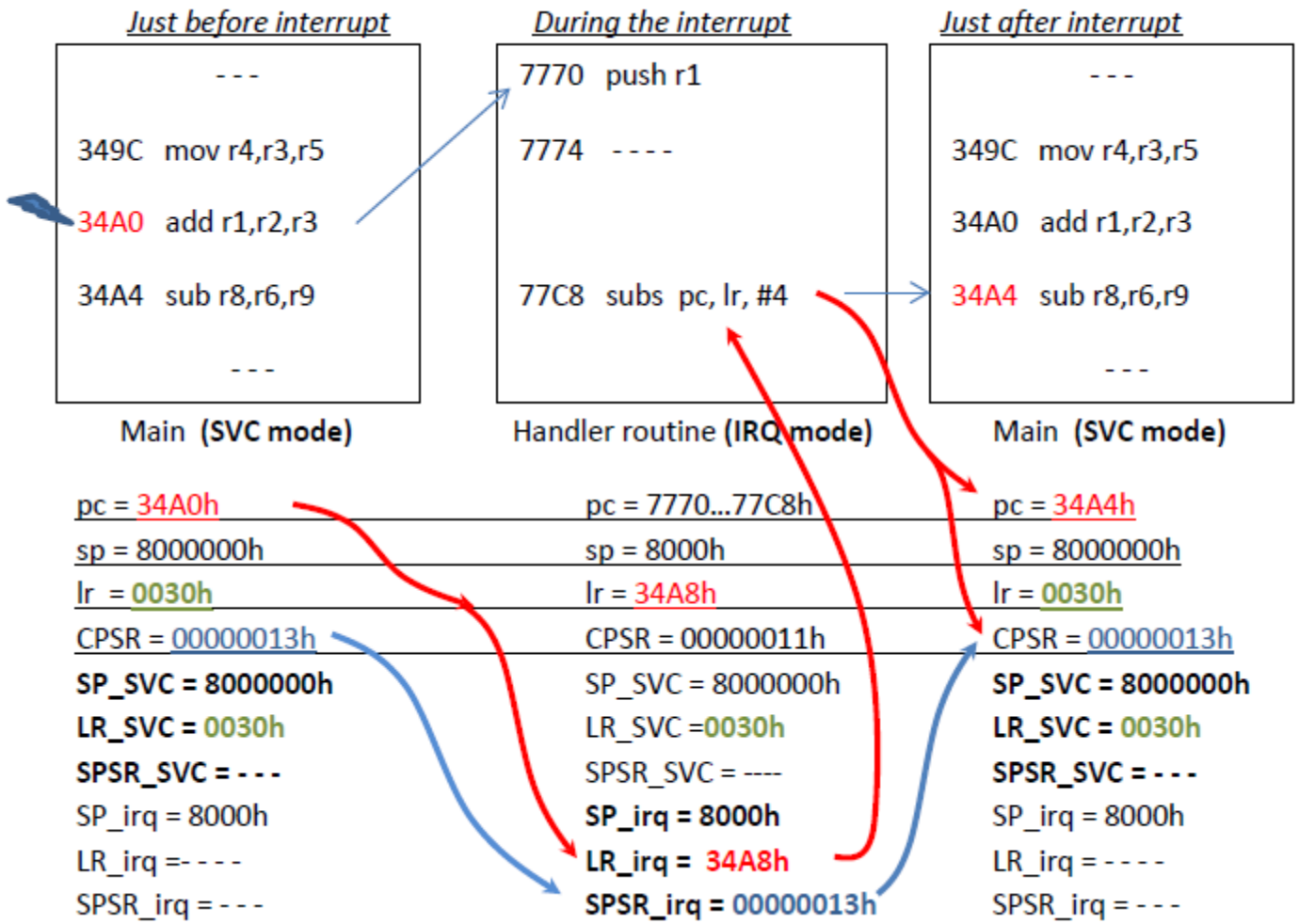
GPIO

Timer

Interrup.

