Lecture 04 ARM Cortex-M4 Processor

Dr. Tushar, Mosaddek Hossain Kamal
Professor

Computer Science and Engineering, University of Dhaka,

BSc Third Year, Semester 2 (July – Dec), Academic Year: 2024

CSE3201: Operating Systems

October 10, 2024

Outline

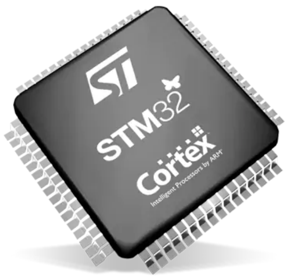
Learning Outcome

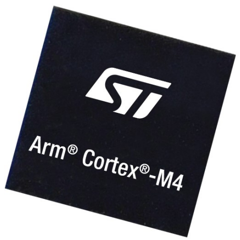
Learning Outcome

- Understand the Cortex-M4 Processor Major Components
  - NVIC, SysTick, FPU, Special Registers, and so on
  - Real-time clock
- Understand the use of Cortex-M4 lower level programming:
  - System call, Task and Memory management
  - thread, Synchronization tools
  - Exception handling
- Understanding ARM Memory map and address space
  - Flash memory, SRAM
  - Peripheral address space
  - Kernel stack frame and process/task stack frame

A look at Cortex-M4 Processor

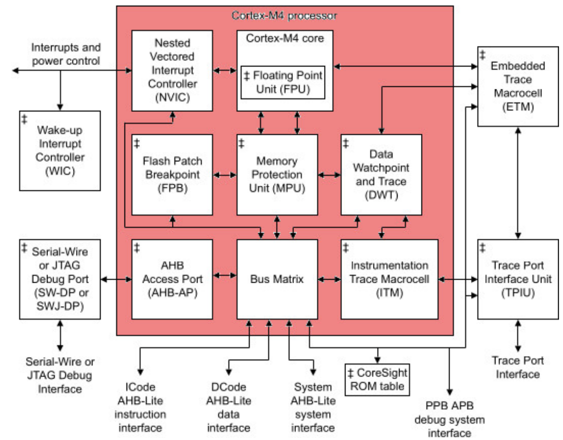
ARM Cortex-M4 Architecture

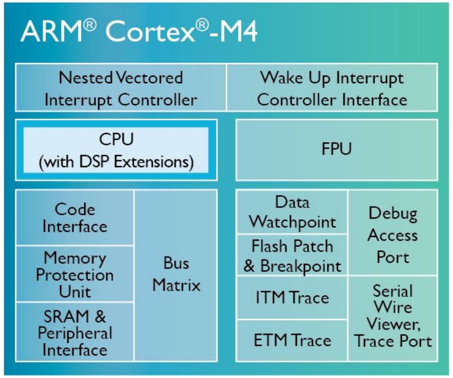




- Cortex-M4 Core
- Nestetd Interrupt Vector (NVIC)
- Memory Protection Unit (MPU)
- Optional Floating Point Unit (FPU)
- Flash Patch Break Point (FPBP)
- Data Watch Point and Trace (DWT)
- AHB Access Port (AHB-AP)
- Bus Matrix
- Instrumentation Trace Microcell (ITM)

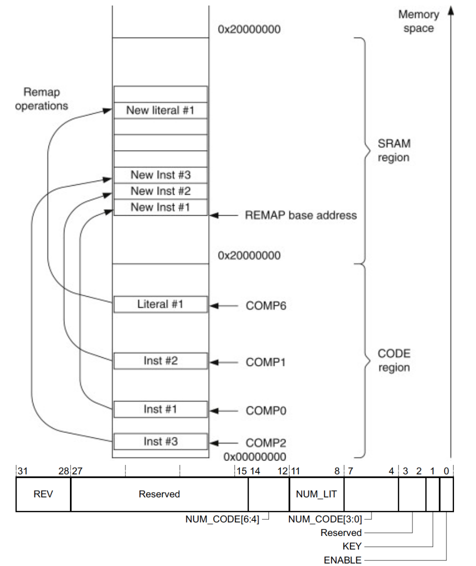
Cortex-M4 Architecture





Cortex-M4 Processor Components – FPBP

- Flash Patch Break Point (FPBP)
    - Hardware Break Point
    - Generates a bp event for debug modes
        - Breakpoint code executed – Halt
        - Debug monitor exception – debug monitor
    - View register's content, memory contents, debug using single stepping
    - To reduce the cost of one-time programmable memory - ROM
        - Small system programmable memory to apply the patch
        - costly to replace whole ROM – a bug in the program
    - Not required when erasing the whole flash and reprogrammable




```
typedef struct {
  volatile uint32_t FP_CTRL;
  volatile uint32_t FP_REMAP;
  // Number Implemented determined by FP_CTRL
  volatile uint32_t FP_COMP[];
} sFpbUnit;
static sFpbUnit *const FPB = (sFpbUnit *)0xE0002000;
```

