# Exception and Interrupt Handling

Hsiao-Lung Chan, Ph.D.

Dept Electrical Engineering

Chang Gung University, Taiwan

chanhl@maili.cgu.edu.tw

# ARM registers and operation modes

- 31 general-purpose 32-bit register
- 6 status register

General registers
& Program counter

Banked registers :
available only
processor is in
a particular mode

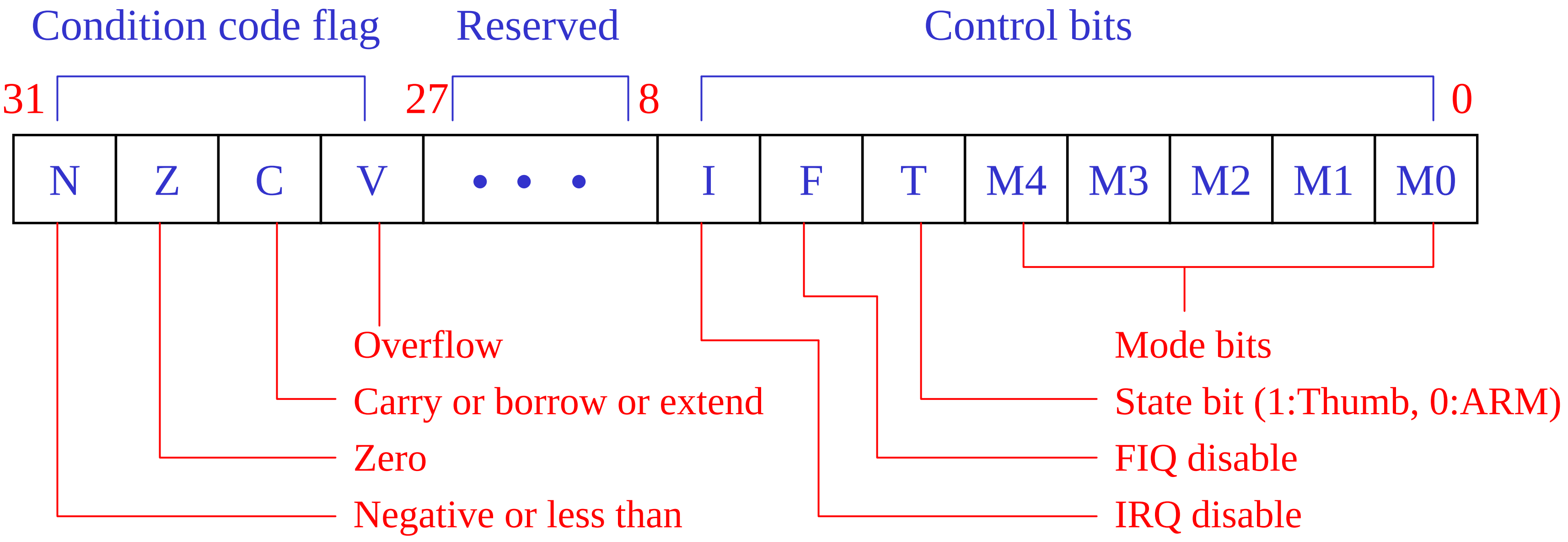
Program
status registers

| System & User | FIQ | Supervisor | Abort | IRQ | Undefined |
|---|---|---|---|---|---|
| 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 | R13_fiq | R13_svc | R13_abt | R13_irq | R13_und |
| r14 | R14_fiq | R14_svc | R14_abt | R14_irq | R14_und |
| r15(PC) | r15(PC) | r15(PC) | r15(PC) | r15(PC) | r15(PC) |
| CPSR | CPSR | CPSR | CPSR | CPSR | CPSR |
|  | SPSR_fiq | SPSR_svc | SPSR_abt | SPSR_irq | SPSR_und |

# Operation modes of ARM

- User (usr) mode
  - Program execution state
- Fast interrupt (fiq) mode
  - Data transfer or channel process
- Interrupt (irq) mode
  - General-purpose interrupt handling
- Supervised (svc) mode
  - Protected mode for operating system
- Abort (abt) mode
  - Entered after a data or instruction prefetch Abort
- System (sys) mode
  - Privileged user mode of operating system
- Undefined (und) mode
  - Entered when a undefined instruction is executed

# Program status register



Note: N, V : Relate to 2’s complement arithmetic

# ARM exception priority order

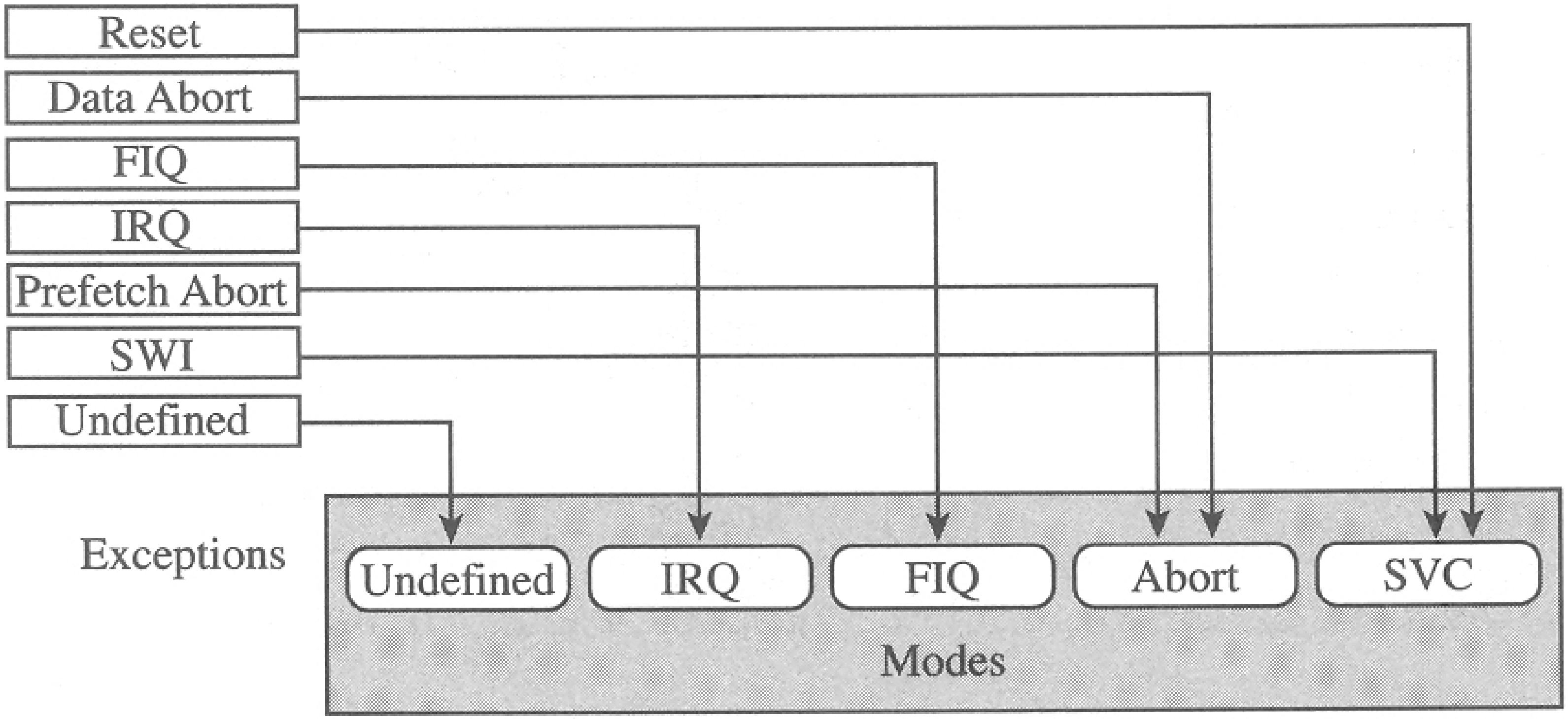
- Exception arise whenever the normal flow of a program has to be halted temporally.

| Priority | Exception |
|---|---|
| Highest | Reset |
| | Data Abort |
| | FIQ |
| | IRQ |
| | Prefetch Abort |
| Lowest | Undefined instruction or software interrupt |

# Exception handling

- Exception
  - Any condition that needs to halt normal sequential execution of instructions
    - ARM core is reset
    - Instruction fetch or memory access fails
    - Undefined instruction is encountered
    - Software interrupt instruction is executed
    - External interrupt has been raised
  - Exception handling
    - The method of processing these exceptions
    - Exception handler

# ARM processor exceptions and modes



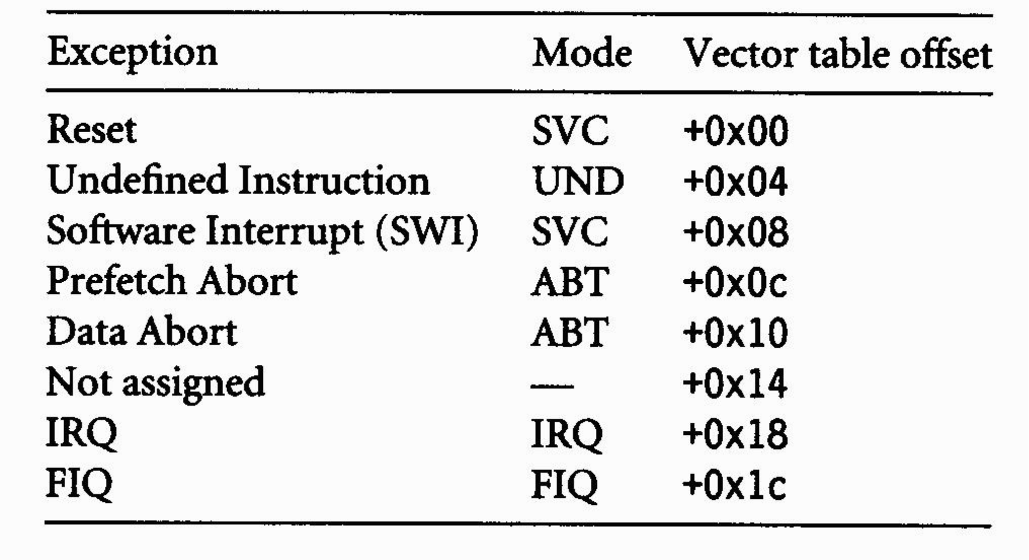
# ARM processor exceptions and modes (cont.)

