Embedded Systems

Interrupts, interrupt controller, vector table

Lesson 08

Francesco Menichelli

francesco.menichelli@uniroma1.it

Interrupts

- Practical problem: how to inform the CPU (software) of external events
  - Software check the device (polling)
    - Software read device status registers periodically
    - CPU busy
  - The device signal the event (interrupt)
    - Requires hardware support
    - CPU is free until events are risen
- Programmable Interrupt Controller
  - combine several sources of interrupt onto one or more CPU lines
  - allow priority levels to be assigned

Interrupts signals

- From the physical point of view they are electrical signal
- Each interrupt has a dedicated physical line
  - If physical lines are shared between interrupts, software has to check which device caused the request (requires time)
- Electrical events on a line that can be interpreted as interrupts
  - Levels (i.e 0 or 1)
  - Edges (i.e. rising or falling)
- An interrupt based on signal level will be asserted continuously until the level change
  - If the level is still asserted at the end of the interrupt handler, a new interrupt will be taken
- An interrupt based on signal edge will be asserted one time
  - If a new interrupt arrives before the interrupt handler clears the interrupt, it will be lost
- Distinguishing between level or edge interrupts is important mainly for external interrupts (interrupts coming from I/O pin)

Interrupt latency

- The delay between an interrupt signal is asserted and the interrupt request is handled by software
- It is influenced by
  - CPU busy in servicing other, higher priority, interrupts
  - Interrupts momentarily disabled
  - Context saving at interrupt arrival
- Can be reduced with hardware support
  - Interrupt priority, interrupt nesting
  - Atomic operations on data
  - Automatic register file bank switch

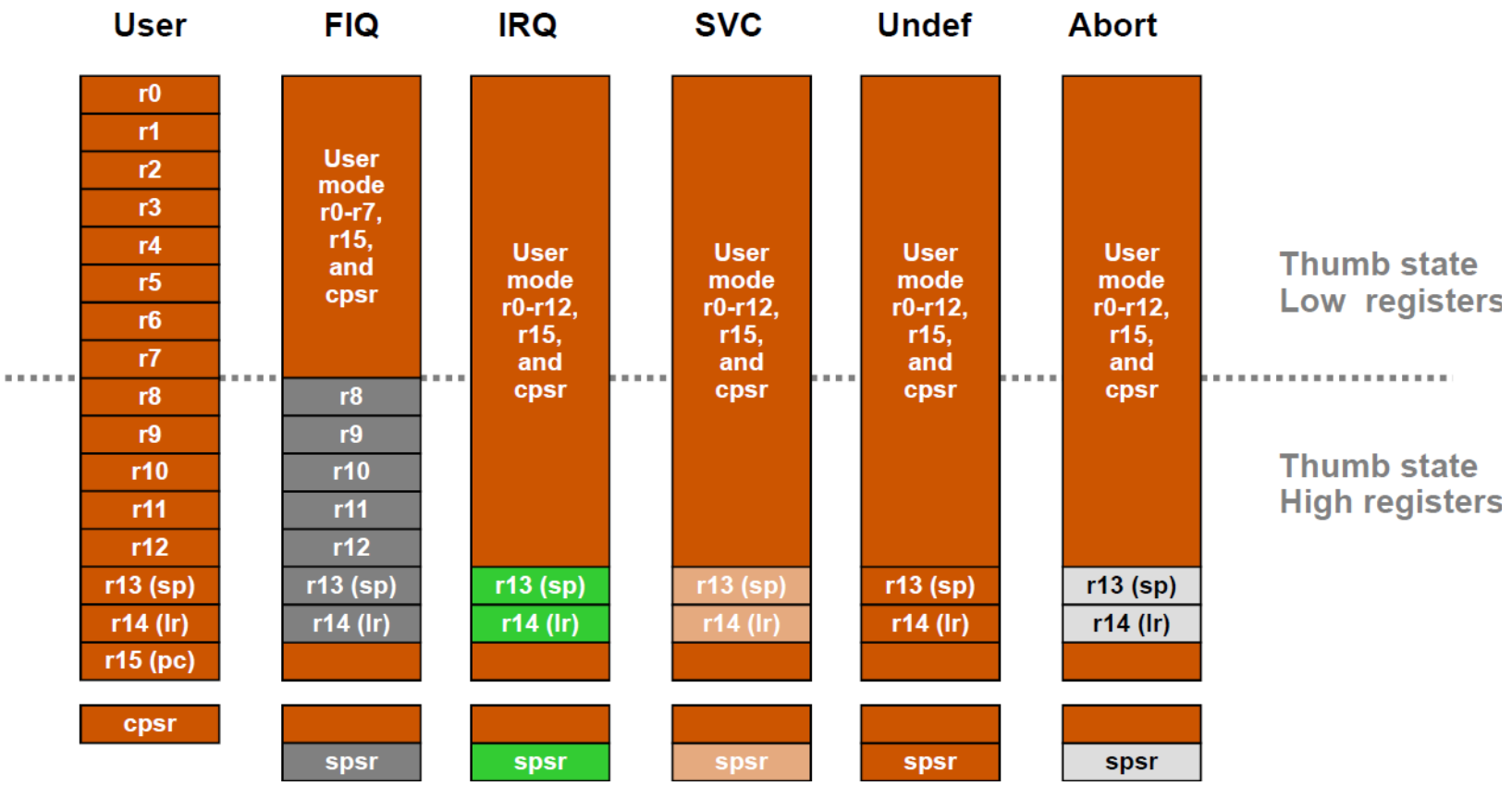
Review of ARM Exceptions

Exception	Description
Reset	Occurs when the processor reset pin is asserted. This exception is only expected to occur for signalling power-up, or for resetting as if the processor has just powered up. A soft reset can be done by branching to the reset vector (`0x0000`).
Undefined Instruction	Occurs if neither the processor, or any attached coprocessor, recognizes the currently executing instruction.
Software Interrupt (SWI)	This is a user-defined synchronous interrupt instruction. It allows a program running in User mode, for example, to request privileged operations that run in Supervisor mode, such as an RTOS function.
Prefetch Abort	Occurs when the processor attempts to execute an instruction that was not fetched, because the address was illegal[a].
Data Abort	Occurs when a data transfer instruction attempts to load or store data at an illegal address[a].
IRQ	Occurs when the processor external interrupt request pin is asserted (LOW) and the I bit in the CPSR is clear.
FIQ	Occurs when the processor external fast interrupt request pin is asserted (LOW) and the F bit in the CPSR is clear.