Banking of registers




Just before interrupt
During the interrupt
Just after interrupt
- - -
349C mov r4,r3,r5
34A0 add r1,r2,r3
34A4 sub r8,r6,r9
- - -
7770 push r1
7774 - - - -
77C8 subs pc, lr, #4
- - -
349C mov r4,r3,r5
34A0 add r1,r2,r3
34A4 sub r8,r6,r9
- - -
Main (SVC mode)
Handler routine (IRQ mode)
Main (SVC mode)
pc = 34A0h
sp = 8000000h
lr = 0030h
CPSR = 00000013h
SP_SVC = 8000000h
LR_SVC = 0030h
SPSR_SVC = - - -
SP_irq = 8000h
LR_irq =- - - -
SPSR_irq = - - -
pc = 7770...77C8h
sp = 8000h
lr = 34A8h
CPSR = 00000011h
SP_SVC = 8000000h
LR_SVC =0030h
SPSR_SVC = ----
SP_irq = 8000h
LR_irq = 34A8h
SPSR_irq = 00000013h
pc = 34A4h
sp = 8000000h
lr = 0030h
CPSR = 00000013h
SP_SVC = 8000000h
LR_SVC = 0030h
SPSR_SVC = - - -
SP_irq = 8000h
LR_irq = - - - -
SPSR_irq = - - -

1- Initialize Vector Table

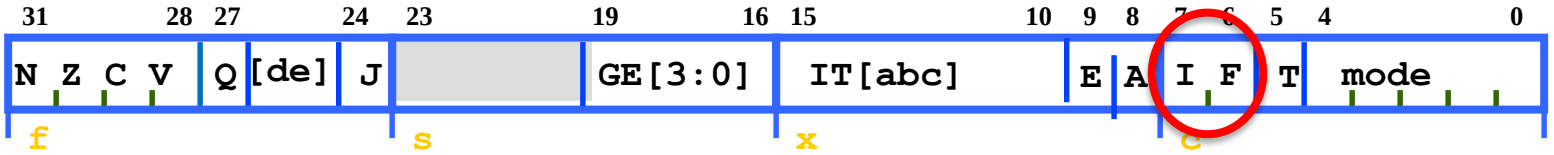
- To write in the Vector Table:
  - Example for IRQ

```
mov     r0, #0          @Vector table Base = 0
ADDEXC  0x18, irq_handler
```

- ADDEXC is a macro that computes the offset of the exception handler and writes the Vector in the Vector Table.

Mem. address	Contents
0x00000000	b reset_handler_routine
0x00000004	b Unexisingcode_hadeler_routine
0x00000008	b SVC_handler_routine
0x0000000C	b Abort1_handler_routine
0x00000010	b Abort2_handler_rountie
0x00000014	-
0x00000018	***b irq_handler***
0x0000001C	b FIQ_hadler_routine

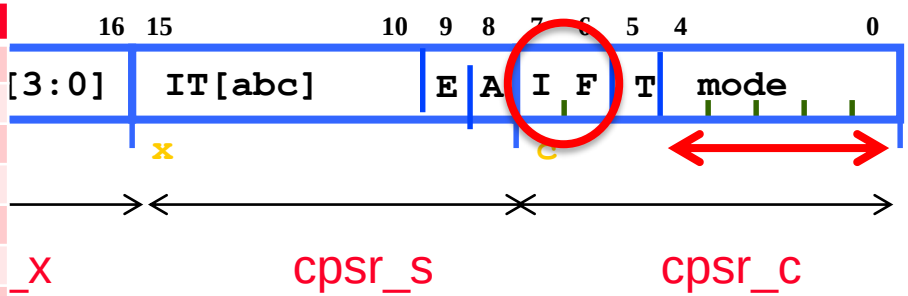
Status register again: cpsr_fsxc



- **Condition code flags**
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- **Sticky Overflow flag - Q flag**
  - Indicates if saturation has occurred
- **SIMD Condition code bits – GE[3:0]**
  - Used by some SIMD instructions
- **IF THEN status bits – IT[abcde]**
  - Controls conditional execution of Thumb instructions
- **T bit**
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
- **J bit**
  - J = 1: Processor in Jazelle state
- **Mode bits**
  - Specify the processor mode
- **Interrupt Disable bits**
  - I = 1: Disables IRQ
  - F = 1: Disables FIQ
- **E bit**
  - E = 0: Data load/store is little endian
  - E = 1: Data load/store is bigendian
- **A bit**
  - A = 1: Disable imprecise data aborts

