Interrupt-Driven Input/Output

Textbook: Chapter 3, Section 3.1 (Cortex regisers & op. modes)

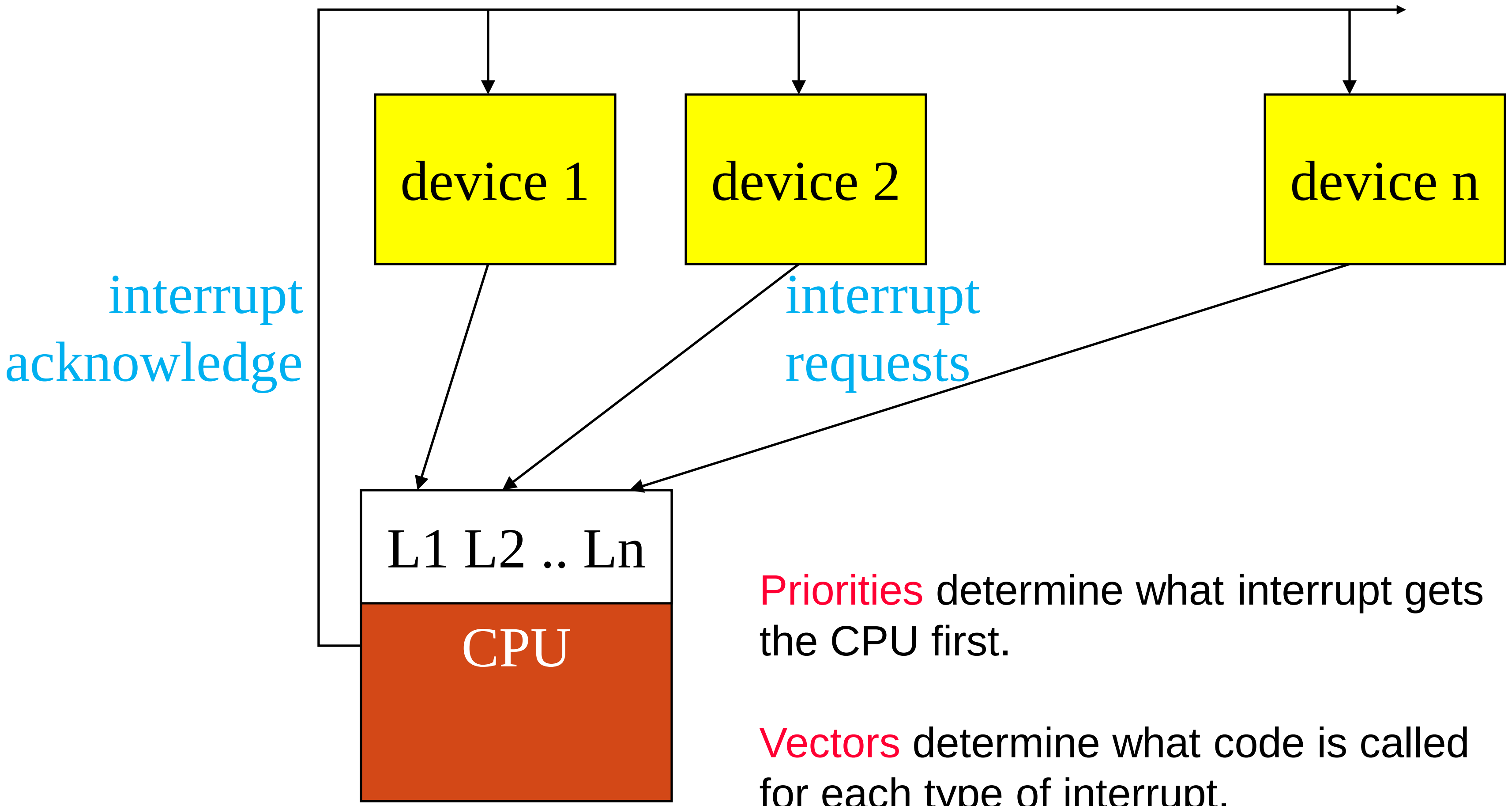
Chapter 9, Section 9.2 (Interrupt concepts)

ARM Cortex-M4 User Guide (Interrupts, exceptions, NVIC)

Outline

- Interrupt vectors and vector table
- Interrupt masks and priorities
- Cortex Nested Vectored Interrupt Controller (NVIC)
- STM32F4 external interrupt signals (EXTI0 – EXTI15)
- System design when interrupts used

