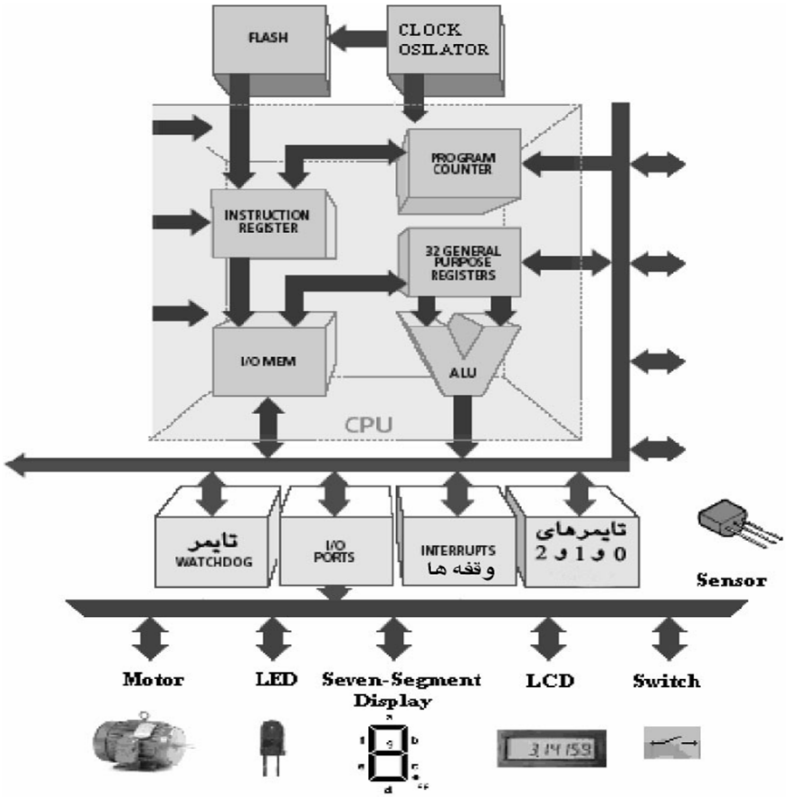
Microprocessors, Lecture 9:

Interrupts in AVR Microcontrollers (Chapter 10 of the text book)

Contents

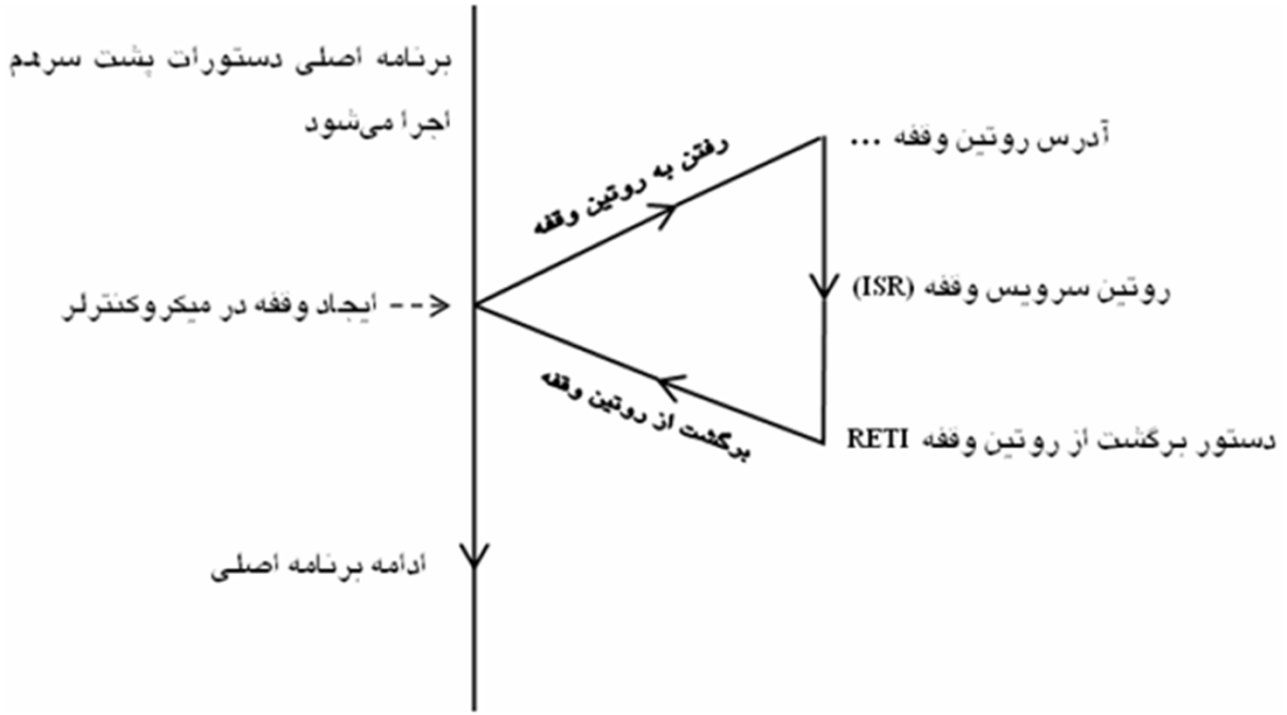
- Interrupts ATmega32
- Using interrupts in C programming

