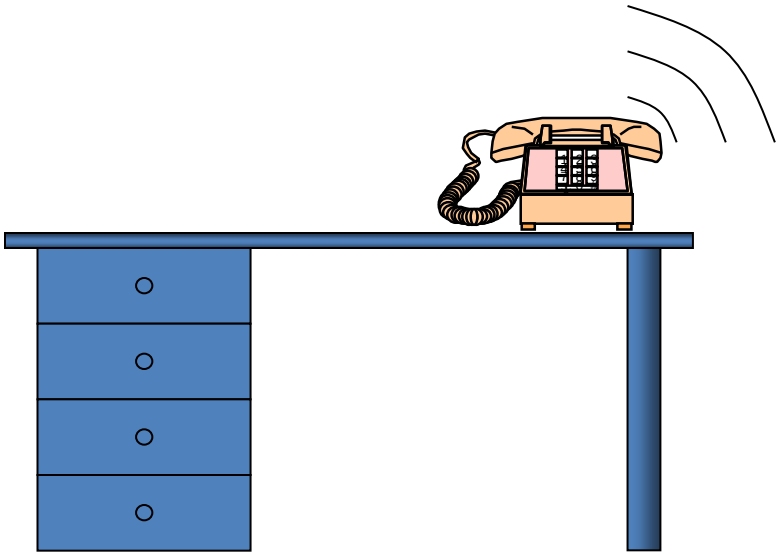
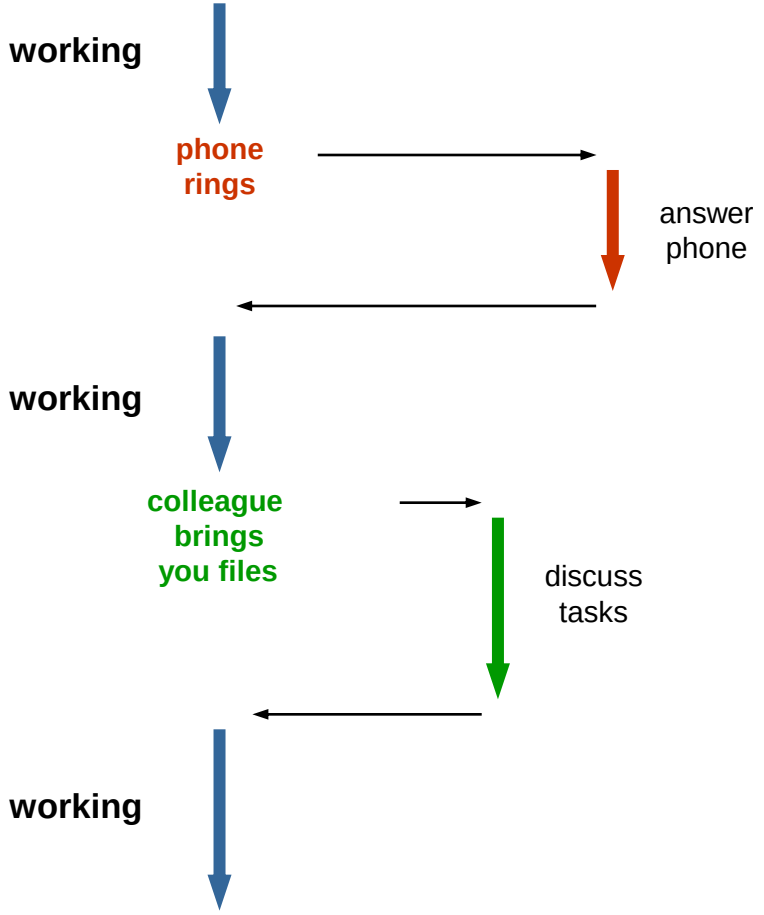
Exceptional Control Flow

Computer Engineering 1

Motivation

- **You are working at your desk!**





Motivation

School of Engineering
InES Institute of Embedded Systems

- **Computers work concurrently on several tasks**
  - Often wait until result is available, e.g. network request
  - Fill idle time with useful tasks → efficient use of processing power
  - Interrupts signal, that result of a request is available
  - Requires switching between programs
- **Unexpected events – exceptions**
  - Division by 0, unaligned accesses, etc.
  - Must be caught and corresponding actions need to be triggered
- **Time critical events with high priority**
  - Interrupt running program → process event

Agenda

School of Engineering
InES Institute of Embedded Systems

- **Polling**
- **Interrupt-Driven I/O**
- **Exceptions Cortex-M3/M4**
- **Interrupt-System Cortex-M3/M4**
- **Vector Table**
- **Storing the Context**
- **External Interrupt Pins**
- **Interrupt Control**
- **Nested Exceptions**
- **Special Interrupt Situations**
- **CMSIS Functions**
- **Data Consistency**

The lecture does not cover all the features of Cortex-M3/4 exception processing, in particular the following simplifications apply:

- No distinction between handler mode and thread mode
- No distinction between preempt priority and sub-priority fields
- Tail chaining and late arrival are not covered
- Instructions that are abandoned and restarted/resumed after interrupt service are not covered (e.g. LDM)

Learning Objectives

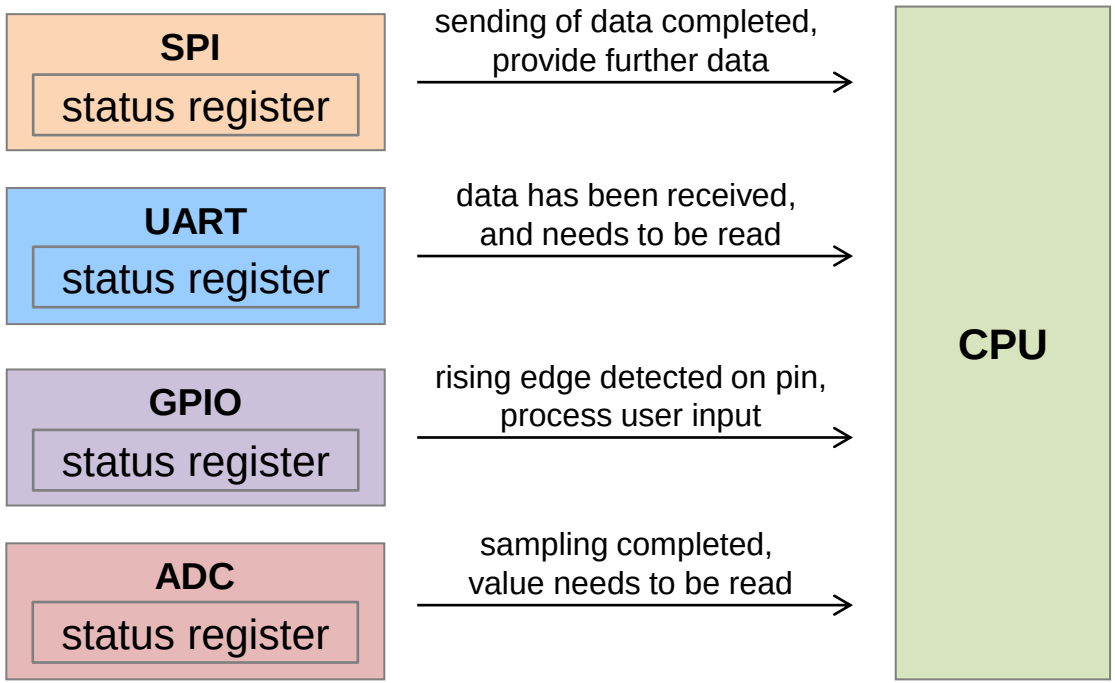
At the end of this lesson you will be able

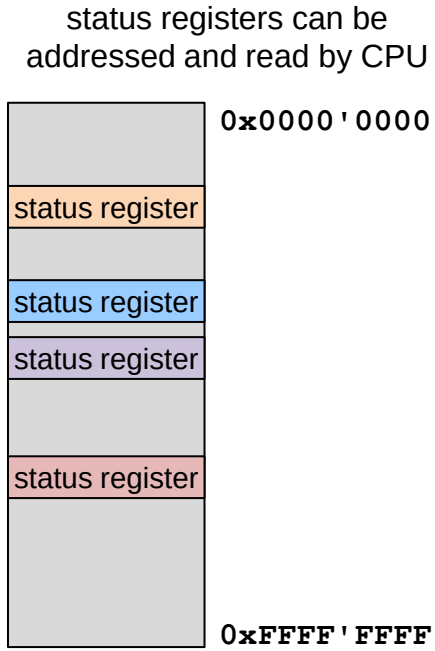
- to explain advantages and disadvantages of polling and interrupt-driven I/O
- to distinguish the different types of exceptions on a Cortex-M3/M4
- to explain how the Cortex-M3/M4 recognizes and processes exceptions
- to explain the vector table of the Cortex-M3/M4
- to understand the basic functionality of the Nested Vectored Interrupt Controller (NVIC)
  - to enable and disable interrupts
  - to set and clear interrupts by software
  - to prioritize exceptions
  - to know how programmed priorities influence preemption of service routines
  - to explain how simultaneously pending interrupts are processed
- to implement a simple interrupt service routine in Cortex-M assembly
- to explain potential data consistency issues due to interrupts and to give potential examples

CPU Has to Act on Events

School of Engineering
InES Institute of Embedded Systems

- **Peripherals signal events to CPU**
  - Something happened that requires a CPU service → call a subroutine



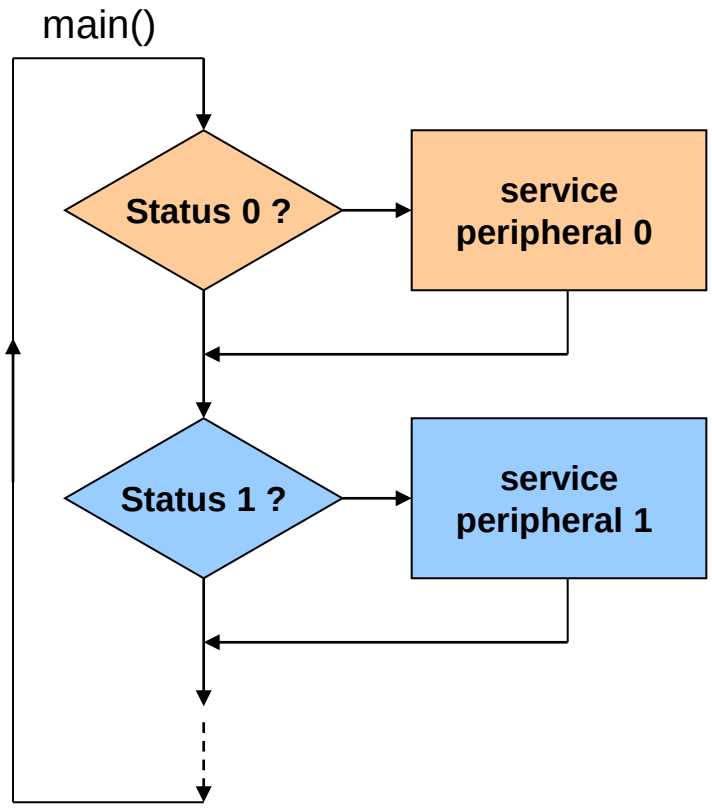


SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
GPIO	General Purpose Input Output
ADC	Analog-Digital Converter

