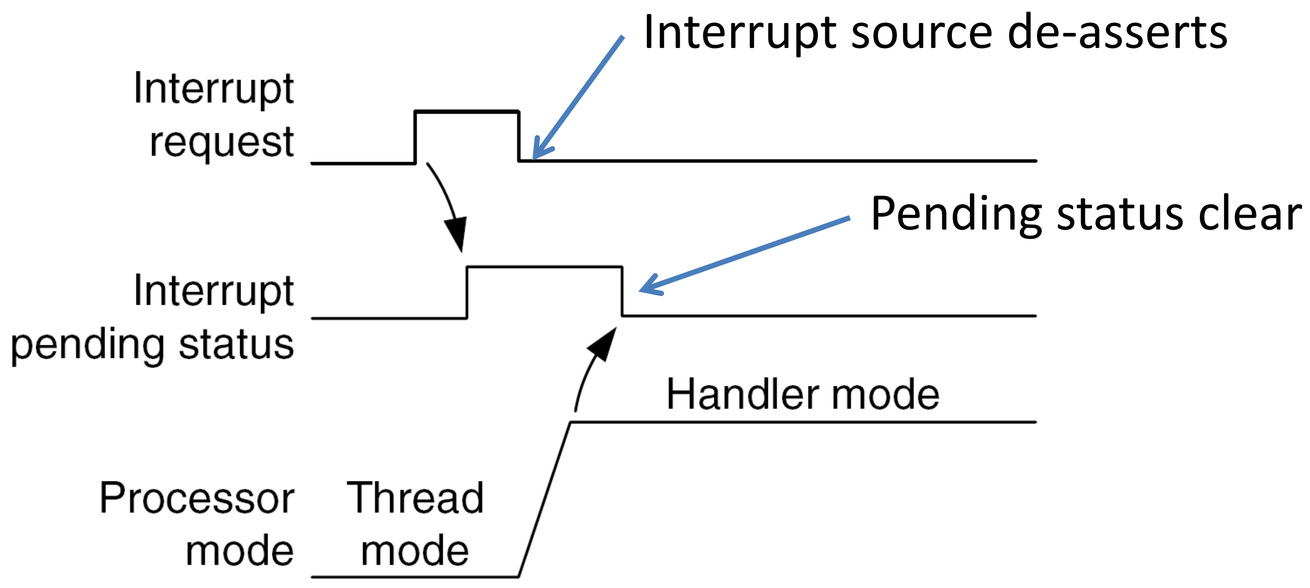
Microprocessor and Interfacings

Lec_3 Interrupt Part2

This lecture note was adapted from the following materials:

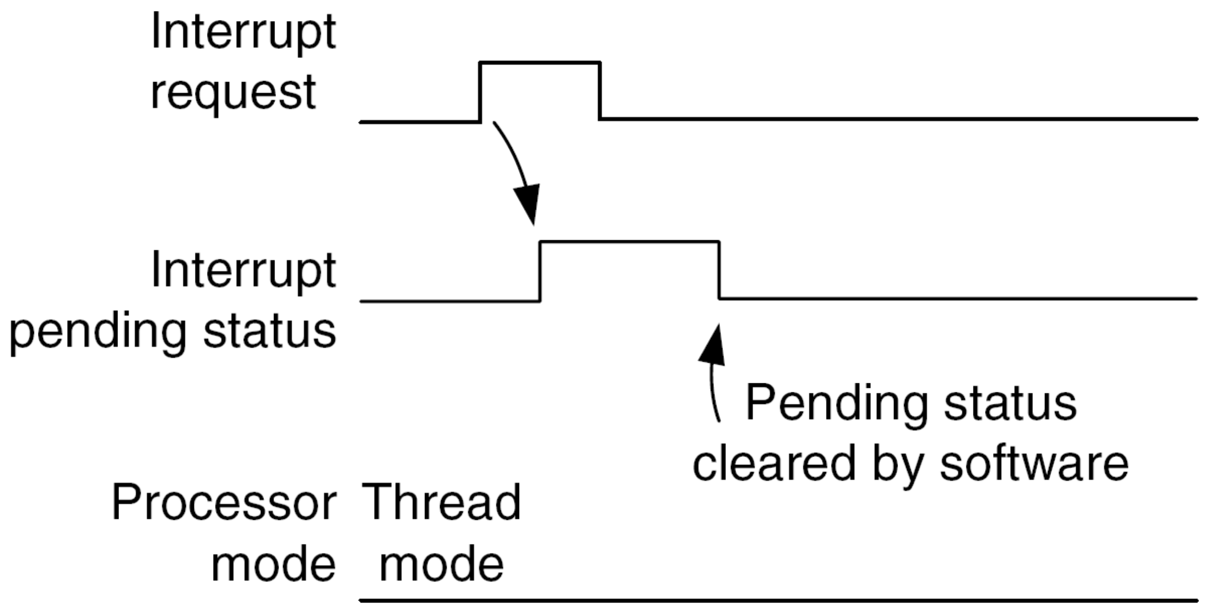
1) The Definitive Guide to the ARM Cortex-M3 by Joseph Yiu
2) UM10360 LPC176x/5x User manual Rev. V3.1 by NXP
3) Lecture Note: Embedded Systems and Applications by Cristinel Ababei
4) Computer Organization and Embedded Systems 6th edition by Carl Hamacher and et. al

Interrupt Behavior: Interrupt Pending



- When an interrupt input is asserted, it will be pended, which means **it is put into a state of waiting for the processor to process the request.**
- Even if the interrupt source **de-asserts** the interrupt, the pended interrupt status will still cause the interrupt handler to be executed when the priority is allowed.
- Once the interrupt handler is started, the pending status is cleared automatically