Interrupts in AVR



Interrupts: A way o improve performance

How do interrupts work?



Interrupt vs. polling

- Polling:
  - Continuously check the flags to know when the expected event occurs, then go for the next event
  - Example: Timer time-out problem of the previous lecture:

```
void T0Delay ( )
{
        TCNT0 = 0x20;            //load TCNT0
        TCCR0 = 0x01;            //Timer0, Normal mode, no prescaler
        while ((TIFR&0x1)==0);   //wait for TF0 to roll over
        TCCR0 = 0;
        TIFR = 0x1;              //clear TF0
}
```

Interrupt vs polling

- Polling: wasting time to check the devices continuously
- What if we are to generate two delays at the same time?
  - Example: Toggle bit PB.5 every 1s and PB.4 every 0.5s.
- What if there are some task to be done simultaneously with the timers?
  - Example: (1) read the contents of port A, process the data, and send them to port D continuously, (2) toggle bit PB.5 every 1s, and (3) PB.4 every 0.5s.

Interrupt vs polling

- Interrupts
  - A mechanism to work with peripheral devices
- No need for the processor to monitor the status of the devices and events
- Let the events notify the processor when they occur
  - By sending an interrupt signal to processor

Interrupt vs polling

- Interrupts:
  - Example: Copy the contents of port A to port D continuously and toggle bit PB.5 every 1s and PB.4 every 0.5s.
  - Solution:
    - » Copying the contents of port A to port D as the main program
    - » Get timers 0 and 1 to generate the delays
    - » Define two interrupts for timers 0 and 1 to notify the processor when they finish counting
    - » Upon an interrupt, stop the main program, service the timers and continue the main program

Interrupts

- Interrupting mechanism in all microprocessors and microcontrollers is almost the same:
  - Define the set of devices and events that can generate an interrupt
  - Write a function for each interrupt that will be executed when the corresponding interrupt is activated
    - » The address of this function must be saved somewhere
  - Set a priority scheme among interrupts
  - A mechanism is needed to disable all or some interrupts

Interrupts

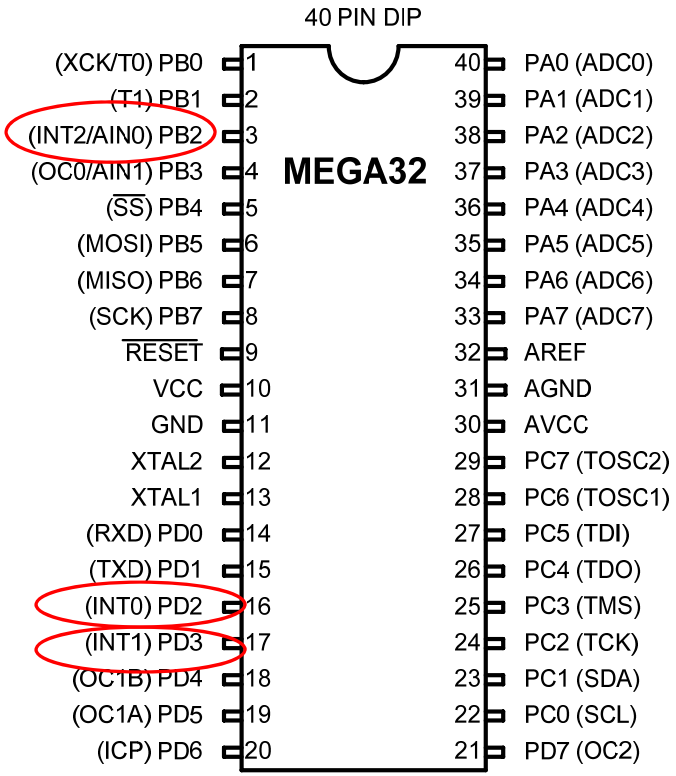
- ISR: Interrupt Service Routine
  - The function that is executed when an interrupt is enabled
- Interrupt Vector Table: a table that keeps the address of each ISR in the instruction memory

