# EIE3105: Interrupt Programming (Chapter 11)

Dr. Lawrence Cheung

Semester 1, 2020/21

# Topics

- Polling vs. interrupt
- Interrupt unit
- Steps in executing an interrupt
- Timer interrupt
- Edge-triggered vs. level-triggered interrupts
- Interrupt priority
- Interrupt inside an interrupt

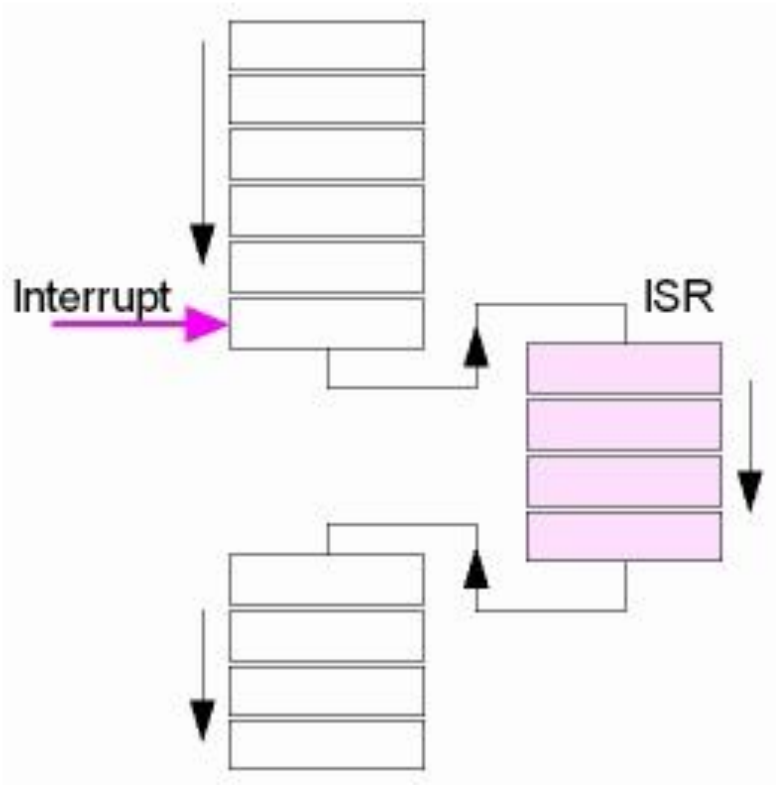
# Polling vs. Interrupt

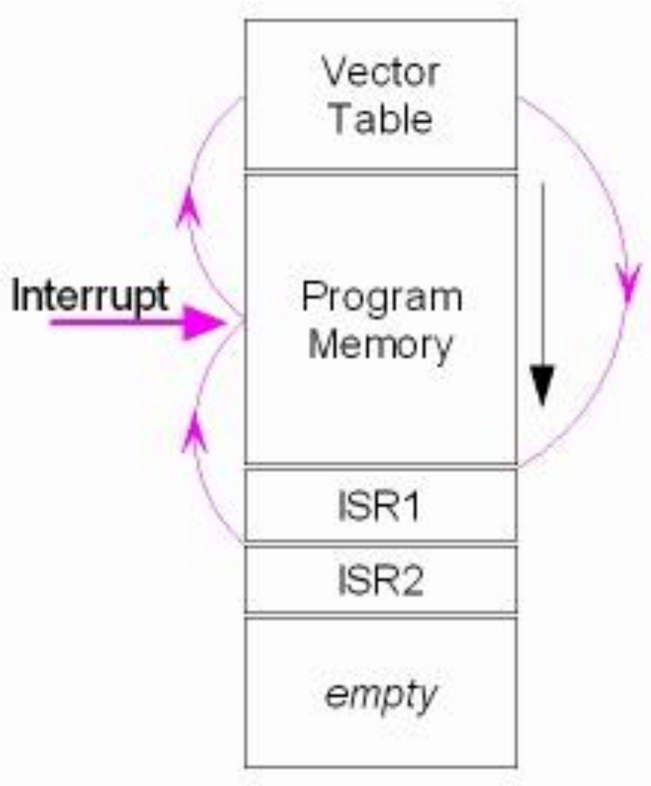
- Polling
  - The microcontroller continuously monitors the status of a given device.
  - When the condition is met, it performs the device.
  - After that, it moves on to monitor the next device until every one is serviced.
  - The microcontroller checks all devices in a round-robin fashion.

# Polling vs. Interrupt

- Interrupt
  - Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal.
  - Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device.
  - The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler.

