Lab 4 – Interrupt-driven operations

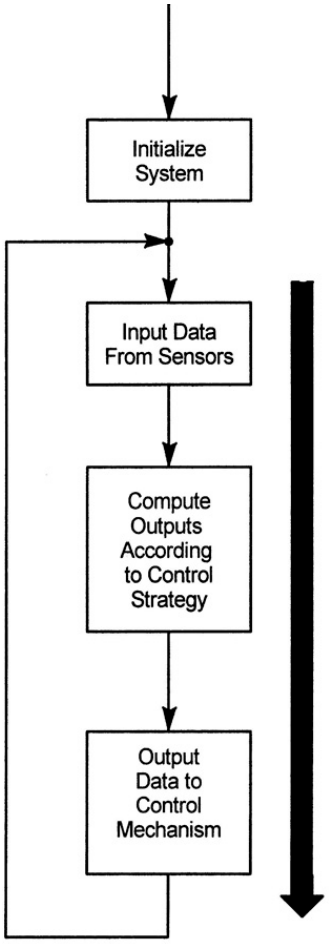
- Interrupt handling in Cortex-M CPUs
- Nested Vectored Interrupt Controller (NVIC)
- Externally-triggered interrupts via GPIO pins
- Software setup for interrupt-driven applications

Interrupt-driven operations

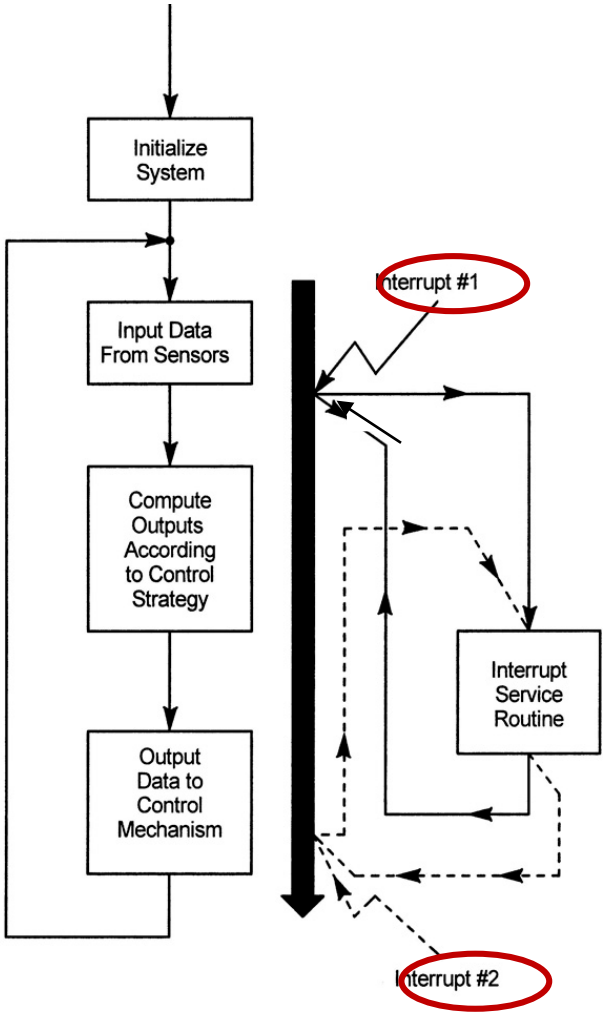
- An **interrupt** is an event that initiates the automatic transfer of software execution from one program thread to an **interrupt handler**
- Event types:
  - Signal from a "device" (keyboard, timer, data converter, etc.)
    - Device external to the CPU (possibly within a microcontroller)
    - Signals that a device needs, or is able to provide service
      (i.e. device goes from "busy" to "ready")
    - Asynchronous to the current program thread
    - Allows CPU to do other work until device needs service!
  - An internal event or "exception" caused by an instruction
    Ex. invalid memory address, divide by 0, invalid op code
  - A software interrupt instruction
    Ex. ARM Cortex SVC (supervisor call) instruction