Interrupt sources in AVR

دستگاه تقاضاکننده وقفه	آدرس یا وکتور روتین سرویس وقفه			
	میکروکنترلر ATmega8 :AVR	میکروکنترلر ATmega16 :AVR	میکروکنترلر ATmega32 :AVR	میکروکنترلر ATtiny :AVR
Reset****	0x000	0x000	0x000	0x000
وقفه خارجی شماره 0 درپایه INT0	0x001	0x002	0x002	0x001
وقفه خارجی شماره 1 در پایه INT1	0x002	0x004	0x004	—
وقفه خارجی شماره 2 در پایه INT2	—	0x024	0x006	—
تایمر 2 در Compare Match	0x003	0x006	0x008	—
تایمر 2 در سرریز Overflow	0x004	0x008	0x00A	—
تایمر 1 در input capture	0x005	0x00A	0x00C	—
تایمر 1 در Compare Match A	0x006	0x00C	0x00E	0x003
تایمر 1 در Compare Match B	0x007	0x00E	0x010	—
تایمر 1 در سرریز یا Overflow	0x008	0x010	0x012	0x004
تایمر 0 در سرریز یا Overflow	0x009	0x012	0x016	0x005
تایمر* 0 در Compare Match	…	0x026	0x014	…
پورت سری USART برای دریافت کامل یک بایت (RXC)	0x00B	0x016	0x01A	…
…	…	…	…	…
مبدل آنالوگ به دیجیتال ADC	0x00E	0x01C	0x020	0x008
…	…	…	…	…
مقایسه‌کننده آنالوگ	0x010	0x020	0x024	0x007
…	…	…	…	…

External interrupts

- **3 pins of ATmega32**



Interrupts in AVR

- Just 2-bytes for each interrupt service routine
- Too small to write the interrupt service routine
- → Write the routine somewhere in the memory and put a code to jump to the address of the function in the 2-byte assigned to the ISR

آدرس یا وکتور روتین سرویس وقفه				دستگاه تقاضاکننده وقفه
میکروکنترلر ATtiny :AVR	میکروکنترلر ATmega32 :AVR	میکروکنترلر ATmega16 :AVR	میکروکنترلر ATmega8 :AVR	
0x000	0x000	0x000	0x000	Reset***
0x001	0x002	0x002	0x001	وقفه خارجی شماره 0 درپایه INT0
—	0x004	0x004	0x002	وقفه خارجی شماره 1 در پایه INT1
—	0x006	0x024	—	وقفه خارجی شماره 2 در پایه INT2
—	0x008	0x006	0x003	تایمر 2 در Compare Match
—	0x00A	0x008	0x004	تایمر 2 در سرریز Overflow
—	0x00C	0x00A	0x005	تایمر 1 در input capture
0x003	0x00E	0x00C	0x006	تایمر 1 در Compare Match A
—	0x010	0x00E	0x007	تایمر 1 در Compare Match B
0x004	0x012	0x010	0x008	تایمر 1 در سرریز یا Overflow
0x005	0x016	0x012	0x009	تایمر 0 در سرریز یا Overflow
...	0x014	0x026	...	تایمر* 0 در Compare Match
...	0x01A	0x016	0x00B	پورت سری USART برای دریافت کامل یک بایت (RXC)
...	...	...	...	...
0x008	0x020	0x01C	0x00E	مبدل آنالوگ به دیجیتال ADC
...	...	...	...	...
0x007	0x024	0x020	0x010	مقایسه‌کننده آنالوگ
...	...	...	...	...