Polling

- **Periodic Query of Status Information**
  - Reading of status registers in loop
  - Synchronous with main program
  - Advantages
    - Simple and straightforward
    - Implicit synchronization
    - Deterministic
    - No additional interrupt logic required
  - Disadvantages
    - Busy wait → wastes CPU time
    - Reduced throughput
    - Long reaction times

in case of many I/O devices or
if the CPU is working on other tasks



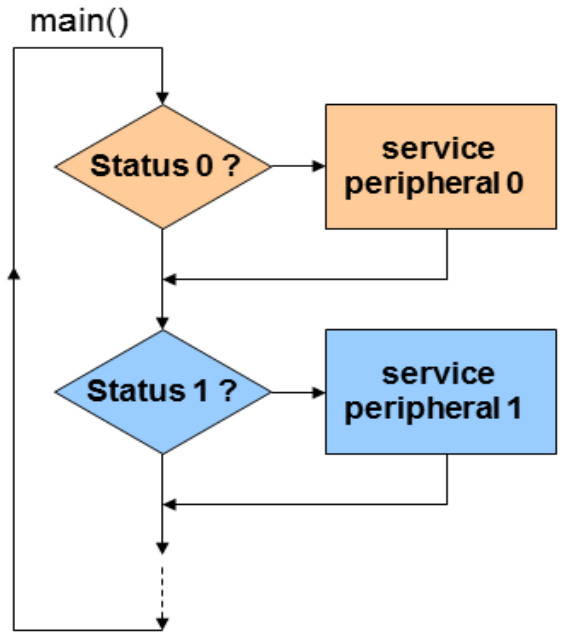
[1] to poll → abfragen

Polling

School of Engineering
InES Institute of Embedded Systems

Architecture

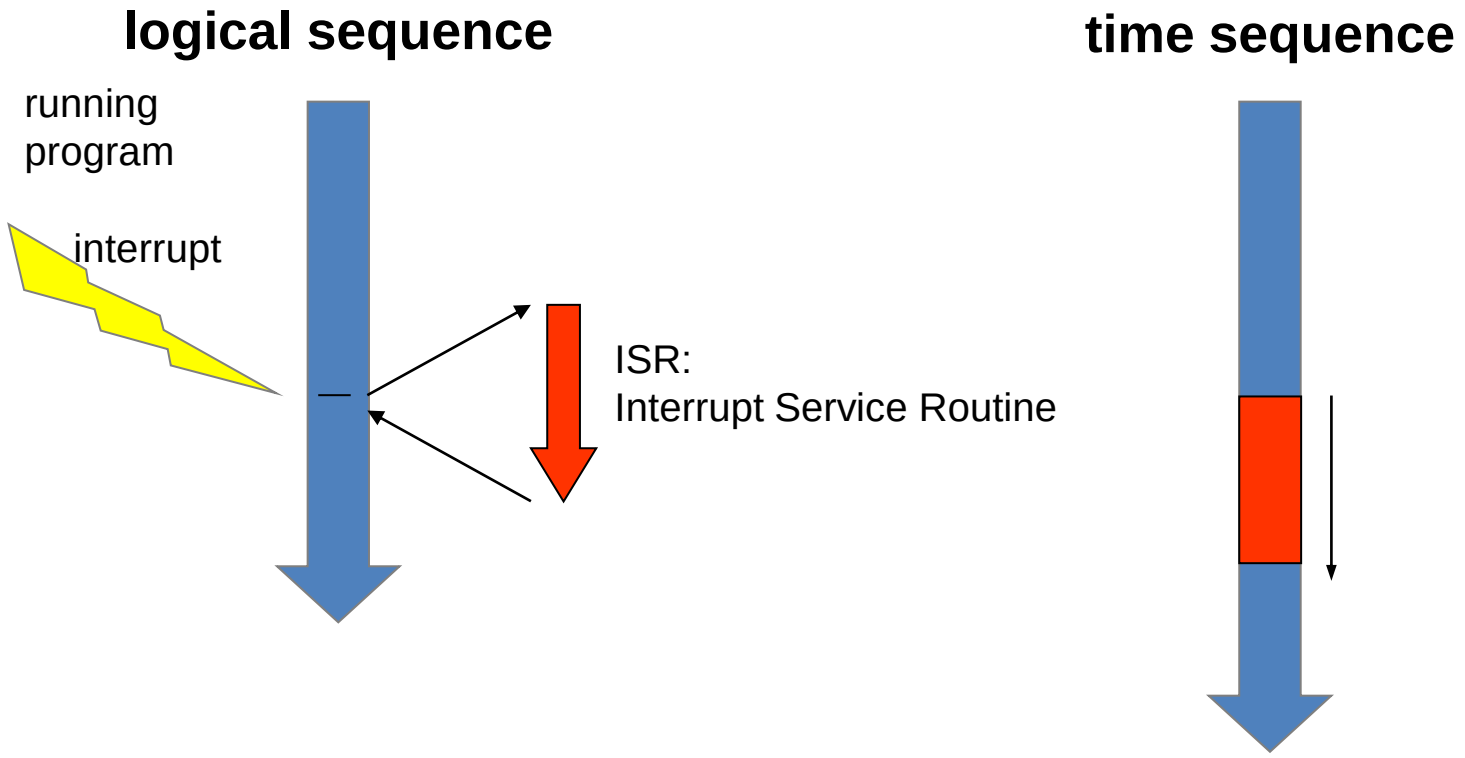
```
while(1)
    {
        if ( read_byte(ADDR_BUTTON_A) ) {
            execute_task_A();
        }

        if ( read_byte(ADDR_BUTTON_B) ) {
            execute_task_B();
        }

        if ( read_byte(ADDR_BUTTON_C) ) {
            execute_task_C();
        }

        …
    }
}
```



Interrupt-Driven I/O

- **Alternative to polling**

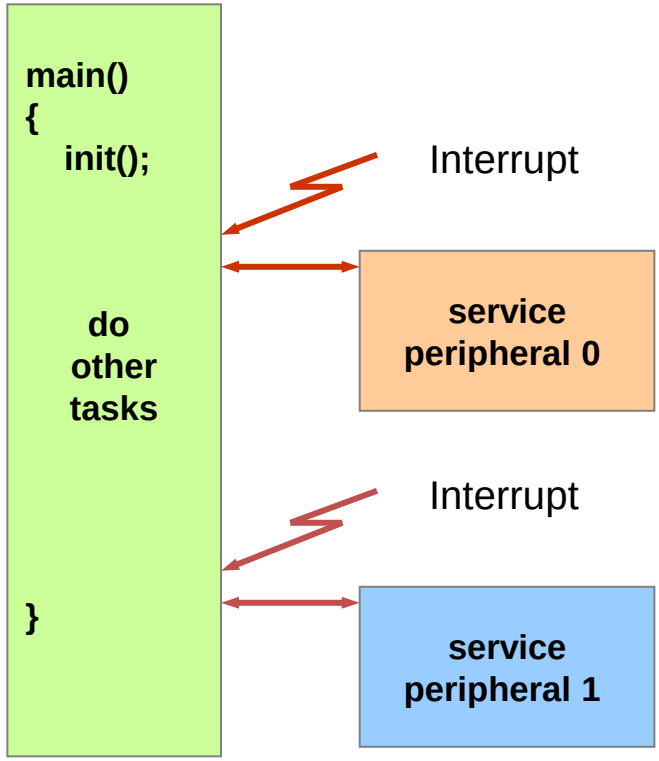
Interrupt: Sudden change of program flow due to an event



Interrupt-Driven I/O

School of Engineering
InES Institute of Embedded Systems

- **Main program**
  - Initializes peripherals
  - Afterwards it executes other tasks
  - Peripherals signal when they require software attention (phone call analogy)
  - Events interrupt program execution
- **Advantage**
  - No busy wait → better use of CPU time
  - Short reaction times
- **Disadvantages**
  - No synchronization (between main program and ISRs)
  - Difficult debugging



Exceptions Cortex-M3/M4

School of Engineering
InES Institute of Embedded Systems

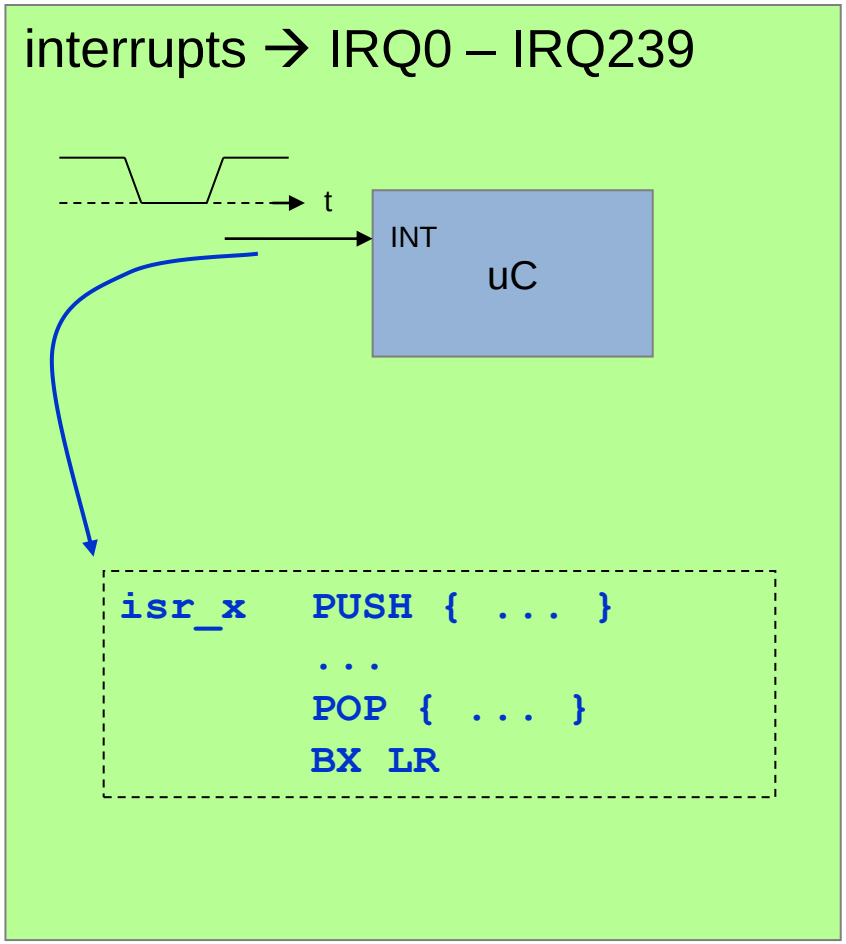
- **Interrupt sources: IRQ0 – IRQ239**
  - Peripherals[1)] signal to CPU that an event needs immediate attention
  - Can alternatively be generated by software request
  - Asynchronous to instruction execution
- **System exceptions**
  - Reset
    - Restart of processor
  - NMI – Non-maskable Interrupt
    - Condition that cannot be ignored, e.g. a critical hardware error
  - Faults
    - E.g. undefined instructions, unaligned accesses, etc.
  - System Level Calls
    - Operating system (OS) calls – Instruction SVC and PendSV

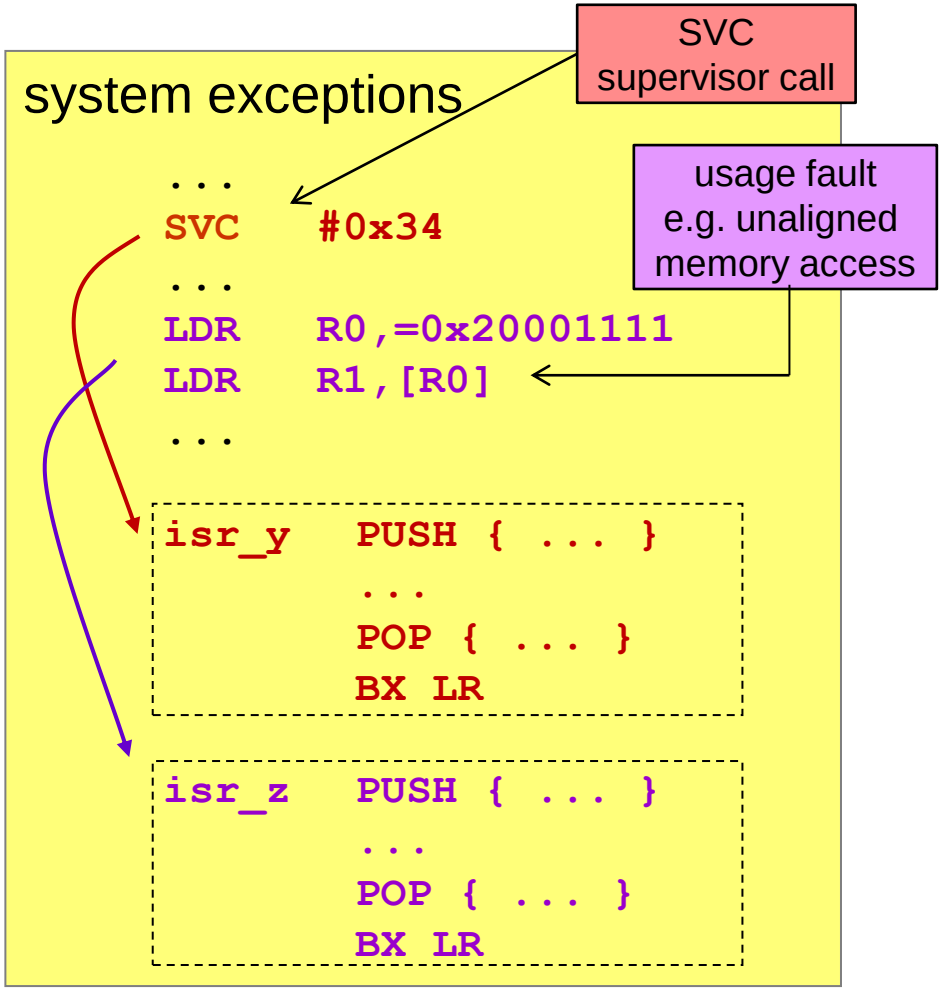
[1)] GPIO (general purpose input/output), timer, serial interfaces, etc