- When an exception causes a mode change, the core automatically
  - Save CPSR to SPSR of the exception mode
  - Save PC to LR of the exception mode
  - Sets CPSR to the execution mode
  - Sets PC to the address of the exception handler

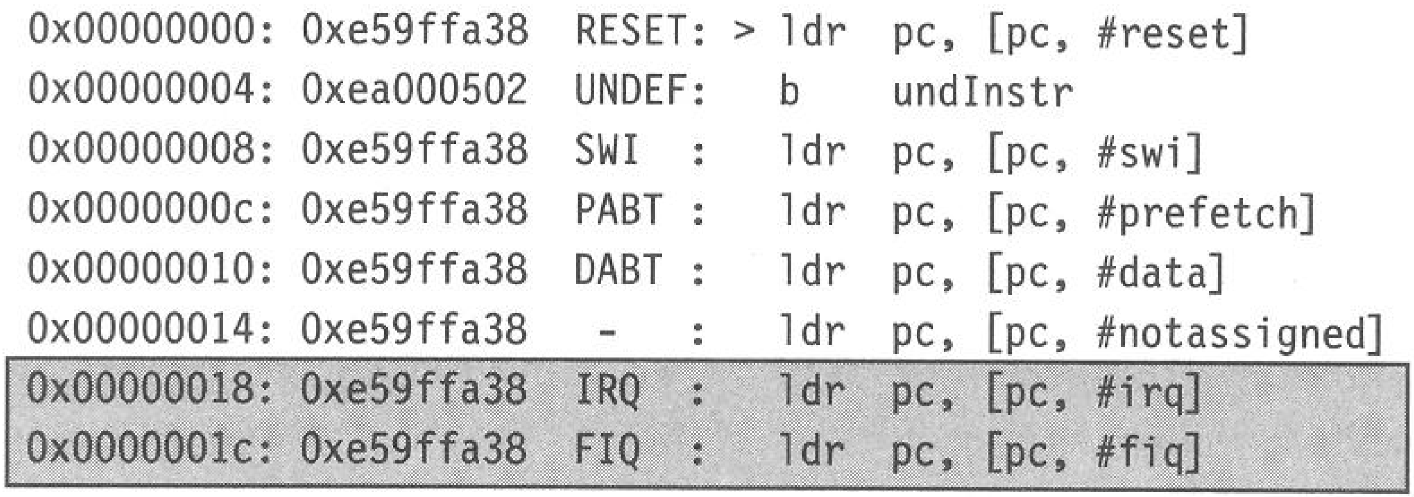
# Vector tables

- A table of addresses that the ARM core branches to when an exception is raised
- These addressed contain branch instructions
  - B <address>
  - LDR pc, [pc, #offset]

# Vector tables (cont.)



| Exception | Mode | Vector table offset |
|---|---|---|
| Reset | SVC | +0x00 |
| Undefined Instruction | UND | +0x04 |
| Software Interrupt (SWI) | SVC | +0x08 |
| Prefetch Abort | ABT | +0x0c |
| Data Abort | ABT | +0x10 |
| Not assigned | — | +0x14 |
| IRQ | IRQ | +0x18 |
| FIQ | FIQ | +0x1c |



```
0x00000000: 0xe59ffa38  RESET: > ldr  pc, [pc, #reset]
0x00000004: 0xea000502  UNDEF:   b    undInstr
0x00000008: 0xe59ffa38  SWI  :   ldr  pc, [pc, #swi]
0x0000000c: 0xe59ffa38  PABT :   ldr  pc, [pc, #prefetch]
0x00000010: 0xe59ffa38  DABT :   ldr  pc, [pc, #data]
0x00000014: 0xe59ffa38   -   :   ldr  pc, [pc, #notassigned]
0x00000018: 0xe59ffa38  IRQ  :   ldr  pc, [pc, #irq]
0x0000001c: 0xe59ffa38  FIQ  :   ldr  pc, [pc, #fiq]
```

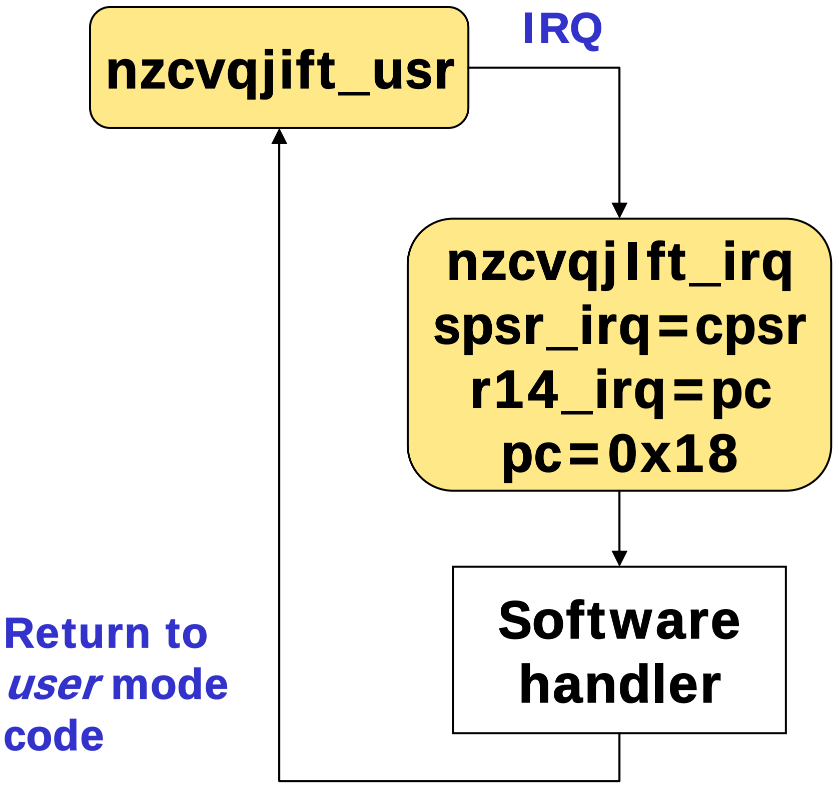
# Interrupts

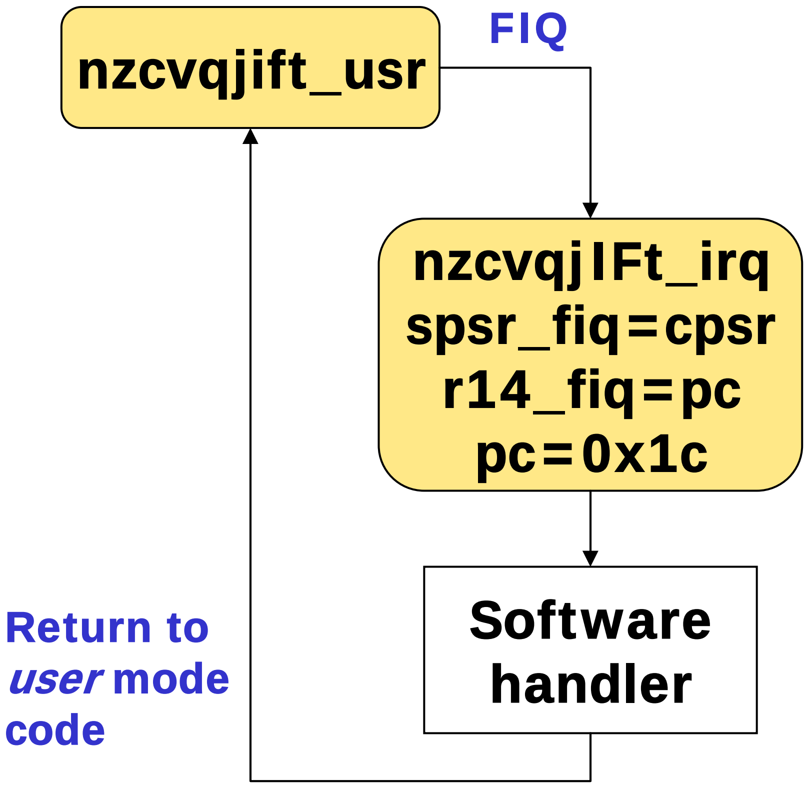
- Assigning interrupts
- IRQ and FIQ exceptions
- Interrupt latency
- Interrupt stack design

# Assigning interrupts

- Interrupt controller
  - Multiple external interrupts to one if the two ARM interrupt requests
- Standard design practice
  - SWI are reserved to call privileged operating system routines
  - IRQ are assigned for general-purpose interrupts
    - A periodic timer
  - FIQ are reserved for a single interrupt source that require a fast response time
    - Direct memory access to move blocks of memory
    - FIQ has a higher priority and shorter interrupt latency than IRQ

