Embedded Controller programming for Real Time Systems: ECE-40097

Lesson 5

Vijay Kumar

Main Topics

What is Interrupt

Interrupt Request Number

Interrupt Service Routine (ISR)

Interrupt Priority

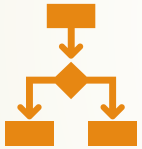
Interrupt Flow

Examples

Interrupt Or Polling

What is Interrupt

To break the continuity



Indication of external, internal or system events

As it happens

Flexible and efficient

Suitable for multiple devices/events

Allows prioritization of events

Absolutely needed for latency-sensitive events /systems

Alternate to Interrupt

It is called Polling

Easy to implement

Suitable of limited events

Less expensive

No Extra HW is needed

Less complexity in Software

And no context switching

Waste of resources in waiting

Comparison

BASIS FOR COMPARISON	INTERRUPT	POLLING
Basic	Device notifies CPU that it needs CPU attention.	CPU constantly checks device status
HW	Required HW (NVIC)	No HW is needed
Service	Interrupt handler services the Device.	CPU services the device
Indication	Interrupt-request line indicates that device needs servicing.	Device gets polled
CPU	CPU is disturbed only when a device needs servicing, which saves CPU cycles.	CPU has to wait and check whether a device needs servicing which wastes lots of CPU cycles.
Periodicity	An interrupt can occur at any time (very important)	CPU polls the devices at regular interval (defined interval).
Efficiency	Interrupt becomes inefficient (takes resources) when devices keep on interrupting the CPU repeatedly	Polling becomes inefficient when CPU rarely gets any data.
Example	Let the bell ring then open the door to check who has come.	Constantly keep on opening the door to check whether anybody has come.

What is needed for Interrupt

- Interrupt Controller
- Interrupt types
- Interrupt Number (IRQ)
- Interrupt Priority
- Interrupt Vector Table
- Interrupt service Routine (ISR)
- Communication between interrupts and tasks

Some mechanism to share data

e.g. Callback(), Shared memory, Signal, Queues

Interrupt Controller (NVIC)

- Up to 240 interrupts
- 16 exceptions
- A programmable **priority level** of 0-255 for each interrupt
- A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority
- Level and pulse detection of interrupt signals

Interrupt Controller (NVIC)

- Dynamic reprioritization of interrupts
- Grouping of priority values into group priority and sub priority fields
- Interrupt tail-chaining
  - Context switching is expensive operation

Interrupt Types

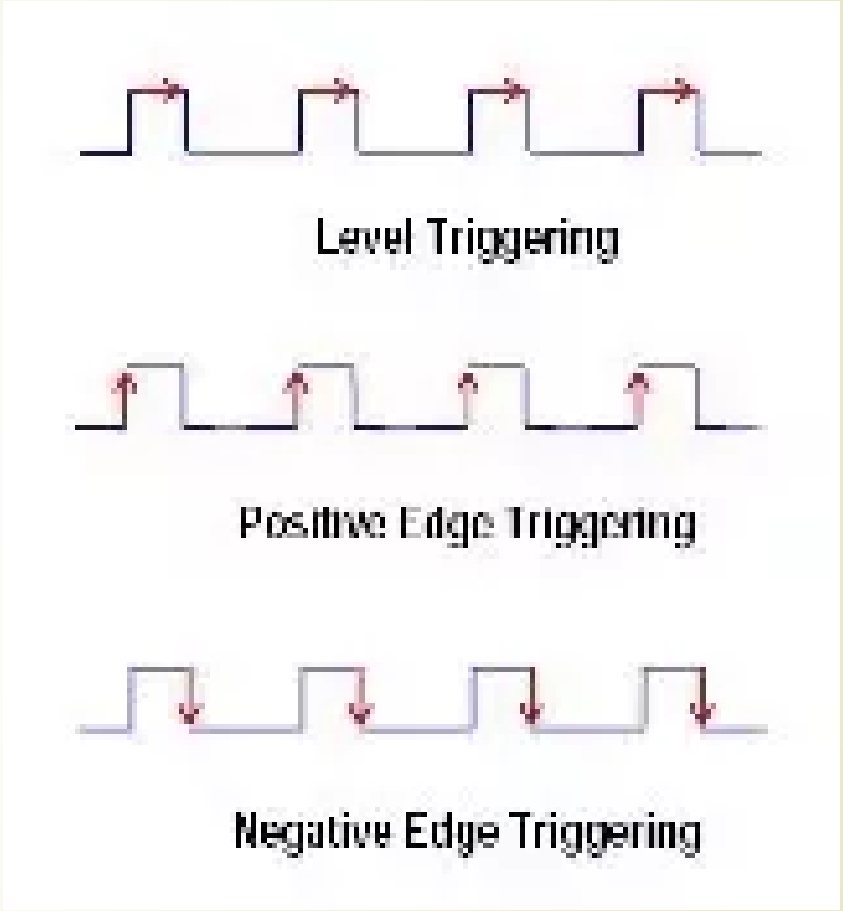
- System exceptions/interrupts
  - First 16 are negative numbers (-1 to -16)
  - Defined by ARM core
  - User doesn't have option to modify
- Peripherals interrupts
  - Remaining 240 starts at 0
  - Defined by chip manufacturers
  - Examples are USB, UART, GPIO

Interrupt Types

- Software Interrupt
  - By setting the interrupt pending registers when exception occurs
  - Software interrupt invoked by software faults is also called Trap
  - Invokes hard fault handler

Level and Edge sensitive Interrupts

- A level-sensitive interrupt is held asserted until the peripheral desserts the interrupt signal.
  - Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request.
- A pulse interrupt is an interrupt signal sampled synchronously on the rising/falling edge of the processor clock.
  - To ensure the controller detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the controller detects the pulse and latches the interrupt.