Cortex-M4 Processor Core

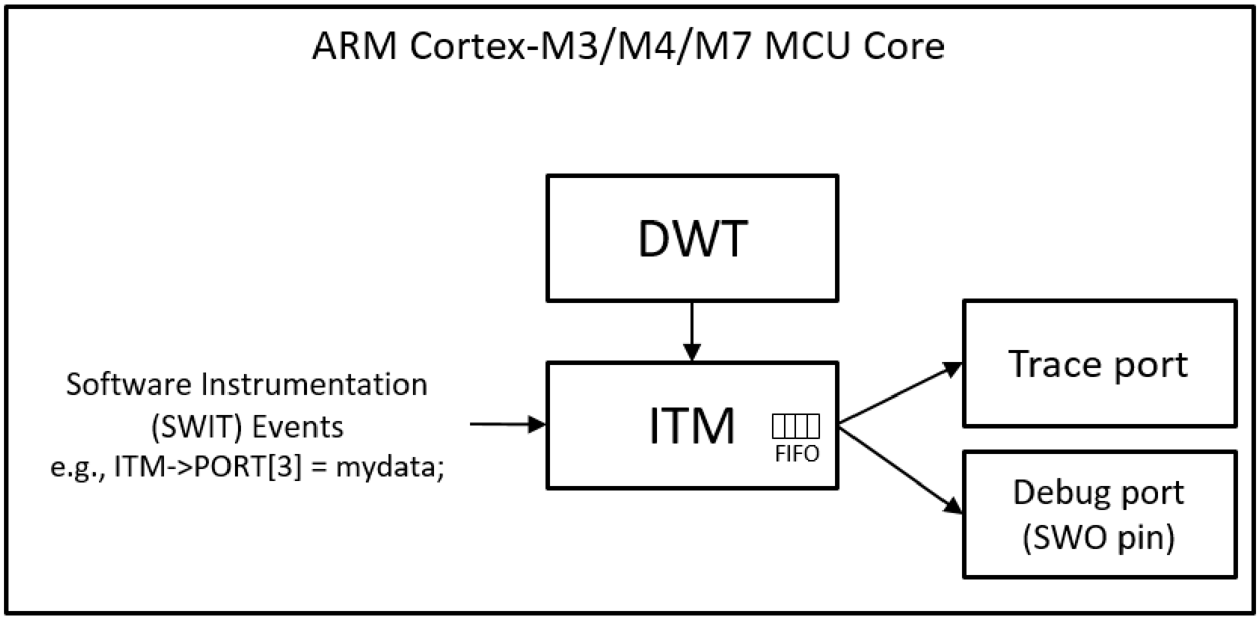
- Data Watch Point and Trace (DWT) unit
  - Counts execution cycle and Cycle Per Instruction

Address	Name	Type	Reset	Description
0xE0001000	DWT_CTRL	RW	See [a]	Control Register
0xE0001004	DWT_CYCCNT	RW	0x00000000	Cycle Count Register
0xE0001008	DWT_CPICNT	RW	-	CPI Count Register

- AHB Access Port (AHB-AP)
  - Optional Debug Access Port (SW-DP or SWJ-DP used to access)
  - Provide Access to all memory and register in the system
  - Master into Bus-Matrix
- Bus Matrix
  - Arbitrates accesses to both the external memory system and to the internal System Control Space (SCS) and debug components, support ARMv7 unaligned accesses, and performs all accesses as single, unaligned accesses

Instrumentation Trace Microcell (ITM)

Instrumentation Trace Microcell (ITM) A hardware unit can transfer diagnostic data of two main types: Debug events generated by the DWT unit, such as exception events and data watchpoint events. Software instrumentation (SWIT) events, i.e., custom data logged by your code. see: Technical Reference Manual



Cortex-M4 Execution Mode

Processor mode and privilege levels for software execution

- Thread mode
  - Used to execute application software
  - Enters after reset
  - Control Register controls: execute in privileged or unprivileged
- Handler mode
  - Used to handle exceptions, execution always in privileged mode
  - returns thread mode after finishing exception procession
- Privileged Level
  - Unprivileged
    - limited access: MSR and MRS instructions
    - no access to CPS (change processor status) instruction, System Timer, NVIC, SCB
    - restricted Access to memory or peripherals
    - **Must use SVC call to transfer control to privileged**
  - Privileged
    - Can access all instructions and resources
    - Can modify CONTROL register to change the privileged level

