Interrupts in TM4C123GH6PM Launchpad

⊙ Interrupts in TM4C123GH6PM

Aim:- To deal with GPIO interrupt that will toggle the on board LED, whenever the switch is pressed.

* micro-controller has built in capacity to perform several tasks without waiting for completion of tasks

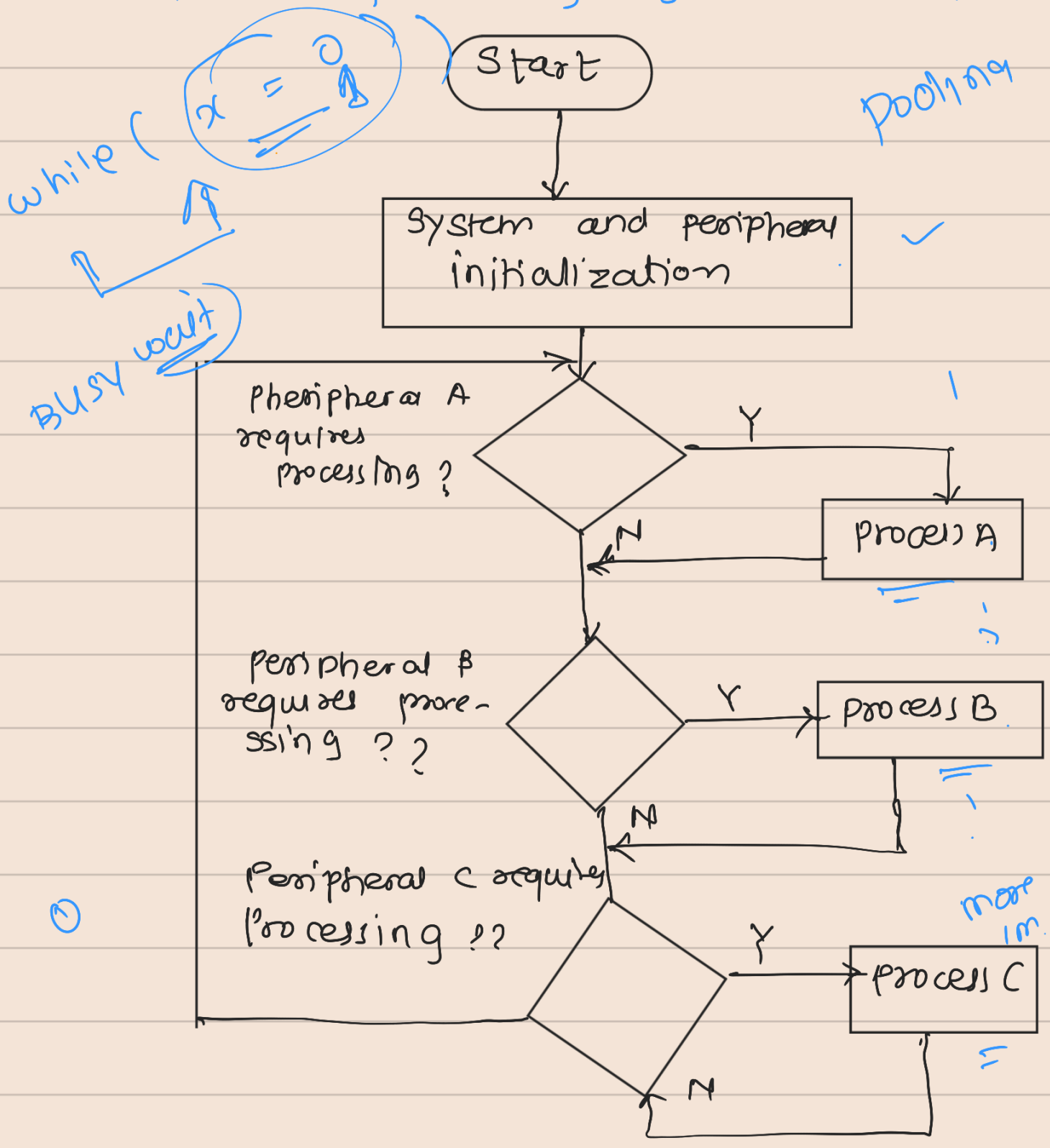
→ there are two methods by which a peripheral device can receive a service from micro-controller

i) Polling ✓

ii) Interrupt based. ✓

→ In polling a micro-controller continuously monitors the status of a device or a particular condition and waits until the required status

or condition is met then it performs the service, after performing goes to next task.



→ When application get complex, the polling loop design get complex & difficult to maintain