Exceptions Cortex-M3/M4

School of Engineering
InES Institute of Embedded Systems

Examples for Exceptions





Exceptions Cortex-M3/M4

School of Engineering
InES Institute of Embedded Systems

System Exceptions

Exception Number	Exception Type	Priority	Description
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Nonmaskable interrupt (external NMI input)
3	Hard Fault	−1	All fault conditions, if the corresponding fault handler is not enabled
4	MemManage Fault	Programmable	Memory management fault; MPU violation or access to illegal locations
5	Bus Fault	Programmable	Bus error; occurs when AHB interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access)
6	Usage Fault	Programmable	Exceptions due to program error or trying to access coprocessor (the Cortex-M3 does not support a coprocessor)
7–10	Reserved	NA	–
11	SVCall	Programmable	System Service call
12	Debug Monitor	Programmable	Debug monitor (breakpoints, watchpoints, or external debug requests)
13	Reserved	NA	–
14	PendSV	Programmable	Pendable request for system device
15	SYSTICK	Programmable	System Tick Timer

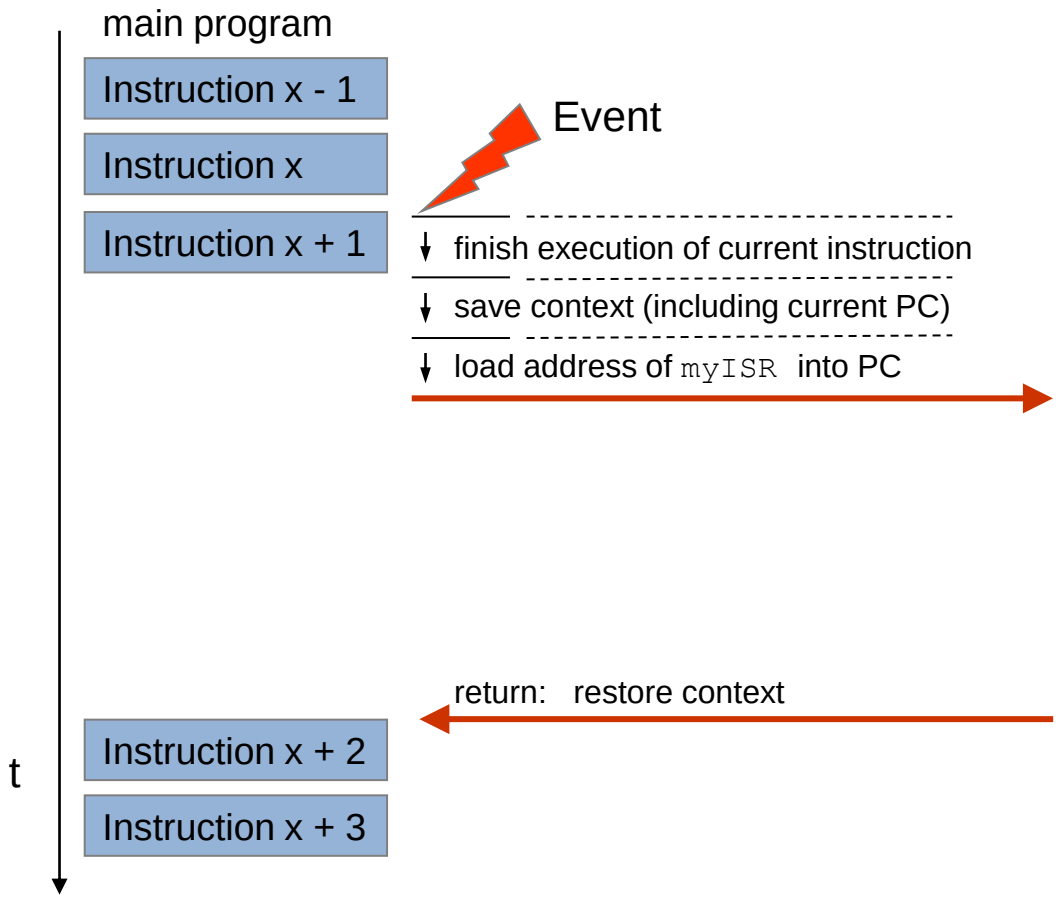
source: "The definitive Guide to the Cortex-M3"

Exceptions Cortex-M3/M4

School of Engineering
InES Institute of Embedded Systems

Interrupts: Event calls ISR

- Change of program flow

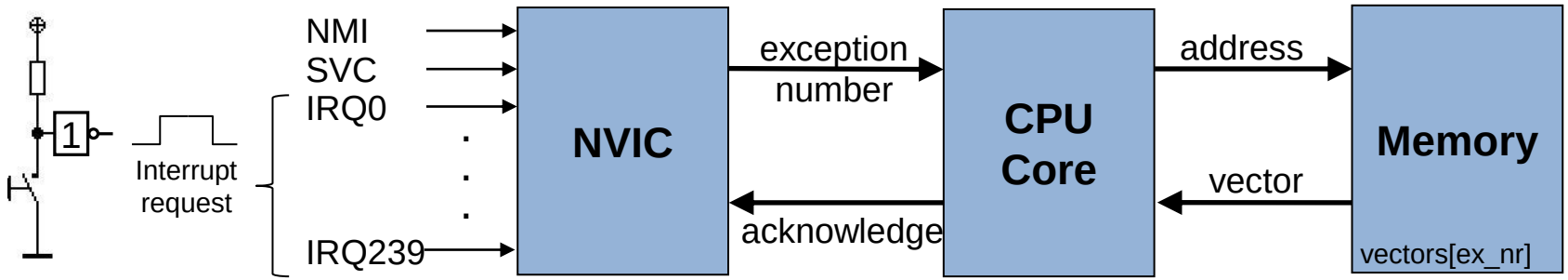



```
main    proc
prog    ...     ; main program
        ...
        B   prog

myISR
        ...     ; service the
        ...     ; event
        BX LR
```

Interrupt-System Cortex-M3/M4

School of Engineering
InES Institute of Embedded Systems

- **Nested Vectored Interrupt Controller (NVIC)**
  - 240 sources can trigger exception → high level signal on IRQx
  - Forwards respective exception number to CPU



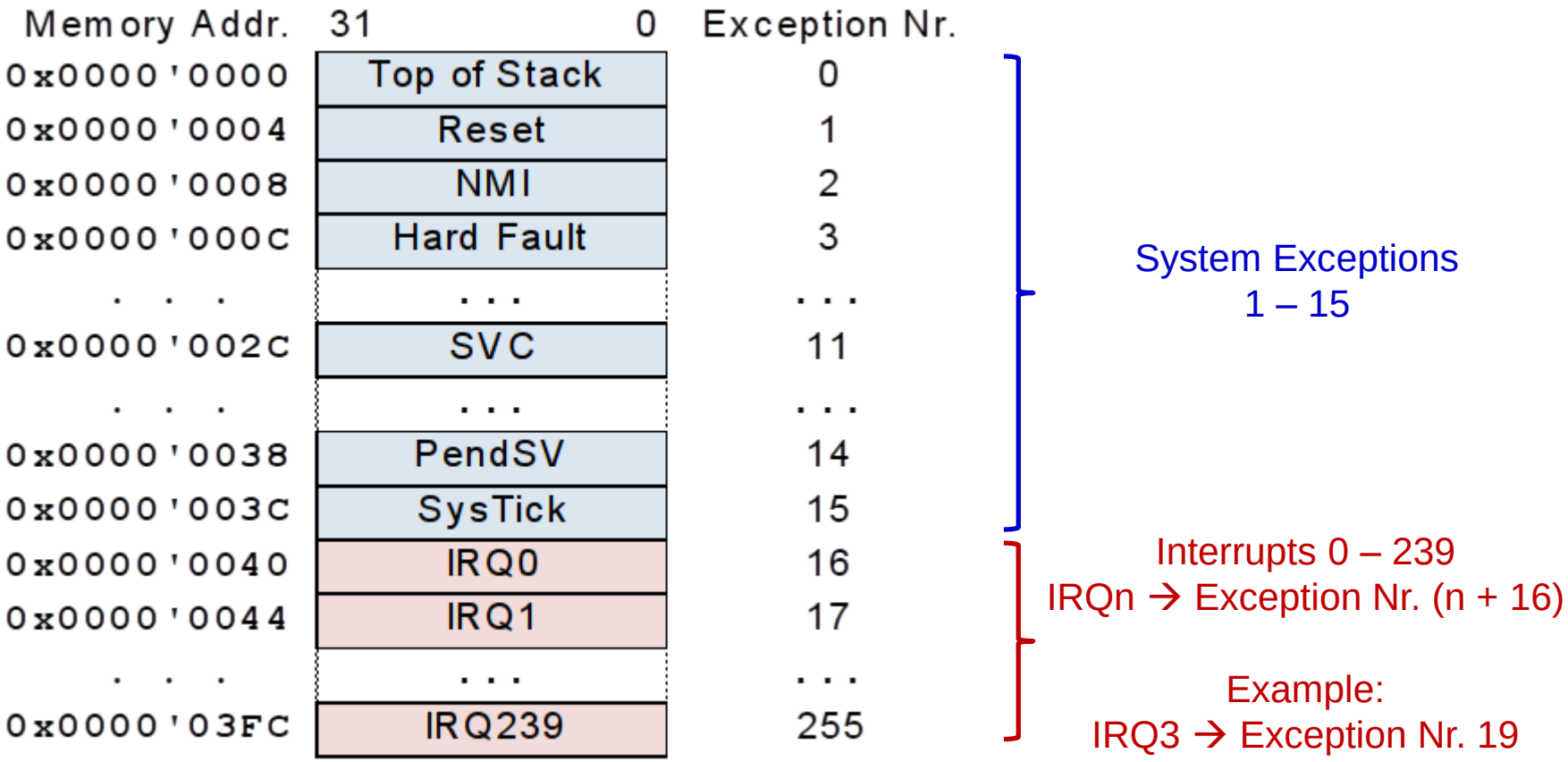
- **CPU**
  - Calculates vector table address based on exception number
  - Uses address to read vector from memory
  - Stores context on stack
  - Loads vector into PC → branch to ISR

Simultaneous exceptions from different sources will be covered later

Vector Table (Cortex-M3/M4)

School of Engineering
InES Institute of Embedded Systems

- **Which ISR Shall the Processor Call?**
  - Each exception has a different ISR



Vector Table (Cortex-M3/M4)

