ARM Microcontroller Internship Programme

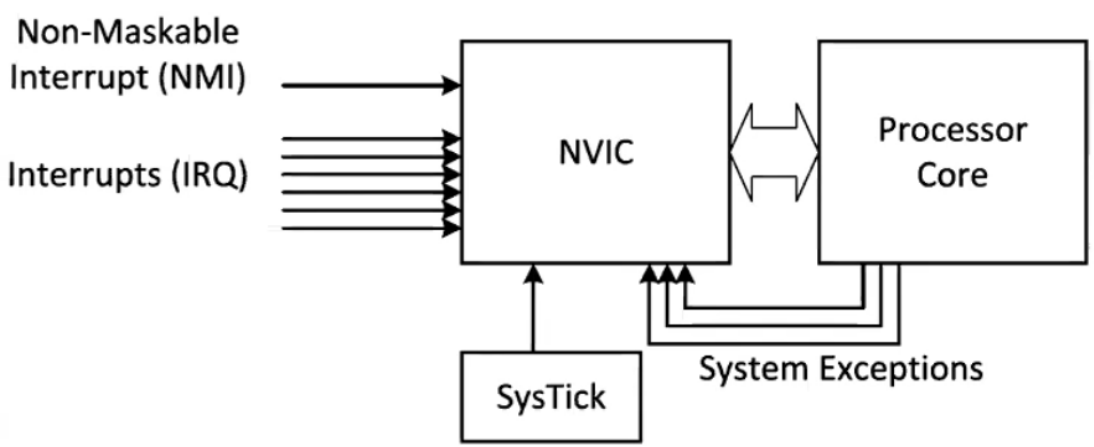
Trainer: *Kiran Jaybhave*

Sunbeam – Industrial Training Programme

Sunbeam Pune (Hinjawadi)

Cortex-M3 Exception and Interrupt Architecture

- **In ARM architecture, Interrupts are a special type of exception.**
- **Cortex-M3 uses a unified exception model for:**
  - **Processor exceptions**
  - **System exceptions**
  - **Peripheral interrupts**



- Role of **NVIC** (Nested Vectored Interrupt Controller)
- **NVIC is Tightly coupled with the Cortex-M3 core.**
- **It receives:**
  - Non-Maskable Interrupt (NMI)
  - External Interrupt Requests (IRQ)
  - System exceptions (SysTick, PendSV, etc.)
- **NVIC:**
  - Resolves interrupt priority
  - Supports nesting
  - Signals the processor core to handle the exception

- **System Exceptions** System exceptions are **generated internally by the processor core**. They handle **critical system events, errors, and operating system services**.
- **Examples of System Exceptions:**
  - **Reset** – Occurs after power-up or system reset
  - **NMI (Non-Maskable Interrupt)** – High-priority emergency interrupt
  - **HardFault** – Occurs due to severe system errors
  - **MemManage** – Memory protection violation
  - **BusFault** – Bus access error
  - **UsageFault** – Illegal instruction or execution error
  - **SVCall** – Used by OS to request system services
  - **DebugMonitor** – Used during debugging
  - **PendSV** – Used for context switching in RTOS
  - **SysTick** – System timer interrupt

Peripheral Interrupts

- Peripheral interrupts are **generated by on-chip peripherals** such as GPIO, Timers, UART, ADC, etc.
  - **Key Points:**
    - Generated outside the CPU core
    - Routed to the processor via **NVIC (Nested Vectored Interrupt Controller)**
    - Each peripheral has a **dedicated interrupt number**
    - Total number of interrupts depends on the **microcontroller variant**

SysTick Exception

- **SysTick** is a **dedicated 24-bit system timer** built into the Cortex-M3 core.
- **Purpose of SysTick:**
  - Generates **periodic interrupts**
  - Provides a **time base** for the system
  - Commonly used for:
    - OS tick generation (RTOS) , Time delays , Task scheduling

- **Single Vector Table Cortex-M3 uses one common vector table for:**
  - **Exceptions**
  - **Interrupts**
  - **Vector table entries are indexed by exception number.**