→ Hence poor responsiveness, cohere more

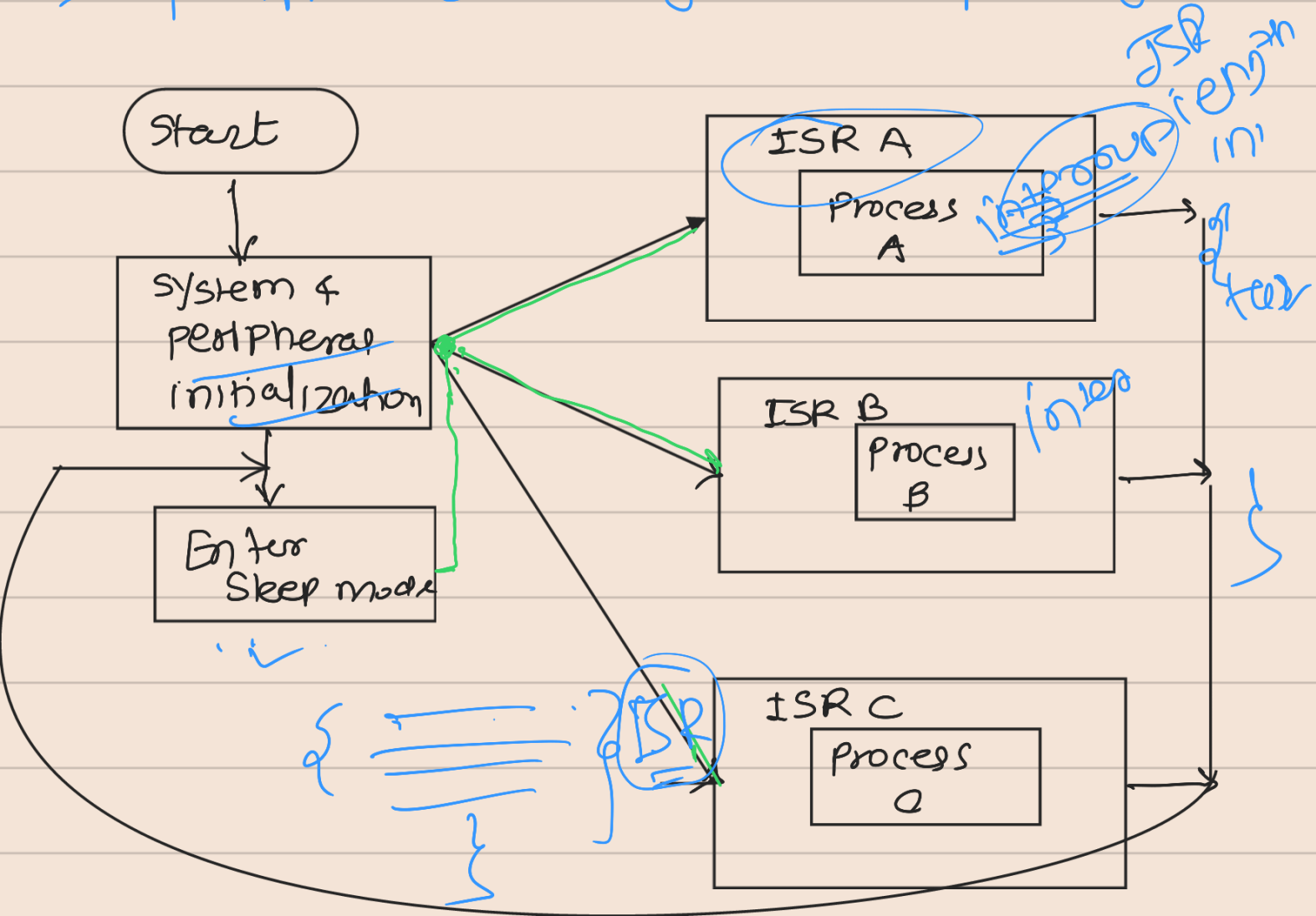
→ (Not energy efficient)

→ importance task will be held on wait while processor doing unimportant task.

→ Interrupt driven:-

To solve the polling problem, almost all micro-controller have some sort of sleep mode support to reduce power, in which peripheral can wake up the processor when it require a service. Known as interrupt driven.

→ peripherals - assigned with priority



⊙ ISR (Interrup handler or ISR)

Exception: (interrupt)

Events that causes change to program flow when processor suspends current execution run the task for exception and after completion comes ISR back.

Sources:

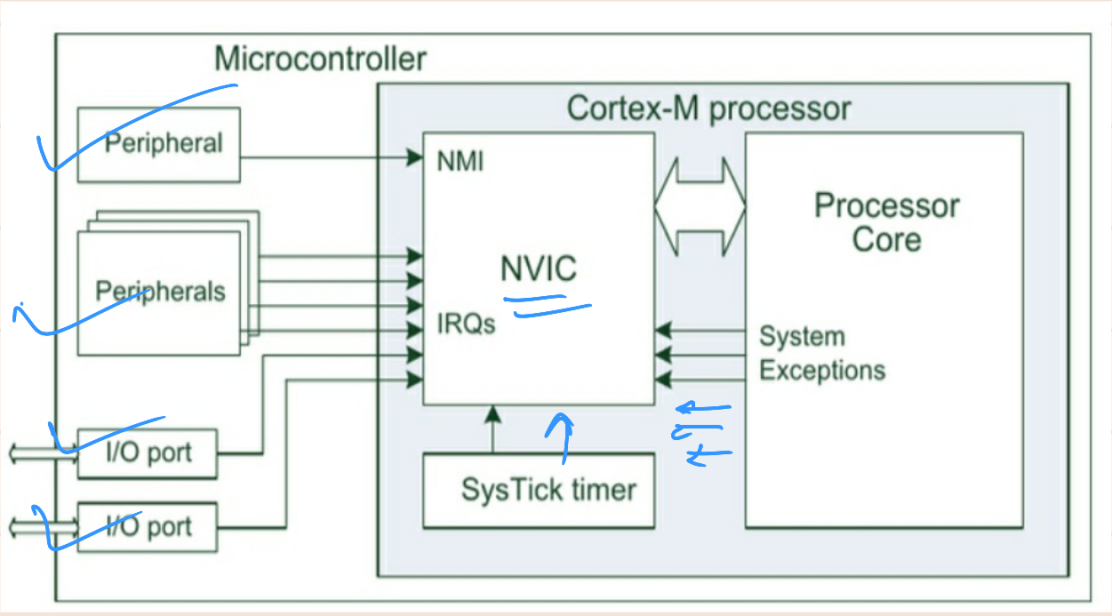
i) Peripherals
ii) external input sources
iii) internal hardware
iv) software triggered

NVIC : (Nested vectored interrupt controller)

→ Exceptions are processed by NVIC.
→ Can handle No. of IRQ & NMI Request.
→ Exceptions source has an exception number.

