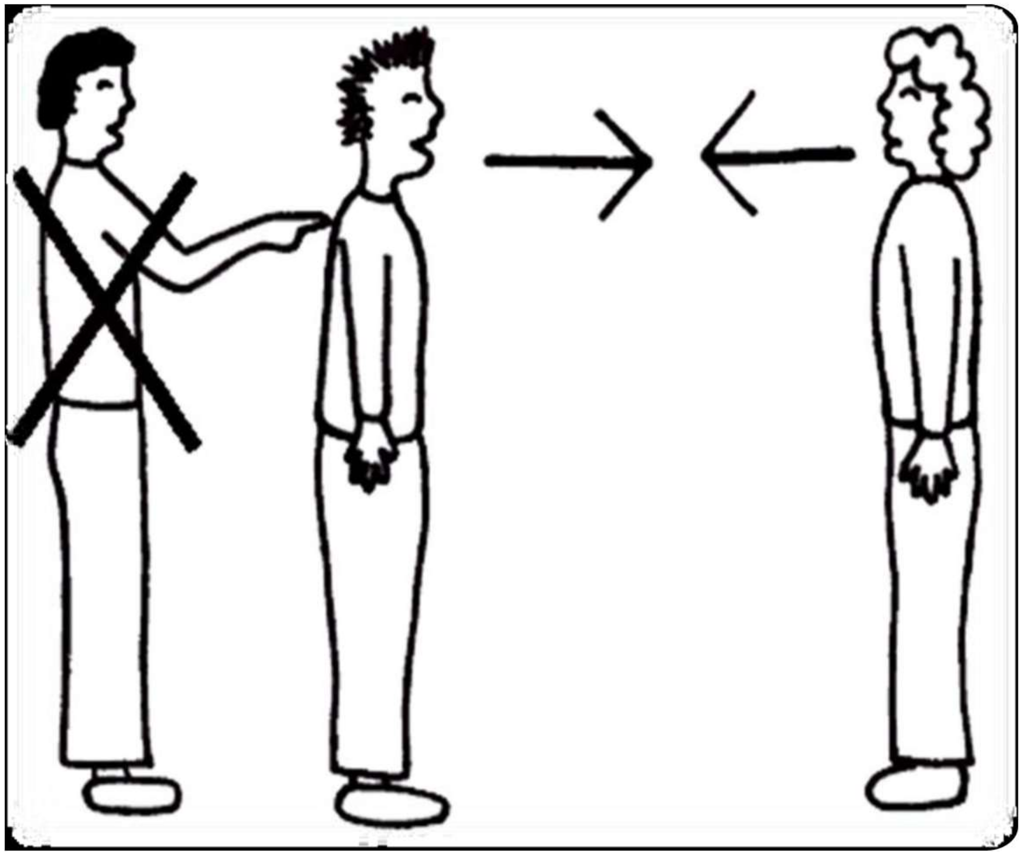
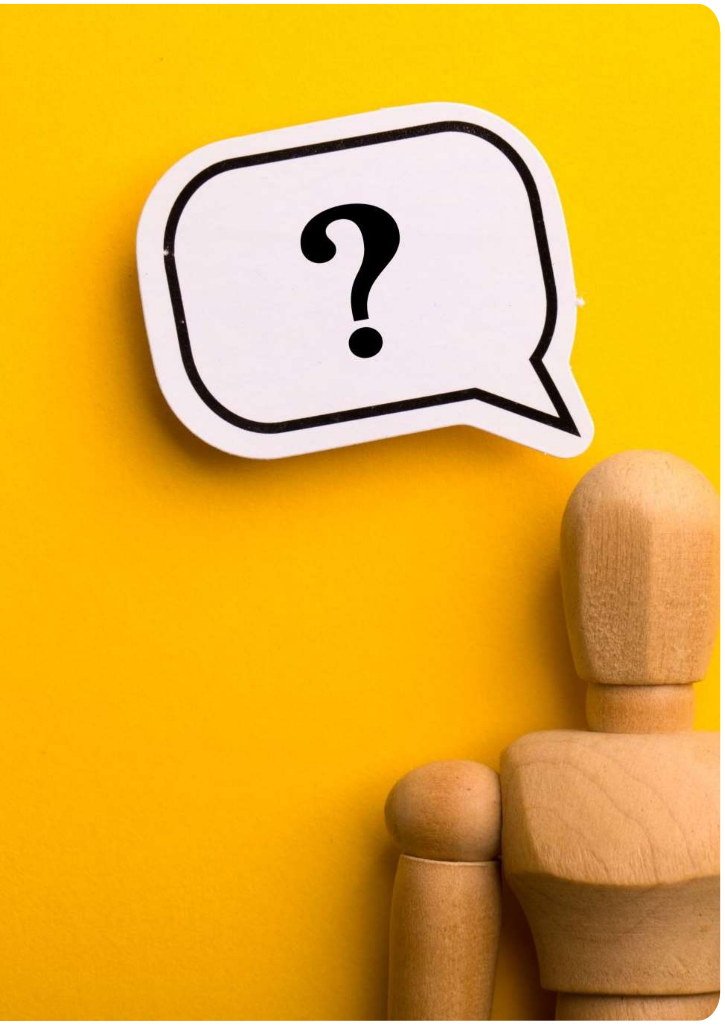
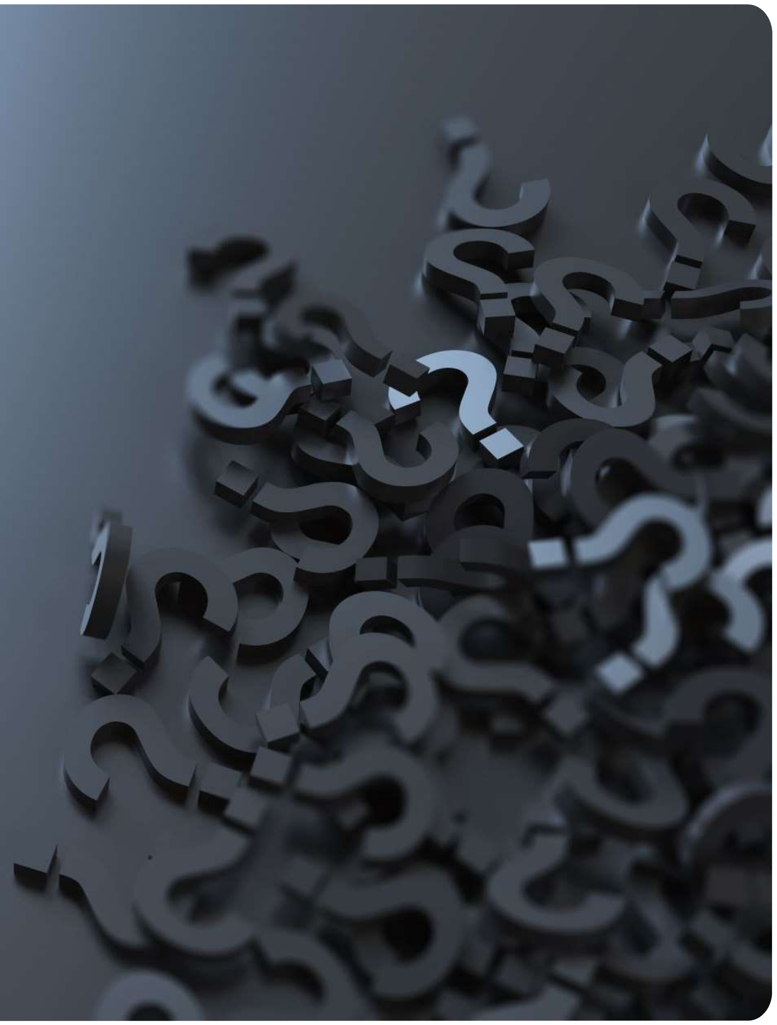


Interrupts and Exception Handling



What is Interrupt?

Interrupt make you to stop what you are doing and jump to the one who request you.



What is exception?

An exception is any condition that needs to halt normal execution of the instructions.