Prioritized, vectored interrupts

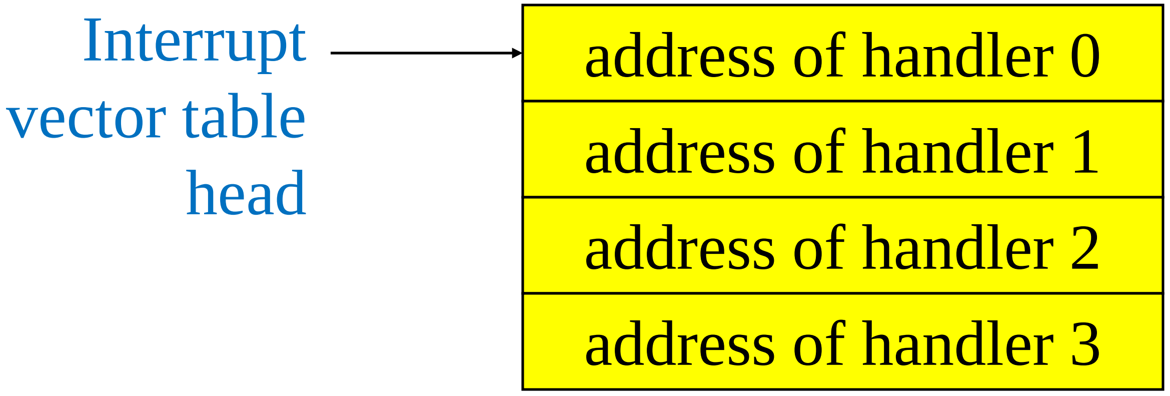


Priorities determine what interrupt gets the CPU first.

Vectors determine what code is called for each type of interrupt.

Interrupt vectors

- Interrupt vector = **address** of handler function
  - Allow different devices to be handled by different code.
- Interrupt vector table:
  - Directly supported by CPU architecture and/or
  - Supported by a separate interrupt-support device/function



Cortex processor exceptions

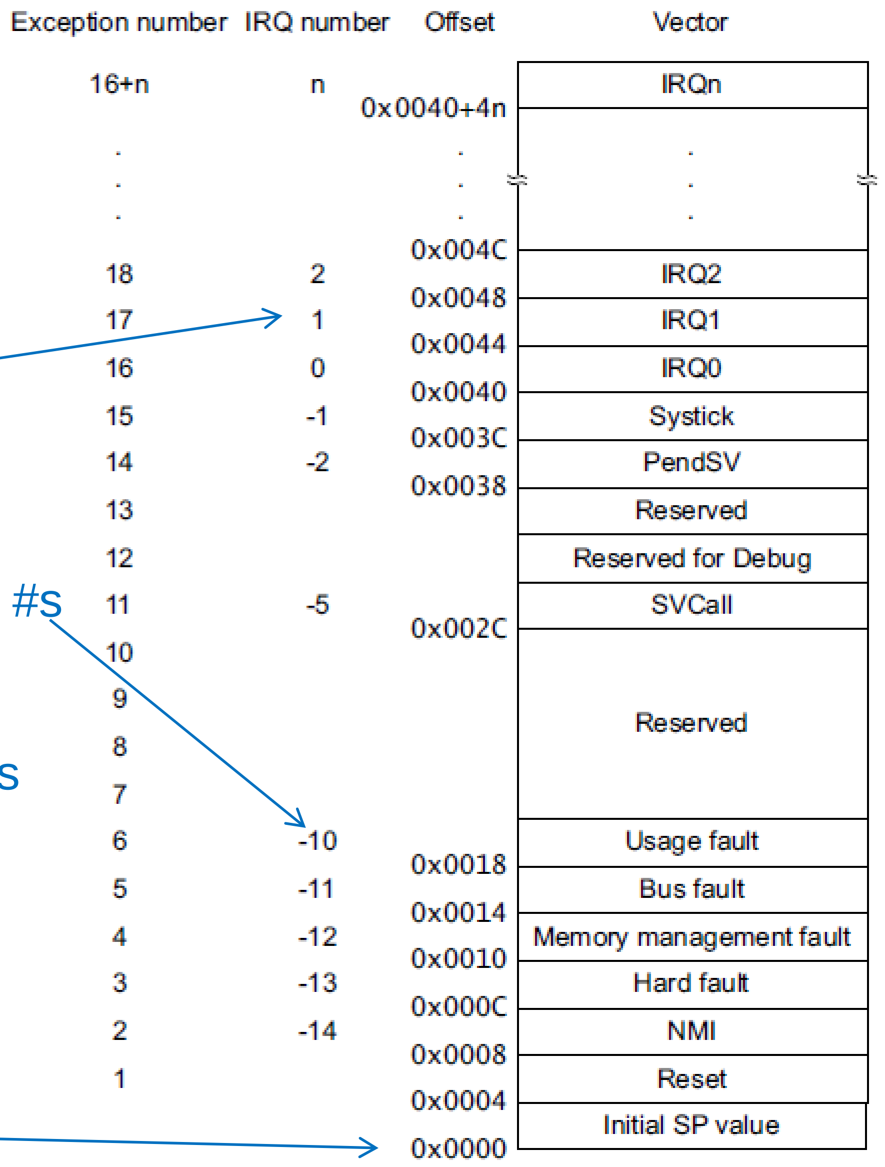
	Priority	IRQ#	Notes
Reset	-3		Power-up or warm reset
NMI	-2	-14	Non-maskable interrupt from peripheral or software
HardFault	-1	-13	Error during exception processing or no other handler
MemManage	Config	-12	Memory protection fault (MPU-detected)
BusFault	Config	-11	AHB data/prefetch aborts
UsageFault	Config	-10	Instruction execution fault - undefined instruction, illegal unaligned access
SVCcall	Config	-5	System service call (SVC) instruction
DebugMonitor	Config		Break points/watch points/etc.
PendSV	Config	-2	Interrupt-driven request for system service
SysTick	Config	-1	System tick timer reaches 0
IRQ0	Config	0	Signaled by peripheral or by software request
IRQ1 (etc.)	Config	1	Signaled by peripheral or by software request