Review of ARM Interrupts

- Vector table
  - Reserved area of 32 bytes at the beginning of the memory map
  - One word of space for each exception type
  - Contains a Branch or Load PC instruction for the exception handler
- Exception modes and registers
  - Handling exceptions changes program from user to non-user mode
  - Each exception handler has access to its own set of registers
    - Its own r13 = stack pointer
    - Its own r14 = link register
    - Its own SPSR (Saved Program Status Register)
- Exception handlers must save (restore) other register on entry (exit)

Register Organization Summary



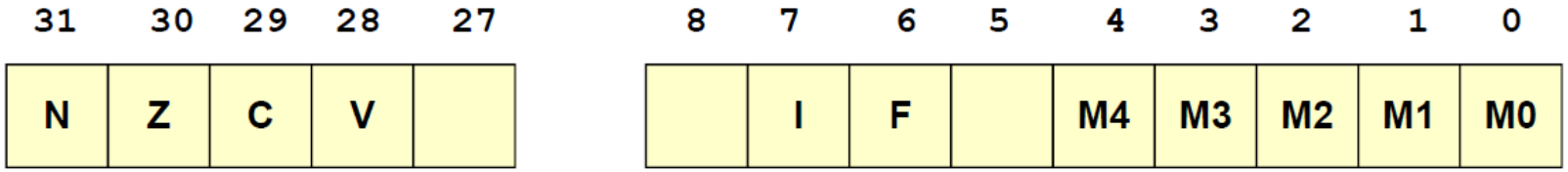
Note: System mode uses the User mode register set

What if Exceptions Happen Simultaneously?

Vector address	Exception type	Exception mode	Priority (1=high, 6=low)
0x0	Reset	Supervisor (SVC)	1
0x4	Undefined Instruction	Undef	6
0x8	Software Interrupt (SWI)	Supervisor (SVC)	6
0xC	Prefetch Abort	Abort	5
0x10	Data Abort	Abort	2
0x14	*Reserved*	*Not applicable*	*Not applicable*
0x18	Interrupt (IRQ)	Interrupt (IRQ)	4
0x1C	Fast Interrupt (FIQ)	Fast Interrupt (FIQ)	3

Enabling IRQ and FIQ

- Program Status Register



- To disable interrupts, set corresponding “F” or “I” bit to 1
- On interrupt, processor does the following
  - Switch register banks
  - Copy CPSR to SPSR_mode(saves mode, interrupt flags, etc.)
  - Change the CPSR mode bits (M[4:0])
  - Disable interrupts
  - Copy PC to R14_mode (to provide return address)
  - Set the PC to the vector address of the exception handler
- Interrupt handlers must contain code to clear the source of the interrupt

Interrupt Details

- On an IRQ interrupt, the ARM processor will ...
  - If the “I” bit in the CPSR is clear, the current instruction is completed and then the processor will
    - Save the address of the next instruction plus 4 in r14_irq
    - Save the CPSR in the SPSR_irq
    - Force the CPSR mode bits M[4:0] to 10010 (binary)
- This switches the CPU to IRQ mode and then sets the “I” flag to disable further IRQ interrupts
- On an FIQ interrupt, the processor will ...
  - If the “F” bit in the CPSR is clear and the current instruction is completed, the ARM will
    - Save the address of the next instruction plus 4 in r14_fiq
    - Force the CPSR mode bits M[4:0] to 10001 (binary)
- This switches the CPU to FIQ mode and then sets the “I” and “F” flags to disable further IRQ or FIQ interrupts

IRQ vs. FIQ

- FIQs have higher priority than IRQs
  - When multiple interrupts occur, FIQs get serviced before IRQs
  - Servicing an FIQ causes IRQs to be disabled until the FIQ handler re-enables them
    - CPSR restored from the SPSR at the end of the FIQ handler
- How are FIQs made faster?
  - They have five extra registers at their disposal, allowing them to store status between calls to the handler
  - FIQ vector is the last entry in the vector table
    - The FIQ handler can be placed directly at the vector location and run sequentially after the location
    - Cache-based systems: Vector table + FIQ handler all locked down into one block

ARM Vector Table and Interrupts

- The interrupt vector is the first instruction executed by the ARM core when getting the interrupt
  - Stored at location 0x18 (IRQ) or 0x1C (FIQ)
- Normally, a jump is performed, either with instruction:

```
B       Interrupt_Service_Routine
```

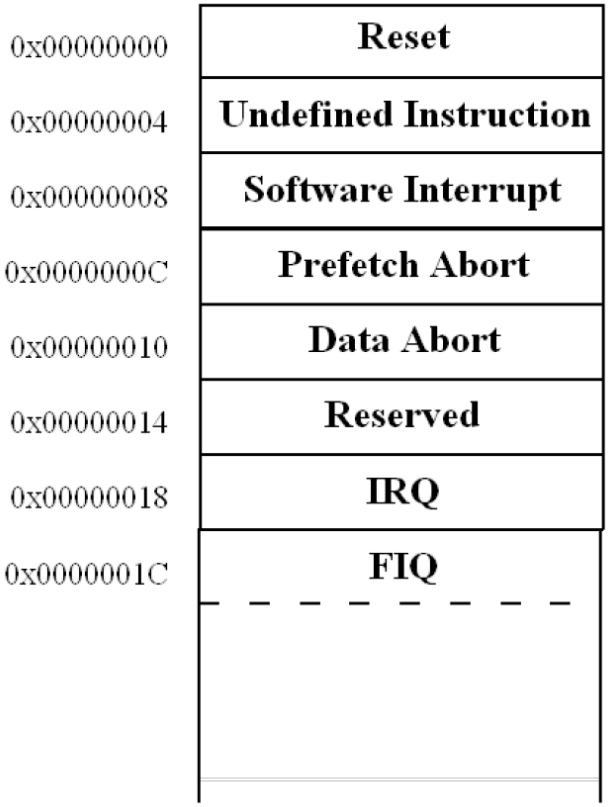
  - Up to 32Mbytes accessible

```
LDR     pc, [pc, #Interrupt_Service_Routine]
```

  - PC relative, requires an additional 32-bit word
  - Full 4GBytes address space accessible

```
Interrupt_Service_Routine
```