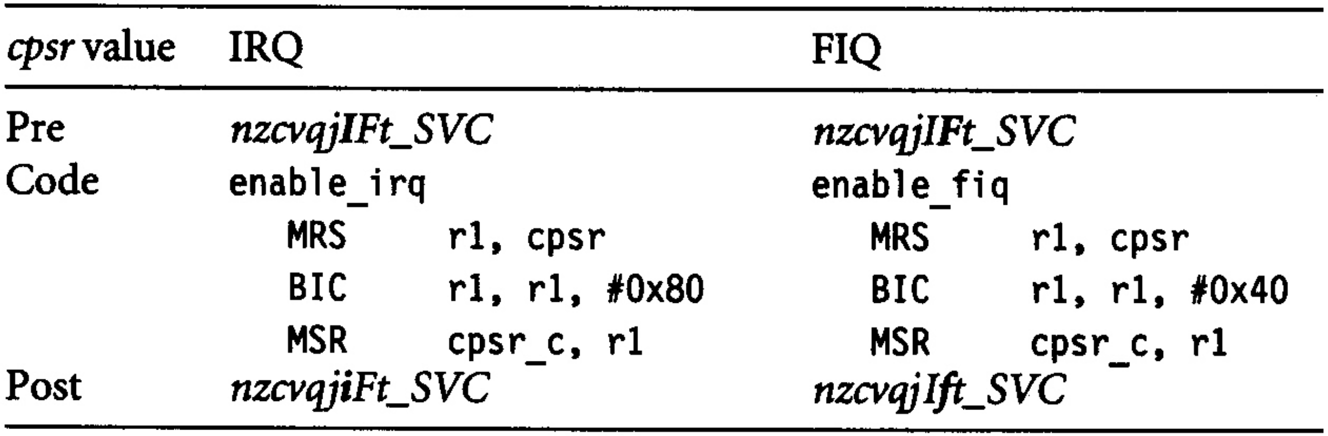
# IRQ and FIQ exceptions





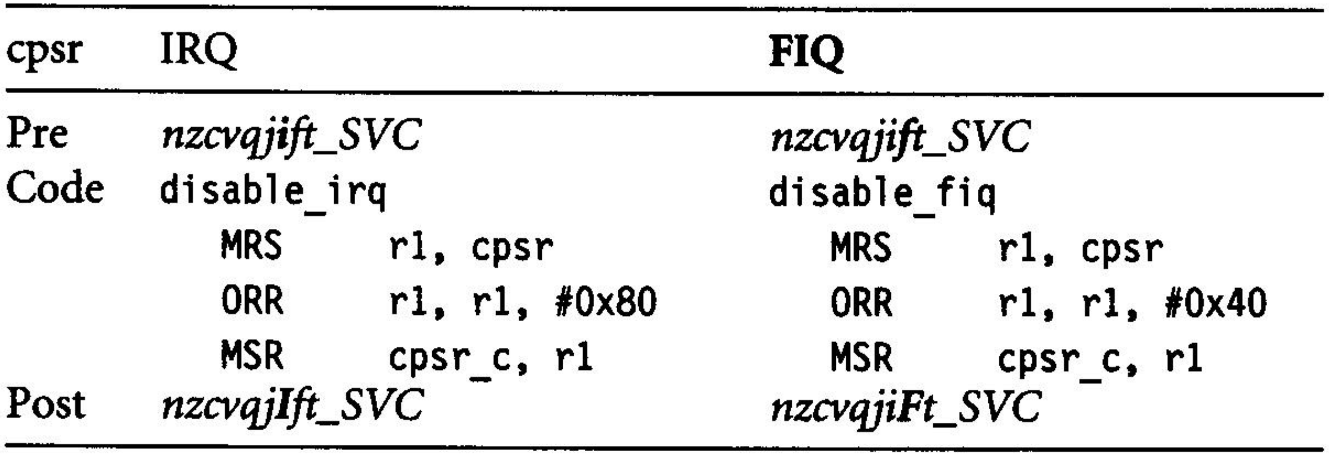
# Enabling and disabling IRQ and FIQ exceptions

**Enabling an interrupt**



| *cpsr* value | IRQ | FIQ |
|---|---|---|
| Pre | *nzcvqjIFt_SVC* | *nzcvqjIFt_SVC* |
| Code | `enable_irq` | `enable_fiq` |
| | `MRS r1, cpsr` | `MRS r1, cpsr` |
| | `BIC r1, r1, #0x80` | `BIC r1, r1, #0x40` |
| | `MSR cpsr_c, r1` | `MSR cpsr_c, r1` |
| Post | *nzcvqjiFt_SVC* | *nzcvqjIft_SVC* |

**Disabling an interrupt**



| cpsr | IRQ | **FIQ** |
|---|---|---|
| Pre | *nzcvqjift_SVC* | *nzcvqjift_SVC* |
| Code | `disable_irq` | `disable_fiq` |
| | `MRS r1, cpsr` | `MRS r1, cpsr` |
| | `ORR r1, r1, #0x80` | `ORR r1, r1, #0x40` |
| | `MSR cpsr_c, r1` | `MSR cpsr_c, r1` |
| Post | *nzcvqjIft_SVC* | *nzcvqjiFt_SVC* |

# Interrupt latency

- Interval of time from an external interrupt request signal being raised to the first fetch of an instruction of interrupt service routine (ISR)

# Program status register instructions

MRS{<Cond>} Rd, <CPSR|SPSR>

- Copy program status register to general-purpose register

MSR{<Cond>} <CPSR|SPSR>_<fields>, Rm

- Copy general-purpose register to program status register

MRS{<Cond>} <CPSR|SPSR>_<fields>, #immediate

Copy program status register to general-purpose register

<Pre-condition>

CPSR = nzcvqIFt_SVC

→

<Post-condition>

CPSR=nzcvqiFt_SVC

```
MRS    r1, cpsr
BIC     r1, r1, #0x80
MSR    cpsr_c, r1
```

# Examples: program state register transfer

```
MRS   r0, cpsr              ; read CPSR
BIC   r0, r0, #f0000000     ; clear N,Z,C,V flags
MSR   cpsr, r0              ; update the flag in CPSR

MRS   r0,cpsr          ; read CPSR
BIC   r0, r0, #0x1F    ; clear mode bits
ORR   r0, r0, #0x13    ; set the mode bits to supervised mode
 MSR   cpsr, r0         ; update the control bits in CPSR

MRS   r0,cpsr          ; read CPSR
ORR   r0, r0, #0x80    ; set interrupt disable bit
 MSR   cpsr, r0         ; update the control bits in CPSR
```

# Software interrupt (SWI) instruction

- Use SWI instruction to enter supervisor mode for calling operating system routines

  SWI{<Cond>}   SWI_number

  - lr_svc = address of next instruction following SWI
  - SPSR_svc = CPSR
  - pc = 0x08
  - CPSR mode = SVC
  - CPSR I = 1 (mask IRQ interrupts)

- After SWI routine, return to the calling program by

  MOV pc, lr_svc

# SWI example

- SWI call with SWI number 0x123456, used by ARM toolkits as a debugging SWI

```
<Pre-condition>
    CPSR = nzcVqift_USER
    pc = 0x00008000
    lr = 0x003fffff
    r0 = 0x12    ; used for parameter passing
 0x00008000   SWI  0x123456
<Post-condition>
    CPSR = nzcVqIft_SVC
    SPSR_svc = nzcVqift_USER
    pc = 0x00000008
    lr = 0x00008004
    r0 = 0x12
```

# Interrupt stack design

- Each operation system has its own requirement for stack design
- Stack pointers are initialized after reset

# Reset handler

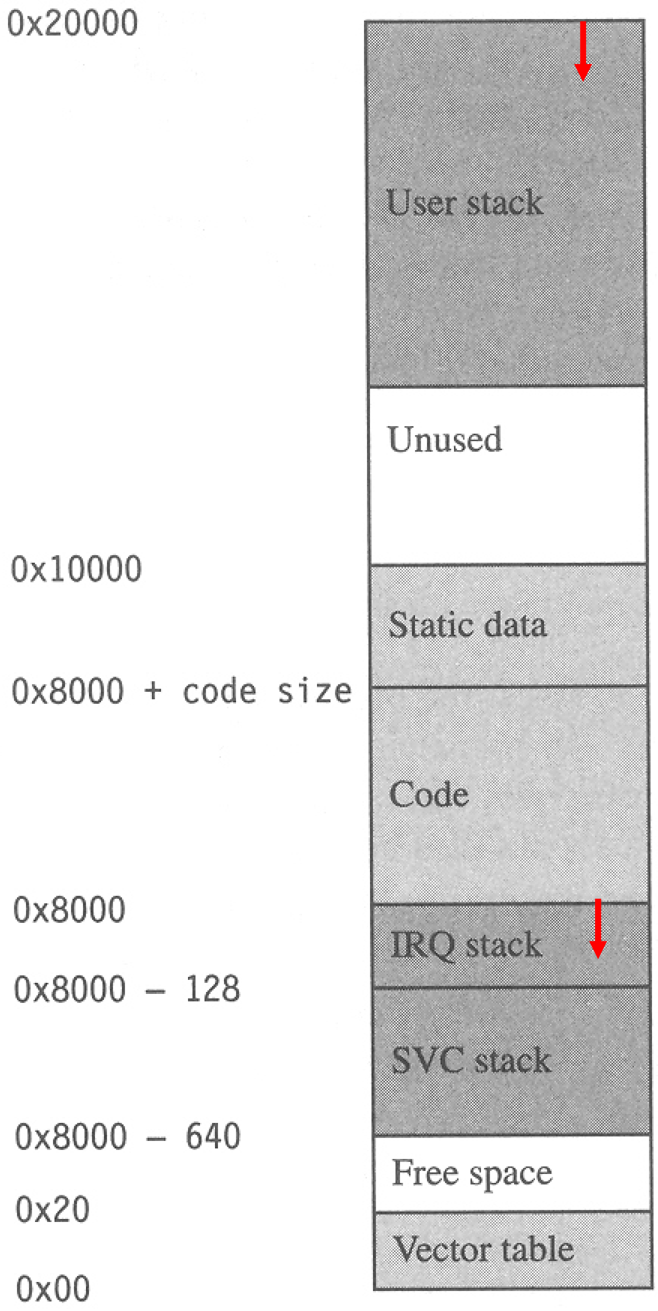
- Initialize system
  - Setting up memory and caches
  - Setting up stack pointers for all processor modes

# Example to set up stacks

```
USR_Stack   EQU  0x20000
IRQ_Stack   EQU  0x8000
SVC_Stack   EQU  IRQ_Stack-128
…
Usr32md   EQU  0x10
FIQ32md    EQU  0x11
IRQ32md   EQU  0x12
SVC32md   EQU  0x13
Abt32md   EQU  0x17
Und32md  EQU  0x1b
Sys32md   EQU  0x1f
NoInt         EQU  0xc0       ; Disable interrupts
```

# Example to set up stacks (cont.)

```
; Set up supervisor mode stack
                LDR   r13, SVC_NewStack     ; set r13_svc
; Set up IRQ stack
                LDR   r2, #NoInt | IRQ32md
                MSR  cpsr_c, r2              ; make r13_irq visible
                LDR   r13, IRQ_NewStack     ; set r13_irq
; Set user mode stack
                LDR   r2, #Sys32md
                MSR  cpsr_c, r2              ; make r13_sys visible
                LDR   r13, USR_NewStack     ; set r13_usr
                …
SVC_NewStack   DCD   SVC_Stack
IRQ_NewStack   DCD   IRQ_Stack
USR_NewStack   DCD   USR_Stack
```