Cortex-M4 Core Registers

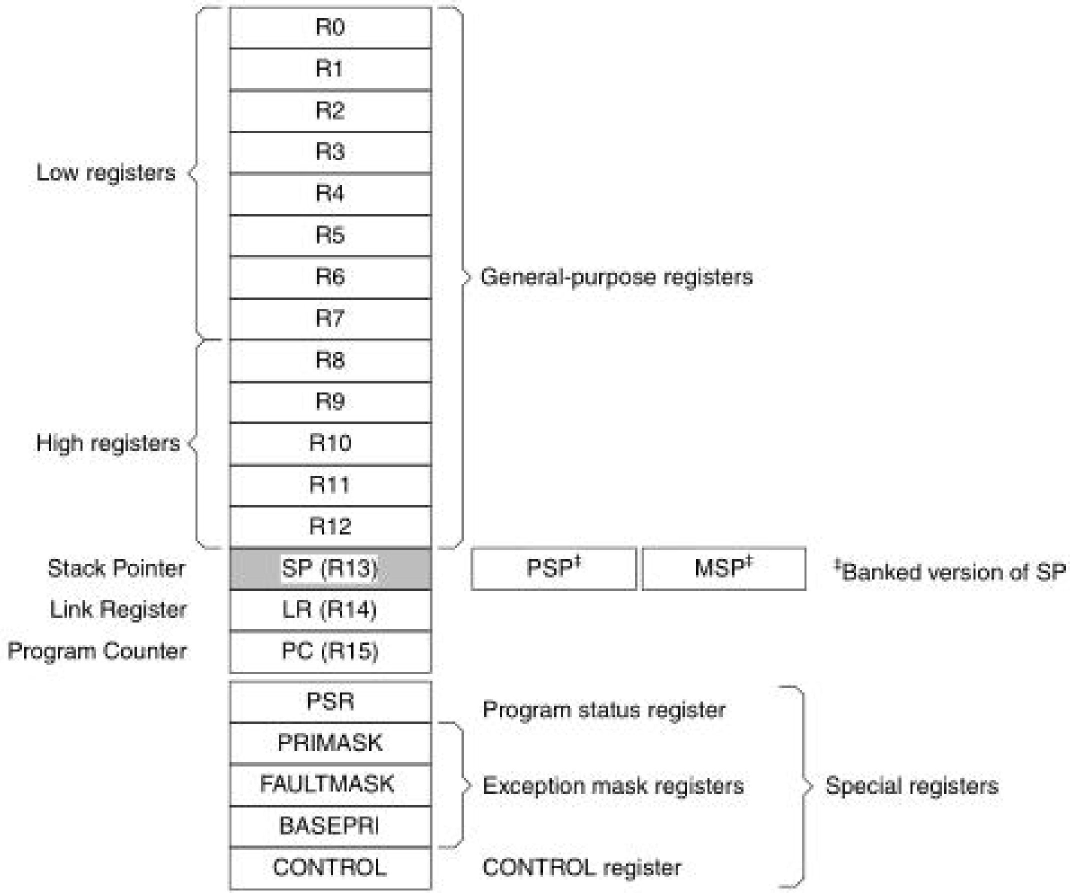


Figure 1: Processor Core Registers

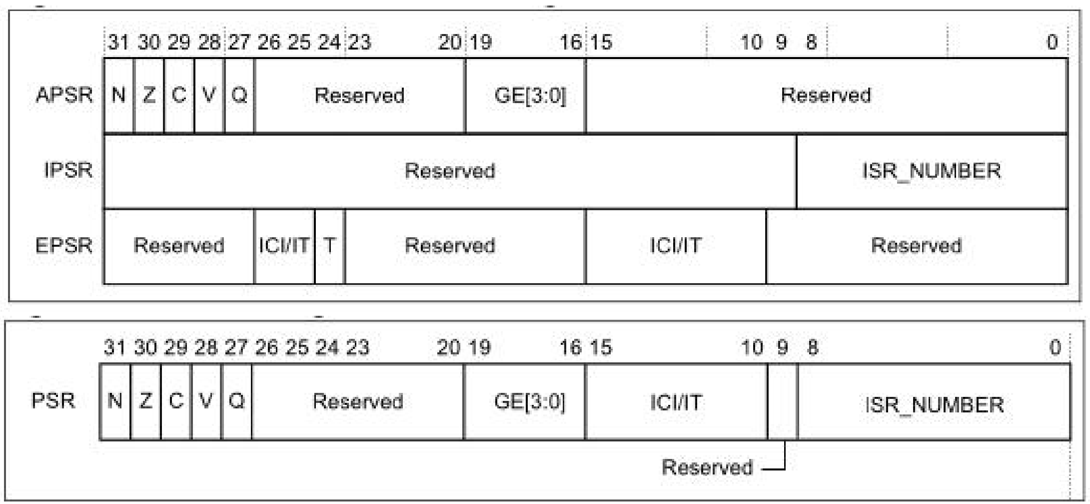
Cortex-M4 Core Registers

Name	Type (1)	Required privilege (2)	Reset value	Description
R0-R12	read-write	Either	Unknown	General-purpose registers on page 18
MSP	read-write	Privileged	See description	Stack pointer on page 18
PSP	read-write	Either	Unknown	Stack pointer on page 18
LR	read-write	Either	0xFFFFFFFF	Link register on page 18
PC	read-write	Either	See description	Program counter on page 18
PSR	read-write	Privileged	0x01000000	Program status register on page 18
ASPR	read-write	Either	Unknown	Application program status register on page 20
IPSR	read-only	Privileged	0x00000000	Interrupt program status register on page 21
EPSR	read-only	Privileged	0x01000000	Execution program status register on page 21
PRIMASK	read-write	Privileged	0x00000000	Priority mask register on page 23
FAULTMASK	read-write	Privileged	0x00000000	Fault mask register on page 23
BASEPRI	read-write	Privileged	0x00000000	Base priority mask register on page 24
CONTROL	read-write	Privileged	0x00000000	CONTROL register on page 24

Figure 2: See STM32F-Programming Manual Pages

Cortex-M4 Core Registers

- R0-R12: General Purpose Register for Data Operation
- R13: SP Stack Pointer, Based on Control Register Value – bit[1]
  - ‘0’: Main Stack Pointer (MSP)
  - ‘1’: Process Stack Pointer (PSP)
  - On reset processor load address ‘0x00000000’ in MSP
- R14: Link Register (LR)
  - Stores subroutine return address; Reset Value: ‘0xFFFFFFFF’
- R15: PC; on reset load (Reset Vector): ‘0x00000004’
- Program Status Register (PSR)
  - Application Program Status Register (APSR)
  - Interrupt Program Status Register (IPSR)
  - Execution Program Status Register (EPSR)
    - Bit[T]



Register	Type	Combination
PSR	read-write(1), (2)	APSR, EPSR, and IPSR
IEPSR	read-only	EPSR and IPSR
IAPSR	read-write(1)	APSR and IPSR
EAPSR	read-write(2)	APSR and EPSR

Figure 3: xPSR

Cortex-M4 Core Registers

- **PRIMASK**
  - Prevent activation of all Exception with configurable priority
  - Bit[0]: ‘0’: not effect; ‘1’ masking all priority configurable exception
- **FAULTMASK**
  - Prevent activation of all Exception except NMI
  - Bit[0]:‘0’ no effect; ‘1’: prevent from activation except NMI
- **BASEPRI**
  - μp does not process exception priority value greater than BASEPRI
  - Bits[7:4]: ‘0x00’ – no effect; NonZero: the lower priority value
- **CONTROL Register – controls**
  - Stack use (MSP, or PSP)
  - Privilege level
  - Bit[2] FPCA: floating point context currently active. ‘0’: No floating-point context active
  - Bit[1]: SPSEL: stack pointer section. ‘0’: MSP, ‘1’: PSP
  - Bit[0]: nPRIV – Thread mode privilege level. ‘0’: privilege mode; ‘1’: unprivileged mode