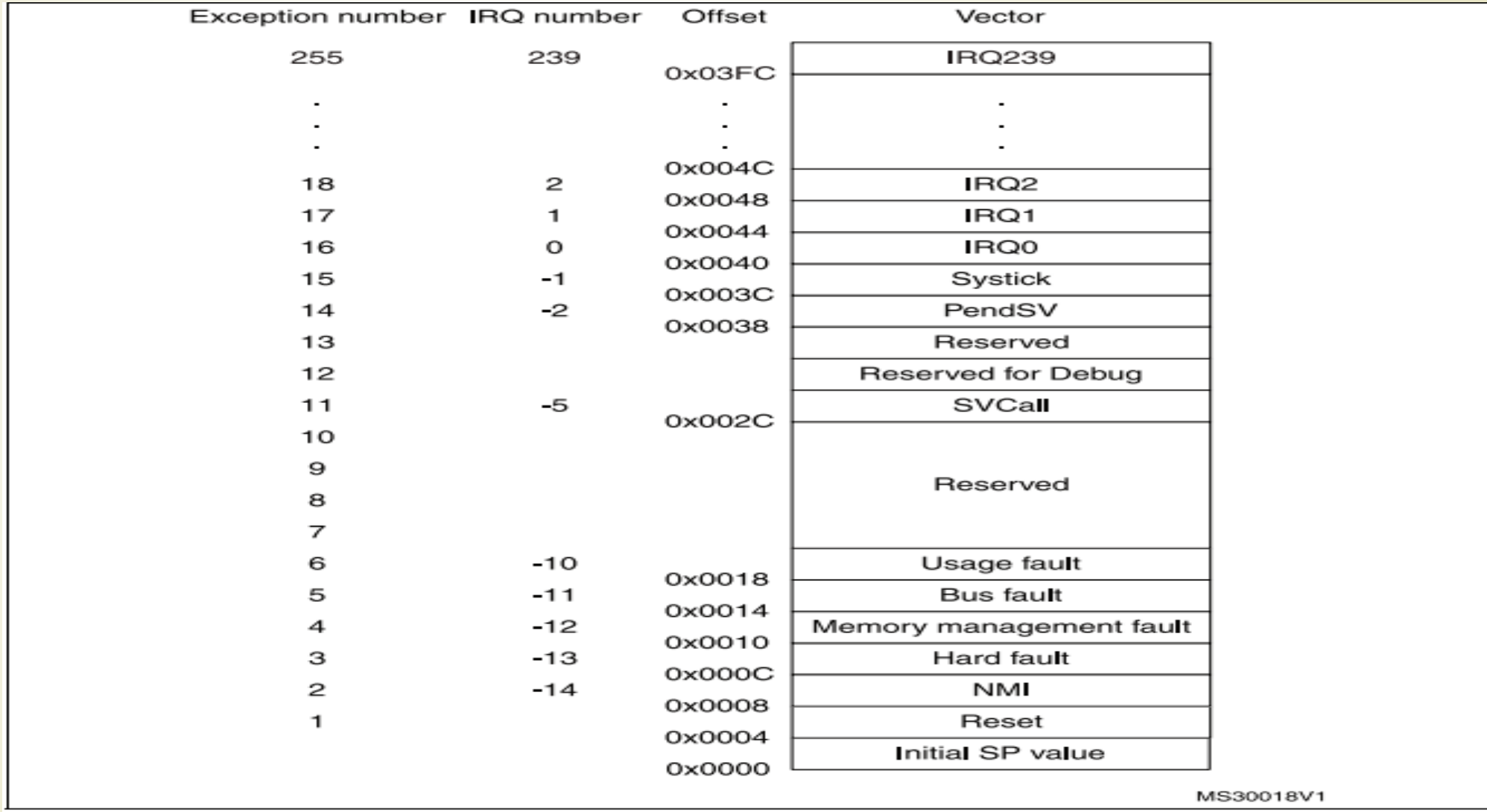


Interrupt types

- Some are configurable (priority) and have different names

Exception number[1]	IRQ number[1]	Exception type	Priority	Vector address or offset[2]	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	Hard fault	-1	0x0000000C	-
4	-12	Memory management fault	Configurable[3]	0x00000010	Synchronous
5	-11	Bus fault	Configurable[3]	0x00000014	Synchronous when precise Asynchronous when imprecise
6	-10	Usage fault	Configurable[3]	0x00000018	Synchronous
7-10	-	-	-	Reserved	-
11	-5	SVCall	Configurable[3]	0x0000002C	Synchronous
12-13	-	-	-	Reserved	-
14	-2	PendSV	Configurable[3]	0x00000038	Asynchronous
15	-1	SysTick	Configurable[3]	0x0000003C	Asynchronous
16 and above	0 and above	Interrupt (IRQ)	Configurable[4]	0x00000040 and above[5]	Asynchronous