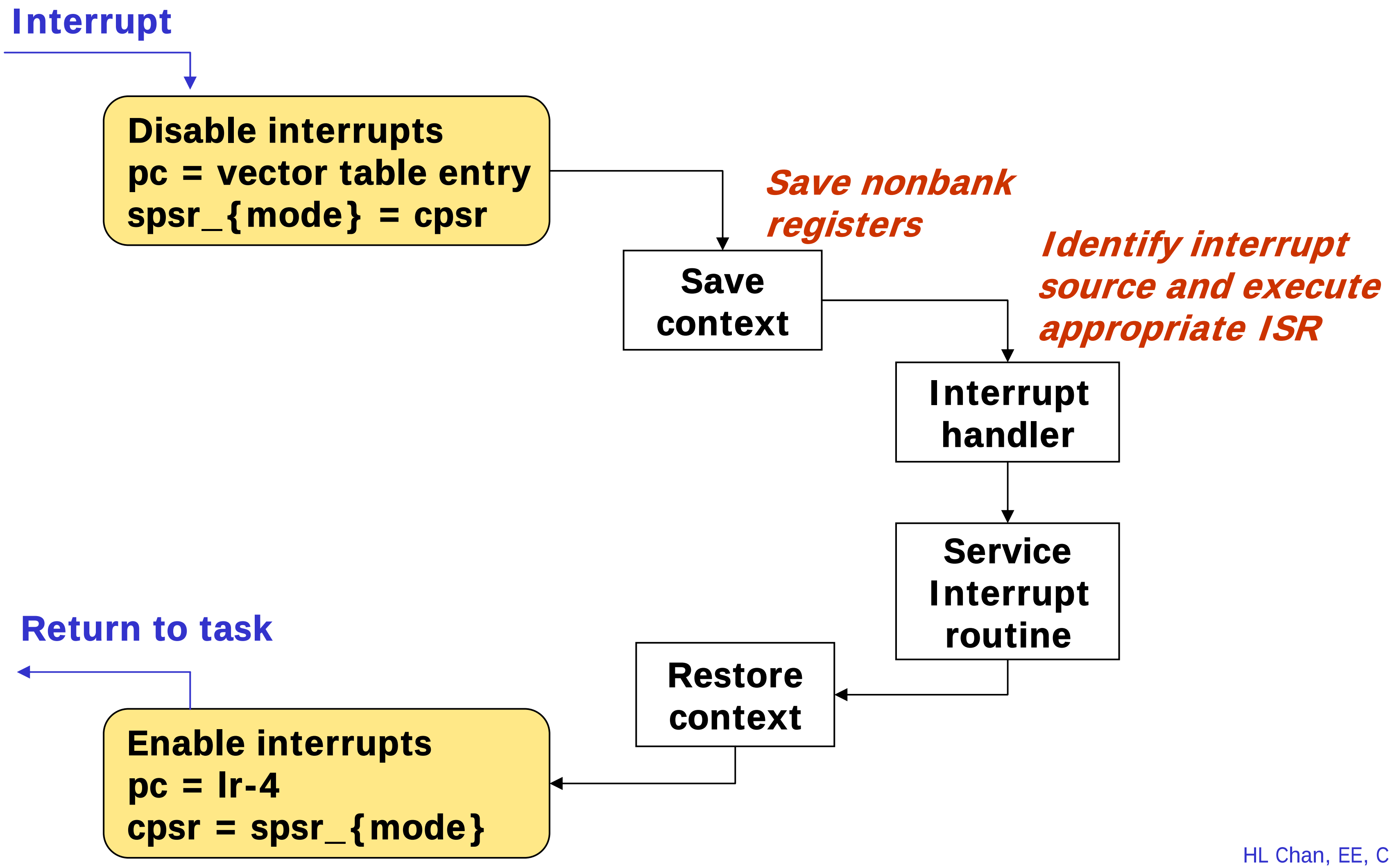


0x20000
User stack
Unused
0x10000
Static data
0x8000 + code size
Code
0x8000
IRQ stack
0x8000 – 128
SVC stack
0x8000 – 640
Free space
0x20
Vector table
0x00

# Exercise

- Write a ARM program to set stacks for all exception modes

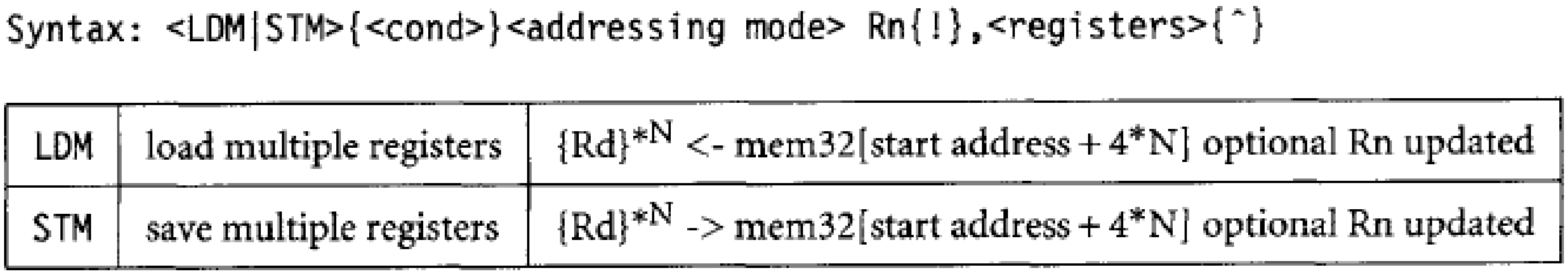
# Non-nested interrupt handler



# Non-nested interrupt handler (cont.)

```
interrupt_handler
        SUB     r14, r14, #4            ; adjust lr
        STMFD   sp!,{r0-r3,r12,r14}     ; save context
        LDR     r0,=IRQStatus           ; interrupt status address
        LDR     r0,[r0]                 ; get interrupt status
        TST     r0,#0x0080
        BNE     timer_isr               ; branch timer ISR
        TST     r0,#0x0001
        BNE     button_isr              ; call button ISR
        LDMFD   sp!,{r0-r3,r12,pc}^     ; restore context and return
```

# Load-Store Instructions (Multiple-Register Transfer)

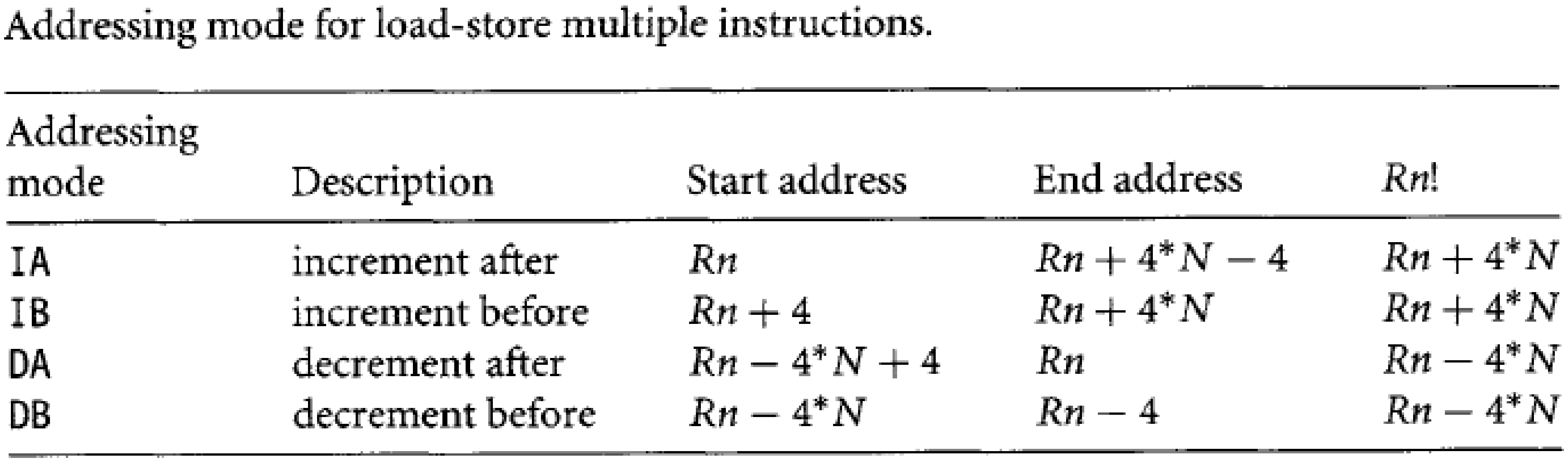


```
Syntax: <LDM|STM>{<cond>}<addressing mode> Rn{!},<registers>{^}
```

| | | |
|---|---|---|
| LDM | load multiple registers | {Rd}$^{*N}$ <- mem32[start address + 4*N] optional Rn updated |
| STM | save multiple registers | {Rd}$^{*N}$ -> mem32[start address + 4*N] optional Rn updated |

{!} : Store the modified address to Rn (W=1)

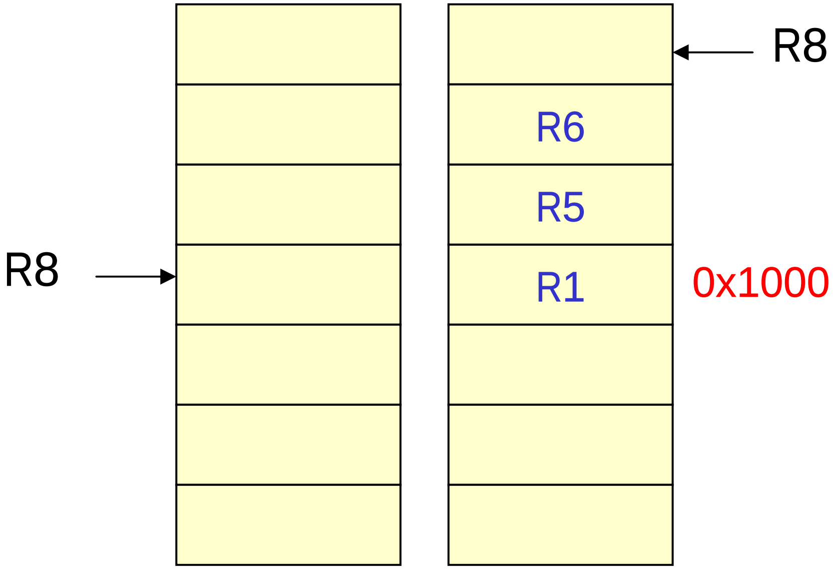
{^} : S=1, determine whether R15 can be in <registers> list