Contex-M4 Stack

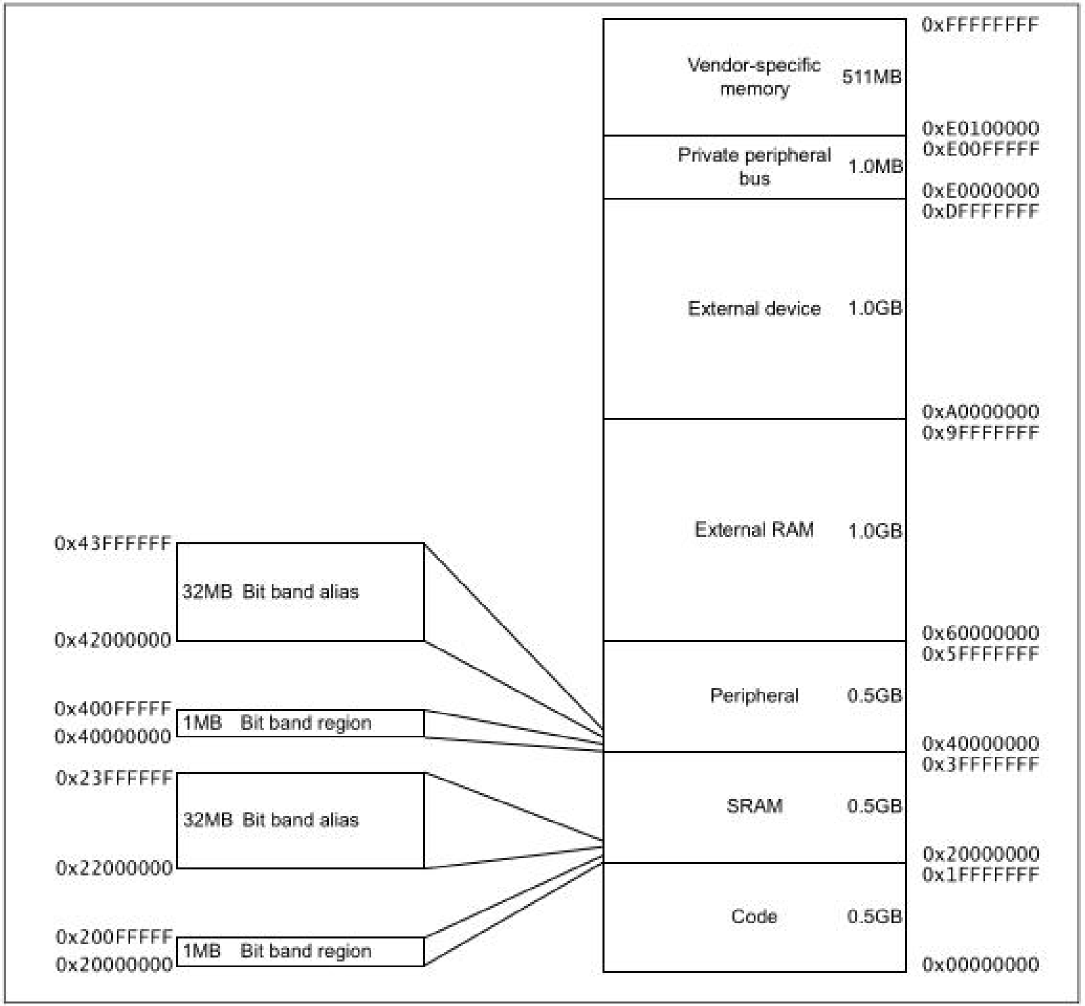
Cortex-M4 Stack

- Full descending stack, insert decrements the address
- Two stack
  - Main Stack
  - Process Stack
  - Control Register bit[1] determines current stack usage

Processor mode	Used to execute	Privilege level for software execution	Stack used
Thread	Applications	Privileged or unprivileged[1]	Main stack or process stack[1]
Handler	Exception handlers	Always privileged	Main stack

Figure 4: Processor mode, execution, and stack

Cortex-M4 Memory Model



- 4GB of fixed memory address
- Memory-Mapped I/O
- Bit-Banding provides automatic operation to bit data

Memory Protection Unit

MPU split memory into regions defines memory type and attributes

Memory Type	**Description**
Normal	The processor can re-order transactions for efficiency, or perform speculative reads.
Device	The processor preserves transaction order relative to other transactions to Device or Strongly-ordered memory.
Strongly-ordered	The processor preserves transaction order relative to all other transactions

Additional memory attributes: **Execute Never (XN)**: Processor prevents instruction to access memory; otherwise causes a memory fault

Odering Memory Access

Memory access ordering: instruction A1 Appears before instruction A2. μP does not guarantee access order. It must ensure using the memory barrier (DMB, DSB, and ISB).

A1	A2			
	Normal access	Device access		Strongly ordered access
		Non-shareable	Shareable	
Normal access	-	-	-	-
Device access, non-shareable	-	<	-	<
Device access, shareable	-	-	<	<
Strongly ordered access	-	<	<	<

Behavior of Memory access

For more, see page 30 Cortex-M4 programming manual

Address range	Memory region	Memory type	XN	Description
0x00000000-0x1FFFFFFF	Code	Normal[1]	-	Executable region for program code. Can also put data here.
0x20000000-0x3FFFFFFF	SRAM	Normal[1]	-	Executable region for data. Can also put code here. This region includes bit band and bit band alias areas, see *Table 14 on page 31*.
0x40000000-0x5FFFFFFF	Peripheral	Device[1]	XN[1]	This region includes bit band and bit band alias areas, see *Table 15 on page 31*.
0x60000000-0x9FFFFFFF	External RAM	Normal[1]	-	Executable region for data.
0xA0000000-0xDFFFFFFF	External device	Device[1]	XN[1]	External Device memory
0xED000000-0xED0FFFFF	Private Peripheral Bus	Strongly-ordered[1]	XN[1]	This region includes the NVIC, System timer, and system control block.
0xED100000-0xFFFFFFFF	Memory mapped peripherals	Device[1]	XN[1]	This region includes all the STM32 standard peripherals.

Cortex-M4 Core Peripheral Units: SysTick timer

Textbook Page: 312

Cortex-M4 Small integrated timer – System Tick

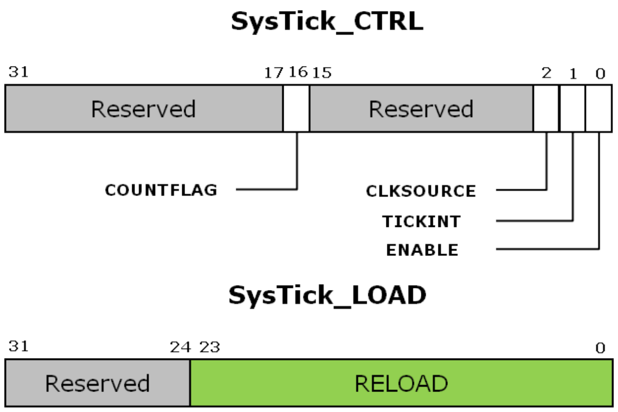
- A part of the NVIC
- Generate SysTick Exception
- We need it for context-switch to execute tasks in the different time schedule
- unprivileged application cannot disable the timer
- Inside the processor – for portability

Address	Name	Type	Required privilege	Reset value
0xE000E010	STK_CTRL	RW	Privileged	0x00000000
0xE000E014	STK_LOAD	RW	Privileged	Unknown
0xE000E018	STK_VAL	RW	Privileged	Unknown
0xE000E01C	STK_CALIB	RO	Privileged	0xC0000000