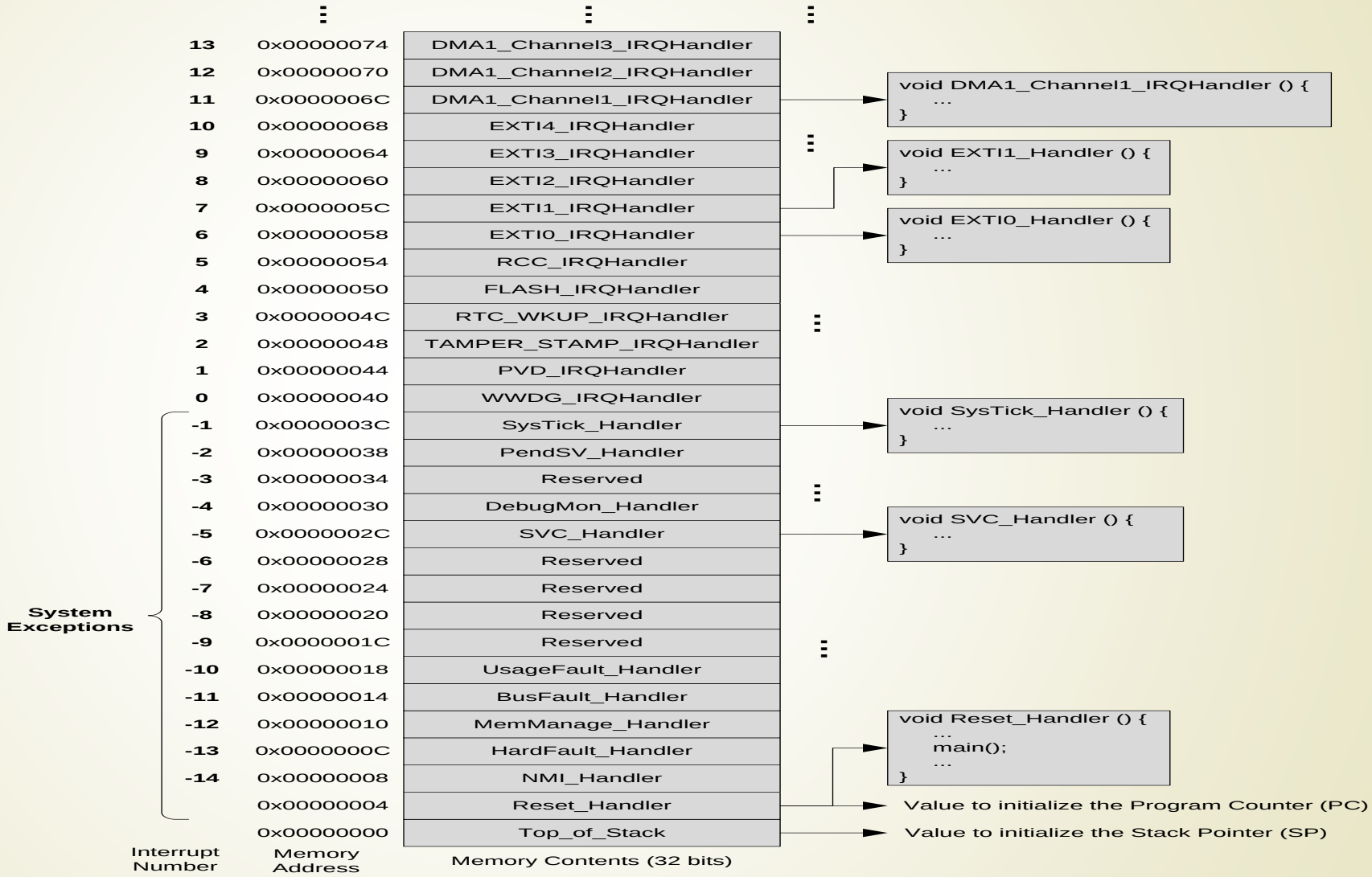
Interrupt Number



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	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

MS30018V1

Vector Table



Interrupt Program Status Register (IPSR)

- Interrupt number is stored in IPSR

Bits	Description
Bits 31:9	Reserved
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) .... 255: IRQ240(1)

Example

- Base address
  - 0x00000004
- Example of SysTick_Handler
  - Number in IPSR = 15
  - Address of ISR = base address + (4 * 14) = 0x0000003C
- Vector table could be relocated or re-mapped to different regions
  - To allow booting from other regions
  - E.g. from on-chip flash memory

