ARM Microcontroller Internship Programme

Trainer: *Kiran Jaybhave*

Sunbeam – Industrial Training Programme

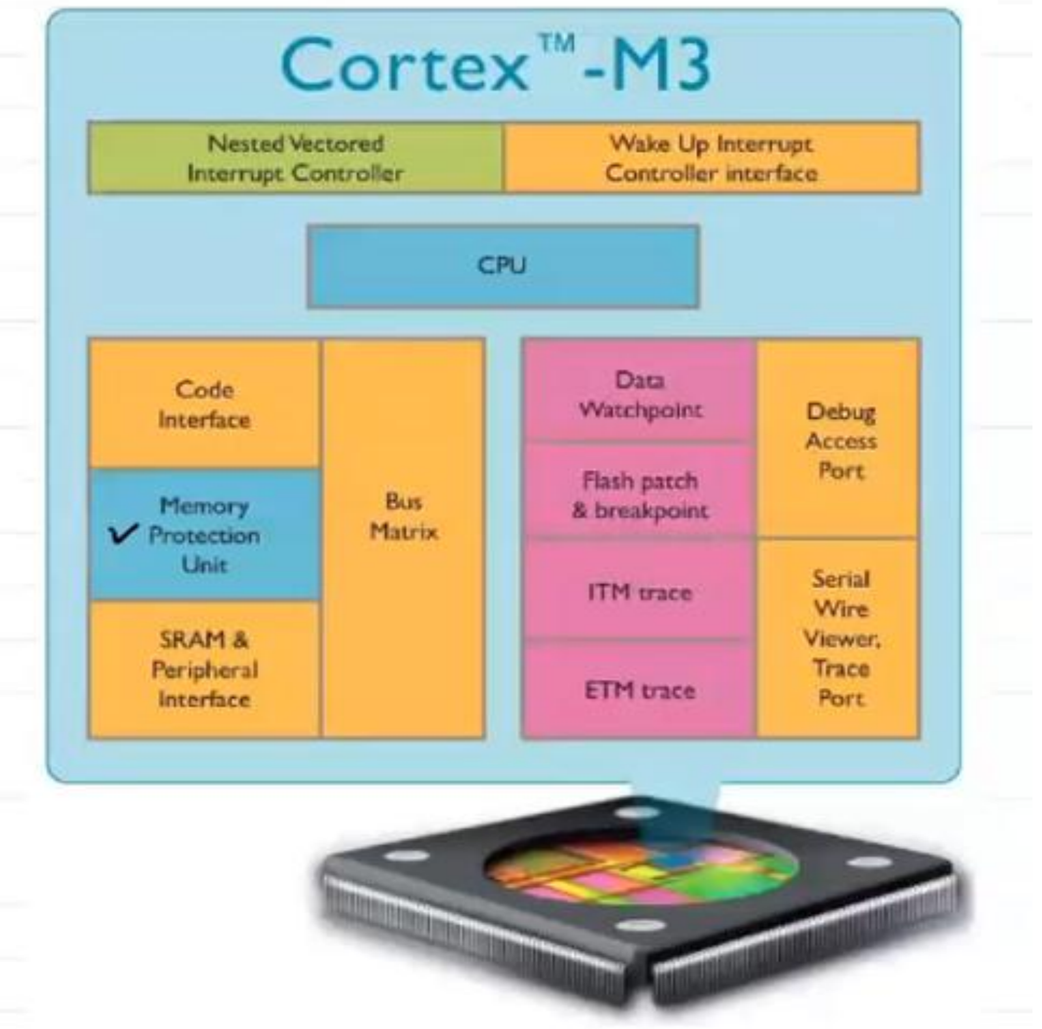
Sunbeam Pune (Hinjawadi)

ARMv7-M Profile Overview (Cortex-M Architecture)