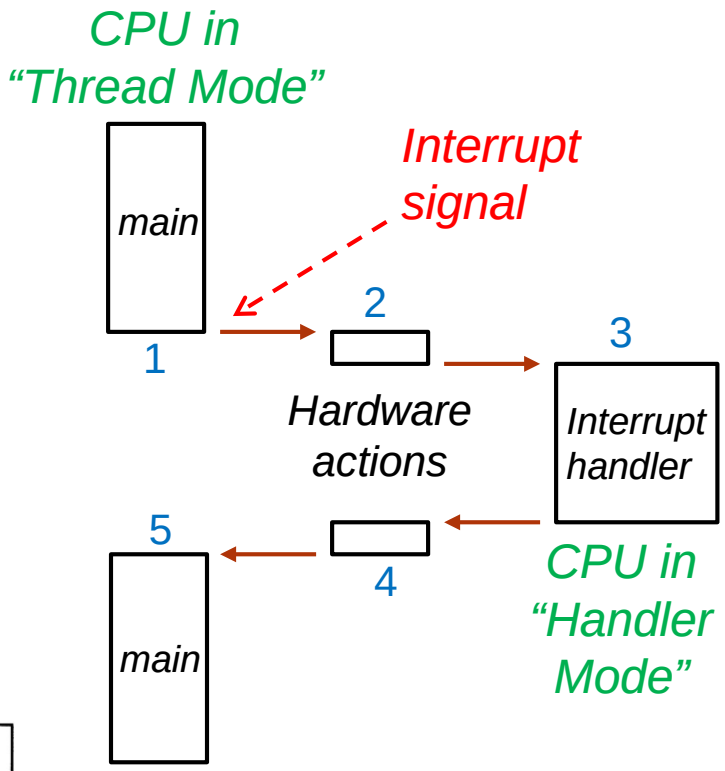
Interrupts in control systems

Continuous loop



Loop with interrupts





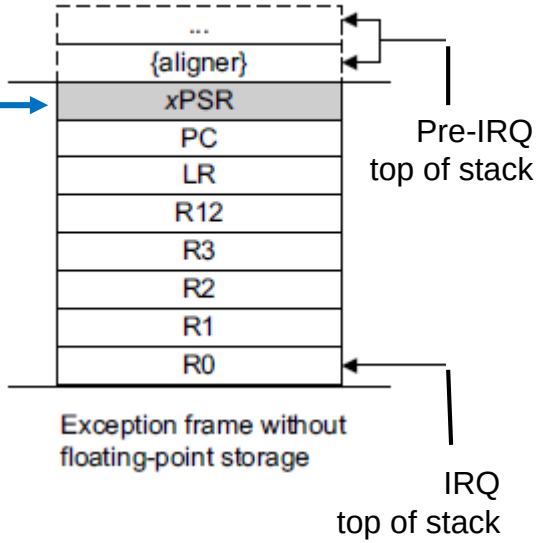
Handling an interrupt request

1. Suspend main thread
2. Save CPU state
3. Execute interrupt handler
4. Restore CPU state
5. Resume main thread

Cortex-M Interrupt Process

(much of this is transparent when using C)

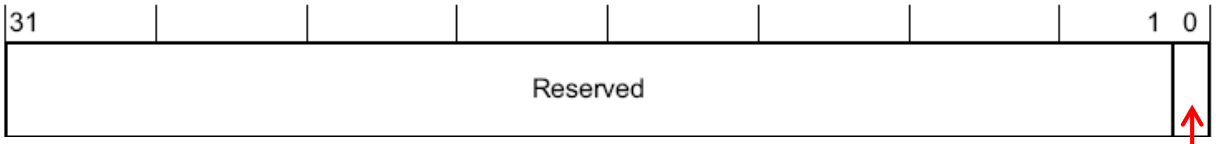
1. Interrupt signal detected by CPU
2. Suspend main program execution
   - finish current instruction
   - save CPU state (push registers onto stack)
   - set LR to 0xFFFFFFF9 (indicates interrupt return)
   - set IPSR to interrupt number
   - load PC with ISR address from vector table
3. Execute interrupt service routine (ISR)
   - save other registers to be used [1]
   - clear the "flag" that requested the interrupt
   - perform the requested service
   - communicate with other routines via global variables
   - restore any registers saved by the ISR [1]
4. Return to and resume main program by executing ***BX LR***
   - saved state is restored from the stack, including PC



[1] C compiler takes care of saving/restoring registers

Special CPU registers

Prioritized Interrupts Mask Register (PRIMASK)



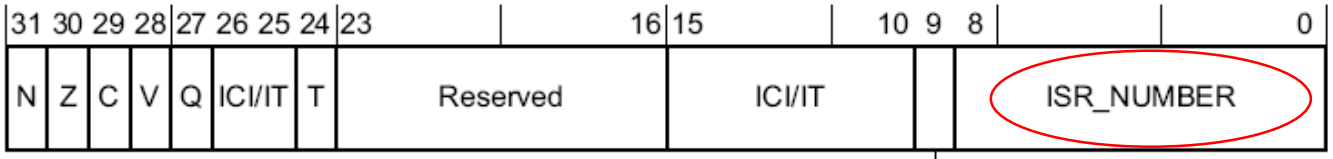
PRIMASK = 1 prevents (masks) activation of all exceptions with configurable priority
PRIMASK = 0 permits (enables) exceptions

Use CMSIS[1] functions to enable/disable interrupts

```
__enable_irq();    //enable interrupts (set PRIMASK=0)
__disable_irq();   //disable interrupts (set PRIMASK=1)
```

double-underscore

Processor Status Register (PSR)