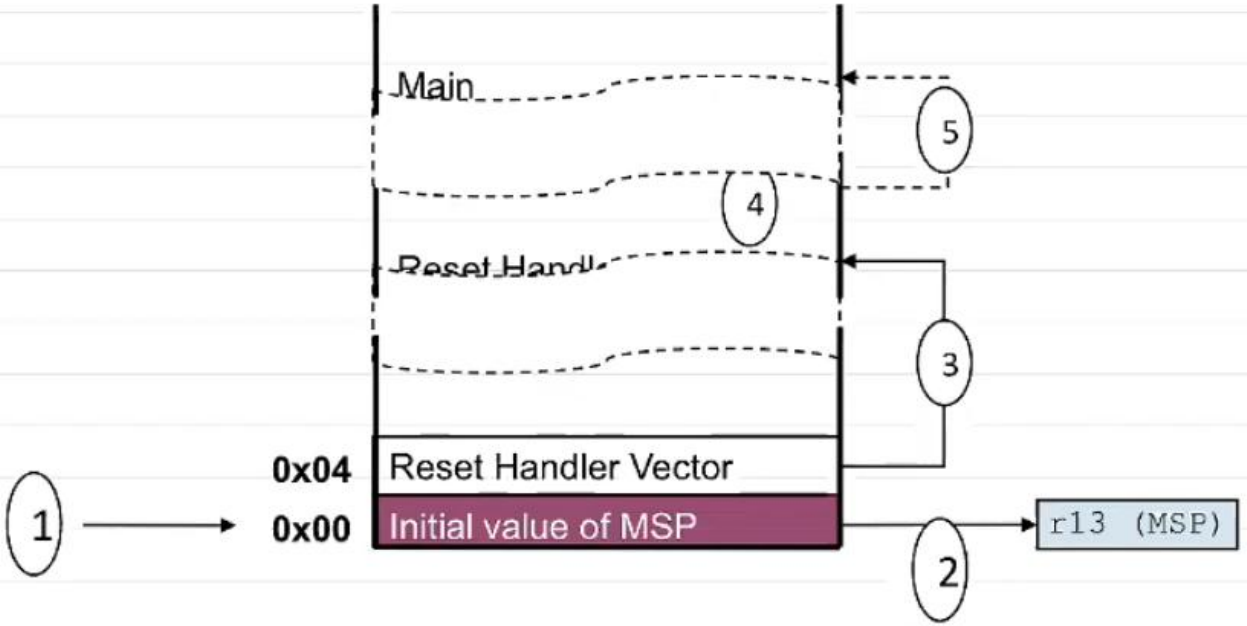
Vector No.	Exception / Interrupt	Type	IRQ No.
0	Initial MSP	System	—
1	Reset	System	−15
2	NMI	System	−14
3	HardFault	System	−13
4	MemManage	System	−12
5	BusFault	System	−11
6	UsageFault	System	−10
7	Reserved	—	—
8	Reserved	—	—
9	Reserved	—	—
10	Reserved	—	—
11	SVCall	System	−5
12	DebugMonitor	System	−4
13	Reserved	—	—
14	PendSV	System	−2
15	SysTick	System	−1
16	Peripheral Interrupt 0	Peripheral	0
17	Peripheral Interrupt 1	Peripheral	1
...	...	...	...
255	Peripheral Interrupt 239	Peripheral	239

Cortex-M3 Vector Table

- **Vector Table**
  - Cortex-M3 uses a **single vector table** for all exceptions and interrupts
  - Vector table consists of **32-bit entries**
  - **Entry address = Base Address + (4 × Vector Number)**
- **First Vector Table Entry**
  - **Entry 0 contains Initial Main Stack Pointer (MSP)**
  - Loaded automatically by hardware on reset
  - **Not a function address**
- **Exception vs Interrupt Position**
  - Vectors **1–15** → Core exceptions
  - Vectors **16 onwards** → Peripheral interrupts
  - Formula:
  - IRQ N → Vector number = 16 + N
- **LSB = 1 : Rule (Thumb)**
  - Handler address **must have LSB = 1**
  - Cortex-M executes **Thumb instructions only**
- **Generating Vector Table in C**
  - Function pointer arrays
  - Startup code internals