SysTick Registers

SysTick – System Clock

- 24-bit decrement counter
- Decrement using processor clock or reference clock



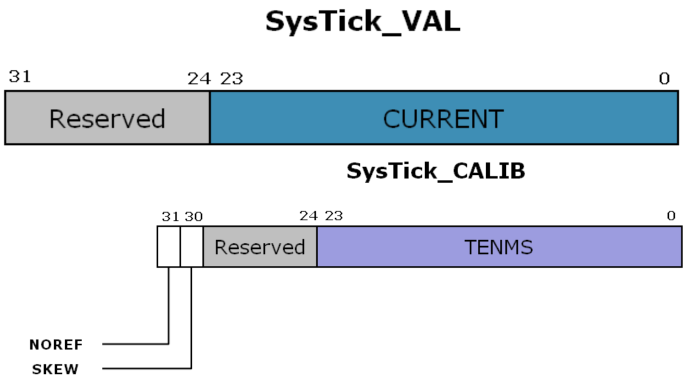
Reload Reg(RW). bit[23:0]
24-bit reload value

SYSTICK Control Reg

- COUNTFLAG – Status(R), '1' if count reaches '0'
- CLKSOURCE – Conf.(RW) '1' if use core clock, '0' for external clock
- TICKINT – '1' enable systick interrupt
- ENABLE – '1' enable SYSTICK Timer.

SYSTICK Current Reg (RWc)

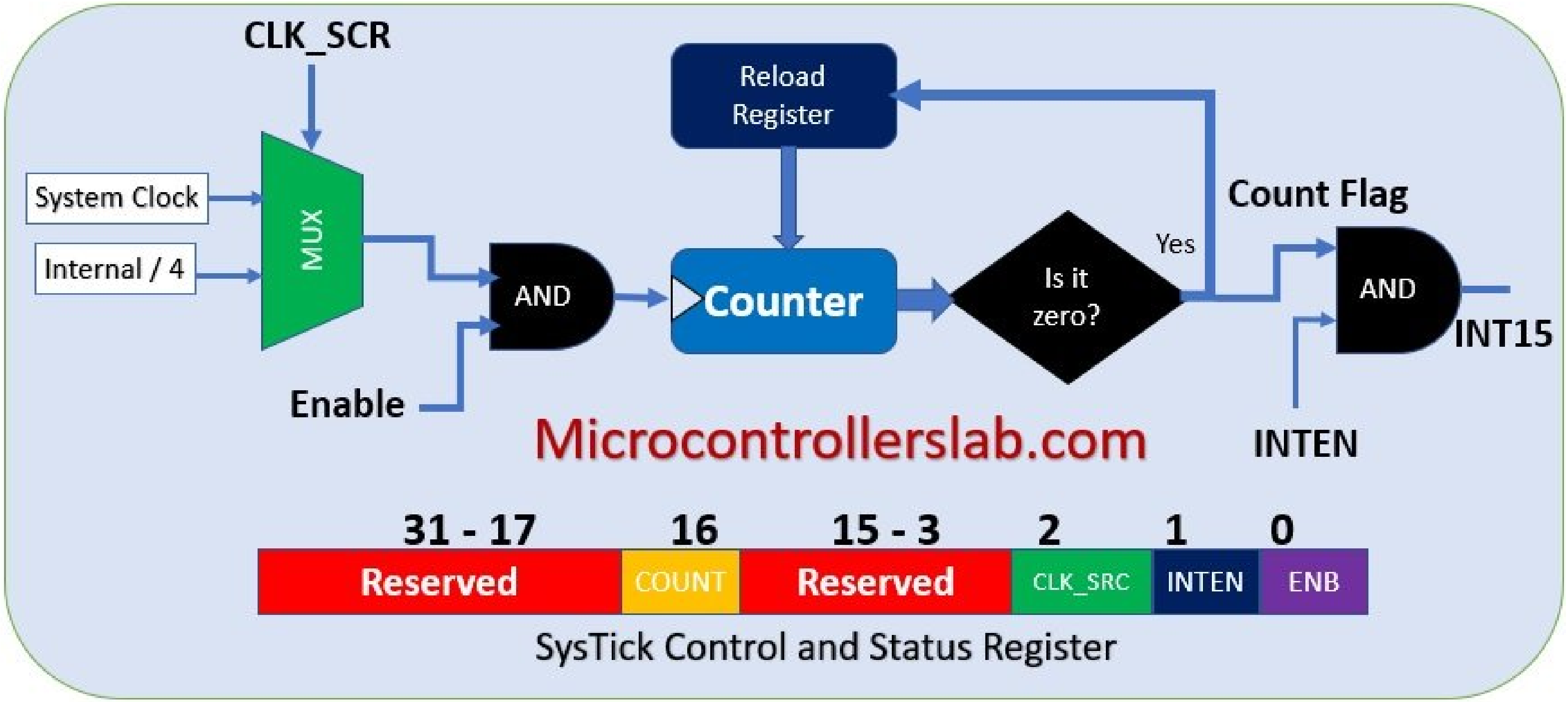
- Read the current value of the timer
- Write '0' to clear



SYSTICK Calibration Reg (R)

- bit[31] – no external reff. clock
- bit[30] – '1' calibration value is not exactly 10ms otherwise accurate
- bit[23:0] – calibration value for 10ms (reload value required)

SysTick – System Clock



SysTick Timer

How to use

- Disable the timer before the configuration; it is optional (disable if configured before)
  - SysTick CTRL register set to '0'
- Set the reload value to SysTick Load Register, must one less the targeted value
- Set current val to '0', VAL register
- Enable SysTick to use processor or external clock
- Finally, Enable SysTick Timer
- Use bit-0 to enable and disable the counter anytime

Tick Counter

- Enable interrupt to send signal/interrupt to process or current process
- A counter or variable update to count the tick or time.

