ARM Cortex Core microcontrollers

6th NVIC

Scherer Balázs

Méréstechnika és
Információs Rendszerek
Tanszék

ARM7, ARM9 interrupt handling

- ARM7, ARM9 two interrupt lines
  - IRQ: Normal priority IT
  - FIQ: Fast IT (own small set of registers)
  - Vector interrupt controller is vendor specific
  - Non deterministic interrupt handling
  - No nested interrupt support
- The Cortex M series gives solution to the problems of ARM7 interrupt handling

Cortex M3 NVIC

- Nested Vector Interrupt Controller
  - Vector independent standard peripheral
    - Easy porting of applications from one micro to an other
  - Deterministic interrupt handling
    - Long instructions can be interrupted
  - Support for nested vector interrupt controll
  - The number of external vectors are depend on the microcontroller vendors
    - The NVIC can have: 1 non-maskable + 15 internal + 240 external IT line
      - Usually 30-60 external lines are used

The properties of NVIC

- The vector table should be specified
- The 0x00000004 is the reset vector
  - The 0x00000000 is the stack pointer
- The first 15 interrupt is for the core
- Then the vendor specific ones

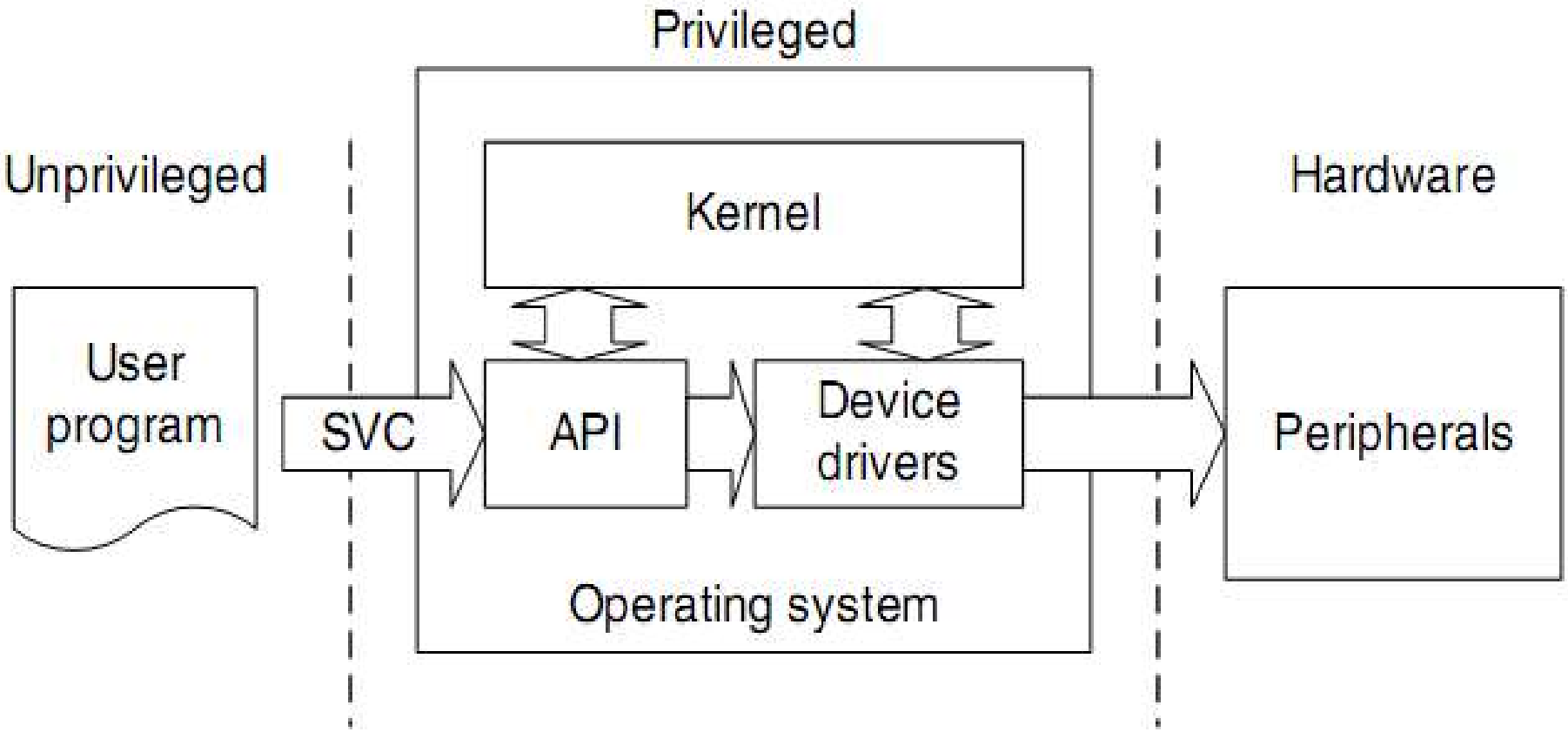
The NVIC vector table

- The reset vector starts from the 0x00000004 address
  - The stack pointer is stored at the 0x00000000 address: enables the usage of C language very early