Lowest priority number = highest priority
IRQ #s are used in CMSIS function calls

Vendor peripheral Interrupts
IRQ0 .. IRQ239

Cortex interrupt vector table

- 32-bit vector (handler address) loaded from table into PC (while saving CPU context)
- Peripherals use positive IRQ #s (up to 240 allowed by Cortex - STM32F407 uses 82)
- CPU exceptions use negative IRQ #s
- IRQ # used in CMSIS function calls
- Exception # stored in CPU PSR
- Reset vector includes initial stack pointer



Exception number	IRQ number	Offset	Vector
16+n	n	0x0040+4n	IRQn
.		.	.
.		.	.
.		.	.
		0x004C	
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

STM32F4 Vector Table (partial)

Tech. Ref. Table 61

(Refer to Startup Code)

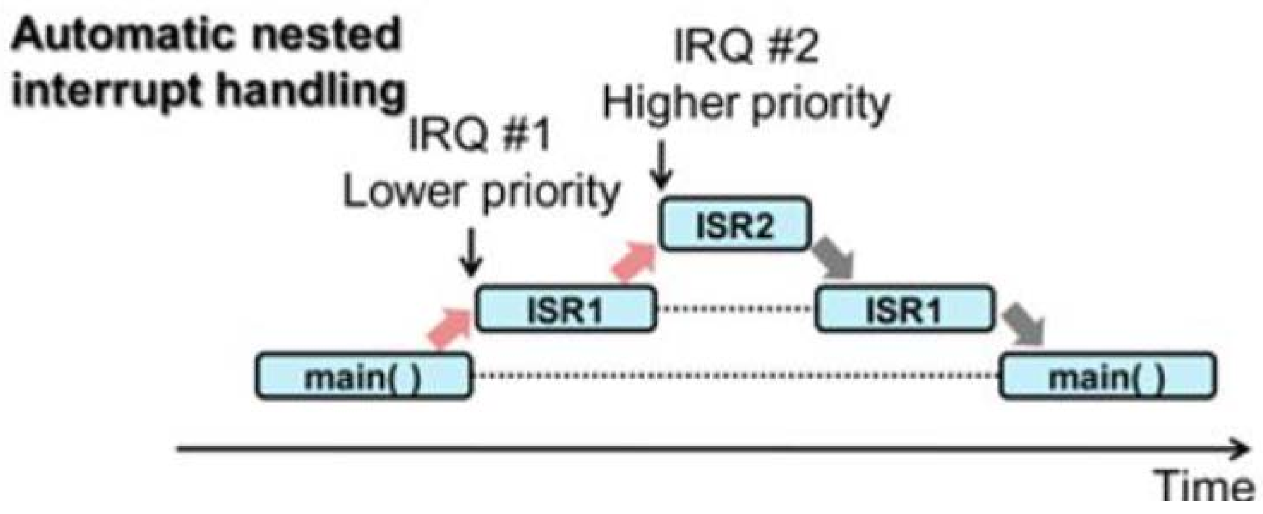
Position	Priority	Type of priority	Acronym	Description	Address
	-	-	-	Reserved	0x0000 0000
	-3	fixed	Reset	Reset	0x0000 0004
	6	settable	SysTick	System tick timer	0x0000 003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	8	settable	PVD	PVD through EXTI line detection interrupt	0x0000 0044
2	9	settable	TAMP_STAMP	Tamper and TimeStamp interrupts through the EXTI line	0x0000 0048
3	10	settable	RTC_WKUP	RTC Wakeup interrupt through the EXTI line	0x0000 004C
4	11	settable	FLASH	Flash global interrupt	0x0000 0050
5	12	settable	RCC	RCC global interrupt	0x0000 0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000 0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000 0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000 0068
11	18	settable	DMA1_Stream0	DMA1 Stream0 global interrupt	0x0000 006C
12	19	settable	DMA1_Stream1	DMA1 Stream1 global interrupt	0x0000 0070
13	20	settable	DMA1_Stream2	DMA1 Stream2 global interrupt	0x0000 0074