Exception and Interrupt

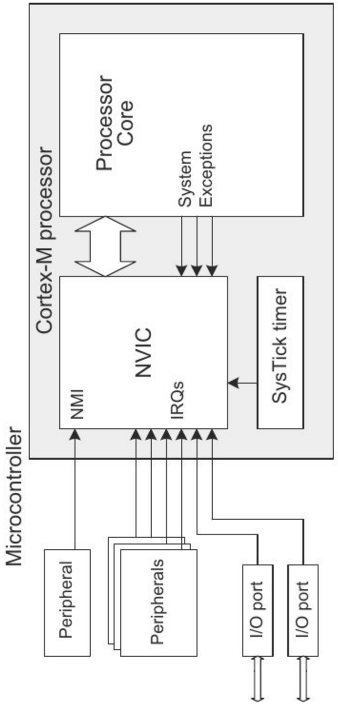
Nested Interrupt Vector Controller (NVIC)

Interrupt Sources are hardware like pins, I/O, and peripherals. There are software Interrupts

Interrupt Processing Sequence (e.g., peripheral)

- The peripheral asserts an interrupt request to the processor
- The processor suspends the currently executing task
- The processor executes an Interrupt Service Routine (ISR) to service
- The processor resumes the previously suspended task

Exceptions events includes interrupt



Exception Types

Cortex-M3 and Cortex-M4 have different numbers of interrupts (1-240) and priority

System Exception [1-15]

- 1-15 Exceptions are system internal those including
  - Reset, NMI. HardFault, MemFault, BusFault SVC, SYSTICK, PenSV
  - Priority for Reset, NMI, and HardFaults are not alterable, and they are -3, -2, and -1

External Interrupt [16+]

- 1-15 Exceptions are system internal those including
  - All other interrupts are external to the processor
  - Priority for these are programmable

Interrupt Management

A set of Registers for Interrupt Management in Cortex-M processor

- Registers are inside NVIC and SCB
- Physically SCB is implemented as a part of NVIC
  - CMSIS-core defines these registers in separate data structure
- Special register in processor core such as PRIMASK, FAULTMASK, and BASEPRI
- NVIC and SCB are located inside SCS (System Control Space) starting from 0xE000E000 (4KB)
- SCS contains SysTick, MPU, Debug registers, and so on.
- privileged mode can access these registers.
- However, Software Trigger Interrupt Register (STIR) can be set up to access from an unprivileged mode
- Reset disable all interrupts with priority-level ‘0’

Interrupt Management – enabling and using interrupt(s)

Before using interrupt, you need to

- Set up the priority of the targeted interrupt
  - void NVIC_SetPriority (IRQn_Type IRQn,uint32_t priority)
- Enable the interrupt generation control in the peripheral that triggers the interrupt
  - Enable IRQ bit of a peripheral such as $USART2->CR1 |= 1 << 7$
- Enable the interrupt in the NVIC
  - void NVIC_EnableIRQ (IRQn_Type IRQn)
- When interrupt triggers corresponding ISR executes
  - USART2_Handler()
- You may need to clear the interrupt in the service routine
- Startup code contains ISR in the vector table
  - (uint32_t) &USART2_Handler
- If not, detail the ISR (weak!! definition) – before using it.
  - void USART2_Handler(void) __attribute__((weak, alias("Default_Handler")));

Interrupt Priority Management

Interrupts are executed according to the priority

- higher priority interrupt executed and preempt lower priority (higher priority number) interrupt
  - Nested interrupt
- Some interrupts has fixed priority (-Negative)– you cannot change such as Reset, NMI, HardFault
- Cortex-M4/M3 Support three fixed highest-level priority and up to 256 level programmable interrupt (128-preemptable)
- Other exception has programmable priority from 0 to 255 with 128 preemptive priority
- Chip designer decides to reduce the complexity of the NVIC

Interrupt Registers ARMv7

Table 7.9 Summary of the Registers in NVIC for Interrupt Control

Address	Register	CMSIS-Core Symbol	Function
0xE000E100 to 0xE000E11C	Interrupt Set Enable Registers	NVIC->ISER [0] to NVIC->ISER [7]	Write 1 to set enable
0xE000E180 to 0xE000E19C	Interrupt Clear Enable Registers	NVIC->ICER [0] to NVIC->ICER [7]	Write 1 to clear enable
0xE000E200 to 0xE000E21C	Interrupt Set Pending Registers	NVIC->ISPR [0] to NVIC->ISPR [7]	Write 1 to set pending status
0xE000E280 to 0xE000E29C	Interrupt Clear Pending Registers	NVIC->ICPR [0] to NVIC->ICPR [7]	Write 1 to clear pending status
0xE000E300 to 0xE000E31C	Interrupt Active Bit Registers	NVIC->IABR [0] to NVIC->IABR [7]	Active status bit. Read only.
0xE000E400 to 0xE000E4EF	Interrupt-Priority Registers	NVIC->IP [0] to NVIC->IR [239]	Interrupt-Priority Level (8-bit wide) for each interrupt
0xE000EF00	Software Trigger Interrupt Register	NVIC->STIR	Write an interrupt number to set its pending status

Interrupt Priority and Setting

- Cortex-M4 has 1-byte (8-bits) for priority of the interrupt
- Stm32F4xx implements 4-MSB of the 8-bits (LSB-3:0 is always '0')
- In reality we do not need more priority (256!)
- Therefore, 16-priority level 0x00, 0x10, 0x20, $\cdots$ 0xF0
- Each 32-bit register presents 4-Interrupt priority; thus PRI0-PRI59, total 60 registers
- Address Range 0xE000E400-0xE000E4EF: total 240 bytes

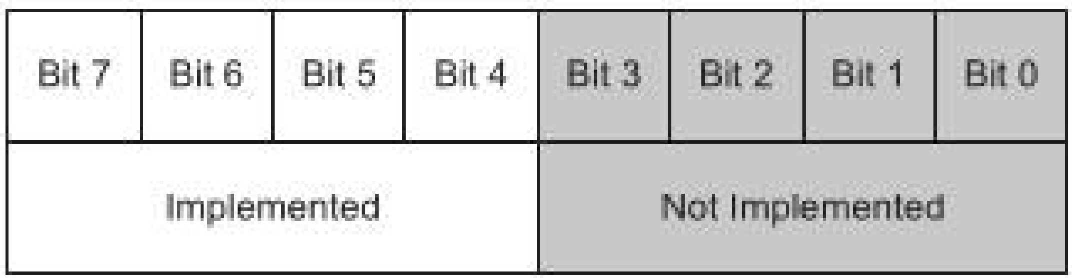


Figure 6: Cortex-M4 Priority Register

Priority Group, Pre-empt and Sub-priority

- The priority bits are divided into two halve
  - Preempt priority and sub-priority
  - Above 4-bit priority: such 2-bit for preempt priority and lower 2-bit sub-priority
- the upper half also known as priority grouping level
- Register AIRCR in SCB is used to determine the number of bits for priority prouping