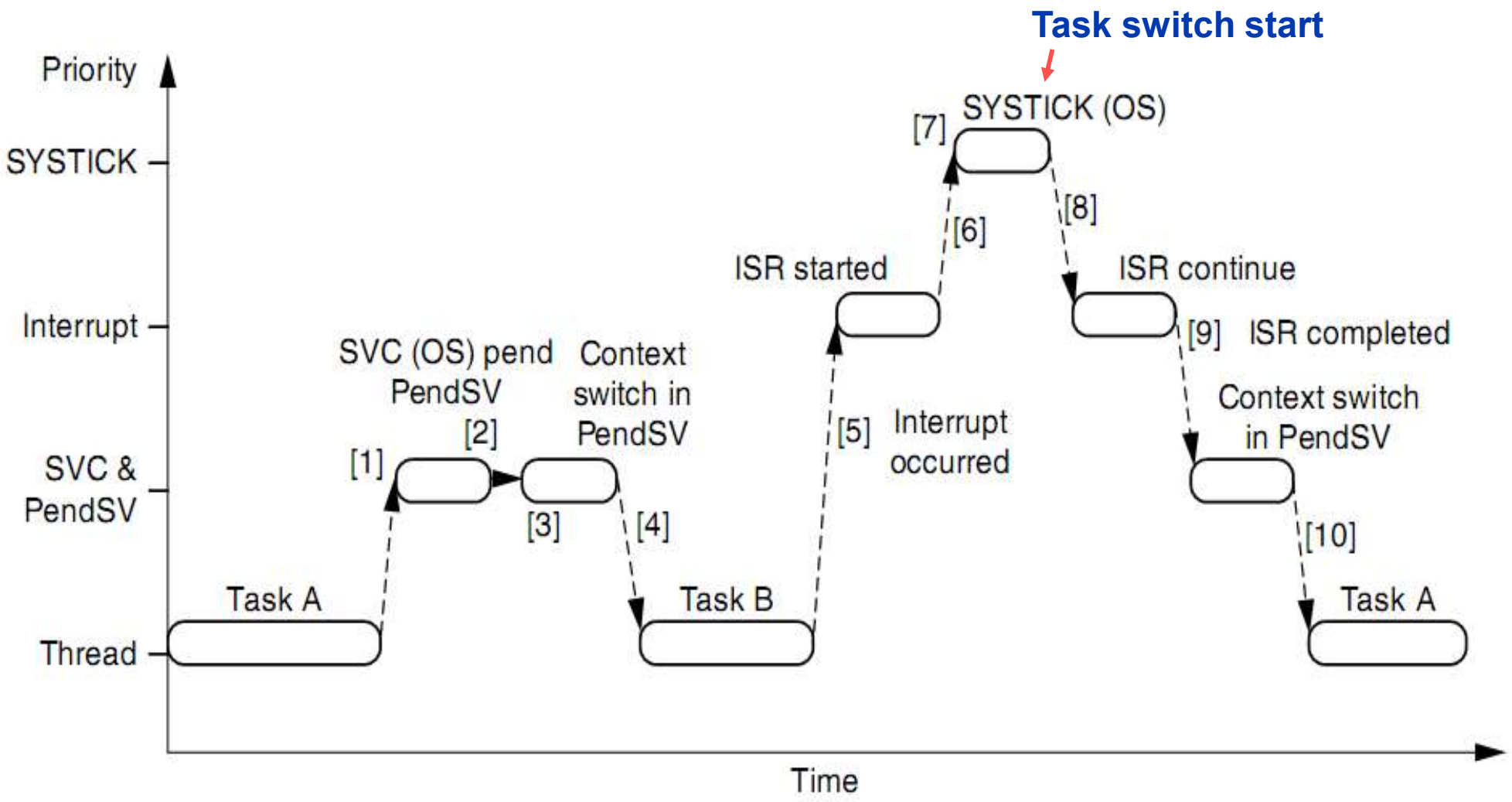
No.	Exception Type	Priority	Type of Priority	Descriptions
1	Reset	-3 (Highest)	fixed	Reset
2	NMI	-2	fixed	Non-Maskable Interrupt
3	Hard Fault	-1	fixed	Default fault if other hander not implemented
4	MemManage Fault	0	settable	MPU violation or access to illegal locations
5	Bus Fault	1	settable	Fault if AHB interface receives error
6	Usage Fault	2	settable	Exceptions due to program errors
7-10	Reserved	N.A.	N.A.	
11	SVCall	3	settable	System Service call
12	Debug Monitor	4	settable	Break points, watch points, external debug
13	Reserved	N.A.	N.A.	
14	PendSV	5	settable	Pendable request for System Device
15	SYSTICK	6	settable	System Tick Timer
16	Interrupt #0	7	settable	External Interrupt #0
.....	.....	.....	settable	.....
256	Interrupt#240	247	settable	External Interrupt #240

