

# Introduction the ARM Cortex-M3 Exception / Interrupt

Hsuan-Cheng Lin

2012.6.6

# Outline

- The ARM processor
- The Cortex-M3
- The Registers
- Exceptions
- The NVIC and Interrupt Control
- Interrupt Behavior
- Reference

# The ARM Processor (1)

- ARM – Advanced RISC Machines (1990)  
Small size, Low cost, Low power consumption, High performance
- License : IP core (Semiconductor intellectual property core)

| Architecture | Family  |
|--------------|---|
| ARMv1        | ARM1  |
| ARMv2        | ARM2, ARM3  |
| ARMv3        | ARM6, ARM7  |
| ARMv4        | StrongARM, ARM7TDMI, ARM9TDMI                                       |
| ARMv5        | ARM7EJ, ARM9E, ARM10E, XScale                                       |
| ARMv6        | ARM11, ARM Cortex-M   |
| ARMv7        | ARM Cortex-A, ARM Cortex-M, ARM Cortex-R                            |
| ARMv8        | No cores available yet.<br>Will support 64-bit data and addressing. |

ARM cores(wiki)

# The ARM processor (2)

## ■ Processor Architecture = Instruction Set + Programmer's model



**ARM7TDMI**  
**ARM922T**

Thumb  
instruction set



**ARM926EJ-S**  
**ARM946E-S**  
**ARM966E-S**

Improved  
ARM/Thumb  
Interworking

DSP instructions

### **Extensions:**

Jazelle (5TEJ)



**ARM1136JF-S**  
**ARM1176JZF-S**  
**ARM11 MPCore**

SIMD Instructions

Unaligned data support

### **Extensions:**

Thumb-2 (6T2)

TrustZone (6Z)

Multicore (6K)



**Cortex-A8/R4/M3/M1**

Thumb-2

### **Extensions:**

v7A (applications) – NEON

v7R (real time) – HW Divide

V7M (microcontroller) – HW  
Divide and Thumb-2 only

- Note: Implementations of the same architecture can be very different
  - ARM7TDMI - architecture v4T. Von Neuman core with 3 stage pipeline
  - ARM920T - architecture v4T. Harvard core with 5 stage pipeline and MMU

Development of the ARM Architecture

# The ARM processor (3)

| Features          | ARM7TDMI-S                      | Cortex-M3                                 |
|-------------------|---------------------------------|---|
| Architecture      | ARMv4T (von Neumann)            | ARMv7-M (Harvard)                         |
| ISA Support       | Thumb / ARM                     | Thumb / Thumb-2                           |
| Pipeline          | 3-Stage                         | 3-Stage + branch speculation              |
| Interrupts        | FIQ / IRQ                       | NMI + 1 to 240 Physical Interrupts        |
| Interrupt Latency | 24-42 Cycles                    | 12 Cycles                                 |
| Sleep Modes       | None                            | Integrated                                |
| Memory Protection | None                            | 8 region Memory Protection Unit           |
| Dhrystone         | 0.95 DMIPS/MHz (ARM mode)       | 1.25 DMIPS/MHz                            |
| Power Consumption | 0.28mW/MHz                      | 0.19mW/MHz                                |
| Area              | 0.62mm <sup>2</sup> (Core Only) | 0.86mm <sup>2</sup> (Core & Peripherals)* |

# The ARM processor (4)

- Many semiconductor or IC design firms hold ARM licenses:  
Analog Devices, AppliedMicro, Atmel, Broadcom, Cirrus Logic, Energy Micro, Faraday Technology, Freescale, Fujitsu, Intel (through its settlement with Digital Equipment Corporation), IBM, Infineon Technologies, Marvell Technology Group, Nintendo, NXP Semiconductors, OKI, Qualcomm, Samsung, Sharp, **STMicroelectronics**, and Texas Instruments.
- ARM Cortex-A8  
Apple A4 (iPhone 4, ipad, iPod touch, Apple TV);  
NVIDIA Tegra 2/3; Samsung Exynos; TI OMAP4....
- ARM Cortex-A9  
Apple A5 (iPhone 4S, ipad2...);



iPhone 4S



iPhone 4

# The Cortex-M3

# The Cortex Series

- A Profile (ARMv7-A): Application processors required to run complex applications .

High-end embedded operating systems.(Symbian, Linux, and Windows Embedded)

Highest processing power, virtual memory system support with Memory Management Units (MMUs).

High-end mobile phones and electronic wallets for financial transactions.

- R Profile (ARMv7-R): Real-time.

High-end breaking systems and hard drive controllers.

High processing power and high reliability are essential and for which low latency is important.

- M Profile (ARMv7-M): Microcontroller targeting low-cost applications.

Processing efficiency is important and cost, low power consumption, low interrupt latency, and ease of use are critical.

Industrial control applications, including real-time control systems.

- The Cortex processor families are the first products developed on architecture v7.
- The Cortex-M3 processor is based on one profile of the v7 architecture, called ARM v7-M, an architecture specification for microcontroller products.



# The Cortex-M3 processor (1)

- **Greater performance efficiency**

Allowing more work to be done without increasing the frequency or power requirements.

- **Low power consumption**

Enabling longer battery life, especially critical in portable products including wireless networking applications.

- **Lower-cost solutions**

Reducing 32-bit-based system costs close to those of legacy 8-bit and 16-bit devices and enabling low-end, 32-bit microcontrollers to be priced at less than US\$1 for the first time.

- **Enhanced determinism**

Guaranteeing that critical tasks and **interrupts** are serviced as quickly as possible but in a known number of cycles.

- **Improved code density**

Ensuring that code fits in even the smallest memory footprints.

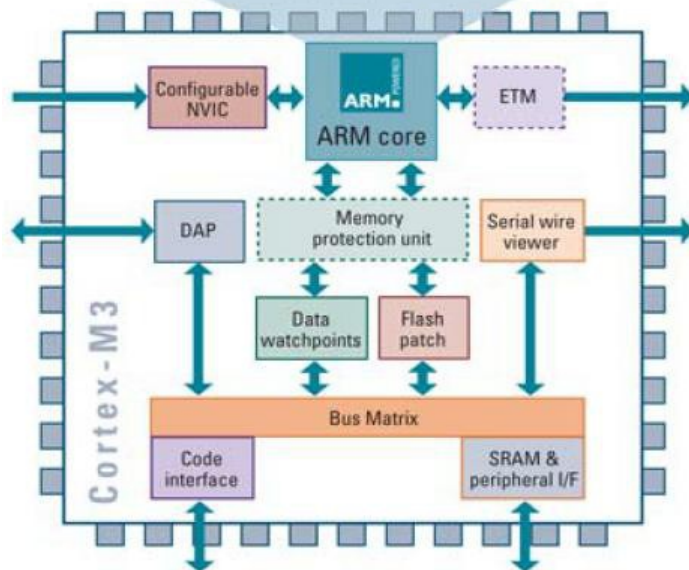
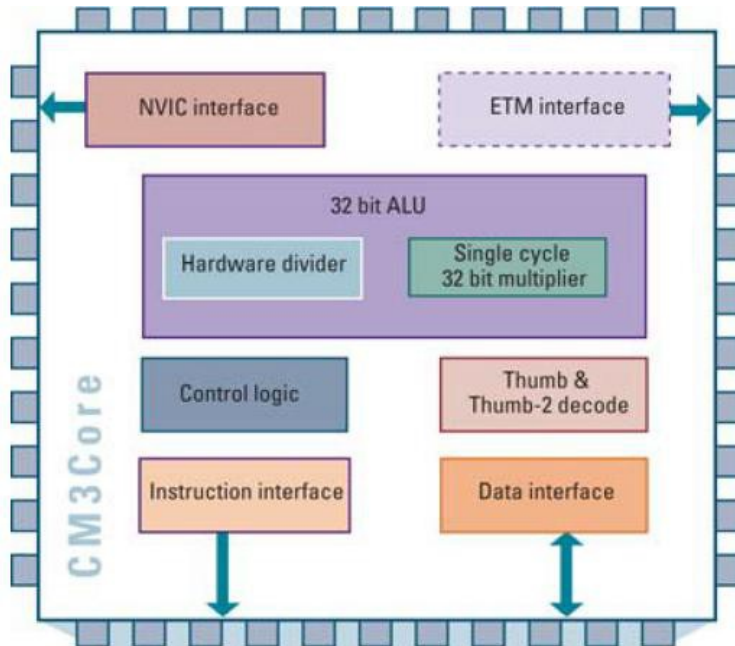
- **Ease of use**

Providing easier programmability and debugging for the growing number of 8-bit and 16-bit users migrating to 32-bit.

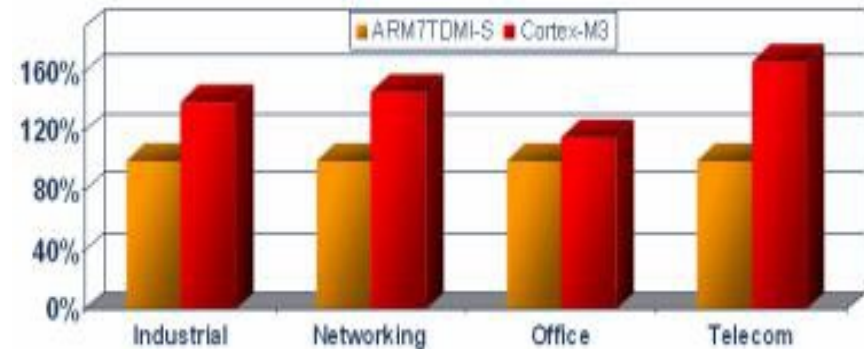
- **Wide choice of development tools**

From low-cost or free compilers to full-featured development suites from many development tool vendors.

# The Cortex-M3 processor (2)

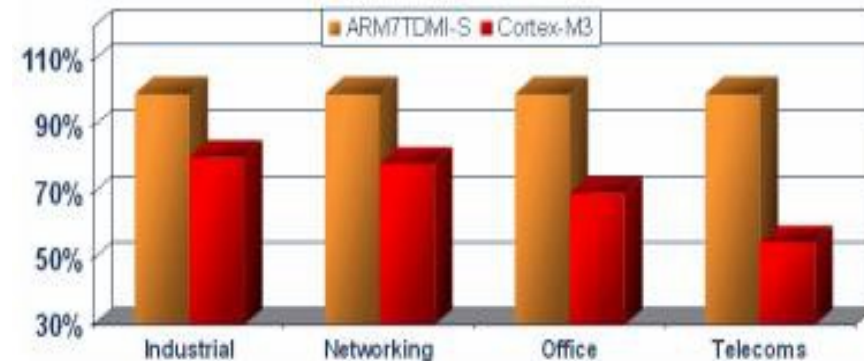


Relative Benchmark Performance (per MHz)



