CSE 3103: Microprocessor and Microcontroller

Professor Upama Kabir

Computer Science and Engineering, University of Dhaka,

Lecture: ARM Instruction Encoding

March 27, 2024

Table of Contents

Interrupt signal: from device to CPU

Interrupt signal: from device to CPU

In each peripheral device:

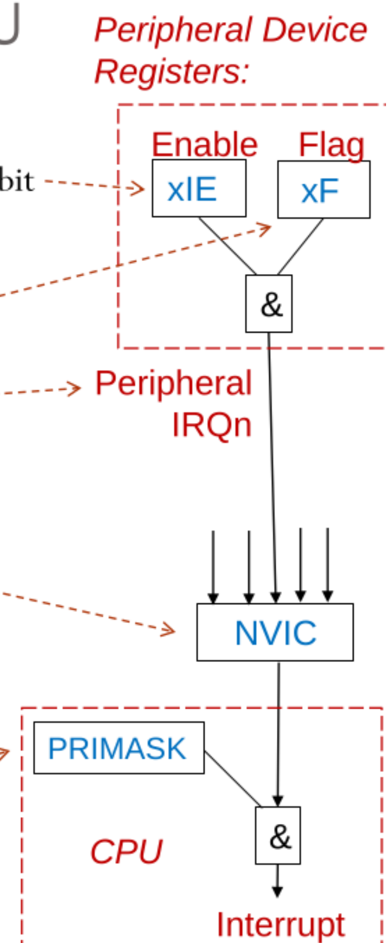
- Each potential interrupt source has a separate **arm** **(enable)** bit
  - Set for devices from which interrupts, are to be accepted
  - Clear to prevent the peripheral from interrupting the CPU
- Each potential interrupt source has a separate **flag** bit
  - hardware sets the flag when an “event” occurs
  - Interrupt request = (flag & enable)
  - ISR software must clear the flag to acknowledge the request
  - test flags in software if interrupts not desired

Nested Vectored Interrupt Controller (NVIC)

- Receives all interrupt requests
- Each has an enable bit and a priority within the VIC
- Highest priority enabled interrupt sent to the CPU

Within the CPU:

- Global interrupt enable bit in PRIMASK register
- Interrupt if priority of IRQ < that of current thread
- Access interrupt vector table with IRQ#



Interrupt Inputs and Pending Behaviors

- Various status attributes applicable to each interrupt:
  - each interrupt can either be disabled (default) or enabled
  - each interrupt can either be pending (a request is waiting to be served) or not pending
  - each interrupt can either be in an active (being served) or inactive state
- Pending status means: put into a state of waiting for the processor to serve the interrupt
- In some cases, processor serves the interrupt as soon as an interrupt becomes pending or the pended request will remain until processor is finished serving a higher priority interrupt