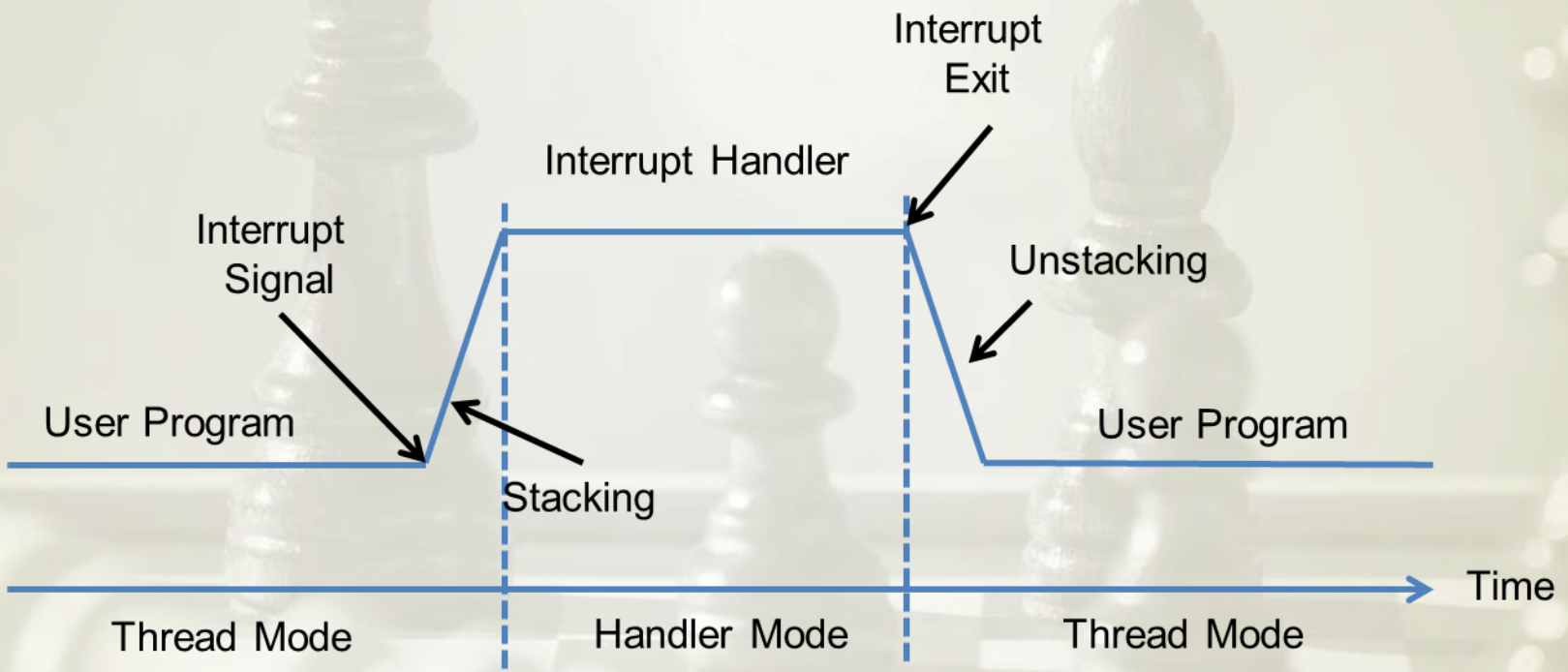
Processor Mode

- Unprivileged:
  - Unprivileged software executes at the unprivileged level and
    - Has limited access to the MSR and MRS instructions, and cannot use the CPS instruction.
    - Cannot access the system timer, NVIC, or system control block
    - Might have restricted access to memory or peripherals
    - Must use the SVC instruction to make a supervisor call to transfer control to privileged software.
- Privileged
  - Privileged software executes at the privileged level and can use all the instructions and has access to all resources
  - Can write to the CONTROL register to change the privilege level for software execution

Stacking and Unstacking

- Before Interrupt handler starts
  - Processor pushes registers into main stack
- After the handler exits
  - Processor pops all the registers
- Not applicable for tail-chained or late-arriving exceptions
  - Processor leverages the existing push
  - Speed up the process

Pictorial View



Priority

- Each interrupt has priority
- Lower the number, higher the priority
  - Could be different for other processor
- Need to prioritize the multiple interrupts
  - To process high priority interrupt first
- Multiple exceptions with same priority
  - Lowest exception takes precedence
- If executing the exception handler
  - Gets preempted with high priority exceptions
  - But not with same priority or lower

Priority Grouping

- To increase the priority controls
- Interrupt priority register has two fields
  - Group priority
  - Sub priority
- Only the group priority determines preemption of interrupt exceptions.
  - When the processor is executing an interrupt exception handler, another interrupt with the same group priority as the interrupt being handled does not preempt the handler
- If multiple pending interrupts have the same group priority
  - Sub priority field determines the order in which they are processed. If multiple pending interrupts have the same group priority and sub priority, the interrupt with the lowest IRQ number is processed first.