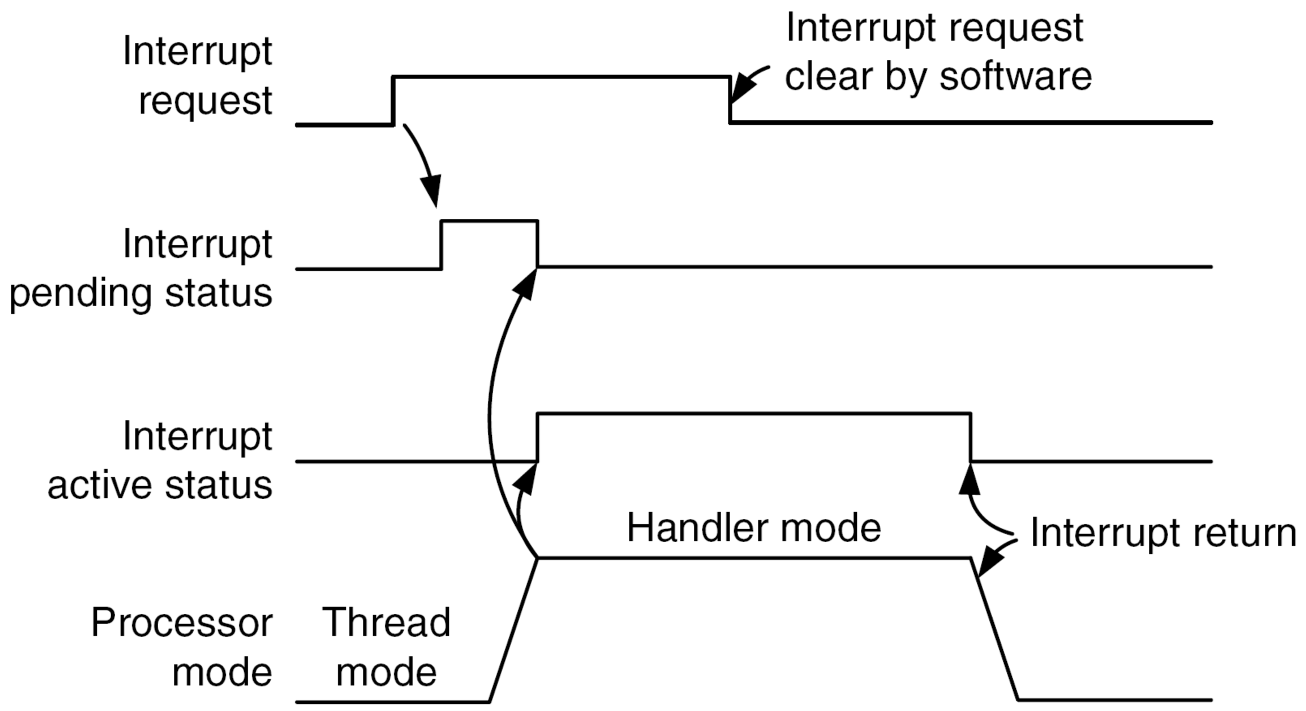
Interrupt Behavior: Interrupt is cancelled



Interrupt Pending Cleared Before Processor Takes Action

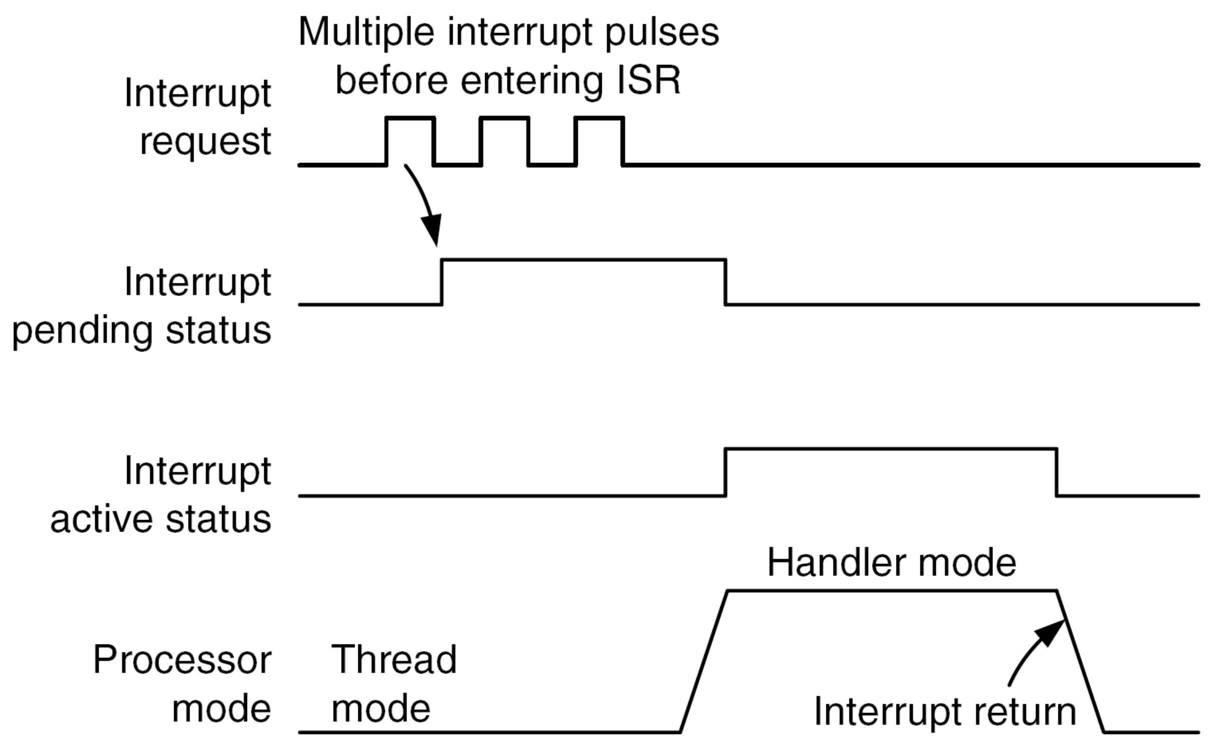
For example, the interrupt was not taken immediately because PRIMASK/FAULTMASK is **set to 1**, and the **pending status was cleared** by **software** writing to NVIC interrupt control registers