# Polling vs. Interrupt





# Polling vs. Interrupt

- Polling
  - Ties down the CPU

```
while (true)
{
    if(PIND.2 == 0)
            //do something;
}
```

- Interrupt
  - Efficient CPU use
  - Has priority
  - Can be masked

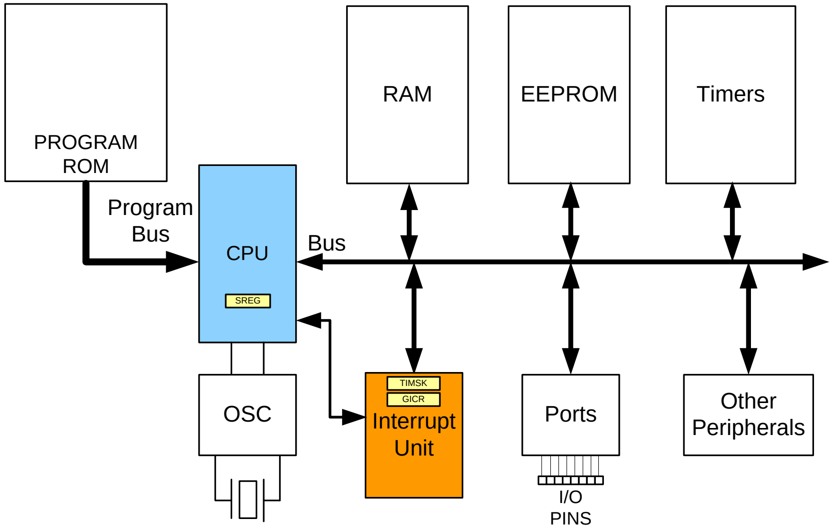
```
main( )
{
    Do your common task
}

whenever PIND.2 is 0 then
   do something
```

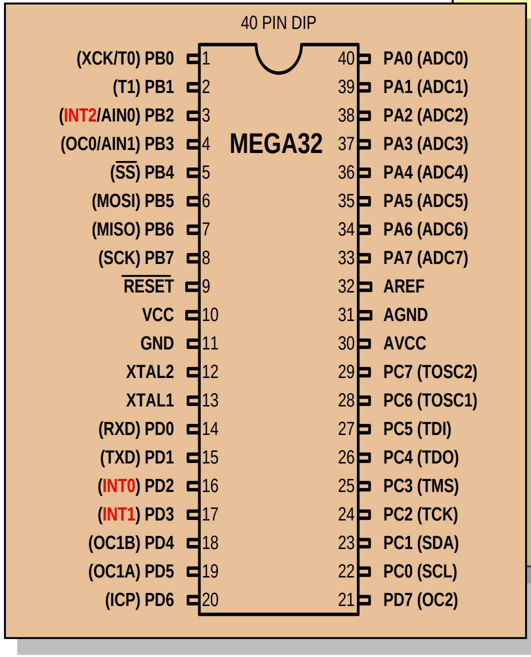
# Interrupt unit

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| TIFR | OCF2 | TOV2 | ICF1 | OCF1A | OCF1B | TOV1 | OCF0 | TOV0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| TIMSK | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | OCIE0 | TOIE0 |
| GICR | INT1 | INT0 | INT2 | - | - | - | IVSEL | IVCE |