- **Designed for Microcontroller Market**
  - ARMv7-M cores are **specifically designed for microcontrollers**
  - Used in **deeply embedded systems**
  - Supports **bare-metal programming** (can run **without OS**)
  - **Entire application can be programmed in C**
  - Has **fewer features** compared to application processors (Cortex-A)
- **Register and ISA(Instruction Set Architecture) Changes from Other ARM Cores**
  - Supports **Thumb / Thumb-2 instruction set only**
  - Has **only one set of registers**
    - No multiple banked registers like classic ARM
    - **Only Stack Pointer is banked**
      - **MSP (Main Stack Pointer)** → used by Handler mode / main program
      - **PSP (Process Stack Pointer)** → used by application tasks
  - Uses **xPSR** instead of CPSR
    - xPSR has **different bits and structure** compared to CPSR

- **Processor Modes and Exception Model**
    - Cortex-M supports **only two modes**:
        - **Thread Mode** → normal program execution
        - **Handler Mode** → interrupt and exception handling
    - **Vector table contains addresses, not instructions**
    - On exception entry, **CPU hardware automatically saves context** on stack:
        - r0 – r3
        - r12
        - LR
        - PC
        - xPSR
    - Context saving is done by hardware, not software
- **System Control and Memory Layout**
    - Cortex-M cores have a **fixed memory map**
    - **System control is memory-mapped**
    - All control and configuration is done via **memory-mapped control registers**
    - Everything is accessed like memory → easier programming

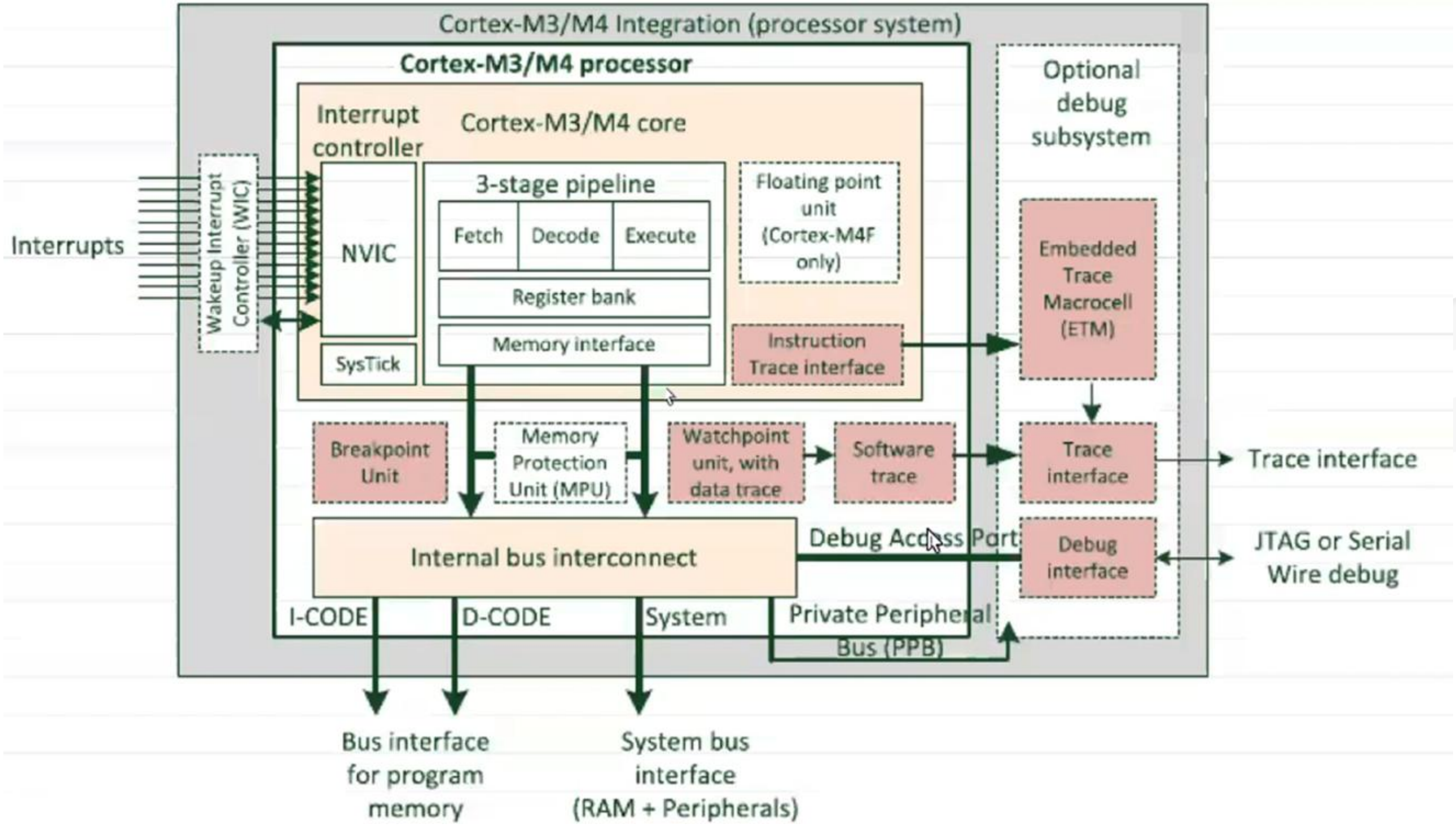
- **CPU Core Features**
  - Uses a **3-stage pipeline**:
    - Fetch -> Decode -> Execute
  - Includes **ALU, multiplier, and barrel shifter**
  - Simple pipeline → **predictable execution timing**
- **Memory Architecture**
  - Uses **von Neumann architecture**
    - **Single address space** for:
      - Flash
      - SRAM
      - I/O registers
  - Code and data share the **same address space**
  - Address regions are **non-overlapping**
- **Bus Interface**
  - Uses **AHB-Lite bus interface**
  - Separate internal buses:
    - **I-Bus** → instruction fetch
    - **D-Bus** → data access
  - Connected through **Bus Matrix** for parallel access
  - Improves performance even with single address space