Manufacturer dependent

The purpose of the SVC



A PendSV kezelése

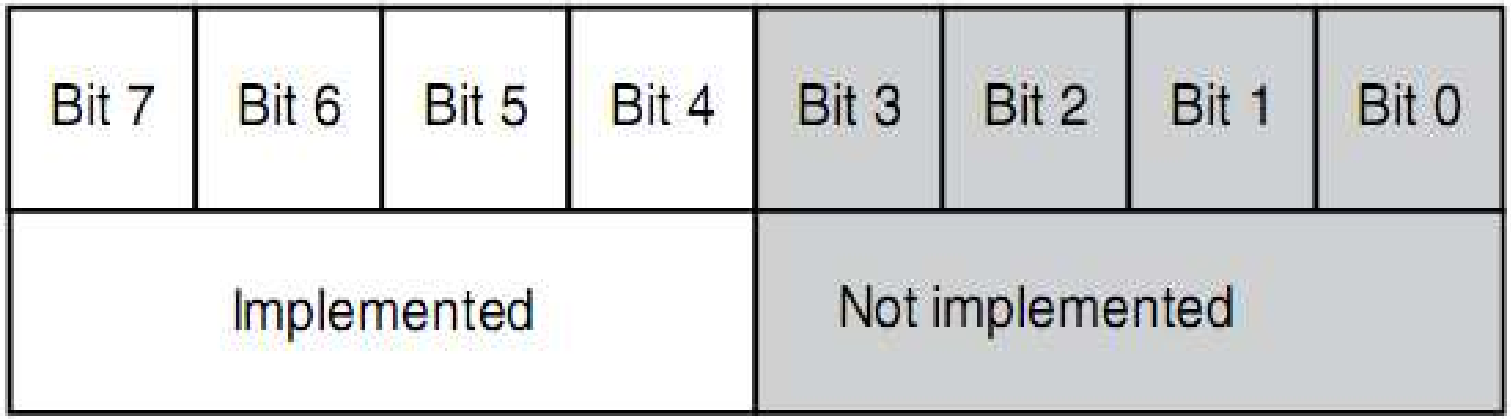


Base registers

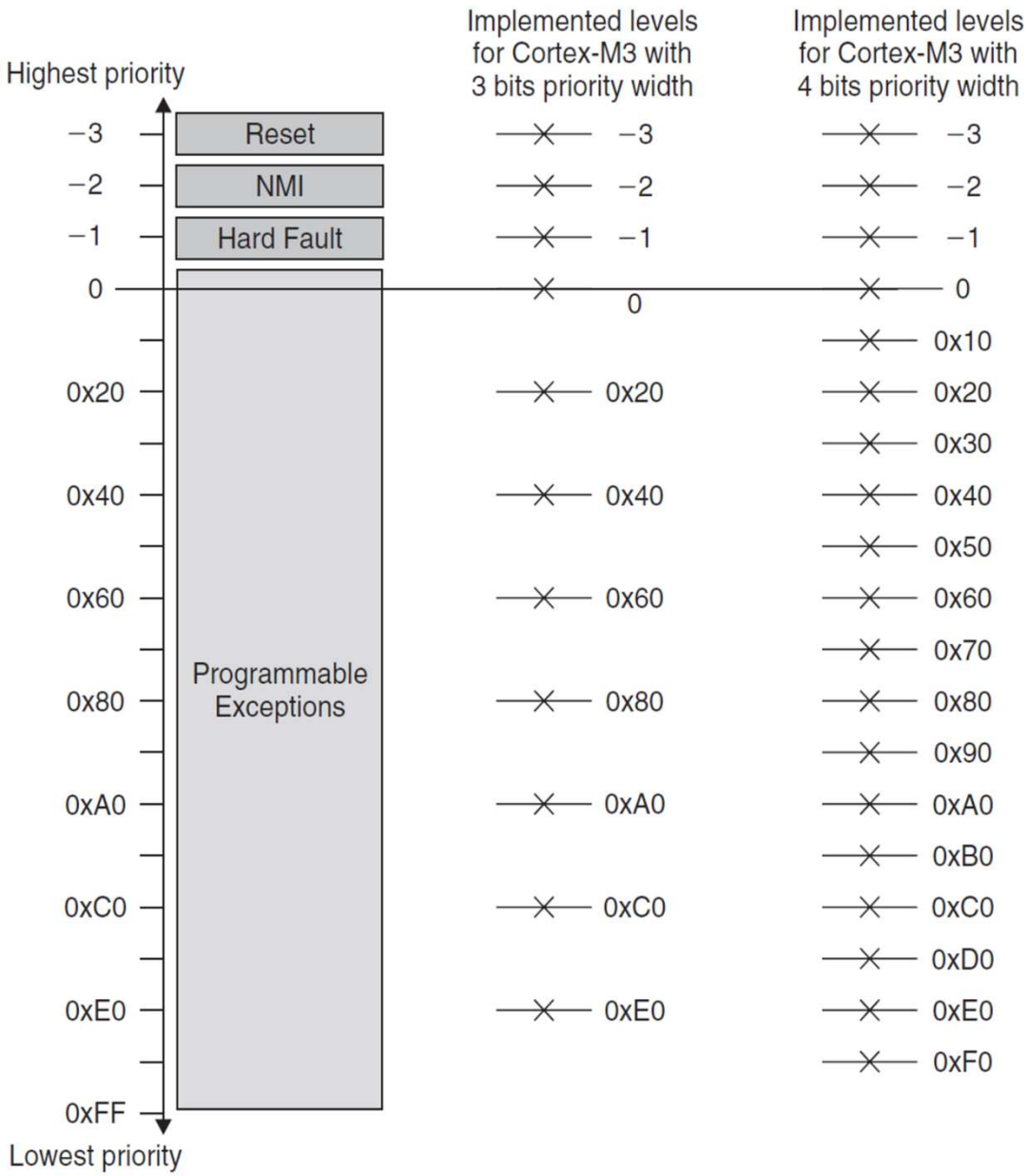
- **Interrupt enable** and **Clear enable** registers
  - SETENA0-n/CLRENA0-n
    - 32-bit enabling and clearing register
- **Interrupt set pending** and **Clear pending**
  - SETPEND0-n/CLRPEND0-n
    - 32-bit registers, usable it identify pending interrupts