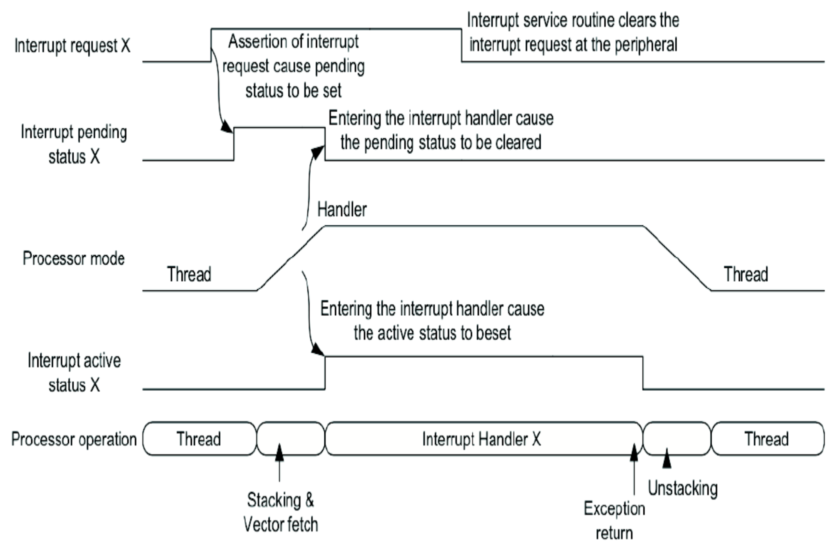


Figure 2: A simple case of interrupt pending and activation behavior

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 Priorities

- Interrupt priority levels allow us to define which interrupts can preempt others
- Cortex-M processors support three fixed highest-priority levels and up to 256 level of programmable priority.
- However, the actual number of available levels is chip dependent since implementing all 256 levels can be costly in terms of power and speed.
- Three negative priorities (hard fault, NMI, and reset) can pre-empt any other exceptions

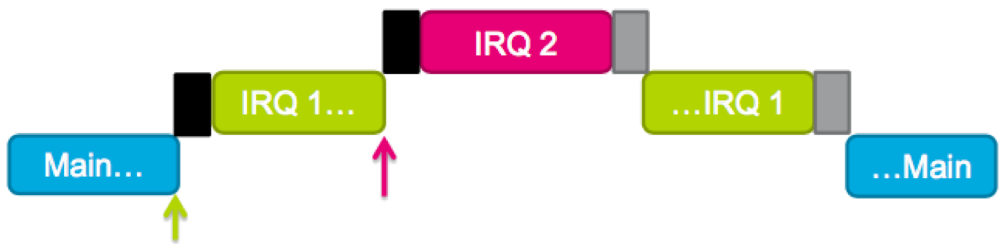


Figure 3

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 Support three fixed highest-level priority and up to 256 level programmable interrupt (128-preemptable)
- Chip designer decides to reduce the complexity of the NVIC

Interrupt Priority Management

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, $\cdots$ 0xE0
- Each 32-bit register presents 4-Interrupt priority; thus PRI0-PRI59, total 60 registers
- Address Range 0xE000E400-0xE000E4EF: total 240 bytes

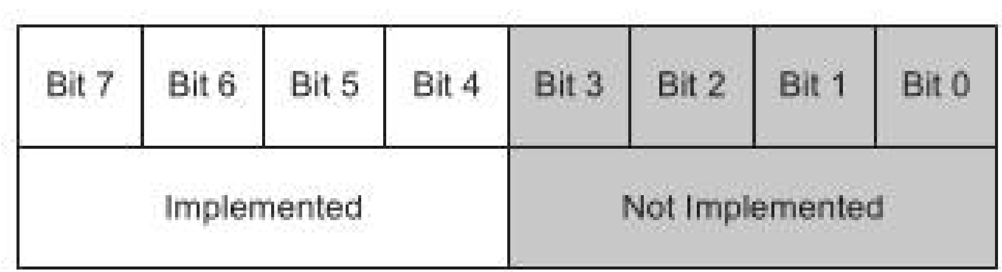


Figure 4: Cortex-M4 Priority Register

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**
ENDIANESS	Reserved				PRIGROUP			Reserved					SYSRESETREQ	VECTCLRACTIVE	VECTRESET
r					rw	rw	rw						w	w	w

Figure 5

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

Vector Table(VT) Relocation

- For any exception,ISR address is stored in VT
- By default, the vector table starts at memory address 0x00000000
- VT is normally defined in the startup codes provided by the microcontroller vendors.
- VT used in startup also contains the initial value of the MSP. It is needed because some exception such as NMI could happen as the processor just came out from reset and before any other initialization steps are executed.
- Usually, the starting address (0x00000000) should be boot memory, and it will usually be either flash memory or ROM devices, and the value cannot be changed at run-time.
- For some applications it is useful to be able to modify or define exception vectors at run-time.
- Cortex-M4 processors support a feature called Vector Table Relocation.
- The Vector Table Relocation feature provides a programmable register called the Vector Table Offset Register (VTOR).
- VTOR as "SCB$-$ $>$VTOR."
- When using VTOR, the base address of the new vector table must be aligned to the size of the vector table extended to the next larger power of 2.