- **Memory Protection**
  - Supports **Optional MPU (Memory Protection Unit)**
  - MPU provides:
    - Memory region protection
    - Privileged vs unprivileged access

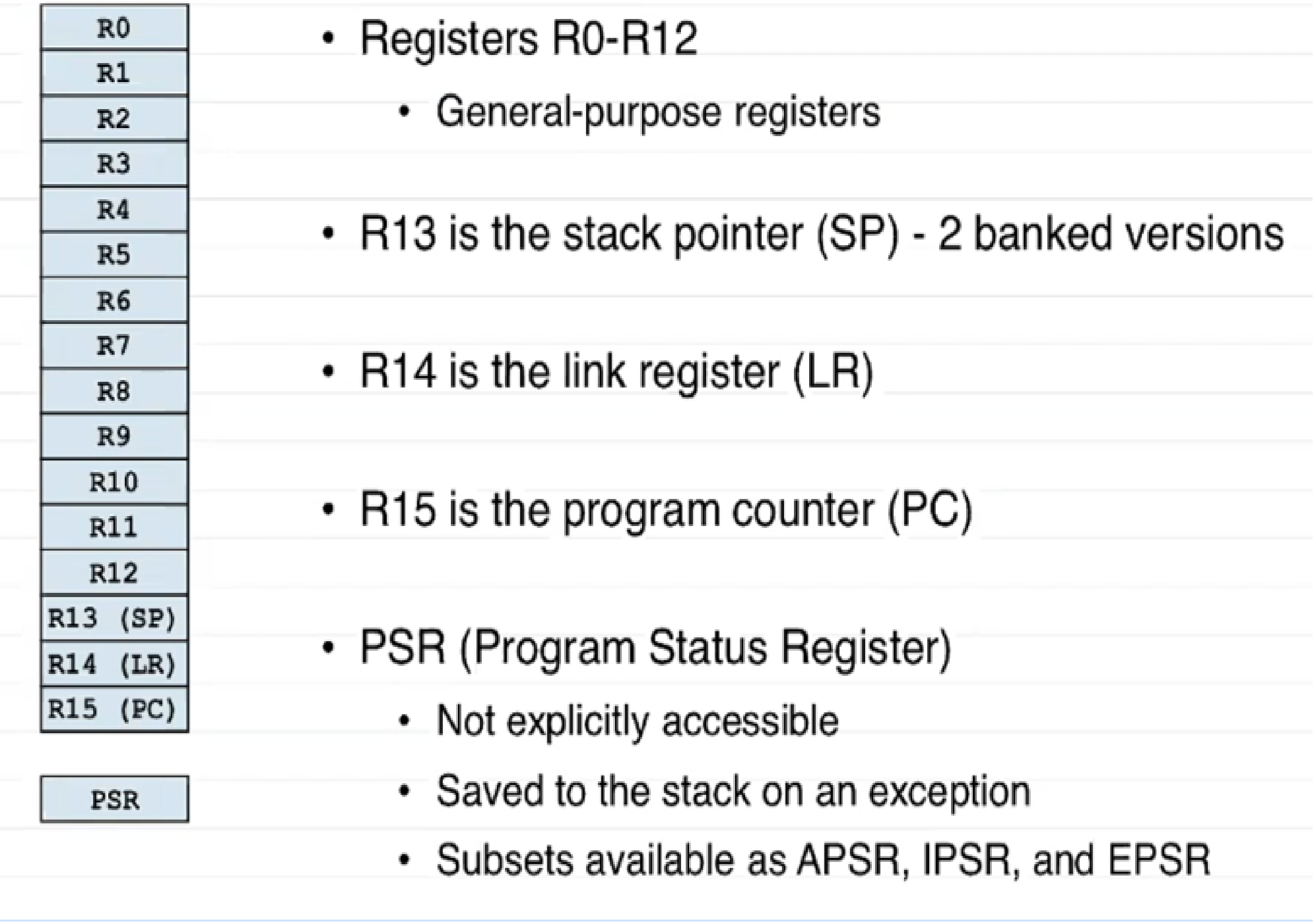
- **Interrupt System (NVIC)**
  - Integrated **Nested Vectored Interrupt Controller (NVIC)**
  - Supports **1 to 240 interrupts**
    (exact number depends on microcontroller vendor)
  - Features:
    - **Configurable priority levels**
    - **Nested interrupts**
    - **Non-Maskable Interrupt (NMI) support**
- **Power Management**
  - Supports **Sleep and low-power modes**
  - Wake-up handled via **Wake-up Interrupt Controller interface**
  - *Optimized for low-power embedded systems*