- The FIQ handler can be directly written



Types of Interrupts

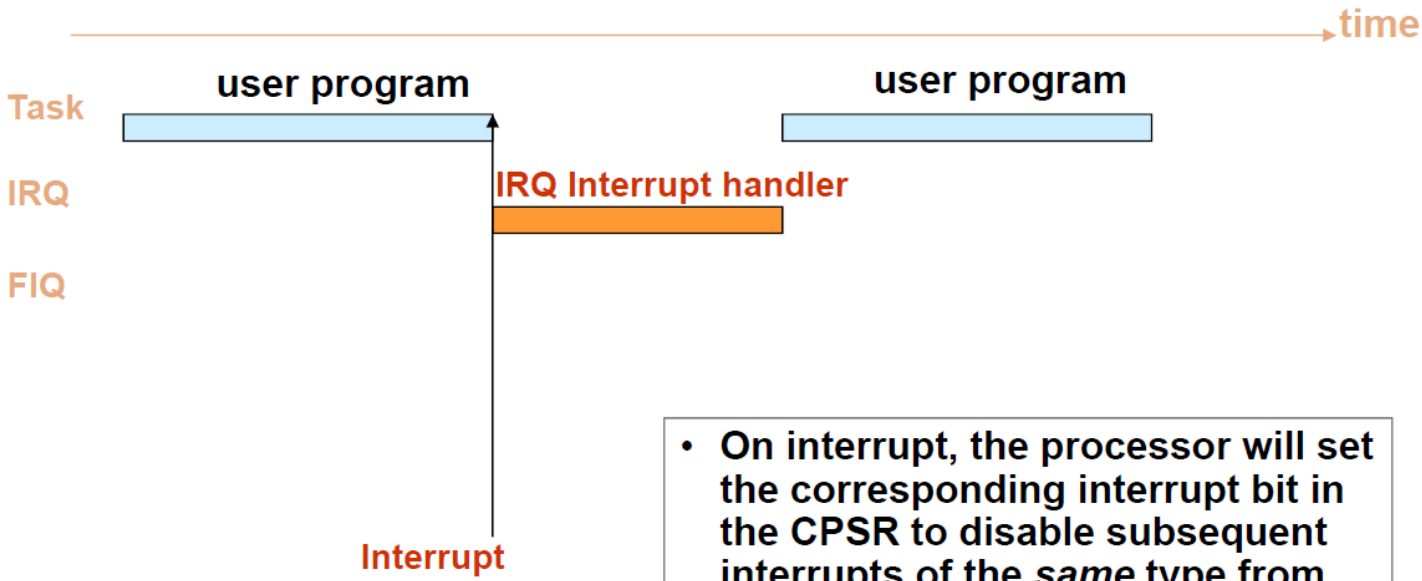
- Synchronous
  - Produced by the processor while executing instructions.
  - Issues only after finishing execution of an instruction.
  - Often called *exceptions*.
  - Example: SWI, page faults, system calls, divide by zero
- Asynchronous
  - Generated by other hardware devices.
  - Occur at arbitrary times, including while CPU is busy executing an instruction.
  - Ex: I/O, timer interrupts

Jumping to the Interrupt Handler

- Auto-vectored
  - Processor-determined address of interrupt handler based on type of interrupt
  - This is what the ARM does
- Vectored
  - Device supplies processor with address of interrupt handler
- Why the different methods?
  - If multiple devices uses the same interrupt type (IRQ vs. FIQ), in an Auto-vectored system the processor must poll each device to determine which device interrupted the processor
    - This can be time-consuming if there are a lot of devices
- In a vectored system, the processor would just take the address from the device (which dumps the interrupt vector onto a special bus).

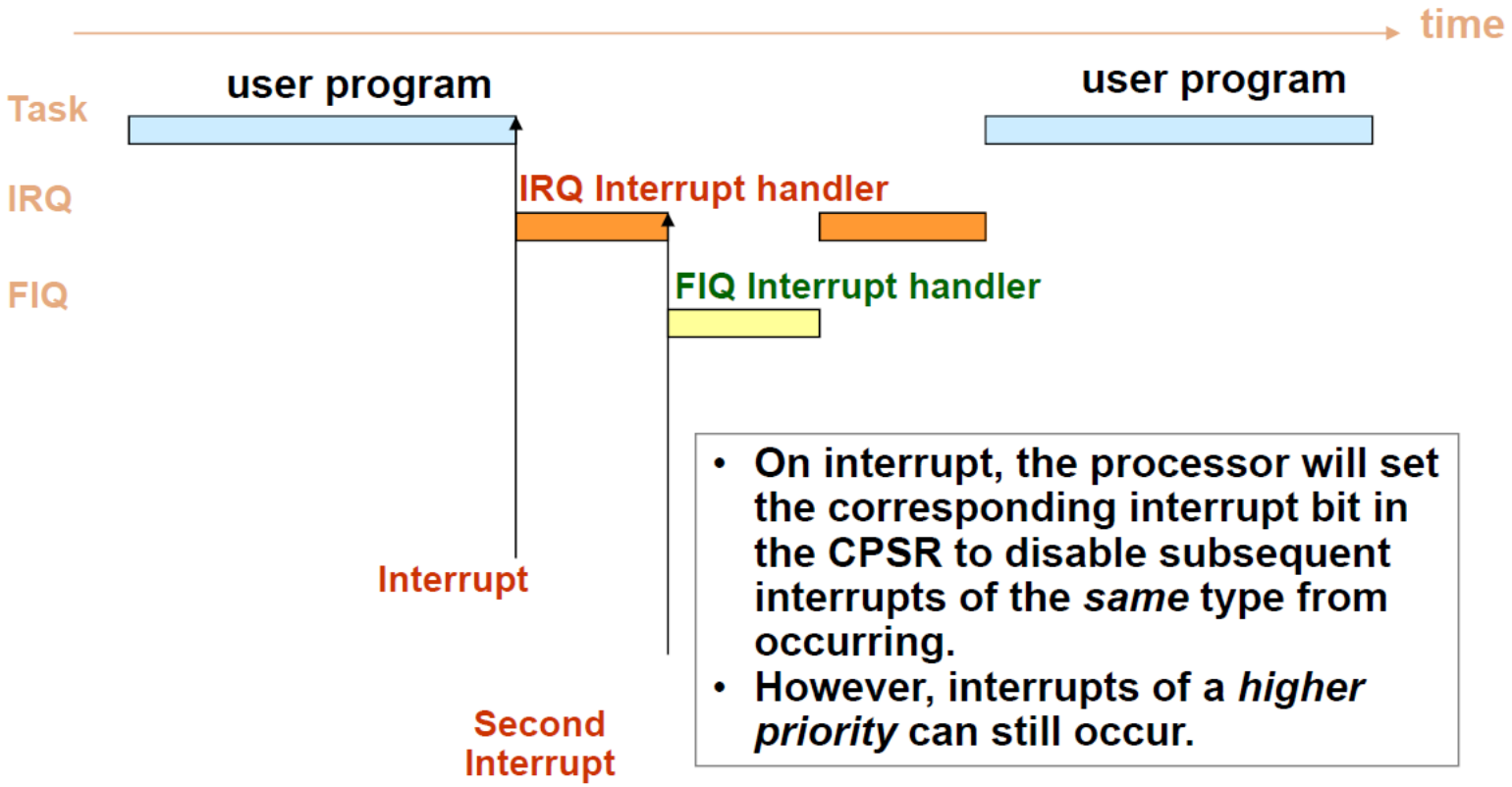
Interrupt Handlers

- When an interrupt occurs, the hardware will jump to an “interrupt handler”