Interrupt Behavior: Interrupt becomes active



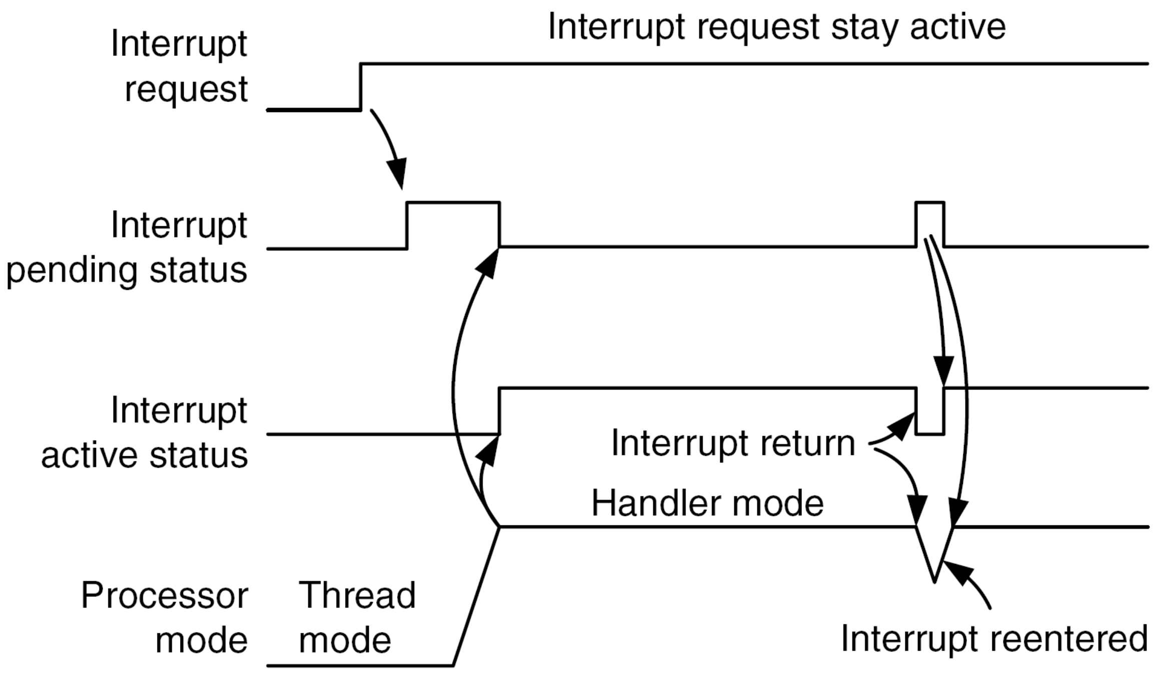
- Interrupt active status set as processor enters handler
- When interrupt is active, we **can not start** processing **the same** interrupt again until the ISR is terminated

Interrupt Behavior: Multiple Interrupt Pulses



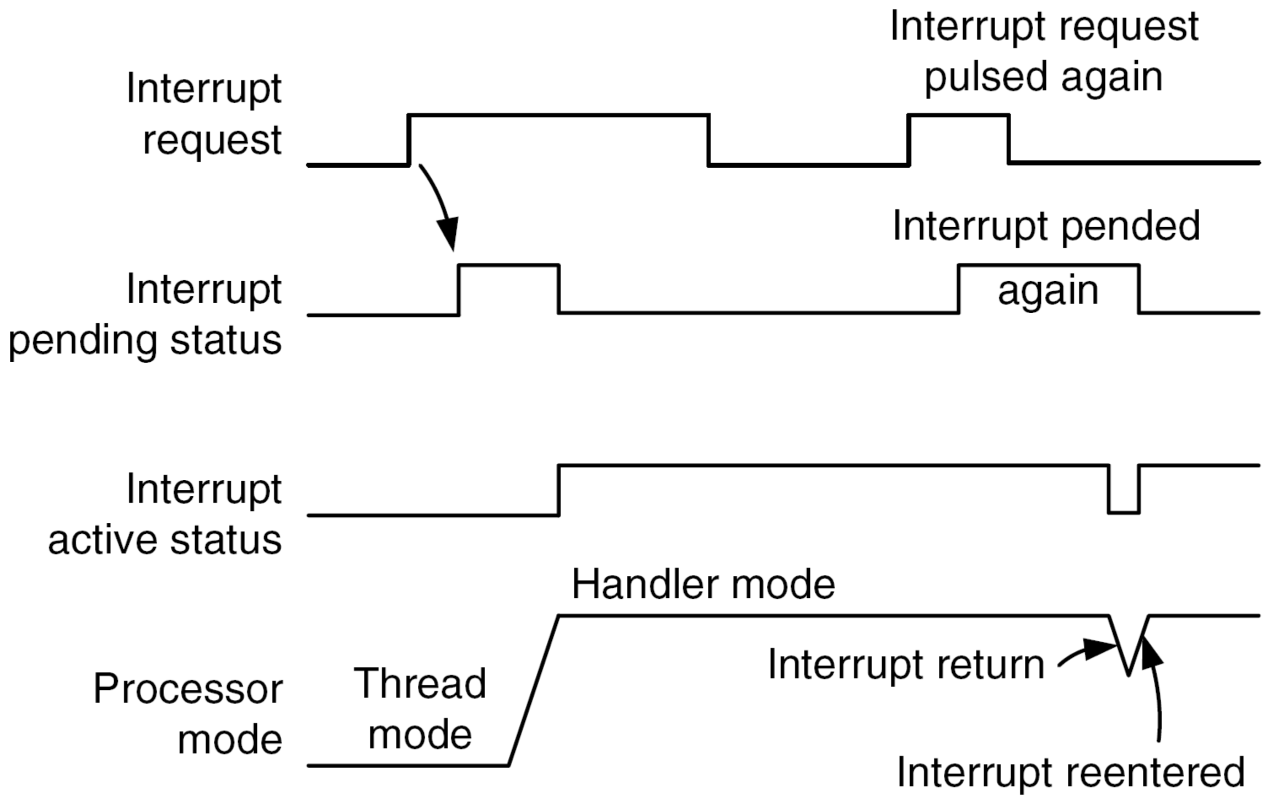
If an interrupt is pulsed several times before the processor starts processing it, it will be **treated as one single** interrupt request.