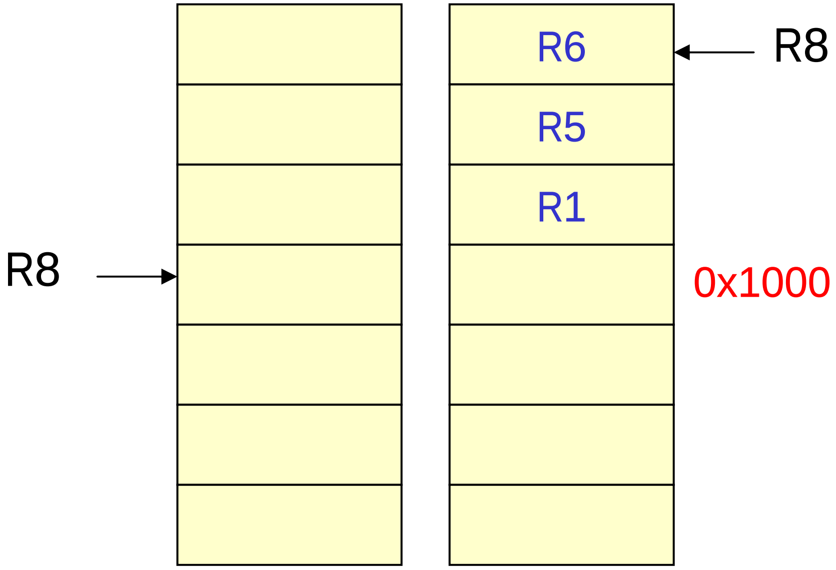
Addressing mode for load-store multiple instructions.

| Addressing mode | Description | Start address | End address | $Rn!$ |
|---|---|---|---|---|
| IA | increment after | $Rn$ | $Rn + 4^{*}N - 4$ | $Rn + 4^{*}N$ |
| IB | increment before | $Rn + 4$ | $Rn + 4^{*}N$ | $Rn + 4^{*}N$ |
| DA | decrement after | $Rn - 4^{*}N + 4$ | $Rn$ | $Rn - 4^{*}N$ |
| DB | decrement before | $Rn - 4^{*}N$ | $Rn - 4$ | $Rn - 4^{*}N$ |