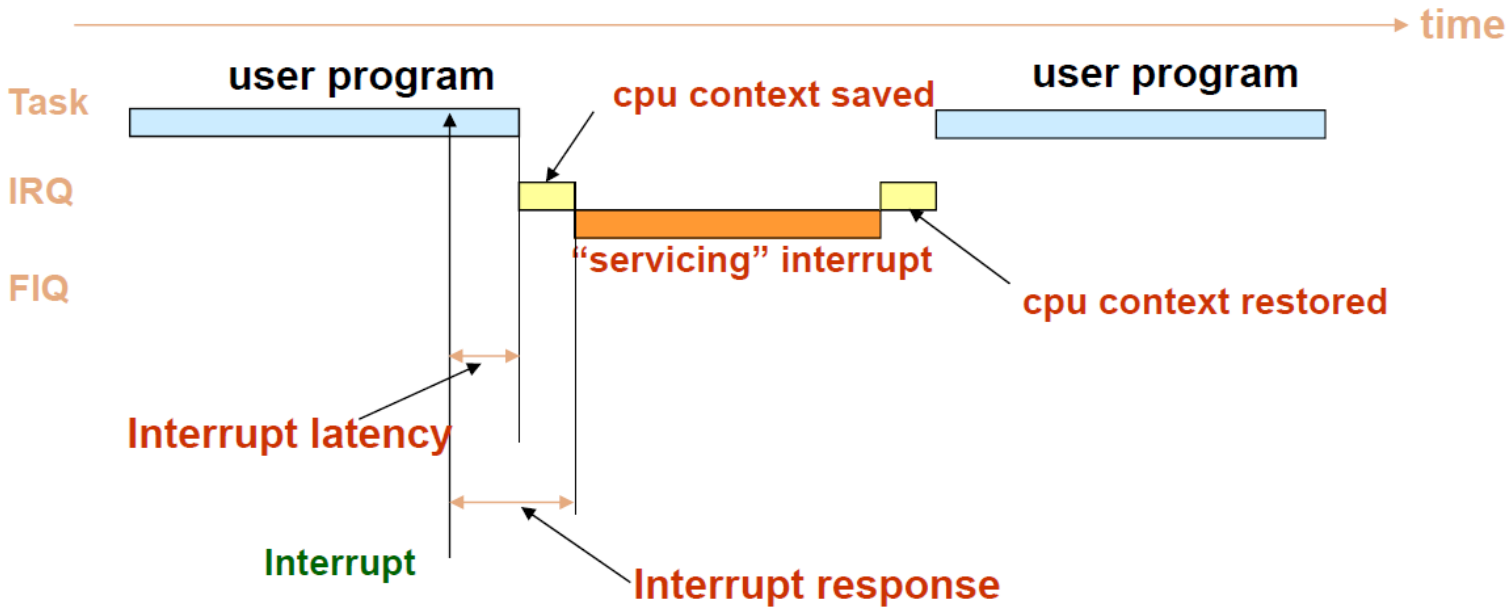
Nested/Re-entrant Interrupts

- Interrupts can occur within interrupt handlers



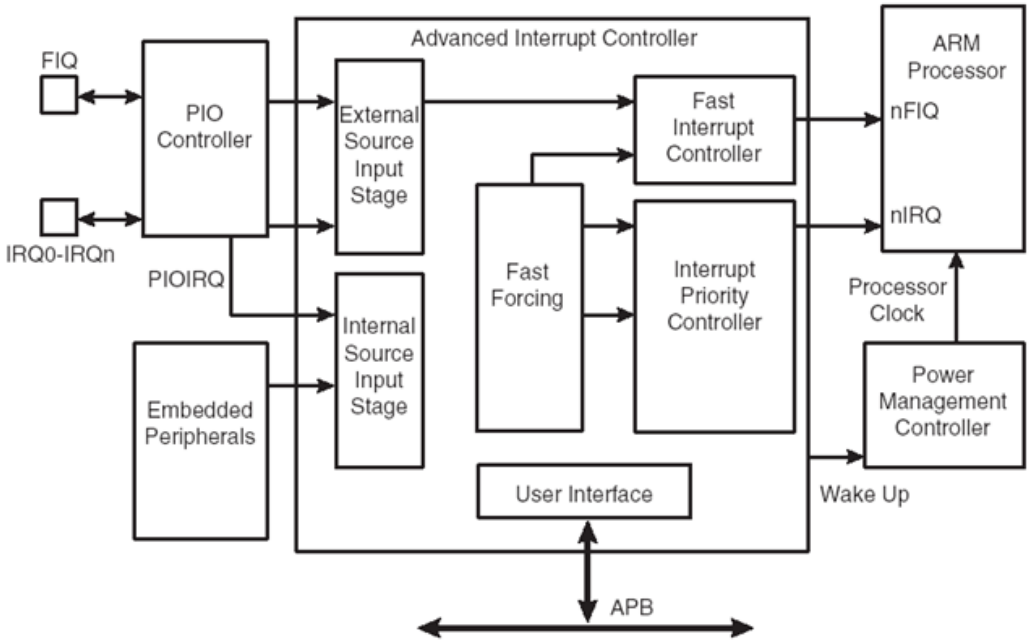
Timing of Interrupts

- Before an interrupt handler can do anything, it must save away the current program's registers (if it touches those registers)
- That's why the FIQ has lots of extra registers - to minimize CPU context saving overhead



The ARM AIC

- Advanced Interrupt Controller (AIC)
- The AIC is an additional layer allowing the handling of up to 32 interrupt sources, coming either from the internal peripherals or from the interrupt input pins



AIC Interrupt Sources

- Two kinds of interrupt sources:
  - External (IRQx and FIQ pins):
    - High or Low Level Sensitive
    - Rising or Falling Edge Triggered
  - Internal (internal peripherals):
    - High Level Sensitive (recommended)
    - Rising Edge Triggered
- The Interrupt Source 0 is always located at FIQ
- The Interrupt Source 1 is always located at System Interrupt.

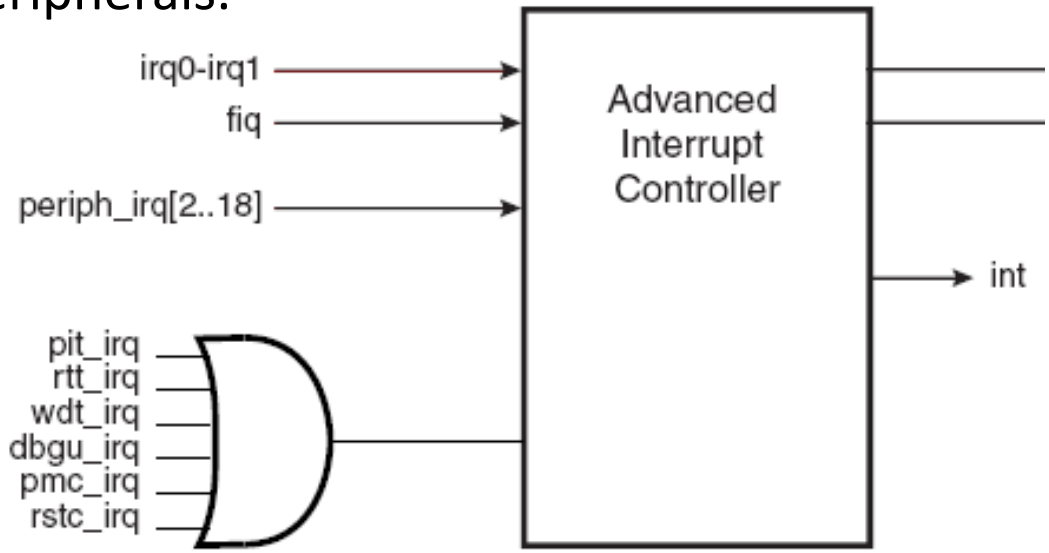
Peripheral ID	Peripheral Mnemonic
0	AIC
1	SYSC(1)
2	PIOA
3	Reserved
4	ADC(1)
5	SPI
6	US0
7	US1
8	SSC
9	TWI
10	PWMC
11	UDP
12	TC0
13	TC1
14	TC2
15 - 29	Reserved
30	AIC
31	AIC

FIQ

System Interrupt

What's the System Interrupt ?