Status register again: cpsr_fsxc

Code (bits 4-0 CPSR)	Mode
10000	User
10001	Fast interrupt (FIQ)
10010	Regular interrupt (IRQ)
10011	Supervisor (SVC)
10110	Sure Monitor
10111	Abort
11011	Undefined
11111	System



- Example 2: disable interrupts FIQ and enable IRQ for Supervisor (SVC) mode and enter in supervisor mode:
  - mov r0,#0b01010011
  - msr cpsr_c, r0

2- Disable Interruptions and initialize the stack

- Each mode has its stack pointer (sp)
  - Change the mode (via cpsr_c)
  - Instructions msr (sr <- reg) y mrs (reg <-sr).
  - Initialize the corresponding sp register
- Initial state in BareMetal is SVC
  - sp_fiq=0x4000, sp_irq=0x8000, sp_svc=0x8000000:

Entering in FIQ mode

Entering in IRQ mode

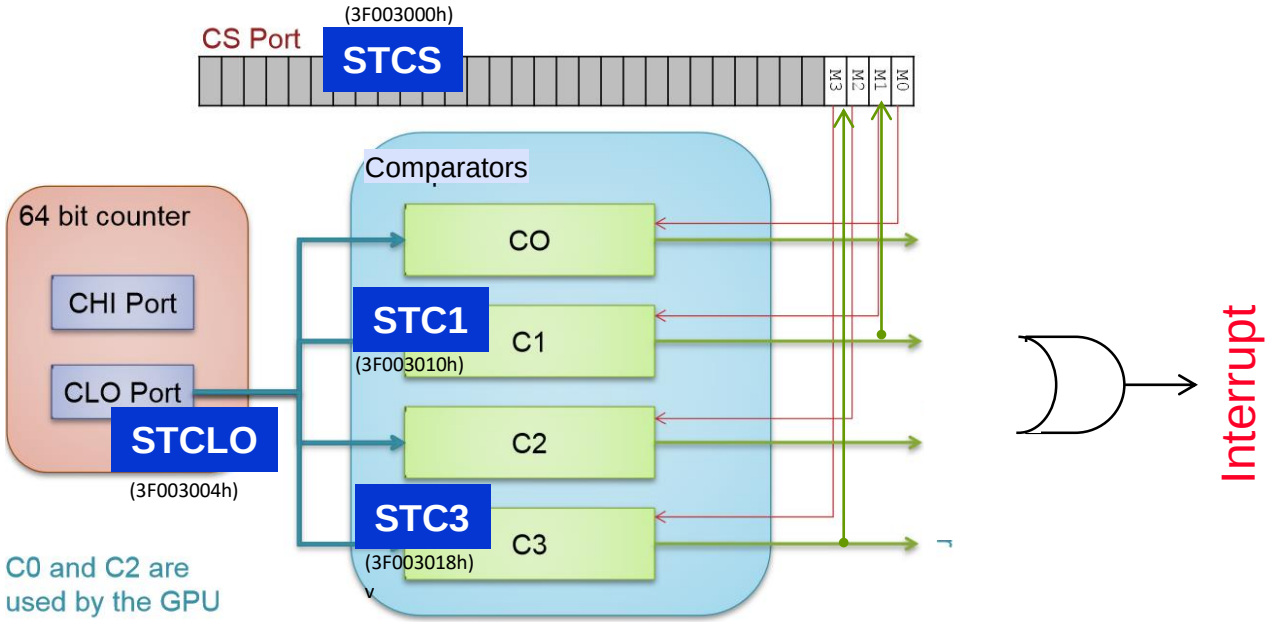
Entering in SVC mode

```
        mov     r0, #0          @apunto tabla excepciones
        ADDEXC  0x18, irq_handler
        ADDEXC  0x1c, fiq_handler
        mov     r0, #0b11010001   @modo FIQ, FIQ&IRQ desact
        msr     cpsr_c, r0
        mov     sp, #0x4000
        mov     r0, #0b11010010   @modo IRQ, FIQ&IRQ desact
        msr     cpsr_c, r0
        mov     sp, #0x8000
        mov     r0, #0b11010011   @modo SVC, FIQ&IRQ desact
        msr     cpsr_c, r0
        mov     sp, #0x8000000
```