1 to 15 = system exception & 16 onward above interrupts

→ For Cortex M3-M4 total 240 interrupt inputs can control



→ Part of cortex M4 which is programmable and it's registers are stored in SCS (system control space) NVIC registers

→ handles the exception, interrupt configuration, prioritization and interrupt masking

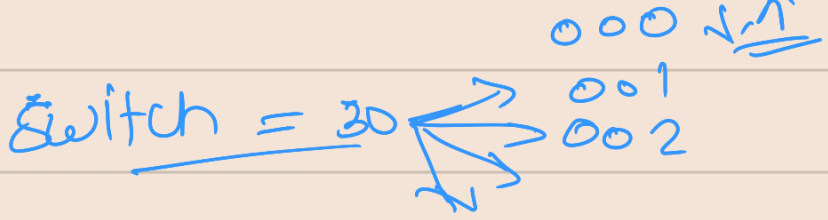
0 ✓
1 ✓

Features :-

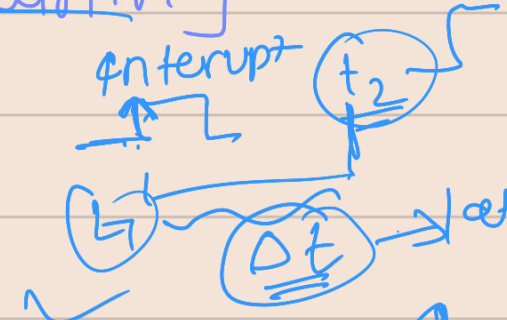
1) flexible exceptions & interrupt management
   → Level triggered interrupt request
   → Pulsed interrupt request

ii> Nested exceptions:

→ processing interrupt of high priority with In another interrupt of lower priority.
→ interrupt with high priority pre-empts the interrupt with Low priority resulting in interrupt in an interrupt known as interrupt nesting.



iii> 8 programmable priority levels
iv> 7 exception & 71 interrupts
v> Automatic state saving & restoring
vi> pre-emptive / Nested interrupt
vii> Tail chaining
viii> Deterministic: always 12 cycles or 6 cycles for tail chaining