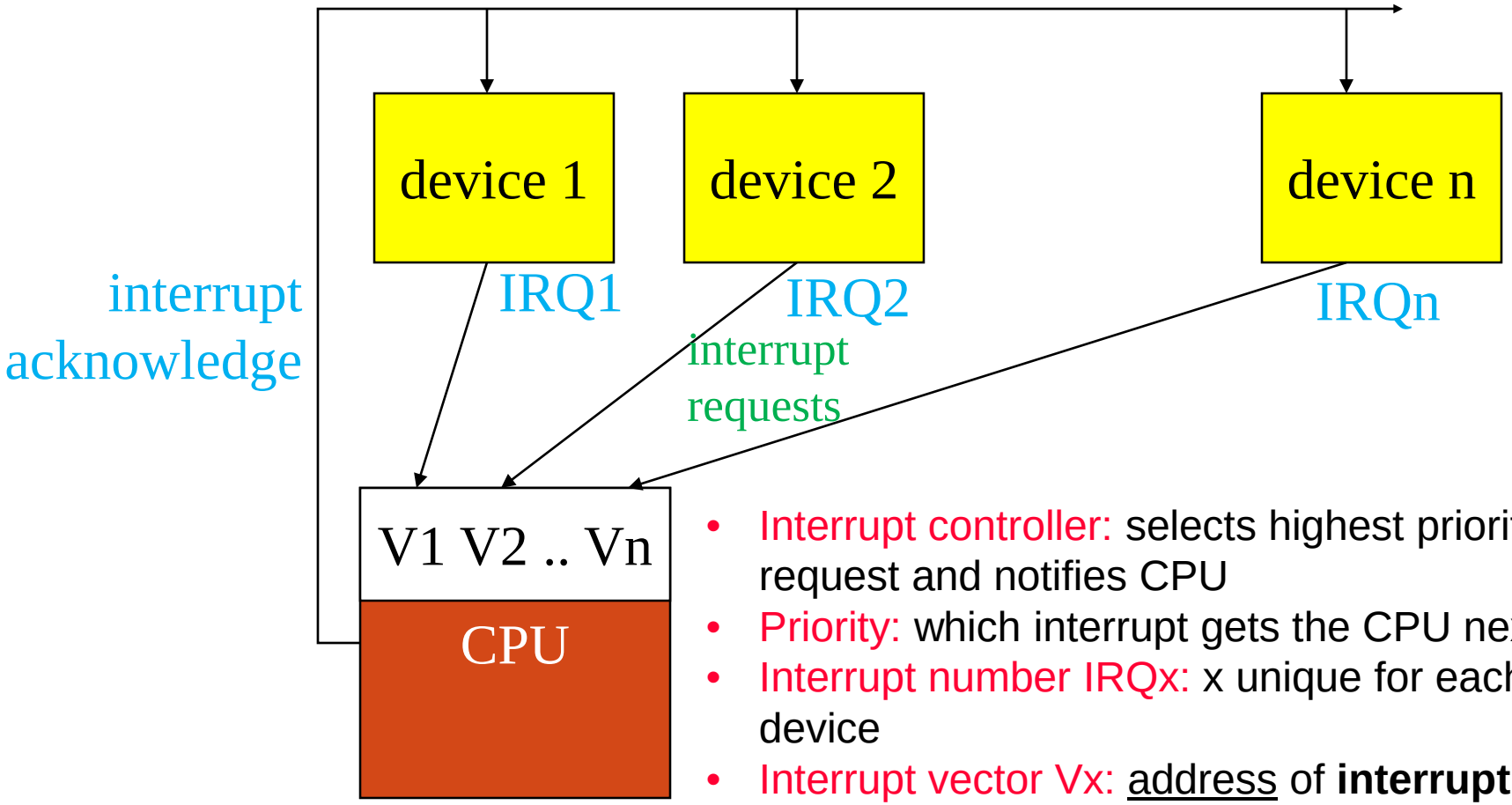


ALU flags

# of current exception
(lower priority cannot interrupt)

[1] **Cortex Microcontroller Software Interface Standard** – Functions for all ARM Cortex-M CPUs. Automatically included in your project; defined in header files: *core_cmFunc.h, core_cm3.h*

Prioritized, vectored interrupts



- Interrupt controller: selects highest priority request and notifies CPU
- Priority: which interrupt gets the CPU next?
- Interrupt number IRQx: x unique for each device
- Interrupt vector Vx: address of **interrupt handler** for device generating IRQx.
  - arranged by interrupt # in a “Vector Table”

Cortex-M CPU and peripheral exceptions

	Priority[1]	IRQ#[2]	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

ARM CPU Exceptions

Vendor peripheral interrupts
IRQ0 .. IRQ44

[1] *Lowest priority # = highest priority*
[2] *IRQ# used in CMSIS function calls*

Interrupt Table for STM32L1xx Peripherals

IRQ6

Tech. Ref. Manual: Table 48

Also - refer to vector table in startup code

Position	Priority	Type of priority	Acronym	Description	Address
0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER_STAMP	Tamper and TimeStamp through EXTI line interrupts	0x0000_0048
3	10	settable	RTC_WKUP	RTC Wakeup through EXTI line interrupt	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_Channel1	DMA1 Channel1 global interrupt	0x0000_006C
25	32	settable	TIM9	TIM9 global interrupt	0x0000_00A4
26	33	settable	TIM10	TIM10 global interrupt	0x0000_00A8
27	34	settable	TIM11	TIM11 global interrupt	0x0000_00AC
28	35	settable	TIM2	TIM2 global interrupt	0x0000_00B0
29	36	settable	TIM3	TIM3 global interrupt	0x0000_00B4
30	37	settable	TIM4	TIM4 global interrupt	0x0000_00B8
31	38	settable	I2C1_EV	I^2C1 event interrupt	0x0000_00BC
32	39	settable	I2C1_ER	I^2C1 error interrupt	0x0000_00C0
33	40	settable	I2C2_EV	I^2C2 event interrupt	0x0000_00C4
34	41	settable	I2C2_ER	I^2C2 error interrupt	0x0000_00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000_00CC