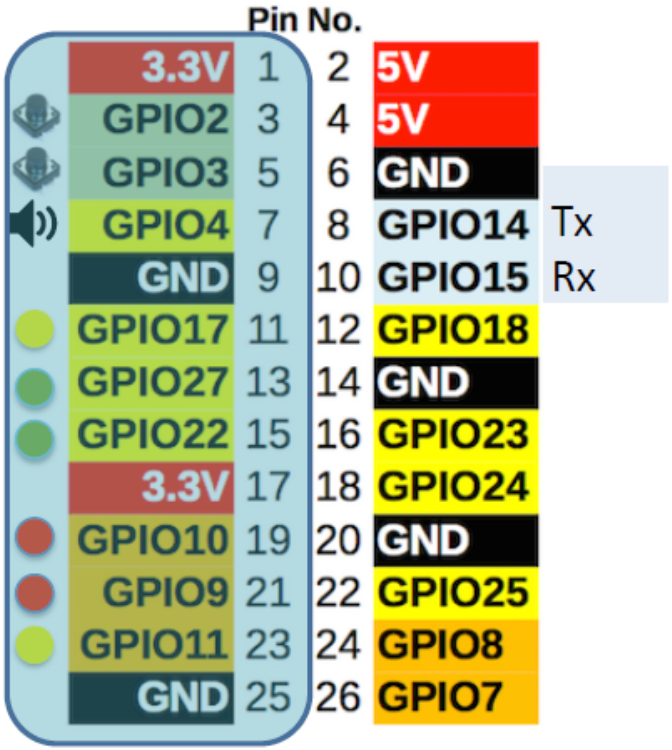
Skeleton for interrupts

```
.include "configuration.inc"
.include "symbolic.inc"

/* Vector Table inicialization */
     mov r0,#0
     ADDEXC 0x18, regular_interrupt @only if used
     ADDEXC 0x1C, fast_interrupt    @only if used


/* Stack init for IRQ mode */
     mov     r0, #0b11010010
     msr     cpsr_c, r0
     mov     sp, #0x8000
/* Stack init for FIQ mode */
     mov     r0, #0b11010001
     msr     cpsr_c, r0
     mov     sp, #0x4000
          mov r8,#0
/* Stack init for SVC mode */
     mov     r0, #0b11010011
     msr     cpsr_c, r0
     mov     sp, #0x8000000

/* Continue my MAIN program here */



end:   b end

/* Regular interrupt (only if used) */
regular_interrupt:
@    push { list of registers}


@    pop { list of registers}
     subs  pc, lr, #4

/* Fast interrupt (only if used) */
fast_interrupt:
@    push { list of registers}


@    pop { list of registers}
     subs  pc, lr, #4
```

System Timer



Rpi cheat sheet



Code structure:

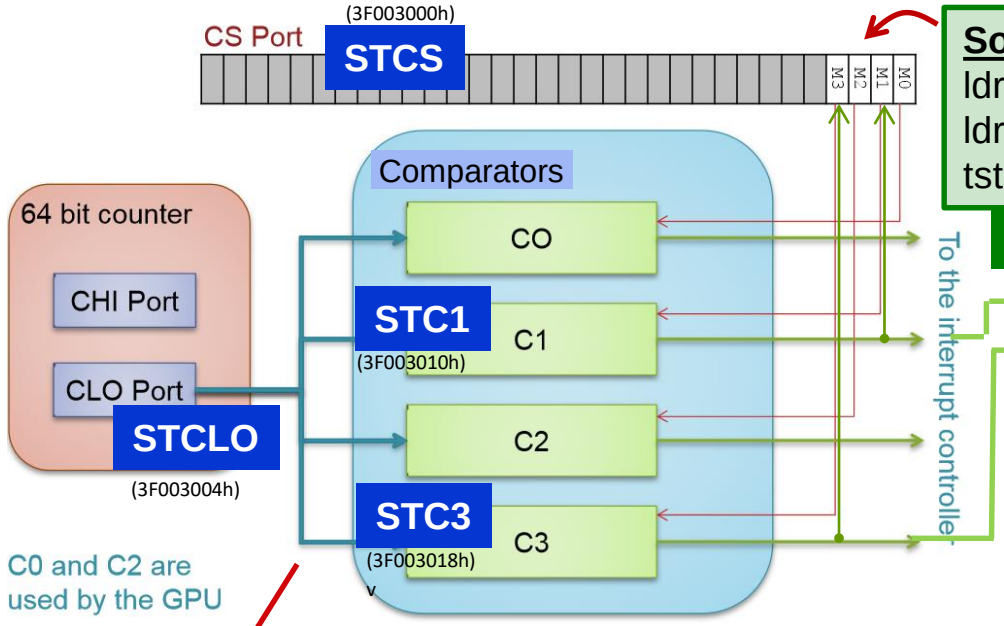
Main program (.text):