# Interrupt unit



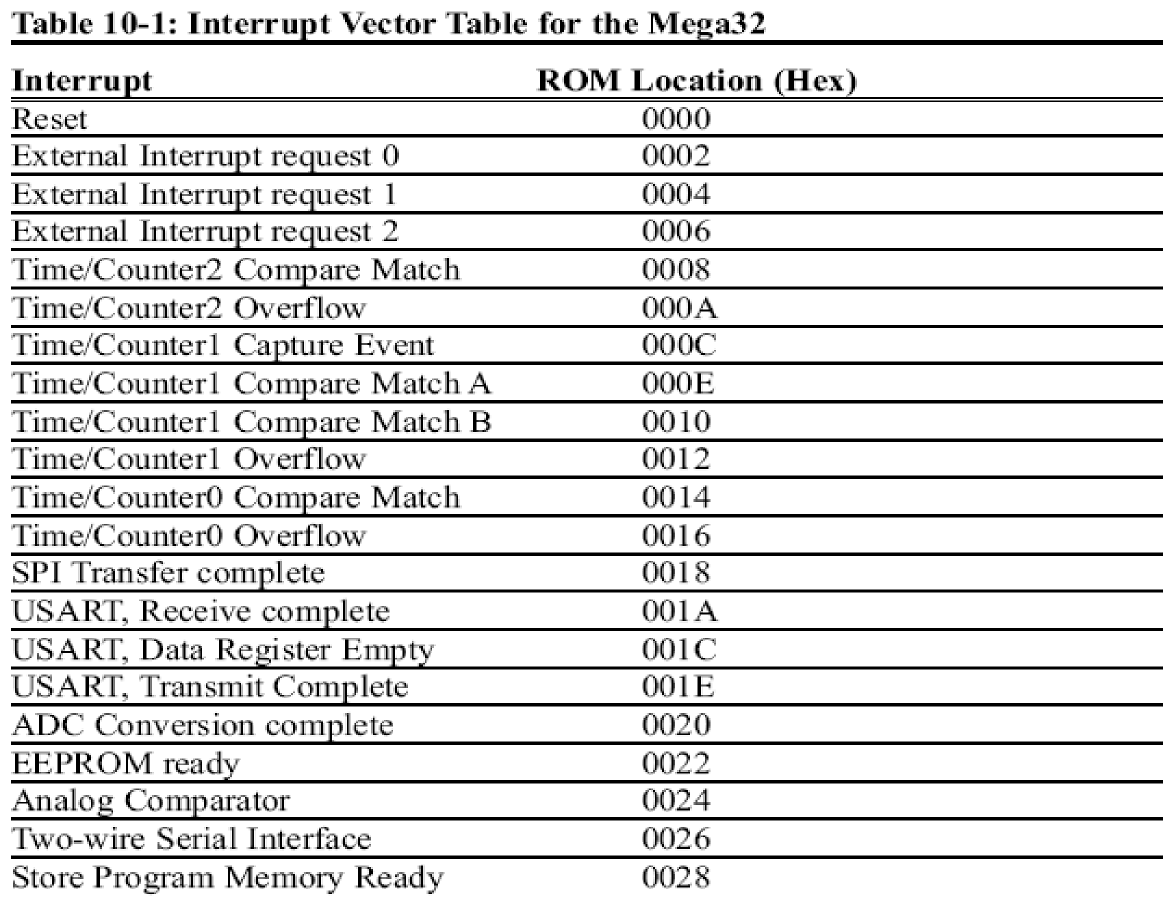


**Table 10-1: Interrupt Vector Table for the Mega32**

| Interrupt | ROM Location (Hex) |
|---|---|
| Reset | 0000 |
| External Interrupt request 0 | 0002 |
| External Interrupt request 1 | 0004 |
| External Interrupt request 2 | 0006 |
| Time/Counter2 Compare Match | 0008 |
| Time/Counter2 Overflow | 000A |
| Time/Counter1 Capture Event | 000C |
| Time/Counter1 Compare Match A | 000E |
| Time/Counter1 Compare Match B | 0010 |
| Time/Counter1 Overflow | 0012 |
| Time/Counter0 Compare Match | 0014 |
| Time/Counter0 Overflow | 0016 |
| SPI Transfer complete | 0018 |
| USART, Receive complete | 001A |
| USART, Data Register Empty | 001C |
| USART, Transmit Complete | 001E |
| ADC Conversion complete | 0020 |
| EEPROM ready | 0022 |
| Analog Comparator | 0024 |
| Two-wire Serial Interface | 0026 |
| Store Program Memory Ready | 0028 |

# Steps in executing an interrupt

- Steps in executing an interrupt:
  1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack.
  2. It also saves the current status of all the interrupts internally.
  3. It jumps to a fixed location in memory called the interrupt vector table.
  4. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it.
     - The microcontroller starts to execute the ISR until it reaches the last instruction of the subroutine which is RETI (return from interrupt).

# Steps in executing an interrupt

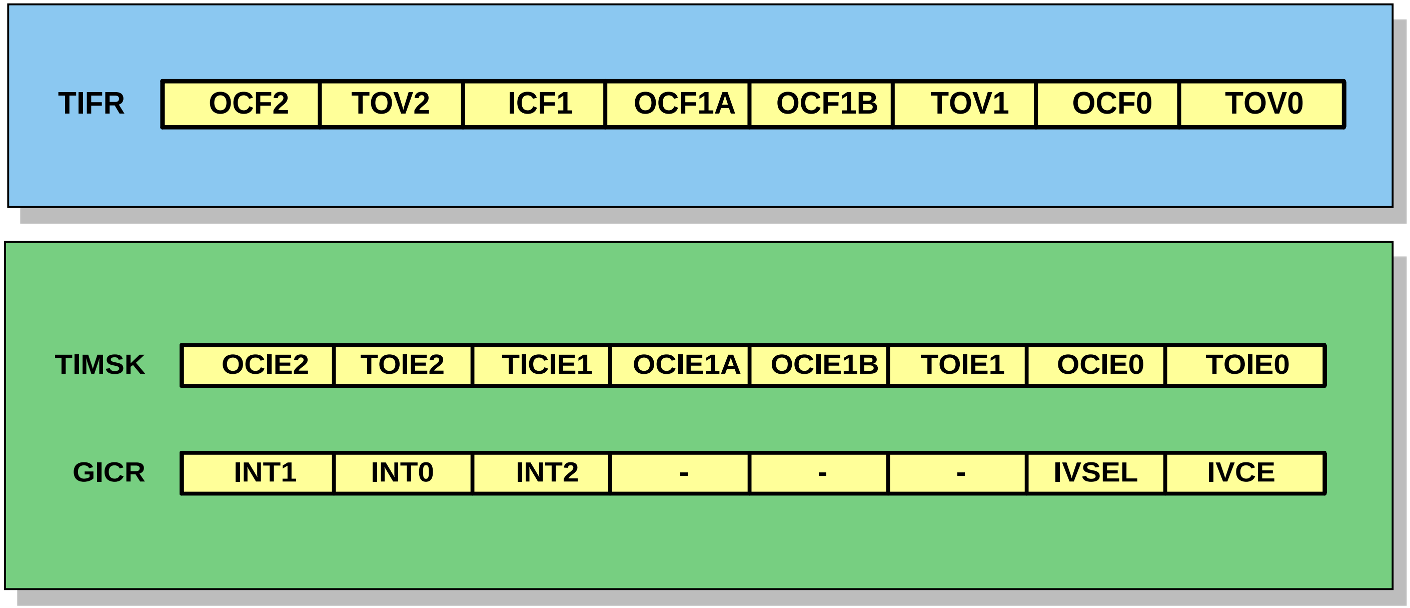
5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted.
   - First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC.
   - Then it starts to execute from that address.