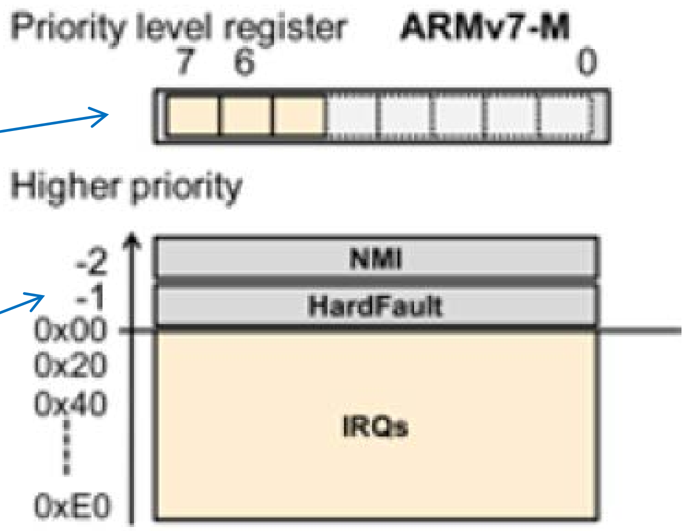
STM32F4 vector table from startup code (partial)

```
__Vectors       DCD     __initial_sp            ; Top of Stack
                DCD     Reset_Handler           ; Reset Handler
                DCD     NMI_Handler             ; NMI Handler
                    ……
                DCD     SVC_Handler             ; SVCall Handler
                DCD     DebugMon_Handler        ; Debug Monitor Handler
                DCD     0                       ; Reserved
                DCD     PendSV_Handler          ; PendSV Handler
                DCD     SysTick_Handler         ; SysTick Handler

                ; External Interrupts
                DCD     WWDG_IRQHandler         ; Window WatchDog
                DCD     PVD_IRQHandler          ; PVD via EXTI Line detection
                DCD     TAMP_STAMP_IRQHandler   ; Tamper/TimeStamps via EXTI
                DCD     RTC_WKUP_IRQHandler     ; RTC Wakeup via EXTI line
                DCD     FLASH_IRQHandler        ; FLASH
                DCD     RCC_IRQHandler          ; RCC
                DCD     EXTI0_IRQHandler        ; EXTI Line0
                DCD     EXTI1_IRQHandler        ; EXTI Line1
                DCD     EXTI2_IRQHandler        ; EXTI Line2
```

Prioritized interrupts





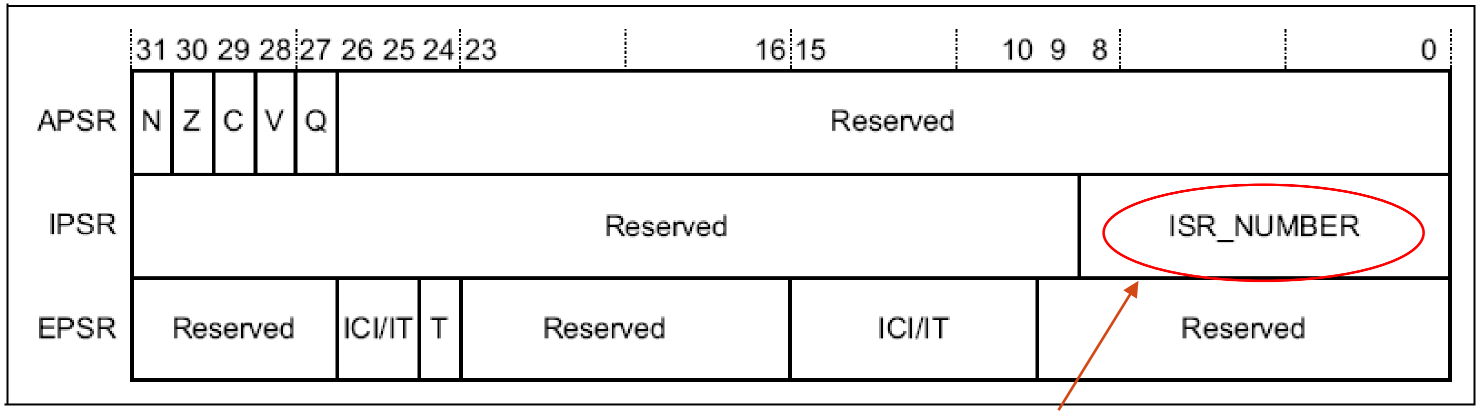
- Up to 256 priority levels
  - 8-bit priority value
  - Implementations may use fewer bits

    STM32F4xx uses upper 4 bits of each priority byte => 16 levels
  - NMI & HardFault priorities are fixed
  - Lowest # = Highest priority

Program Status Register

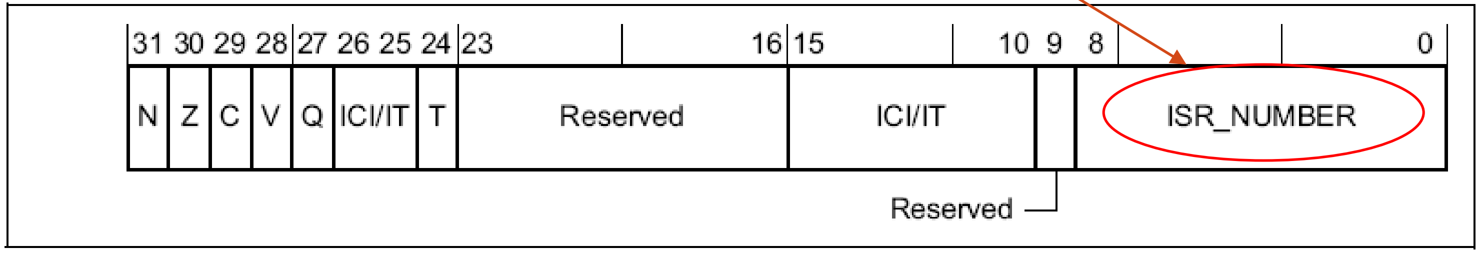
❑ Access 3-parts separately or all at once