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VECTKEYSTAT[15:0](read)/ VECTKEY[15:0](write)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ENDIA NESS	Reserved				PRIGROUP			Reserved					SYS RESET REQ	VECT CLR ACTIVE	VECT RESET
r					rw	rw	rw						w	w	w

PRIGROUP [2:0]	Interrupt priority level value, PRI_N[7:4]			Number of	
	Binary point[(1)]	Group priority bits	Subpriority bits	Group priorities	Sub priorities
0b011	0bxxxx	[7:4]	None	16	None
0b100	0bxxx.y	[7:5]	[4]	8	2
0b101	0bxx.yy	[7:6]	[5:4]	4	4
0b110	0bx.yyy	[7]	[6:4]	2	8
0b111	0b.yyyy	None	[7:4]	None	16

Interrupt Enable Register

Interrupt Set Enable Register

- Address: 0xE000E100 to 0xE000E11C – NVIC_ISER[0] – NVIC_ISER[7]
- Example: to enable interrupt 45, calculation for selecting register and bit position
  $ISER = NVIC\_ISER[45/32]$ *or* $NVIC\_ISER[45 >> 5]$ or and $bit\ \ position = NVIC\_ISER[45\%32]$

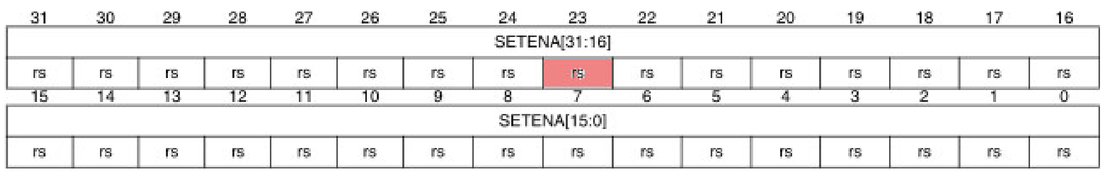


Figure 7: Enable Interrupt ‘7’ writing ‘1’ in the shaded region

CMSIS function void NVIC_EnableIRQ(IRQn_Type IRQn)

Interrupt Clear Enable Register

Interrupt Set Clear Enable Register

- Address: 0xE000E180 to 0xE000E19C – NVIC_ICER[0] – NVIC_ICER[7]
- Example: to clear enable interrupt 45, calculation for selecting register and bit position $ICER = NVIC\_ICER[45/32]$ *or* $NVIC\_ICER[45 >> 5]$ or and $bit\ \ position = NVIC\_ICER[45\%32]$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLRENA[31:16]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLRENA[15:0]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Clear interrupt enable bit of ISER register. CMSIS function void NVIC_DisableIRQ(IRQn_Type IRQn)

Interrupt Pending and Clear Pending

- Another interrupt arrives while executing a higher priority interrupt
- The interrupt is pending, and the corresponding bit of Interrupt Set Pending Register is '1'
- Clear the interrupt pending bit when a waiting interrupt is active.
- completion of an interrupt ISR should clear the active bit

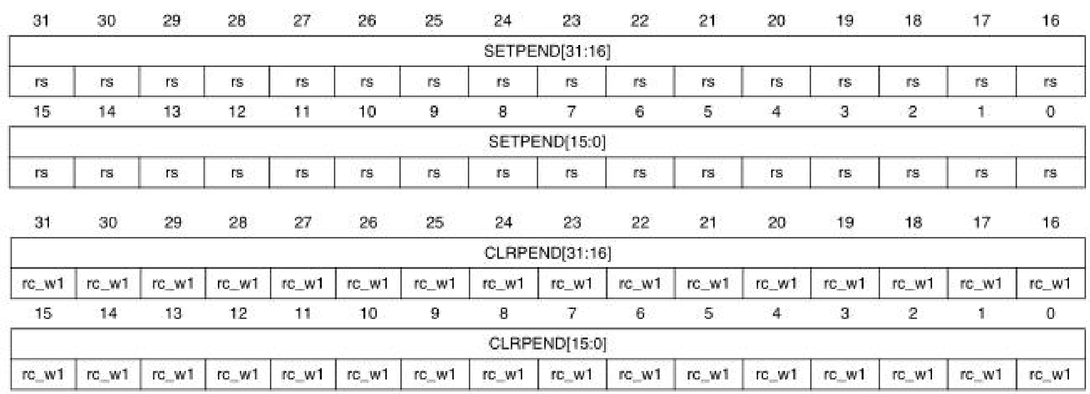


Figure 8: Interrupt Pneding and Clear Register

Interrupt Active Register

- The active interrupt bit is set when the ISR starts executing
- Completion of ISR disable or reset active-bit in the Interrupt Active bit register (IABR)
- IABR is the status register presenting the status of an interrupt

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ACTIVE[31:16]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACTIVE[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Figure 9: Interrupt Active Register

Exception/interrupt State

- Inactive: The exception is not active or inactive. If a higher priority interrupt preempts a lower priority active interrupt
- Pending: When an interrupt waiting to finish a higher priority or the same group interrupts in an active state
- Active: Currently serving by the processor
- Active and Pending: The MCU serves the exception by the processor, and there is a pending exception from the same source

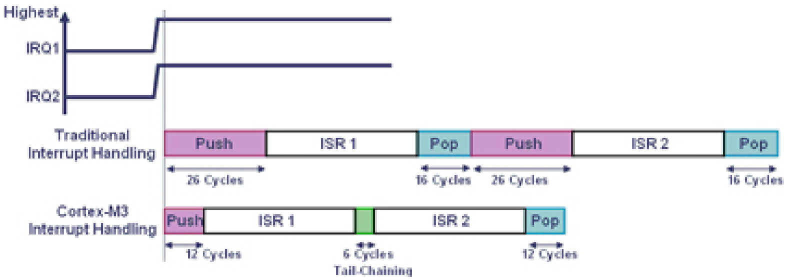


Figure 10: IRQ's arrives one after another

Note: When an exception/interrupt occur microprocessor pushes (stacking) register PC, R0-R3, R12, LR, xPSP and FPU (if FPU active) registers. For sequential interrupts instead of **pop and push** operation it is enough to just change the **LR and PC**

For an IRQ, the registers listed earlier are stacking. However, if the ISR overwrites or uses other registers, the developer should take care.