- The wired-OR of all the sources coming from the System Controller's peripherals, including:
  - System Timer, Real Time Clock,
  - Power Management Controller
  - Memory Controller.
- When a system interrupt occurs, the service routine must distinguish the cause of the interrupt. This is performed by reading successively the status registers of the above mentioned system peripherals.

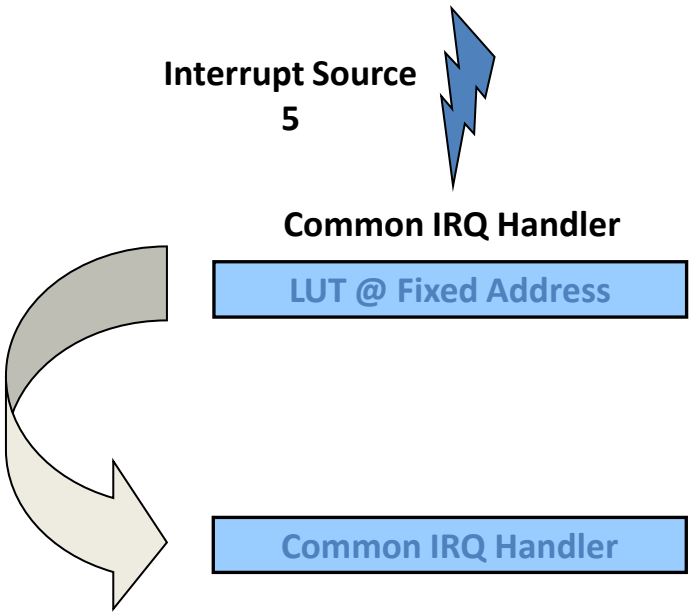


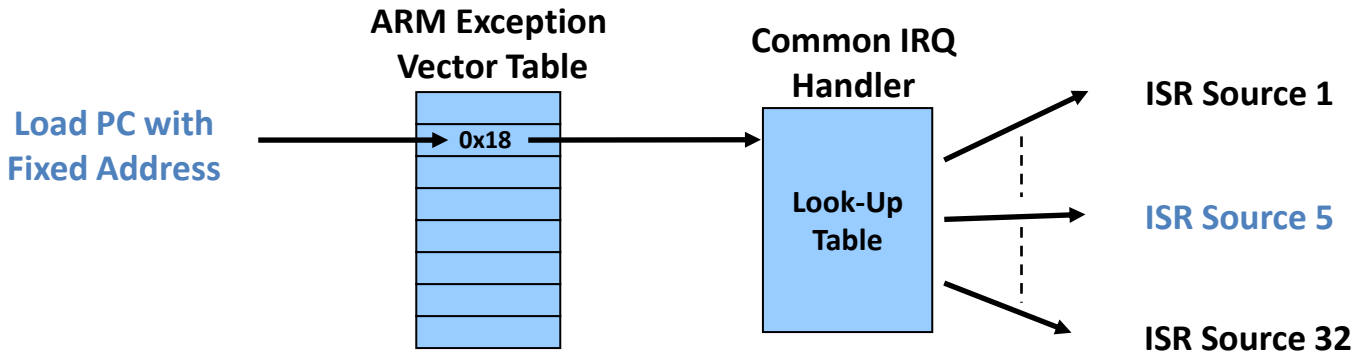
How to manage the System Interrupt ?

- By reading the Status Registers of all the system peripherals and check which interrupt(s) is/are pending:

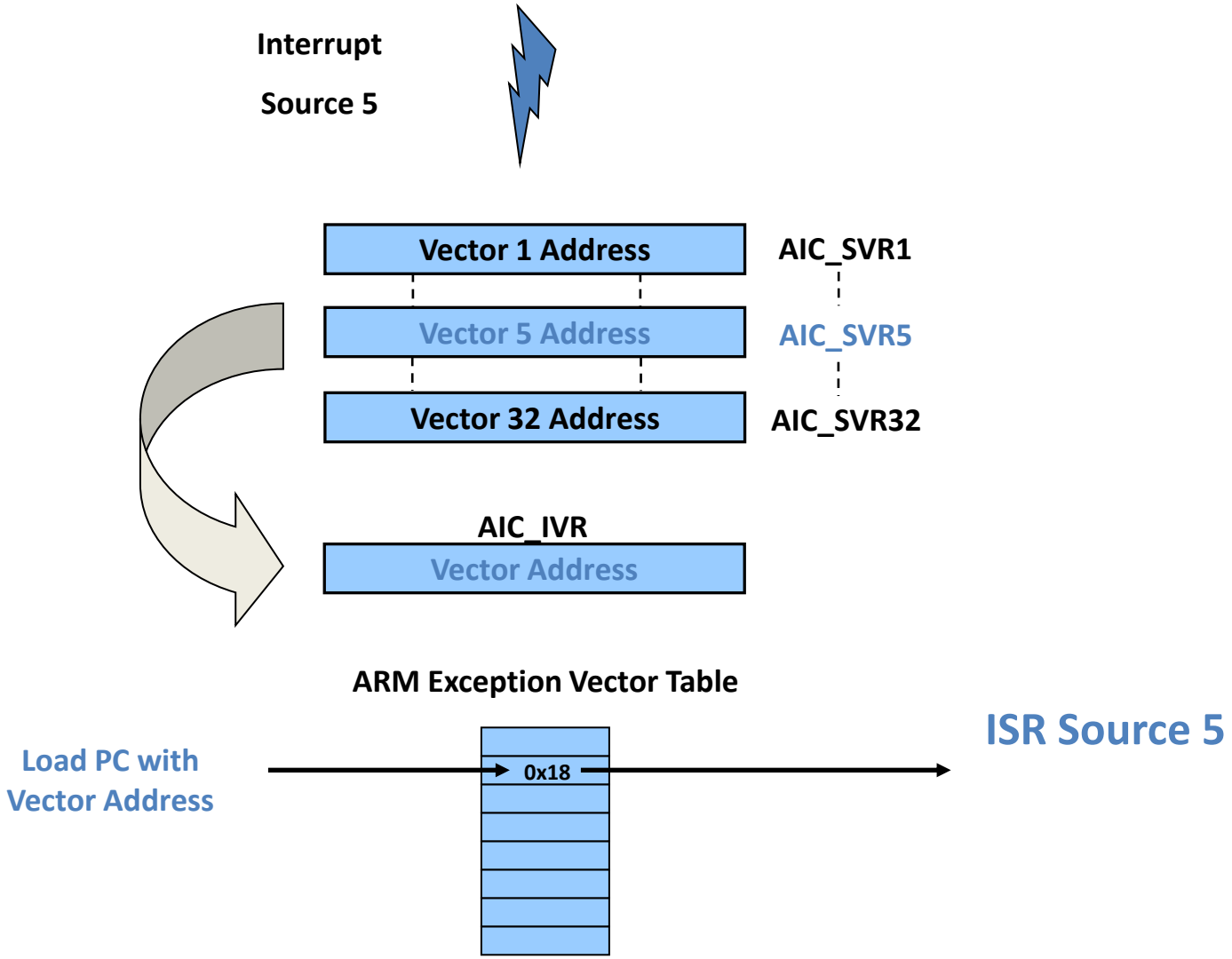
```
Void ISR_SystemInterrupt (void){

    // PIT Timer Interrupt Processing
    status = PIT_SR & PIT_IMR;
    if (status  != 0x0){
      …
    }

    // DBGU Interrupt Processing
    status = DBGU_SR & DBGU_IMR;
    if (status  != 0x0){
      …
    }
}
```