Example

	Interrupt priority level value, PRI_N[7:0]			Number of	
PRIGROUP	Binary point[a]	Group priority bits	Subpriority bits	Group priorities	Subpriorities
0b000	bxxxxxxx.y	[7:1]	[0]	128	2
0b001	bxxxxxx.yy	[7:2]	[1:0]	64	4
0b010	bxxxxx.yyy	[7:3]	[2:0]	32	8
0b011	bxxxx.yyyy	[7:4]	[3:0]	16	16
0b100	bxxx.yyyyy	[7:5]	[4:0]	8	32
0b101	bxx.yyyyyy	[7:6]	[5:0]	4	64
0b110	bx.yyyyyyy	[7]	[6:0]	2	128
0b111	b.yyyyyyyy	None	[7:0]	1	256

Registers

Address	Name	Type	Required privilege	Reset value
0xE000E100-0xE000E11F	NVIC_ISER0-NVIC_ISER7	RW	Privileged	0x00000000
0XE000E180-0xE000E19F	NVIC_ICER0-NVIC_ICER7	RW	Privileged	0x00000000
0XE000E200-0xE000E21F	NVIC_ISPR0-NVIC_ISPR7	RW	Privileged	0x00000000
0XE000E280-0xE000E29F	NVIC_ICPR0-NVIC_ICPR7	RW	Privileged	0x00000000
0xE000E300-0xE000E31F	NVIC_IABR0-NVIC_IABR7	RW	Privileged	0x00000000
0xE000E400-0xE000E4EF	NVIC_IPR0-NVIC_IPR59	RW	Privileged	0x00000000
0xE000EF00	STIR	WO	Configurable	0x00000000

Enable Interrupt

- Some interrupts are always enabled
  - System exceptions
- ISER register to enable interrupt

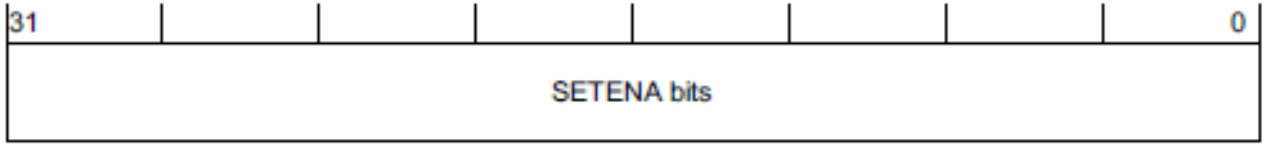


Table 4-4 ISER bit assignments

Bits	Name	Function
[31:0]	SETENA	Interrupt set-enable bits. Write: 0 = no effect 1 = enable interrupt. Read: 0 = interrupt disabled 1 = interrupt enabled.

Disable Interrupt

- ICER register to disable interrupt

The bit assignments are:

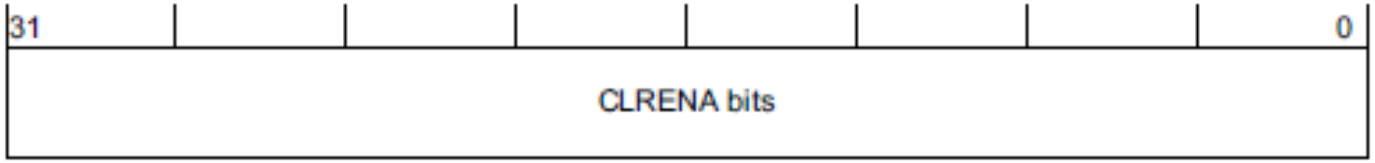


Table 4-5 ICER bit assignments

Bits	Name	Function
[31:0]	CLRENA	Interrupt clear-enable bits. Write: 0 = no effect 1 = disable interrupt. Read: 0 = interrupt disabled 1 = interrupt enabled.

Interrupt Set pending

- ISPR register to Set pending

The bit assignments are:

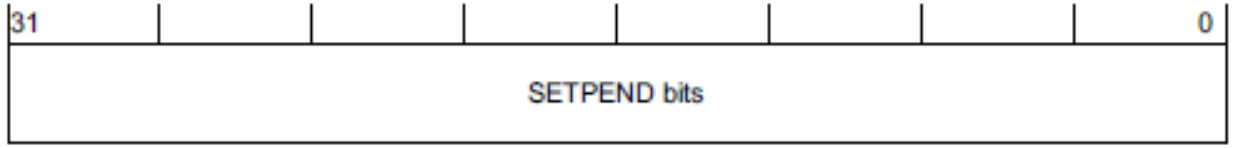


Table 4-6 ISPR bit assignments

Bits	Name	Function
[31:0]	SETPEND	Interrupt set-pending bits. Write: 0 = no effect 1 = changes interrupt state to pending. Read: 0 = interrupt is not pending 1 = interrupt is pending.

Interrupt Clear pending

- ISPR register to Clear pending

The bit assignments are:

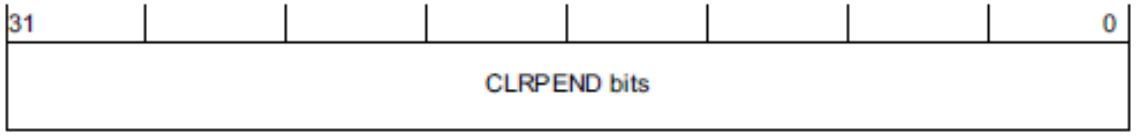


Table 4-7 ICPR bit assignments

Bits	Name	Function
[31:0]	CLRPEND	Interrupt clear-pending bits. Write: 0 = no effect 1 = removes pending state an interrupt. Read: 0 = interrupt is not pending 1 = interrupt is pending.