# Timer interrupt

- Timer interrupt flag bits and associated registers
  - TOIE0: Timer 0 overflow interrupt enable (enable = 1)

| Interrupt | Overflow flag bit | Register | Enable bit | Register |
|---|---|---|---|---|
| Timer 0 | TOV0 | TIFR | TOIE0 | TIMSK |
| Timer 1 | TOV1 | TIFR | TOIE1 | TIMSK |
| Timer 2 | TOV2 | TIFR | TOIE2 | TIMSK |

# Timer interrupt

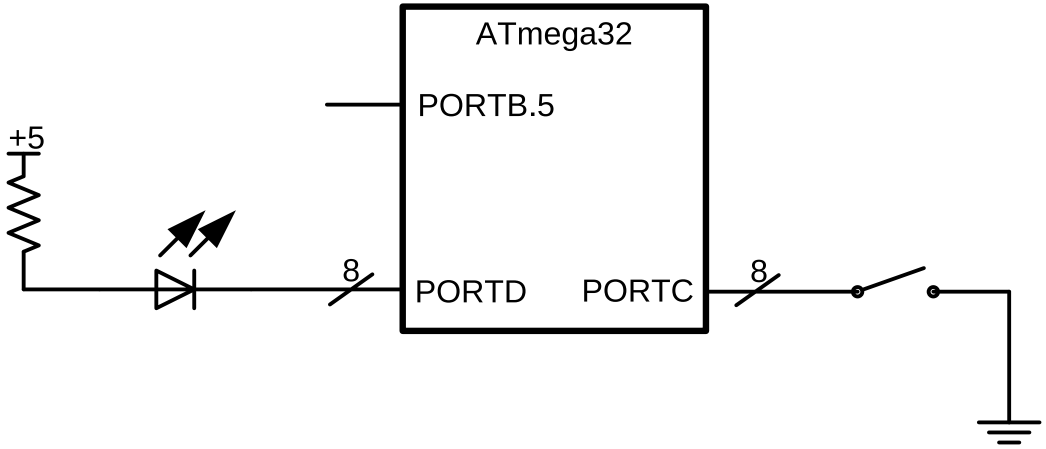


| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| TIFR | OCF2 | TOV2 | ICF1 | OCF1A | OCF1B | TOV1 | OCF0 | TOV0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| TIMSK | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | OCIE0 | TOIE0 |
| GICR | INT1 | INT0 | INT2 | - | - | - | IVSEL | IVCE |

# Timer interrupt

- Example: This program uses Timer 0 to generate a square wave on pin PORTB.5, while at the same time data is being transferred from PORTC to PORTD.