1. Initialize Vector Table (IRQ/FIQ)
2. Init the stack/s for FIQ/IRQ modes
3. Init the stack for SVC mode (SVC mode selected)
4. Configure GPIOs (I&O)
5. Configure peripheral interruption: timer/push-buttons)
6. Local enabling of configured interrupts
7. Global enabling of interrupts (SVC mode)
8. Infinite loop (polling of device/s?)

IRQ/FIQ Handler:

1. Push registers to be used
2. Source of interruption?
3. Perform handler work depending on 2
4. Clear event (notify to device IRQ/FIQ has been served)
5. Pop registers
6. Return from handler

Regular Interrupts by timer *(using comparator C1)*



Main(7)

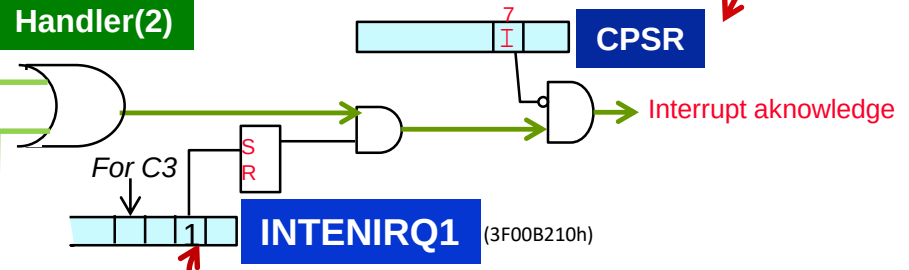
Source of interrupt ?

```
ldr   r0, =STBASE
ldr   r2, [r0, #STCS]
tst   r2,#0b0010 (1000 for C3)
```

Handler(2)

Enable IRQ (SVC mode):

```
mov   r1, #0b01010011
msr   cpsr_c, r1
```



Enable timer IRQ by comparator C1:

```
ldr   r0,=INTBASE
mov   r1, #0b0010 @(#0b1000 for C3)
str   r1,[r0,#INTENIRQ1]
```

Main(6)

Preparing C1 to send an interrupt after "*y*" microseconds

```
ldr   r0, =STBASE
ldr   r1, [r0, #STCLO]
add   r1, #y    @y microseconds
str   r1, [r0, #STC1] @#STC3 for C3
```

Main(5) and/or handler(3)

Reset (end) of the timer interrupt by comparator C1 *(to allow a new interrupt)*:

```
ldr   r0, =STBASE
mov   r1, #0b0010  @(#0b1000 for C3)
str   r1,[r0,#STCS]
   (insert just here for rearming the timer)
subs  pc, lr, #4   @to return to main
```