Priority handling

- The basic internal IT-s has fix priority
- Other IT has setable priority
  - max. 8 bit, min. 3 bit priority regiszter
  - Usually 4 bits are used to enable versatile priority handling and to reduce system cost.
  - The implementation start from the MSB bits (easyer to port)

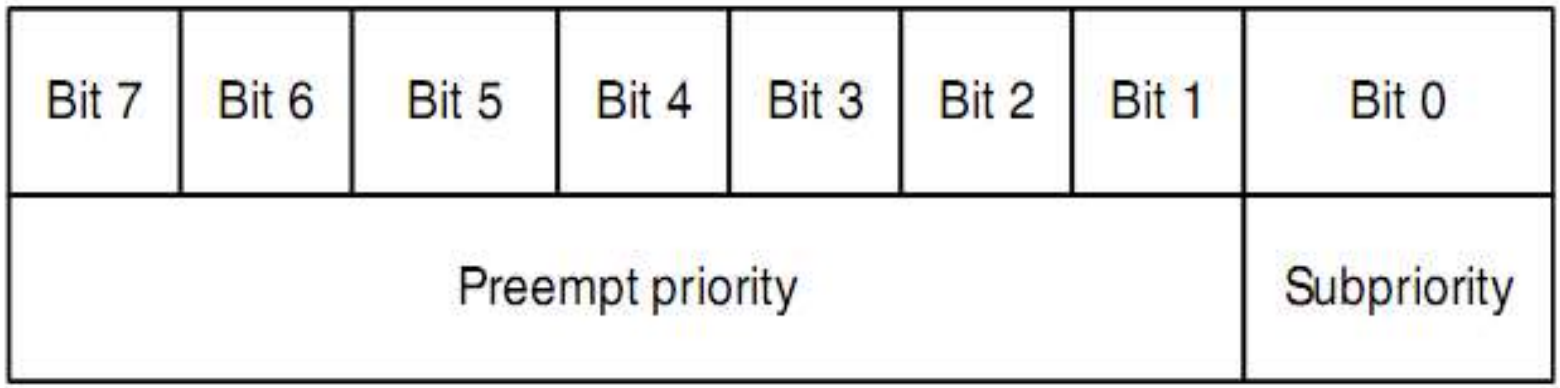


Prioritás handling



Preempt priority and Subpriority

- The 8 bit priority register has 127 preemption levels
- Subprioritás
  - Same preemption level, but different priority
  - PRIGROUP register



Preempt priority and Subpriority

- The 8 bit priority register has 127 preemption levels
- Subprioritás
  - Same preemption level, but different priority
  - PRIGROUP register

PRIGROUP (3 Bits)	Binary Point (group.sub)		Preempting Priority (Group Priority)		Sub-Priority	
			Bits	Levels	Bits	Levels
011	4.0	gggg	4	16	0	0
100	3.1	gggs	3	8	1	2
101	2.2	ggss	2	4	2	4
110	1.3	gsss	1	2	3	8
111	0.4	ssss	0	0	4	16

Additional NVIC registers and options

- Interrupt masks
  - PRIMASK: mask everything except the faults
  - FAULTMASK: mask the faults as well to priority -1
  - BASEPRI: Masking below a given priority
- Vector Table Offset Regiszter
  - The Vector table can be relocated

Table 7.7 Vector Table Offset Register (Address 0xE000ED08)

Bits	Name	Type	Reset Value	Description
29	TBLBASE	R/W	0	Table base in Code (0) or RAM (1)
28:7	TBLOFF	R/W	0	Table offset value from Code region or RAM region

Additional NVIC registers and options

- Interrupt masks
  - PRIMASK: mask everything except the faults
  - FAULTMASK: mask the faults as well to priority -1
  - BASEPRI: Masking below a given priority
- Vector Table Offset Regiszter
  - The Vector table can be relocated
  - Helping the implementation of a bootloader

Table 7.7 Vector Table Offset Register (Address 0xE000ED08)

Bits	Name	Type	Reset Value	Description
29	TBLBASE	R/W	0	Table base in Code (0) or RAM (1)
28:7	TBLOFF	R/W	0	Table offset value from Code region or RAM region

The effect of the interrupt I.