Interrupt

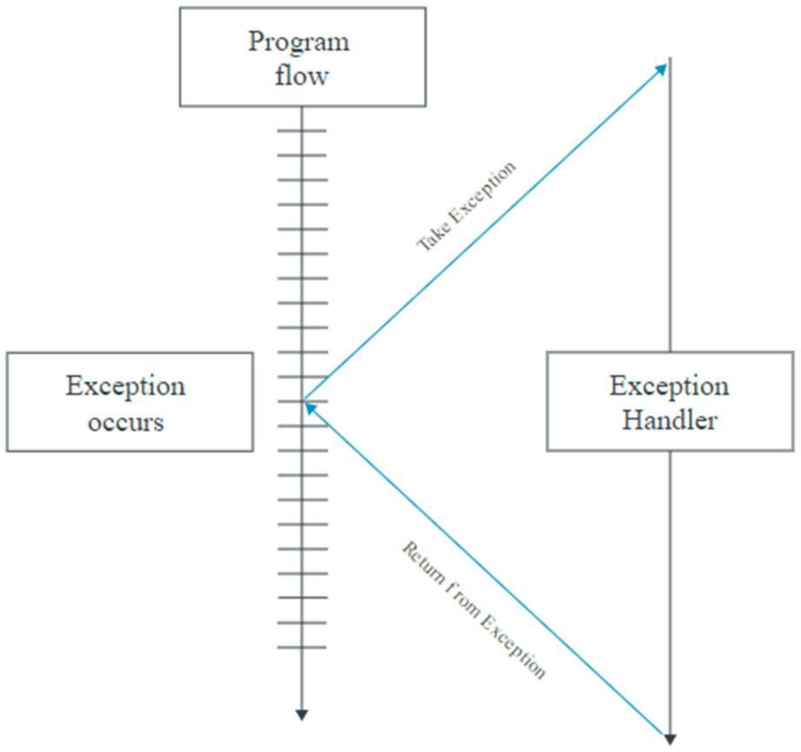
- An interrupt is a signal(an "interrupt request") generated by some event external to the CPU , which causes the CPU to stop what it is doing(stop executing the code it is currently running) and jump to a separate piece of code designed by the programmer.

- GPIO, Timers, UART, I2C, SPI, etc…

Why Interrupts?

- External devices are comparatively slower than CPU.
- CPU would waste a lot of time waiting for external devices to match its speed with that of CPU.

What happens?



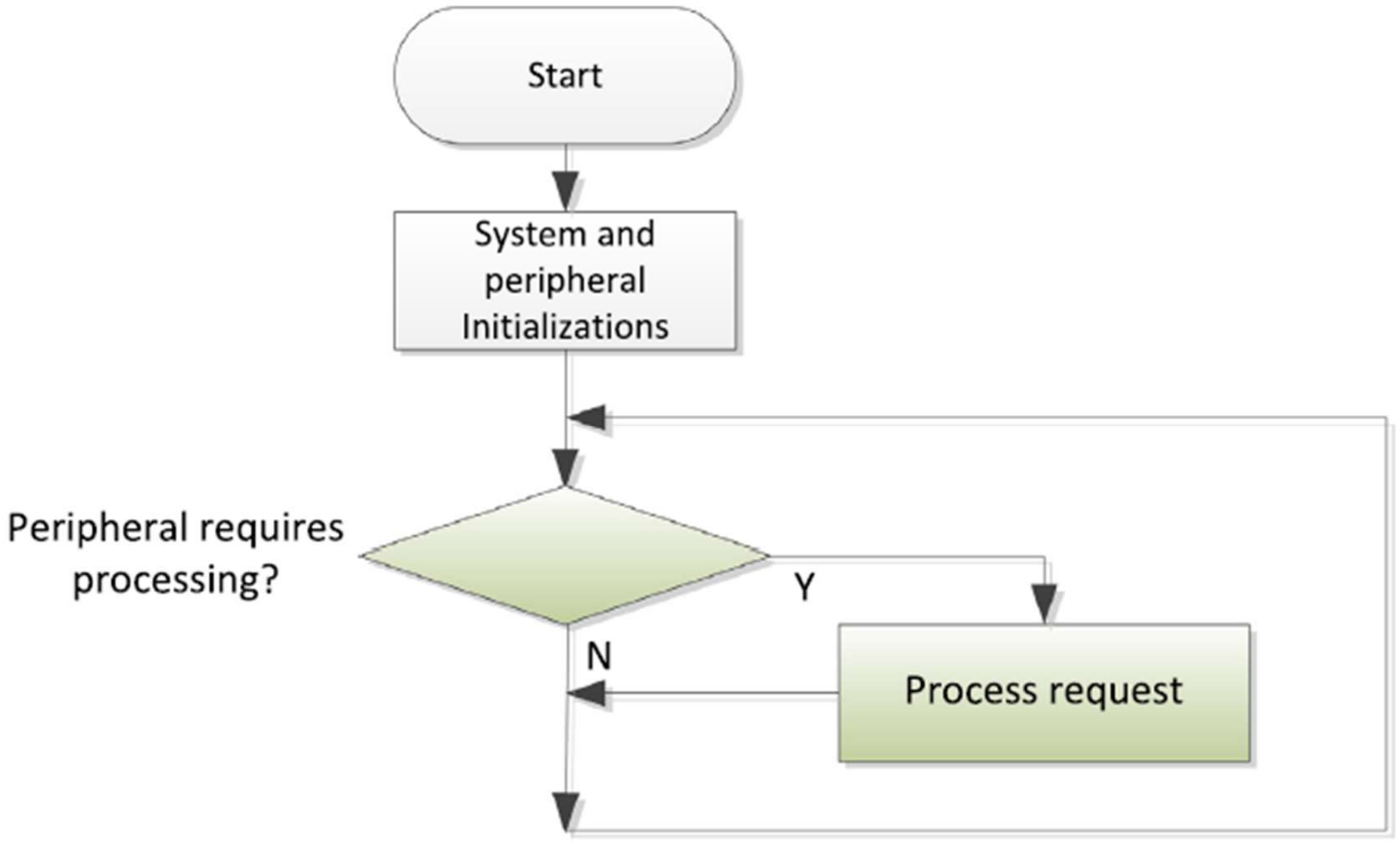
Software Flow:

- Polling
- Interrupt
- Multi- Tasking

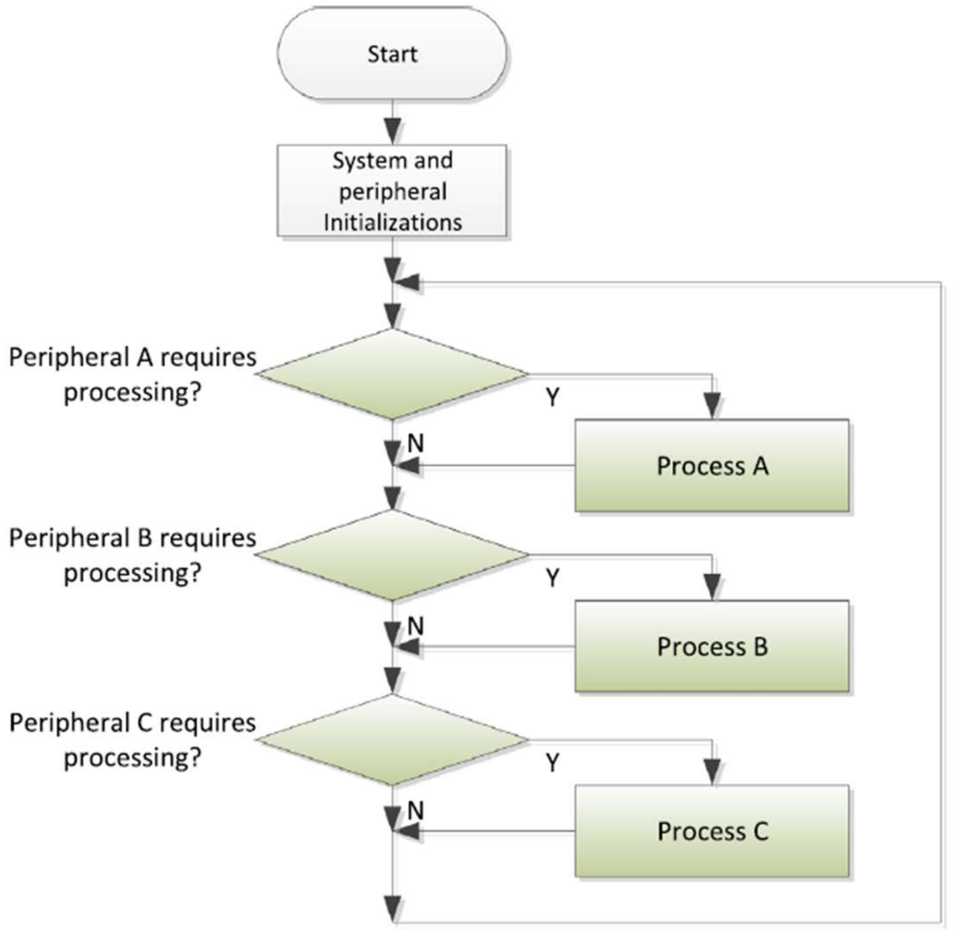
Polling

- The processor wait until there is data ready for processing, process it, and then wait again.