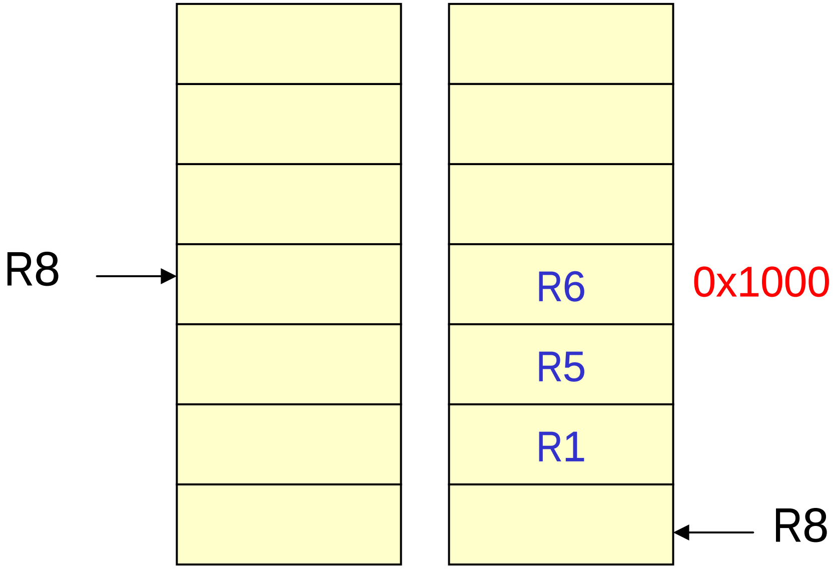
LDMIA r8!, {r1, r5, r6}



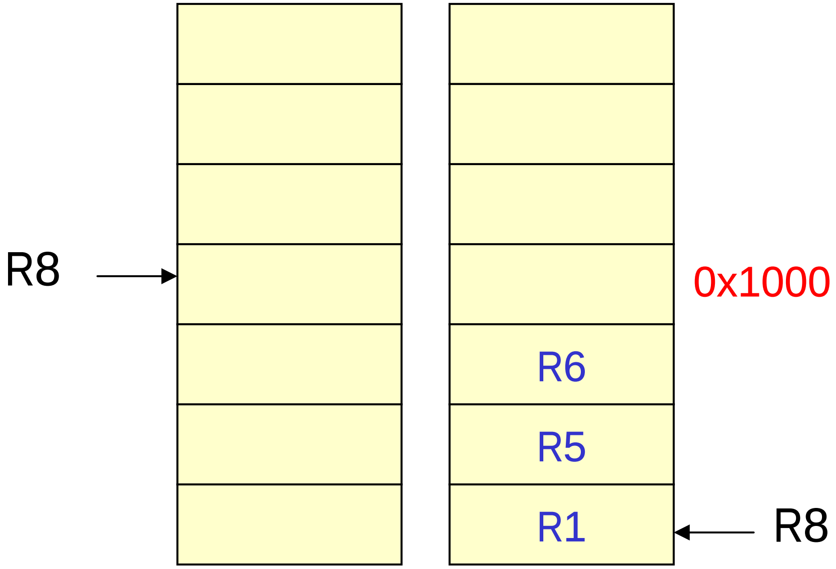
LDMIB r8!, {r1, r5, r6}



LDMDA r8!, {r1, r5, r6}



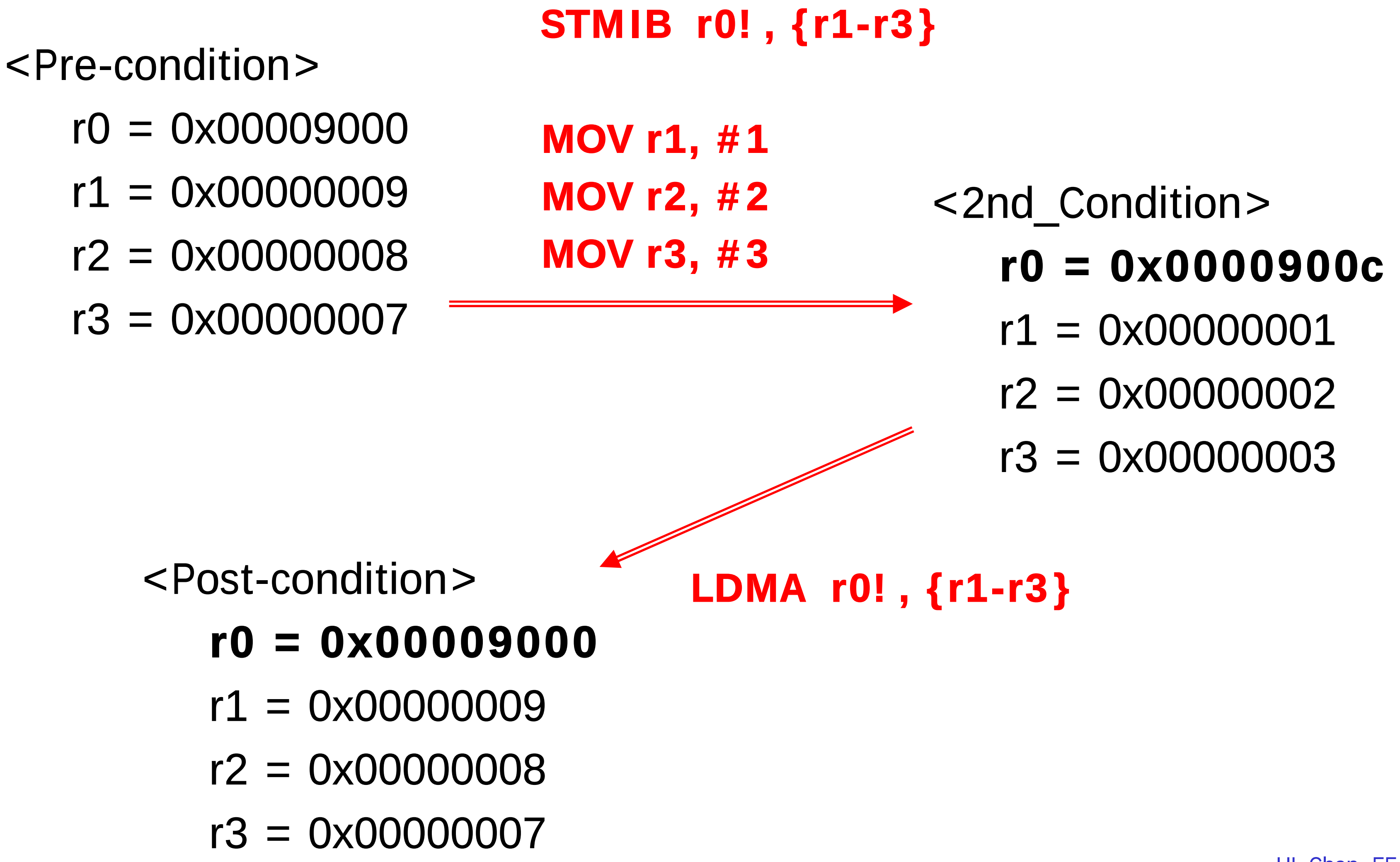
LDMDB r8!, {r1, r5, r6}



# Load-Store Multiple Pairs when base update used

- STMIA <- - -> LDMDB
- STMIB <- - -> LDMDA
- STMDA <- - -> LDMIB
- STMDB <- - -> LDMIA

# Load-Store Multiple Instructions Example



**STMIB r0! , {r1-r3}**

<Pre-condition>

r0 = 0x00009000

r1 = 0x00000009

r2 = 0x00000008

r3 = 0x00000007

**MOV r1, #1**

**MOV r2, #2**

**MOV r3, #3**

<2nd_Condition>

**r0 = 0x0000900c**

r1 = 0x00000001

r2 = 0x00000002

r3 = 0x00000003

**LDMA r0! , {r1-r3}**

<Post-condition>

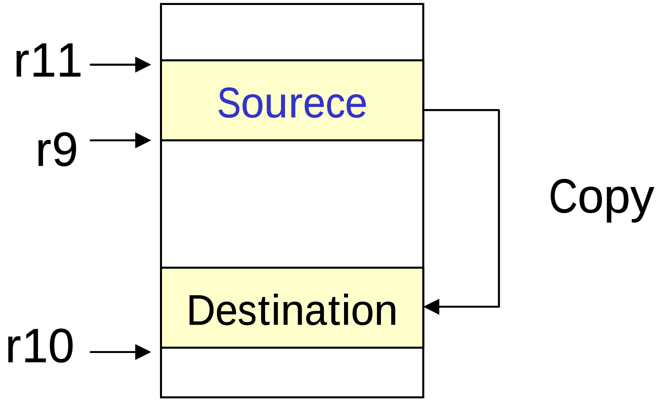
**r0 = 0x00009000**

r1 = 0x00000009

r2 = 0x00000008

r3 = 0x00000007

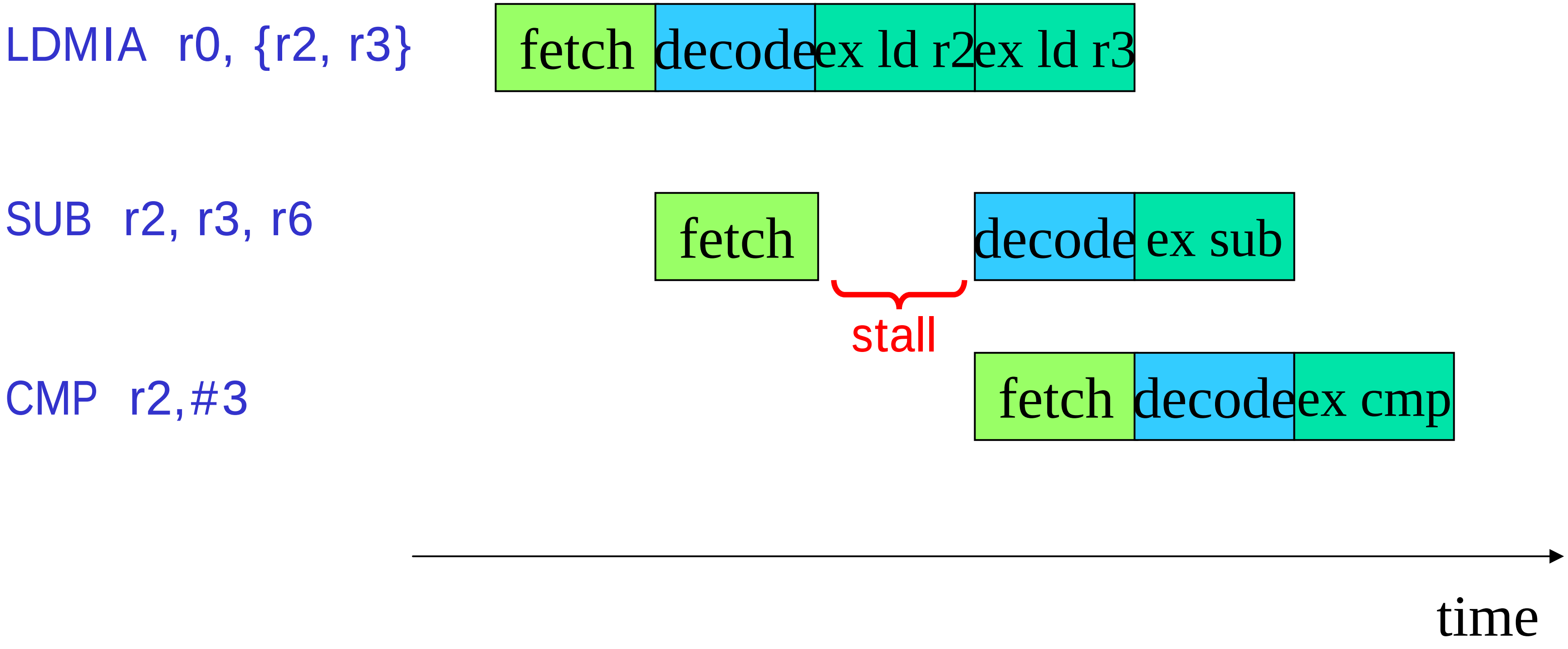
# Load-Store Multiple Instruction for Block Memory Copy



```
; r9 points to start of source data
; r10 points to start of destination data
; r11 points to end of the source

loop
    ; load 32 bytes from source and update r9 pointer
    LDMIA  r9!, {r0-r7}
    ; store 32 bytes to destination and update r10 pointer
    STMIA r10!, {r0-r7}
    ; have reached the end?
    CMP    r9, r11
    BNE    loop
```