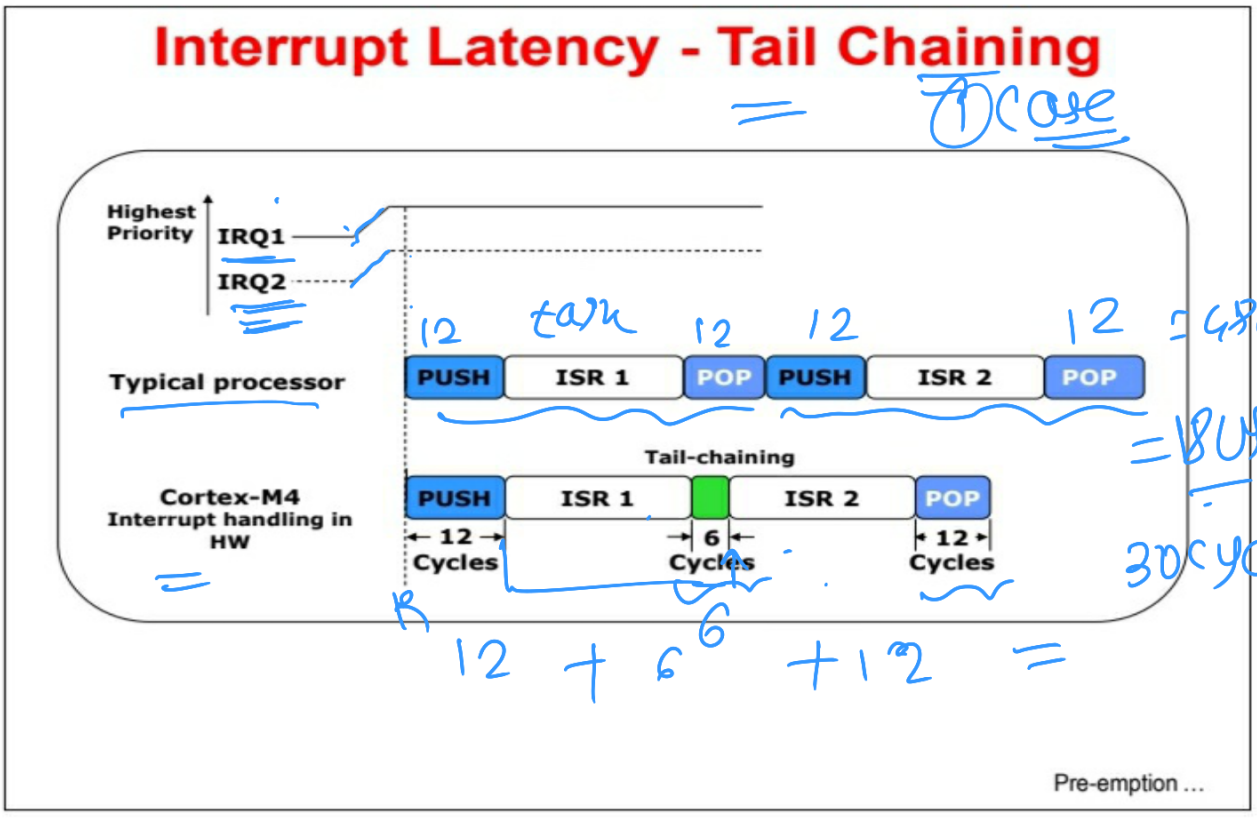


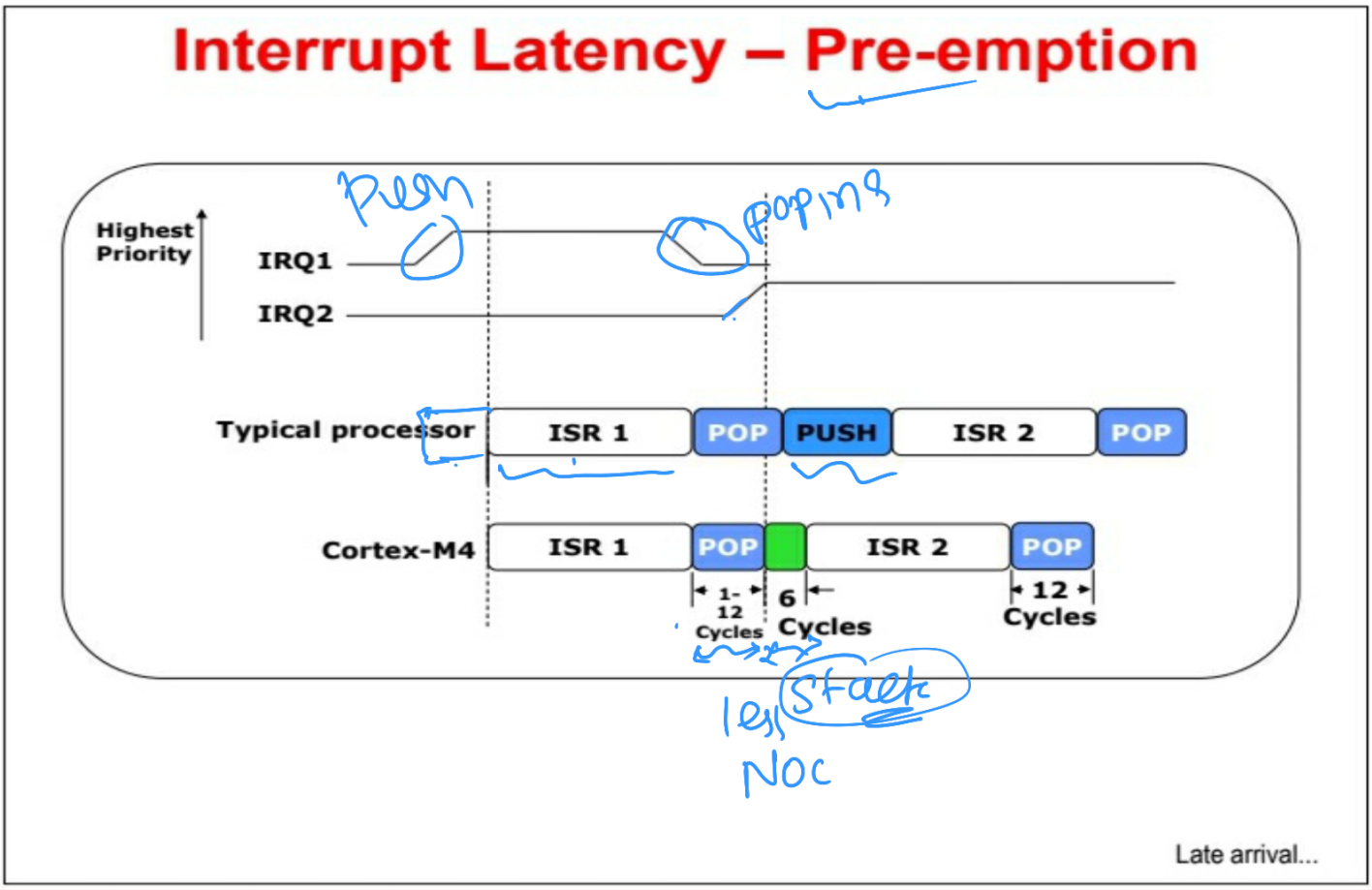
ix> Vectored exceptions
x> interrupt masking

Global level
- PRIMASK → disable interrupts
- BASEPRI → interrupt select mask which case below certain priority level.