# C programming

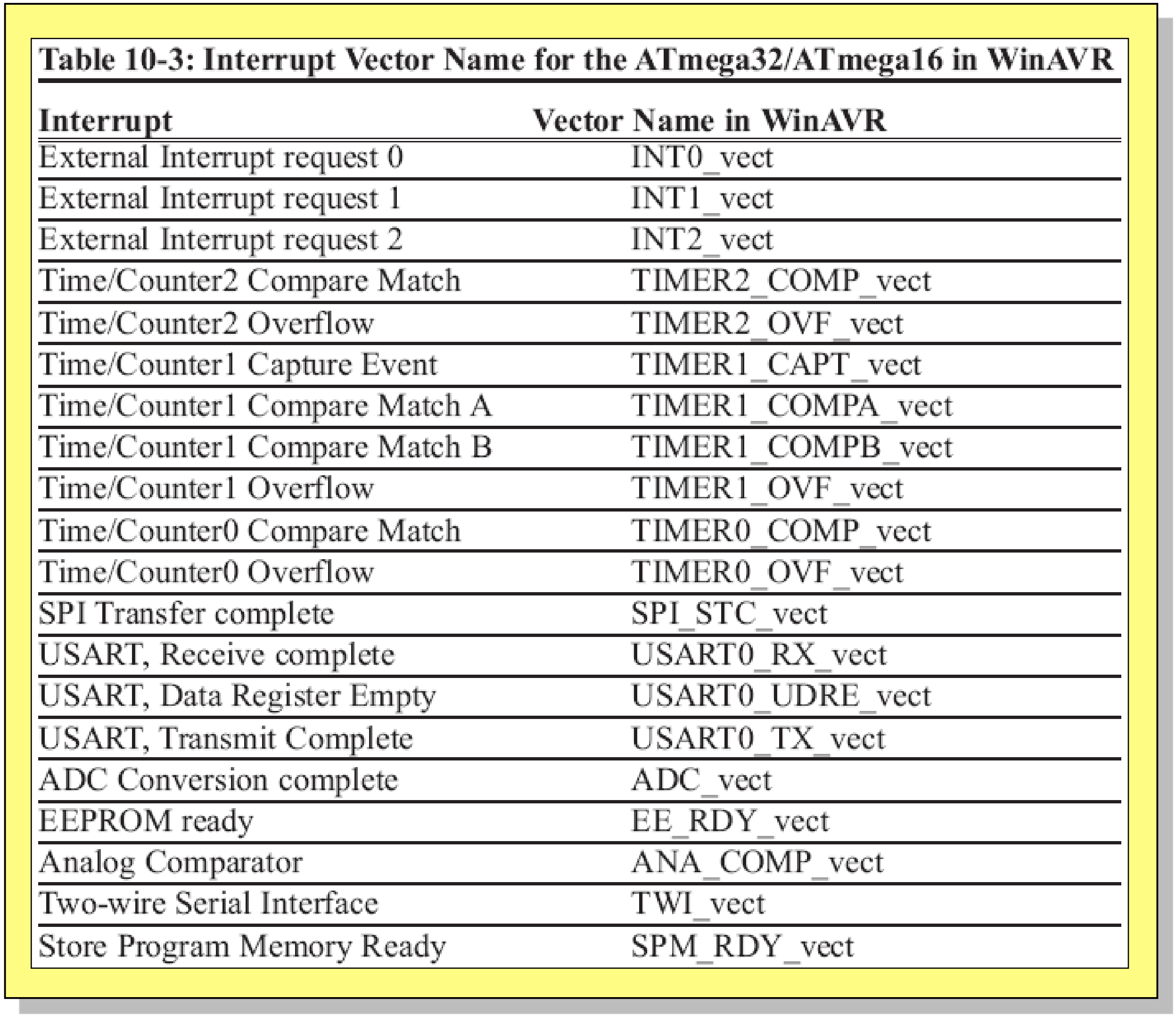


**Table 10-3: Interrupt Vector Name for the ATmega32/ATmega16 in WinAVR**

| Interrupt | Vector Name in WinAVR |
|---|---|
| External Interrupt request 0 | INT0_vect |
| External Interrupt request 1 | INT1_vect |
| External Interrupt request 2 | INT2_vect |
| Time/Counter2 Compare Match | TIMER2_COMP_vect |
| Time/Counter2 Overflow | TIMER2_OVF_vect |
| Time/Counter1 Capture Event | TIMER1_CAPT_vect |
| Time/Counter1 Compare Match A | TIMER1_COMPA_vect |
| Time/Counter1 Compare Match B | TIMER1_COMPB_vect |
| Time/Counter1 Overflow | TIMER1_OVF_vect |
| Time/Counter0 Compare Match | TIMER0_COMP_vect |
| Time/Counter0 Overflow | TIMER0_OVF_vect |
| SPI Transfer complete | SPI_STC_vect |
| USART, Receive complete | USART0_RX_vect |
| USART, Data Register Empty | USART0_UDRE_vect |
| USART, Transmit Complete | USART0_TX_vect |
| ADC Conversion complete | ADC_vect |
| EEPROM ready | EE_RDY_vect |
| Analog Comparator | ANA_COMP_vect |
| Two-wire Serial Interface | TWI_vect |
| Store Program Memory Ready | SPM_RDY_vect |

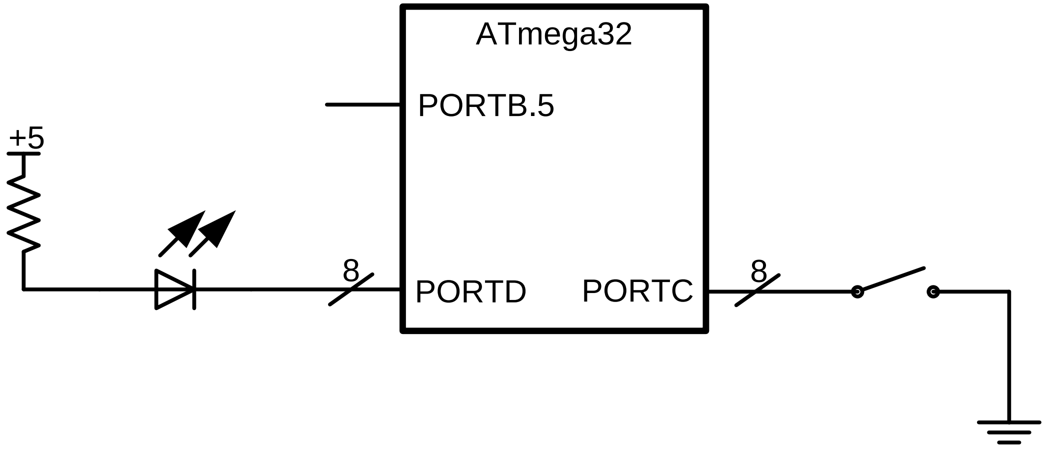
# C programming

```
sei ( );       //set I
cli  ( );      //clear I
```

```
#include "avr/io.h"
#include "avr/interrupt.h"
int main ()
{
        DDRB |= 0x20;         //DDRB.5 = output
        TCNT0 = -32;          //timer value for 4 µs
        TCCR0 = 0x01;         //Normal mode, int clk, no prescaler
        TIMSK = (1<<TOIE0);   //enable Timer0 overflow interrupt
        sei ();               //enable interrupts
        DDRC = 0x00;          //make PORTC input
        DDRD = 0xFF;          //make PORTD output
        while (1)             //wait here
          PORTD = PINC;
}
ISR (TIMER0_OVF_vect)         //ISR for Timer0 overflow
{
        TCNT0 = -32;
        PORTB ^= 0x20;        //toggle PORTB.5
}
```

# Timer interrupt

- Example: Using Timer 0 and CTC mode generate a square wave on pin PORTB.5, while at the same time data is being transferred from PORTC to PORTD.