Figure 3. APSR, IPSR and EPSR bit assignments



ISR_NUMBER = # of current exception

Figure 4. PSR bit assignments



Q = Saturation, T = Thumb bit

ARM instructions to “access special registers”

MRS Rd,spec ;move from special register (other than R0-R15) to Rd
MSR spec,Rs ;move from register Rs to special register

Interrupt Program Status Register (ISPR)

Bits	Description
Bits 31:9	Reserved
Bits 8:0	**ISR_NUMBER:** 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 .... 10: Reserved 11: SVCall 12: Reserved for Debug 13: Reserved 14: PendSV 15: SysTick 16: IRQ0[1] ....

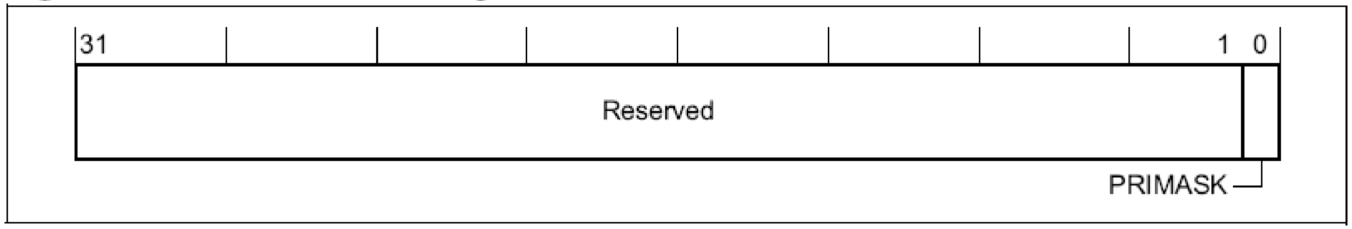
No active interrupt

Cortex CPU interrupts

User (vendor) interrupts IRQ0 – IRQ239

CPU Priority Mask Register

Figure 5. PRIMASK bit assignments



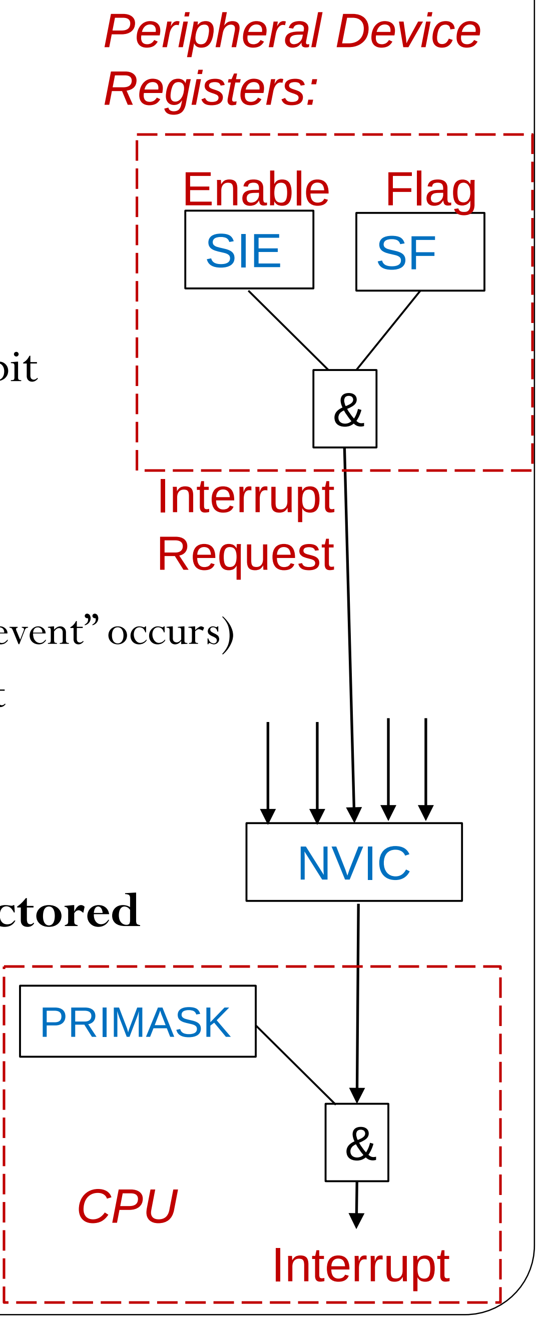
PRIMASK = 1 prevents (masks) activation of all exceptions with configurable priority
PRIMASK = 0 permits (enables) exceptions

Special Cortex-M Assembly Language Instructions

```
CPSIE  I      ;Change Processor State/Enable Interrupts (sets PRIMASK = 0)
CPSID  I      ;Change Processor State/Disable Interrupts (sets PRIMASK = 1)
```

Execute CPSIE I in a program to enable interrupts, after other initialization done.