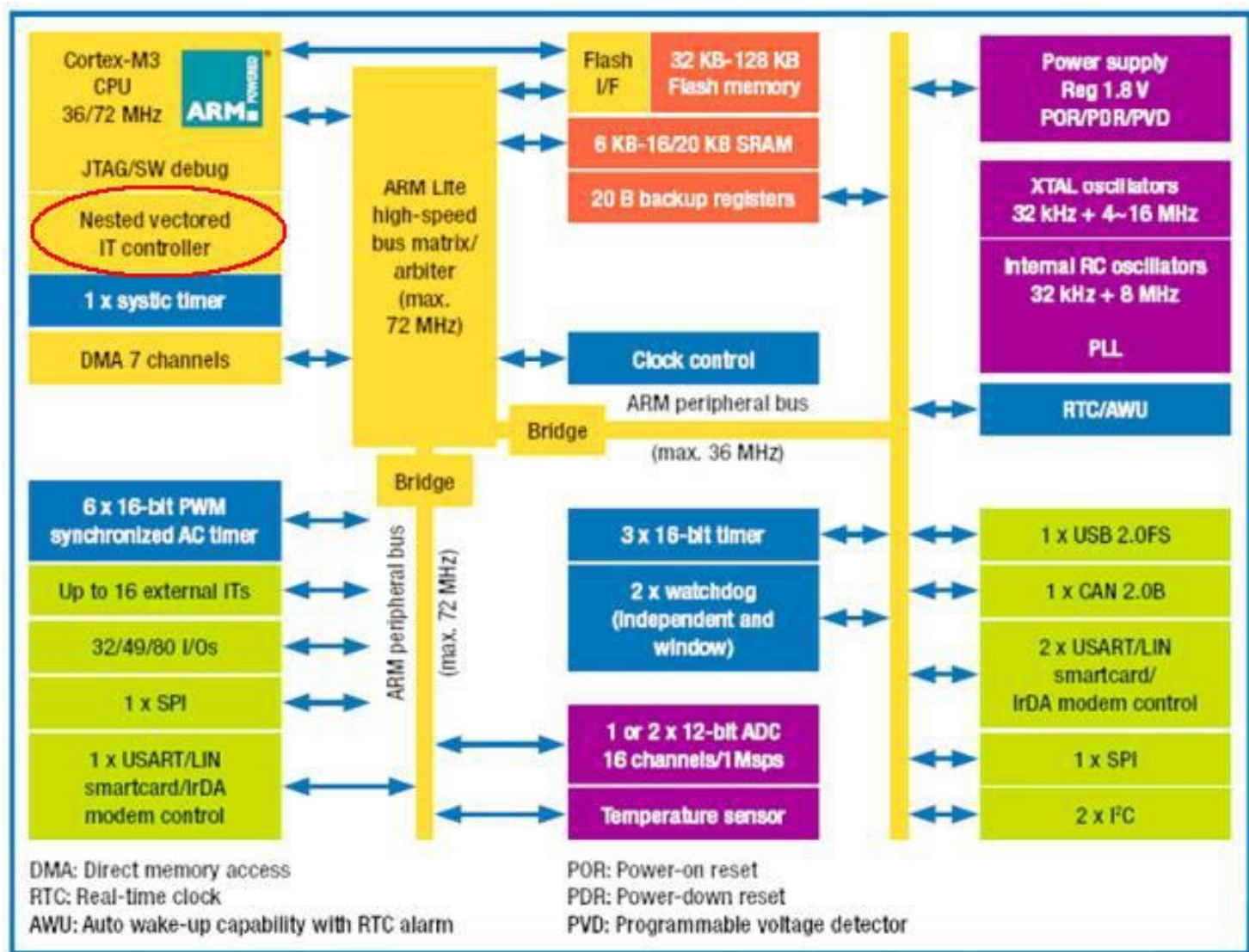
Relative **performance** for ARM7TDMI-S (ARM) and Cortex-M3 (Thumb-2)

Relative Benchmark Code Size



Relative **code size** for ARM7TDMI-S (ARM) and Cortex-M3 (Thumb-2)

# The Cortex-M3 processor (3)



STM32F10x Block Diagram

# The Cortex-M3 processor (4)





















- The Cortex-M3 processor has two modes and two privilege levels.
- The **operation** modes (**thread** mode and **handler** mode) determine whether the processor is running a normal program or running an exception handler like an interrupt handler or system exception handler.
- The **privilege** levels (**privileged** level and **user** level) provide a mechanism for safeguarding memory accesses to critical regions as well as providing a basic security model.

ARM7: User/FIQ/IRQ/Supervisor/Abort/Undefined/System

|                                  | <i>Privileged</i> | <i>User</i> |
|----------------------------------|-------------------|-------------|
| <i>When running an exception</i> | Handle Mode       |             |
| <i>When running main program</i> | Thread Mode       | Thread Mode |

Operation Modes and Privilege Levels in Cortex-M3

# The Registers (ARM7TDMI)

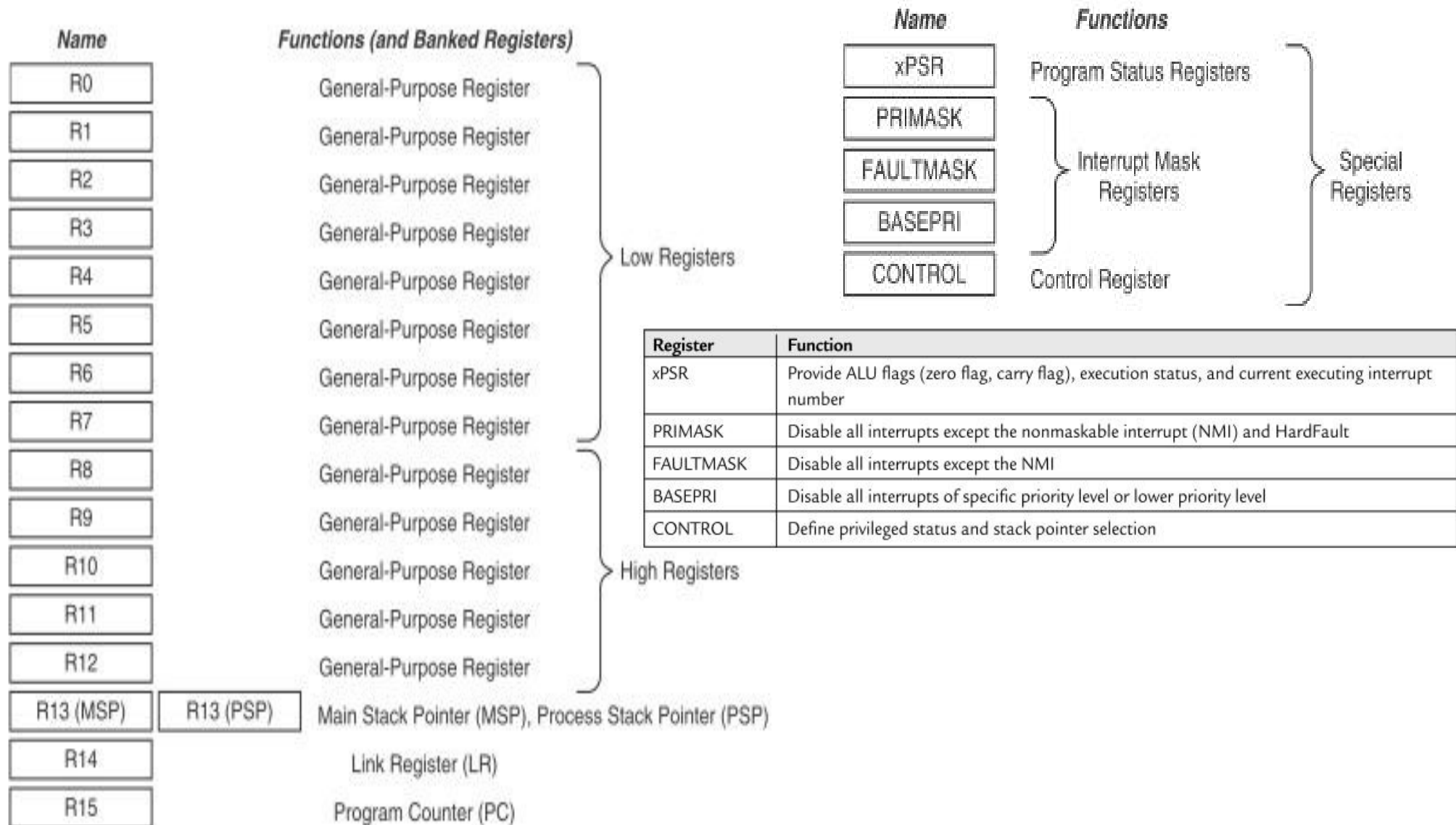
| Modes            |        |  |  |  |  |   |
|------------------|--------|--|--|--|--|---|
| Privileged modes |        |  |  |  |  |   |
| Exception modes  |        |  |  |  |  |   |
| User             | System | Supervisor   | Abort  | Undefined  | Interrupt  | Fast interrupt  |
| R0               | R0     | R0   | R0   | R0   | R0   | R0  |
| R1               | R1     | R1   | R1   | R1   | R1   | R1  |
| R2               | R2     | R2   | R2   | R2   | R2   | R2  |
| R3               | R3     | R3   | R3   | R3   | R3   | R3  |
| R4               | R4     | R4   | R4   | R4   | R4   | R4  |
| R5               | R5     | R5   | R5   | R5   | R5   | R5  |
| R6               | R6     | R6   | R6   | R6   | R6   | R6  |
| R7               | R7     | R7   | R7   | R7   | R7   | R7  |
| R8               | R8     | R8   | R8   | R8   | R8   |  R8_lq     |
| R9               | R9     | R9   | R9   | R9   | R9   |  R9_lq   |
| R10              | R10    | R10  | R10  | R10  | R10  |  R10_lq  |
| R11              | R11    | R11  | R11  | R11  | R11  |  R11_lq  |
| R12              | R12    | R12  | R12  | R12  | R12  |  R12_lq  |
| R13              | R13    |  R13_svc  |  R13_abt  |  R13_und  |  R13_irq  |  R13_lq  |
| R14              | R14    |  R14_svc  |  R14_abt  |  R14_und  |  R14_irq  |  R14_lq  |
| PC               | PC     | PC   | PC   | PC   | PC   | PC  |
| CPSR             | CPSR   | CPSR   | CPSR   | CPSR   | CPSR   | CPSR  |
|                  |        |  SPSR_svc |  SPSR_abt |  SPSR_und |  SPSR_irq |  SPSR_lq |



Indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode

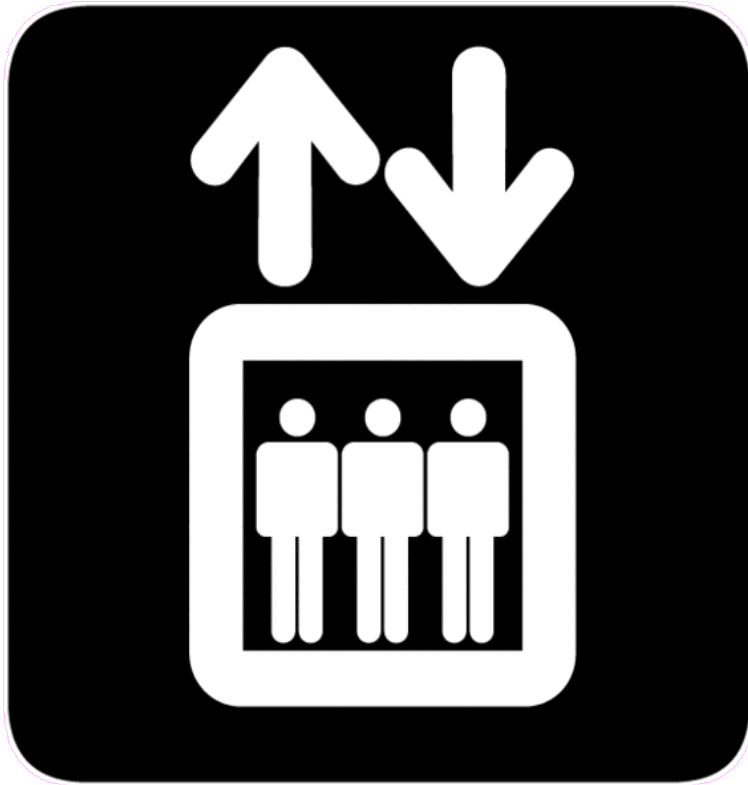


# The Registers (Cortex-M3)

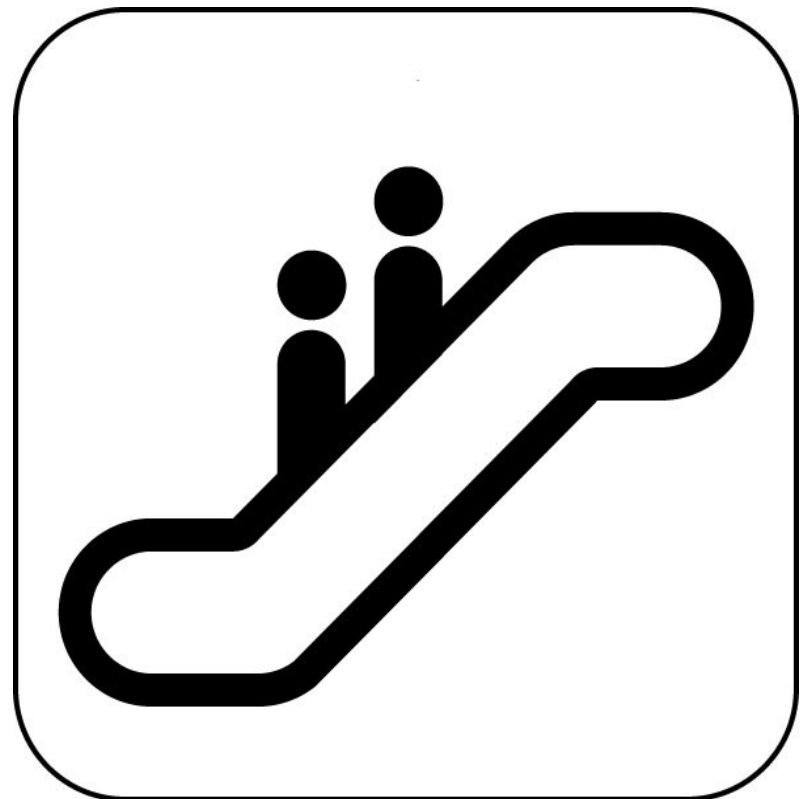


Exceptions/Interrupt

# Interrupt vs Polling



Interrupt

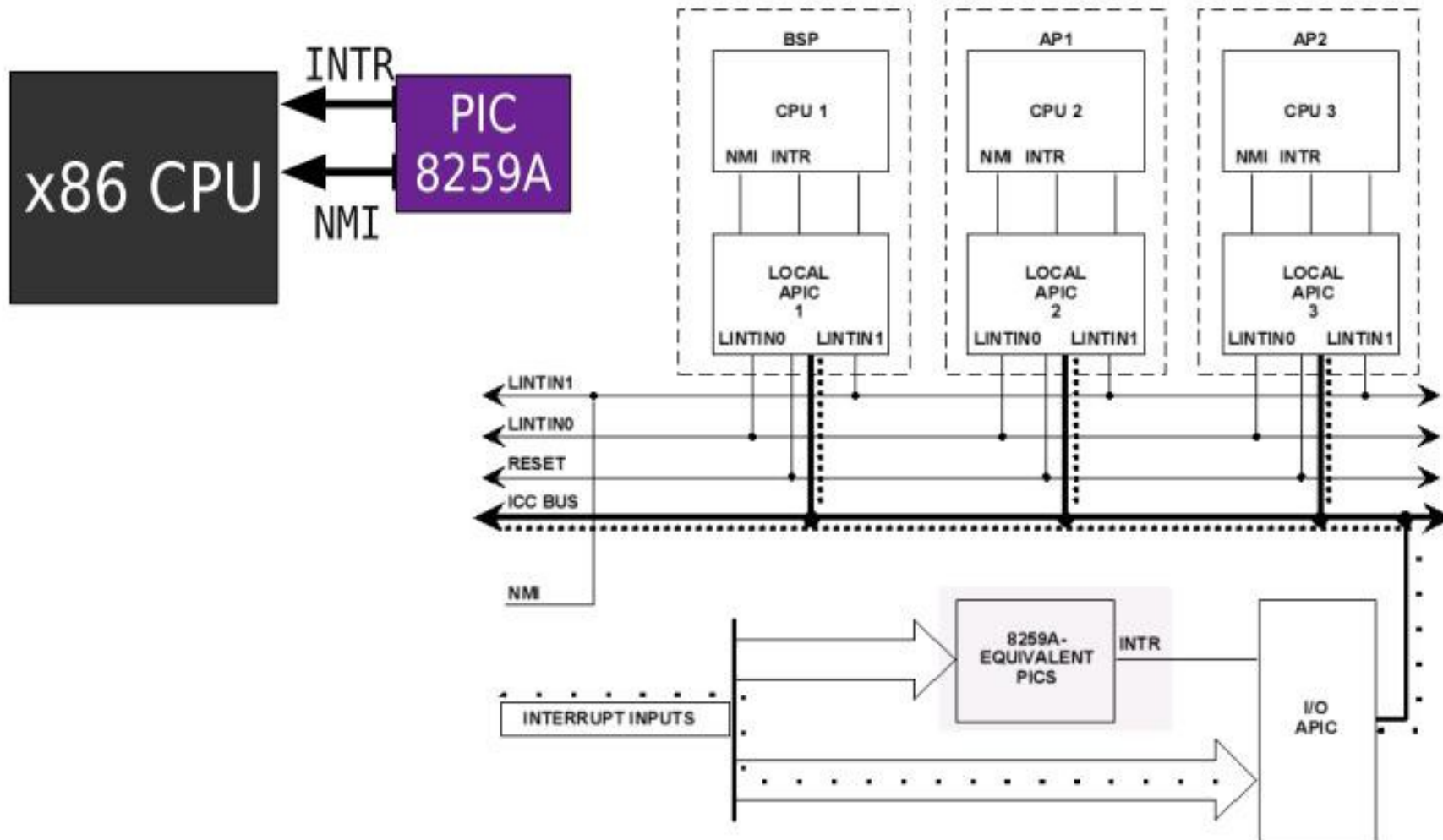


Polling



# Interrupt Controller

The Cortex-M3 processor includes an interrupt controller called the Nested Vectored Interrupt Controller (NVIC).



# Exception Types

- Exceptions are numbered:
  - 1 to 15 for system exceptions.
  - 16 and above for external interrupt inputs.
- Most of the exceptions have programmable priority, and a few have fixed priority.
- Support 1-240 interrupt.

# List of System Exceptions

Note: There is no exception number 0

| Exception Number | Exception Type  | Priority     | Description  |
|------------------|-----------------|--------------|--|
| 1                | Reset           | −3 (Highest) | Reset  |
| 2                | NMI             | −2           | Nonmaskable interrupt (external NMI input)   |
| 3                | Hard Fault      | −1           | All fault conditions, if the corresponding fault handler is not enabled  |
| 4                | MemManage Fault | Programmable | Memory management fault; MPU violation or access to illegal locations  |
| 5                | Bus Fault       | Programmable | Bus error; occurs when AHB interface receives an error response from a bus slave (also called <i>prefetch abort</i> if it is an instruction fetch or <i>data abort</i> if it is a data access) |
| 6                | Usage Fault     | Programmable | Exceptions due to program error or trying to access coprocessor (the Cortex-M3 does not support a coprocessor)   |
| 7-10             | Reserved        | NA           | –  |
| 11               | SVCall          | Programmable | System Service call  |
| 12               | Debug Monitor   | Programmable | Debug monitor (breakpoints, watchpoints, or external debug requests)   |
| 13               | Reserved        | NA           | –  |
| 14               | PendSV          | Programmable | Pendable request for system device   |
| 15               | SYSTICK         | Programmable | System Tick Timer  |

# List of External Interrupts

Need to check the chip manufacturer's datasheets to determine the numbering of the interrupts.

| Exception Number | Exception Type          | Priority     |
|------------------|-------------------------|--------------|
| 16               | External Interrupt #0   | Programmable |
| 17               | External Interrupt #1   | Programmable |
| ...              | ...                     | ...          |
| 255              | External Interrupt #239 | Programmable |

- When an enabled exception occurs but cannot be carried out immediately, it will be pended.  
(a higher-priority interrupt service routine is running or if the interrupt mask register is set)
- This means that a register in the NVIC (pending status) will hold the exception request until the exception can be carried out.
- This is different from traditional ARM processors.  
CPSR → SPSR  
Switch to ARM mode and Disable IRQ  
...

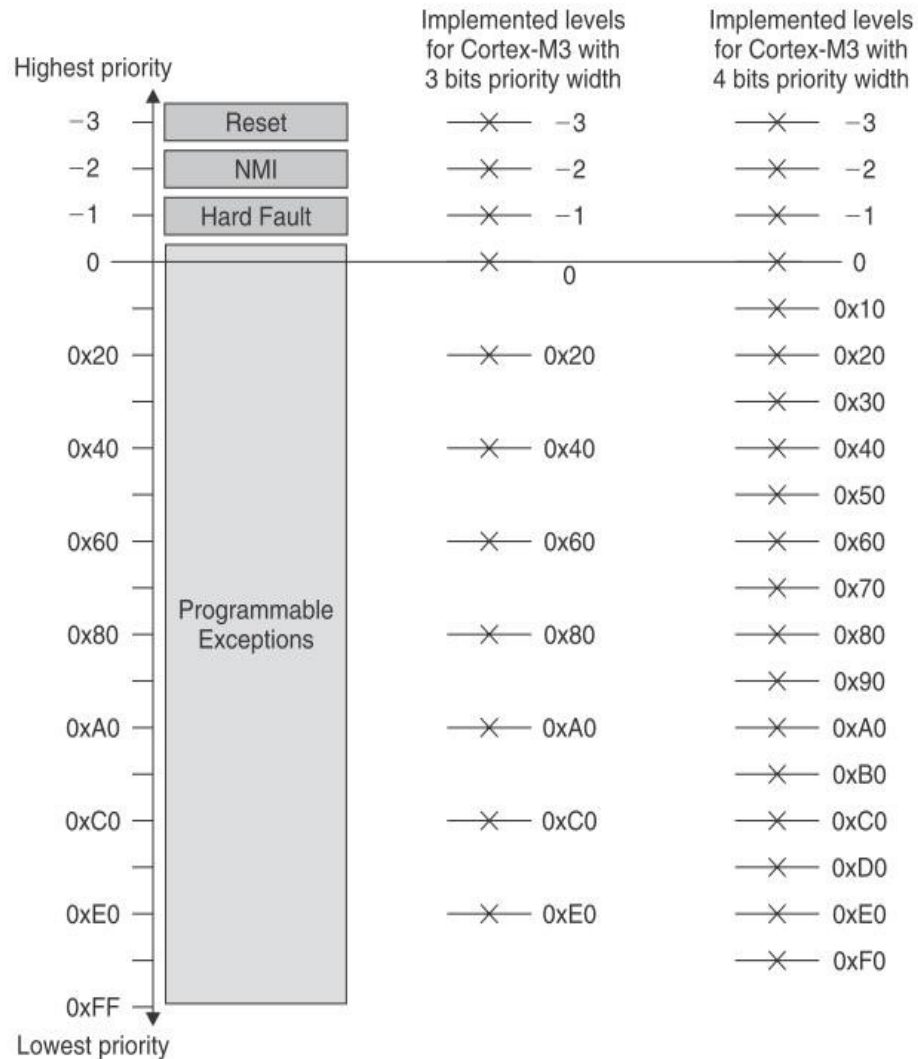
| Address                    | Name                       | Type | Reset      | Description                              |
|----------------------------|----------------------------|------|------------|--|
| 0xE000E004                 | ICTR                       | RO   | -          | Interrupt Controller Type Register, ICTR |
| 0xE000E100 -<br>0xE000E11C | NVIC_ISER0 -<br>NVIC_ISER7 | RW   | 0x00000000 | Interrupt Set-Enable Registers           |
| 0xE000E180 -<br>0xE000E19C | NVIC_ICER0 -<br>NVIC_ICER7 | RW   | 0x00000000 | Interrupt Clear-Enable Registers         |
| 0xE000E200 -<br>0xE000E21C | NVIC_ISPR0 -<br>NVIC_ISPR7 | RW   | 0x00000000 | Interrupt Set-Pending Registers          |
| 0xE000E280 -<br>0xE000E29C | NVIC_ICPR0 -<br>NVIC_ICPR7 | RW   | 0x00000000 | Interrupt Clear-Pending Registers        |
| 0xE000E300 -<br>0xE000E31C | NVIC_IABR0 -<br>NVIC_IABR7 | RO   | 0x00000000 | Interrupt Active Bit Register            |
| 0xE000E400 -<br>0xE000E4EC | NVIC_IPR0 -<br>NVIC_IPR59  | RW   | 0x00000000 | Interrupt Priority Register              |