- **Fixed Memory Map**
- Cortex-M3 uses a **fixed memory map**
- Standard address regions for:
  - Flash
  - SRAM
  - Peripherals
  - System control
- **Debug and Trace Support**
  - Supports **Serial Wire Debug (SWD)** or **JTAG**
  - Includes:
    - Breakpoints
    - Watchpoints
    - Debug access port
  - Optional **ETM (Embedded Trace Macrocell)** for instruction trace
  - Supports **Debug and Sleep control**
  - *Very strong debugging support for developers*


Cortex-M3/M4 Integration (processor system)
Cortex-M3/M4 processor
Interrupt controller
Cortex-M3/M4 core
Interrupts
Wakeup Interrupt Controller (WIC)
NVIC
SysTick
3-stage pipeline
Fetch
Decode
Execute
Register bank
Memory interface
Floating point unit (Cortex-M4F only)
Instruction Trace interface
Optional debug subsystem
Embedded Trace Macrocell (ETM)
Breakpoint Unit
Memory Protection Unit (MPU)
Watchpoint unit, with data trace
Software trace
Trace interface
Trace interface
Internal bus interconnect
Debug Access Port
Debug interface
JTAG or Serial Wire debug
I-CODE
D-CODE
System
Private Peripheral Bus (PPB)
Bus interface for program memory
System bus interface (RAM + Peripherals)

Cortex-M Processor Register Set

- **General-Purpose Registers**
  - **R0 – R12**
  - Used for:
    - Data storage
    - Arithmetic operations
    - Function arguments
    - Temporary variables
  - *Fully general-purpose registers*
- **Stack Pointer – R13 (SP)**
  - R13 is the **Stack Pointer**
  - Has **two banked versions**:
    - **MSP (Main Stack Pointer)**
    - **PSP (Process Stack Pointer)**
  - **Usage:**
  - **MSP**
    - Used after reset
    - Used in **Handler mode**
    - Used in **bare-metal applications**
  - **PSP**
    - Used in **Thread mode**
    - Used by **user programs in OS-based systems**