ARM Cortex-M Interrupts



In the Device:

- Each potential interrupt source has a separate **arm** **(enable)** bit
  - Set for those devices from which interrupts, are to be accepted
  - Deactivate in those devices from which interrupts are not allowed
- Each potential interrupt source has a separate **flag** bit
  - hardware sets the flag when it wishes to request an interrupt (an "event" occurs)
  - software clears the flag in ISR to signify it is processing the request
  - flags can be tested by software if interrupts not desired

In the CPU:

- Cortex-M CPUs receive interrupt requests via the **Nested Vectored Interrupt Controller (NVIC)**
  - NVIC sends highest priority request to the CPU
- Interrupt **enable** conditions in processor
  - Global interrupt enable bit, I, in PRIMASK register
  - Priority level, BASEPRI, of allowed interrupts (0 = all)

Interrupt Conditions

- Four conditions must be true simultaneously for an interrupt to occur:
  1. Arm: control bit for each possible source is set within the peripheral device
  2. Enable: interrupts globally enabled in CPU (I=0 in PRIMASK)
  3. Level: interrupt level must be less than BASEPRI
  4. Trigger: hardware action sets source-specific flag in the peripheral device
- Interrupt remains **pending** if trigger is set but any other condition is not true
  - Interrupt serviced once all conditions become true
- Need to **acknowledge** interrupt
  - Clear trigger flag or will get endless interrupts!

Nested Vectored Interrupt Controller (NVIC)

- Hardware unit that coordinates interrupts from multiple sources
  - Separate enable flag for each interrupt source (**NVIC_ISERx**)
  - Define priority level of each interrupt source (**NVIC_IPRx**)
  - Set/clear interrupts to/from pending state
  - Trigger interrupts through software

- Higher priority interrupts can interrupt lower priority ones
  - Lower priority interrupts are not sent to the CPU until higher priority interrupt service has been completed

NVIC Interrupt Enable Registers

- Three "set interrupt enable" registers – **`NVIC_ISER0, NVIC_ISER1, NVIC_ISER2`**
  - Each 32-bit register has a single enable bit for a particular device
  - Write 1 to a bit to set the corresponding interrupt enable bit
  - Writing 0 has no effect

Register	IRQ numbers	Interrupt numbers
NVIC_ISER0	0-31	16-47
NVIC_ISER1	32-63	48-79
NVIC_ISER2	64-81	80-97

- Three corresponding "clear interrupt enable" registers

  `NVIC_ICER0, NVIC_ICER1, NVIC_ICER2`
  - Write 1 to clear the interrupt enable bit (disable the interrupt)
  - Writing 0 has no effect

NVIC interrupt priority registers

NVIC_IPRx (x=0..20) – Interrupt Priority Registers

- 8-bit priority field for each interrupts (4-bit field in STM32F4)
  - Four 8-bit priority values per register
  - 0 = highest priority level
  - IPR Register# x = IRQ# DIV 4
  - Byte offset within the IPR register = IRQ# MOD 4

  Example: IRQ45

    - 45/4 = 11 with remainder 1 (register NVIC_IPR11, byte offset 1)

      Write priority<<8 to NVIC_IPR11
    - 45/32 = 1 with remainder 13:

      Write 1<<13 to NVIC_ISER1

NVIC register addresses

NVIC_ISER0/1/2 = 0xE000E100/104/108

NVIC_ICER0/1/2 = 0xE000E180/184/188

NVIC_IPR0/1/2/…/20 = 0xE00E400/404/408/40C/…./500

```
;Example – Enable EXTI0 with priority 5 (EXTI0 = IRQ6)
NVIC_ISER0  EQU  0xE000E100        ;bit 6 enables EXTI0
NVIC_IPR1   EQU  0xE000E404        ;3rd byte = EXTI0 priority
        ldr     r0,=NVIC_ISER0
        mov     r1,#0x0040                  ;Set bit 6 of ISER0 for EXTI0
        str     r1,[r0]
        ldr     r0,=NVIC_IPR1               ;IRQ6 priority in IPR1[23:16]
        ldr     r1,[r0]                     ;Read IPR1
        bic     r1,#0x00ff0000              ;Clear [23:16] for IRQ6
        orr     r1,#0x00500000              ;Bits [23:20] = 5
        str     r1,[r0]                     ;Upper 4 bits of byte = priority
```