School of Engineering
InES Institute of Embedded Systems

- **Initialization**

```
                ; Vector Table Mapped to Address 0 at Reset
                AREA  RESET, DATA, READONLY

0   __Vectors   DCD   __initial_sp        ; Top of Stack
1               DCD   Reset_Handler       ; Reset Handler
2               DCD   NMI_Handler         ; NMI Handler
3               DCD   HardFault_Handler   ; Hard Fault Handler
…               DCD   ...
…               DCD   ...

                ; Interrupts
16              DCD   IRQ0_Handler        ; ISR for IRQ0
17              DCD   IRQ1_Handler        ; ISR for IRQ1
…               DCD   ...
```

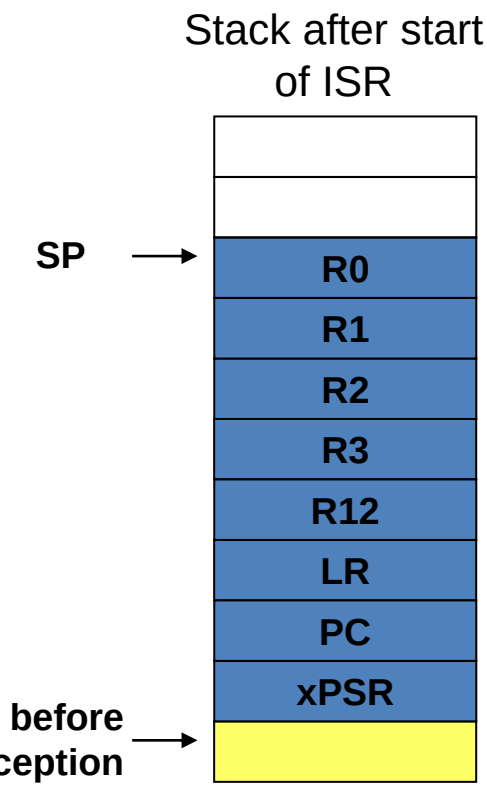
System Exceptions 15 vectors

Interrupts

```
                AREA  SOURCE_CODE, CODE, READONLY
IRQ0_Handler    PUSH { ... }

                ...          ; interrupt service

                POP { ... }
                BX LR
```

Storing the Context

- **Interrupt event can take place at any time**
  - E.g. between **`TST`** and **`BEQ`** instructions
    - ISR call requires automatic save of flags and caller saved registers
- **ISR Call**
  - Stores xPSR, PC, LR, R12, R0 – R3 on stack
  - Stores EXC_RETURN[1)] to LR
- **ISR Return**
  - Use **`BX LR`** or **`POP {…, PC}`**[2)]
  - Loading EXC_RETURN[1)] into PC
    - restores R0 – R3, R12, LR, PC and xPSR from stack



[1)] **`EXC_RETURN = 0xFFFF'FFF9`**

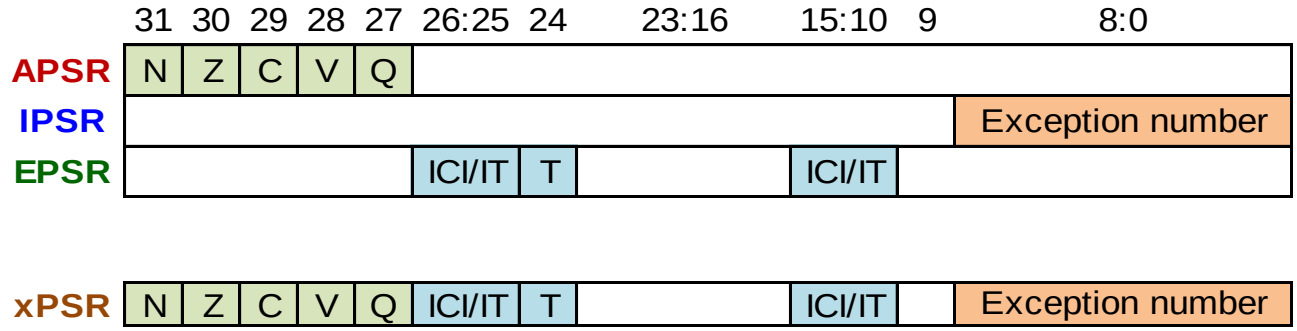
[2)] if LR has been saved with PUSH {..., LR} in ISR

Storing the Context

School of Engineering
InES Institute of Embedded Systems

Program Status Registers (PSRs)

- APSR Application Program Status Register
- IPSR Interrupt Program Status Register
- EPSR Execution Program Status Register

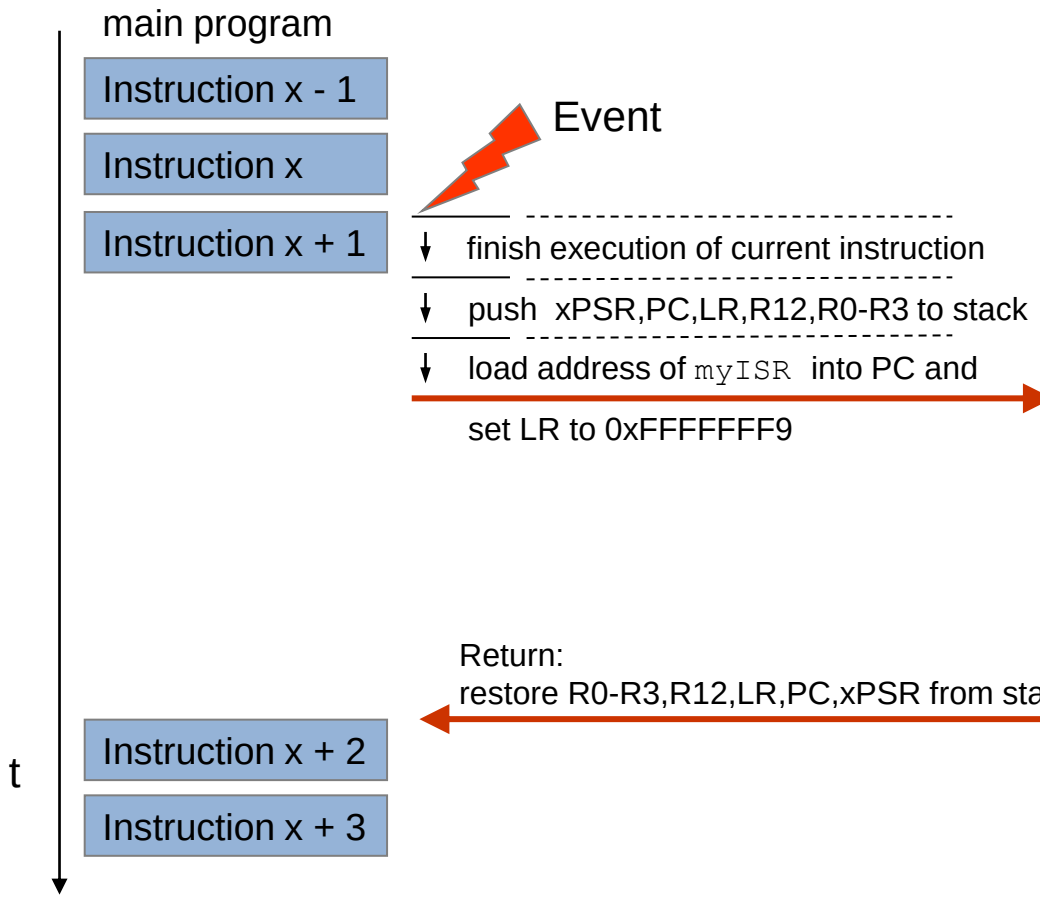


- xPSR Combination of all three PSRs

Exception number indicates which exception the processor is handling
ICI/IT Bits used in case of Interrupt-Continuable Instructions e.g. LDM/STM or conditional execution of instructions (IF-THEN)
T Thumb mode. Always one

Storing the Context

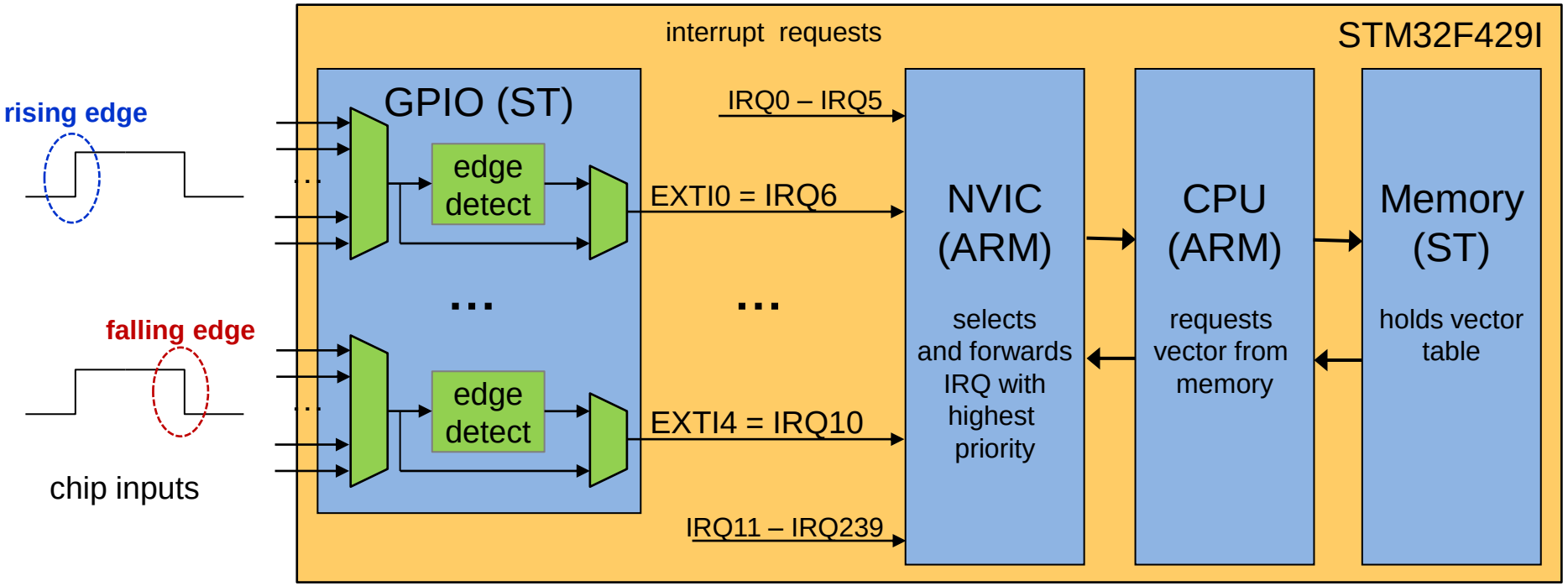
- **Asynchronous event calls ISR**




```
main    proc
prog    ...     ; main program
        ...
        B   prog

myISR
        ...     ; service the
        ...     ; event
        BX LR
```