## NVIC registers

| Address    | Name     | Type | Reset Value | Description   |
|------------|----------|------|-------------|---|
| 0xE000E200 | SETPEND0 | R/W  | 0           | Pending for external interrupt #0-31<br>bit[0] for interrupt #0 (exception #16)<br>bit[1] for interrupt #1 (exception #17)<br>...<br>bit[31] for interrupt #31 (exception #47)<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current status                 |
| 0xE000E204 | SETPEND1 | R/W  | 0           | Pending for external interrupt #32-63<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current status  |
| 0xE000E208 | SETPEND2 | R/W  | 0           | Pending for external interrupt #64-95<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current status  |
| ...        | -        | -    | -           | -   |
| 0xE000E280 | CLRPEND0 | R/W  | 0           | Clear pending for external interrupt #0-31<br>bit[0] for interrupt #0 (exception #16)<br>bit[1] for interrupt #1 (exception #17)<br>...<br>bit[31] for interrupt #31 (exception #47)<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current pending status |
| 0xE000E284 | CLRPEND1 | R/W  | 0           | Clear pending for external interrupt #32-63<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current pending status  |
| 0xE000E288 | CLRPEND2 | R/W  | 0           | Clear pending for external interrupt #64-95<br>Write 1 to clear bit to 1; write 0 has no effect<br>Read value indicates the current pending status  |
| ...        | -        | -    | -           | -   |

## Interrupt Set Pending Registers and Interrupt Clear Pending Registers

# Definitions of Priority(1)



Available Priority Levels with 3-Bit or 4-Bit Priority Width

| Bit 7       | Bit 6 | Bit 5 | Bit 4                         | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|-------------|-------|-------|-------------------------------|-------|-------|-------|-------|
| Implemented |       |       | Not implemented, read as zero |       |       |       |       |

| Bit 7,6,5,4 | Bit 3,2,1,0 | Value |
|-------------|-------------|-------|
| 0000        | 0000        | 0x00  |
| 0010        | 0000        | 0x20  |
| 0100        | 0000        | 0x40  |
| 0110        | 0000        | 0x60  |
| 1000        | 0000        | 0x80  |
| 1010        | 0000        | 0xA0  |
| 1100        | 0000        | 0xC0  |
| 1110        | 0000        | 0xE0  |

A Priority Level Register with 3-bit Implemented

| Bit 7       | Bit 6 | Bit 5 | Bit 4 | Bit 3                      | Bit 2 | Bit 1 | Bit 0 |
|-------------|-------|-------|-------|----------------------------|-------|-------|-------|
| Implemented |       |       |       | Not implemented, read as 0 |       |       |       |

A Priority Level Register with 4-bit Implemented

# Definitions of Priority(2)

- Supports three fixed highest-priority levels.(-3,-2,-1)
- Up to 256 levels of programmable priority.  
(a maximum of 128 levels of preemption)
- More priority bits can also increase gate counts and hence power consumption.
- The minimum number of implemented priority register widths is 3 bits. (eight levels)
- This reduction of levels is implemented by cutting out the LSB part of the priority configuration registers.  
The reason for removing the LSB of the register instead of the MSB is to make it easier to port software from one Cortex-M3 device to another.

# Definitions of Priority(3)

| Bit    | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |
|--------|---|---|---|---|---|---|---|---|------|
| IRQ #0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0x05 |
| IRQ #1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x03 |

| Bit    | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |
|--------|---|---|---|---|---|---|---|---|------|
| IRQ #0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0x01 |
| IRQ #1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0x03 |

MSB

LSB

| Bit    | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |
|--------|---|---|---|---|---|---|---|---|------|
| IRQ #0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0x50 |
| IRQ #1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0x30 |

| Bit    | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |      |
|--------|---|---|---|---|---|---|---|---|------|
| IRQ #0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0x40 |
| IRQ #1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0x10 |



# Definitions of Priority(4)

| Priority Level | Exception Type                              | Devices with 3-Bit Priority Configuration Registers | Devices with 5-Bit Priority Configuration Registers | Devices with 8-Bit Priority Configuration Registers |
|----------------|---|---|---|---|
| -3 (Highest)   | Reset                                       | -3  | -3  | -3  |
| -2             | NMI   | -2  | -2  | -2  |
| -1             | Hard fault                                  | -1  | -1  | -1  |
| 0,             | Exceptions with programmable priority level | 0x00  | 0x00  | 0x00, 0x01  |
| 1,             |   | 0x20  | 0x08  | 0x02, 0x03  |
| ...            |   | ...   | ...   | ...   |
| 0xFF           |   | 0xE0  | 0xF8  | 0xFE, 0xFE  |

Available Priority Levels for Devices with 3-bit, 5-bit, and 8-bit Priority Level Registers

Q: If the priority level configuration registers are 8 bits wide, why there are only 128 preemption levels?

A: 8-bit register is further divided into two parts: preempt priority and subpriority

# Definitions of Priority(5)

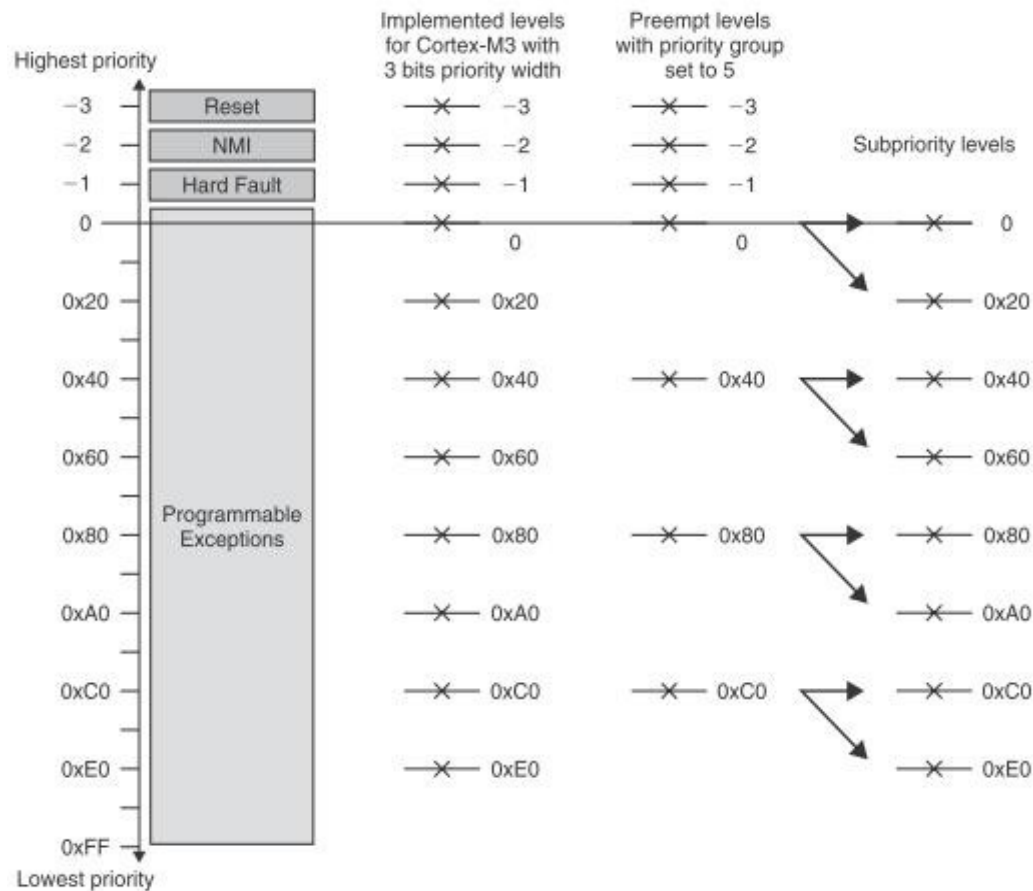
| Bits  | Name          | Type | Reset Value | Description   |
|-------|---------------|------|-------------|---|
| 31:16 | VECTKEY       | R/W  | –           | Access key; 0x05FA must be written to this field to write to this register, otherwise the write will be ignored; the read-back value of the upper half word is 0xFA05 |
| 15    | ENDIANNESS    | R    | –           | Indicates endianness for data: 1 for big endian (BE8) and 0 for little endian; this can only change after a reset   |
| 10:8  | PRIGROUP      | R/W  | 0           | Priority group  |
| 2     | SYSRESETREQ   | W    | –           | Requests chip control logic to generate a reset   |
| 1     | VECTCLRACTIVE | W    | –           | Clears all active state information for exceptions; typically used in debug or OS to allow system to recover from system error (Reset is safer)                       |
| 0     | VECTRESET     | W    | –           | Resets the Cortex-M3 processor (except debug logic), but this will not reset circuits outside the processor   |

| 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]         |

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

- Using a configuration register in the NVIC called **Priority Group**.
- The priority-level configuration registers for each exception with programmable priority levels is divided into two halves.
- The upper half (left bits) is the preempt priority.
- The lower half (right bits) is the subpriority.

# Definitions of Priority(6)

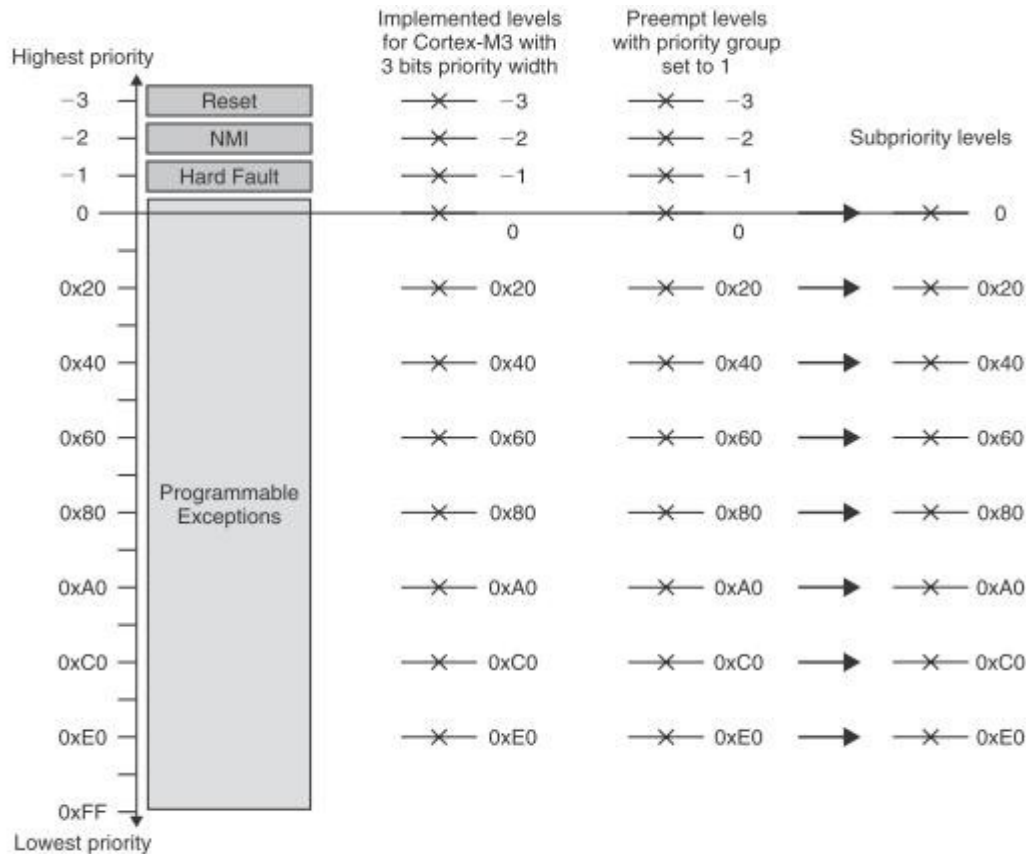


| Bit 7            | Bit 6 | Bit 5        | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|------------------|-------|--------------|-------|-------|-------|-------|-------|
| Preempt priority |       | Sub priority |       |       |       |       |       |