Vector Table(VT) Relocation

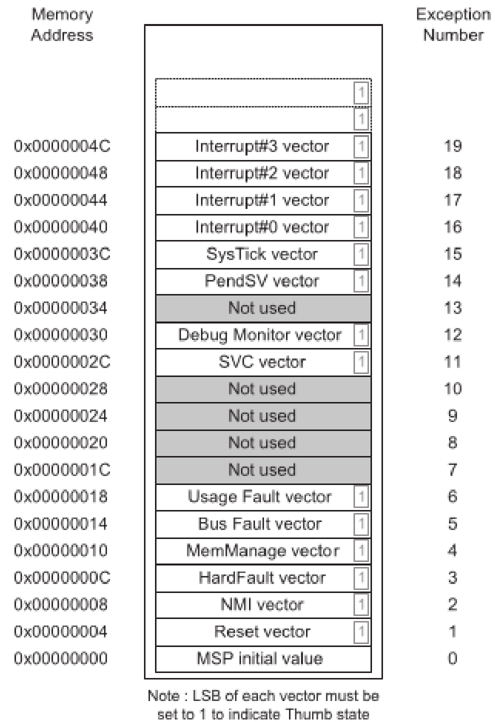


Figure 7

Vector Table(VT) Relocation

- **Example:** 32 interrupt sources in the microcontroller
  The vector table size is (32 (for interrupts) +16 (for system exception space)) x 4 (bytes for each vector) = 192 (0xC0). Extending it to the next power of two makes it 256 bytes. So the vector table base address can be programmed as 0x00000000, 0x00000100, 0x00000200, and so on.
- **Example**: Devices with boot loader
  In some microcontrollers there are multiple program memories: boot ROM and user flash memory. The boot loaders are often pre-programmed in the boot ROM by the microcontroller manufacturer. When the microcontrollers start, they first execute the boot loader code in the boot ROM, and before branching to the user application in the user flash, the VTOR is programmed to point to the starting point of the user flash memory so that the vector table in user flash will be used.

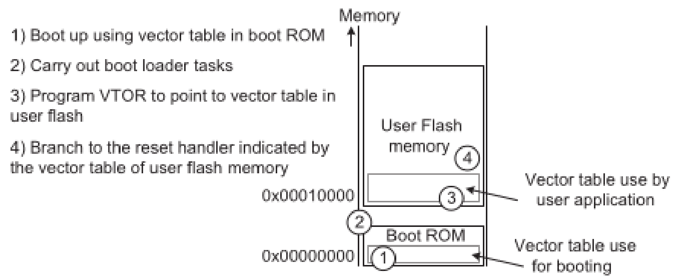


Figure 8

Nested Vector Interrupt Controller

- On Cortex-M4 the NVIC supports up to 240 IRQs, a Non-Maskable Interrupt, a SysTick timer interrupt, and a number of system exceptions.
- When handling and IRQ, some of the registers are stored on the stack automatically and are automatically restored. This allows exception handlers to be written as normal C functions.
- "Nested" refers to the fact that we have different priorities and therefore can handle an interrupt with a higher priority in the middle of handling an interrupt of a lower priority.
- "Vector" refers to the fact that the interrupt service handlers