Interrupt Behavior: Continuous Interrupt Request



If an **interrupt source** continues to hold the interrupt request signal active, the interrupt **will be pended again** at the end of the interrupt service routine

Interrupt Behavior: Request during its being service



If an interrupt is de-asserted and then pulsed again **during the interrupt service routine**, it will be pended again.

Priority Level for Exceptions

- NVIC supports up to 256 levels of priority

❑ Divided to **2 priority groups**

1. **Preempt priority level**:

   ❖ Defines whether an interrupt can take place when processor is already running another interrupt handler.

   ❖ Support up to 128 levels

2. **Sub priority level:**

   ❖ Used only when two exceptions with the same preempt priority level occurred at the same time.

   ❖ Sub priority levels = total # of priority / # preempt

Priority Level for Exceptions

- NVIC supports up to 256 levels of priority

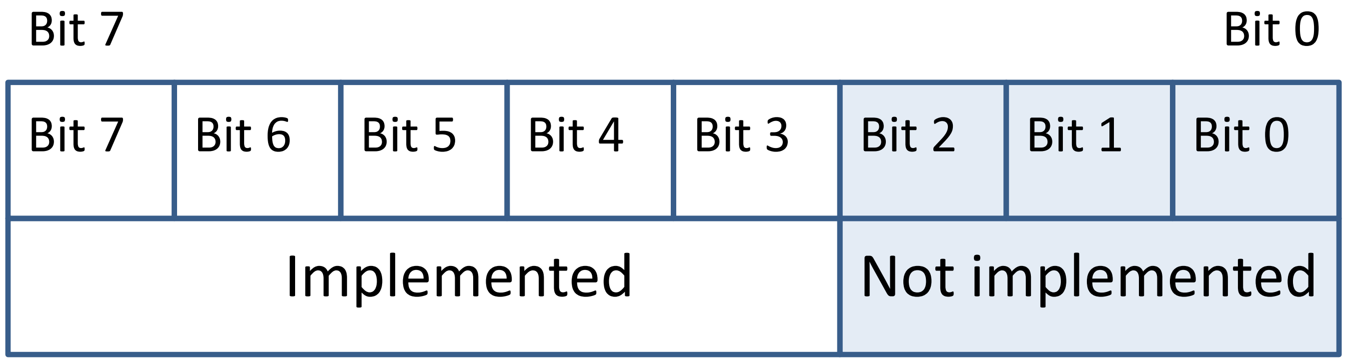
  - There are 2 control registers to set

1. Use 8-bit "**Priority Level Configuration Register**" to set **total number of priority level**, (therefore max is 256 levels)

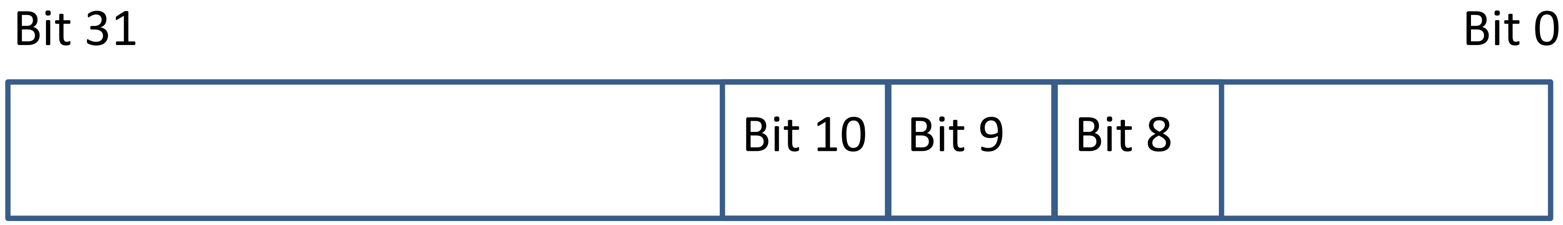
   - If the MCU has fewer supporting level (such as our LPC1769 support only 35 levels), some of **LSBs** of this register will be cut out

2. Use bit **10:8** of the 32-bits "**Application Interrupt and Reset Control Register**" to set **priority group**. Each priority group is predefined how many preempt priority and sub priority.