External Interrupt Pins

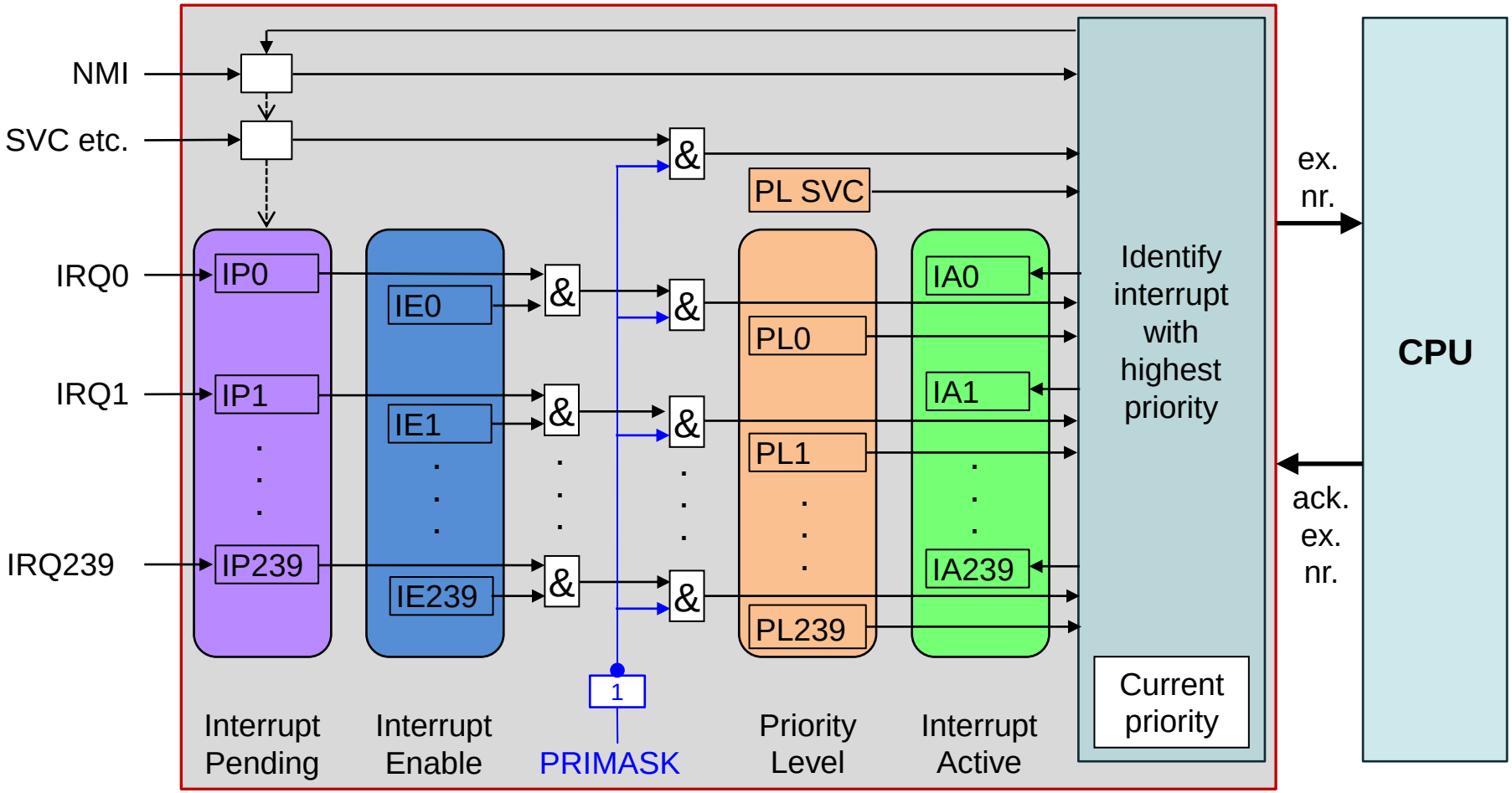
- **Chip Vendor (ST) adds GPIO logic**
  - EXTI0 through EXTI4 connected to IRQ6 through IRQ10
    - IRQ0 – IRQ5 / IRQ11 – IRQ239 used for other sources e.g. SPI, UART, ADC
  - Select chip input for each EXTIx line
  - Select level or edge



Interrupt Control

School of Engineering
InES Institute of Embedded Systems

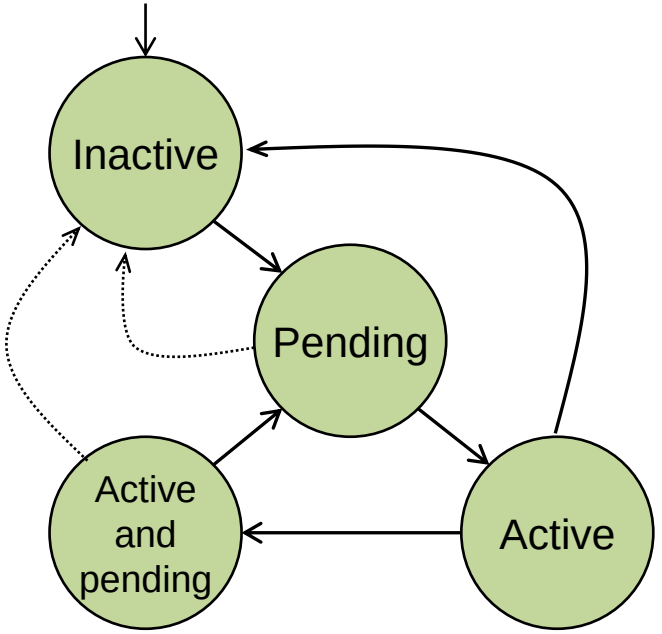
Nested Vectored Interrupt Controller (NVIC)



Interrupt Control

School of Engineering
InES Institute of Embedded Systems

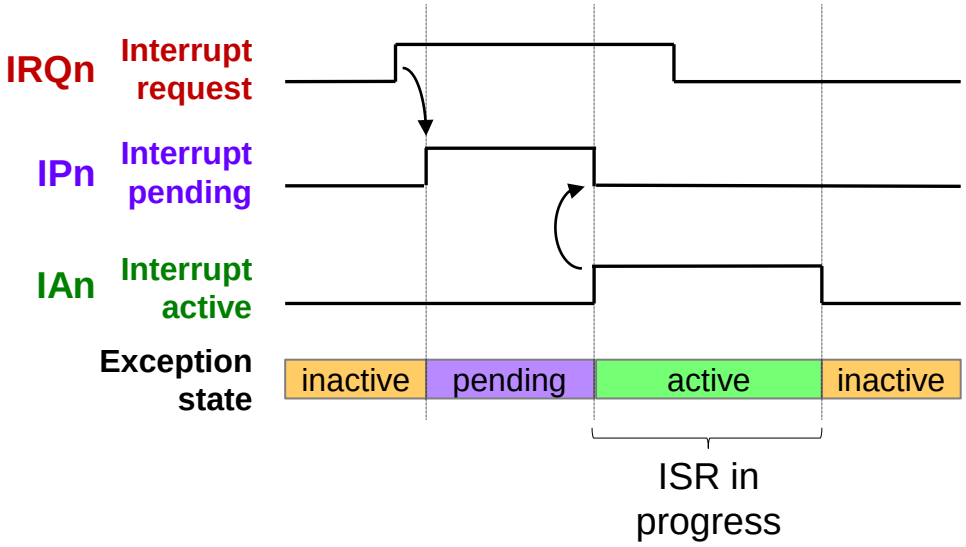
- **Exception States (for each interrupt source, e.g. IRQ17)**
  - Inactive
    - Exception is not active and not pending
  - Pending
    - Exception is waiting to be serviced by CPU
    - E.g. an interrupt event occurred (IRQn = 1) but interrupts are disabled (PRIMASK)
  - Active
    - Exception is being serviced by the CPU but has not completed
  - Active and pending
    - Exception is being serviced by the CPU and there is a pending exception from the same source



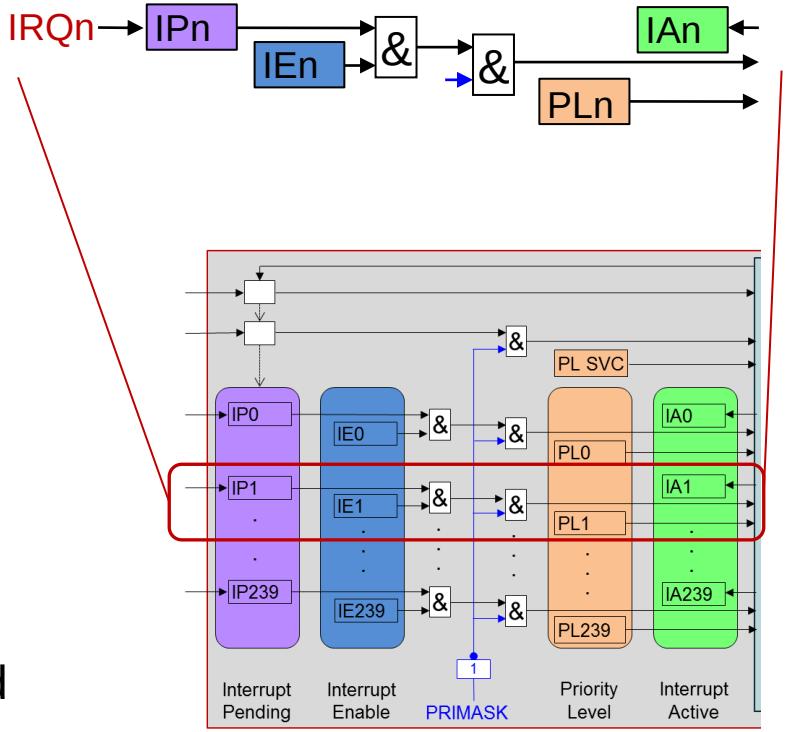
source: ARM Cortex-M3 Devices Generic User Guide

Interrupt Control

IRQ Inputs and Pending Behavior



Same logic exists for each interrupt n = 0 … 239

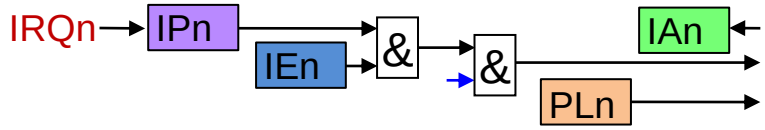


- High level on IRQn sets pending bit (IPn)
- Active Bit (IAn)
  - Set as soon as exception is being serviced
  - Resets pending bit

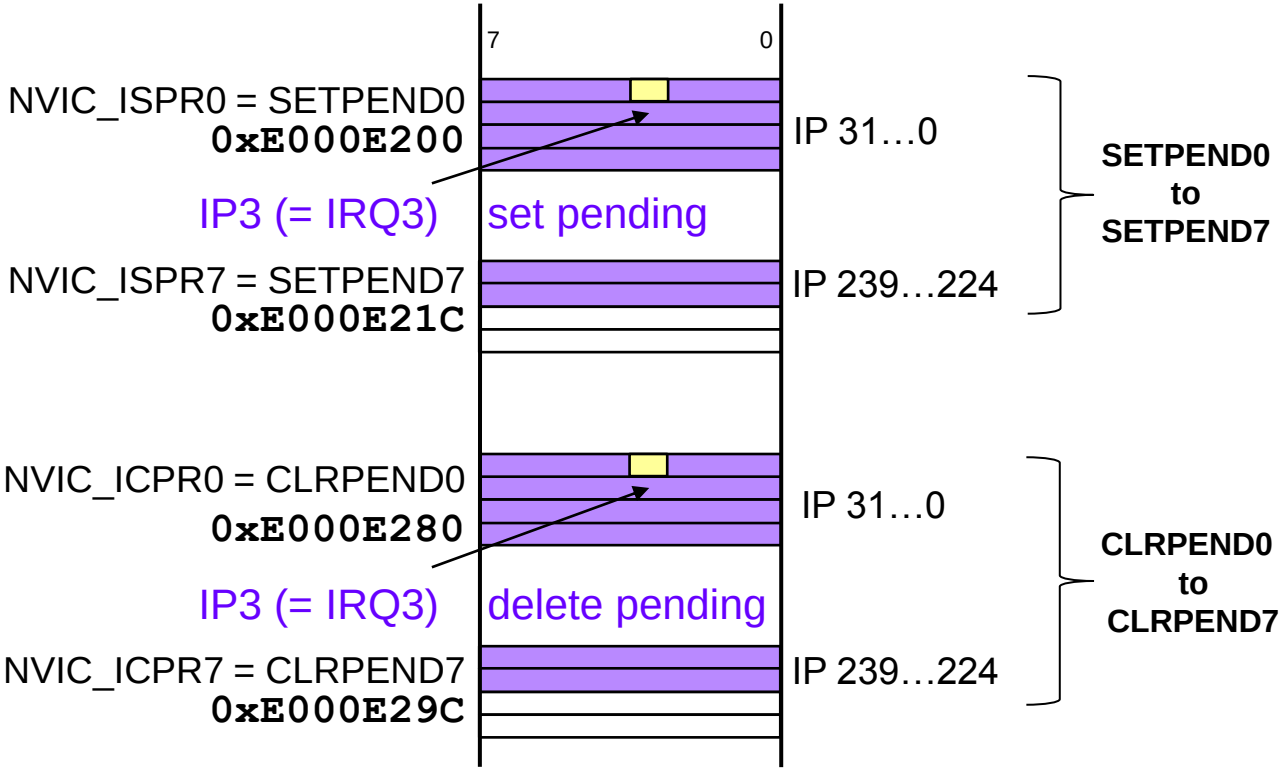
[1] IRQn is set by hardware and usually reset by software