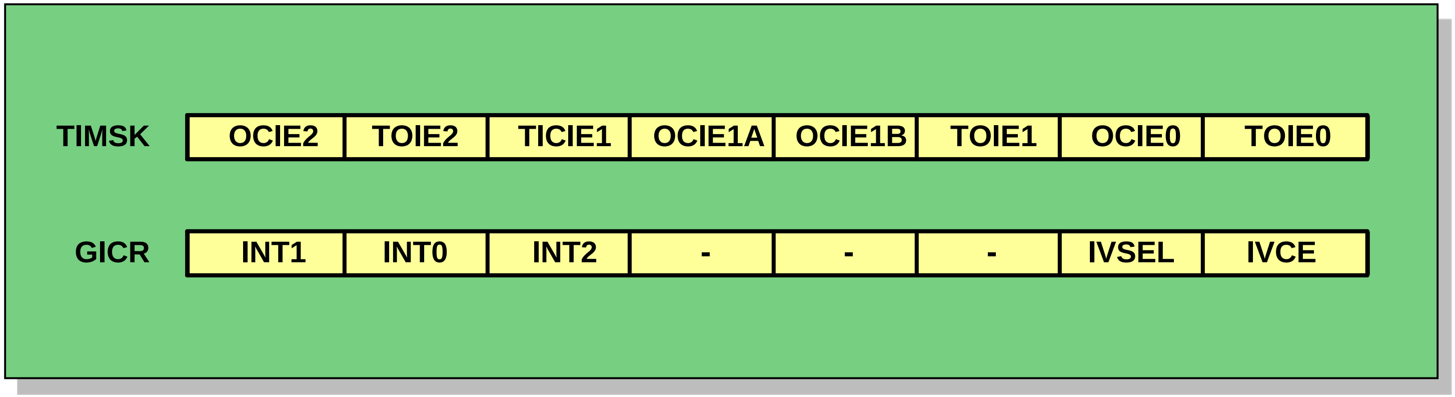


# C programming

```
#include "avr/io.h"
#include "avr/interrupt.h"
int main ()
{
        DDRB |= 0x20;         //make DDRB.5 output
        OCR0 = 40;
        TCCR0 = 0x09;         //CTC mode, internal clk, no prescaler
        TIMSK = (1<<OCIE0);   //enable Timer0 compare match int.
        sei ();               //enable interrupts
        DDRC = 0x00;          //make PORTC input
        DDRD = 0xFF;          //make PORTD output
        while (1)             //wait here
          PORTD = PINC;
}
ISR (TIMER0_COMP_vect)        //ISR for Timer0 compare match
{
        PORTB ^= 0x20;        //toggle PORTB.5
}
```

# Edge-triggered vs. level-triggered interrupts

- GICR: General Interrupt Control Register
  - INT0: External Interrupt Request 0 Enable (enable external interrupt = 1)
  - IVSEL: Interrupt Vector Select (will not be discussed here)
  - IVCE: Interrupt Vector Change Enable (will not be discussed here)



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| TIMSK | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | OCIE0 | TOIE0 |
| GICR | INT1 | INT0 | INT2 | - | - | - | IVSEL | IVCE |

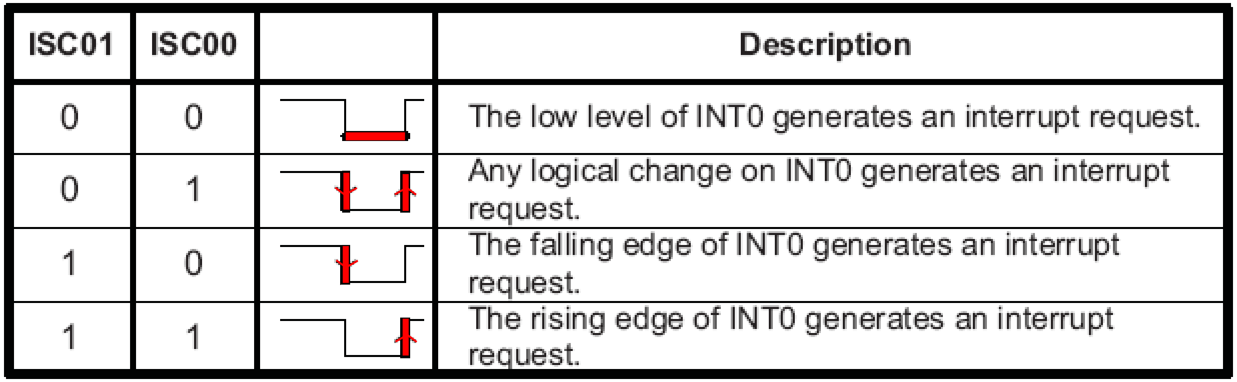
# Edge-triggered vs. level-triggered interrupts