Interrupts in AVR

- How is an interrupt serviced?

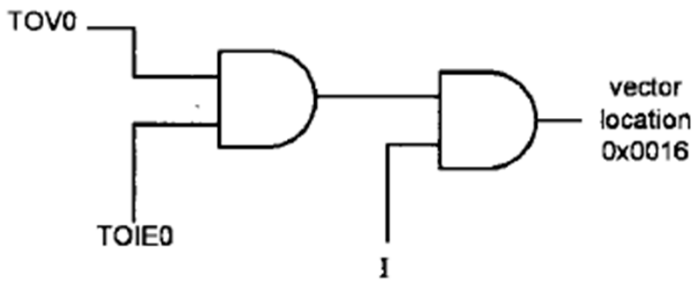
1. Stop fetching the next instruction and save PC
2. Go to Interrupt Vector Table to find the address of the ISR of the interrupting device
3. Execute the function
4. Resume normal execution by retrieving PC

Enabling Interrupts

- Interrupts can be enabled or disabled by programmer
  - Bit7 (I) in SREG (status register)
  - SREG keeps the processor status (remember the first lecture on AVR)
  - Disabled on reset (I=0)

Enabling Interrupts

- In addition to Bit7 (I) in SREG each interrupt should be enabled independently
- The enable bit of each interrupt is in some register
  - Example: TIMSK register to enable/disable timer interrupts
  - TIMSK= Timer Interrupt Mask
  - Mask? پوشاندن ، غیرفعال کردن
- To enable timer1 overflow interrupt:
  - Bit7 (I) in SREG ← 1
  - Bit 0 of TIMSK (TOIE0)← 1

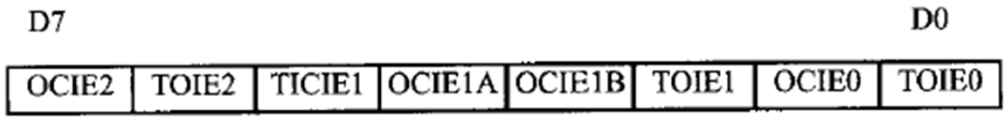


Interrupt priority

- **What if two or more interrupts occur at the same time?**
  - The interrupt with lower ISR address is prioritized (external int. 0 has the highest priority)
- **When an interrupt is serviced, the *I* bit becomes automatically 0 to disable interrupts**
  - Will be enabled when returning from the ISR

دستگاه تقاضاکننده وقفه	آدرس یا وکتور روتین سرویس وقفه			
	میکروکنترلر ATmega8 :AVR	میکروکنترلر ATmega16 :AVR	میکروکنترلر ATmega32 :AVR	میکروکنترلر ATtiny :AVR
Reset ***	0x000	0x000	0x000	0x000
وقفه خارجی شماره 0 درپایه INT0	0x001	0x002	0x002	0x001
وقفه خارجی شماره 1 در پایه INT1	0x002	0x004	0x004	—
وقفه خارجی شماره 2 در پایه INT2	—	0x024	0x006	—
تایمر 2 در Compare Match	0x003	0x006	0x008	—
تایمر 2 در سرریز Overflow	0x004	0x008	0x00A	—
تایمر 1 در input capture	0x005	0x00A	0x00C	—
تایمر 1 در Compare Match A	0x006	0x00C	0x00E	0x003
تایمر 1 در Compare Match B	0x007	0x00E	0x010	—
تایمر 1 در سرریز یا Overflow	0x008	0x010	0x012	0x004
تایمر 0 در سرریز یا Overflow	0x009	0x012	0x016	0x005
تایمر * 0 در Compare Match	...	0x026	0x014	...
پورت سری USART برای دریافت کامل یک بایت (RXC)	0x00B	0x016	0x01A	...
...	...	...	...	...
مبدل آنالوگ به دیجیتال ADC	0x00E	0x01C	0x020	0x008
...	...	...	...	...
مقایسه‌کننده آنالوگ	0x010	0x020	0x024	0x007
...	...	...	...	...