Interrupt Control

School of Engineering
InES Institute of Embedded Systems



- **Interrupt Pending Registers**
  - Trigger hardware interrupt by software → set pending bit
  - Cancel a pending interrupt → clear pending bit



Trigger IRQ3 by Software

```
SETPEND0   EQU 0xE000E200
           ...

           LDR  R0,=SETPEND0
           MOVS R1,#0x08
           STR  R1,[R0]
```

Delete Pending IRQ3

```
CLRPEND0 EQU 0xE000E280
         ...

         LDR  R0,=CLRPEND0
         MOVS      R1,#0x08
         STR  R1,[R0]
```

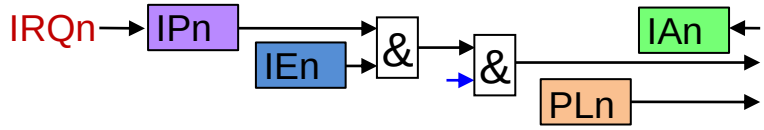
ARM and ST use different names for the same register

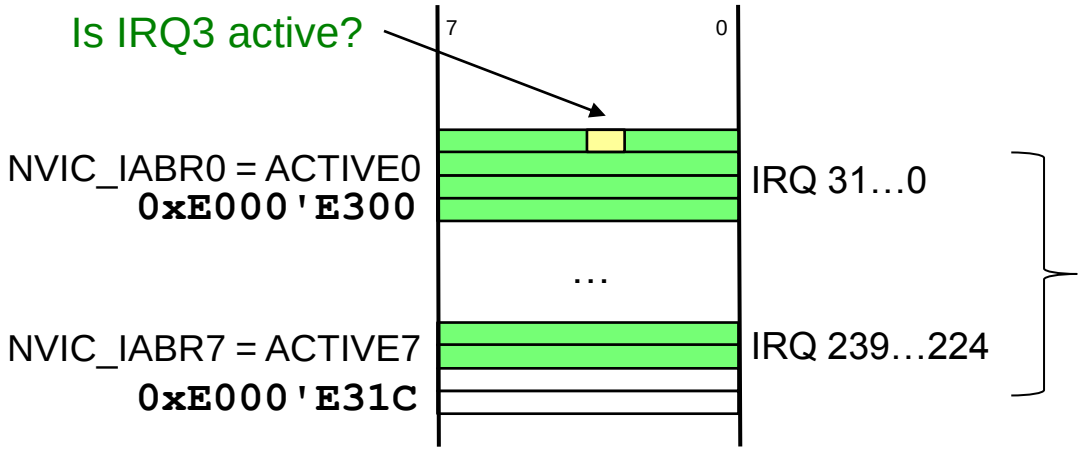
Reading CLRPENDx by software will return the pending interrupts

Interrupt Control

Interrupt Active Status Registers

- Read-only
- Corresponding bit is set when ISR starts
- Corresponding bit is cleared when interrupt return is executed





Test if IRQ3 is active

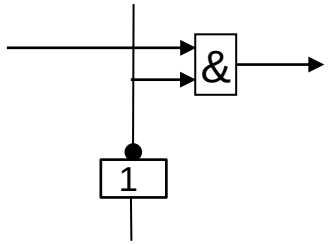
```
ACTIVE0 EQU 0xE000E300
        ...

        LDR  R0, =ACTIVE0
        MOVS R1, #0x08
        LDR  R2, [R0]
        TST  R1, R2
        BEQ  ...
```

Interrupt Control

School of Engineering
InES Institute of Embedded Systems

General Masking of Interrupts

- PRIMASK
  - Single bit controlling all maskable interrupts



  - Disable set PRIMASK
  - Enable clear PRIMASK

Assembly	C
`CPSID`[1] `i`	`__disable_irq();`
`CPSIE`[1] `i`	`__enable_irq();`

- On reset PRIMASK = 0 → enabled

Non-maskable Interrupt (NMI)

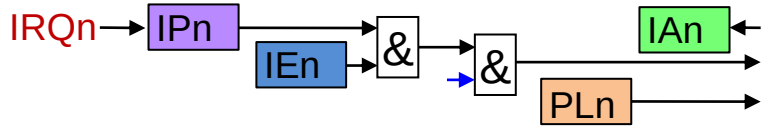
- Power-fail, emergency button, watchdog, ...

[1] `CPSIE/D` = **C**hange **P**rocessor **S**tate **I**nterrupt **E**nable **/ D**isable

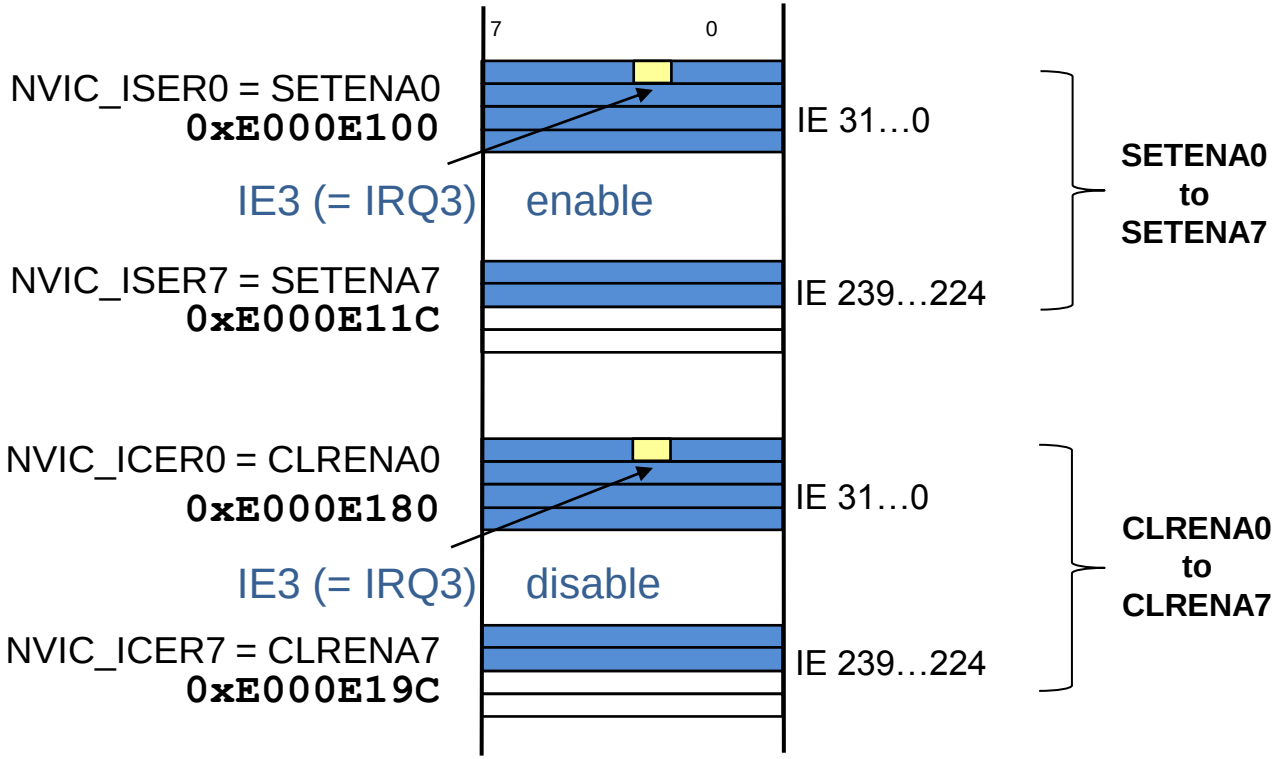
Interrupt Control

Interrupt Enable Registers

- Individual masking of interrupt sources
  - IEn cleared pending bit not forwarded
  - IEn set interrupt enabled



CLRENA and SETENA registers can be read by software and will return the settings of the IE bits



Enable IRQ3

```
SETENA0 EQU   0xE000E100
        ...

        LDR   R0, =SETENA0
        MOVS  R1, #0x08
        STR   R1, [R0]
```

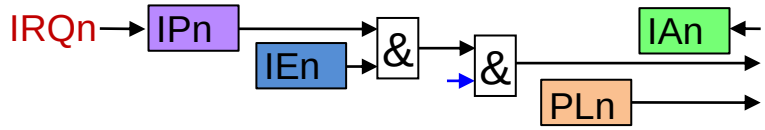
Disable IRQ3

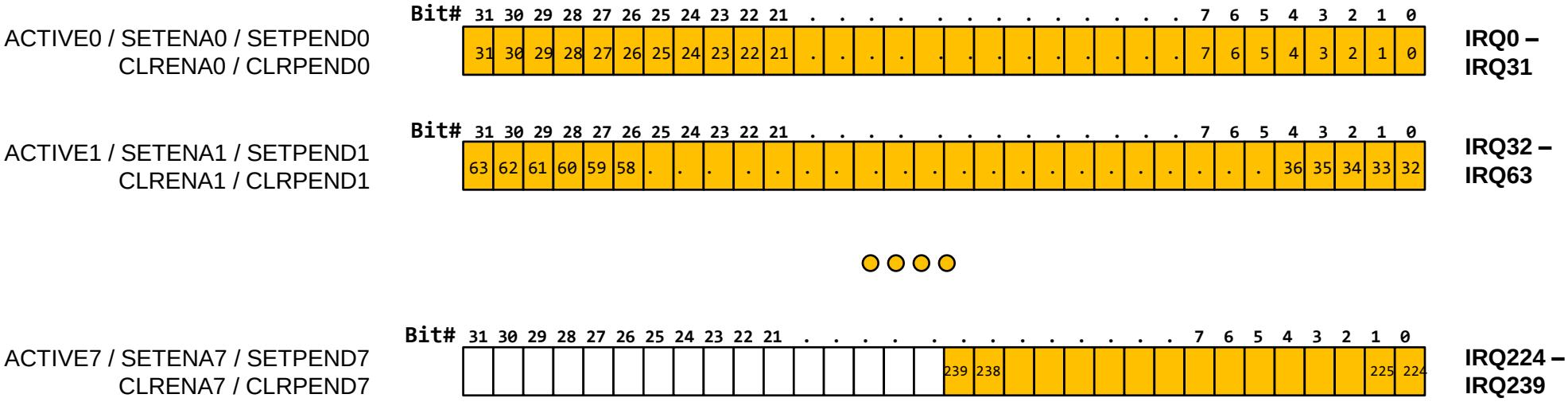
```
CLRENA0 EQU   0xE000E180
        ...

        LDR   R0, =CLRENA0
        MOVS  R1, #0x08
        STR   R1, [R0]
```

Interrupt Control – Register View

School of Engineering
InES Institute of Embedded Systems

- **1 Bit for each IRQ**
  - Registers are written as 32-bit words
  - Only bits with value “1” have an effect, “0” is ignored
  - Separate registers to read and write

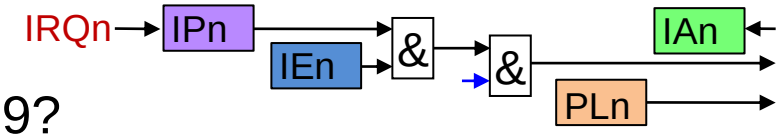




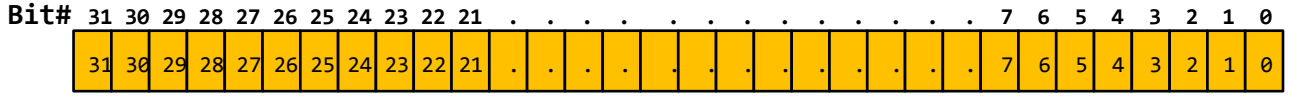
Interrupt Control – Register View

School of Engineering
InES Institute of Embedded Systems

- **Exercise:**
  - Enable IRQ 46 and clear pending IRQ 129?