Shared interrupt requests



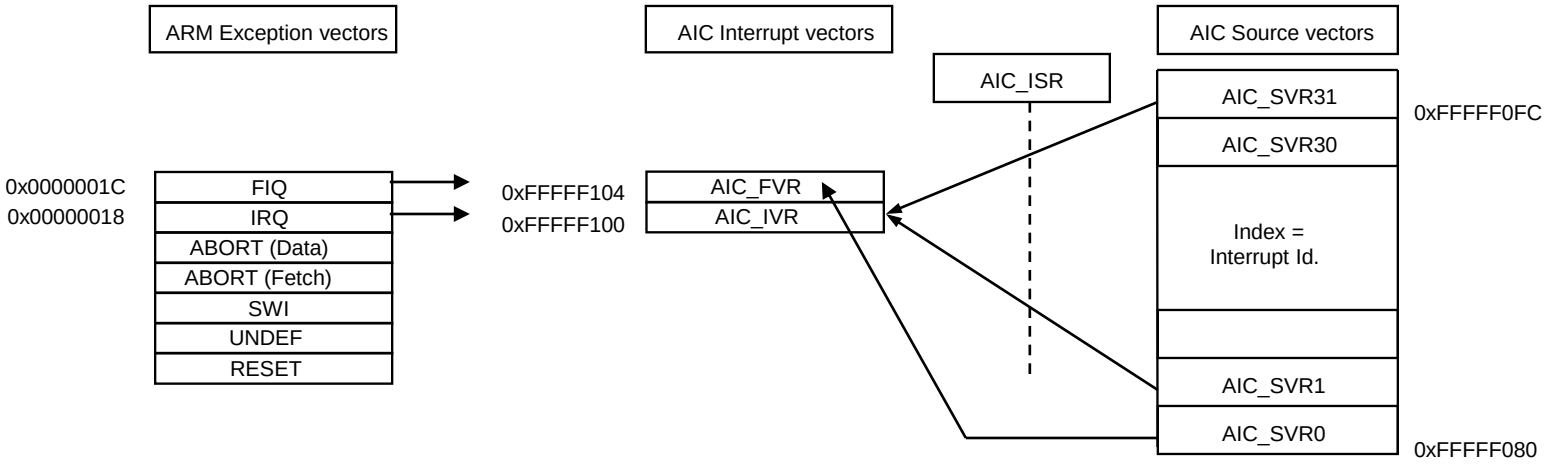


Dedicated interrupt requests



AIC Automatic Vectoring Benefit

- All sources are individually and dynamically vectored
- Management for FIQ and IRQ are independent:
  - AIC_IVR returns the handler address of the current IRQ
  - AIC_FVR returns the handler address of the current FIQ
- Handler addresses are saved in their corresponding Source Vector Register (AIC_SVRx)



AIC IRQ Priority Management

- The ARM Core IRQ line is controlled by an 8-level priority encoder
  - Each source has a programmable priority level of 7 to 0 (AIC_SMR).
  - Level 7 is the highest priority.

- The AIC manages the IRQ prioritization by using an hardware internal stack on which the current interrupt level is:
  - Automatically pushed when AIC_IVR is read
  - Automatically popped when AIC_EOICR is written

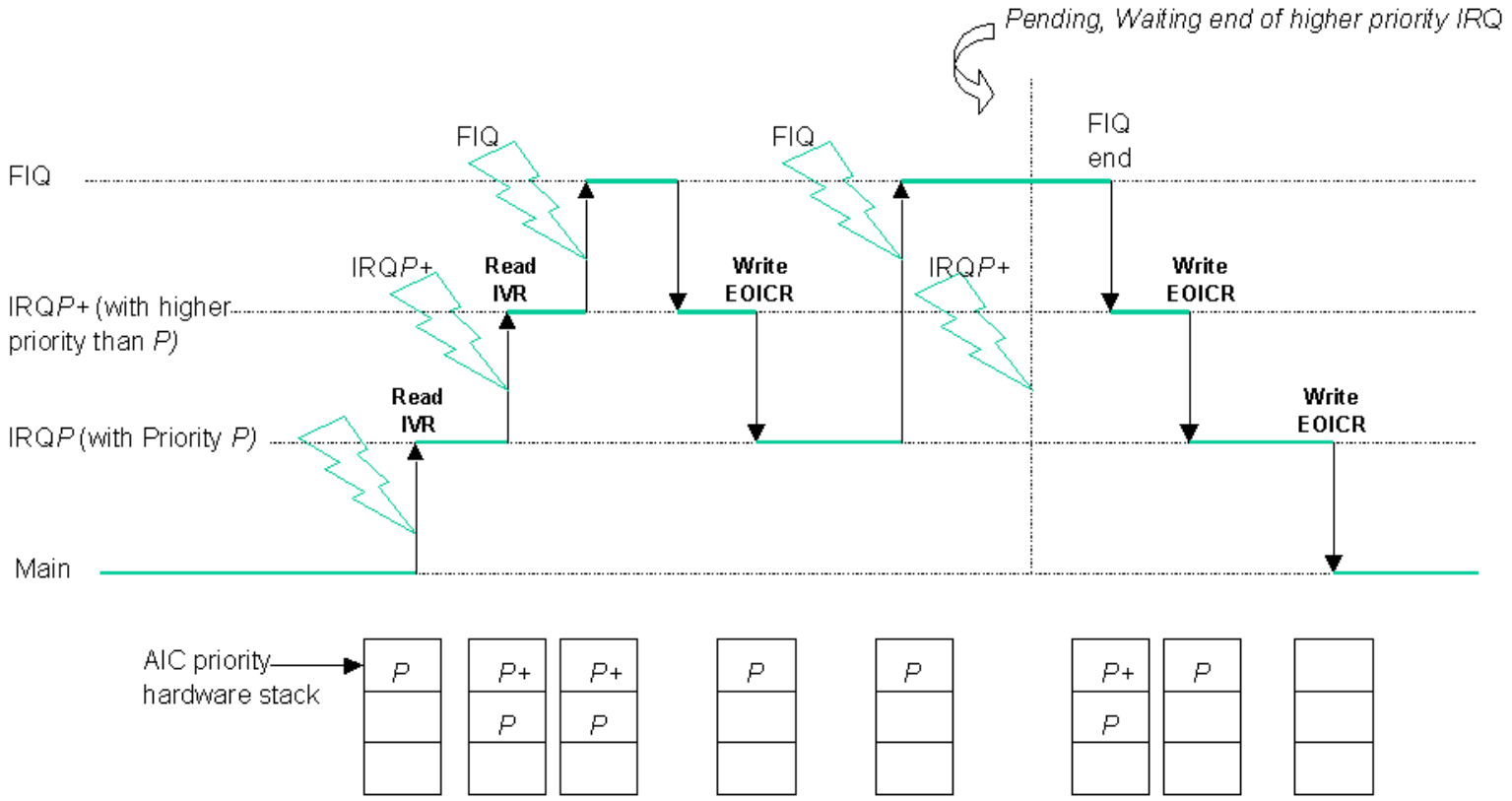
- No internal stack needed for FIQ (no prioritization)