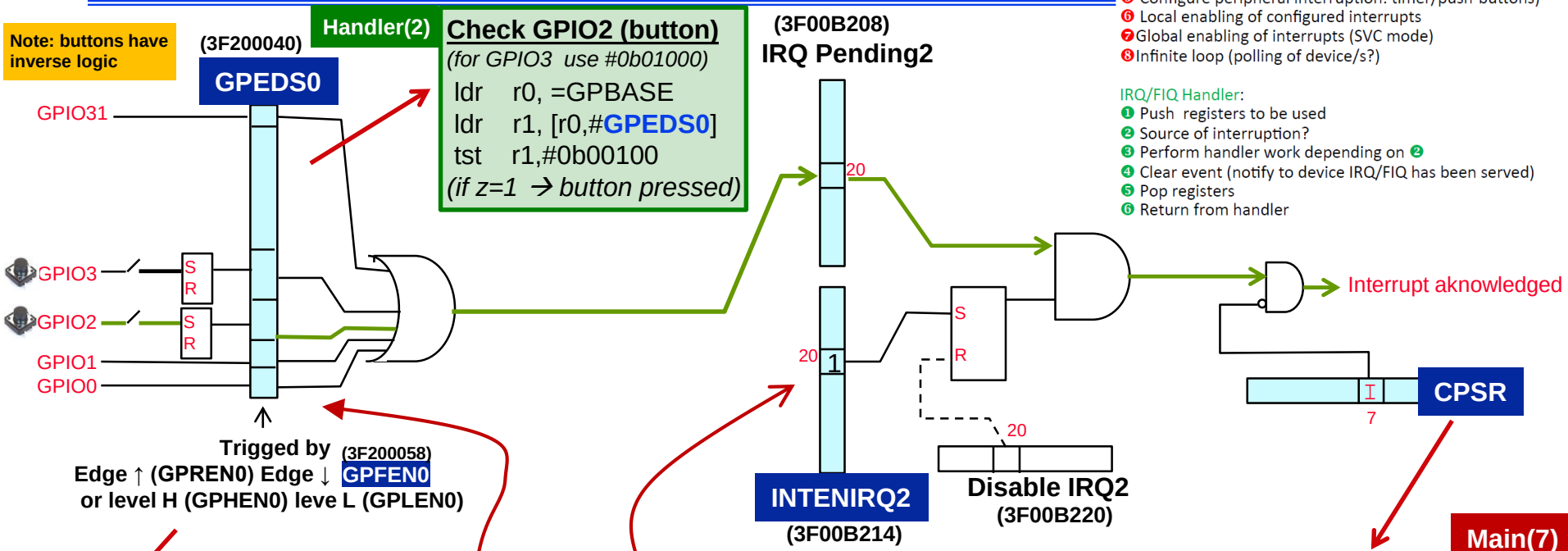
Handler(4&6))

Regular int. (IRQ) by push buttons

Main program (.text):
1. Initialize Vector Table (IRQ/FIQ)
2. Init the stack/s for FIQ/IRQ modes
3. Init the stack for SVC mode (SVC mode selected)
4. Configure GPIOs (I&O)
5. Configure peripheral interruption: timer/push-buttons)
6. Local enabling of configured interrupts
7. Global enabling of interrupts (SVC mode)
8. Infinite loop (polling of device/s?)

IRQ/FIQ Handler:
1. Push registers to be used
2. Source of interruption?
3. Perform handler work depending on 2
4. Clear event (notify to device IRQ/FIQ has been served)
5. Pop registers
6. Return from handler

Note: buttons have inverse logic



Handler(2)

Check GPIO2 (button)

(for GPIO3 use #0b01000)

```
ldr   r0, =GPBASE
ldr   r1, [r0,#GPEDS0]
tst   r1,#0b00100
```

(if z=1 → button pressed)

GPED0 Trigged by falling edge

(for both GPIO2 & 3)

```
ldr   r0, =GPBASE
mov   r1, #0b011100
str   r1,[r0,#GPFEN0]
```

Main(5)

Enable local interrupt IRQ

```
ldr   r0, =INTBASE
ldr   r1, =0x00100000
str   r1,[r0,#INTENIRQ2]
```

Main(6)

Main(7)