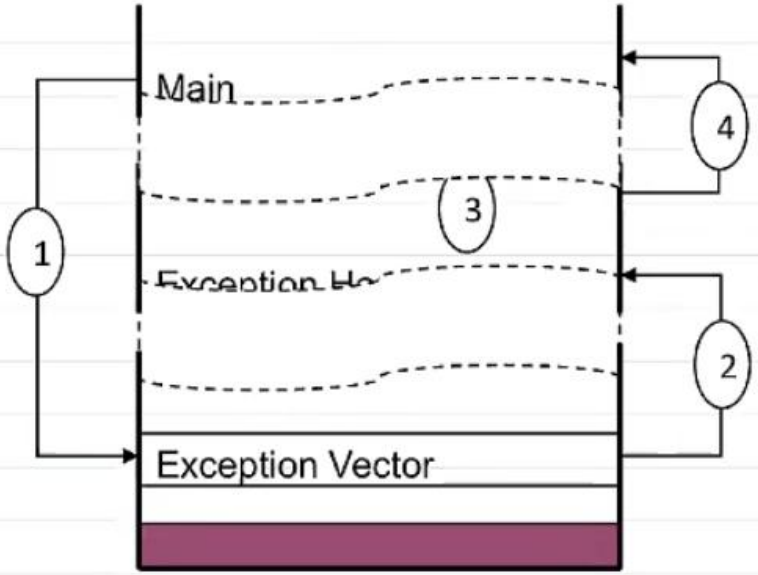
Address		
0x40 + 4*N	External N	16 + N
...	...	...
0x40	External 0	16
0x3C	SysTick	15
0x38	PendSV	14
0x34	Reserved	13
0x30	Debug Monitor	12
0x2C	SVC	11
0x1C to 0x28	Reserved (x4)	7-10
0x18	Usage Fault	6
0x14	Bus Fault	5
0x10	Mem Manage Fault	4
0x0C	Hard Fault	3
0x08	NMI	2
0x04	Reset	1
0x00	Initial Main SP	N/A

Reset Behavior



1. A reset occurs (Reset input was asserted)
2. Load MSP (Main Stack Pointer) register initial value from address 0x00
3. Load reset handler vector address from address 0x04
4. Reset handler executes in Thread Mode
5. Optional: Reset handler branches to the main program

Exception Behavior



1. Exception occurs
   - Current instruction stream stops
   - Processor accesses vector table
2. Vector address for the exception loaded from the vector table
3. Exception handler executes in Handler Mode
4. Exception handler returns to main

Startup Code and Linker Script

- Startup Code and Linker Script are **essential parts of embedded systems.**
- They control **how the program starts** and **how memory is used.**
- Without them, an ARM program **cannot run.**
- **Startup Code**
- **What is Startup Code?**
  - Startup code is the **first code that executes when the microcontroller is powered ON or reset**.
    It prepares the system so that the **C program can start correctly**.
- **Why Startup Code is Needed**
  - The ARM CPU **does not know about main()** , It only knows:
  - Stack pointer
  - Reset address
  - Interrupt addresses
- Startup code acts as a **bridge between hardware reset and the C program**.

- **Main Responsibilities of Startup Code**
  - Initializes the **stack pointer**
  - Sets up the **interrupt vector table**
  - Copies initialized global variables from **Flash to RAM**
  - Clears uninitialized global variables (**.bss section**)
  - Calls the main() function
- **Without startup code, main() will never execute.**

Linker Script

- **What is a Linker Script?**
  - A linker script is a **configuration file** that tells the compiler:
  - Where **program code** should be placed
  - Where **data and variables** should be stored
  - How much memory is reserved for **stack and heap.**

- **Why Linker Script is Needed**
  - ARM microcontrollers have **separate memory regions**:
  - Flash memory (program storage)
  - SRAM (data, stack, heap)
  - The linker script ensures that:
    - Code goes to **Flash**
    - Variables go to **RAM**
  - Stack does not overlap with data.

- **FLASH MEMORY**
  - **Program code (.text)**
  - **Constants (.rodata)**
- **RAM MEMORY**
  - **Global variables (.data, .bss)**
  - **Stack**
  - **Heap**
- **The linker script controls memory layout and prevents memory corruption.**
- **Relationship Between Startup Code and Linker Script**
  - **Linker script defines memory locations**
  - **Startup code uses these locations** to initialize the system
  - Both work together to start program execution correctly
  - Startup code initializes the system and starts main(),
    while the linker script defines how memory is organized in the microcontroller.

Flow of Execution

- **Linker Script**

  ↓

- **Startup Code**

  ↓

- **Vector Table**

  ↓

- **main()**

  ↓

- **Interrupt**

  ↓

- **ISR**

Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com