EX: Exception Priority



Priority Level Configuration Register (in this case use only 5 MSB, therefore support only 32 interrupt inputs)



Application Interrupt and Reset Control Register (Address 0xE000ED0C)

Priority Group Setting

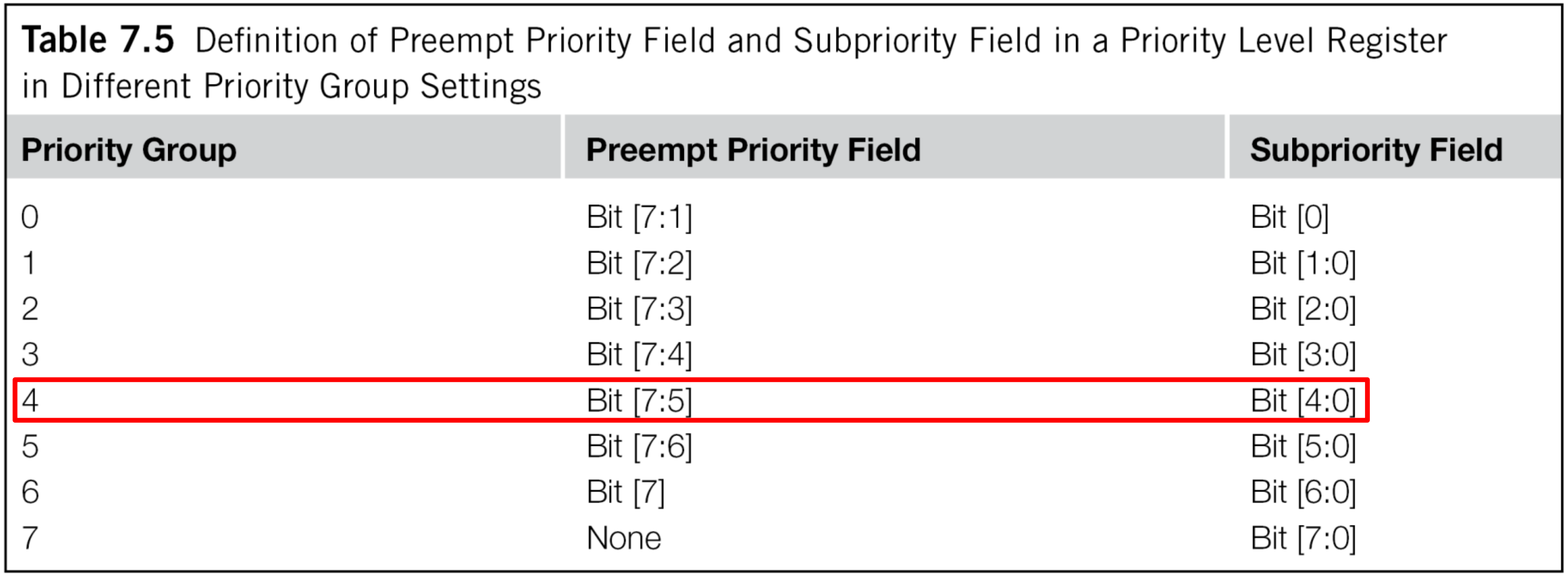


Table 7.5 Definition of Preempt Priority Field and Subpriority Field in a Priority Level Register in Different Priority Group Settings

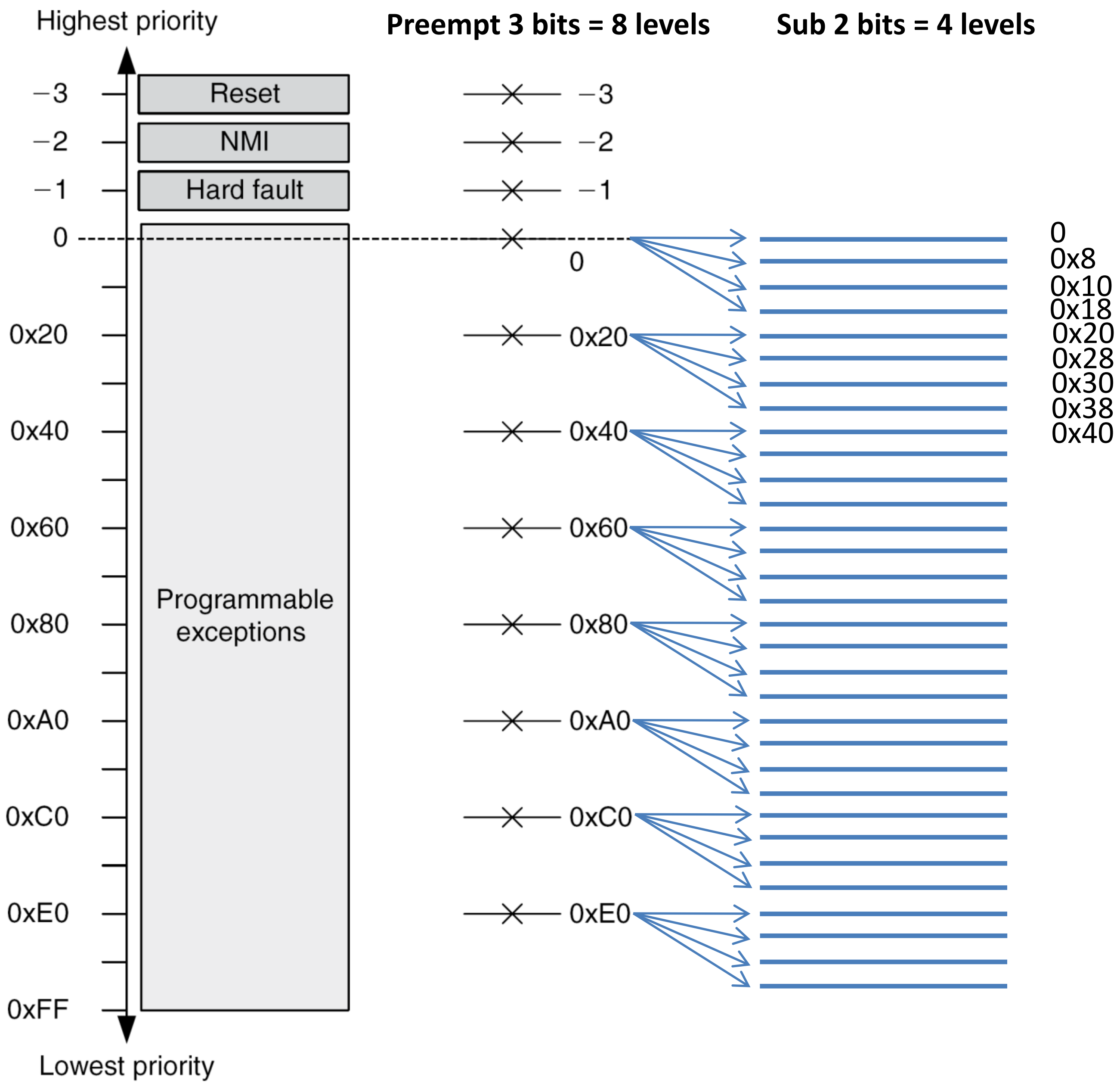
Priority Group	Preempt Priority Field	Subpriority Field
0	Bit [7:1]	Bit [0]
1	Bit [7:2]	Bit [1:0]
2	Bit [7:3]	Bit [2:0]
3	Bit [7:4]	Bit [3:0]
4	Bit [7:5]	Bit [4:0]
5	Bit [7:6]	Bit [5:0]
6	Bit [7]	Bit [6:0]
7	None	Bit [7:0]