Interrupt Active Bit Register (IABR)

- Read Only
- IABR register to Clear pending

The bit assignments are:

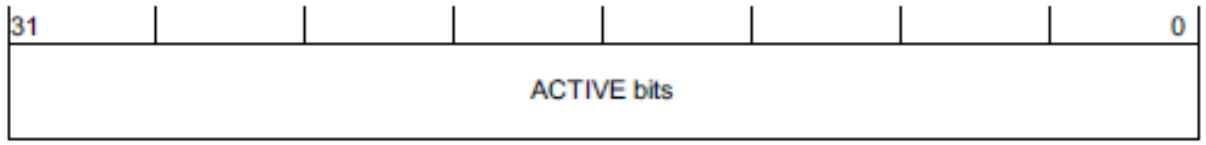
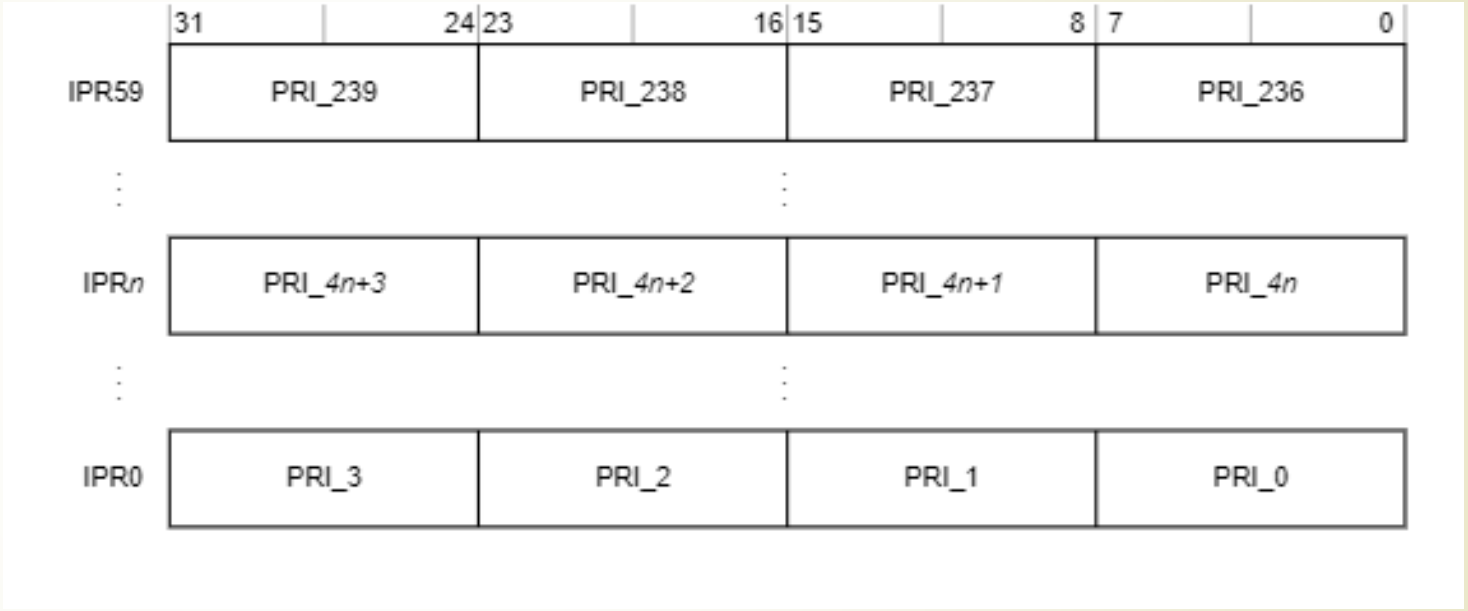


Table 4-8 IABR bit assignments

Bits	Name	Function
[31:0]	ACTIVE	Interrupt active flags: 0 = interrupt not active 1 = interrupt active.

Priority Registers

- The NVIC_IPR0-NVIC_IPR59 registers provide an 8-bit priority field for each interrupt
- Each registers hold 4 priority fields
- Each field could hold 0-255 priority value
- IPR number is IRQ Mod 4



How to Enable Interrupt

- Using CMSIS (recommended if available)
  - NVIC_EnableIRQ (IRQn); // enable Interrupt
  - NVIC_DisableIRQ (IRQn); // disable interrupt
- Using registers
  - Enable:

    NVIC->ISER[IRQn / 32] = 1 << (IRQn % 32);

    OR

    NVIC->ISER[IRQn >> 5] = 1 << (IRQn & 0x1F);
  - Disable:
    - NVIC->ICER[IRQn >> 5] = 1 << (IRQn & 0x1F); Directly with regsiters

Example

TIM7_IRQn = 44

// Must use ISER1 as it is more then 31
44/32 = 1 // ISER1
44%32 = 12
1 << 12 // Will set the 12 location
NVIC->[1] = 1 // enable timer 7 interrupt

Masking Interrupt

- FAULTMASK: Used to disable all exceptions except Non-maskable interrupt (NMI)
- The PRIMASK register prevents activation of all exceptions with configurable priority
- The BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the BASEPRI value

Software Triggered Interrupt Register (STIR)