TIMSK register



TOIE0 Timer0 overflow interrupt enable
= 0 Disables Timer0 overflow interrupt
= 1 Enables Timer0 overflow interrupt

OCIE0 Timer0 output compare match interrupt enable
= 0 Disables Timer0 compare match interrupt
= 1 Enables Timer0 compare match interrupt

TOIE1 Timer1 overflow interrupt enable
= 0 Disables Timer1 overflow interrupt
= 1 Enables Timer1 overflow interrupt

OCIE1B Timer1 output compare B match interrupt enable
= 0 Disables Timer1 compare B match interrupt
= 1 Enables Timer1 compare B match interrupt

OCIE1A Timer1 output compare A match interrupt enable
= 0 Disables Timer1 compare A match interrupt
= 1 Enables Timer1 compare A match interrupt

TICIE1 Timer1 input capture interrupt enable
= 0 Disables Timer1 input capture interrupt
= 1 Enables Timer1 input capture interrupt

TOIE2 Timer2 overflow interrupt enable
= 0 Disables Timer2 overflow interrupt
= 1 Enables Timer2 overflow interrupt

OCIE2 Timer2 output compare match interrupt enable
= 0 Disables Timer2 compare match interrupt
= 1 Enables Timer2 compare match interrupt

Interrupt programming in C

- **Enable interrupts**
- **Set the mask register (TIMSK for timers)**
  - **Example: TIMSK=0x01;**
- **Write the ISR function to specify what operation should be done when the interrupt occurs**

→ Compiler dependent: different in different compilers!

Interrupt programming in C

- A way to use assembly instructions in C:

  #asm("instruction")
- SEI: (Set I) an assembly instruction that enables interrupts (bit 7 of SREG=1)
- CLI: Clear I
- #asm("sei"); enable interrupts in C
- No way to access SREG.7 in C
- Mazidi's book uses a different compiler→ different instructions: sei() instead of #asm("sei"), different ISR names

Example 10-8 (C version of Program 10-1)

Using Timer0 generate a square wave on pin PORTB.5, while at the same time transferring data from PORTC to PORTD.

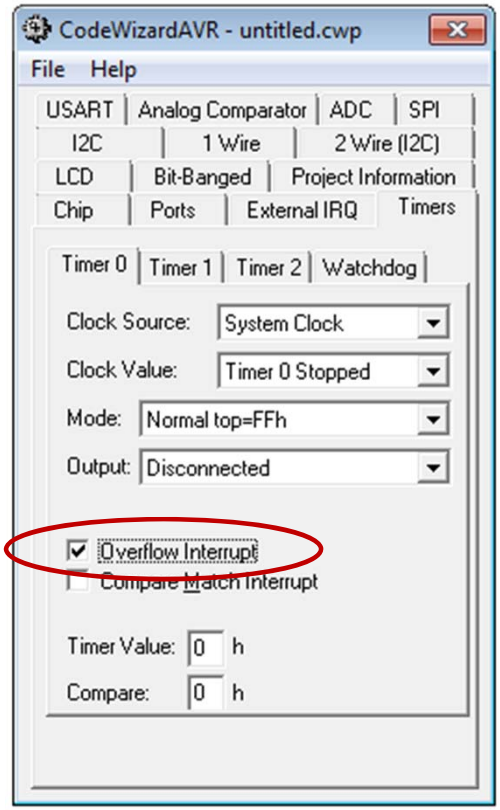
Solution:

```
#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
                               //enable interrupts
      #asm("sei");
      DDRC = 0x00;             //make PORTC input
      DDRD = 0xFF;             //make PORTD output

      while (1)                //wait here
            PORTD = PINC;
}

 interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
      TCNT0 = -32;
      PORTB ^= 0x20;           //toggle PORTB.5
}
```

→ On entering the timers ISR, the TOV0 bit is automatically cleared, no need to ~~be cleared by software~~