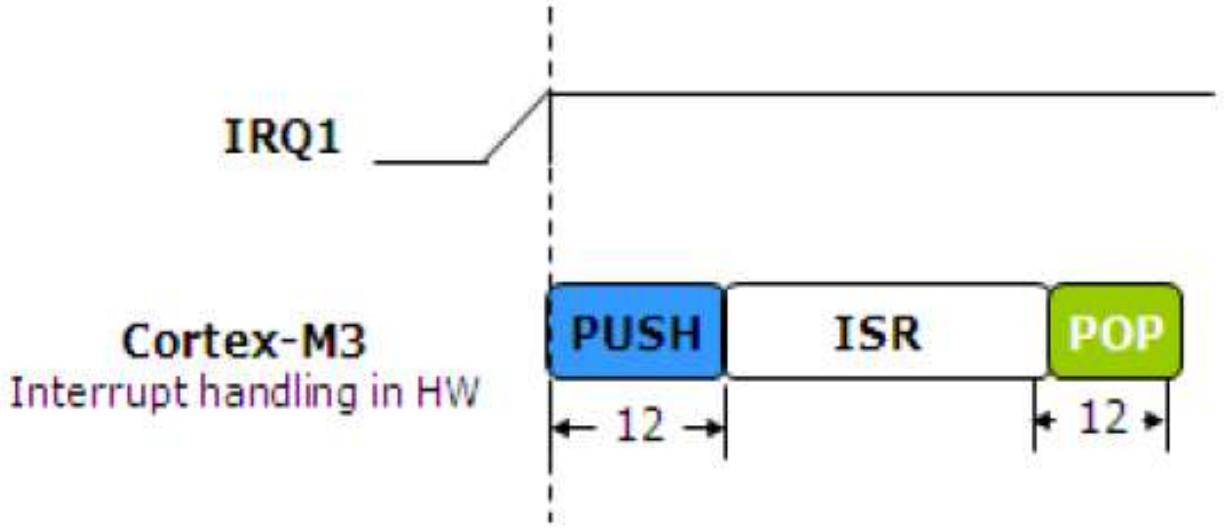
- The M3 core goes to handler mode and saves the register bank to the stack
  - It is done in a micro code, there is no need for programmer's interaction
  - Saved registers are
    - Program Status Register
    - Program Counter
    - Link Register
    - R0 – R3 registers (these are used during c function calls, and the R12 register (compiler support register)
    - Process Stack switched to Main Stack if necessery
  - The IT vector addres is fetched from the program memory

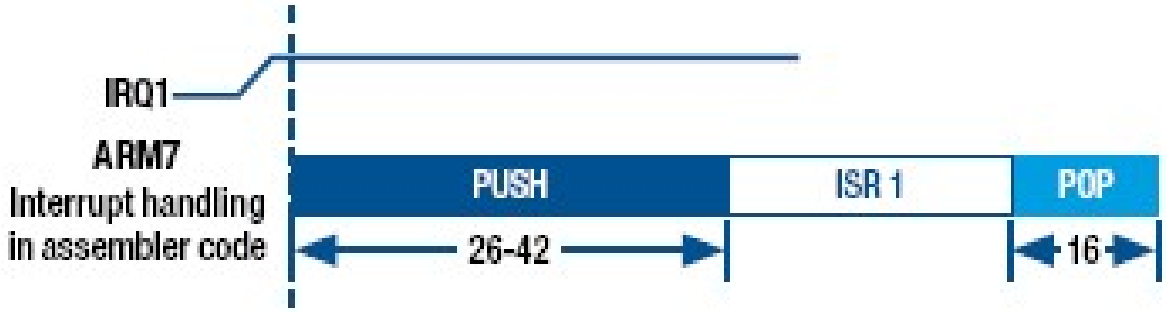
The effect of the interrupt II.

- After fetching the start address register refressing is done
  - Stack Pointer
  - Link Register
  - Program Counter
  - Interupt Program Status Register: IPSR
- 12 cycle after the IT event the serving of the IT is started
- The return from IT is also 12 cycles
  - There is no special instruction for return from interrupt

NVIC operation summary

The NVIC will respond to an interrupt with a latency of just six cycles. This includes a microcoded routine to automatically push a set of registers onto the stack.