Ex: MCU support 32 priority levels therefore **use 5 MSB** of PLCR

If AIRC Register 10:8 set to 100 , then priority **group is 4**

In group 4, **Preempt** = 7:5 (= 3bit = 8) , **Sub** = 4:0 (but we use only 5 MSB, therefore **Sub** will be = 4:3 (= 2bits = 4)

Thus, there will be **8 levels for Preempt** and in **each Preempt** there are **4 levels of Sub priority**



Highest priority
Preempt 3 bits = 8 levels
Sub 2 bits = 4 levels
−3
−2
−1
0
0x20
0x40
0x60
0x80
0xA0
0xC0
0xE0
0xFF
Reset
NMI
Hard fault
Programmable exceptions
−3
−2
−1
0
0x20
0x40
0x60
0x80
0xA0
0xC0
0xE0
0
0x8
0x10
0x18
0x20
0x28
0x30
0x38
0x40
Lowest priority

Vector Table

- When an exception takes place and is being handled by the Cortex-M3, the processor will need to **locate the starting address of the exception handler**
- This information is stored in the **vector table**
- Each exception has an associated **32-bit vector** that **points to** the memory location where the ISR that handles the exception is located.
- **By default**, the vector table **starts at memory address 0**, and is arranged according to the exception number **times four**.
- Since it is in ROM it **can not change at run time**

Vector Table

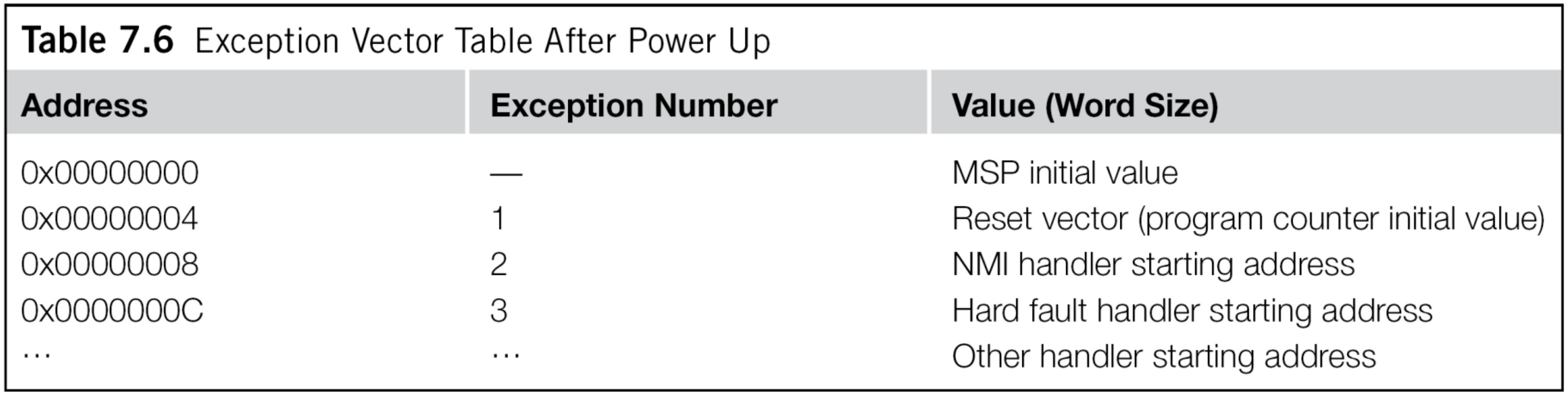


Table 7.6 Exception Vector Table After Power Up

Address	Exception Number	Value (Word Size)
0x00000000	—	MSP initial value
0x00000004	1	Reset vector (program counter initial value)
0x00000008	2	NMI handler starting address
0x0000000C	3	Hard fault handler starting address
…	…	Other handler starting address

- Exception vector table after power-up is located at address 0x00000000
- Location 0x00000004 contains the initial program counter (PC), which is called the **reset vector**
- Reset vector points to a function called **reset handler**, which is **the first thing executed following reset**
- To change interrupt handlers at **runtime,** vector table have to be **relocated** by using **vector table offset register**

Vector Table Offset Register

- A register of NVIC locates at address 0xE000ED08
- Store the **table offset value**
- The address offset should be aligned to the vector table size extended to the next larger power of 2.