Enable global interrupt IRQ

(for SVC mode)

```
mov   r1, #0b01010011
msr   cpsr_c, r1
```

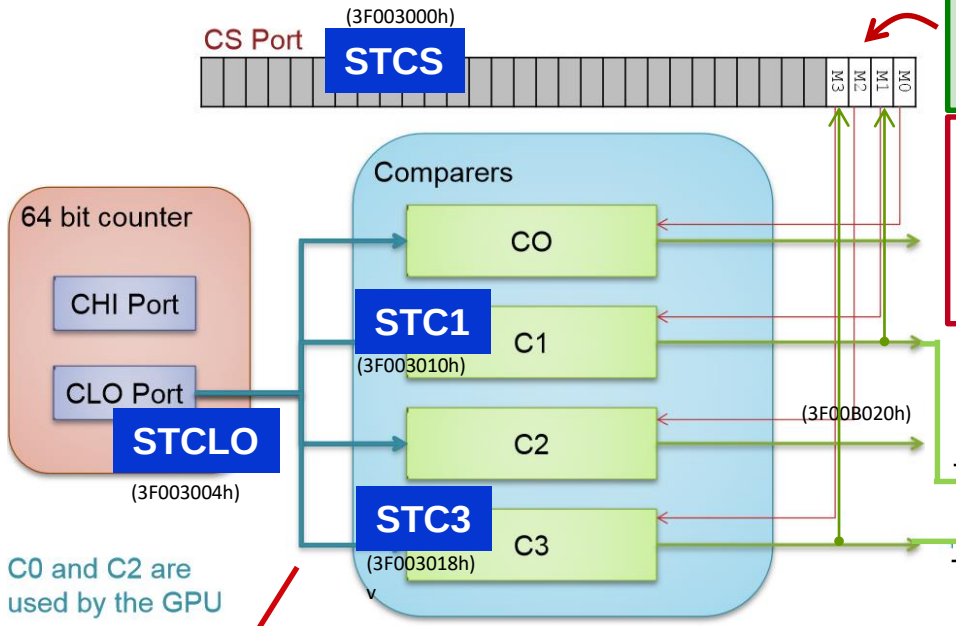
Reset (end) of interrupt IRQ2 (GPIO2&3)

(to allow a new interrupt via GPIO2&3):

```
ldr   r0, =GPBASE
mov   r1, #0b011100
str   r1,[r0,#GPEDS0]
  ...   ..............
subs  pc, lr, #4  @Return
```

Handler(4&6)

Fast Interrupts by timer *(using comparator C3)*



Handler(2)

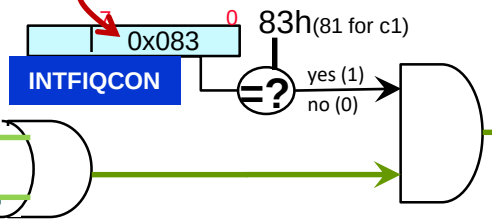
Source of interrupt ?

```
ldr   r0, =STBASE
ldr   r2, [r0, #STCS]
```

Main(6)

Enable FIQ

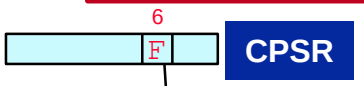
```
ldr   r0, =INTBASE
ldr   r1, =0x083
str   r1, [r0, #INTFIQCON
```



Main(7)

Enable FIQ (SVC mode):

```
mov   r1, #0b10010011
msr   cpsr_c, r1
```



Interrupt aknowledge

Main program (.text):
1. Initialize Vector Table (IRQ/FIQ)
2. Init the stack/s for FIQ/IRQ modes
3. Init the stack for SVC mode (SVC mode selected)
4. Configure GPIOs (I&O)
5. Configure peripheral interruption: timer/push-buttons)
6. Local enabling of configured interrupts
7. Global enabling of interrupts (SVC mode)
8. Infinite loop (polling of device/s?)

IRQ/FIQ Handler:
1. Push registers to be used
2. Source of interruption?
3. Perform handler work depending on 2
4. Clear event (notify to device IRQ/FIQ has been served)
5. Pop registers
6. Return from handler

Preparing C3 to send an interrupt after "*y*" microseconds

```
ldr   r0, =STBASE
ldr   r1, [r0, #STCLO]
add   r1, #y     @y microseconds
str   r1, [r0, #STC3] @#STC1 for C1
```

Main(5) and/or handler(3)

Reset (end) of the timer interrupt by comparator C3
(to allow a new interrupt):

```
ldr   r0, =STBASE
mov   r1, #0b01000 @ #0b0010 for C1
str   r1,[r0,#STCS]
   (insert just here for rearming the timer)
subs  pc, lr, #4
```

Handler(4&6)

Fast interrupts by push buttons

Main program (.text):
1. Initialize Vector Table (IRQ/FIQ)
2. Init the stack/s for FIQ/IRQ modes
3. Init the stack for SVC mode (SVC mode selected)
4. Configure GPIOs (I&O)
5. Configure peripheral interruption: timer/push-buttons)
6. Local enabling of configured interrupts
7. Global enabling of interrupts (SVC mode)
8. Infinite loop (polling of device/s?)