Cortex-M4's Memory Map

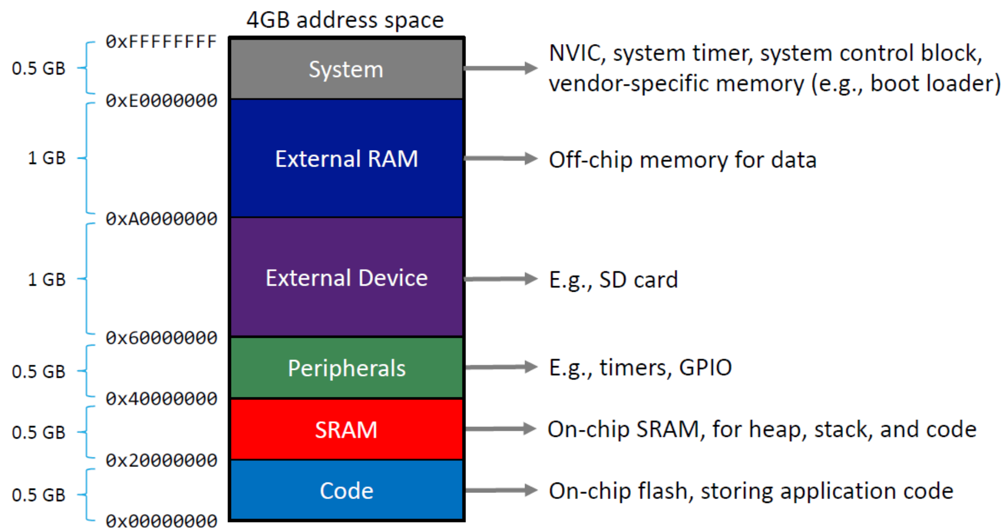
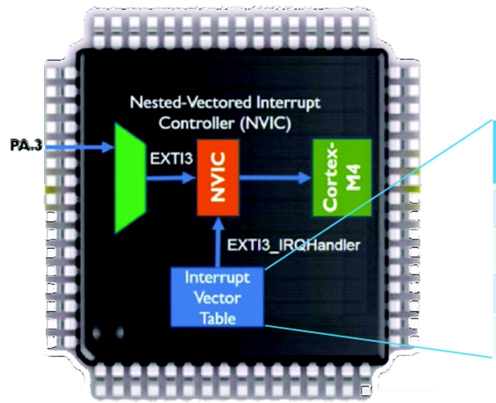


Figure 9

NVIC Operation

- When an interrupt signal is detected, NVIC looks up the ISR address from the interrupt vector table
- The address is then passed to the CPU



Interrupt Vector Table

Interrupt Position (8 bits)	Memory Address of ISR (32 bits)
1	Address of ISR for interrupt 1
2	Address of ISR for interrupt 2
3	Address of ISR for interrupt 3
:	:

Figure 10

Core Registers for NVIC

6.3 NVIC programmers model

Table 6-1 shows the NVIC registers.

Table 6-1 NVIC registers

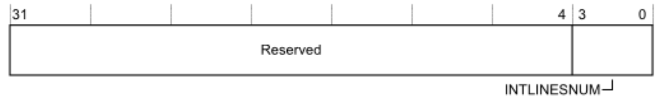
Address	Name	Type	Reset	Description
0xE000E004	ICTR	RO	-	*Interrupt Controller Type Register, ICTR*
0xE000E100 - 0xE000E11C	NVIC_ISER0 - NVIC_ISER7	RW	0x00000000	Interrupt Set-Enable Registers
0xE000E180 - 0E000xE19C	NVIC_ICER0 - NVIC_ICER7	RW	0x00000000	Interrupt Clear-Enable Registers
0xE000E200 - 0xE000E21C	NVIC_ISPR0 - NVIC_ISPR7	RW	0x00000000	Interrupt Set-Pending Registers
0xE000E280 - 0xE000E29C	NVIC_ICPR0 - NVIC_ICPR7	RW	0x00000000	Interrupt Clear-Pending Registers
0xE000E300 - 0xE000E31C	NVIC_IABR0 - NVIC_IABR7	RO	0x00000000	Interrupt Active Bit Register
0xE000E400 - 0xE000E41F	NVIC_IPR0 - NVIC_IPR59	RW	0x00000000	Interrupt Priority Register

The following sections describe the NVIC registers whose implementation is specific to this processor. Other registers are described in the *ARMv7M Architecture Reference Manual*.

Figure 11

Interrupt Controller Type Register: ICTR

Interrupt Controller Type Register in address 0xE000E004. ICTR (read-only) register gives the number of interrupt inputs supported by the NVIC in granularities of 32



Bits[31:4] Reserved.