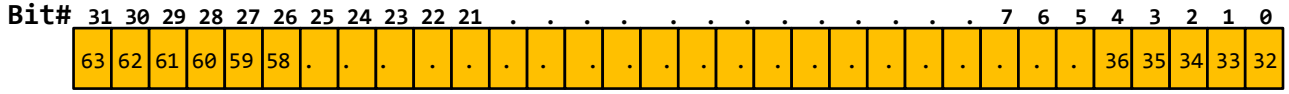


ACTIVE0 / SETENA0 / SETPEND0
CLRENA0 / CLRPEND0



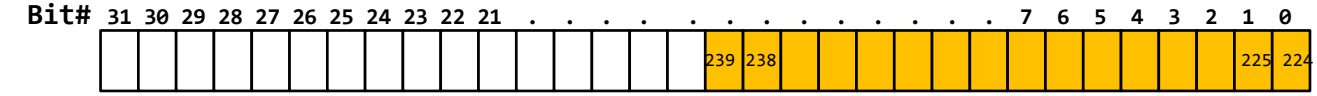
IRQ0 – IRQ31

ACTIVE1 / SETENA1 / SETPEND1
CLRENA1 / CLRPEND1



IRQ32 – IRQ63

ACTIVE7 / SETENA7 / SETPEND7
CLRENA7 / CLRPEND7

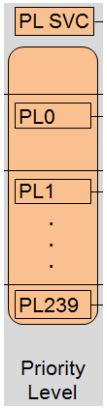


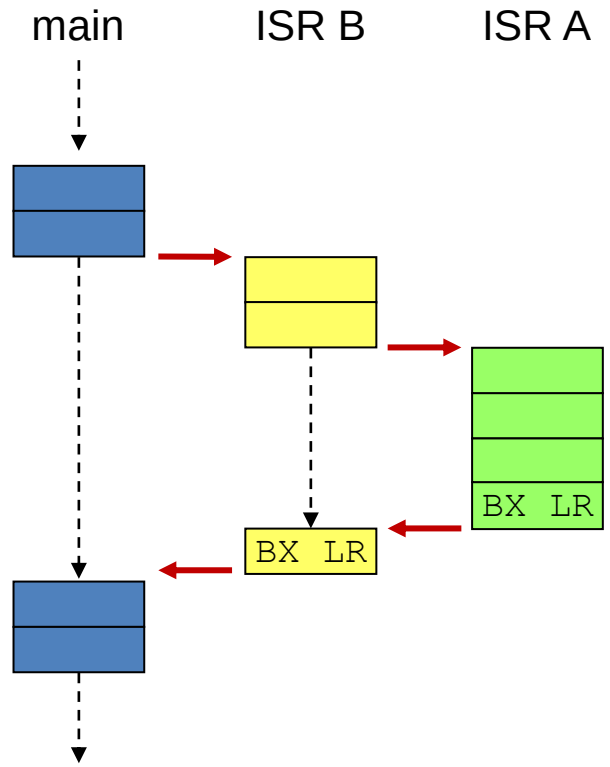
IRQ224 – IRQ239

Nested Exceptions

School of Engineering
InES Institute of Embedded Systems

- **Preemption**
  - Service routine A temporarily interrupts service routine B
  - Assigned priority level for each exception
    - Levels define whether A can preempt B
  - Fixed priorities
    - Reset (-3), NMI (-2), hard fault (-1)
  - All other priorities are programmable





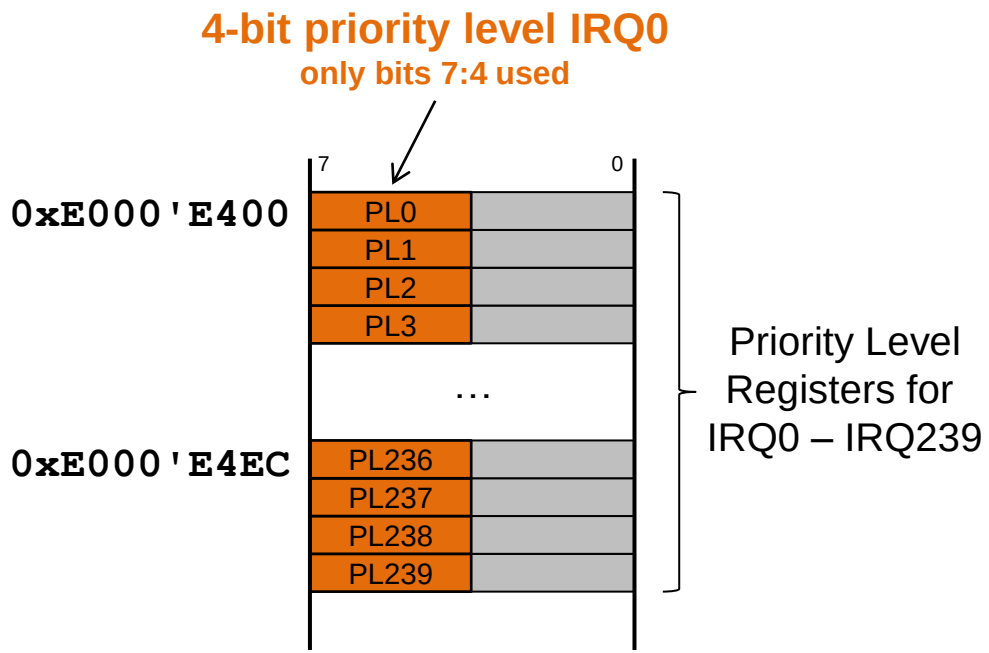
if A has been programmed to higher priority than B

Nested Exceptions

School of Engineering
InES Institute of Embedded Systems

Priority Level Registers

- The lower a priority level, the greater the priority
- 4-bit priority level [1)] 0x0 – 0xF



Set priority of IRQ3 to 5

```
PL_IRQ3   EQU   0xE000E403
          ...

          LDR   R0, =PL_REG_IRQ3
          MOVS  R1, #0x50
          STRB  R1, [R0]
```

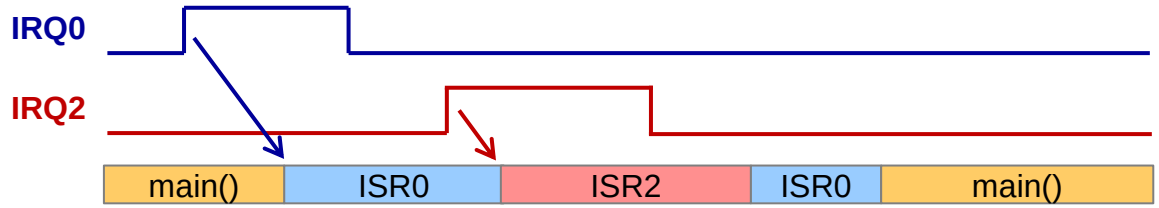
priority level registers for system exceptions
0xE000'ED18 – 0xE000'ED23

1) Cortex-M3/4 can handle up to 8 bits but STM uses only 4
Registers are called NVIC_IPRx on ST

Nested Exceptions

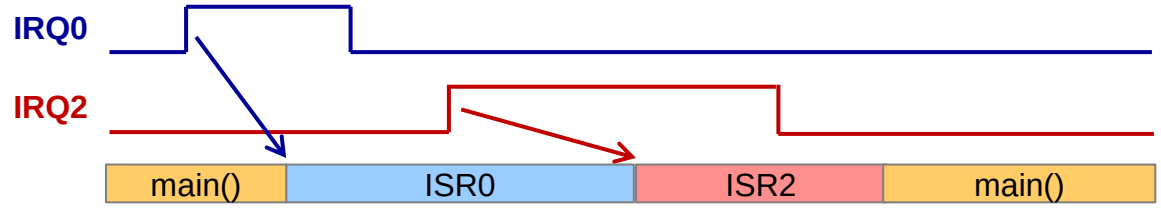
School of Engineering
InES Institute of Embedded Systems

- **IRQ request while other ISR is running**
  - ISR0 is executing (triggered by IRQ0)
  - Interrupt request IRQ2 arrives
  - PL2 < PL0 → IRQ2 has higher priority than IRQ0



ISR2 preempts ISR0

  - PL2 >= PL0 → IRQ2 has lower or same priority than IRQ0



ISR2 has to wait until ISR0 completes

Nested Exceptions

School of Engineering
InES Institute of Embedded Systems

- **Two or more pending interrupts**
  - (1) IRQ requests arrive simultaneously
  - (2) More than one request has been waiting for a higher priority ISR to complete

  - NVIC will evaluate the priorities of the pending interrupts
    - Selects IRQx with highest priority i.e.
      - ► with lowest priority level value (PL)
      - ► with lowest number (in case of identical PL values)
    - Other interrupts remain pending