External interrupts

Timer interrupts

STM32L1 vector table in startup code *(partial)*

```
__Vectors
        DCD    __initial_sp                ; Pointer to 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
        DCD    EXTI3_IRQHandler            ; EXTI Line3
        DCD    EXTI4_IRQHandler            ; EXTI Line4
```

Use these names for your interrupt handler functions

Interrupt signal: from device to CPU

("Enabled" in three places)

In each peripheral device:

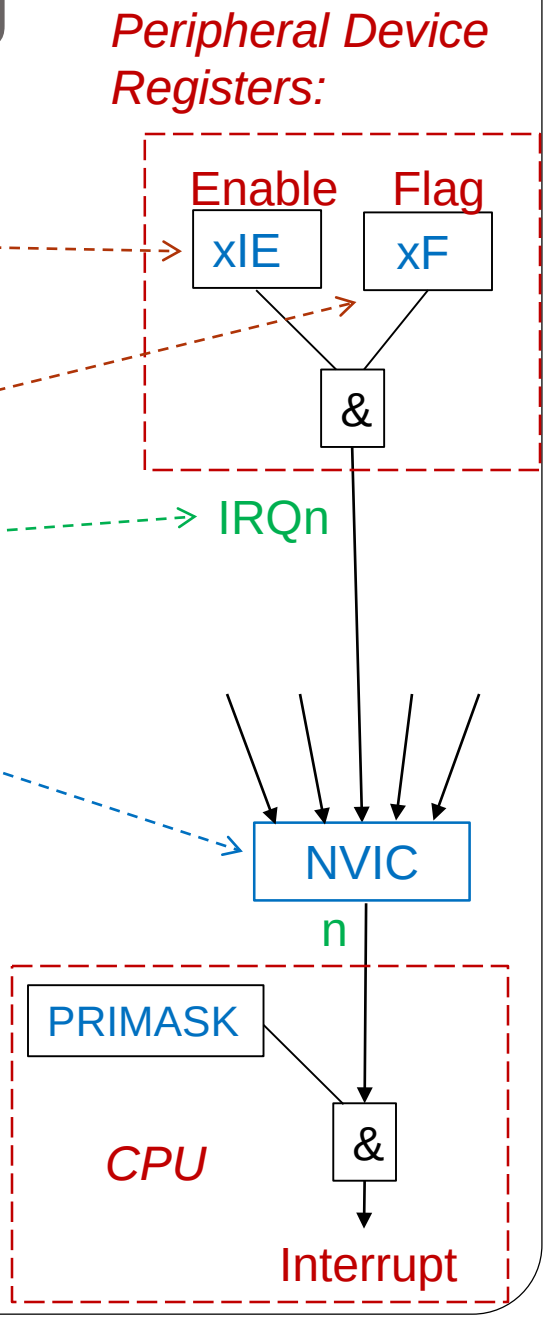
- Each potential interrupt source has a separate **enable** bit
  - Set to enable the peripheral to send an interrupt signal to the CPU
  - Clear to prevent the peripheral from interrupting the CPU
- Each potential interrupt source has a separate **flag** bit
  - Flag set by hardware when an "event" occurs
  - Interrupt request = (flag & enable)
  - Test flag in software if interrupt not desired
  - **ISR software must clear the flag** to acknowledge the request

Nested Vectored Interrupt Controller (NVIC)

- Receives all interrupt requests
- Each has an enable bit and a priority within the NVIC
- # of highest-priority enabled interrupt sent to the CPU