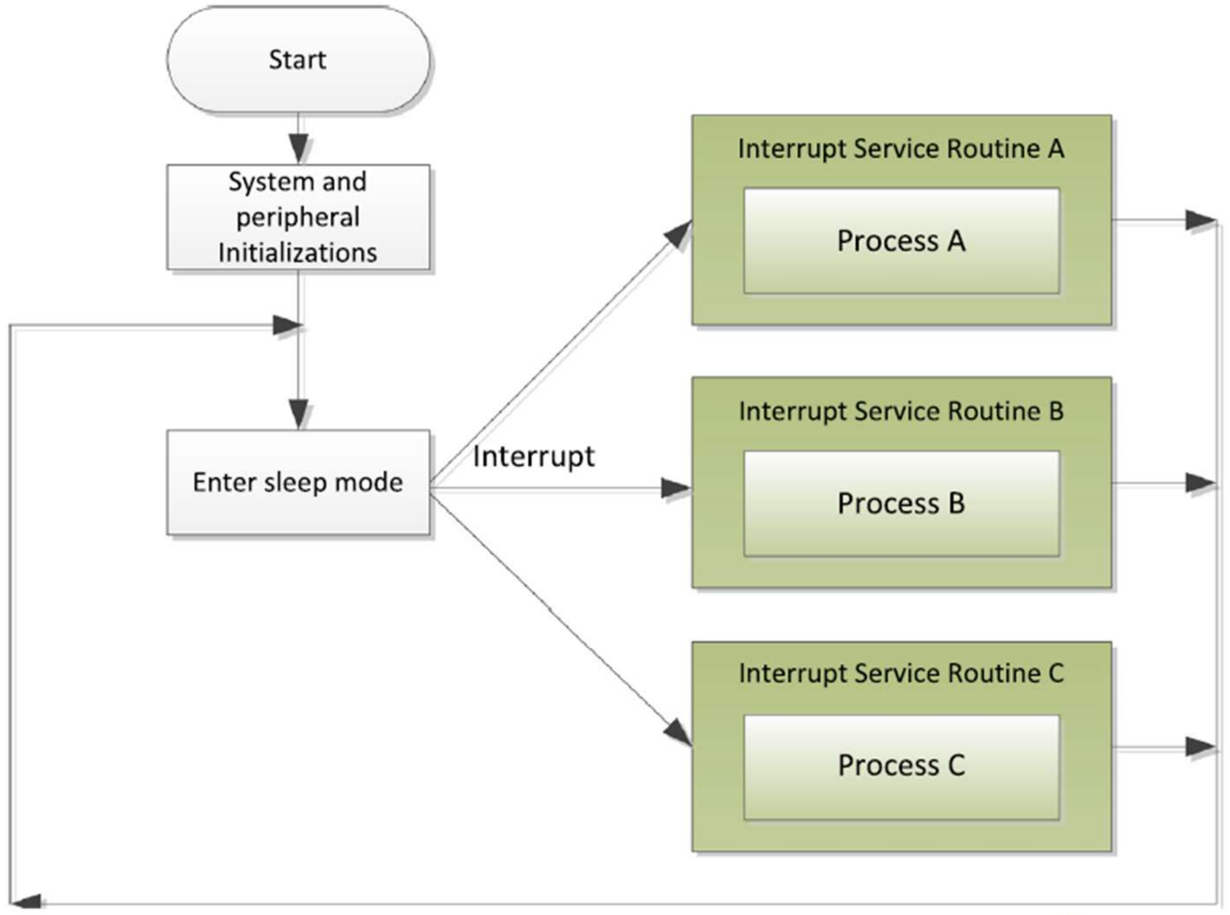
Polling method for simple application processing


Start
System and peripheral Initializations
Peripheral requires processing?
Y
N
Process request

Polling method for application with multiple devices that need processing.

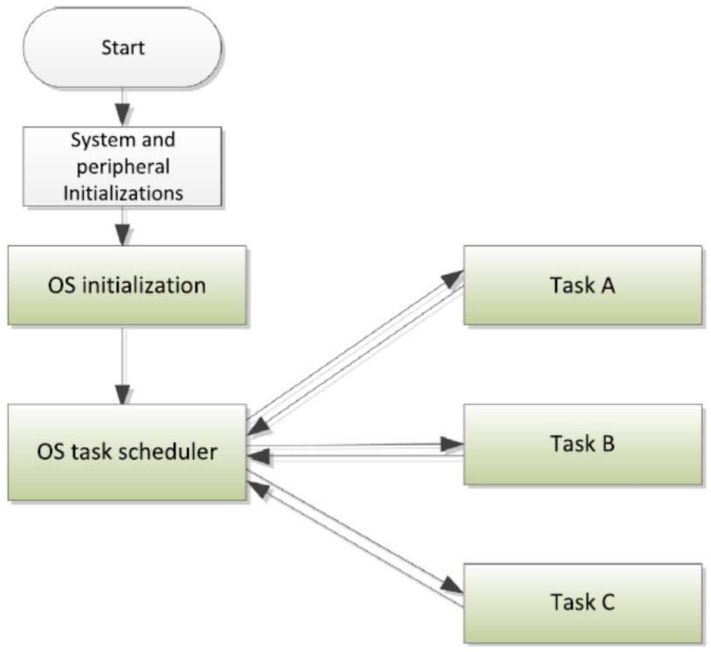

Start
System and peripheral Initializations
Peripheral A requires processing?
Y
N
Process A
Peripheral B requires processing?
Y
N
Process B
Peripheral C requires processing?
Y
N
Process C

Simple interrupt-driven application


Start
System and peripheral Initializations
Enter sleep mode
Interrupt
Interrupt Service Routine A
Process A
Interrupt Service Routine B
Process B
Interrupt Service Routine C
Process C

Multi-tasking Systems

Using an RTOS to handle multiple tasks



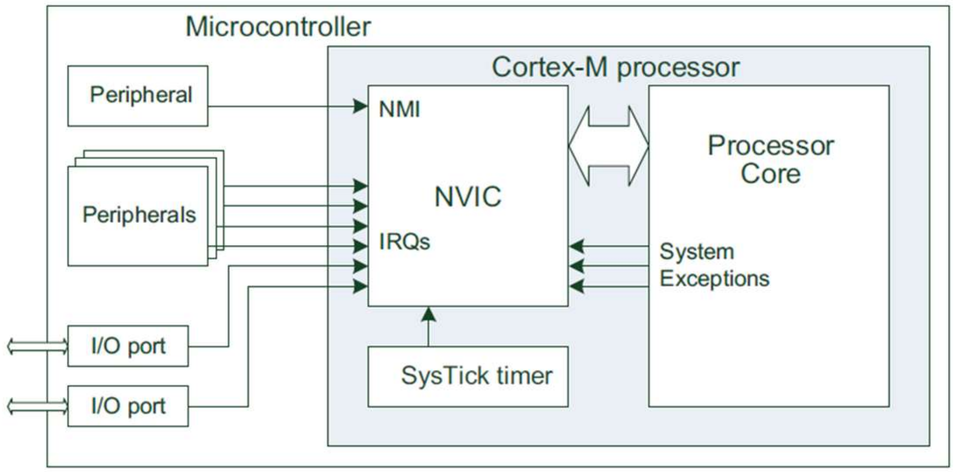
Interrupt Service Routine(ISR):

- An interrupt handler, also known as an interrupt service routine (ISR), is a callback whose execution is triggered by the reception of an interrupt request(IRQ).

Interrupt Latency

- Interrupt latency is the time between the occurrence of an interrupt and the execution of the first instruction of the interrupt service routine (ISR) that handles the interrupt.