Figure 12: ICTR

Table 6-2 ICTR bit assignments

Bits	Name	Function	Notes
[31:4]	-	Reserved.	
[3:0]	INTLINESNUM	Total number of interrupt lines in groups of 32: `0b0000` = 0...32 `0b0001` = 33...64 `0b0010` = 65...96 `0b0011` = 97...128 `0b0100` = 129...160 `0b0101` = 161...192 `0b0110` = 193...224 `0b0111` = 225...256	The processor supports a maximum of 240 external interrupts.

Interrupt Set/Clear-Enable Registers: ISER, ICER

- The Interrupt Enable register is programmed via two addresses
  - To set the enable bit: $NVIC-> ISER[n]$
  - To clear the enable bit: $NVIC-> ICER[n]$
  - The $NVIC-> ISER[n]/NVIC-> ICER[n]$ registers are 32 bits wide; each bit represents one interrupt input.
  - Since there could be more than 32 external interrupts, there exists more than one of the above register

Table 7.10 Interrupt Set Enable Registers and Interrupt Clear Enable Registers (0xE000E100-0xE000E11C, 0xE000E180-0xE000E19C)

Address	Name	Type	Reset Value	Description
0xE000E100	NVIC->ISER [0]	R/W	0	Enable for external interrupt #0–31 bit [0] for interrupt #0 (exception #16) bit [1] for interrupt #1 (exception #17) ... bit [31] for interrupt #31 (exception #47) Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
0xE000E104	NVIC->ISER [1]	R/W	0	Enable for external interrupt #32–63 Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
0xE000E108	NVIC->ISER [2]	R/W	0	Enable for external interrupt #64–95 Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
...	...	...	...	...
0xE000E180	NVIC->ICER [0]	R/W	0	Clear enable for external interrupt #0–31 bit [0] for interrupt #0 bit [1] for interrupt #1 ... bit [31] for interrupt #31 Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current enable status
0xE000E184	NVIC->ICER [1]	R/W	0	Clear Enable for external interrupt #32–63 Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current enable status
0xE000E188	NVIC->ICER [2]	R/W	0	Clear enable for external interrupt #64–95 Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current enable status
...	...	...	...	...

Figure 14

Interrupt Set Pending and Clear Pending Register: ISPR, ICPR

- If an interrupt takes place but cannot be executed immediately (for instance, if another higher-priority interrupt handler is running), it will be pended.
- The interrupt-pending status can be accessed through the Interrupt Set Pending $NVIC->ISPR[n]$ and Interrupt Clear Pending $NVIC->ICPR[n]$ registers.
- The pending status controls might contain more than one register if there are more than 32 external interrupt inputs.

Table F.3 Interrupt Set Pending Registers (0xE000E200-0xE000E21C)

Address	Name	Type	Reset Value	Description
0xE000E200	NVIC->ISPR[0]	R/W	0	Pending for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status

Table F.3 Interrupt Set Pending Registers (0xE000E200-0xE000E21C)—Cont'd

Address	Name	Type	Reset Value	Description
0xE000E204	NVIC->ISPR[1]	R/W	0	Pending for external interrupt #32–63 Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
0xE000E208	NVIC->ISPR[2]	R/W	0	Pending for external interrupt #64–95 Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
...	...	...	...	...

F.1.4 Interrupt clear pending registers

Table F.4 Interrupt Clear Pending Registers (0xE000E280-0xE000E29C)

Address	Name	Type	Reset Value	Description
0xE000E280	NVIC->ICPR[0]	R/W	0	Clear pending for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) ... bit[31] for interrupt #31 (exception #47) Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current pending status
0xE000E284	NVIC->ICPR[1]	R/W	0	Clear pending for external interrupt #32–63 Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current pending status

Figure 16

Interrupt Active Status Registers: IABR

- Each external interrupt has an active status bit.
- When the processor starts the interrupt handler, that bit is set to 1 and cleared when the interrupt return is executed.
- During an Interrupt Service Routine (ISR) execution, a higher-priority interrupt might occur and cause pre-emption. During this period, although the processor is executing another interrupt handler, the previous interrupt is still defined as active. Although the IPSR indicates the currently executing exception services, it cannot ensure whether an exception is active when there is a nested exception.