The Define of priority group 5

| Bit<br>7,6 | Bit<br>5 | Bit<br>4,3,2,1,0 | Preempt | Subpriority  |
|------------|----------|------------------|---------|--------------|
| 00         | 01       | 0 0000           | 0x00    | 0x00<br>0x20 |
| 01         | 01       | 0 0000           | 0x40    | 0x40<br>0x60 |
| 10         | 01       | 0 0000           | 0x80    | 0x80<br>0xA0 |
| 11         | 01       | 0 0000           | 0xC0    | 0xC0<br>0xE0 |

# Definitions of Priority(7)



| Bit 7                 | Bit 6 | Bit 5 | Bit 4                                | Bit 3 | Bit 2 | Bit 1                  | Bit 0 |
|-----------------------|-------|-------|--------------------------------------|-------|-------|------------------------|-------|
| Preempt priority[5:7] |       |       | Preempt priority bit[4:2] (always 0) |       |       | Subpriority (always 0) |       |

The Define of priority group 1

| Bit 7,6,5 | Bit 4,3,2,1,0 | Preempt | Subpriority |
|-----------|---------------|---------|-------------|
| 000       | 0 0000        | 0x00    | 0x00        |
| 001       | 0 0000        | 0x20    | 0x20        |
| 010       | 0 0000        | 0x40    | 0x40        |
| 011       | 0 0000        | 0x60    | 0x60        |
| 100       | 0 0000        | 0x80    | 0x80        |
| 101       | 0 0000        | 0xA0    | 0xA0        |
| 110       | 0 0000        | 0xC0    | 0xC0        |
| 111       | 0 0000        | 0xE0    | 0xE0        |

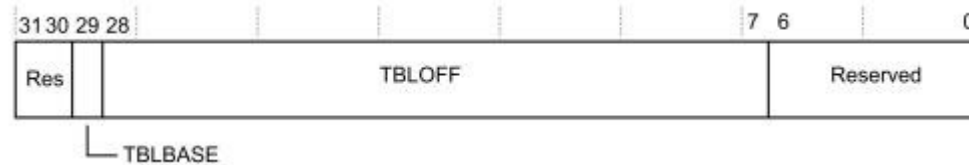
# Vector Tables

| 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               |

By default, the vector table starts at address zero,  
and the vector address is arranged according to the exception number times 4.

- Since the address 0x0 should be boot code, usually it will either be Flash memory or ROM devices, and the value cannot be changed at run time.
- However, the vector table can be relocated to other memory locations in the Code or RAM region where the RAM is so that we can change the handlers during run time.  
This is done by setting a register in the NVIC called the vector table offset register (address 0xE000ED08).

# Vector Tables



| Bits    | Field   | Function   |
|---------|---------|--|
| [31:30] | -       | Reserved   |
| [29]    | TBLBASE | Table base is in Code (0) or RAM (1)   |
| [28:7]  | TBLOFF  | Vector table base offset field. Contains the offset of the table base from the bottom of the SRAM or CODE space. |
| [6:0]   | -       | Reserved.  |

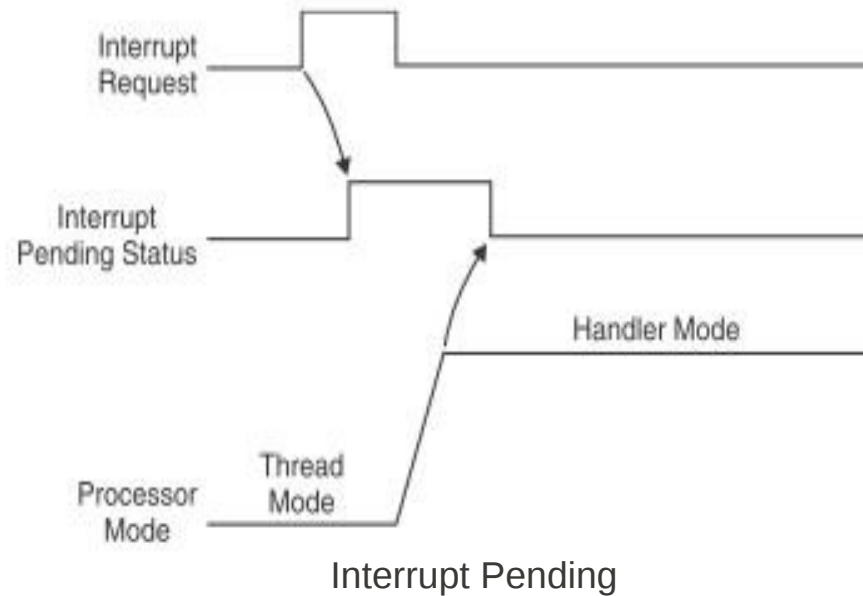
- The address offset should be aligned to the vector table size, extended to the power of 2.
- For example, if there are 32 IRQ inputs, the total number of exceptions will be  $32 + 16$  (system exceptions) = 48.

Extending it to the power of 2 makes it 64. (2,4,8,16,32,64)

Multiplying it by 4 makes it 256 (0x100). ( $64 * 4 = 256$ )

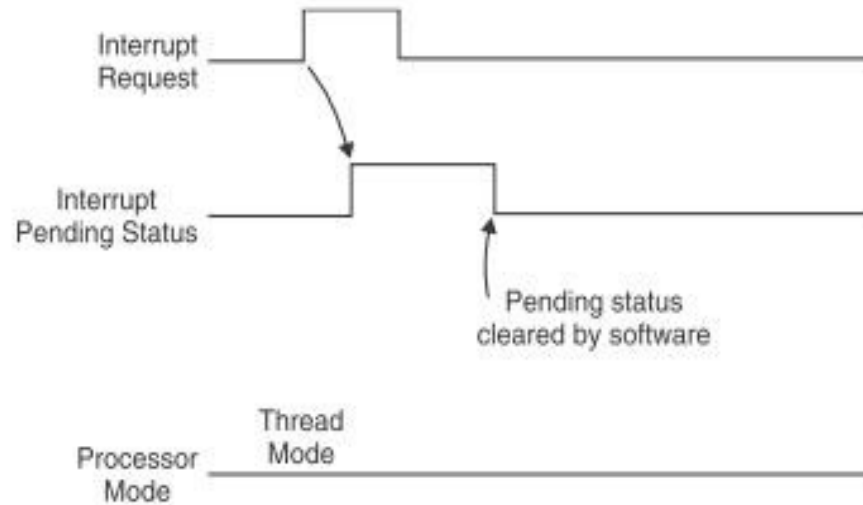
Therefore, the vector table offset can be programmed as 0x0, 0x100, 0x200, and so on.

# Interrupt Inputs and Pending Behavior(1)



- When an interrupt input is asserted, it will be pended. 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.

## Interrupt Inputs and Pending Behavior (2)

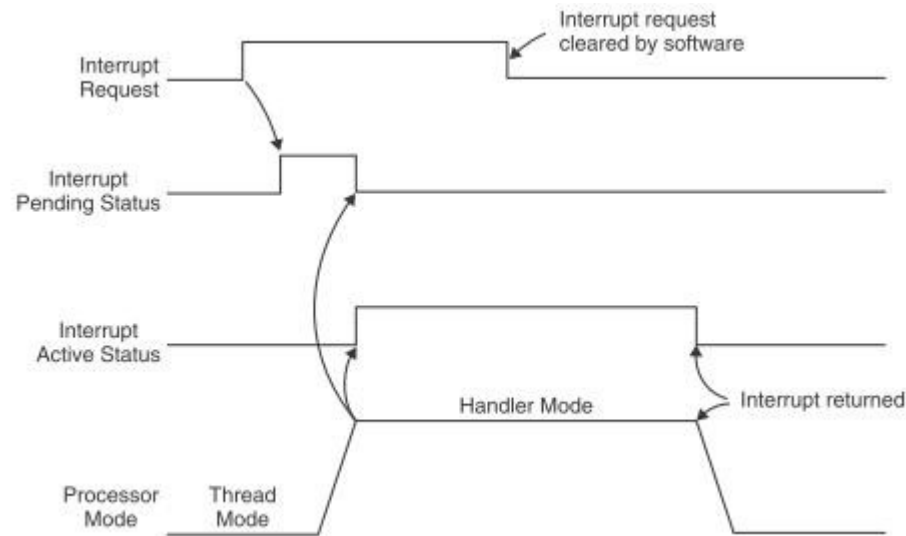


Interrupt Pending Cleared Before Processor Takes Action

- If the pending status is cleared before the processor starts responding to the pending interrupt, the interrupt can be canceled. (for example, because pending status register is cleared while PRIMASK/FAULTMASK is set to 1)
- The pending status of the interrupt can be accessed in the NVIC and is writable, so you can clear a pending interrupt or use software to pend a new interrupt by setting the pending register.



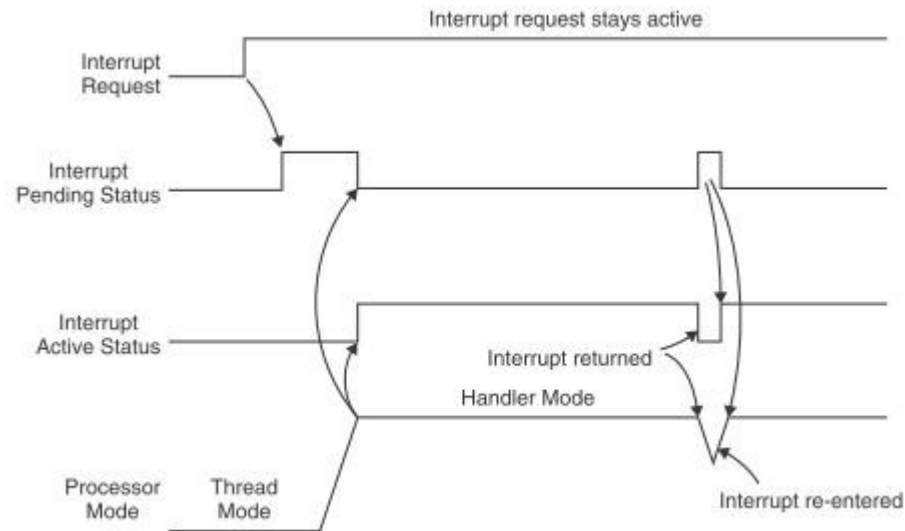
# Interrupt Inputs and Pending Behavior (3)



Interrupt Active Status Set as Processor Enters Handler

- When the processor starts to execute an interrupt, the interrupt becomes active and the pending bit will be cleared automatically.
- When an interrupt is active, you cannot start processing the same interrupt again until the interrupt service routine is terminated with an interrupt return (also called an exception exit).
- Then the active status is cleared and the interrupt can be processed again if the pending status is 1.