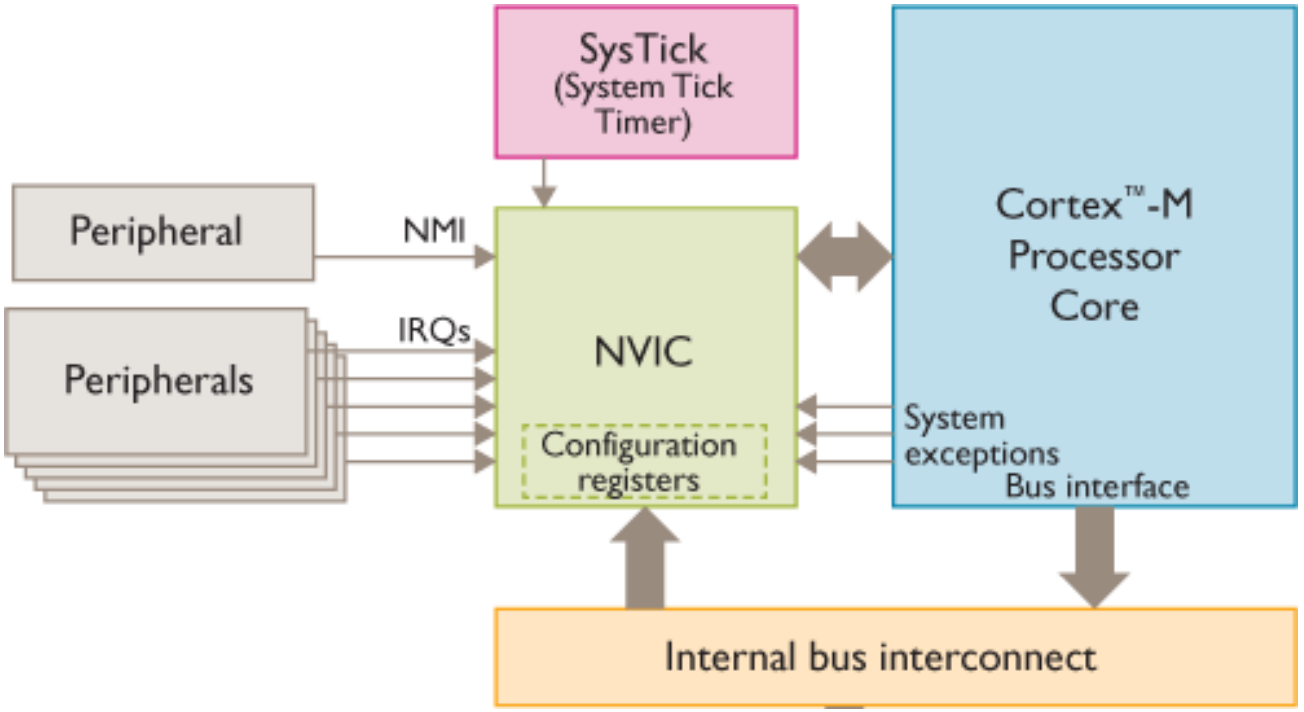
Within the CPU:

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



Nested Vectored Interrupt Controller (NVIC)

- NVIC manages and prioritizes external interrupts in Cortex-M
  - 45 IRQ sources from STM32L1xx peripherals
- NVIC interrupts CPU with IRQ# of highest-priority IRQ signal
  - CPU uses IRQ# to access the vector table & get intr. handler start address



NVIC setup: enable interrupts

- Each IRQ has its own **enable** bit within the NVIC
  - NVIC only considers IRQs whose enable bits are set
  - *Interrupt Set Enable Register*: each bit **enables** one interrupt
    - CMSIS function: **NVIC_EnableIRQ**(n); //set bit n to **enable** IRQn
  - *Interrupt Clear Enable Register:* each bit **disables** one interrupt
    - CMSIS function: **NVIC_DisableIRQ**(n); //set bit n to **disable** IRQn

- For convenience, *stm3211xx.h* defines symbols for each IRQn

  Examples: EXTI0_IRQn = 6 ; //External interrupt EXTI0 is IRQ #6

  TIM3_IRQn = 29 ; //Timer TIM3 interrupt is IRQ #29

  Usage:

```
NVIC_EnableIRQ(EXTI0_IRQn);   //enable external interrupt EXTI0
NVIC_DisableIRQ(TIM3_IRQn);   //disable interrupt from timer TIM3
```

NVIC: interrupt pending flags

- Each IRQ has an **interrupt pending flag** within the NVIC
  - Pending flag set by NVIC when it detects IRQn request, and IRQn status changed to "pending"
  - IRQn status changes to "active" when its interrupt handler is entered
  - NVIC clears pending flag when handler exited, changing status to "inactive"

    - If IRQn still active when exiting handler, or IRQn reactivates while executing the handler, the pending flag remains set and **triggers another interrupt**

      Avoid duplicate service by clearing IRQn pending flag in software:

      CMSIS function: ***NVIC_ClearPendingIRQ(IRQn);***

  - Pending status can be checked:
    - CMSIS function: **NVIC_GetPendingIRQ**(IRQn);
  - Software can force IRQn into pending state to simulate an IRQn request
    - CMSIS function: **NVIC_SetPendingIRQ**(IRQn);