- Registers R0-R12
  - General-purpose registers
- R13 is the stack pointer (SP) - 2 banked versions
- R14 is the link register (LR)
- R15 is the program counter (PC)
- PSR (Program Status Register)
  - Not explicitly accessible
  - Saved to the stack on an exception
  - Subsets available as APSR, IPSR, and EPSR

Cortex-M Processor Register Set

- **Link Register – R14 (LR)**
  - Stores **return address** during:
    - Function calls (BL instruction)
  - During interrupt/exception:
    - LR stores a **special value called EXC_RETURN**
    - This value tells CPU:
      - Which stack was used (MSP / PSP)
      - Which mode to return to
  - LR behavior changes during exceptions

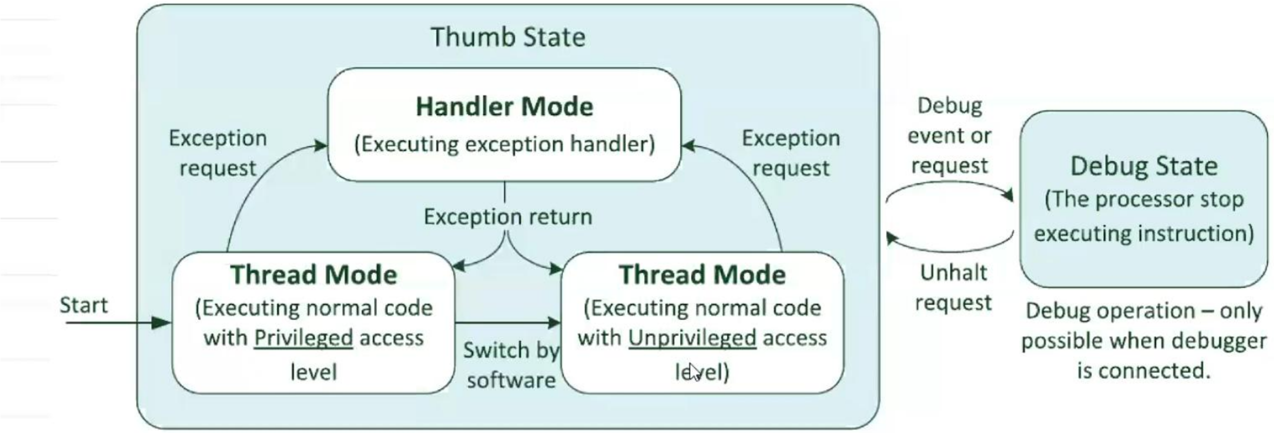
- **Program Counter – R15 (PC)**
  - Holds the **address of the next instruction**
  - Automatically updated during execution
  - Controls program flow (jumps, branches, calls)

- **Program Status Register – xPSR**
  - **Not directly accessible** like general registers
  - Automatically **saved to stack on exception**
  - Contains:
    - Status flags
    - Execution state
    - Interrupt number
  - **xPSR Sub-Registers:**
    - **APSR** – Application Program Status Register (condition flags: N, Z, C, V)
    - **IPSR** – Interrupt Program Status Register (current interrupt number)
    - **EPSR** – Execution Program Status Register (Thumb state, execution info)

Bare-Metal vs Embedded OS Based Development (Cortex-M)

- **Bare-Metal Development**
  - The program is **burned directly into Flash memory**
  - Code **executes without any Operating System**
  - Application runs in **Thread mode**
  - **Only Main Stack Pointer (MSP) is used**
    - Program Stack Pointer (PSP) is **not used**
  - Interrupts and main program both use **MSP**
  - Simple, fast, low overhead
  - Used in small applications and learning stages

- **Embedded OS Based Development**
  - **Program + Embedded OS** are burned into Flash memory
  - Program executes **with OS support** (RTOS)
  - Stack usage:
    - **MSP (Main Stack Pointer)**: Used by **exception handlers and OS kernel**
    - **PSP (Process Stack Pointer)** :Used by **user tasks / application threads**
  - **Enables:**
    - Multitasking ,Task isolation ,Better memory management


Thumb State
Handler Mode
(Executing exception handler)
Exception request
Exception request
Exception return
Thread Mode
(Executing normal code with Privileged access level
Thread Mode
(Executing normal code with Unprivileged access level)
Start
Switch by software
Debug event or request
Unhalt request
Debug State
(The processor stop executing instruction)
Debug operation – only possible when debugger is connected.