- Write-only register
- Provides a mechanism for software to generate an interrupt
- Has the same effect as setting the NVIC ISPR bit

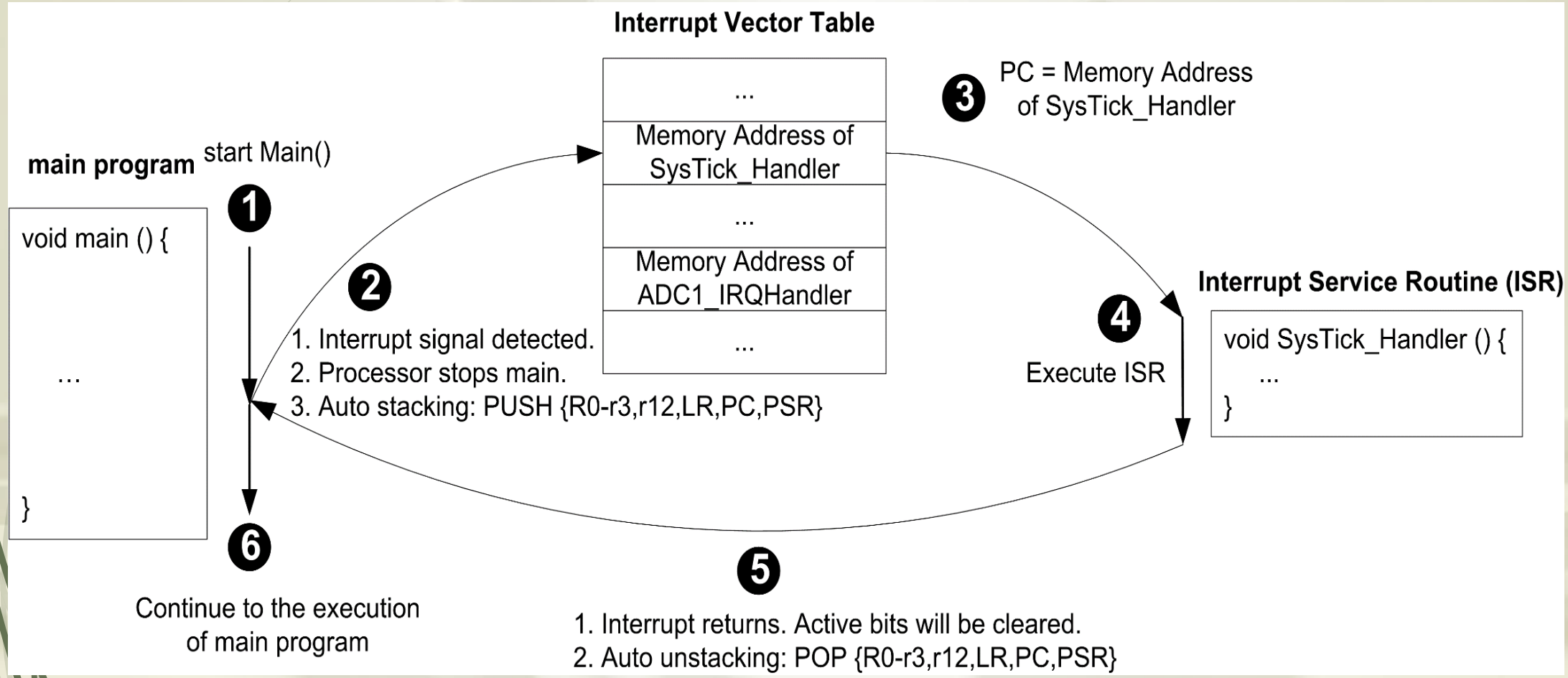
NVIC->STIR = IRQn

; Enter the IRQ Number

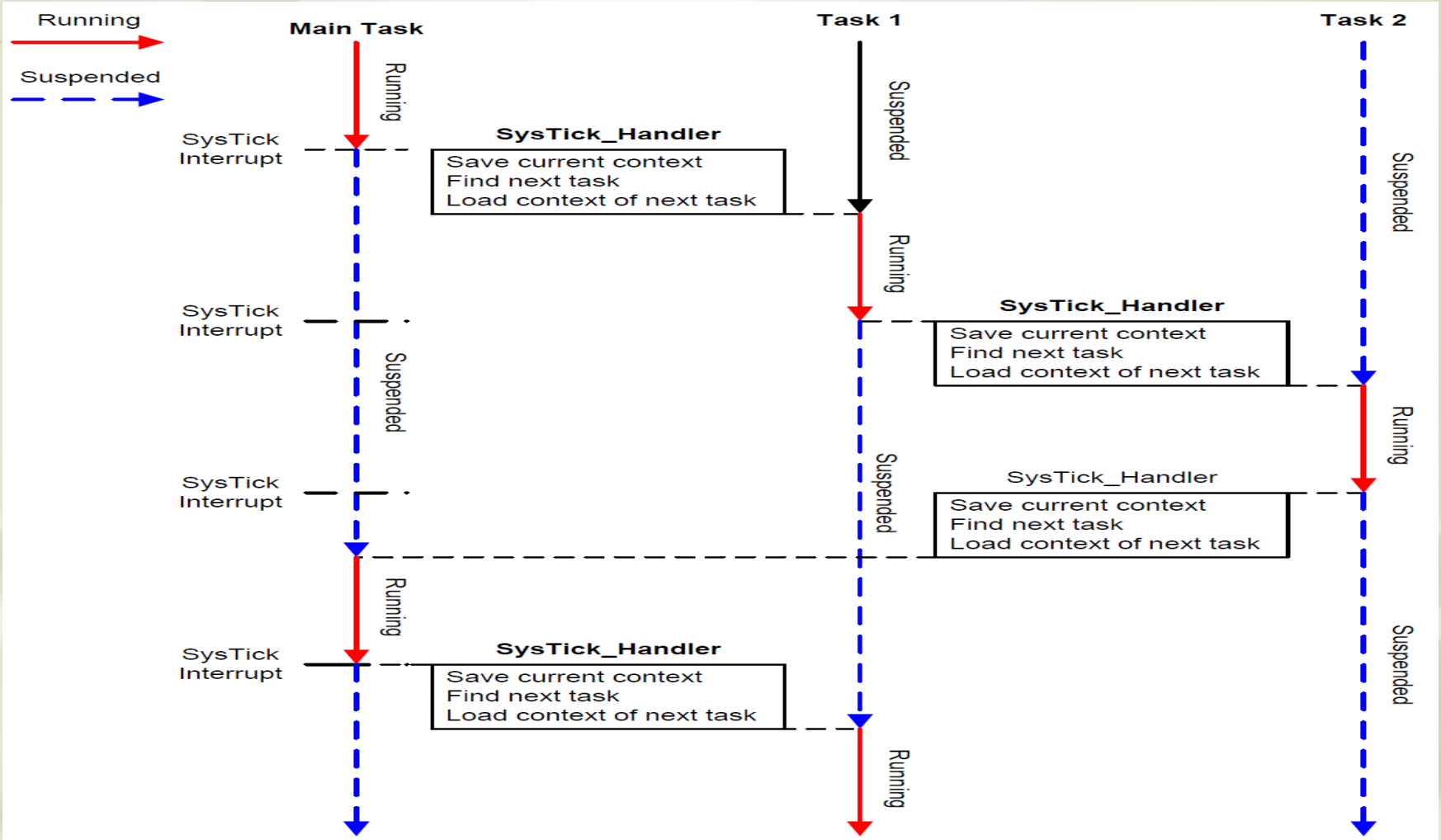
The bit assignments are:

Bits	Field	Function
[31:9]	-	Reserved.
[8:0]	INTID	Interrupt ID of the interrupt to trigger, in the range 0-239. For example, a value of 0x03 specifies interrupt IRQ3.

Interrupt Flow



Interrupt and Task Switching



Exception Entry and Return

- Preemption
  - When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled.
  - When one exception preempts another, the exceptions are called nested exceptions
- Tail-chaining
  - On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.
  - This speeds up exception servicing
- Late-arriving
  - If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception
  - This mechanism speeds up preemption
- Return
  - Occurs when exception handler is completed

Interrupt Latency

- It is time taken by processor to act on after interrupt was generated
- Also known as Interrupt response time
- Could be affected by the types of instruction CPU was executing
  - More if instructions being executed lasts for many instruction cycles compared to one cycle
- Depends on controller, interrupt controller, clock speed and OS
- What about Interrupt Priority ?
- **Low latency for hard real time system is preferred**

ISR's

- The interrupt service routine is the software module that is executed when the hardware requests an interrupt
- The design of the interrupt service routine requires careful consideration of many factors
- The software in this class will exclusively use the MSP. It is imperative that the ISR software balance the stack before exiting.