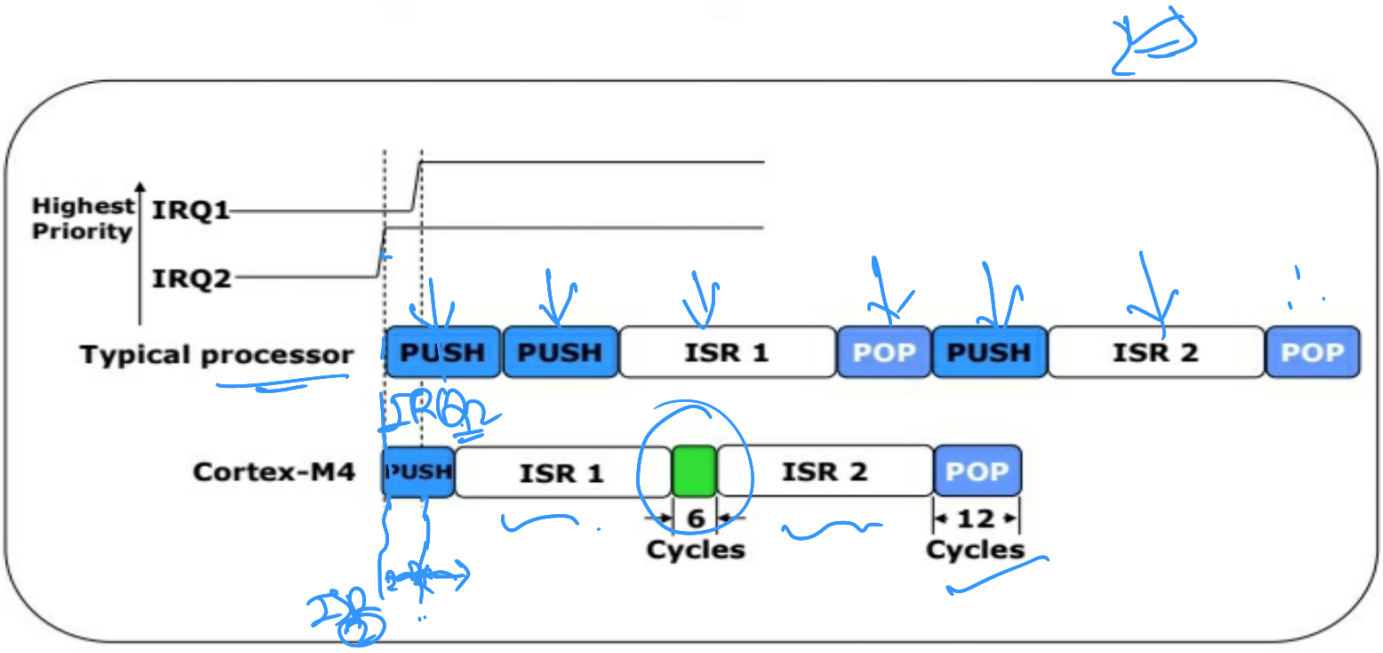
BASEPRI = 3 : {0, 1, 2} 3 ↓ ↑

* tail chaining :-



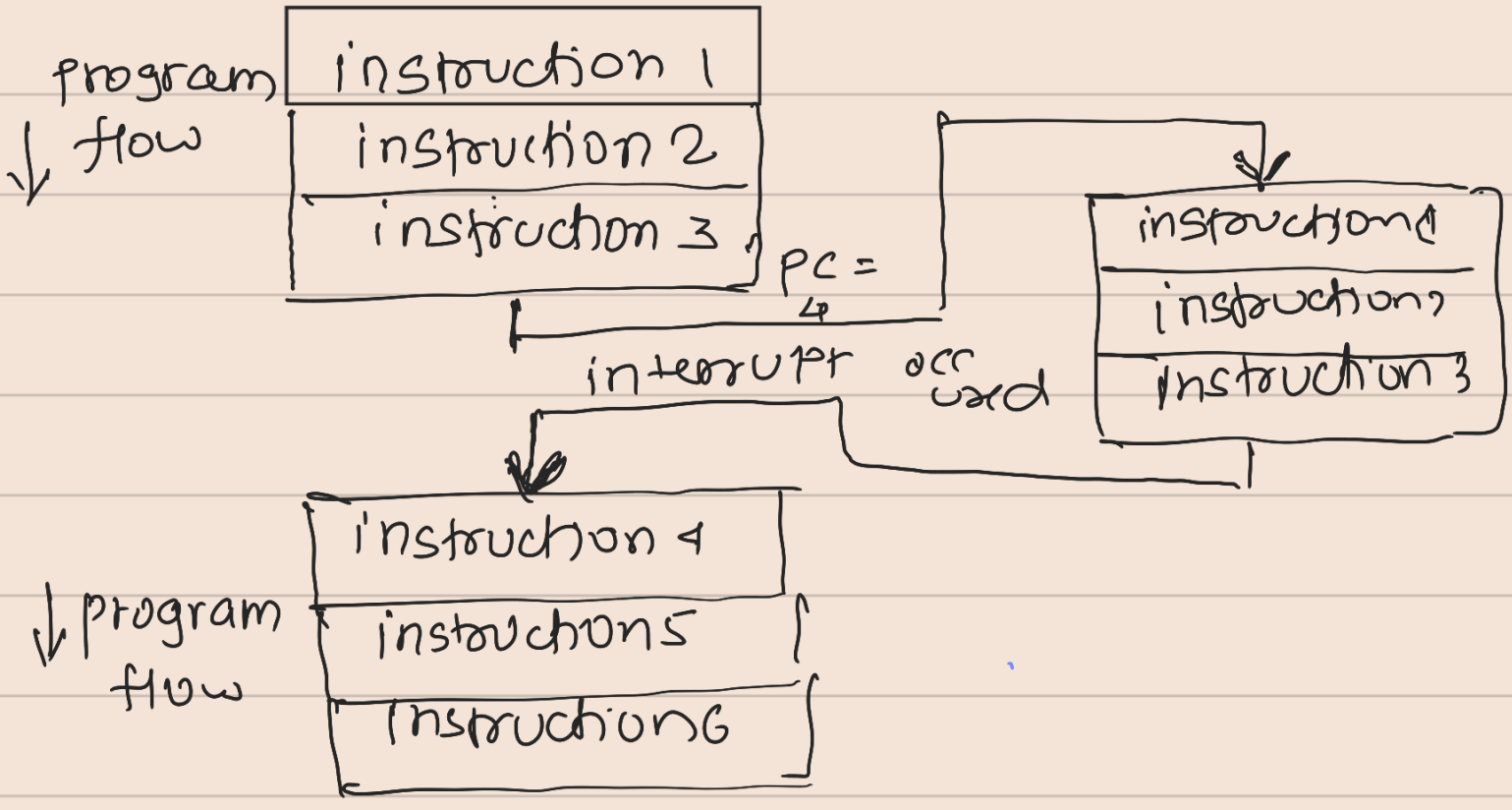


Interrupt Latency – Late Arrival

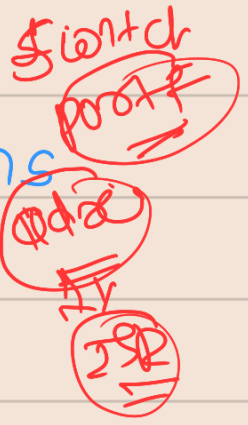


Interrupt handling...

① What happens when interrupt occur??



⊙ Vector Table :-



→ It is a table that contains vectors.

→ vectors means memory address.

→ address of what ??
address of exceptions & interrupts

→ In short contain address of ISR, int handler functions.

→ IVT = 154 entries in vector table
= 15 exception & 138 peripheral interrupts
= only 78 are available

* where IVT is stored :
→ code memory / flash memory
→ Starting from 0x0000_0000 ⇐
total 154 exceptions

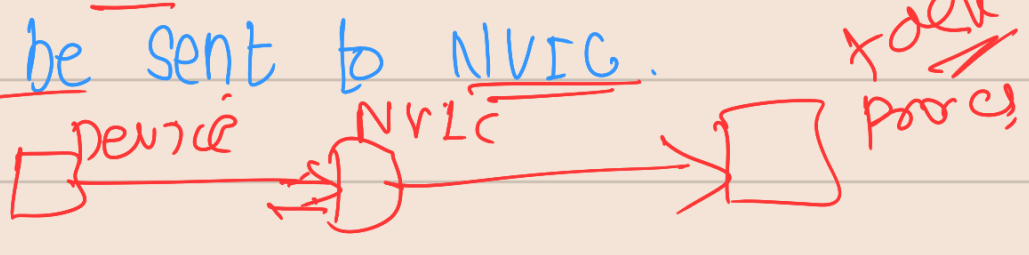
→ The vector table and interrupt service routine / exceptions are defined inside the startup file of a micro-controller.

→ 154 exceptions & interrupts so, 154 words of ROM or Code memory.

Exception number	IRQ number	Offset	Vector
16+n	n	0x0040+4n	IRQn
.		.	.
.		.	.
.		.	.
		0x004C	
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			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 → Program
		0x0000	Initial SP value