Table F.5 Interrupt Active Status Registers (0xE000E300-0xE000E31C)

Address	Name	Type	Reset Value	Description
0xE000E300	NVIC-> IABR[0]	R	0	Active status for external interrupt #0–31 bit[0] for interrupt #0 bit[1] for interrupt #1 ... bit[31] for interrupt #31
0xE000E304	NVIC-> IABR[1]	R	0	Active status for external interrupt #32–63
...	–	–	–	–

Figure 17

Interrupt Priority Register: IP

- Each interrupt has an associated priority-level register, which has a maximum width of 8 bits and a minimum width of 3 bits.
- Each register can be further divided into group priority level and sub-priority level based on priority group settings.
- The number of priority-level registers depends on how many external interrupts the chip contains

Table F.6 Interrupt Priority Level Registers (0xE000E400-0xE000E4EF)

Address	Name	Type	Reset Value	Description
0xE000E400	NVIC->IP[0]	R/W	0 (8-bit)	Priority level external interrupt #0
0xE000E401	NVIC->IP[1]	R/W	0 (8-bit)	Priority level external interrupt #1
...	–	–	–	–
0xE000E41F	NVIC->IP[31]	R/W	0 (8-bit)	Priority level external interrupt #31
...	–	–	–	–

Figure 18

Software Trigger Interrupt Register: STIR

- A software Trigger Interrupt Register $NVIC- > STIR$ register can be used to trigger an interrupt using software.
- For example, an interrupt #3 can be generated by writing the following code in C:
  $NVIC- > STIR = 3;$

Table F.7 Software Trigger Interrupt Register (0xE000EF00)

Bits	Name	Type	Reset Value	Description
8:0	NVIC->STIR	W	–	Writing the interrupt number sets the pending bit of the interrupt; for example, write 0 to pend external interrupt #0

Figure 19

SCB Registers for Interrupt Control

Besides the NVIC registers, the System Control Block (SCB) also contains some registers that are commonly used for interrupt control

- Interrupt Control and Status Registers[ICSR]
  - Set and clear the pending status os system exceptions including SysTick, PendSV, and NMI
  - Determine the currently executing exception/interrupt number
- Vector table offset register[VTOR]
  - Defines the starting address of the memory being used as the vector table: $SCB->VTOR$
- Application Interrupt and Reset Control Registers[AIRCR]
  - Controlling priority grouping in exception/interrupt priority
  - Providing information about the endianness of the system
- System Handler Priority Register:$SCB->SHP[0to11]$ for system exception
- System Handler Control and State Register:$SCB->SHPCSR$: enable usage faults, memory management faults, bus fault exception for system exception. The pending status of faults and active status of most system exceptions are also avail- able from this register.