AIC Interrupt Handling

- When an interrupt source is detected, the AIC asserts the ARM Core IRQ line:
  - The ARM Core is interrupted (automatic branch on ARM Vector 0x18)
  - The ARM Core enters its IRQ mode (IRQ are automatically disabled at core level)
  - The application must read the Interrupt Vector register (AIC_IVR)
- By reading the AIC_IVR register, the AIC:
  - De-asserts the ARM Core IRQ line
  - Determines the highest level pending and enabled interrupt source
  - Pushes the level of the current interrupt in the AIC hardware stack
  - Clears the interrupt if it's configured as edge triggered
  - Returns the vector corresponding to the current interrupt
- The ISR can re-enable as soon as possible the interrupt at core level (nested interrupts)
  - A higher level interrupt can occur and restarts this sequence
  - The ISR Exit code must disable the interrupt at core level
- Finally, the application must write the End Of Interrupt Command Register (EOICR) in the ISR Exit code
  - AIC pops from its hardware stack the current level

AIC Interrupt Handling Example

- IRQ *P* is interrupted by IRQ *P+* which is interrupted by FIQ (nested interrupts)



The True Interrupt latency

- Interrupt latency in a system might be affected by:
  - AIC External/Internal Interrupt latency (less than 5 cycles)
  - Return from Core Idle mode (max. 2 cycles)

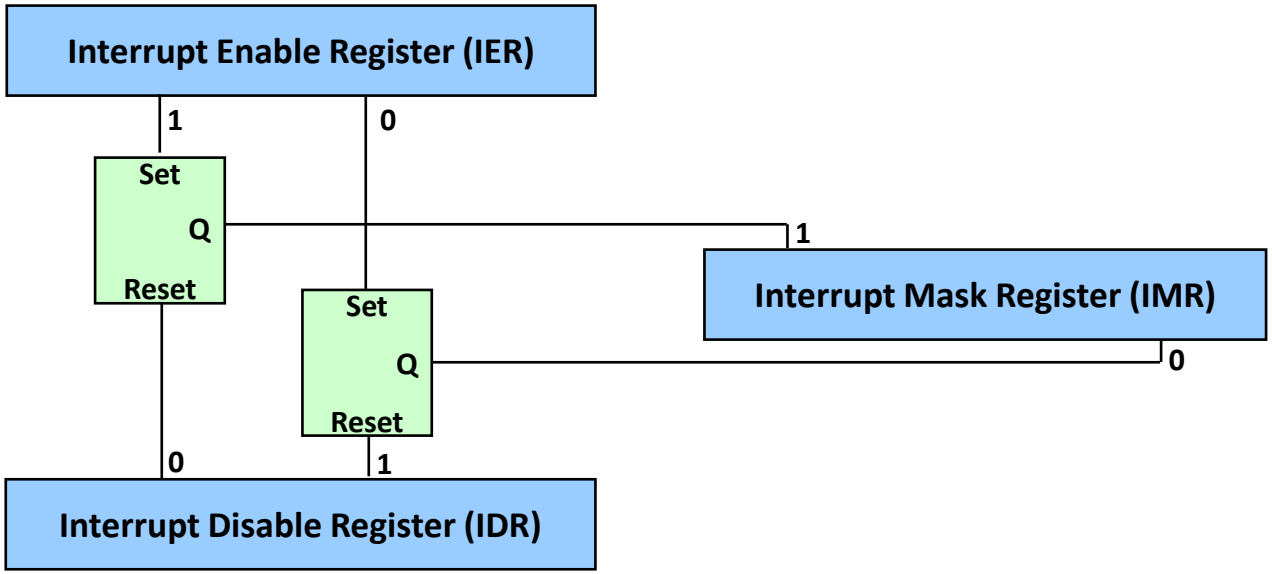
  - Interrupt Disabling
    - several hundreds/thousands of cycles depending on the tasks required for the interrupt to be masked

    That's why it is so important to get Atomic Interrupt
  - Other interrupts handling
    - might be up to 100 cycles depending on the interrupt management policy