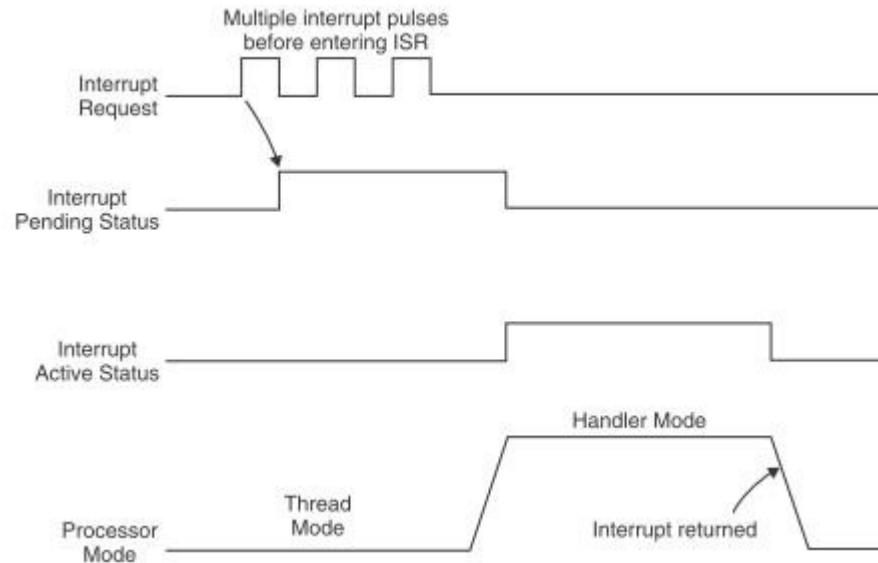
# Interrupt Inputs and Pending Behavior (4)



Continuous Interrupt Request Pends Again After Interrupt Exit

- 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.
- This is just like the traditional ARM7TDMI.

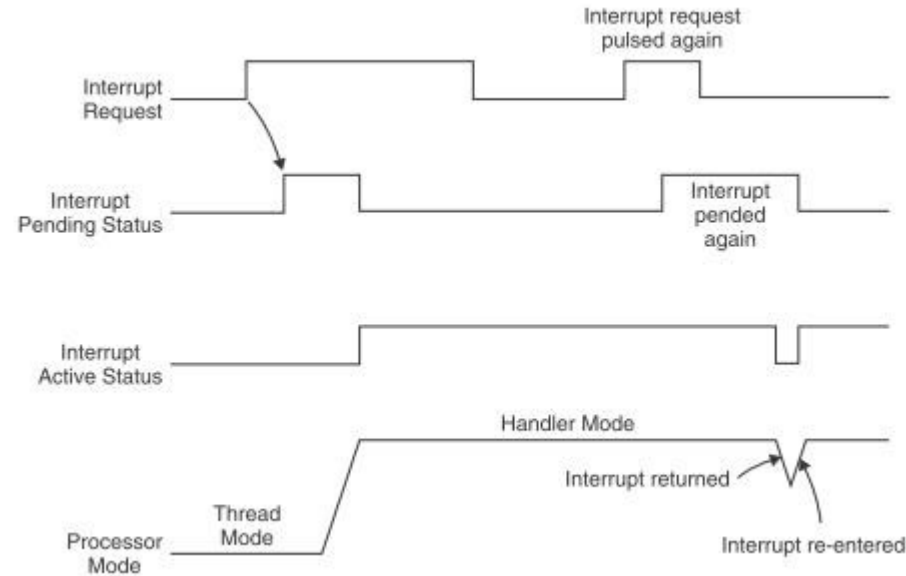
## Interrupt Inputs and Pending Behavior (5)



Interrupt Pending Only Once, Even with Multiple Pulses Before the Handler

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

# Interrupt Inputs and Pending Behavior (6)



Interrupt Pending Occurs Again During the Handler

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

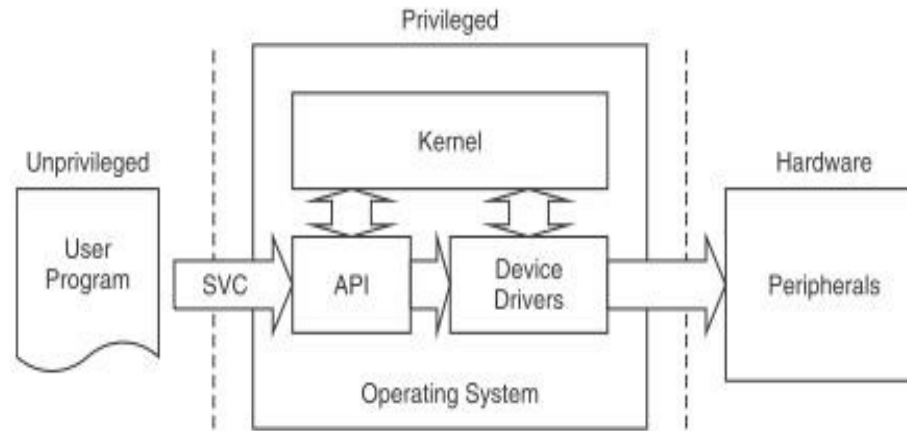
# Fault Exceptions

- A number of system exceptions are useful for fault handling.

There are several categories of faults:

- Bus faults
- Memory management faults
- Usage faults
- Hard faults

# SVC and PendSV

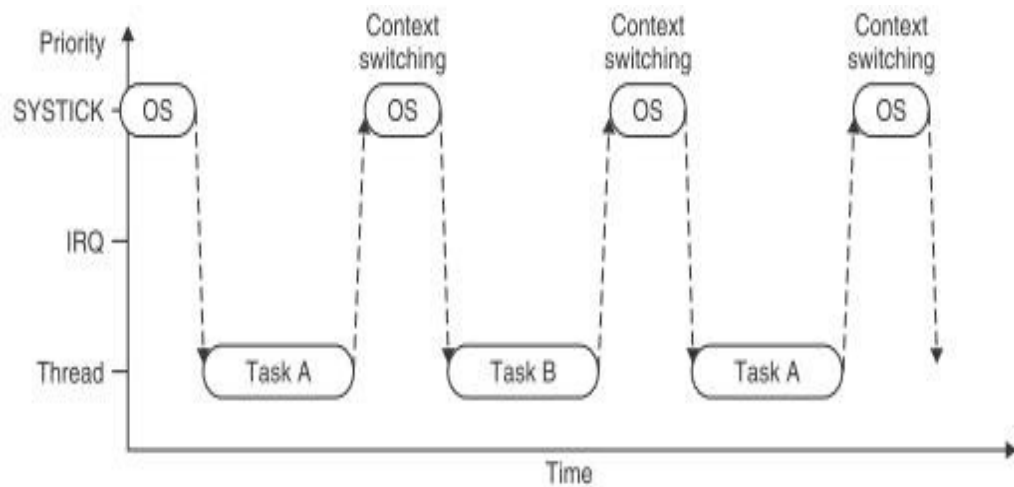


SVC as a Gateway for OS Functions

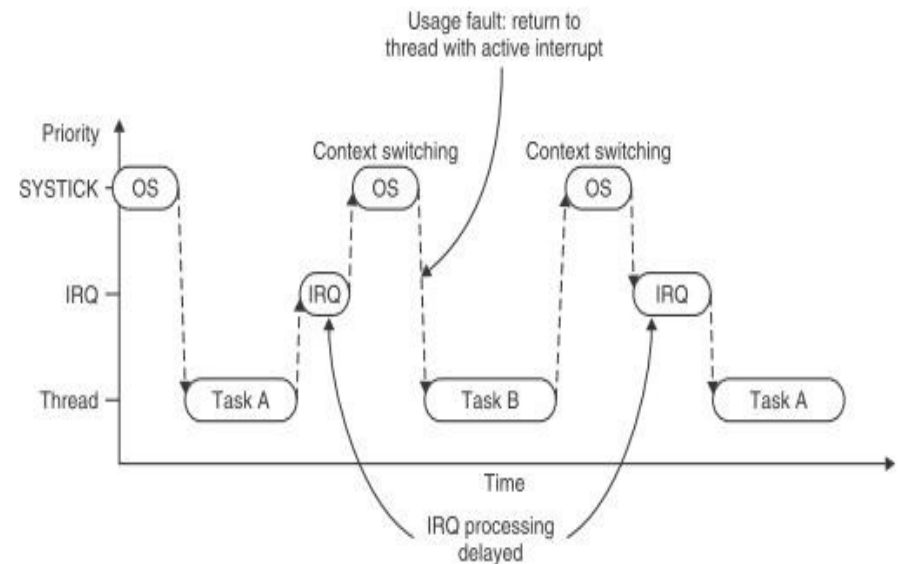
- SVC (System Service Call) and PendSV (Pended System Call) are two exceptions targeted at software and operating systems.
- SVC is for generating system function calls.
- Which can provide a more robust system by preventing the user applications from directly accessing hardware.
- The actual hardware-level programming is handled by device drivers.

# SVC and PendSV (1)

- PendSV (Pended System Call) works with SVC in the OS. Although SVC (by SVC instruction) cannot be pended (an application calling SVC will expect the required task to be done immediately), PendSV can be pended and is useful for an OS to pend an exception so that an action can be performed after other important tasks are completed.
- PendSV is generated by writing 1 to the NVIC PendSV pending register.

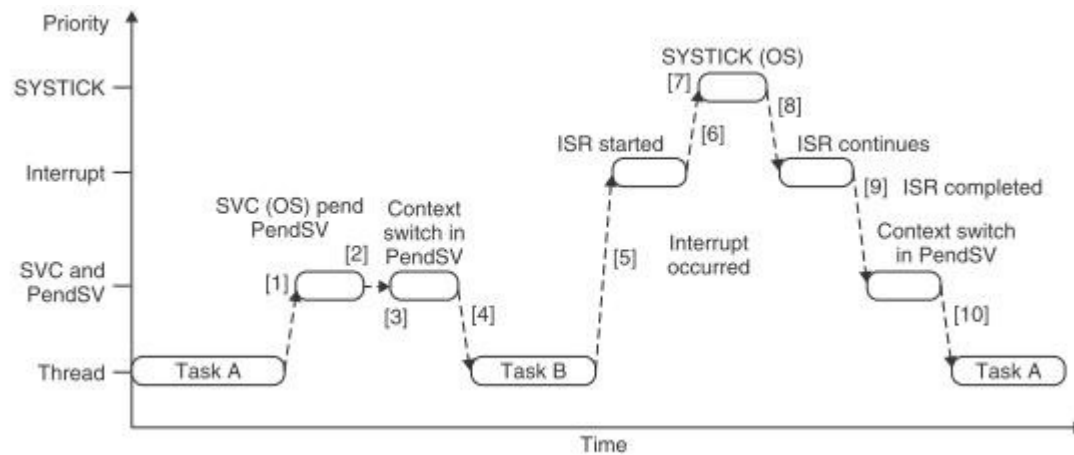


## A Simple Scenario Using SYSTICK to Switch Between Two Tasks



## Problem with Context Switching at the IRQ

# SVC and PendSV (2)



Example Context Switching with PendSV

1. Task A calls SVC for task switching (for example, waiting for some work to complete).
2. The OS receives the request, prepares for context switching, and pends the PendSV exception.
3. When the CPU exits SVC, it enters PendSV immediately and does the context switch.
4. When PendSV finishes and returns to Thread level, it executes Task B.
5. An interrupt occurs and the interrupt handler is entered.
6. While running the interrupt handler routine, a SYSTICK exception (for OS tick) takes place.
7. The OS carries out the essential operation, then pends the PendSV exception and gets ready for the context switch.
8. When the SYSTICK exception exits, it returns to the interrupt service routine.
9. When the interrupt service routine completes, the PendSV starts and does the actual context switch operations.
10. When PendSV is complete, the program returns to Thread level; this time it returns to Task A and continues the processing.



## SVC and SWI (ARM7)

- If you have used traditional ARM processors (such as the ARM7), you might know that they have a software interrupt instruction (SWI).
- The SVC has a similar function, and in fact the binary encoding of SVC instructions is the same as SWI in ARM7.

However, since the exception model has changed, this instruction is renamed to make sure that programmers will properly port software code from ARM7 to the Cortex-M3.

# The NVIC and Interrupt Control

# NVIC Overview

- NVIC – Nested Vectored Interrupt Controller.
- An integrated part of the Cortex-M3 processor.
- Its control registers are accessible as memory-mapped devices.
- NVIC also contains control registers for the MPU, the SYSTICK Timer, and debugging controls.
- The NVIC supports 1 to 240 external interrupt inputs (IRQs).
- NVIC also has a Nonmaskable Interrupt (NMI) input.
- NVIC can be accessed as memory location 0xE000E000.

