Interrupts and Nested Interrupt Handling

1. Interrupts

1.1 Definition

An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service.

1.2 Serving Several Devices

A single microcontroller can serve several devices in two ways: **interrupt** and **polling**.

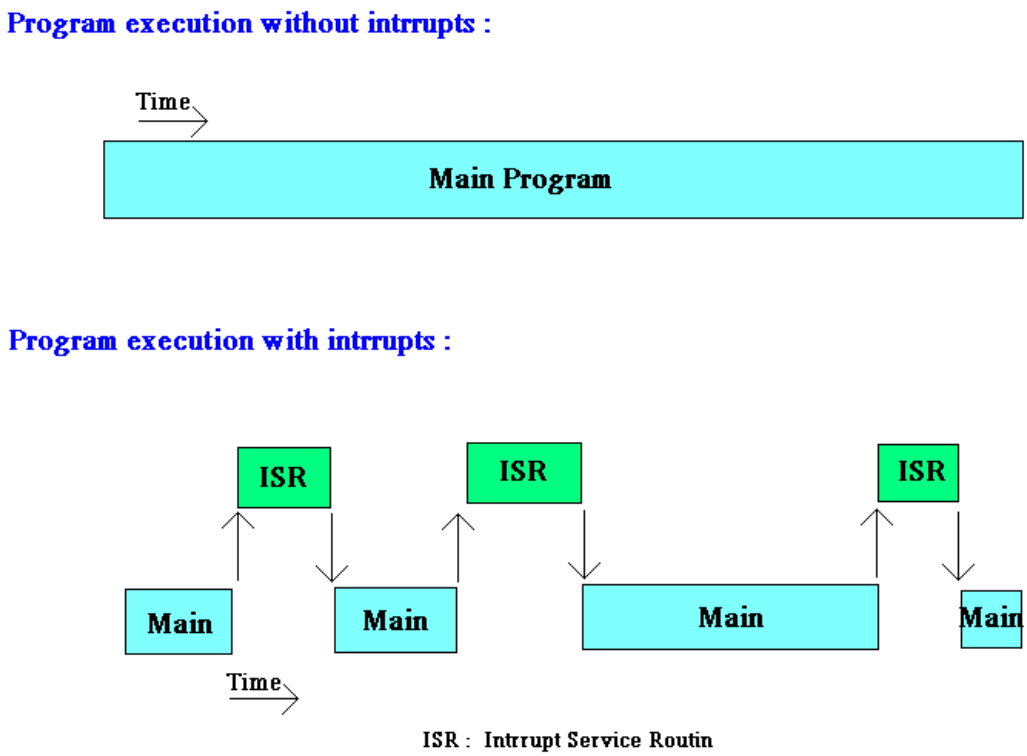


Figure 2.1 Program execution without and with interrupts (ISR = Interrupt Service Routine)

1.3 Steps in Executing an Interrupt

1. Finish the current instruction and save the PC on the stack.
2. Jump to a fixed memory location that depends on the type of interrupt.
3. Execute the interrupt service routine until RETI (return from interrupt).
4. On RETI the microcontroller returns to the place where it was interrupted (the PC is popped from the stack).