Interrupts in ARM



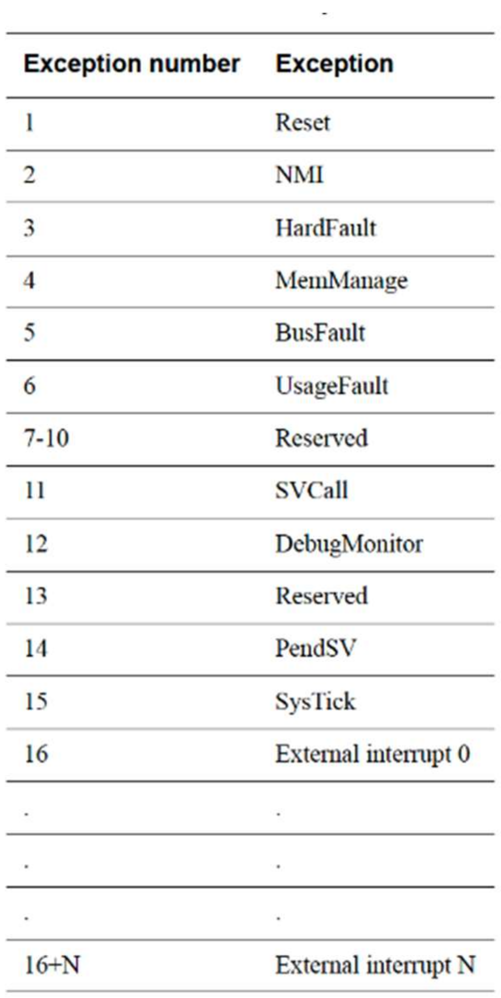
- All Cortex-M processors provide a Nested Vectored Interrupt Controller (NVIC) for interrupt handling.

- The Cortex-M3 and Cortex-M4 NVIC supports up to 240 IRQs (Interrupt Requests), a Non-Maskable Interrupt (NMI), a SysTick (System Tick) timer interrupt, and a few system exceptions.

- Most of the IRQs are generated by peripherals such as timers, I/O ports, and communication interfaces (e.g., UART, I2C).

List of Exceptions:

- Refer Pg No. 232 of Textbook

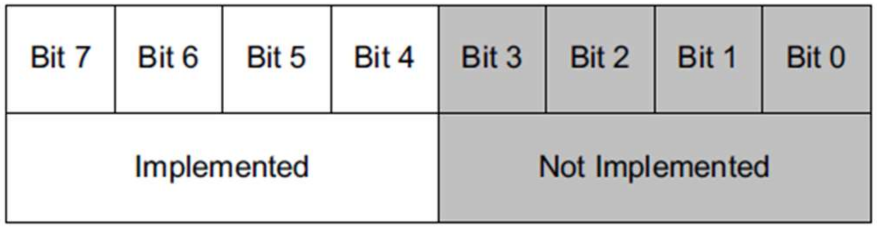


Exception number	Exception
1	Reset
2	NMI
3	HardFault
4	MemManage
5	BusFault
6	UsageFault
7-10	Reserved
11	SVCall
12	DebugMonitor
13	Reserved
14	PendSV
15	SysTick
16	External interrupt 0
.	.
.	.
.	.
16+N	External interrupt N

Priorities

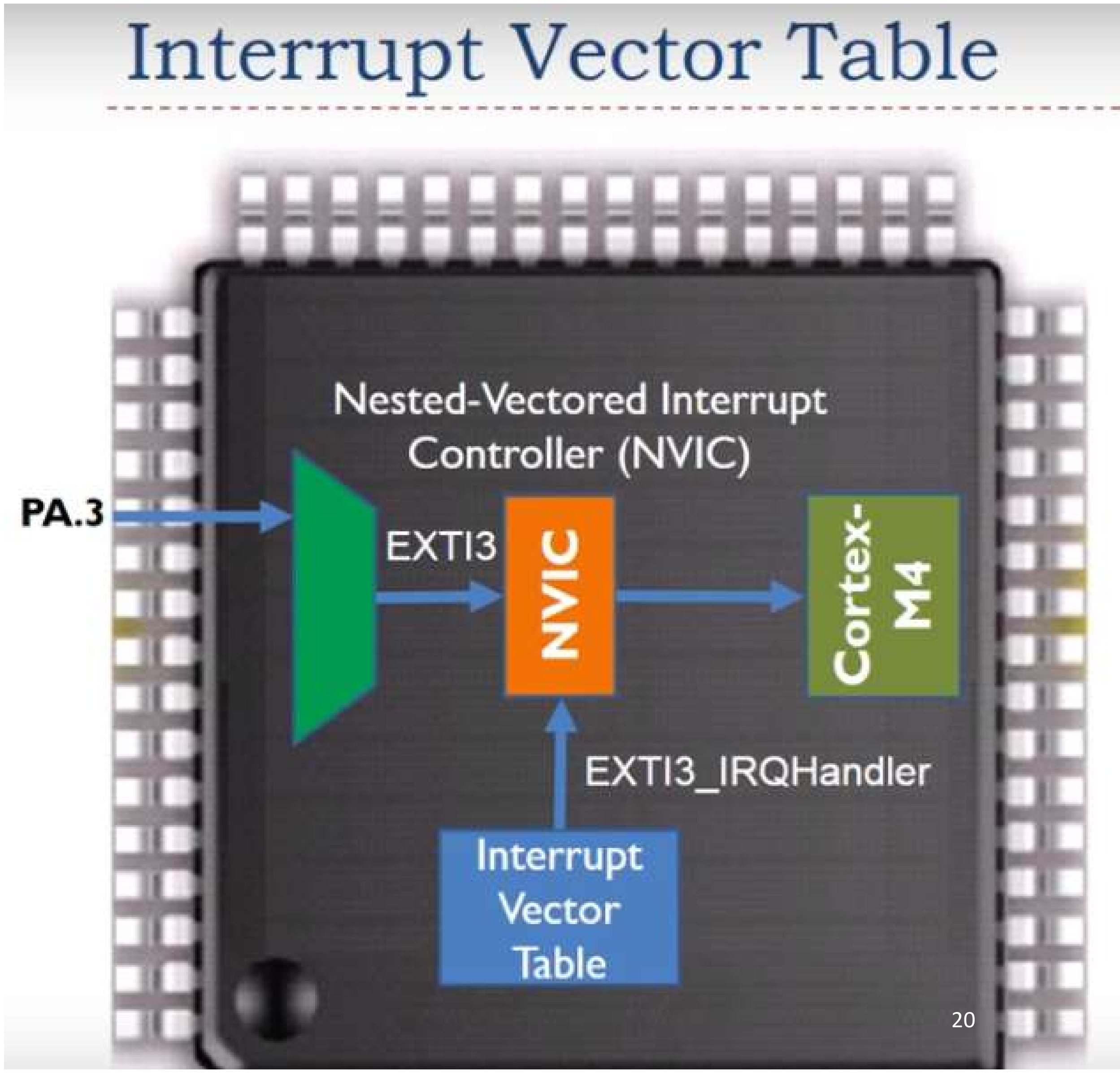
- Smaller the number, Higher the priority
- A Higher Priority can pre-empt the lower priority (This is called Nested).
- Some of the exceptions (reset, NMI, and Hard-Fault) have fixed priority levels. (Negative Values)
- The actual number of available programmable priority levels is decided by silicon chip designers.
- Refer Datasheet

- Interrupt-priority levels are controlled by priority-level registers.
- 4 bits of priority level means 16 levels of programmable priority level.
- The more bits are implemented, the more priority levels will be available.



Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Implemented				Not Implemented			

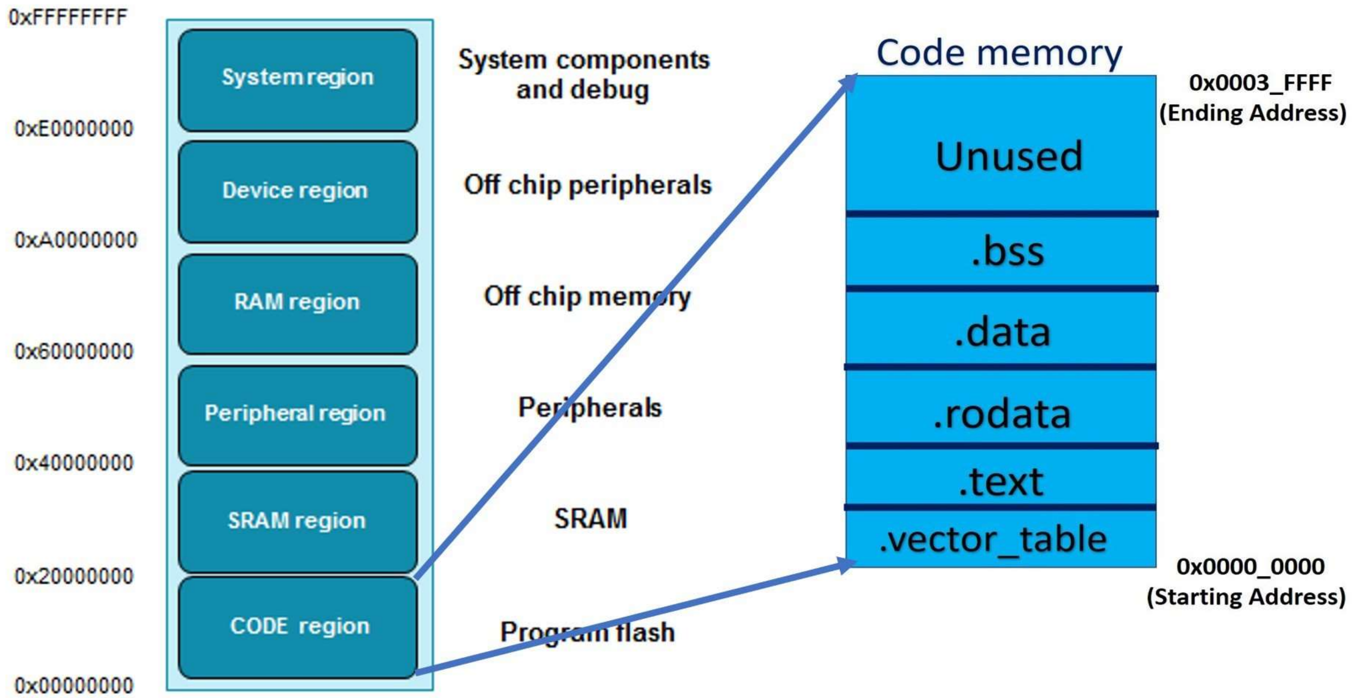
How Controller Know which interrupt?



Vector Table?

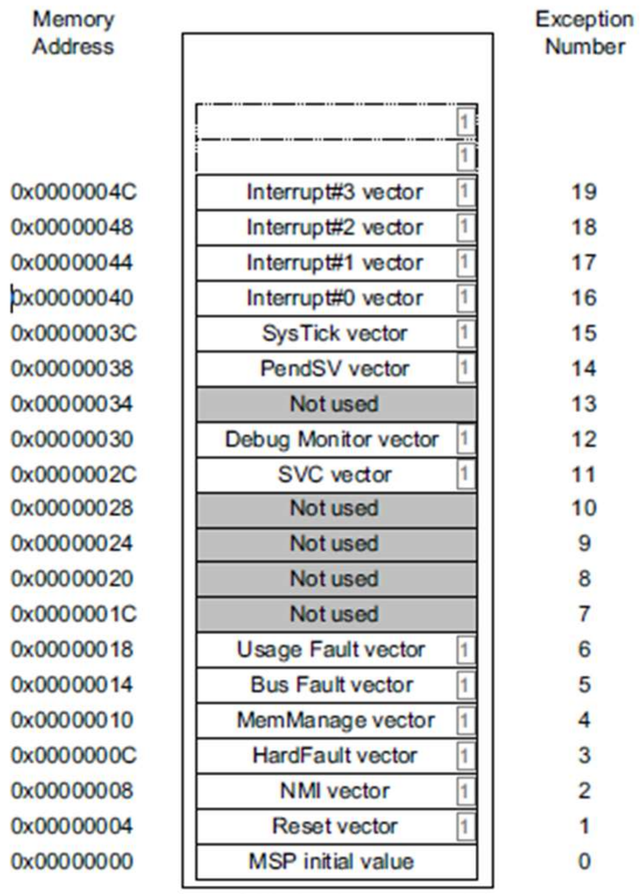
- The vector table is normally defined in the startup codes provided by the microcontroller vendors.
- The vector table contains the starting addresses of exception handlers.

Where it is stored?



- The vector table starts at memory address 0.

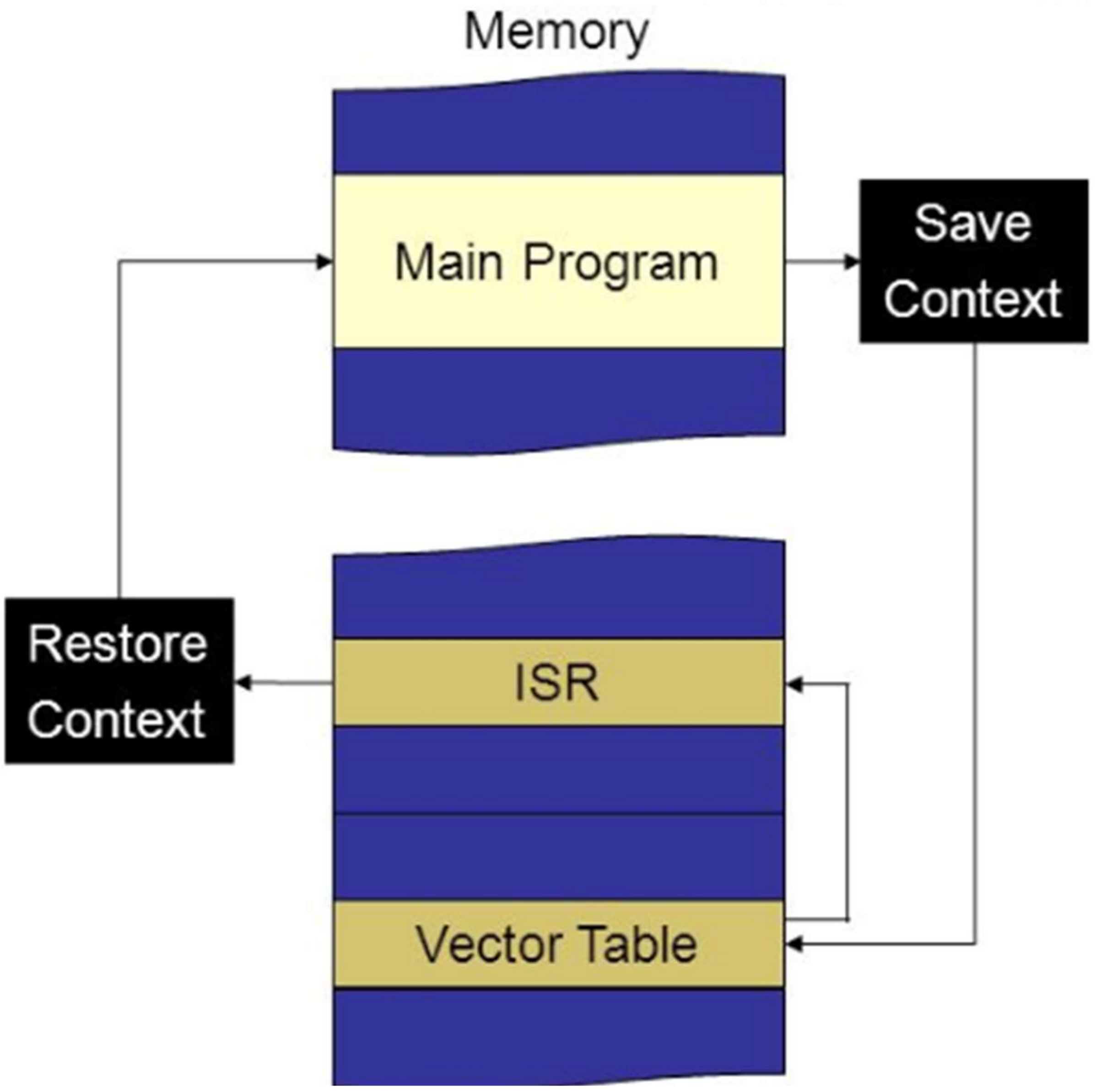
- The vector address is arranged according to the exception number times four
- In Cortex-M, the vector table contains the starting addresses of exception handlers.