# Pipeline Stalls: Data Stall
# (ARM multi-cycle LDMIA instruction)



LDMIA r0, {r2, r3}

fetch | decode | ex ld r2 | ex ld r3

SUB r2, r3, r6

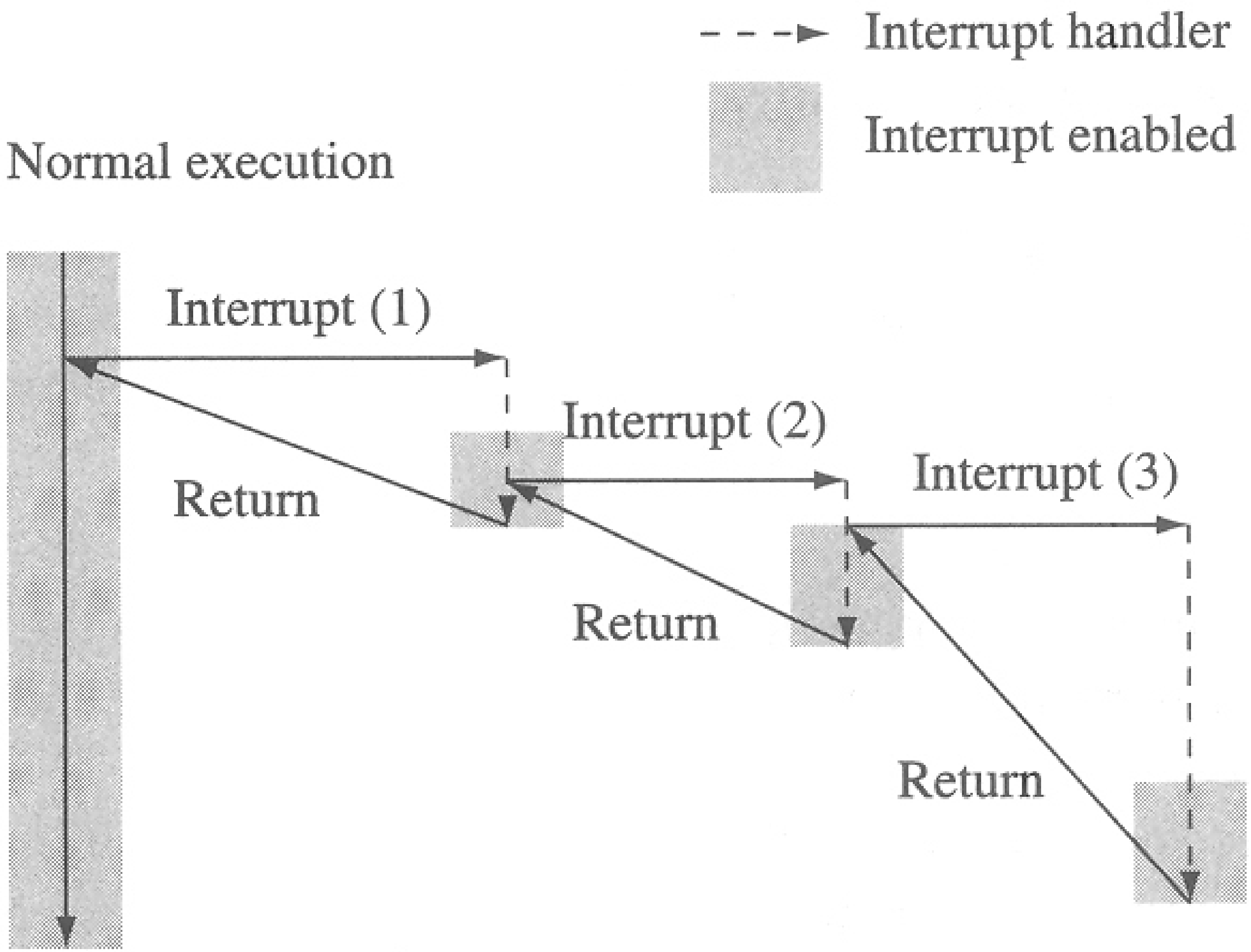
fetch | stall | decode | ex sub

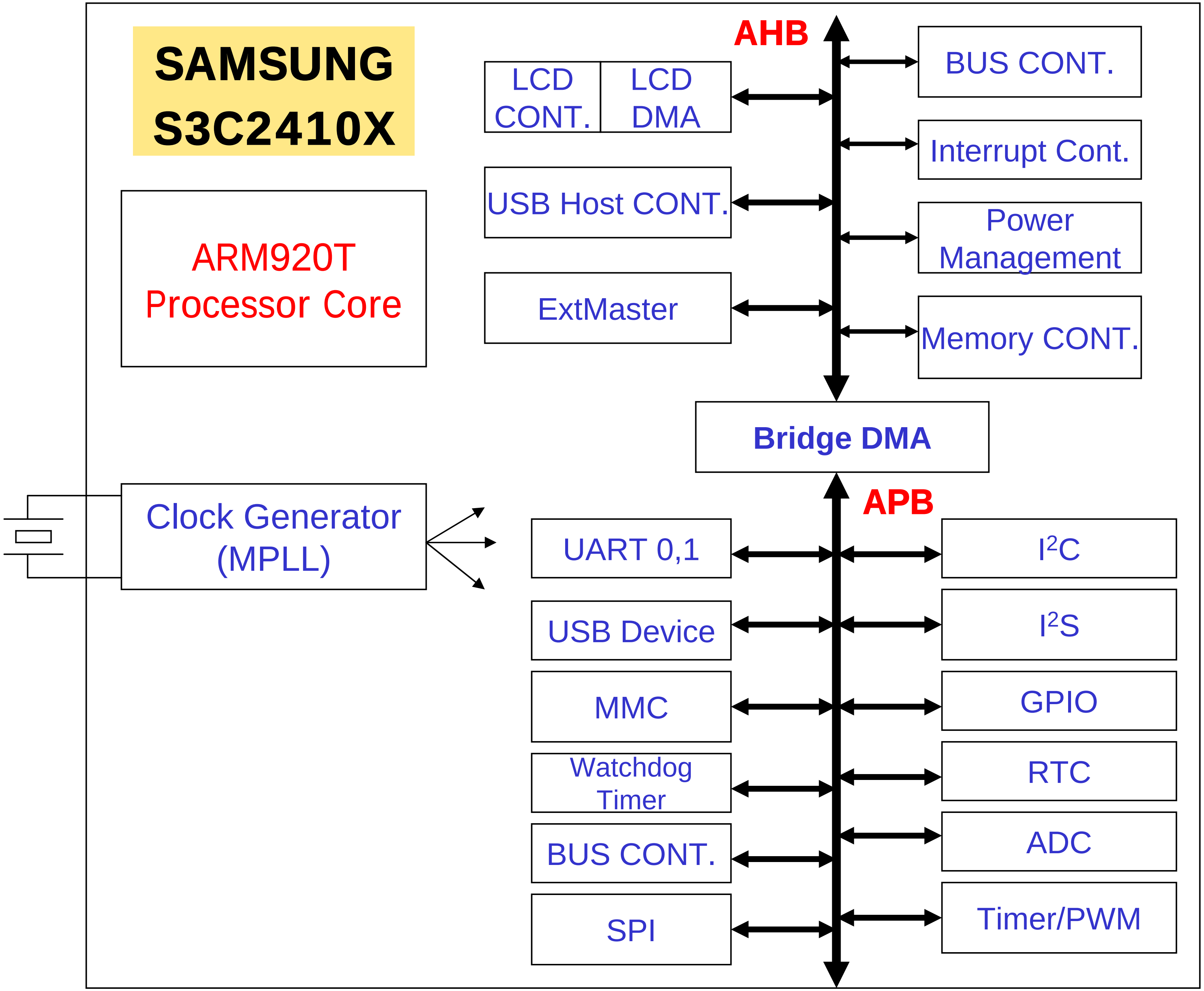
CMP r2,#3

fetch | decode | ex cmp

time

# Nested interrupt





SAMSUNG S3C2410X

ARM920T Processor Core

AHB

LCD CONT.

LCD DMA

USB Host CONT.

ExtMaster

BUS CONT.

Interrupt Cont.

Power Management

Memory CONT.

Bridge DMA

APB

Clock Generator (MPLL)

UART 0,1

USB Device

MMC

Watchdog Timer

BUS CONT.

SPI

$I^2C$

$I^2S$

GPIO

RTC

ADC

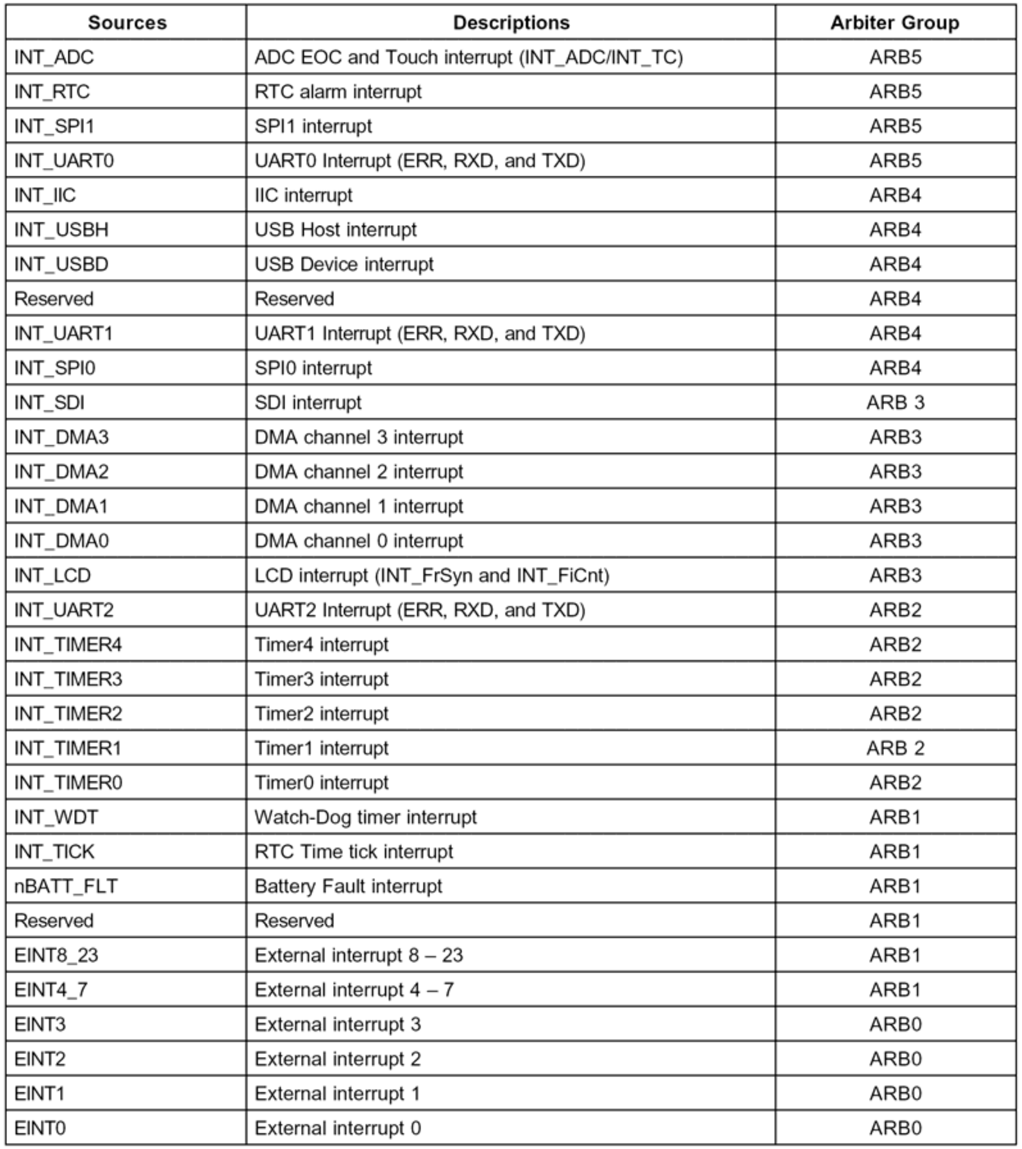
Timer/PWM

# Interrupt controller in S3C2410

- Receives request from 56 interrupt sources
- When receiving multiple interrupt requests, interrupt controller requests FIQ or IRQ interrupt of ARM920T core after arbitration procedure
- Arbitration procedure
  - Depends on hardware priority logic
  - Result is written to the interrupt pending register, which helps users notify which interrupt is generated

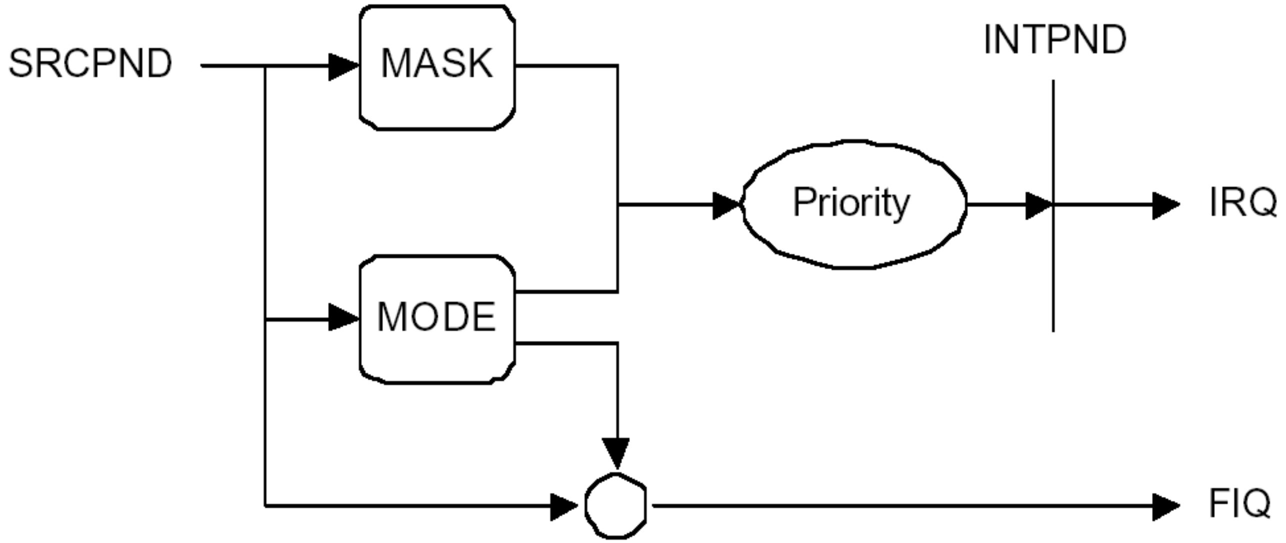
# Interrupt sources in S3C2410

- Internal peripherals
  - DMA controller
  - UART
  - IIC
  - others
- External interrupts



| Sources | Descriptions | Arbiter Group |
|---|---|---|
| INT_ADC | ADC EOC and Touch interrupt (INT_ADC/INT_TC) | ARB5 |
| INT_RTC | RTC alarm interrupt | ARB5 |
| INT_SPI1 | SPI1 interrupt | ARB5 |
| INT_UART0 | UART0 Interrupt (ERR, RXD, and TXD) | ARB5 |
| INT_IIC | IIC interrupt | ARB4 |
| INT_USBH | USB Host interrupt | ARB4 |
| INT_USBD | USB Device interrupt | ARB4 |
| Reserved | Reserved | ARB4 |
| INT_UART1 | UART1 Interrupt (ERR, RXD, and TXD) | ARB4 |
| INT_SPI0 | SPI0 interrupt | ARB4 |
| INT_SDI | SDI interrupt | ARB 3 |
| INT_DMA3 | DMA channel 3 interrupt | ARB3 |
| INT_DMA2 | DMA channel 2 interrupt | ARB3 |
| INT_DMA1 | DMA channel 1 interrupt | ARB3 |
| INT_DMA0 | DMA channel 0 interrupt | ARB3 |
| INT_LCD | LCD interrupt (INT_FrSyn and INT_FiCnt) | ARB3 |
| INT_UART2 | UART2 Interrupt (ERR, RXD, and TXD) | ARB2 |
| INT_TIMER4 | Timer4 interrupt | ARB2 |
| INT_TIMER3 | Timer3 interrupt | ARB2 |
| INT_TIMER2 | Timer2 interrupt | ARB2 |
| INT_TIMER1 | Timer1 interrupt | ARB 2 |
| INT_TIMER0 | Timer0 interrupt | ARB2 |
| INT_WDT | Watch-Dog timer interrupt | ARB1 |
| INT_TICK | RTC Time tick interrupt | ARB1 |
| nBATT_FLT | Battery Fault interrupt | ARB1 |
| Reserved | Reserved | ARB1 |
| EINT8_23 | External interrupt 8 – 23 | ARB1 |
| EINT4_7 | External interrupt 4 – 7 | ARB1 |
| EINT3 | External interrupt 3 | ARB0 |
| EINT2 | External interrupt 2 | ARB0 |
| EINT1 | External interrupt 1 | ARB0 |
| EINT0 | External interrupt 0 | ARB0 |