SCB Registers for Interrupt Control

Table 7.16 Summary of the Registers in SCB

Address	Register	CMSIS-Core symbol	Function
0xE000ED00	CPU ID	SCB->CPUID	An ID code to allow identification of processor type and revision
0xE000ED04	Interrupt Control and State Register	SCB->ICSR	Control and status of system exceptions
0xE000ED08	Vector Table Offset Register	SCB->VTOR	Enable the vector table to be relocated to other address location
0xE000ED0C	Application Interrupt / Reset Control Register	SCB->AIRCR	Configuration for priority grouping, and self-reset control
0xE000ED10	System Control Register	SCB->SCR	Configuration for sleep modes and low power features
0xE000ED14	Configuration Control Register	SCB->CCR	Configuration for advanced features
0xE000ED18 to 0xE000ED23	System Handler Priority Registers	SCB->SHP[0] to SCB->SHP[11]	Exception priority setting for system exceptions
0xE000ED24	System Handle Control and State Register	SCB->SHCSR	Enable control of fault exceptions, and status of system exceptions
0xE000ED28	Configurable Fault Status Register	SCB->CFSR	Hint information for causes of fault exceptions
0xE000ED2C	HardFault Status Register	SCB->HFSR	Hint information for causes of HardFault exception
0xE000ED30	Debug Fault Status Register	SCB->DFSR	Hint information for causes of debug events
0xE000ED34	MemManage Fault Address Register	SCB->MMFAR	Address Value of Memory Management Fault
0xE000ED38	Bus Fault Address Register	SCB->BFAR	Address Value of Bus Fault
0xE000ED3C	Auxiliary Fault Status Register	SCB->AFSR	Information for device-specific fault status
0xE000ED40 to 0xE000ED44	Processor Feature Registers	SCB->PFR[0] to SCB->PFR[1]	Read only information on available processor features
0xE000ED48	Debug Feature Register	SCB->DFR	Read only information on available debug features
0xE000ED4C	Auxiliary Feature Register	SCB->AFR	Read only information on available auxiliary features

Figure 20

SCB Registers for Interrupt Control

Table 7.16 Summary of the Registers in SCB—*Cont'd*

Address	Register	CMSIS-Core symbol	Function
0xE000ED50 to 0xE000ED5C	Memory Model Feature Registers	SCB->MMFR [0] to SCB->MMFR [3]	Read only information on available memory model features
0xE000ED60 to 0xE000ED70	Instruction Set Attributes Register	SCB->ISAR [0] to SCB-> ISAR [4]	Read only information on instruction set features
0xE000ED88	Co-processor Access Control Register	SCB->CPACR	Register to enable floating point unit feature; available on Cortex®-M4 with floating point unit only

Figure 21

SCB Registers for Exception Priority

Table 7.21 System Handler Priority Registers (SCB->SHP[0 to 11])

Address	Name	Type	Reset Value	Description
0xE000ED18	SCB->SHP[0]	R/W	0 (8-bit)	MemManage Fault Priority Level
0xE000ED19	SCB->SHP[1]	R/W	0 (8-bit)	Bus Fault Priority Level
0xE000ED1A	SCB->SHP[2]	R/W	0 (8-bit)	Usage Fault Priority Level
0xE000ED1B	SCB->SHP[3]	–	–	– (not implemented)
0xE000ED1C	SCB->SHP[4]	–	–	– (not implemented)
0xE000ED1D	SCB->SHP[5]	–	–	– (not implemented)
0xE000ED1E	SCB->SHP[6]	–	–	– (not implemented)
0xE000ED1F	SCB->SHP[7]	R/W	0 (8-bit)	SVC Priority Level
0xE000ED20	SCB->SHP[8]	R/W	0 (8-bit)	Debug Monitor Priority Level
0xE000ED21	SCB->SHP[9]	–	–	– (not implemented)
0xE000ED22	SCB->SHP[10]	R/W	0 (8-bit)	PendSV Priority Level
0xE000ED23	SCB->SHP[11]	R/W	0 (8-bit)	SysTick Priority Level

Figure 22

Special Registers for Handling Exception/Interrupt

- PRIMASK: used to disable all exceptions except NMI and Hard faults
  -
  - To disable all interrupts:
    MOVS R0, #1
    MSR PRIMASK, R0
  - To allow all interrupts:
    MOVS R0, #0
    MSR PRIMASK, R0
- FAULTMASK: similar to the previous one except that it changes the effective current priority level to -1, only the NMI exception handler can works
  - To disable all interrupts:
    MOVS R0, #1
    MSR FAULTMASK, R0
  - To allow all interrupts:
    MOVS R0, #0
    MSR FAULTMASK, R0
- BASEPRI: disable interrupts with priority lower than a certain level
  - To disable all interrupts with priority 0x60-0xFF:
    MOVS R0, #0x60
    MSR BASRPRI, R0
  - To cancel the masking:
    MOVS R0, #0x0
    MSR BASEPRI, R0