Instruction Set State

- **Cortex-M3/M4 supports Thumb / Thumb-2 only**
  - **ARM state is NOT supported**
  - Processor operates in:
    - **Thumb State** → normal execution
    - **Debug State** → used during debugging (step-by-step execution)
  - Hence Cortex-M is called "Thumb-only" architecture

- **Processor Modes**
  - Cortex-M supports **only two modes**:

- **Thread Mode**
  - Used for **normal program execution**
  - Can run: Application code , OS code
  - Can use **MSP or PSP**

- **Handler Mode**
  - Entered when an **exception or interrupt occurs**
  - Used to execute:
    - Interrupt Service Routines (ISR)
    - Exception handlers
  - Always : **Privileged** , Uses **MSP only**

- **Privilege Levels**
  - Cortex-M supports **two privilege levels**:
- **Privileged Mode**
  - Full access to:
    - All CPU instructions
    - System control registers
    - Peripherals
  - Used by:
    - Reset code
    - Exception handlers
    - OS kernel

- **Unprivileged Mode**
  - Restricted access
  - Cannot access:
    - Certain system registers
    - Critical CPU instructions
  - Used by:
    - User applications (in OS-based systems)
  - Bare-metal applications usually run fully privileged

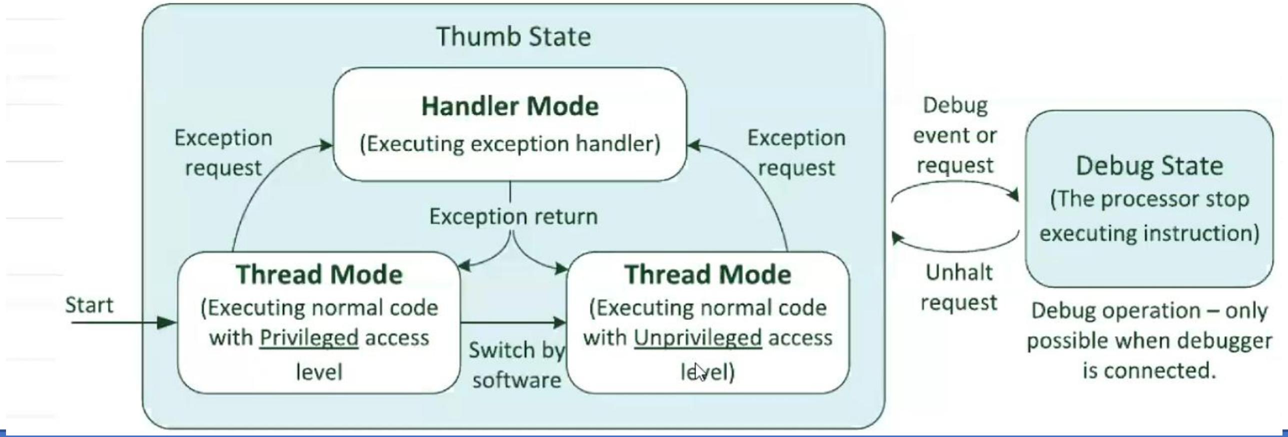
Stack Usage

- **MSP (Main Stack Pointer)**
  - Used:
    - After reset
    - In **Handler mode**
    - In **bare-metal development**
  - Used by:
    - Exception handlers
    - OS kernel
- **PSP (Process Stack Pointer)**
  - Used only in **Thread mode**
  - Used by:
    - User tasks in **RTOS-based systems**
  - **Not used in bare-metal development**

Execution Flow

- **Bare-Metal Flow**
  1. **Reset → Thread mode**
  2. Privileged execution
  3. Uses **MSP**
  4. **Interrupt occurs →**
     1. Switch to **Handler mode**
     2. MSP used
  5. **Return back to Thread** mode