    Priority and Interrupt Nesting can improve a lot

AT91 Single-cycle Bit Manipulation

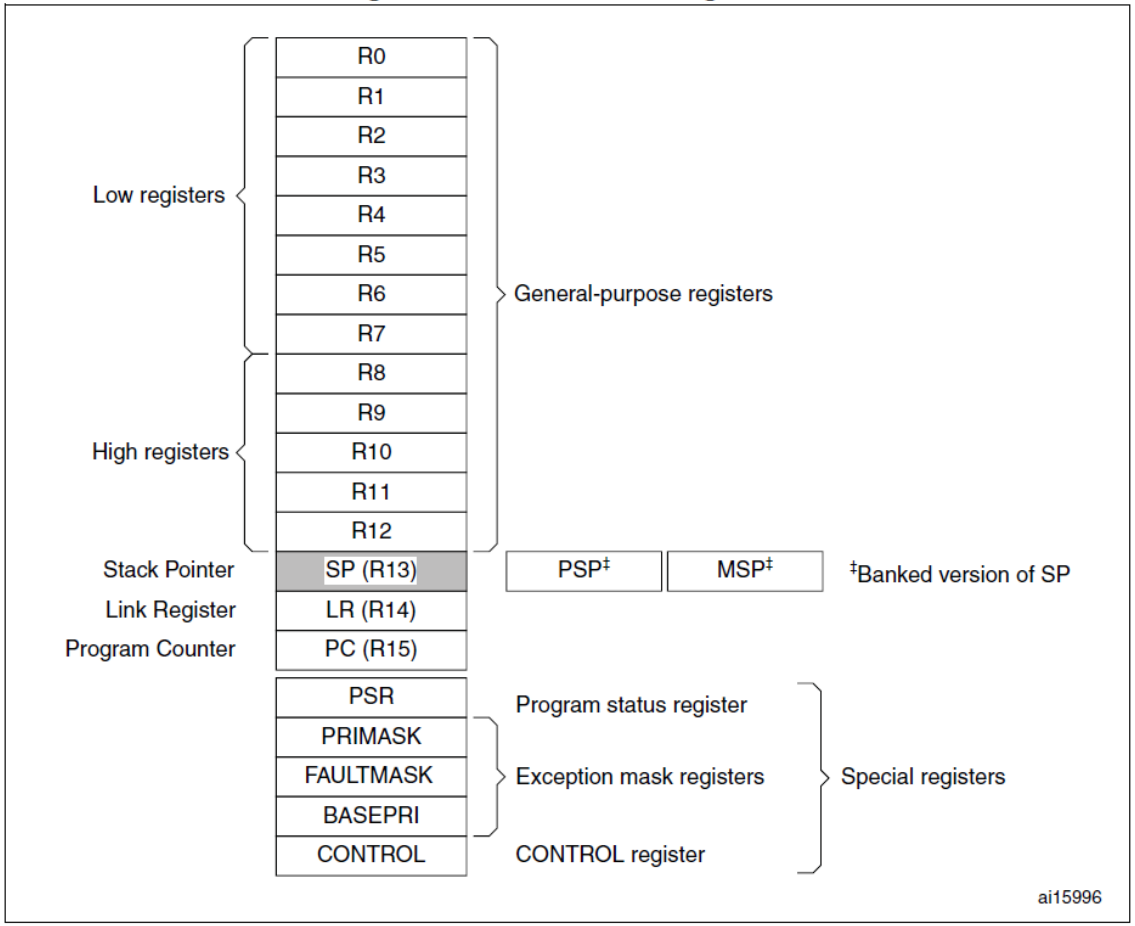
- ARM7TDMI processors do not provide any bit manipulation
  - Requires read-modify-write sequence of operations
- AT91SAM series feature atomic bit set/reset facility for all peripheral control registers
- Enabling or Disabling an interrupt is performed with only one instruction (store STR), thus is un-interruptible.



ARMv7 (Cortex) architecture

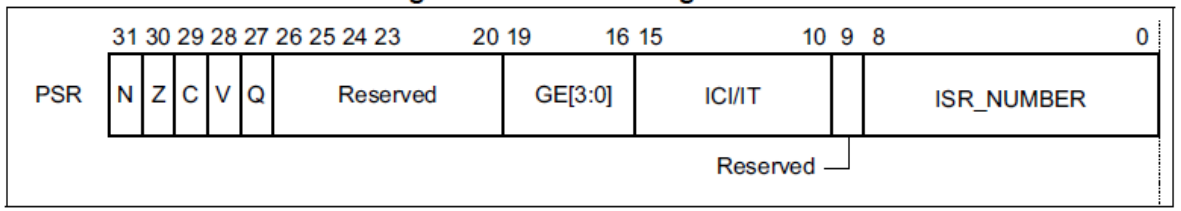
- CPU mode have been suppressed
- Banked registers have been suppressed except SP
  - Main Stack Pointer (MSP)
  - Process Stack Pointer (PSP)
- PSR has a new format

Figure 2. Processor core registers



ARMv7 status register

Figure 4. PSR bit assignments



Bits	Description
Bit 31	**N:** Negative or less than flag: 0: Operation result was positive, zero, greater than, or equal 1: Operation result was negative or less than.
Bit 30	**Z:** Zero flag: 0: Operation result was not zero 1: Operation result was zero.
Bit 29	**C:** Carry or borrow flag: 0: Add operation did not result in a carry bit or subtract operation resulted in a borrow bit 1: Add operation resulted in a carry bit or subtract operation did not result in a borrow bit.
Bit 28	**V:** Overflow flag: 0: Operation did not result in an overflow 1: Operation resulted in an overflow.
Bit 27	**Q:** DSP overflow and saturation flag: Sticky saturation flag. 0: Indicates that saturation has not occurred since reset or since the bit was last cleared to zero 1: Indicates when an SSAT or USAT instruction results in saturation, or indicates a DSP overflow. This bit is cleared to zero by software using an MRS instruction.
Bits 26:20	Reserved.
Bits 19:16	**GE[3:0]:** Greater than or Equal flags. See *SEL on page 104* for more information.

Bits 26:25, 15:10	**ICI**: Interruptible-continuable instruction bits, see *Interruptible-continuable instructions on page 22*.
Bits 26:25, 15:10	**IT**: Indicates the execution state bits of the IT instruction, see *IT on page 144*.

Bits 8:0	**ISR_NUMBER**: This is the number of the current exception: 0: Thread mode 1: Reserved 2: NMI 3: Hard fault 4: Memory management fault 5: Bus fault 6: Usage fault 7: Reserved .... 10: Reserved 11: SVCall 12: Reserved for Debug 13: Reserved 14: PendSV 15: SysTick 16: IRQ0(1) .... 83: IRQ81(1) see *Exception types on page 36* for more information.

ARMv7 vector table

- **Reset** : Reset is invoked on power up or a warm reset.
- **NMI**: A NonMaskable Interrupt (NMI) can be signalled by a peripheral or triggered by software.
- **Hard fault**: A hard fault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism.
- **Memory management fault**: A memory management fault is an exception that occurs because of a memory protection related fault.
- **Bus fault**: A bus fault is an exception that occurs because of a memory related fault for an instruction or data memory transaction.
- **Usage fault**: undefined instruction, illegal unaligned access, etc.
- **SVCall**: A supervisor call (SVC) is an exception that is triggered by the SVC instruction. In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.
- **PendSV**: PendSV is an interrupt-driven request for system-level service.
- **SysTick**: A SysTick exception is an exception the system timer generates when it reaches zero.
- Interrupt (**IRQ**): A interrupt, or IRQ, is an exception signalled by a peripheral, or generated by a software request.

Exception number	IRQ number	Offset	Vector
255	239	0x03FC	IRQ239
.		.	.
.		.	.
.		.	.
18	2	0x004C	IRQ2
17	1	0x0048	IRQ1
16	0	0x0044	IRQ0
15	-1	0x0040	Systick
14	-2	0x003C	PendSV
13		0x0038	Reserved
12			Reserved for Debug
11	-5		SVCall
10		0x002C	Reserved
9			
8			
7			
6	-10		Usage fault
5	-11	0x0018	Bus fault
4	-12	0x0014	Memory management fault
3	-13	0x0010	Hard fault
2	-14	0x000C	NMI
1		0x0008	Reset
		0x0004	Initial SP value
		0x0000	