NVIC: interrupt priorities

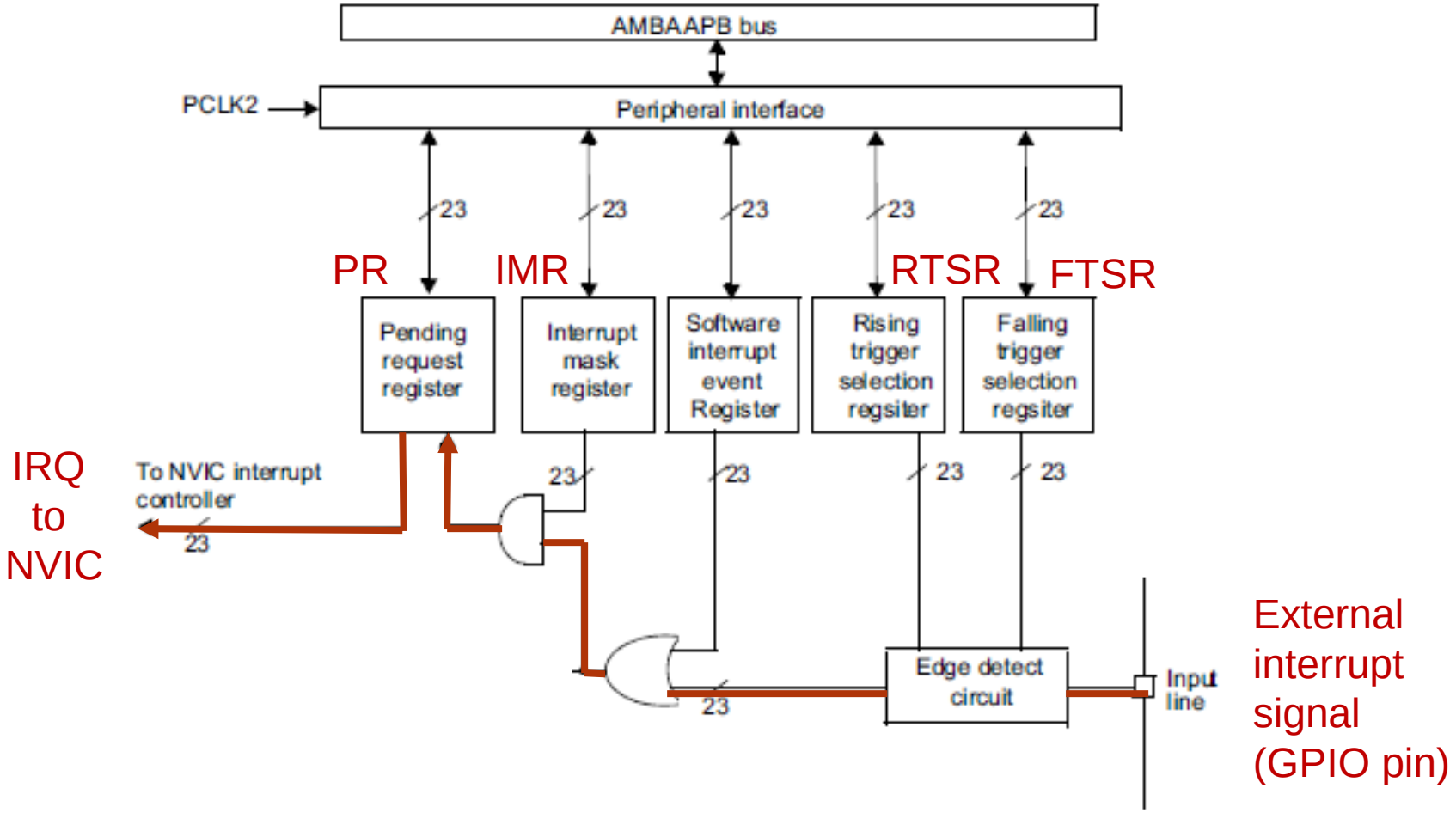
- Each IRQn assigned a **priority** within the NVIC
  - NVIC selects highest-priority pending IRQ to send to CPU
    - Lower priority# = higher priority (default value = 0)
    - Higher priority interrupt can interrupt lower priority one
    - Lower priority interrupt not sent to CPU until higher priority interrupt service completed
    - If equal priorities, lower IRQ# selected
  - Priorities stored in NVIC *Interrupt Priority Registers*
    - STM32L1xx uses 4-bit priority value (0..15)

      (NVIC registers allocate 8 bits per IRQ#, but vendors may use fewer bits)
  - Set priority via CMSIS function: *NVIC_SetPriority(IRQn, priority);*

    Ex: *NVIC_SetPriority(EXTI0_IRQn, 1);* //set ext. intr. EXTI0 priority = 1

STM32L1xx external interrupt/event controller

- External devices can interrupt CPU via GPIO pins
  (Some microcontrollers have dedicated interrupt pins)
- Up to 16 external interrupts (EXTI0-EXTI15), plus 7 internal events