CMSIS[1] functions

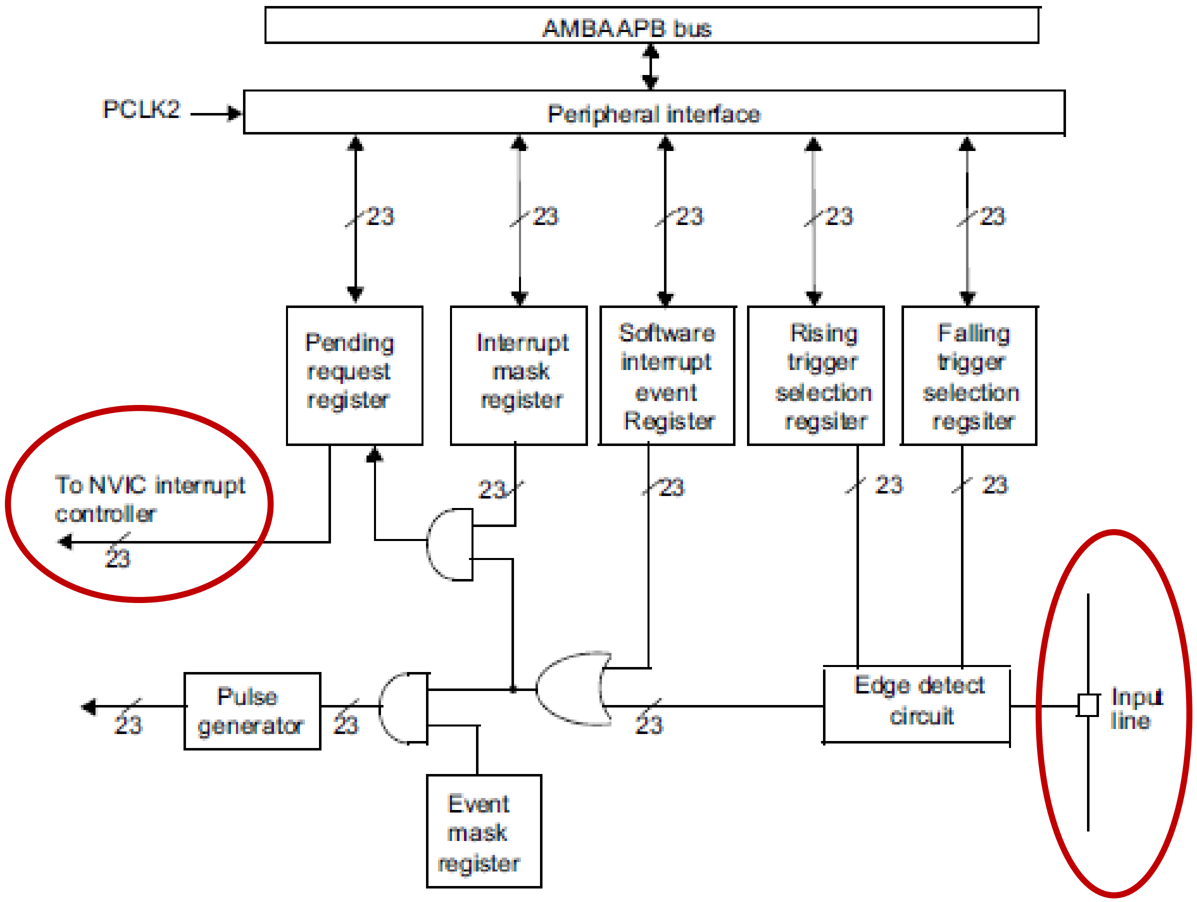
- NVIC_Enable(IRQn_Type IRQn)
- NVIC_Disable(IRQn_Type IRQn)
- NVIC_SetPending(IRQn_Type IRQn)
- NVIC_ClearPending(IRQn_Type IRQn)
- NVIC_GetPending(IRQn_Type IRQn)
- NVIC_SetPriority(IRQn_Type IRQn,unit32_t priority)
- NVIC_GetPriority(IRQn_Type IRQn)

[1]CMSIS = Cortex Microcontroller Software Interface Standard

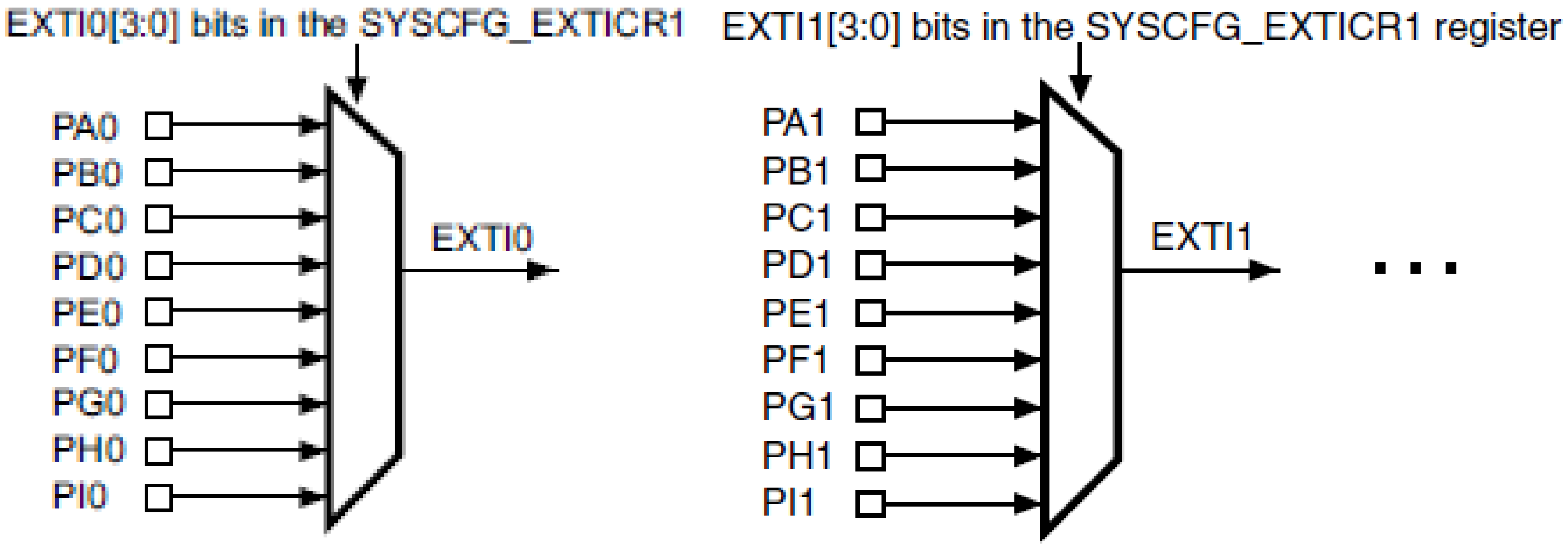
- Vendor-independent hardware abstraction layer for Cortex-M
- Facilitates software reuse
- Other CMSIS functions: System tick timer, Debug interface, etc.