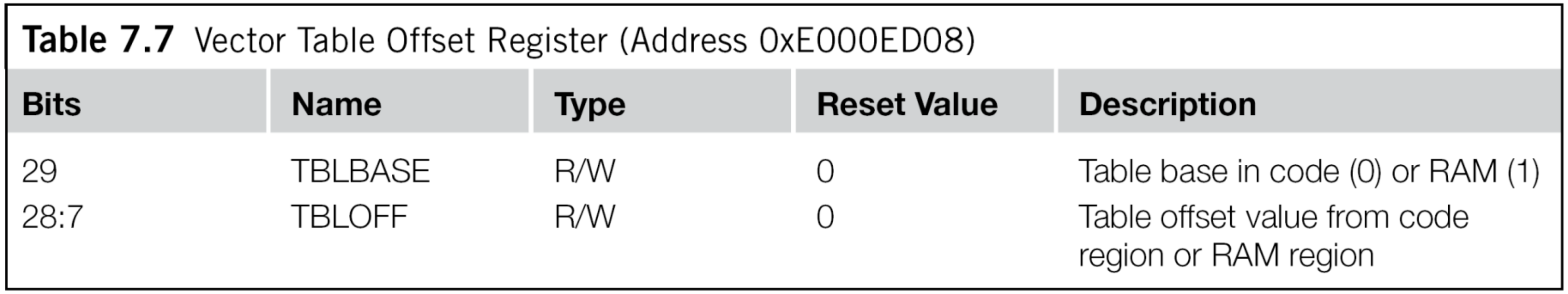


Table 7.7 Vector Table Offset Register (Address 0xE000ED08)

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

Some Tips

- If software needs to be able to run on a number of hardware devices (meaning you need a number of interrupts)
  - Make sure that you have **enough stack memory** if you allow a large number of nested interrupt levels.
  - To determine the number of interrupts supported in the design
    - Detect the exact number of external interrupts by performing a read/write test to interrupt configuration registers such as ISER or priority registers.
  - To determine the number of bits in priority-level registers
    - write 0xFF to one of the priority-level registers, then read it back and see how many bits are set.

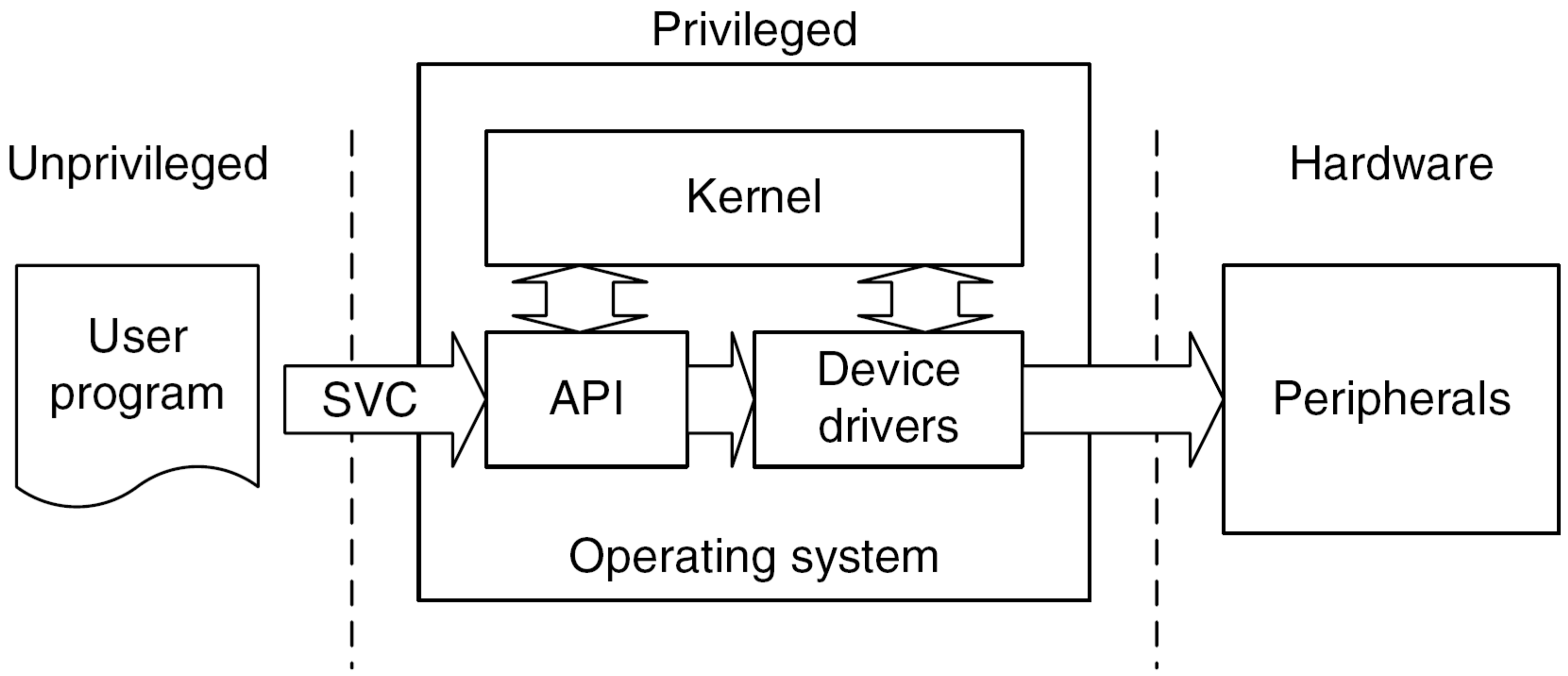
Software Interrupt

- There are several ways to trigger an interrupt
  - ❑ External interrupt input
  - ❑ Setting an interrupt pending register
  - ❑ Via the Software Trigger Interrupt Register (STIR) in the NVIC
- In most cases, some of the external interrupts are unused and can be used as software interrupts. Software interrupts can work like Supervisor Calls (next slide) to allow access to system services.

SVC Instructions

- Supervisor Calls (SVC) are a common way to allow user applications to access the application programming interface (API) of an OS.
  - ❑ By using an SVC instruction applications only need to know what parameters to pass to the OS; they don’t need to know the memory address of the OS function being called.
  - ❑ SVCs are often used to call device drivers (operating system functions that control peripherals).