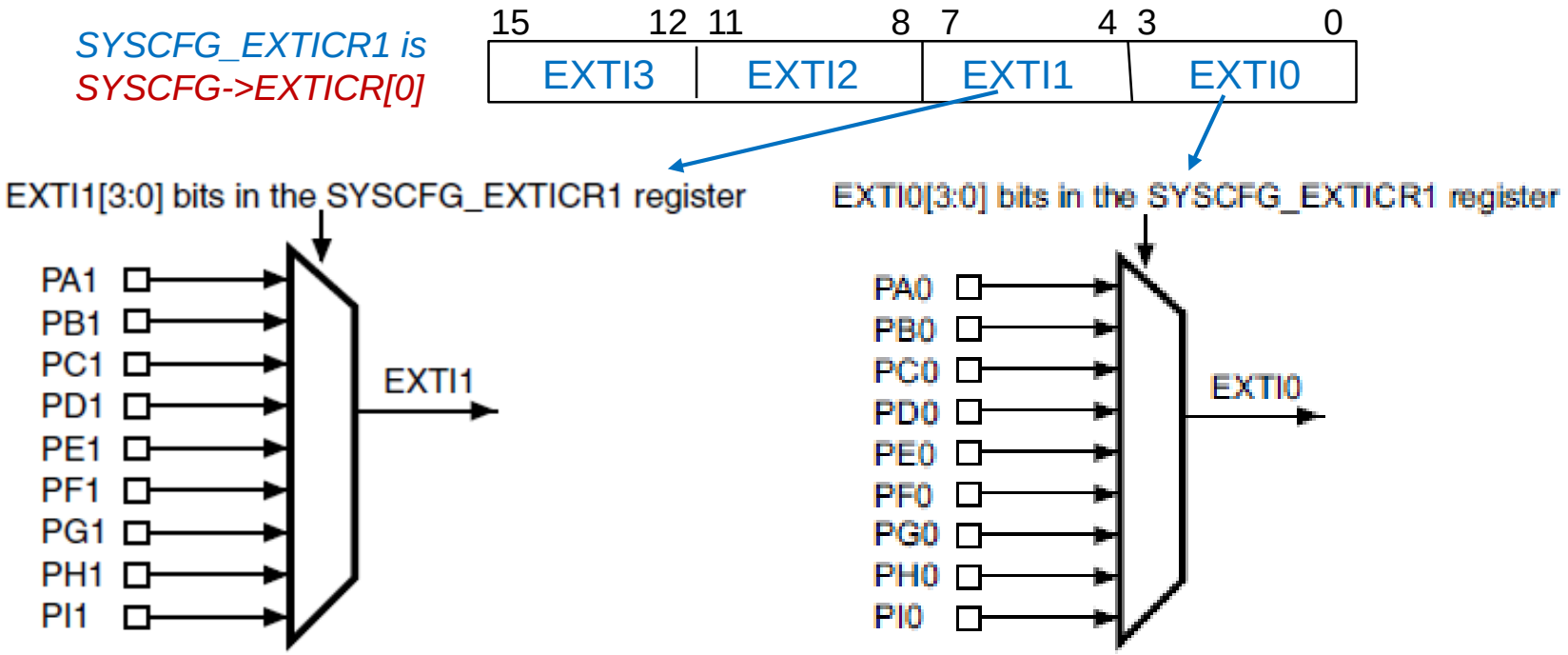


STM32L1xx external interrupt sources

(Select in System Configuration Module – SYSCFG)

- 16 multiplexers select GPIO pins as external interrupts EXTI0..EXTI15
- Mux inputs selected via 4-bit fields of EXTICR[k] registers (k=0..3)
  - EXTIx = 0 selects PAx, 1 selects PBx, 2 selects PCx, etc.
  - EXTICR[0] selects EXTI3-EXTI0; EXTICR[1] selects EXTI7-EXTI4, etc

SYSCFG_EXTICR1 is SYSCFG->EXTICR[0]



Example: Select pin PC2 as external interrupt EXTI2

```
SYSCFG->EXTICR[0] &= 0xF0FF;   //clear EXTI2 bit field
SYSCFG->EXTICR[0] |=  0x0200;   //set EXTI2 = 2 to select PC2
```

STM32L1xx EXTI configuration registers

- Register bits 15-0 control EXTI15-EXTI0, respectively
- **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_IMR** – interrupt mask register
  - 1 enables ("unmasks") the corresponding interrupt
  - 0 disables ("masks") the interrupt
- **EXTI_PR** – interrupt pending register
  - bit set to 1 by hardware if interrupt/event occurred *(bit is readable)*
  - clear bit by writing 1 *(writing 0 has no effect)*
  - **interrupt handler** must write **1** to this bit to clear the pending state of the interrupt (to cancel the IRQn request)

Example: Configure EXTI2 as rising-edge triggered

```
EXTI->RTSR |= 0x0004;      //Bit2=1 to make EXTI2 rising-edge trig.
EXTI->IMR  |= 0x0004;      //Bit2=1 to enable EXTI2
EXTI->PR   |= 0x0004;      //Bit2=1 to clear EXTI2 pending status
```

Clearing pending status needs to be done in the interrupt handler after every interrupt.