Interrupt programming in C

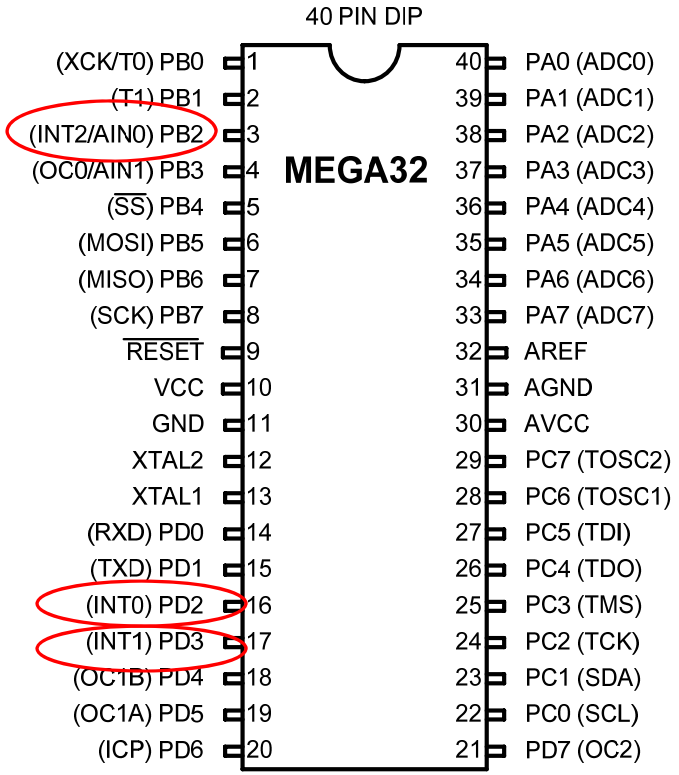
- **In *codevision*, ISR is generated during project setup. Just fill the function body!**



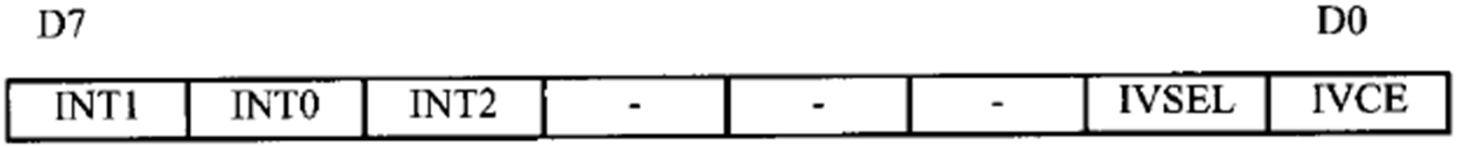

```
#include <mega32.h>


// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Place your code here
}

// Declare your global variables here
 void main(void)
{

TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;

// Global enable interrupts
#asm("sei")

while (1)
      {
      // Place your code here

      };
}
```

External Interrupts

- **To allow external sources interrupt the microcontroller**
- **Can be masked by CIGR register**
- **GICR: General interrupt control register**



External Interrupts- GICR



INT0 External Interrupt Request 0 Enable
= 0 Disables external interrupt 0
= 1 Enables external interrupt 0

INT1 External Interrupt Request 1 Enable
= 0 Disables external interrupt 1
= 1 Enables external interrupt 1

INT2 External Interrupt Request 2 Enable
= 0 Disables external interrupt 2
= 1 Enables external interrupt 2

These bits, along with the I bit, must be set high for an interrupt to be responded to.

External interrupts

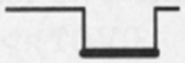
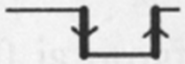
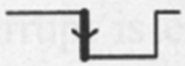
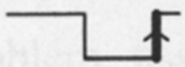
- Interrupts can be edge triggered or level triggered
- Edge trigger: activated when a change in signal level occurs
- Level trigger: activated when a signal has a specific value
- INT0 and INT1 can be programmed to be edge or level triggered
  - Low-level active by default
- INT 2 is only edge triggered

External interrupts

- **A register called ISC (interrupt sense control) can set the interrupt type of INT0 and INT1**

ISC Register

SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

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.

ISC10 and ISC11 set the same setting for INT1

External Interrupts- C programming

Example 10-12 (C version of Example 10-5)

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

Solution:

```
#include "avr/io.h"
#include "avr/interrupt.h"

int main ()
{
        DDRC = 1<<3;                    //PC3 as an output
        PORTD = 1<<2;                   //pull-up activated
        GICR = (1<<INT0);               //enable external interrupt 0
       #asm("sei");                     //enable interrupts

        while (1);                      //wait here
}

ISR (INT0_vect)                         //ISR for external interrupt 0
{
        PORTC ^= (1<<3);                //toggle PORTC.3
}
```