Flow :-

i) Interrupt A occurs, the interrupt request will be sent to NVIC.



ii> then NVIC decides to accept or reject based on it configuration.

iii> when NVIC accept, then it find the starting address of ISR or exception handler, which is stored in IVT (interrupt vector table)

iv> Pc will be loaded with address of handler and CPU starts execute the exception.

Relocation of vector table:-

→ Relocation is controlled by programmable register in NVIC called Vector table offset register (VTOR)

→ After reset VTOR = 0

VTOR =

© Configure and setup interrupt priority

PRIn Register Bit field	Interrupt
Bits 31:29	interrupt [4n+3]
Bits 23:21	interrupt [4n+2]
Bits 15:13	interrupt [4n+1]
Bits 7:5	interrupt [4n]

priority register bit field against interrupt sources

→ 3 bit wide ✓
i.e for an interrupt 8 priority levels

→ n = group number
can control priority levels for 4n to 4n+3 peripherals

→ For example:
interrupt handler with IRQ = 0
4[n] = 0
∴ n = 0
4[n] = 0
n = 0, 1, 2, 3

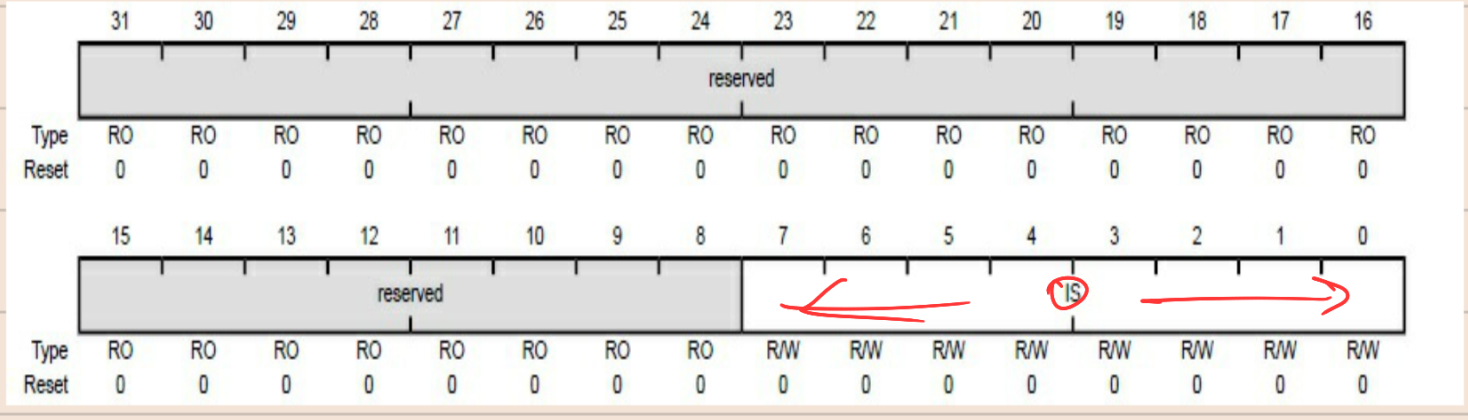
Hence same priority level register can be

vsed for configure [4n to 4n+3] interrupt whose IRQ

→ IRQ 0,1,2,3 GPIO (A), (B), (C), (D).