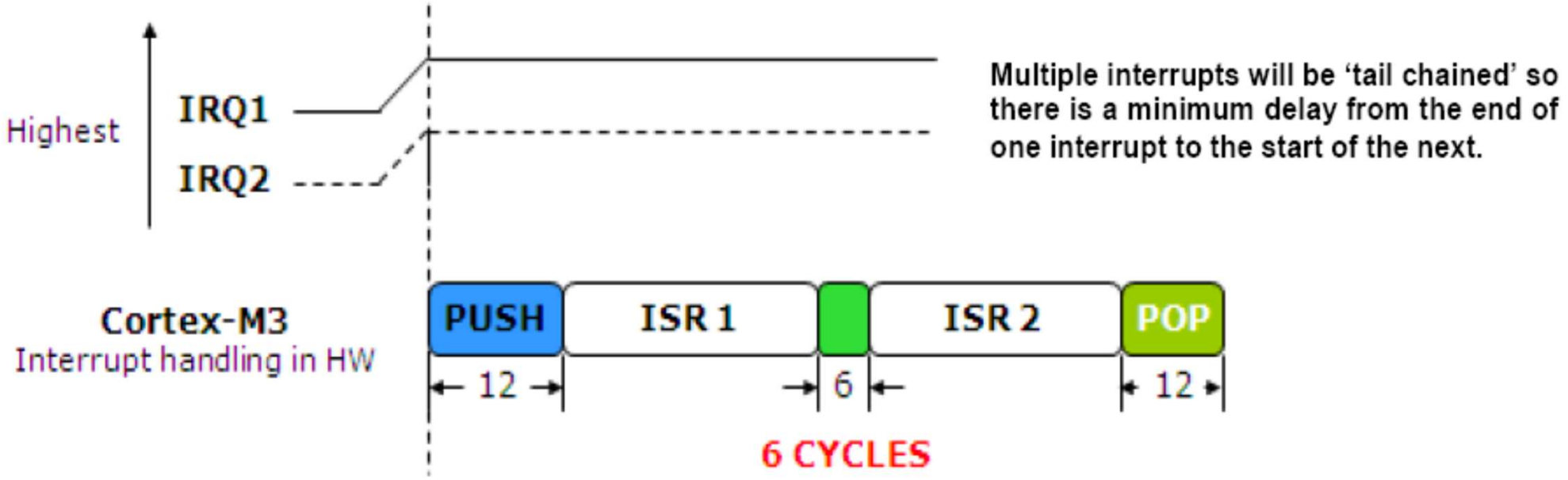


Behavior in case of multiple interrupts I.

- In real-time system it is important to control the priority of interrupts
  - Traditional hard real-time systems do not prefer interrupts, because interrupts can delay other interupts and the system can be timing calculation too chaotic
- Preemptive IT handling
  - Higher priority interrupt can preempt a lower priority one
  - The registers of the lower priority IT is saved, and after 12 cycles the higher priority IT starts

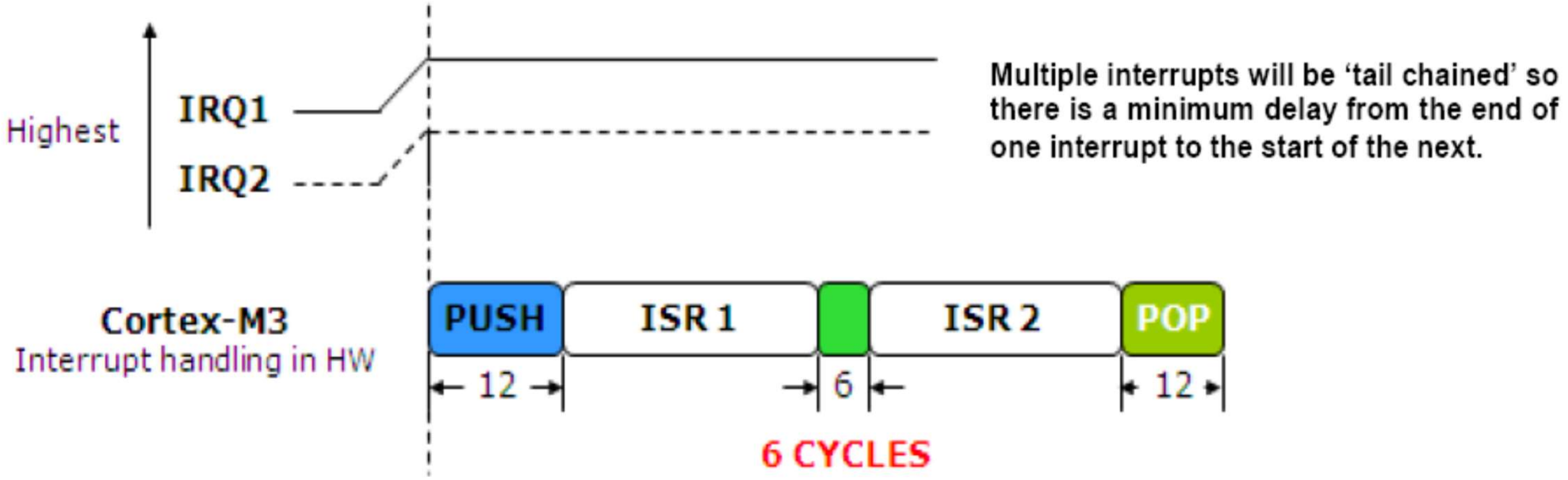
Behavior in case of multiple interrupts II.

- Tail chaining: executing IT after IT with small delay
  - ARM7 didn’t have this feature (42 cycle operation, POP(16) and PUSH(26)