IRQ/FIQ Handler:
1. Push registers to be used
2. Source of interruption?
3. Perform handler work depending on 2
4. Clear event (notify to device IRQ/FIQ has been served)
5. Pop registers
6. Return from handler

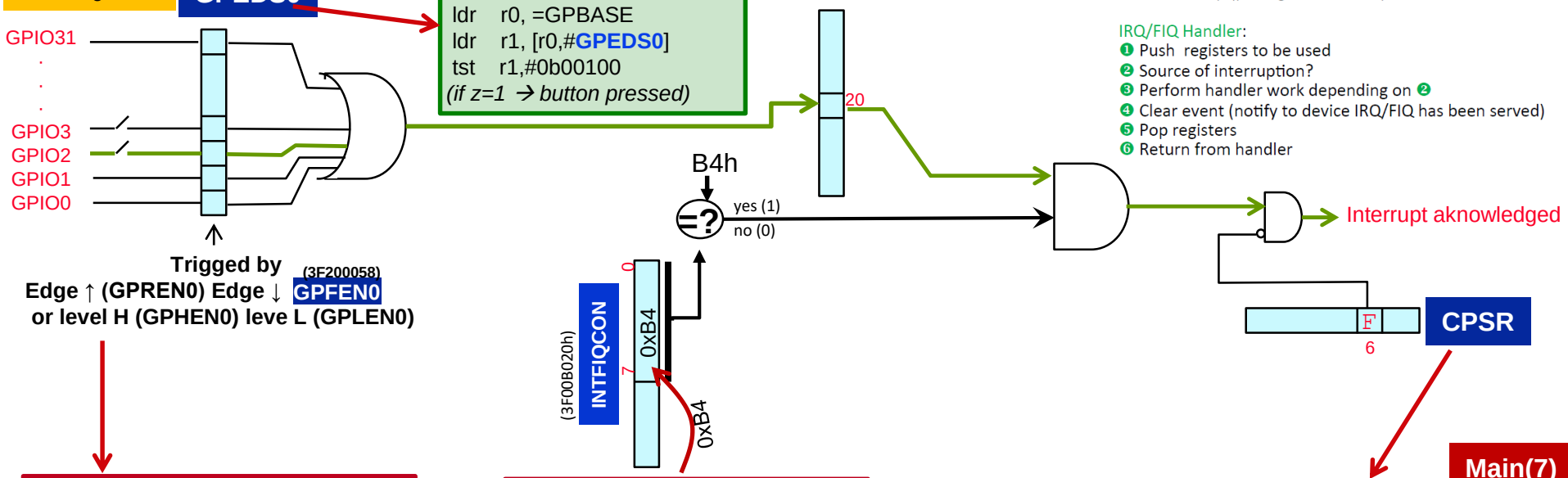
Note: buttons have inverse logic

(3F200040) **GPEDS0**

Handler(2)

Check GPIO2 (button)
(for GPIO3 use #0b01000)
```
ldr   r0, =GPBASE
ldr   r1, [r0,#GPEDS0]
tst   r1,#0b00100
```
(if z=1 → button pressed)

(3F00B208) **IRQ Pending2**



Trigged by
Edge ↑ (GPREN0) Edge ↓ (3F200058) **GPFEN0**
or level H (GPHEN0) leve L (GPLEN0)

Trigged by falling edge
(for both GPIO2 & 3)
```
ldr   r0, =GPBASE
mov   r1, #0b01100
str   r1,[r0,#GPFEN0]
```
Main(5)

Enable local FIQ **Main(6)**
```
ldr   r0, =INTBASE
ldr   r8, =0x0B4
str   r8, [r0, #INTFIQCON]
```

Main(7)

Enable global interrupt FIQ
(for SVC mode)
```
mov   r1, #0b10010011
msr   cpsr_c, r1
```

Reset (end) of interrupt IRQ2 (GPIO2&3)
(to allow a new interrupt):
```
ldr   r0, =GPBASE
mov   r1, #0b01100
str   r1,[r0,#GPEDS0]
.....
subs  pc, lr, #4
```
Handler (4&6)

Exception handler

- Basic structure of a exception handler
  - Interruption: the return is done by lr-4
  - Internal exception (as data abort): the return is done by lr-8

```
irq_handler:
        push    {lista registros}
        ...
        pop     {lista registros}
        subs    pc, lr, #4
```

  - User must manage A, I and F flags to disable/enable nesting of new exceptions and interruptions.
    - Initially the interruptions are disabled (I=F=1).