- **OS-Based Flow**
  1. **Reset** → Thread mode (privileged, MSP)
  2. **OS switches to**:
     1. Thread mode
     2. **Unprivileged**
     3. **PSP** for user tasks
  3. **Interrupt occurs →**
     1. Switch to **Handler mode**
     2. Privileged
     3. MSP used
  4. Return to user task (Thread mode, PSP)


Thumb State
Handler Mode
(Executing exception handler)
Exception request
Exception request
Exception return
Start
Thread Mode
(Executing normal code with Privileged access level
Switch by software
Thread Mode
(Executing normal code with Unprivileged access level)
Debug event or request
Unhalt request
Debug State
(The processor stop executing instruction)
Debug operation – only possible when debugger is connected.

Memory Map and Addressing Modes (Cortex-M)

❖Memory Map

- Cortex-M processors use a **fixed memory map**
  - The **address space is divided into predefined regions**
  - These regions are **non-overlapping**
  - Same memory map structure is followed across different Cortex-M based microcontrollers (STM32, NXP, etc.)
- **Major Memory Regions**
  - **Flash Memory**
    - Stores program code
    - Non-volatile memory
  - **SRAM**
    - Stores variables, stack, and heap
    - Volatile memory
  - **Peripheral Region**
    - Used for GPIO, Timers, UART, SPI, I2C, ADC, etc.
    - All peripherals are **memory-mapped**
  - **System Control Region**
    - Contains NVIC, SysTick, and system control registers
  - CPU does not differentiate between memory and peripherals , it accesses everything using addresses.

Memory-Mapped I/O

- Cortex-M uses **memory-mapped I/O**
- Peripheral registers are mapped into the processor address space
- CPU accesses peripherals using **load and store instructions**
- **Example:**
  - **#define GPIO_ODR (*(volatile int*)0x40020C14)**
  - **GPIO_ODR = 1;** **// Turn ON LED**
- **Writing to an address** → controls hardware
- **Reading from an address** → reads hardware status
- **volatile** is used because hardware values can change independently of program flow

- **Memory Architecture View**
  - Cortex-M uses a **single unified address space** (von Neumann view)
  - Internally optimized using multiple buses:
    - **I-CODE bus** → Instruction fetch
    - **D-CODE bus** → Data access
    - **System bus** → SRAM and peripherals
  - Programmer sees a single address space,.

Cortex-M Instruction Set(Thumb / Thumb-2)

1. Data Transfer Instructions

- Used to move data between registers
- Also used to load immediate values
- Does not directly access memory
- Example concept: Move data inside CPU

2. Load–Store Instructions

- Cortex-M follows Load–Store architecture
- Only load and store instructions access memory
- All arithmetic and logic operations work on registers only
- Address update types (conceptual):
- Post increment
- Pre increment
- Pre increment with write-back
- Teach conceptually, not syntax

3. Arithmetic Instructions

- Used for mathematical operations
- Examples:
  - Addition
  - Subtraction
  - Increment / Decrement
  - Multiply
- Used in counters, loops, calculations

4. Logical Instructions

- Used for bit manipulation
- Examples:
  - AND
  - OR
  - XOR
  - NOT
  - Bit clear
- Directly related to register programming

5. Conditional Branching

- Used to control **program flow**
- Supports:
    - Conditional branch (if / else)
    - Loop control
- Cortex-M uses **branch instructions**, not full ARM conditional execution

6. Shift Operations (Barrel Shifter)

- Supports:
- Logical shift
- Arithmetic shift
- Used for:
- Bit masking
- Fast multiply/divide by 2

7. Load–Store Multiple Instructions

- Used to transfer multiple registers at once
- Mainly used for:
- Stack operations
- Context save / restore (interrupts)

8. Function Call and Stack Operations

- Function call
- Function return
- Stack handling
- Uses:
- Stack Pointer (SP)
- Link Register (LR)
- PUSH / POP operations
- Important for functions and interrupts

9. DSP Instructions (Cortex-M4 only)

- Available in Cortex-M4 / M7
- Used for:
- Signal processing
- Audio / filtering
- only if using STM32F4

10. Miscellaneous Instructions

- NOP
- System control instructions
- Exception return instructions

Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com