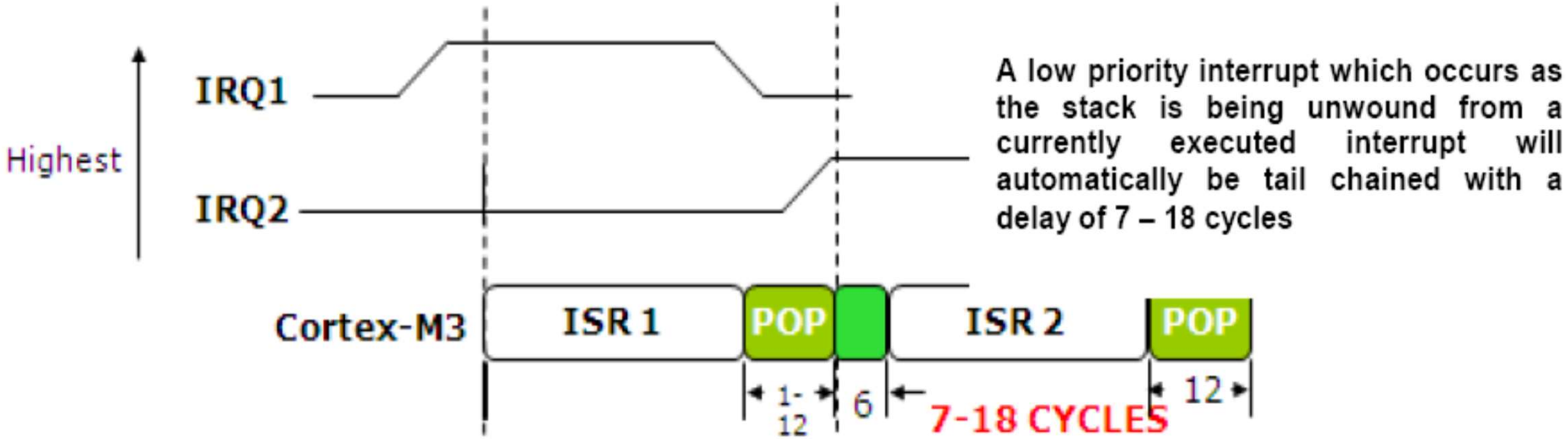
Behavior in case of multiple interrupts II.

- Tail chaining: executing IT after IT with small delay
  - ARM7 didn’t have this feature (42 cycle operation, POP(16) and PUSH(26)



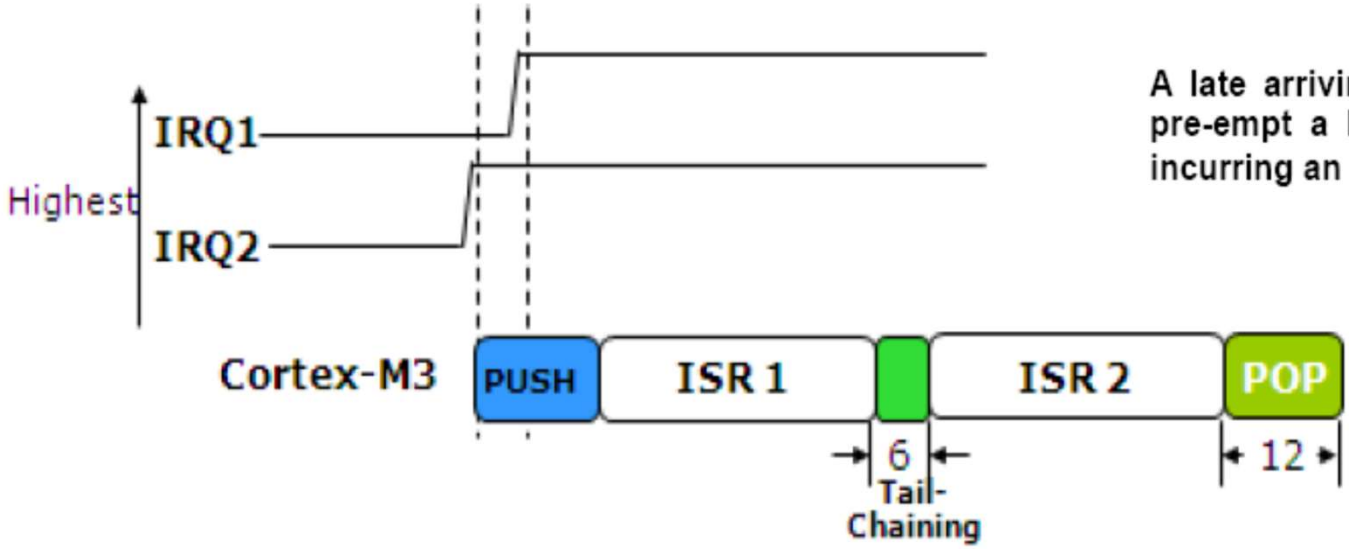
Behavior in case of multiple interrupts III.

- Interrupting the return from interrupt
  - The POP operation is interrupted



Behavior in case of multiple interrupts IV.

- Late arriving
  - Can tolerate 6 clock cycles, which is the time required to save the state of the main thread



A late arriving high priority interrupt will pre-empt a low priority interrupt without incurring an additional stacking overhead.

NVIC programming:

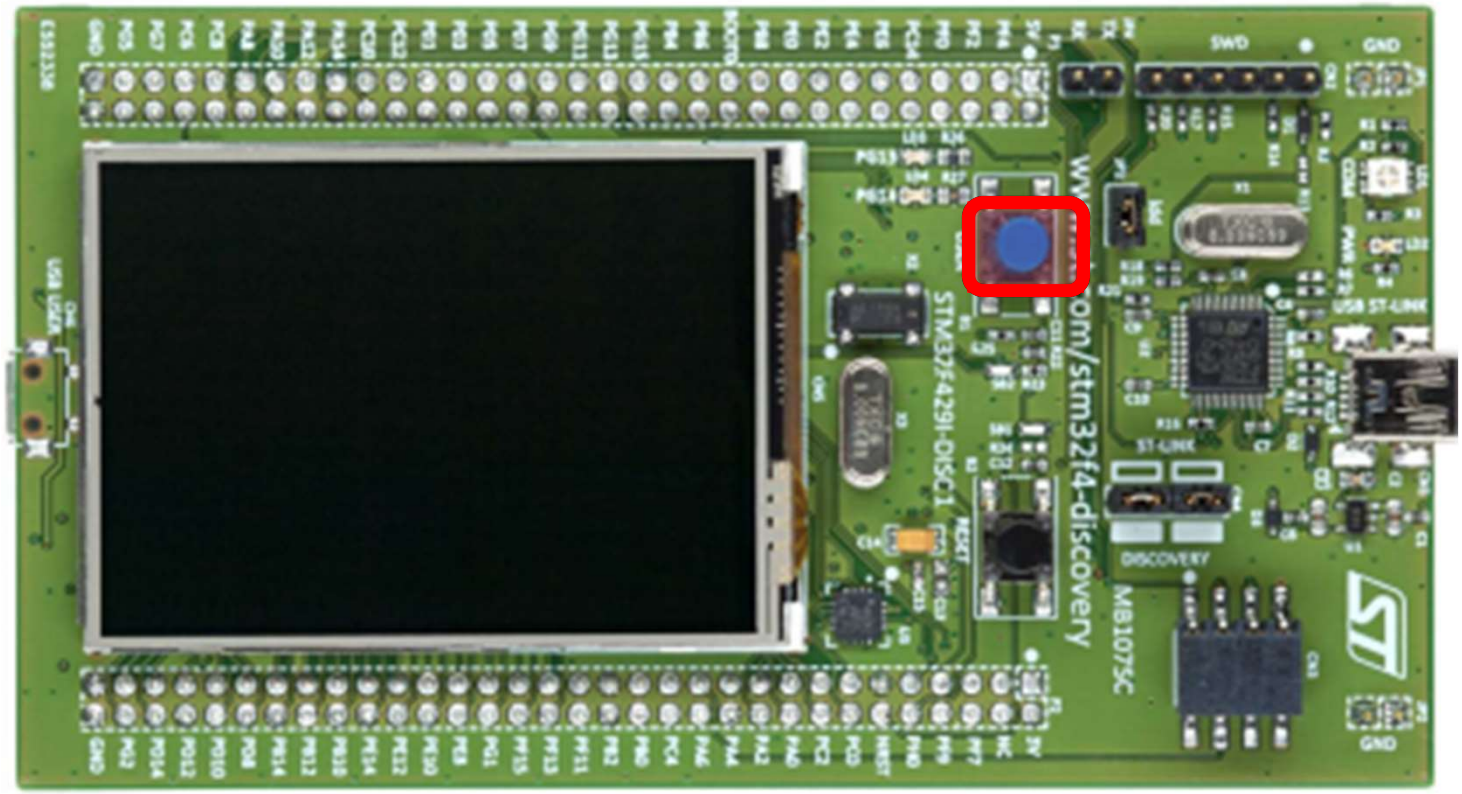
Push button IT

Implementaion

- 1st step: GPIO pin configuration
  - GPIO which PIN is used as Push button input
  - GPIO pin configuration
  - Connecting EXTI to that pin
  - EXTI configuration
- 2nd step: IT configuration
  - Enabling the interrupt line
  - Writing the IRQ handler

Push button on the STM32F429 Disc1

- Chapter 6.6: PA0 pin can be used as User button



PIN configuration

- PA0 as input, and important to pull it down due to schematic drawing
  - GPIO Init

```
GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_HIGH;
GPIO_InitStruct.Pin = LL_GPIO_PIN_0;
GPIO_InitStruct.Alternate = LL_GPIO_AF_0;
GPIO_InitStruct.Pull = LL_GPIO_PULL_DOWN;
LL_GPIO_Init  (GPIOA,&GPIO_InitStruct);
```

External interrup source configuration

- Every PIN on an STM32 micro can be used as external interrupt, but there are restictions
  - Only one pin can be used for one EXTI source
  - The EXTI0 can accept only input from PA0, PB0, PC0 …
- System Control has the appropriate API

```
LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA ,LL_SYSCFG_EXTI_LINE0 );
```

Configuring the External IT function

- Configuring the EXTI peripheral block
  - Falling/rising edge
  - Enabling

```
LL_EXTI_InitTypeDef EXTI_InitStruct;
EXTI_InitStruct.LineCommand = ENABLE;
EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_0;
EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_FALLING;
LL_EXTI_Init  (&EXTI_InitStruct);
LL_EXTI_EnableIT_0_31 (LL_EXTI_LINE_0);
```

Interrupt configuration

- NVIC IT number: device.h
- If needed priority setting
- Enabling

```
NVIC_EnableIRQ(EXTI0_IRQn);
```

Writing the interrupt handler

- Startup_device.h has the prototype for all IT sources, we need to overwrite the one we need

```
.weak       EXTI0_IRQHandler
.thumb_set EXTI0_IRQHandler,Default_Handler
```

Writing the interrupt handler

- Startup_device.h has the prototype for all IT sources, we need to overwrite the one we need

```
void EXTI0_IRQHandler(void)
{
    if(LL_EXTI_IsActiveFlag_0_31 (LL_EXTI_LINE_0)) // Not needed
    {
        printf(TERM_BRED "Push Button IT \r\n" TERM_DEFAULT);
        LL_EXTI_ClearFlag_0_31  (LL_EXTI_LINE_0);
    }
}
```