Project setup for interrupt-driven applications

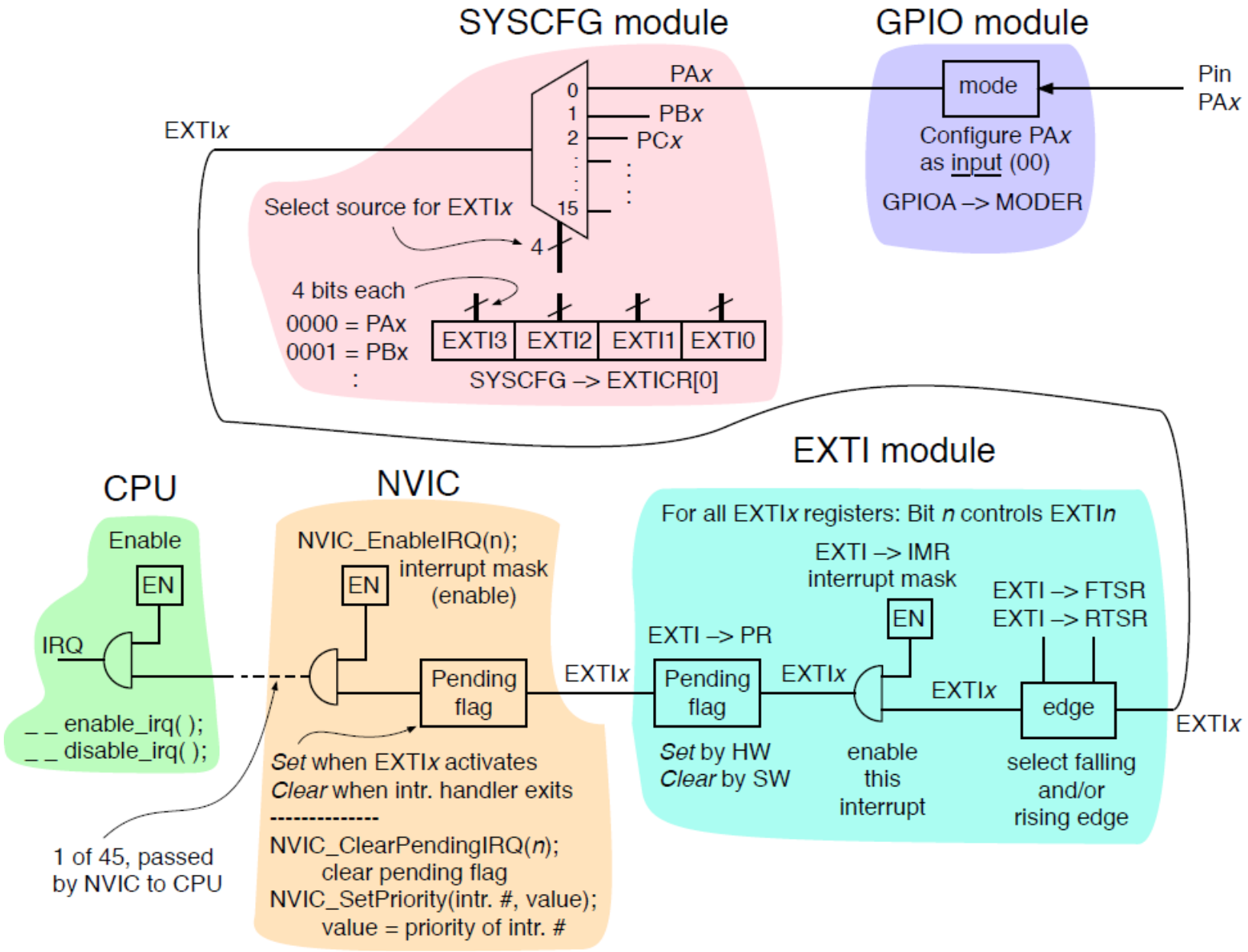
1. Write an **interrupt handler** for each peripheral
   - Clear the flag that requested the interrupt (acknowledge the intr. request)
   - Perform the desired action(s)
     - communicate with other functions via shared global variables
   - Use function names from the startup file vector table

     Example: *void EXTI4_IRQHandler () { statements }*
2. Perform all **initialization** for each peripheral device:
   - Initialize the **device**, "enable" its interrupt, and clear its "flag"

     Example: External interrupt EXTIn
     - Configure GPIO pin n as a digital input
     - Select the pin as the EXTIn source (in SYSCFG module)
     - Enable interrupt to be requested when a flag is set by the desired event (rising/falling edge)
     - Clear the pending flag (to ignore any previous events)
   - **NVIC**
     - Enable interrupt: *NVIC_EnableIRQ (IRQn);*
     - Set priority (if desired): *NVIC_SetPriority (IRQn, priority);*
     - Clear pending status: *NVIC_ClearPendingIRQ (IRQn);*
3. Initialize counters, pointers, global variables, etc.
4. Enable CPU Interrupts: *__enable_irq();*

(diagram on next slide)

Signal flow/setup for External Interrupt EXTI*x*, *x* = 0...15



Lab experiment

- Main program displays two counting sequences on LEDs,
  - First increases at period ½ second from 0-9 & repeat
  - Second increases at period 1 second from 0-9 & repeat
- If **user button** (connected to PA0) pressed, interrupt the main program and change count direction to decreasing
- If **"Static IO" button** (connect to PA1) pressed, interrupt the main program and change count direction to increasing

  (no change if new direction = old direction)
- Each interrupt should toggle one of the on-board LEDs
- Interrupt routines can set a global "direction" variable