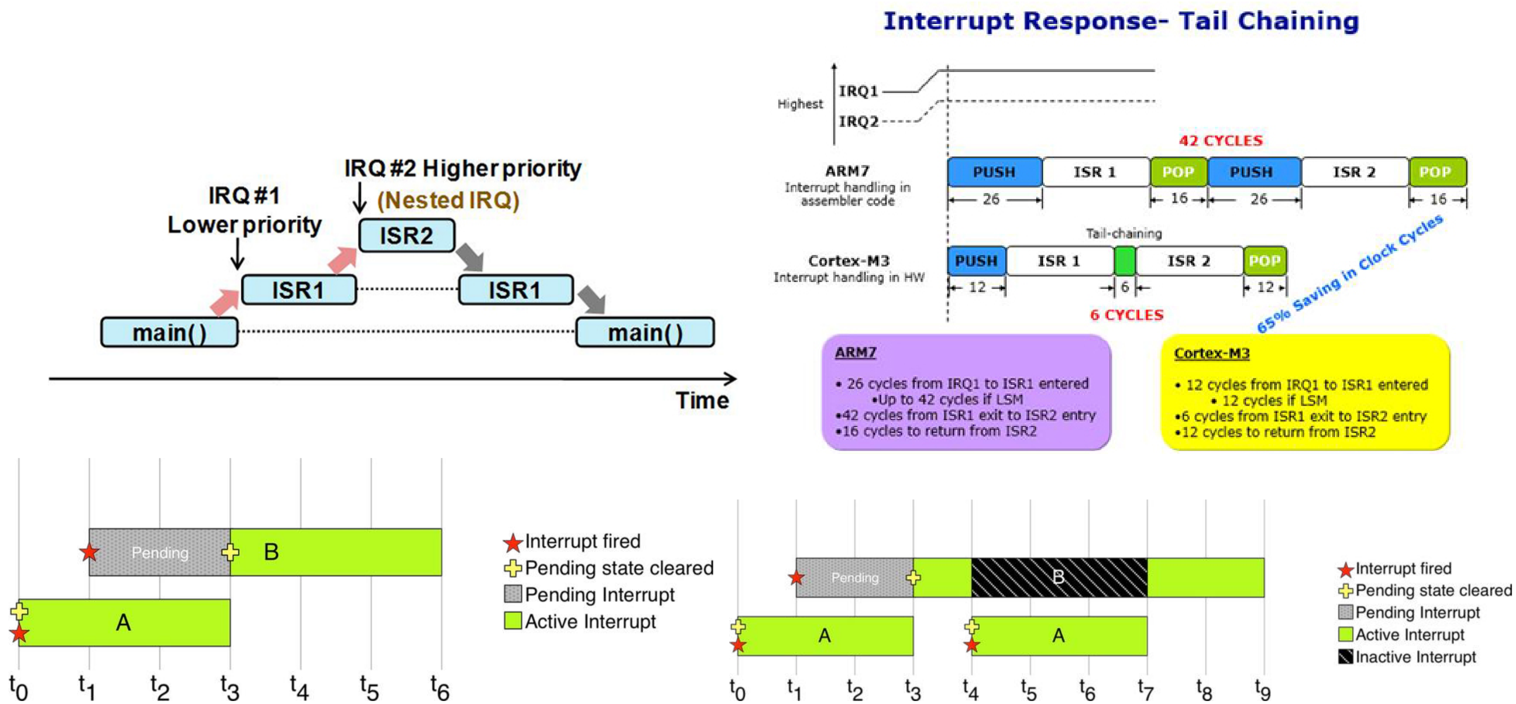


Figure 11: IRQ Life Cycle

Arm Cortex-M Interrupt and Masking

Sometime the system/application need to disable (Mask) some or all interrupt to carryout a critical task

Disable or Masking Interrupts

- SCB contains three special registers for masking interrupt: (i) PRIMASK, (ii) FAULTMASK, and (iii) BASEPRI
  - PRIMASK, a 32 bit register. Bit position '0' is available other bits are reserved.
    - Set PRIMASK disable (put mask on) all interrupts except Hardfault and NMI
      `void __disable_irq(); //Set PRIMASK`
    - Clearing PRIMASK disable masking or take-off mask on all interrupts except Hardfault and NMI
      `void __enable_irq(); // Clear PRIMASK`
  - `void __set_PRIMASK(uint32_t priMask); // Set PRIMASK to value`
    `uint32_t __get_PRIMASK(void); // Read the PRIMASK value`

ARM Cortex-M Interrupt and Masking

Disable or Masking Interrupts

- FAULTMASK – same as PRIMASK however it can mask all interrupts other than NMI
  - Related functions (you will write) are:
    void __enable_fault_irq(void); // Clear FAULTMASK void
    __disable_fault_irq(void); // Set FAULTMASK to disable interrupts
    void __set_FAULTMASK(uint32_t faultMask);
    uint32_t __get_FAULTMASK(void);
- BASEPRI – 32-bit register, however, only 8-bit is currently available
  - The register mask interrupts based on priority,
  - Writing '0' cancel the masking
  - use to disable interrupt lower than a certain interrupt level
  - Writing 0x20 disable interrupts with priority value 0x20 and higher (lower priority)
  - Related Functions: __set_BASEPRI(uint32_t value); // Disable interrupts with priority // $\geq value$
    __set_BASEPRI(0x0); // Turn off BASEPRI masking

Interrupt Program Status Register (IPSR)

- IPSR contains the exception type number of the current Interrupt Service Routine (ISR) Such as (This is the number of the current exception)
  - 0: Thread mode
  - 1: Reserved
  - 2: NMI
  - 3: Hard fault
  - 4: Memory management fault
  - 5: Bus fault
  - 6: Usage fault
  - 7: Reserved
  - $\cdots$
  - 10: Reserved
  - 11: SVCall
  - 12: Reserved for Debug
  - and so on $\cdots$