- MCUCR: MCU Control Register
- MCUCSR: MCU Control and Status Register

# Edge-triggered vs. level-triggered interrupts

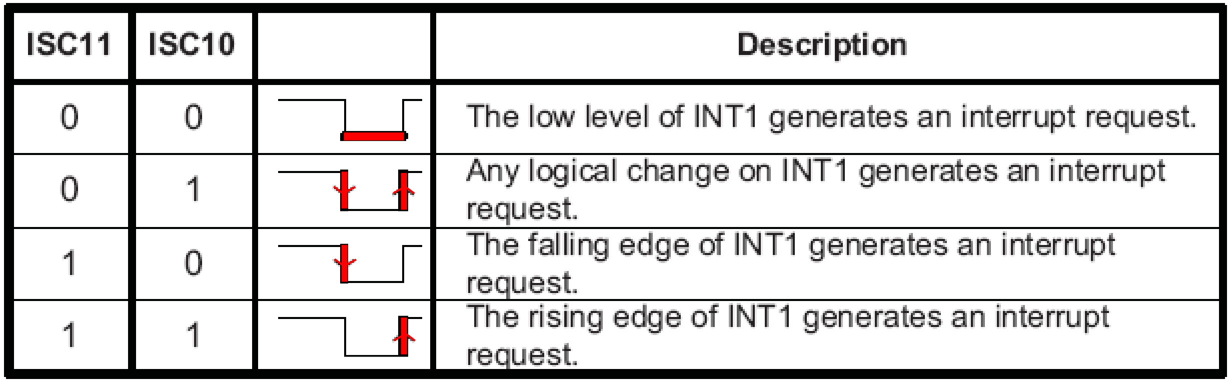
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **MCUCR** | **SE** | **SM2** | **SM1** | **SM0** | **ISC11** | **ISC10** | **ISC01** | **ISC00** |

**ISC01, ISC00 (Interrupt Sense Control bits)** These bits define the level or edge on the external INT0 pin that activates the interrupt, as shown in the following table:



| ISC01 | ISC00 | | Description |
|---|---|---|---|
| 0 | 0 | | The low level of INT0 generates an interrupt request. |
| 0 | 1 | | Any logical change on INT0 generates an interrupt request. |
| 1 | 0 | | The falling edge of INT0 generates an interrupt request. |
| 1 | 1 | | The rising edge of INT0 generates an interrupt request. |

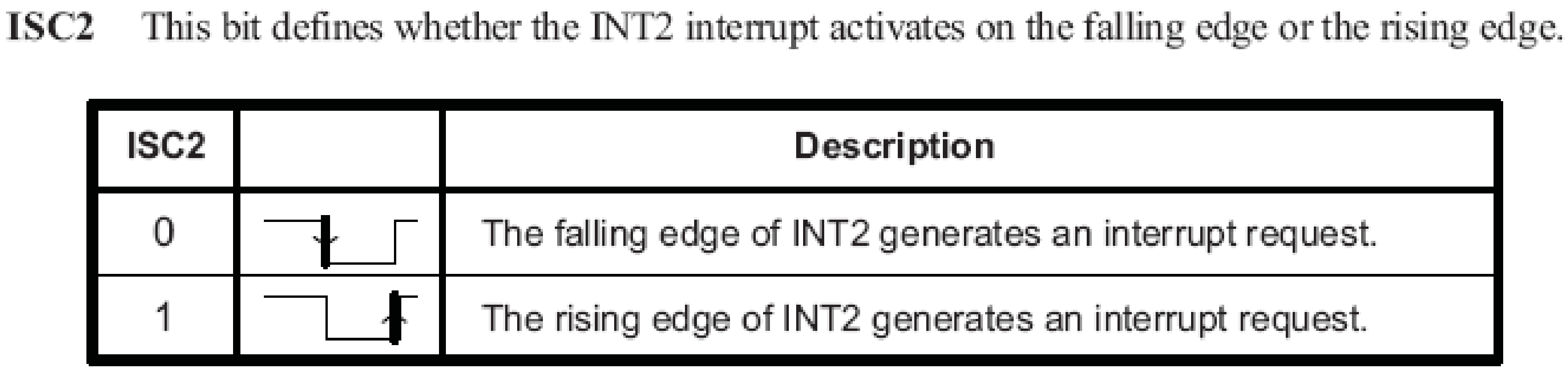
**ISC11, ISC10** These bits define the level or edge that activates the INT1 pin.



| ISC11 | ISC10 | | Description |
|---|---|---|---|
| 0 | 0 | | The low level of INT1 generates an interrupt request. |
| 0 | 1 | | Any logical change on INT1 generates an interrupt request. |
| 1 | 0 | | The falling edge of INT1 generates an interrupt request. |
| 1 | 1 | | The rising edge of INT1 generates an interrupt request. |

# Edge-triggered vs. level-triggered interrupts

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| MCUCSR | JTD | ISC2 | - | JTRF | WDRF | BORF | EXTRF | PORF |



**ISC2** This bit defines whether the INT2 interrupt activates on the falling edge or the rising edge.

| ISC2 | | Description |
|---|---|---|
| 0 | ↓ | The falling edge of INT2 generates an interrupt request. |
| 1 | ↑ | The rising edge of INT2 generates an interrupt request. |

# Class Exercise 1

- Assume that the INT0 pin is connected to a switch that is normally high. Write a C program that toggles PORTC.3, whenever INT0 pin goes low. Use the external interrupt in level-triggered mode.

# Class Exercise 1 (Your work)

# Class Exercise 1 (Answer)

# Class Exercise 2

- Rewrite Class Exercise 1 so that whenever INT0 goes low, it toggles PORTC.3 only once.