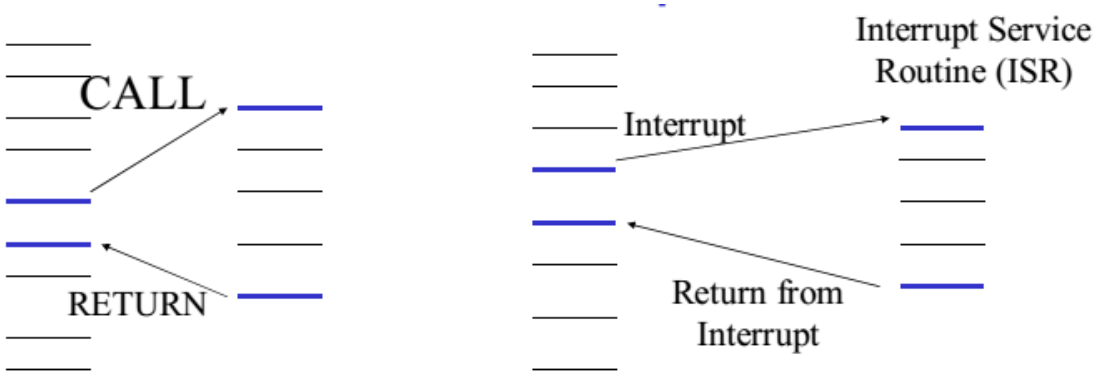


Figure 2.2 Subroutine CALL/RETURN compared with Interrupt/Return from interrupt

2. Types of Interrupts

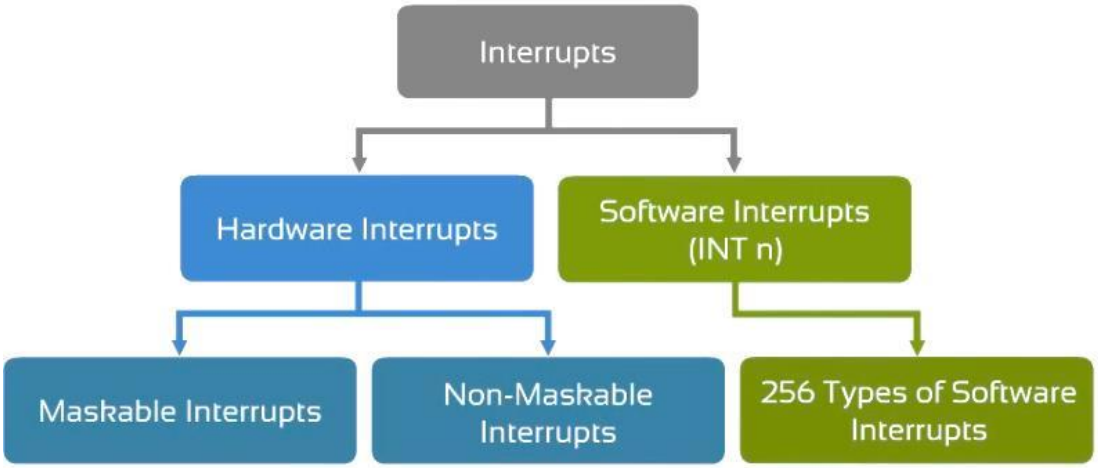


Figure 2.3 Classification of interrupts

Type	Description
Hardware interrupts	Generated by hardware devices; **external** (from outside the chip) or **internal** (from on-chip peripherals)
Software interrupts	Generated by software instructions (INT n; 256 types); used for exceptions and operating-system services

2.1 Maskable vs Non-Maskable Interrupts

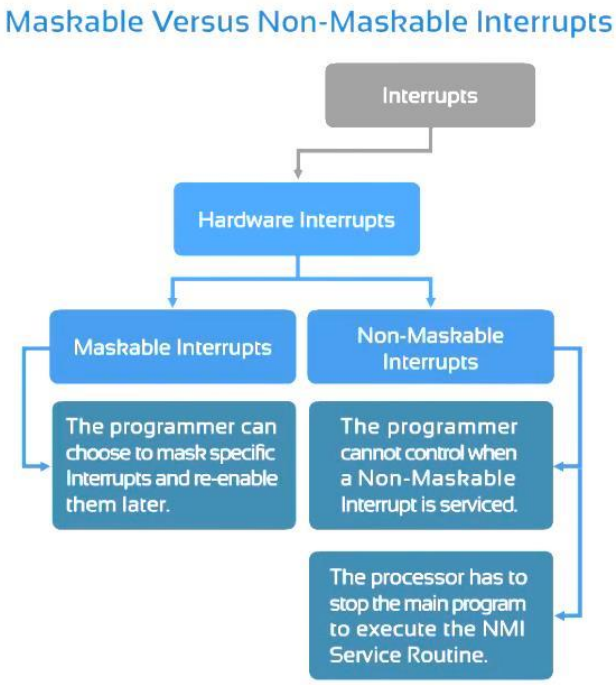


Figure 2.4 Maskable versus non-maskable interrupts

Maskable interrupts	**Non-maskable interrupts (NMI)**
The programmer can choose to mask specific interrupts and re-enable them later.	The programmer cannot control when an NMI is serviced; the processor has to stop the main program to execute the NMI service routine.

3. Nested Interrupt Handling

Nested interrupt handling means the software is prepared to accept another interrupt even before it finishes handling the current one. This allows interrupts to be prioritised and greatly improves the latency of high-priority events, at the cost of additional complexity.

Note: Nesting is a choice made by the software through interrupt priority configuration and interrupt control; it is not imposed by hardware.