STM32F4 external interrupt/event controller

23 edge detectors to trigger events and interrupts signaled by 240 GPIO pins and 7 internal events.



STM32F4 external interrupt sources


EXTI0[3:0] bits in the SYSCFG_EXTICR1
EXTI1[3:0] bits in the SYSCFG_EXTICR1 register
PA0
PB0
PC0
PD0
PE0
PF0
PG0
PH0
PI0
EXTI0
PA1
PB1
PC1
PD1
PE1
PF1
PG1
PH1
PI1
EXTI1
. . .

- ❖ 16 multiplexers: one each for EXTI0..EXTI15.
- ❖ Mux x select bit x of Port "N" as EXTIx.
  - x = 0..15
  - "N" = 0..8 for ports A..I

	15 12	11 8	7 4	3 0
SYSCFG_EXTICR1	EXTI3	EXTI2	EXTI1	EXTI0

STM32F4 EXTI Registers

- 23 bits in each register - control 23 interrupts/events
- EXTI_IMR – interrupt mask register
  - 0 masks (disables) the interrupt
  - 1 unmasks (enables) the interrupt
- EXTI_RTSR/FTSR – rising/falling trigger selection register
  - 1 to enable rising/falling edge to trigger the interrupt/event
  - 0 to ignore the rising/falling edge
- EXTI_PR – interrupt/event pending register
  - read 1 if interrupt/event occurred
  - clear bit by writing 1 *(writing 0 has no effect)*
  - write 1 to this bit in the interrupt handler to clear the pending state of the interrupt

Example: Enable EXTI0 as rising-edge triggered

```
;System Configuration Registers
SYSCFG    EQU       0x40013800
EXTICR1   EQU       0x08
;External Interrupt Registers
EXTI      EQU       0x40013C00
IMR       EQU       0x00      ;Interrupt Mask Register
RTSR      EQU       0x08      ;Rising Trigger Select
FTSR      EQU       0x0C      ;Falling Trigger Select
PR        EQU       0x14      ;Pending Register

        ;select PC0 as EXTI0
        ldr     r1,=SYSCFG          ;SYSCFG selects EXTI sources
        ldrh    r2,[r1,#EXTICR1]    ;EXTICR1 = sources for EXTI0 - EXTI3
        bic     r2,#0x000f          ;Clear EXTICR1[3-0] for EXTI0 source
        orr     r2,#0x0002          ;EXTICR1[3-0] = 2 to select PC0 as EXTI0 source
        strh    r2,[r1,#EXTICR1]    ;Write to select PC0 as EXTI0
        ;configure EXTI0 as rising-edge triggered
        ldr     r1,=EXTI        ;EXTI register block
        mov     r2,#1           ;bit #0 for EXTI0 in each of the following registers
        str     r2,[r1,#RTSR]   ;Select rising-edge trigger for EXTI0
        str     r2,[r1,#PR]     ;Clear any pending event on EXTI0
        str     r2,[r1,#IMR]    ;Enable EXTI0
```

Interrupt Rituals

- Things you must do in every ritual
  - Initialize data structures (counters, pointers)
  - Arm interrupt in the peripheral device
    - Enable a flag to trigger an interrupt
    - Clear the flag (to ignore previous events)
  - Configure NVIC
    - Enable interrupt (**NVIC_ISERx**)
    - Set priority (**NVIC_IPRx**)
  - Enable CPU Interrupts
    - Assembly code `CPSIE I`
    - C code `EnableInterrupts();`

Interrupt Service Routine (ISR)

- Things you must do in every interrupt service routine
  - Acknowledge
    - clear flag that requested the interrupt
    - SysTick is exception; automatic acknowledge
  - Maintain contents of R4-R11 (AAPCS)
  - Communicate via shared global variables

Sources of interrupt overhead

- Handler execution time.
- Interrupt mechanism overhead.
- Register save/restore.
- Pipeline-related penalties.
- Cache-related penalties.
- Interrupt "latency" = time from activation of interrupt signal until event serviced.
- ARM worst-case latency to respond to interrupt is 27 cycles:
  - 2 cycles to synchronize external request.
  - Up to 20 cycles to complete current instruction.
  - 3 cycles for data abort.
  - 2 cycles to enter interrupt handling state.