# NVIC

## The Basic Interrupt Configuration

- Each external interrupt has several registers associated with it:
  - Enable and clear enable registers
  - Set-pending and clear-pending registers
  - Priority level
  - Active status
- A number of other registers can also affect the interrupt processing:
  - Exception-masking registers. (PRIMASK, FAULTMASK, and BASEPRI)
  - Vector Table Offset register
  - Software Trigger Interrupt register
  - Priority Group

# NVIC

## Interrupt Enable and Clear Enable

- The SETENA/CLRENA registers are 32 bits wide; each bit represents one interrupt input.
- Since the first 16 exception types are system exceptions, external interrupt #0 has a start exception number of 16.

| Address    | Name    | Type | Reset Value | Description   |
|------------|---------|------|-------------|---|
| 0xE000E100 | SETENA0 | R/W  | 0           | Enable for external interrupt #0-31<br>bit[0] for interrupt #0 (exception #16)<br>bit[1] for interrupt #1 (exception #17)<br>...<br>bit[31] for interrupt #31<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current enable status |
| 0xE000E104 | SETENA1 | R/W  | 0           | Enable for external interrupt #32-63<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current enable status  |
| 0xE000E108 | SETENA2 | R/W  | 0           | Enable for external interrupt #64-95<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current enable status  |
| ...        | -       | -    | -           | -   |
| 0xE000E180 | CLRENA0 | R/W  | 0           | Clear enable for external interrupt #0-31<br>bit[0] for interrupt #0<br>bit[1] for interrupt #1<br>...<br>bit[31] for interrupt #31<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current enable status                         |
| 0xE000E184 | CLRENA1 | R/W  | 0           | Clear Enable for external interrupt #32-63<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current enable status  |
| 0xE000E188 | CLRENA2 | R/W  | 0           | Clear enable for external interrupt #64-95<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current enable status  |
| ...        | -       | -    | -           | -   |

Interrupt Set Enable Registers and Interrupt Clear Enable Registers  
(0xE000E100-0xE000E11C, 0xE000E180-0xE000E19C)

# NVIC

## Interrupt Pending and Clear Pending

- If an interrupt takes place but cannot be executed immediately, it will be pended.

For instance, if another higher-priority interrupt handler is running.

| Address    | Name    | Type | Reset Value | Description   |
|------------|---------|------|-------------|---|
| 0xE000E100 | SETENA0 | R/W  | 0           | Enable for external interrupt #0-31<br>bit[0] for interrupt #0 (exception #16)<br>bit[1] for interrupt #1 (exception #17)<br>...<br>bit[31] for interrupt #31<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current enable status |
| 0xE000E104 | SETENA1 | R/W  | 0           | Enable for external interrupt #32-63<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current enable status  |
| 0xE000E108 | SETENA2 | R/W  | 0           | Enable for external interrupt #64-95<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current enable status  |
| ...        | -       | -    | -           | -   |
| 0xE000E180 | CLRENA0 | R/W  | 0           | Clear enable for external interrupt #0-31<br>bit[0] for interrupt #0<br>bit[1] for interrupt #1<br>...<br>bit[31] for interrupt #31<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current enable status                         |
| 0xE000E184 | CLRENA1 | R/W  | 0           | Clear Enable for external interrupt #32-63<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current enable status  |
| 0xE000E188 | CLRENA2 | R/W  | 0           | Clear enable for external interrupt #64-95<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current enable status  |
| ...        | -       | -    | -           | -   |

Interrupt Set Enable Registers and Interrupt Clear Enable Registers  
(0xE000E100-0xE000E11C, 0xE000E180-0xE000E19C)

# NVIC – Priority Levels

- Each register can be further divided into preempt priority level and subpriority level based on priority group settings.  
(Maximum width of 8 bits and a minimum width of 3 bits.)

| Address    | Name   | Type | Reset Value | Description                           |
|------------|--------|------|-------------|---------------------------------------|
| 0xE000E400 | PRI_0  | R/W  | 0 (8-bit)   | Priority-level external interrupt #0  |
| 0xE000E401 | PRI_1  | R/W  | 0 (8-bit)   | Priority-level external interrupt #1  |
| ...        | –      | –    | –           | –                                     |
| 0xE000E41F | PRI_31 | R/W  | 0 (8-bit)   | Priority-level external interrupt #31 |
| ...        | –      | –    | –           | –                                     |

Interrupt Priority-Level Registers (0xE000E400-0xE000E4EF)

| Address    | Name   | Type | Reset Value | Description                                |
|------------|--------|------|-------------|--|
| 0xE000ED18 | PRI_4  | R/W  | 0           | Priority level for memory management fault |
| 0xE000ED19 | PRI_5  | R/W  | 0           | Priority level for bus fault               |
| 0xE000ED1A | PRI_6  | R/W  | 0           | Priority level for usage fault             |
| 0xE000ED1B | –      | –    | –           | –  |
| 0xE000ED1C | –      | –    | –           | –  |
| 0xE000ED1D | –      | –    | –           | –  |
| 0xE000ED1E | –      | –    | –           | –  |
| 0xE000ED1F | PRI_11 | R/W  | 0           | Priority level for SVC                     |
| 0xE000ED20 | PRI_12 | R/W  | 0           | Priority level for debug monitor           |
| 0xE000ED21 | –      | –    | –           | –  |
| 0xE000ED22 | PRI_14 | R/W  | 0           | Priority level for PendSV                  |
| 0xE000ED23 | PRI_15 | R/W  | 0           | Priority level for SYSTICK                 |

System Exceptions Priority-Level Register  
(0xE000ED18–0xE000ED23; Listed as Byte Addresses)

# NVIC – Active Status

- Each external interrupt has an active status bit.
- When the processor starts the interrupt handler, the bit is set to 1 and cleared when the interrupt return is executed.

| Address    | Name    | Type | Reset Value | Description  |
|------------|---------|------|-------------|--|
| 0xE000E300 | ACTIVE0 | R    | 0           | Active status for external interrupt #0-31<br>bit[0] for interrupt #0<br>bit[1] for interrupt #1<br>...<br>bit[31] for interrupt #31 |
| 0xE000E304 | ACTIVE1 | R    | 0           | Active status for external interrupt #32-63  |
| ...        | -       | -    | -           | -  |

Interrupt Active Status Registers (0xE000E300-0xE000E31C)



# NVIC

## Example Procedures in Setting Up an Interrupt

1. When the system boots up, the priority group register might need to be set up. By default the priority group 0 is used.
2. Copy the hard fault and NMI handlers to a new vector table location if vector table relocation is required.
3. The Vector Table Offset register should also be set up to get the vector table ready.
4. Set up the interrupt vector for the interrupt.  
Since the vector table could have been relocated, you might need to read the Vector Table Offset register, then calculate the correct memory location for your interrupt handler.
5. Set up the priority level for the interrupt.
6. Enable the interrupt.

Make sure that you have enough stack memory if you allow a large number of nested interrupt levels.

# Interrupt Behavior

# Interrupt/Exception Sequences

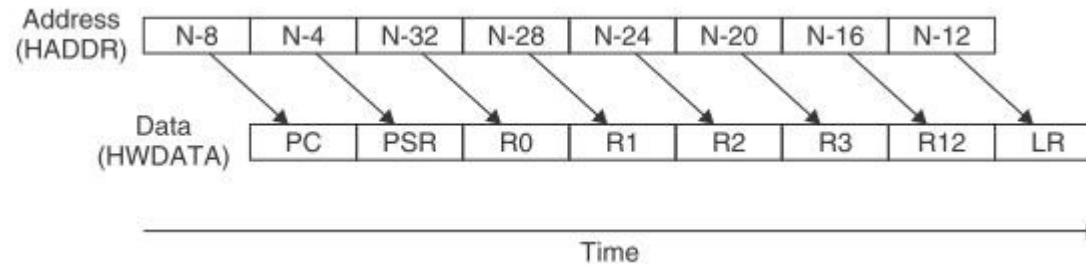
- When an exception takes place, a number of things happen:
  - Stacking (pushing eight registers contents to stack)
  - Vector fetch (reading the exception handler starting address from the vector table)
  - Update of the stack pointer(SP), link register(LR), and program counter(PC)

# Stacking(1)

- When an exception takes place, the registers PC, PSR, R0 – R3, R12, and LR are pushed to the stack.
- If the code that is running uses the PSP, the process stack will be used.  
If the code that is running uses the MSP, the main stack will be used.
- Afterward, the main stack will always be used during the handler, so all nested interrupts will use the main stack.
- The reason the registers R0—R3, R12, LR, PC, and PSR are stacked is that these are caller saved registers, according to C standards. (C/C++ standard Procedure Call Standard for the ARM Architecture, AAPCS, Ref 5)
- The general registers (R0—R3, R12) are located at the end of the stack frame so that they can be easily accessed using SP-related addressing.

# Stacking (2)

(assuming that the SP value is N before the exception)



Stacking Sequence

| Address          | Data                     | Push Order |
|------------------|--------------------------|------------|
| Old SP (N) ->    | (Previously pushed data) | -          |
| (N-4)            | PSR                      | 2          |
| (N-8)            | PC                       | 1          |
| (N-12)           | LR                       | 8          |
| (N-16)           | R12                      | 7          |
| (N-20)           | R3                       | 6          |
| (N-24)           | R2                       | 5          |
| (N-28)           | R1                       | 4          |
| New SP (N-32) -> | R0                       | 3          |

Stack Memory Content After Stacking and Stacking Order

# Vector Fetches

- While the data bus is busy stacking the registers, the instruction bus carries out another important task of the interrupt sequence.
- It fetches the exception vector from the vector table.
- Since the stacking and vector fetch are performed on separate bus interfaces, they can be carried out at the same time.

# Register Updates

- After the stacking and vector fetch are completed, the exception vector will start to execute. On entry of the exception handler, a number of registers will be updated:
  - SP: The Stack Pointer (either the MSP or the PSP) will be updated to the new location during stacking.  
During execution of the interrupt service routine, the MSP will be used if the stack is accessed.
  - PSR: The IPSR (the lowest part of the PSR) will be updated to the new exception number.
  - PC: This will change to the vector handler as the vector fetch completes and starts fetching instructions from the exception vector.
  - LR: The LR will be updated to a special value called EXC\_RETURN. This special value drives the interrupt return operation. The last 4 bits of the LR have a special meaning.
- A number of other NVIC registers will also be updated.