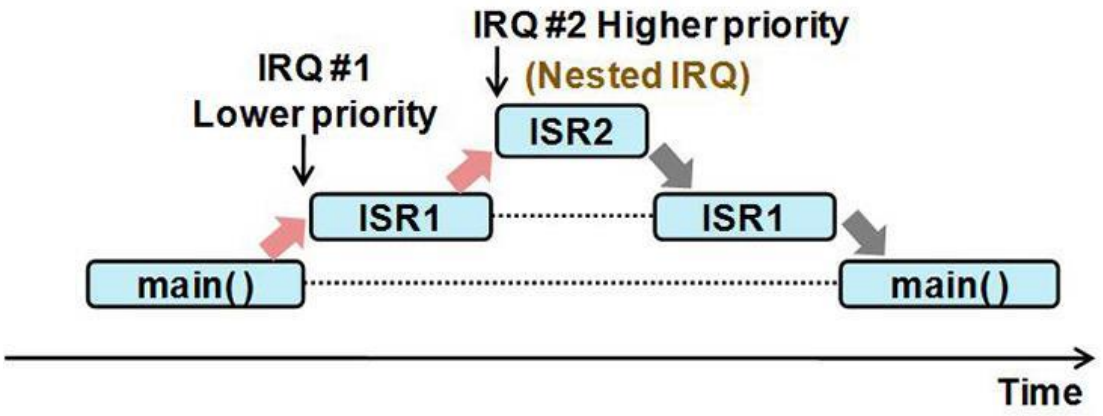


Figure 2.5 Nested interrupt: higher-priority IRQ #2 pre-empts ISR1

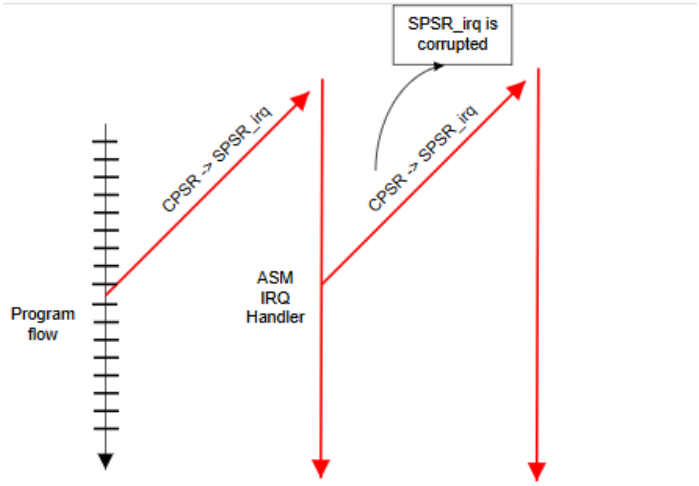


Figure 2.6 Program flow in a nested IRQ handler (SPSR_irq is corrupted if not saved)

Interrupt priority determines which interrupt is serviced first when multiple interrupts occur.

4. NVIC and Interrupt Vector Table

4.1 NVIC Features (Cortex-M4)

The NVIC belongs to the Cortex-M4 CPU, enabling close coupling with the processor core. Main features:

- 102 interrupt sources
- 16 programmable priority levels
- Low-latency exception and interrupt handling
- Automatic nesting
- Power management control

4.2 Interrupt Vector Table (IVT)

A table stored in memory containing the starting addresses of exception and interrupt service routines (ISRs). On an interrupt, the processor uses the table to find which ISR to run.

Sequence: Peripheral generates → NVIC receives → NVIC checks priority → Vector table identifies ISR → Cortex-M executes ISR → processor returns to the main program.

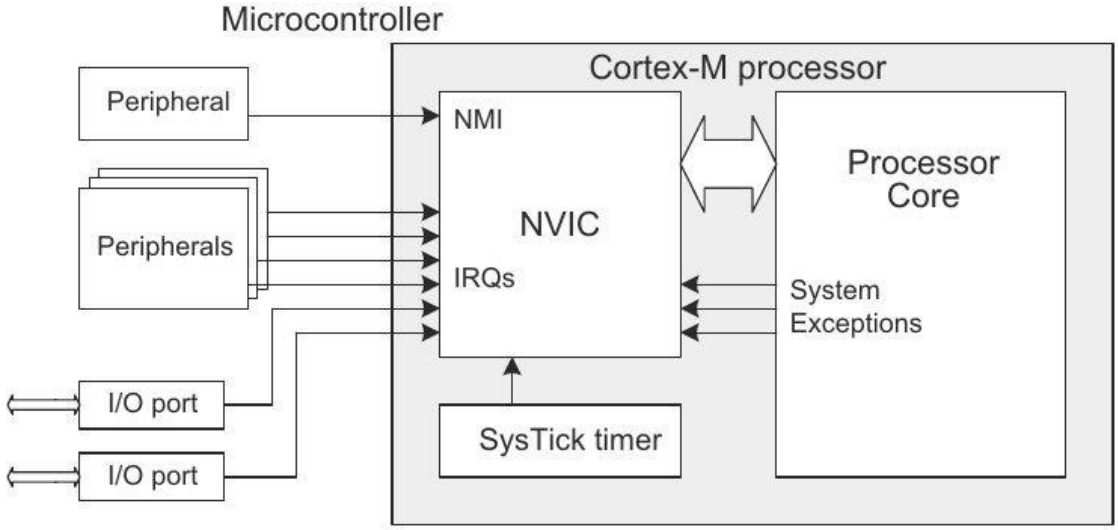


Figure 2.7 NVIC with peripherals, I/O ports, NMI, system exceptions and SysTick timer

4.3 How the Cortex-M Handles an Interrupt

1. A peripheral generates an interrupt; the request goes to the NVIC.
2. The NVIC checks whether the interrupt is enabled and determines its priority.
3. The highest-priority pending interrupt is selected and the core is directed to the corresponding ISR via the vector mechanism.
4. The processor suspends the main program, executes the ISR, then returns to the interrupted program.

The SysTick timer can also generate periodic interrupts through the interrupt system, while NMI and system exceptions provide special mechanisms for critical system events.