Memory Address			Exception Number
		1	
		1	
0x0000004C	Interrupt#3 vector	1	19
0x00000048	Interrupt#2 vector	1	18
0x00000044	Interrupt#1 vector	1	17
0x00000040	Interrupt#0 vector	1	16
0x0000003C	SysTick vector	1	15
0x00000038	PendSV vector	1	14
0x00000034	Not used		13
0x00000030	Debug Monitor vector	1	12
0x0000002C	SVC vector	1	11
0x00000028	Not used		10
0x00000024	Not used		9
0x00000020	Not used		8
0x0000001C	Not used		7
0x00000018	Usage Fault vector	1	6
0x00000014	Bus Fault vector	1	5
0x00000010	MemManage vector	1	4
0x0000000C	HardFault vector	1	3
0x00000008	NMI vector	1	2
0x00000004	Reset vector	1	1
0x00000000	MSP initial value		0

Note : LSB of each vector must be set to 1 to indicate Thumb state

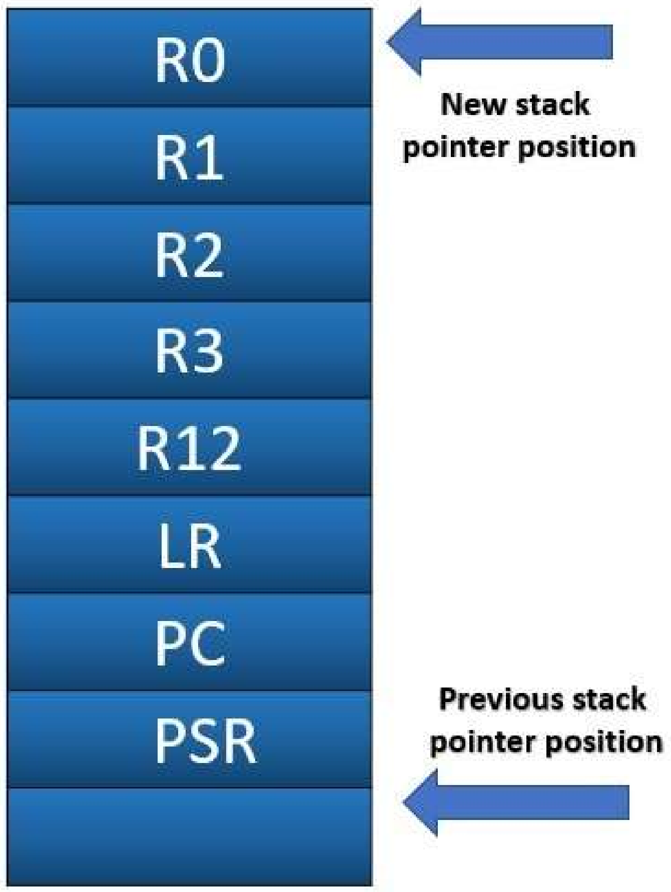
What happens when interrupt comes?



What is Context?

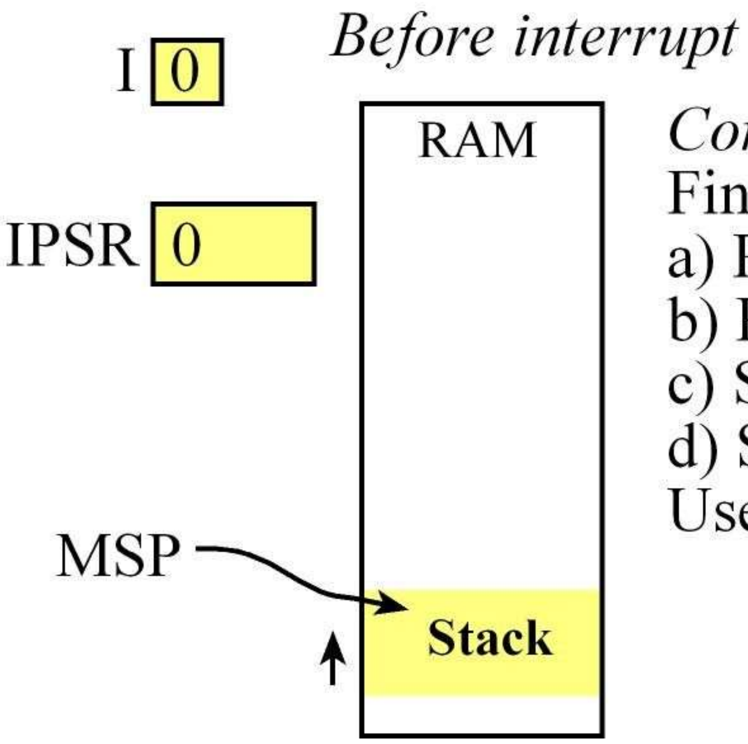
- Current instruction will be finished.
- **R0, R1, R2, R3, R12, LR, PC,** and **PSR** will restore.
- The **LR** is set to a specific value signifying an interrupt service routine
- The **IPSR (**Interrupt Program Status Register**)** is set to the interrupt number being processed.
- The **PC** is loaded with the address of the ISR (vector).

These five steps, called a context switch

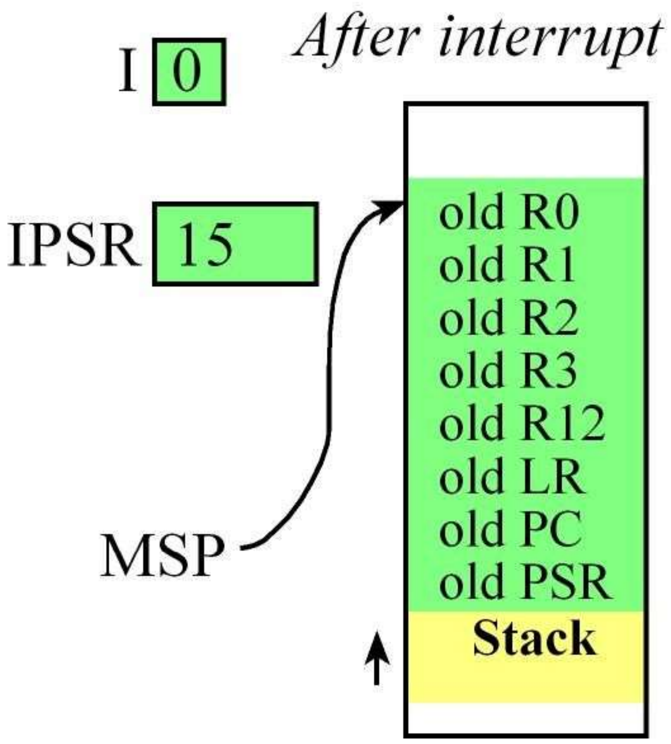


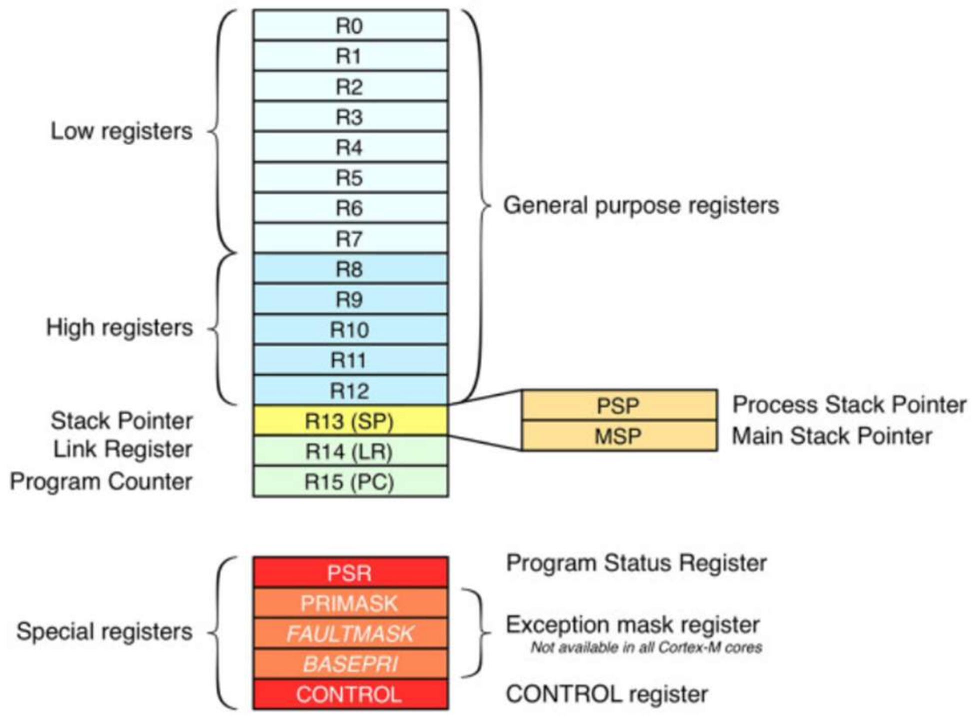
Why only these?