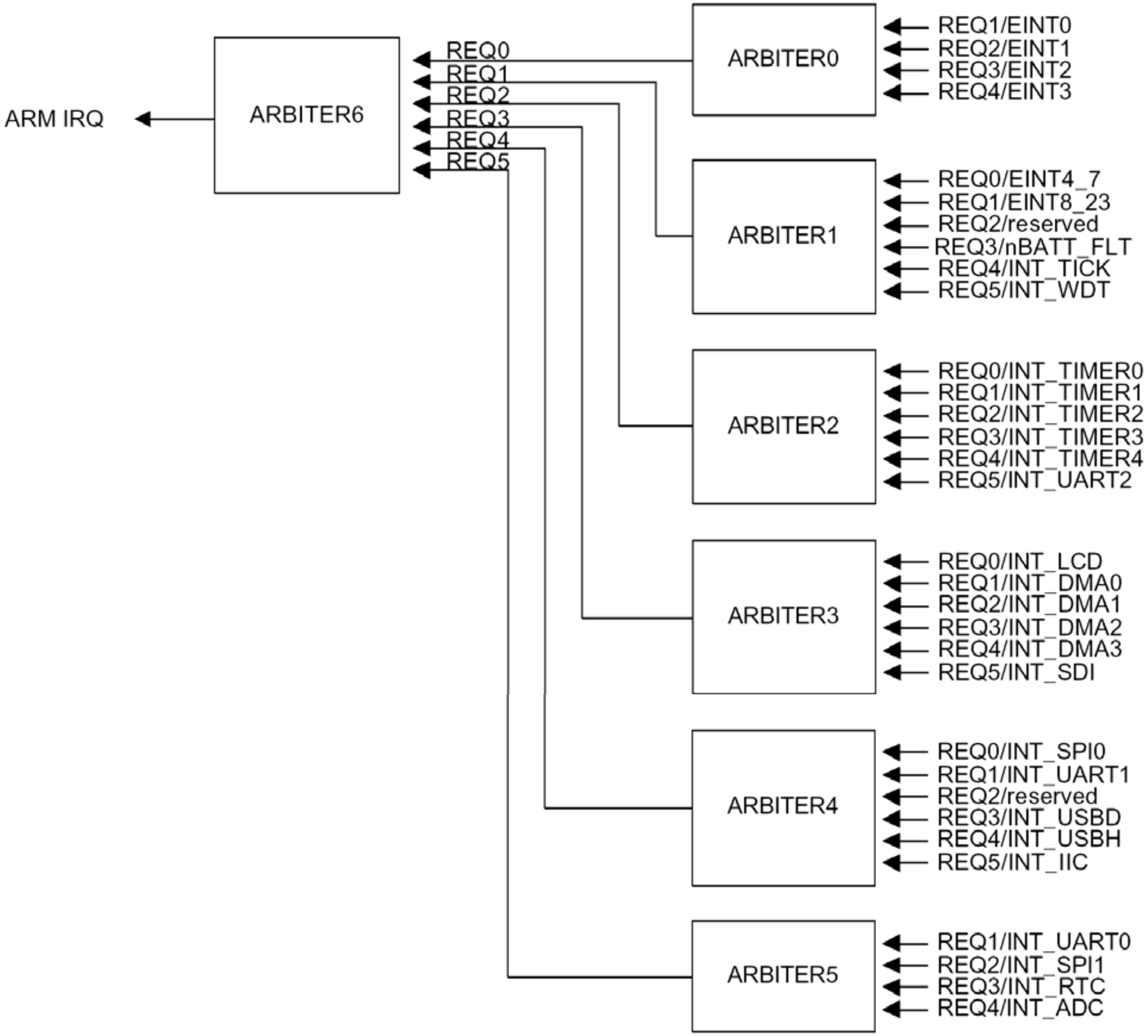
# Interrupt operation in S3C2410



**PRIORITY REGISTER (PRIORITY)**

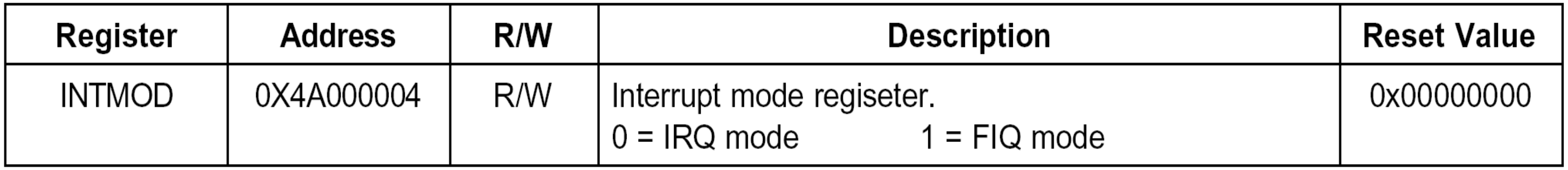
| Register | Address | R/W | Description | Reset Value |
|---|---|---|---|---|
| PRIORITY | 0x4A00000C | R/W | IRQ priority control register | 0x7F |

# Interrupt priority logic



# Interrupt mode

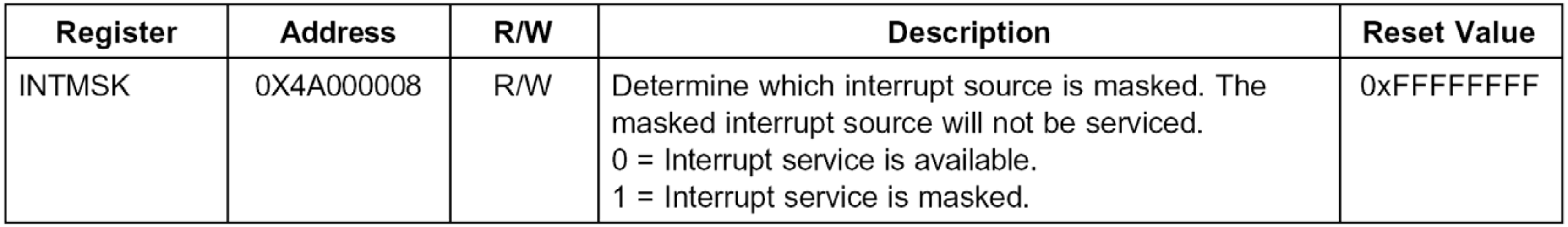
- F-bit and I-bit of program status register (PSR)
  - If F-bit is set to 1, CPU does not accept fast interrupt request (FIQ) from interrupt controller
  - If I-bit is set to 1, CPU does not accept interrupt request (IRQ) from interrupt controller
- Interrupt mode set by interrupt mode register (INTMOD)



| Register | Address | R/W | Description | Reset Value |
|---|---|---|---|---|
| INTMOD | 0X4A000004 | R/W | Interrupt mode regiseter.<br>0 = IRQ mode 1 = FIQ mode | 0x00000000 |

# Interrupt mask register (INTMSK)

- An interrupt is disabled if corresponding mask bit of INTMSK is set to 1. Otherwise, the interrupt will be serviced normally.
- If corresponding mask bit is 1 and the interrupt is generated, the source pending bit will be set



| Register | Address | R/W | Description | Reset Value |
|---|---|---|---|---|
| INTMSK | 0X4A000008 | R/W | Determine which interrupt source is masked. The masked interrupt source will not be serviced.<br>0 = Interrupt service is available.<br>1 = Interrupt service is masked. | 0xFFFFFFFF |

# Interrupt pending registers

- S3C2410 has two interrupt pending resisters
  - Source pending register (SRCPND)
  - Interrupt pending register (INTPND)
- When interrupt sources request interrupt service
  - Corresponding bits of SRCPND are set to 1
  - Only one bit of INTPND is set to 1 after arbitration procedure.
- If interrupts are masked, corresponding bits of SRCPND are set to 1. This does not cause the bit of INTPND changed.
- Service routine must clear pending condition by writing a 1 to corresponding bit in SRCPND first and then clear pending condition in INTPND

# Interrupt pending registers

- Source pending register

| Register | Address | R/W | Description | Reset Value |
|---|---|---|---|---|
| SRCPND | 0X4A000000 | R/W | Indicate the interrupt request status.<br>0 = The interrupt has not been requested.<br>1 = The interrupt source has asserted the interrupt request. | 0x00000000 |

- Interrupt pending register

| Register | Address | R/W | Description | Reset Value |
|---|---|---|---|---|
| INTPND | 0X4A000010 | R/W | Indicate the interrupt request status.<br>0 = The interrupt has not been requested.<br>1 = The interrupt source has asserted the interrupt request. | 0x00000000 |

# Reference

- A.N. Sloss, D. Symes, C. Wright, “ARM System Developer’s Guide”, Elsevier Inc., 2004.
- D.Seal, ARM Architecture Reference Manual, 2nd Edition, Addison-Wesley, 2001.
- S3C2410 User Mannual, Samsung Semiconductor