Registers:-

1) GPIO Interrupt sense register (GPIOIS):



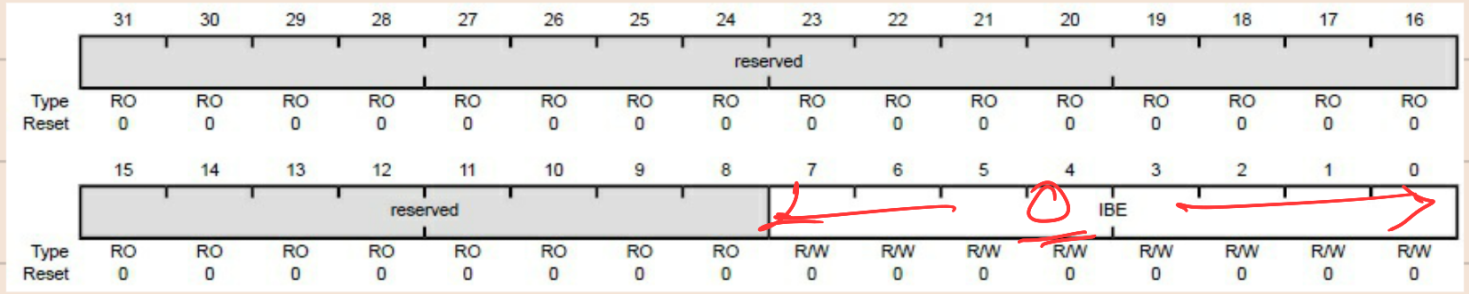
→ To select between edge or level triggered interrupts.

✓0 → The edge on corresponding pins detected

✓1 → The level on corresponding pin detected.

2) GPIO IBE (interrupt both edge)

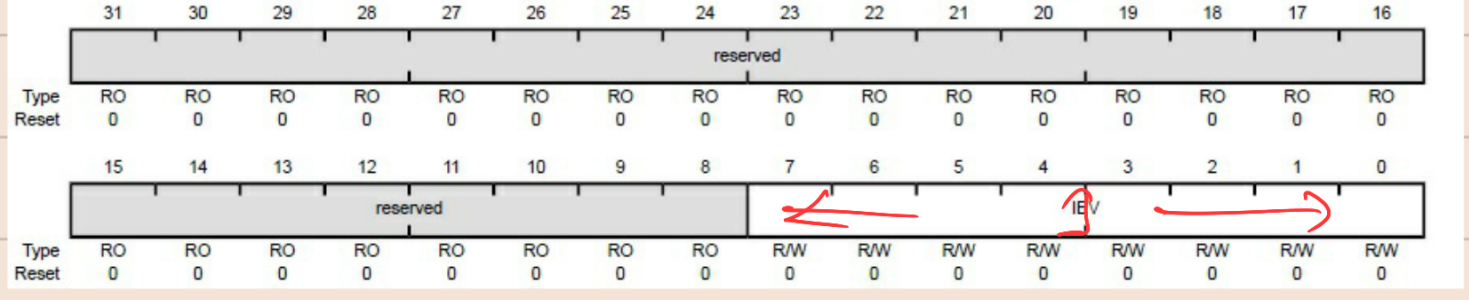
→ Allow both edges to cause interrupt



0 → interrupt generation is controlled by GPIOIBE register

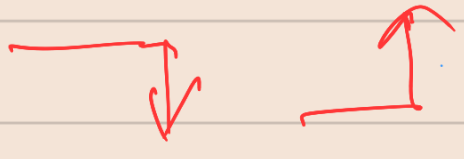
1 → Both edges of corresponding pins trigger an interrupt.

3) GPIO IEV (Interrupt Event register)