- Although interrupt handlers can create and use local variables, parameter passing between threads must be implemented using shared global memory variables
- The execution of the main program is called the foreground thread, and the executions of the various interrupt service routines are called background threads.

Interrupt/Exception State

- Each exception could be in one of the following states:
  - Inactive
    - Exception is not active and not pending.
  - Pending
    - Exception is waiting to be serviced by the processor.
    - An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.
  - Active
    - Exception that is being serviced by the processor but has not completed.
  - Active and pending
    - Exception is being serviced by the processor and there is a pending exception from the same source.

Interrupt/Exception Handlers

- Interrupt Service Routines (ISRs)
- Fault handlers
  - Hard fault, memory management fault, usage fault, bus fault are fault exceptions handled by the fault handlers
- System handlers
  - NMI, PendSV, SVCall SysTick, and the fault exceptions are all system exceptions that are handled by system handlers

Useful Links

- https://developer.arm.com/documentation/dui0552/a/cortex-m3-peripherals/nested-vectored-interrupt-controller?lang=en
- https://developer.arm.com/documentation/dui0552/a/cortex-m3-peripherals/nested-vectored-interrupt-controller/nvic-usage-hints-and-tips?lang=en
- https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-processor/programmers-model/core-registers?lang=en
- https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-processor/programmers-model/processor-mode-and-privilege-levels-for-software-execution?lang=en
- Attached PDF

Next Lesson Topic

- IDE File structure
- HAL API
- CMSIS Layer
- Interrupt controller registers
- Code examples