- ARM do not use R4-R11 registers during ISR execution.
- So only these registers (R0, R1, R2, R3, R12, LR, PC, and PSR) will be changed.
- Other Registers does not change.



Context Switch
Finish instruction
a) Push registers
b) PC = {0x0000003C}
c) Set IPSR = 15
d) Set LR = 0xFFFFFFF9
Use MSP as stack pointer

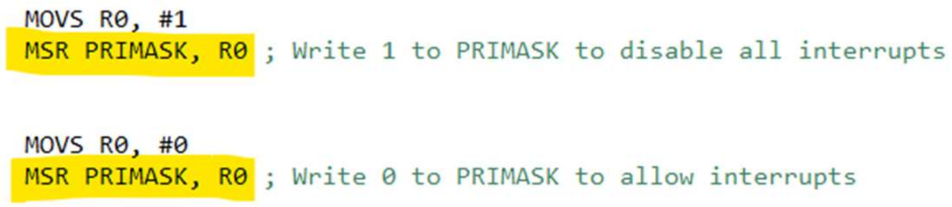




PRIMASK, FAULTMASK, and BASEPRI registers

- PRIMASK
  - The PRIMASK register is used to disable all exceptions except NMI and Hard-Fault.
  - If this is 1, disable Interrupts.
  - If this is 0, enable Interrupts.

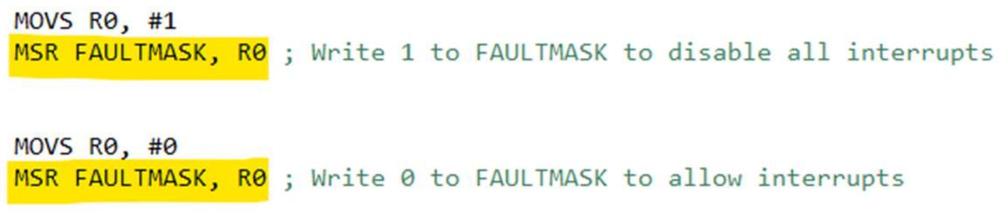
```
__set_PRIMASK(priMask);
```



```
MOVS R0, #1
MSR PRIMASK, R0 ; Write 1 to PRIMASK to disable all interrupts

MOVS R0, #0
MSR PRIMASK, R0 ; Write 0 to PRIMASK to allow interrupts
```

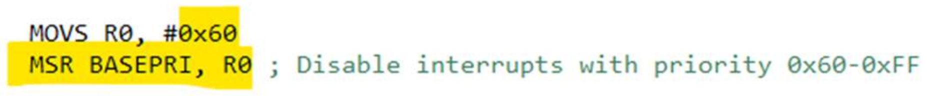
- FAULTMASK
  - FAULTMASK is very similar to PRIMASK except that it changes the effective current priority level to -1, so that even the Hard Fault handler is blocked.
  - If this is 1, which we will define as disabled.
  - If this is 0, which we will define as enabled.

```
__set_FAULTMASK(faultMask);
```



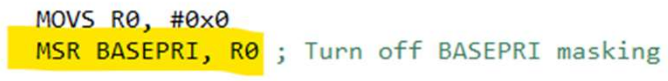
```
MOVS R0, #1
MSR FAULTMASK, R0 ; Write 1 to FAULTMASK to disable all interrupts

MOVS R0, #0
MSR FAULTMASK, R0 ; Write 0 to FAULTMASK to allow interrupts
```

- BASEPRI
  - Disable interrupts with priority lower than a certain level.
  - For example, if you want to block all exceptions with priority level equal to or lower than 0x60, you can write the value to BASEPRI:



```
MOVS R0, #0x60
MSR BASEPRI, R0 ; Disable interrupts with priority 0x60-0xFF
```

```
__set_BASEPRI(basePri);
```



```
MOVS R0, #0x0
MSR BASEPRI, R0 ; Turn off BASEPRI masking
```

SVCall

- SVCall (Super Visor Call) is a Software-triggered exception.
- Allowing a piece of code to execute without interruption.
- SVC executes in privileged mode.
- Mostly used in RTOS.