# Class Exercise 2 (Your work)

# Interrupt priority

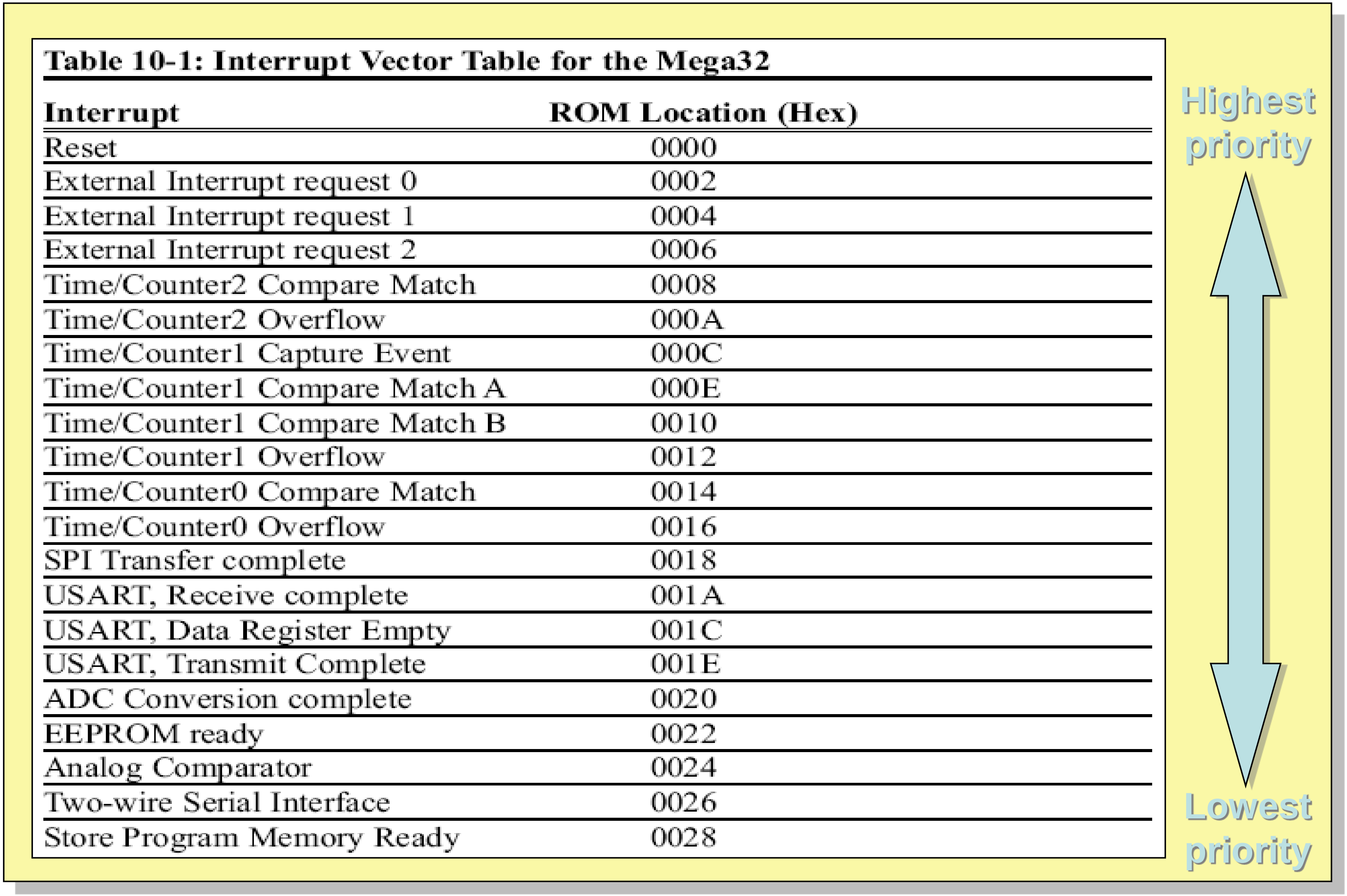


**Table 10-1: Interrupt Vector Table for the Mega32**

| Interrupt | ROM Location (Hex) |
|---|---|
| Reset | 0000 |
| External Interrupt request 0 | 0002 |
| External Interrupt request 1 | 0004 |
| External Interrupt request 2 | 0006 |
| Time/Counter2 Compare Match | 0008 |
| Time/Counter2 Overflow | 000A |
| Time/Counter1 Capture Event | 000C |
| Time/Counter1 Compare Match A | 000E |
| Time/Counter1 Compare Match B | 0010 |
| Time/Counter1 Overflow | 0012 |
| Time/Counter0 Compare Match | 0014 |
| Time/Counter0 Overflow | 0016 |
| SPI Transfer complete | 0018 |
| USART, Receive complete | 001A |
| USART, Data Register Empty | 001C |
| USART, Transmit Complete | 001E |
| ADC Conversion complete | 0020 |
| EEPROM ready | 0022 |
| Analog Comparator | 0024 |
| Two-wire Serial Interface | 0026 |
| Store Program Memory Ready | 0028 |

# Interrupt inside an interrupt

- The I flag is cleared when the AVR begins to execute an ISR. So, interrupts are disabled.
- The I flag is set when RETI is executed.

# ATmega328p

- ATmega32: GICR (General Interrupt Control Register)
- ATmega328p
  - TIMSK0/1 (Timer Interrupt Mask Register)
  - EIMSK (External Interrupt Mask Register)

# Reference Readings

- Chapter 11 – *The AVR Microcontroller and Embedded Systems : Using Assembly and C*, M. A. Mazidi, S. Naimi, and S. Naimi, Pearson, 2014.

End