SVC



SVC as a Gateway for OS Functions

Software Interrupt

- Program code can trigger interrupt as well

It can be an exception handler itself

Software can trigger interrupt by

1. Use Software Trigger Interrupt Register (STIR) 8:0 (address 0xE000EF00)

Ex:
```
NVIC->STIR = 3;
/* NVIC->STIR is defined in CMSIS compliant device driver
   library */
```

2. Use ISP Register (can not pend system exceptions)

Ex:
```
NVIC_SetPendingIRQ(3);
```

Both ways will generate interrupt request #3

Example: Setting up an Interrupt

- **Example 1:** Suppose the application is stored in CODE region
  - ❑ Vector table coded in the beginning of ROM in the CODE region (0x00000000)
  - ❑ Vector offset = 0 and interrupt vector is already in ROM
- Step to do

1. Set up the priority group setting. (the default: set to zero)
2. Set up the priority level of the interrupt. (the default: priority level 0)
3. Enable the interrupt

Example: Setting up an Interrupt

- **Example1:** MCU supports 32 priority levels, want to have 4 preempt levels, set IRQ#7 to have priority level = 0xC0 and enable it.

Chose Priority group = 5 which gives 4 levels of preempt and 8 levels of Sub priority.
Preempt = 7:6 (2 bits = 4 levels) are 0x00, 0x40, 0x80, 0xC0.
Sub priority = 5:3 (3 bits = 8 levels) are **0x00**, 0x08, 0x10, 0x18, 0x20, 0x28, 0x30, 0x38, **0x40** … 0xB8, **0xC0**, 0xC8, 0xD0, 0xD8, 0xE0, 0xE8, 0xF0, 0xF8
In C programming:

```
NVIC_SetPriorityGrouping(5);
NVIC_SetPriority(7, 0xC0); // Set IRQ#7 priority level to 0xC0
NVIC_EnableIRQ(7);
```

****See **CMSIS functions for NVIC control** in Cortex-M3 User Guide

Example: Setting up an Interrupt

If the interrupt handlers need to be changed at different stage of the application, we might need to relocate the vector table to SRAM, so that we can **modify the exception vectors**. In this case, the following extra steps would be required:

1. When the system boots up, the priority group register might need to be set up. By default, the priority group 0 is used (bit[7:1] of priority level is the preemption level and bit[0] is the sub-priority level).
2. Copy the **hard fault**, **NMI** handlers and other required vector to a new vector table location in SRAM.
3. Set up the Vector Table Offset register (VTOR) **to point to the new vector table.**
4. Set up the interrupt vector for the interrupt in the new vector table.
5. Set up the priority level for the interrupt.
6. Enable the interrupt.

Example: Setting up an Interrupt

- **Example 2:** Set up an interrupt with vector offset

In C programming:

```
// HW_REG is a macro to convert address value to pointer
#define HW_REG(addr) (*((volatile unsigned long *)(addr)))
#define NEW_VECT_TABLE 0x20008000 // An SRAM region for vector table
    NVIC_SetPriorityGrouping(5);
    ...
    HW_REG((NEW_VECT_TABLE +0x8)) = HW_REG(0x8); // Copy NMI vector
    HW_REG((NEW_VECT_TABLE +0xC)) = HW_REG(0xC); // Copy HardFault
    ...
    SCB->VTOR = NEW_VECT_TABLE; // Relocate vector table to SRAM
    ...
    HW_REG(4*(7+16)) = (unsigned) IRQ7_Handler; // Setup vector
    ...
    NVIC_SetPriority(7, 0xC0); // Set IRQ#7 priority level to 0xC0
    ...
    NVIC_EnableIRQ(7);
```

Example: Setting up an Interrupt

- **Example 2** (cont.)

In assembly:

```
LDR R0, =0xE000ED0C           ; Application Interrupt and Reset
                              ; Control Register
LDR R1, =0x05FA0500           ; Priority Group 5 (2/6)
STR R1, [R0]                  ; Set Priority Group
...
MOV R4,#8                     ; Vector Table in ROM
LDR R5,=(NEW_VECT_TABLE+8)
LDMIA R4!,{R0-R1}             ; Read vectors address for NMI and
                              ; Hard Fault
STMIA R5!,{R0-R1}             ; Copy vectors to new vector table
...
LDR R0,=0xE000ED08            ; Vector Table Offset Register
LDR R1,=NEW_VECT_TABLE
```

Example: Setting up an Interrupt

- **Example 2** (cont.)

```
STR R1,[R0]                   ; Set vector table to new location
...
LDR R0,=IRQ7_Handler          ; Get starting address of IRQ#7 handle
LDR R1,=0xE000ED08            ; Vector Table Offset Register
LDR R1,[R1]
ADD R1, R1, #(4*(7+16))       ; Calculate IRQ#7 handler vector
                              ; address
STR R0,[R1]                   ; Setup vector for IRQ#7
...
LDR R0,=0xE000E400            ; External IRQ priority base
MOV R1, #0x0
STRB R1,[R0,#7]               ; Set IRQ#7 priority to 0x0
...
LDR R0,=0xE000E100            ; SETEN register
MOV R1,#(1<<7)                ; IRQ#7 enable bit (value 0x1 shifted
                              ; by 7 bits)
STR R1,[R0]                   ; Enable the interrupt
```

Interrupt/Exception Handlers

- Example

```
void UART1_Handler(void) {
    ... // processing task for the peripheral
    return;
}
```

- ❑ Interrupt **handler name** should be **match** the interrupt handler name defined by MCU vendor to ensure that the vector is set up in the vector table correctly.
- ❑ You can find the handler function name in the vector table inside the startup codes.