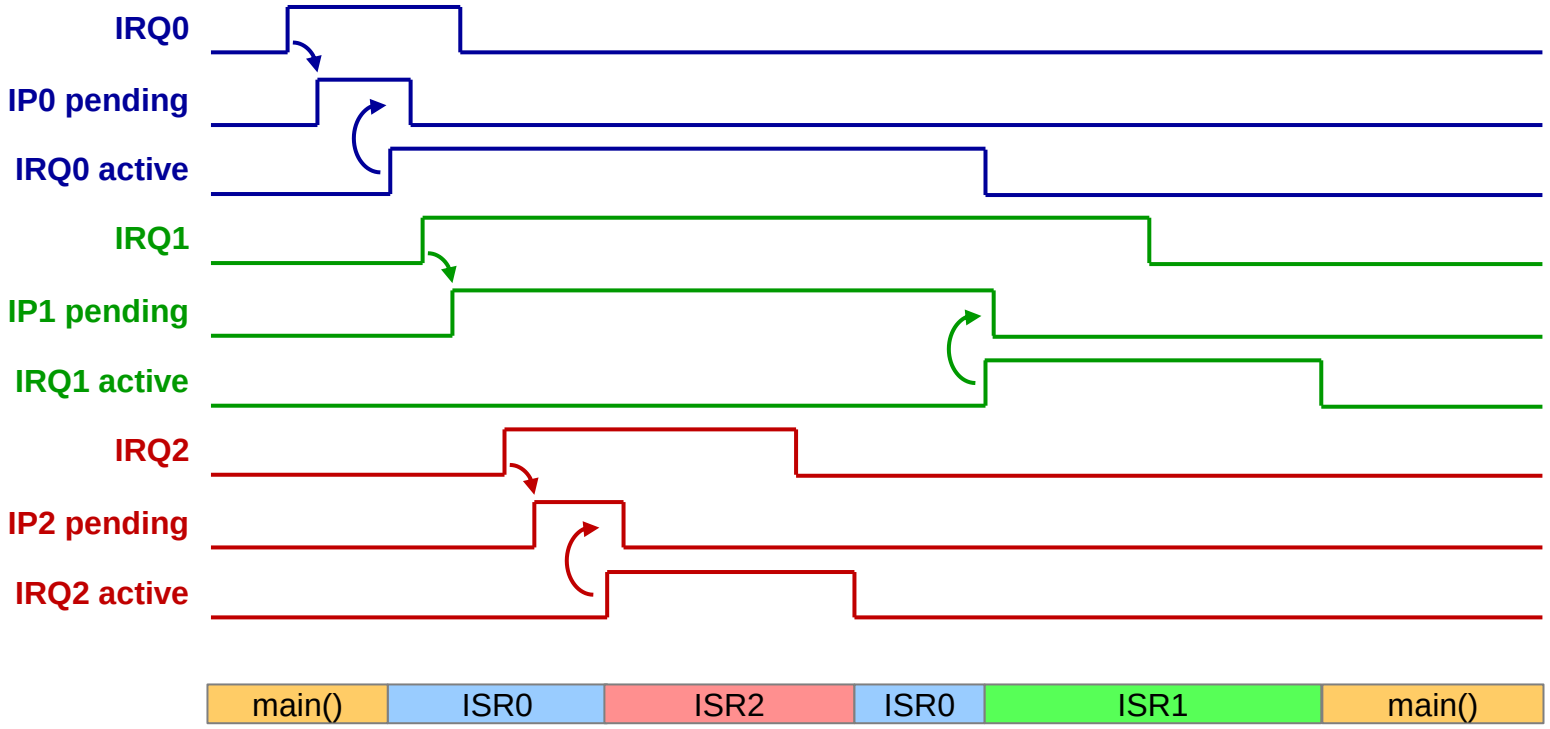
Nested Exceptions

School of Engineering
InES Institute of Embedded Systems

- **Example Priorities**
  - ISR1 does not preempt ISR0
  - ISR2 preempts ISR0

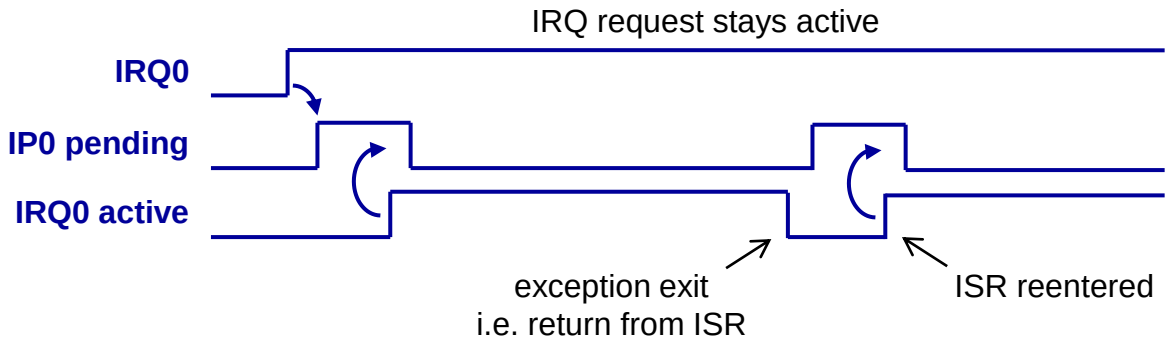
assuming

IRQ0	PL0 = 0x2	medium priority
IRQ1	PL1 = 0x3	lowest priority
IRQ2	PL2 = 0x1	highest priority



Special Interrupt Situations

- **IRQ request stays active**
  - Interrupt becomes pending again

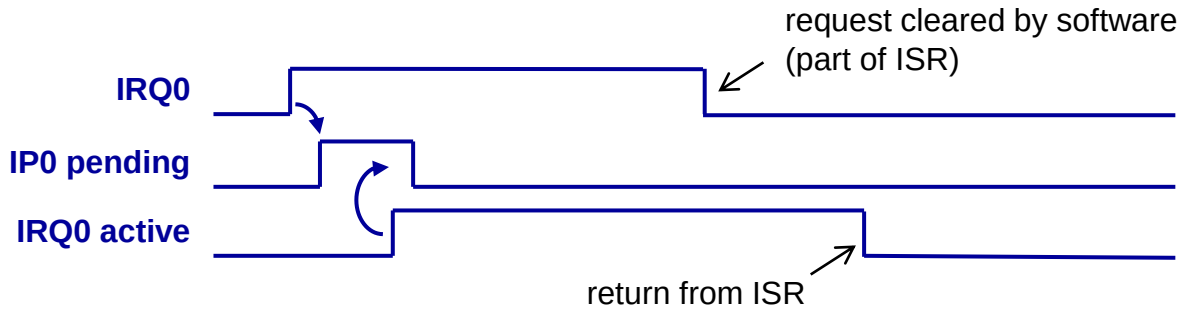


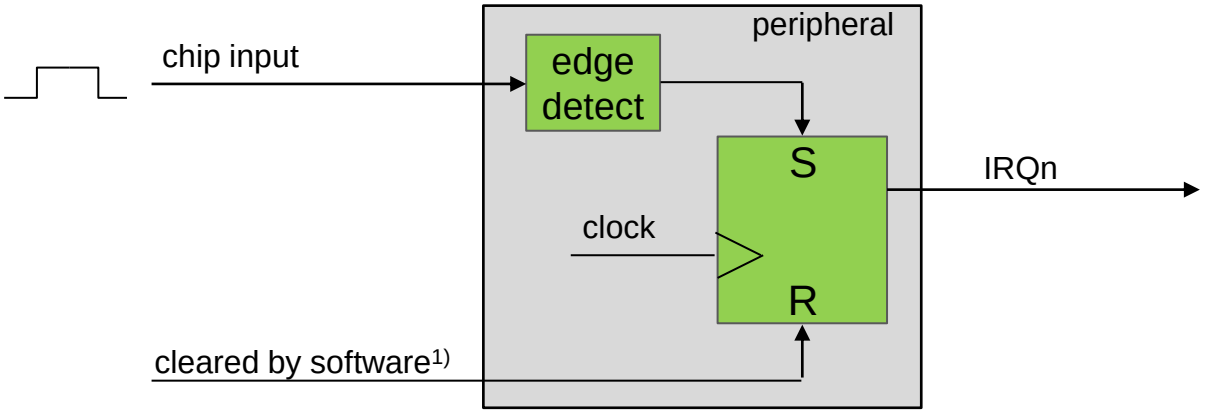
Special Interrupt Situations

School of Engineering
InES Institute of Embedded Systems

Common Configuration on Microcontrollers

- IRQ Set by Hardware – Cleared by Software

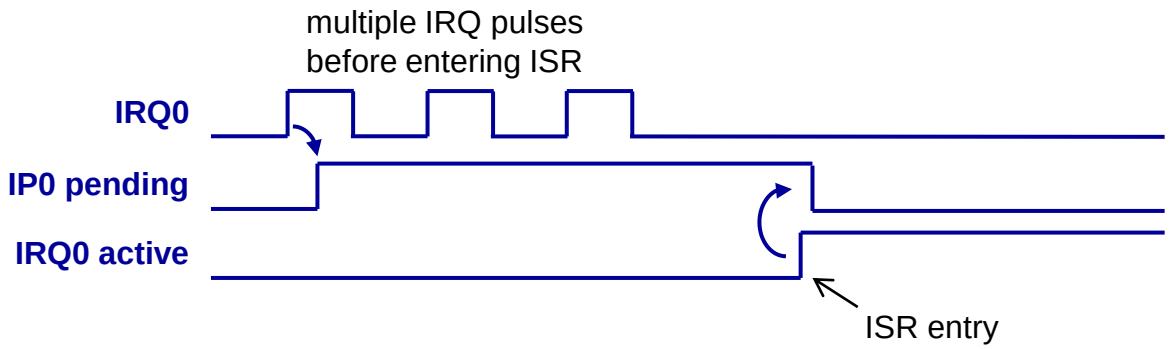




1) The software clears the bit at the correct position at the defined register address

Special Interrupt Situations

- **Multiple IRQ request pulses before entering ISR**
  - Treated as single interrupt
  - Events are lost



CMSIS

School of Engineering
InES Institute of Embedded Systems

- **What is CMSIS?**
  - Cortex Microcontroller Software Interface Standard
  - Vendor-independent hardware abstraction layer for Cortex-M
  - Defines generic tool interfaces
  - ISO/IEC C code cannot directly access some Cortex-M3 instructions
    - intrinsic functions required

CMSIS Functions

School of Engineering
InES Institute of Embedded Systems

NVIC Control

`void NVIC_EnableIRQ(IRQn_t IRQn)`	Enable IRQn
`void NVIC_DisableIRQ(IRQn_t IRQn)`	Disable IRQn
`uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)`	Return true (IRQ-Number) if IRQn is pending
`void NVIC_SetPendingIRQ (IRQn_t IRQn)`	Set IRQn pending
`void NVIC_ClearPendingIRQ (IRQn_t IRQn)`	Clear IRQn pending status
`uint32_t NVIC_GetActive (IRQn_t IRQn)`	Return '1' if active bit of IRQn is set. '0' otherwise
`void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)`	Set priority for IRQn
`uint32_t NVIC_GetPriority (IRQn_t IRQn)`	Read priority of IRQn
`void NVIC_SystemReset (void)`	Reset the system

Animation

Data Consistency

School of Engineering
InES Institute of Embedded Systems

Example

```
typedef struct  {
    uint8_t minutes;
    uint8_t seconds;
} time_t;

static time_t time = { 0, 0 };

int main(void)
{
  while (1) {

    write_byte(ADDR_LED_7_0,
                  time.seconds);
    write_byte(ADDR_LED_15_8,
                  time.minutes);

  }
}
```

```
void IRQ1_Handler(void)
{
  time.seconds++;
  if (time.seconds > 59) {
    time.seconds = 0;
    time.minutes++;
    }
}
```

Animation

Data Consistency

School of Engineering
InES Institute of Embedded Systems

- **Example**

```
typedef struct  {
    uint8_t minutes;
    uint8_t seconds;
} time_t;

static time_t time = { 0, 0 };

int main(void)
{
  while (1) {

    write_byte(ADDR_LED_7_0,
                  time.seconds);
    write_byte(ADDR_LED_15_8,
                  time.minutes);

  }
}
```

```
void IRQ1_Handler(void)
{
  time.seconds++;
  if (time.seconds > 59) {
    time.seconds = 0;
    time.minutes++;
    }
}
```

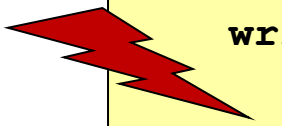
Animation

Data Consistency

School of Engineering
InES Institute of Embedded Systems

Example

```
typedef struct  {
    uint8_t minutes;
    uint8_t seconds;
} time_t;

static time_t time = { 0, 0 };

int main(void)
{
  while (1) {

    write_byte(ADDR_LED_7_0,
                    time.seconds);
    write_byte(ADDR_LED_15_8,
                    time.minutes);

  }
}
```



```
void IRQ1_Handler(void)
{
  time.seconds++;
  if (time.seconds > 59) {
    time.seconds = 0;
    time.minutes++;
    }
}
```

time = { 15, 59 }

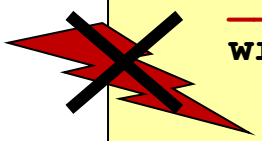
1) Output 59 → LED_7_0
2) Interrupt→ time = { 16, 0 }
3) Output 16 → LED_15_8

→ display: 16 59 !!!

Data Consistency

Example

```
typedef struct  {
    uint8_t minutes;
    uint8_t seconds;
} time_t;

static time_t time = { 0, 0 };

int main(void)
{
  while (1) {
    __disable_irq();
    write_byte(ADDR_LED_7_0,
                    time.seconds);
    write_byte(ADDR_LED_15_8,
                    time.minutes);
    __enable_irq();
  }
}
```



```
void IRQ1_Handler(void)
{
  time.seconds++;
  if (time.seconds > 59) {
    time.seconds = 0;
    time.minutes++;
    }
}
```

time = { 15, 59 }
1) Output 59 → LED_7_0
2) Interrupt→ time = { 16, 0 }
3) Output 16 → LED_15_8
→ display: 16 59 !!!

Data Consistency

School of Engineering
InES Institute of Embedded Systems

- **Data structure must not be changed during output**
- **Disable Interrupts during output**
- **Multitasking problem**

Conclusions

School of Engineering
InES Institute of Embedded Systems

- **Polling vs. Interrupt-driven I/O**
- **Exceptions**
  - System exceptions: Reset, NMI, Faults, System Level Calls
  - Interrupts IRQ0 – IRQ239
- **Nested Vectored Interrupt Controller**
  - Vector Table holds addresses of interrupt service routines (ISR)
  - Save context including xPSR
  - Masking of interrupts
  - Pending and active bits
  - Priority levels
  - IRQn: Often hardware set; software reset
- **Data Consistency Issues**