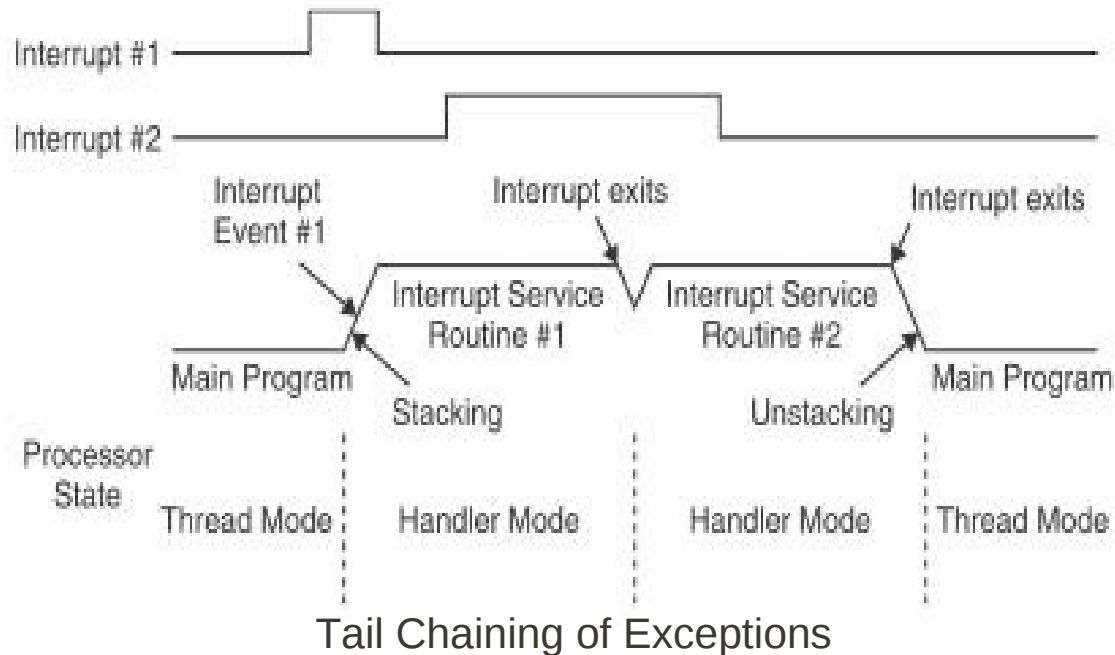
# Exception Exits

- At the end of the exception handler, an exception exit (known as an interrupt return in some processors) is required to restore the system status so that the interrupted program can resume normal execution.
- When the interrupt return instruction is executed, the following processes are carried out:
  1. Unstacking:  
The registers pushed to the stack will be restored.
  2. NVIC register update:  
The active bit of the exception will be cleared.

| Return Instruction                          | Description  |
|---|--|
| <code>BX &lt;reg&gt;</code>                 | If the <code>EXC_RETURN</code> value is still in <code>LR</code> , we can use the <code>BX LR</code> instruction to perform the interrupt return.  |
| <code>POP {PC}, or<br/>POP {..., PC}</code> | Very often the value of <code>LR</code> is pushed to the stack after entering the exception handler. We can use the <code>POP</code> instruction, either a single <code>POP</code> or multiple <code>POPs</code> , to put the <code>EXC_RETURN</code> value to the program counter. This will cause the processor to perform the interrupt return. |
| <code>LDR, or LDM</code>                    | It is possible to produce an interrupt return using the <code>LDR</code> instruction with <code>PC</code> as the destination register.   |

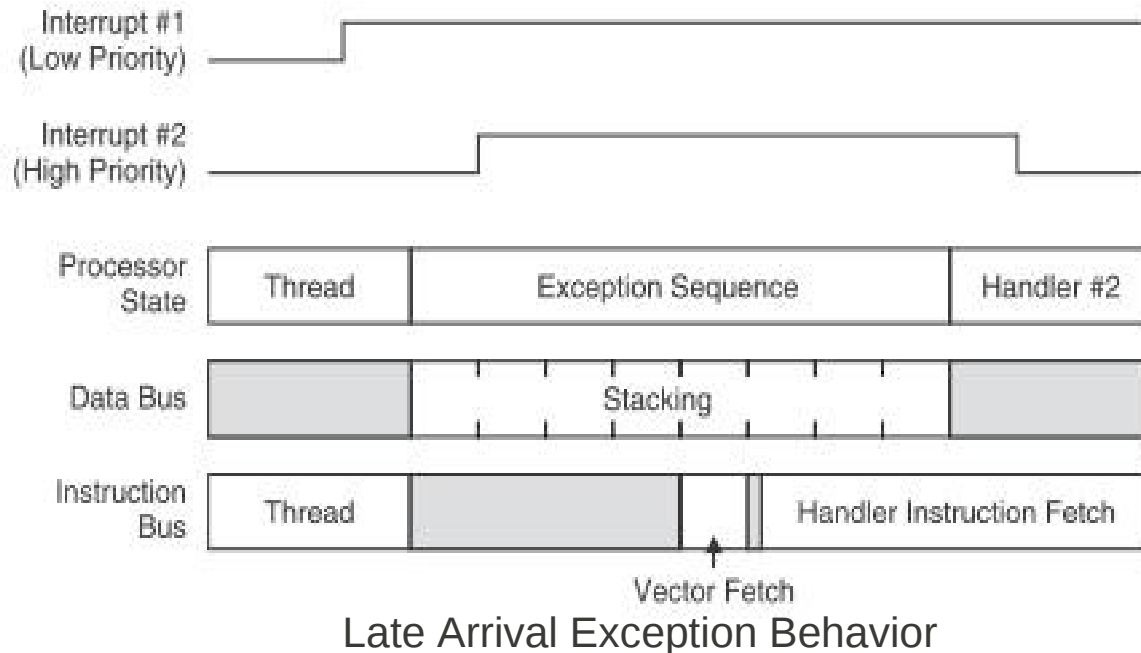


# Tail-Chaining Interrupts



- When an exception takes place but the processor is handling another exception of the same or higher priority, the exception will be pended.
- Skipping the unstacking and the stacking.
- The timing gap between the two exception handlers is greatly reduced.

# Late Arrivals



- When an exception takes place and the processor has started the stacking process, and if during this delay a new exception arrives with higher preemption priority, the late arrival exception will be processed first.

# More on the Exception Return Value(1)

- When entering an exception handler, the LR is updated to a special value called EXC\_RETURN, with the upper 28 bits all set to 1.
- When loaded into the PC at the end of the exception handler execution, will cause the processor to perform an exception return sequence.
- The EXC\_RETURN value has bit [31:4] all set to 1, and bit[3:0] provides information required by the exception return operation.
- When the exception handler is entered, the LR value is updated automatically, so there is no need to generate these values manually.

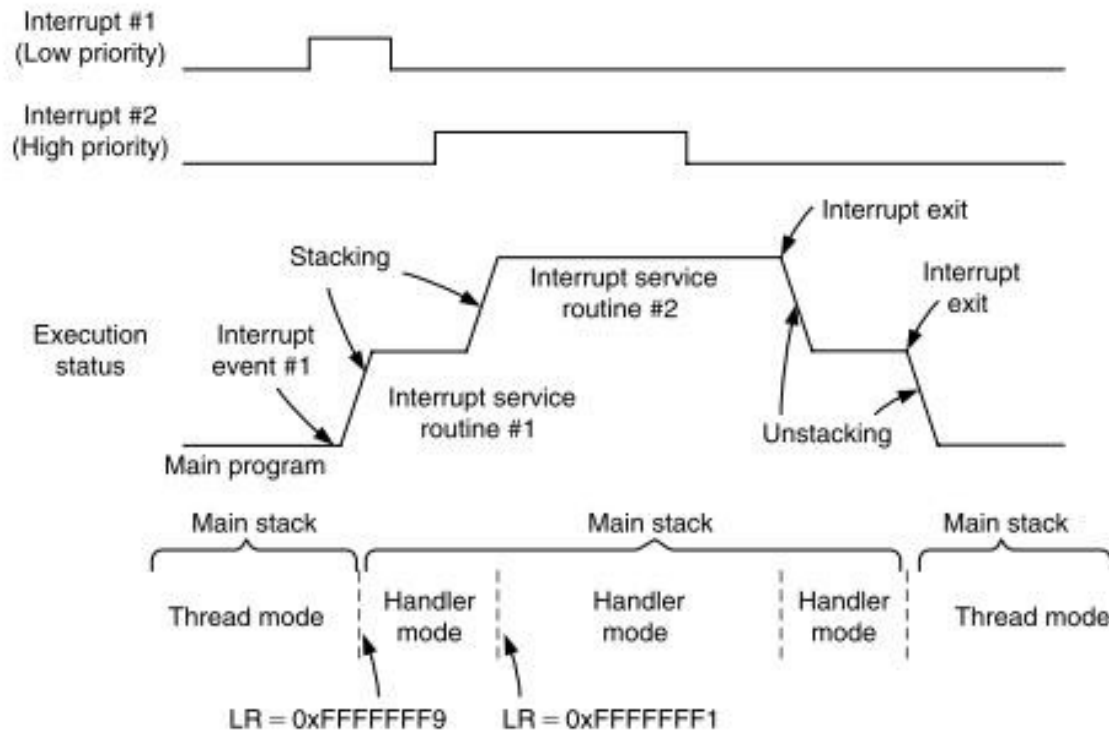
| Bits         | 31:4       | 3                               | 2            | 1                         | 0                            |
|--------------|------------|---------------------------------|--------------|---------------------------|------------------------------|
| Descriptions | 0xFFFFFFFF | Return mode<br>(Thread/handler) | Return stack | Reserved;<br>must be<br>0 | Process state<br>(Thumb/ARM) |

Description of Bit Fields in EXC\_RETURN Value

| Value       | Condition   |
|-------------|---|
| 0xFFFFFFFF1 | Return to handler mode                                    |
| 0xFFFFFFFF9 | Return to Thread mode and on return use the main stack    |
| 0xFFFFFFF9D | Return to Thread mode and on return use the process stack |

Allowed EXC\_RETURN Values on Cortex-M3

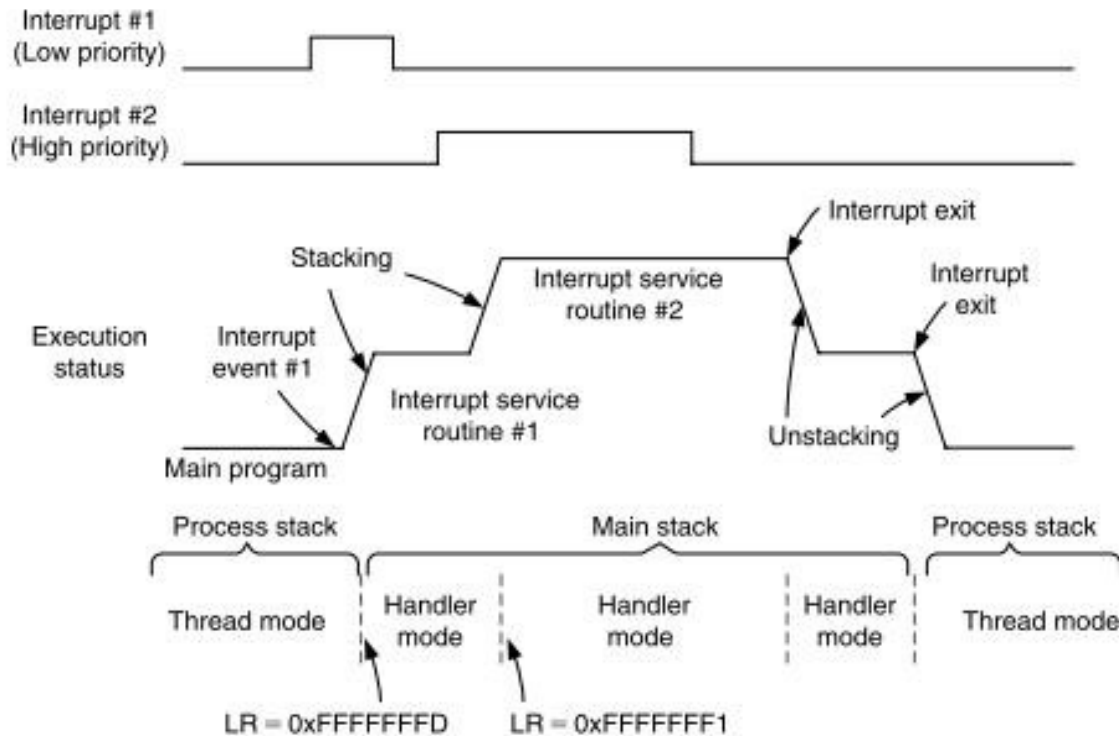
## More on the Exception Return Value(2)



LR Set to EXC\_RETURN at Exception (Main Stack Used in Thread Mode)

- If the thread is using the MSP (main stack), the value of LR will be set to 0xFFFFFFFF9 when it enters an exception, and 0xFFFFFFFF1 when a nested exception is entered.

## More on the Exception Return Value(3)



LR Set to EXC\_RETURN at Exception (Process Stack Used in Thread Mode)

- If the thread is using PSP (process stack), the value of LR would be 0xFFFFFFFFD when entering the first exception and 0xFFFFFFFF1 for entering a nested exception.

# Reference

- Definitive\_Guide\_To\_The\_ARM\_Cortex\_M3
- ARM ® v7-M Architecture Reference Manual
- ARM\_Architecture\_Overview
- STM32F103V100\_Manual