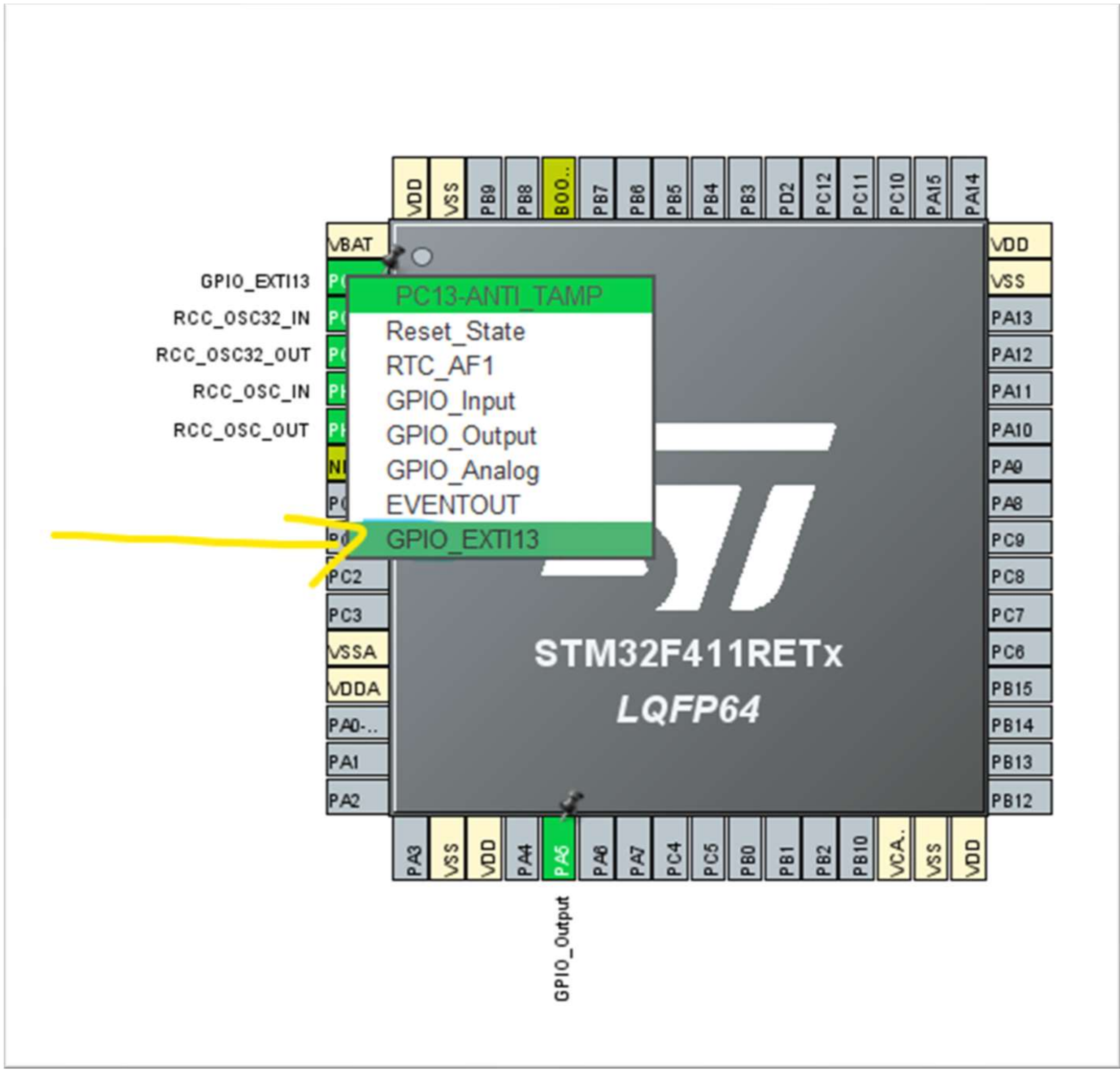
PendSV & SysTick

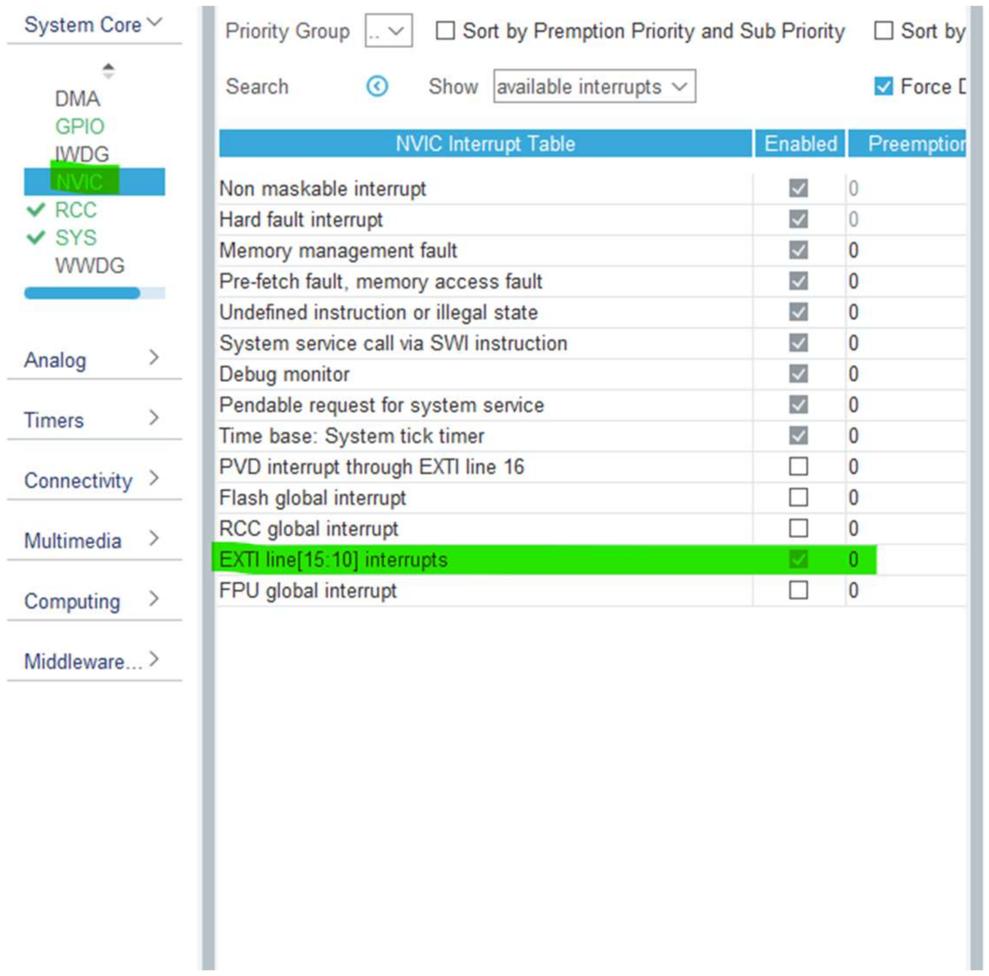
- PendSV (Pended Service Call) is another exception type that is important for supporting OS operations.
- System Tick Timer; Exception generates by a timer peripheral which is included in the processor.
- They are typically used when running a RTOS.

Programming in STM32CUBEIDE

- When you press a user button then LED should blink.
  - Polling – done?
  - Interrupt based.

Configuration:

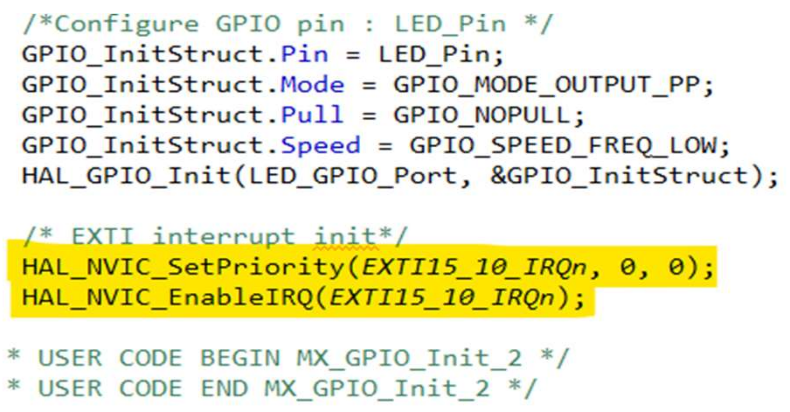




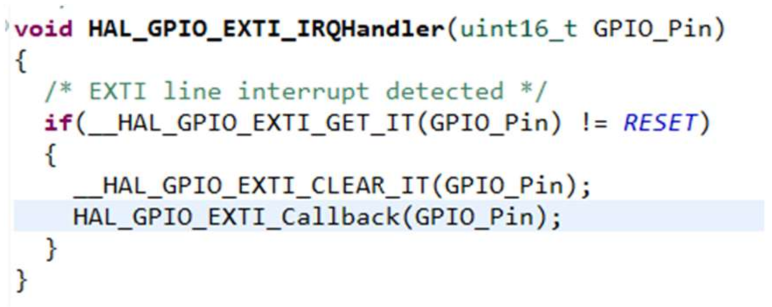
NVIC Interrupt Table	Enabled	Preemption
Non maskable interrupt	✓	0
Hard fault interrupt	✓	0
Memory management fault	✓	0
Pre-fetch fault, memory access fault	✓	0
Undefined instruction or illegal state	✓	0
System service call via SWI instruction	✓	0
Debug monitor	✓	0
Pendable request for system service	✓	0
Time base: System tick timer	✓	0
PVD interrupt through EXTI line 16	☐	0
Flash global interrupt	☐	0
RCC global interrupt	☐	0
EXTI line[15:10] interrupts	✓	0
FPU global interrupt	☐	0

- Find out the priority of the Interrupt?

- What is the Handler of the Interrupt?

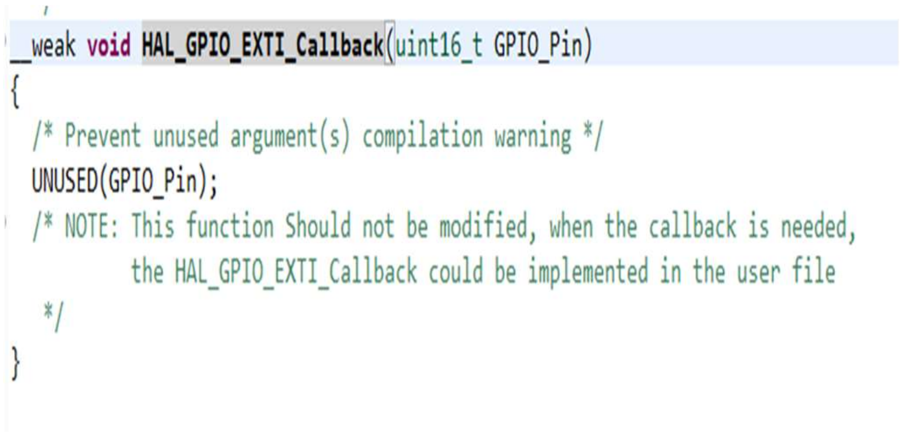


```
/*Configure GPIO pin : LED_Pin */
GPIO_InitStruct.Pin = LED_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(LED_GPIO_Port, &GPIO_InitStruct);

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

* USER CODE BEGIN MX_GPIO_Init_2 */
* USER CODE END MX_GPIO_Init_2 */
```



```
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  /* EXTI line interrupt detected */
  if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
  {
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Callback(GPIO_Pin);
  }
}
```

- Call back Function
- Toggle LED



```
__weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function Should not be modified, when the callback is needed,
           the HAL_GPIO_EXTI_Callback could be implemented in the user file
   */
}
```

Queries?

Thank you