0 → falling edge

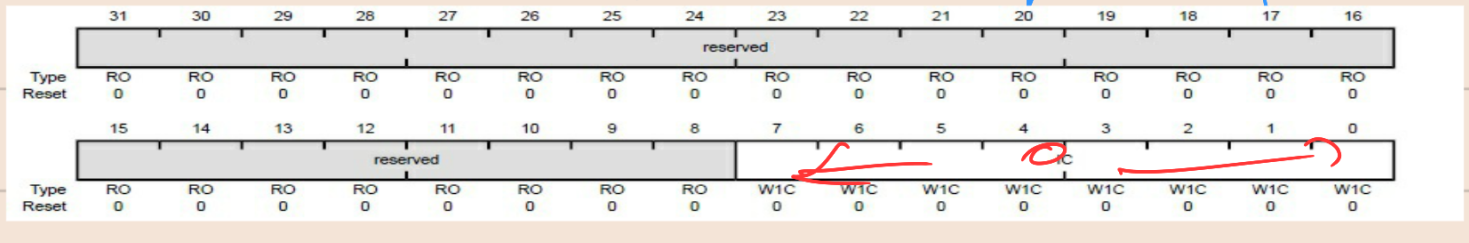
1 → rising edge



trigger

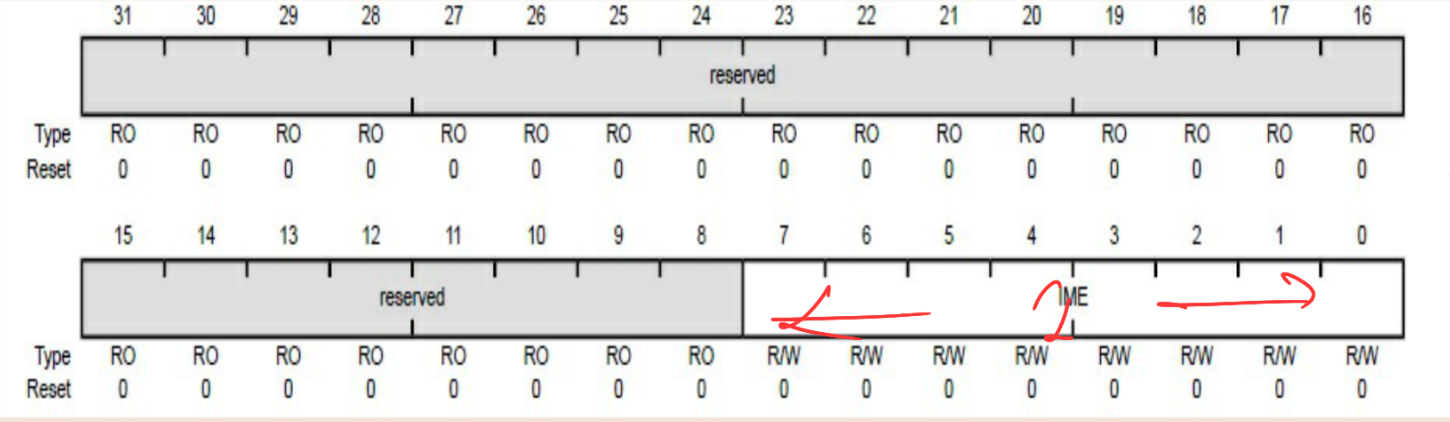
4) GPIOICR (interrupt clear register)

→ clears the interrupt for respective pin



5) GPIO IM (Interrupt Mask register)

→ In order to allow interrupt it should be unmasked so that the interrupt can be sent to interrupt controller



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								IME							
Type	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0 → mask ✓

1 → unmask ✓

6) NVIC IP (NVIC interrupt priority Register)

→ In order to configure this register we have to find IRQ Number.

ex: PORTF = 30 (30)

→ Each priority level register can handle upto 4 peripherals:

4n to 4n+3

PRI7 register

4n + x = 30

4(7) + 2 = 30

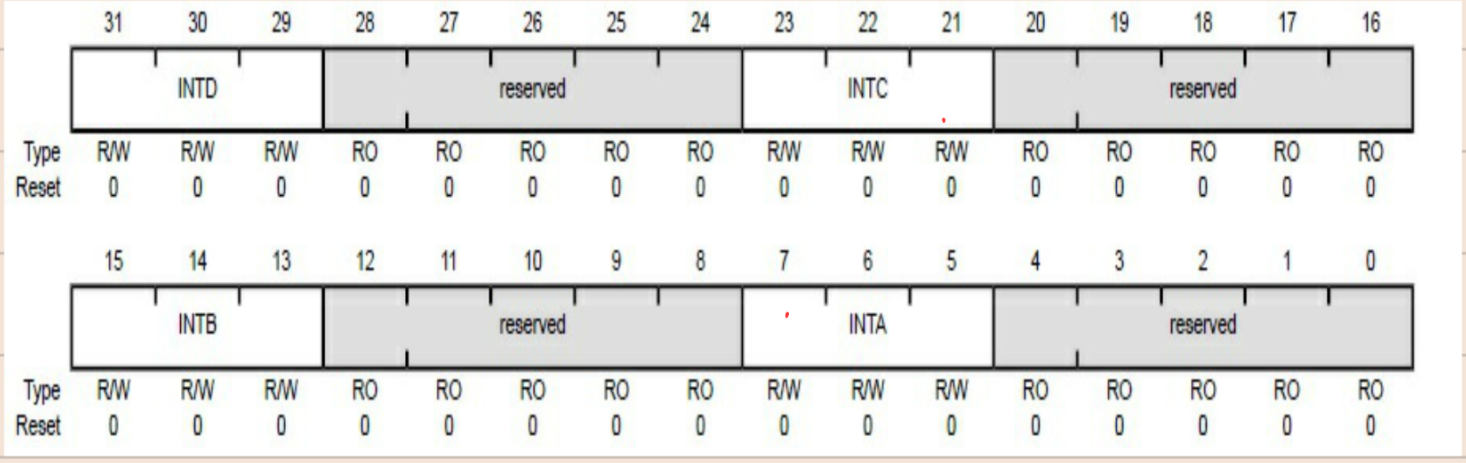
n = 7 ⇒ group

4n+2 ⇒

n = 7 (Group number)

bit field for 4n+2 = bit 23:21

3 bit → 8 priority levels



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	INTD			reserved					INTC			reserved				
Type	R/W	R/W	R/W	RO	RO	RO	RO	RO	R/W	R/W	R/W	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	INTB			reserved					INTA			reserved				
Type	R/W	R/W	R/W	RO	RO	RO	RO	RO	R/W	R/W	R/W	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

7) NVIC ISER [Interrupt Set Enable Register]

→

NVIC . . . .

ISER[0] = IRQ 0-31 ⇒

[1] = IRQ 31-64

GPIOf = 30

IRQ