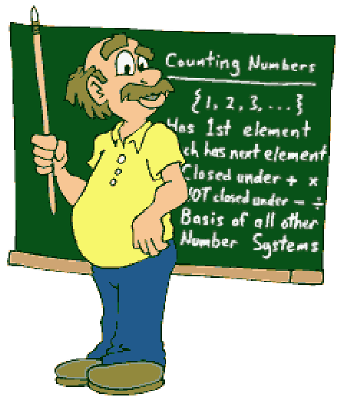
IECA

Embedded Computer Architecture

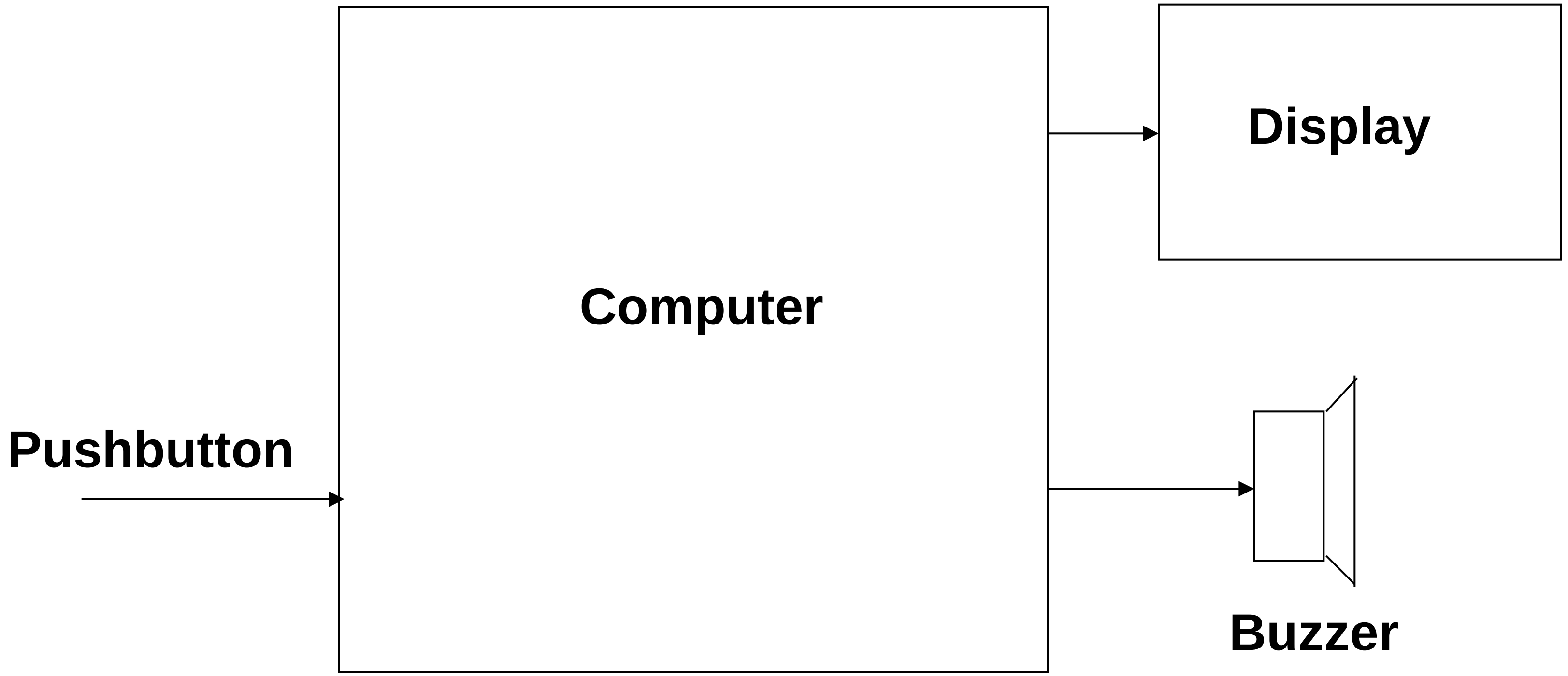
Lesson 16: Interrupts



Version: 12-11-2015, Henning Hargaard

Interrupt = (Temporary) break

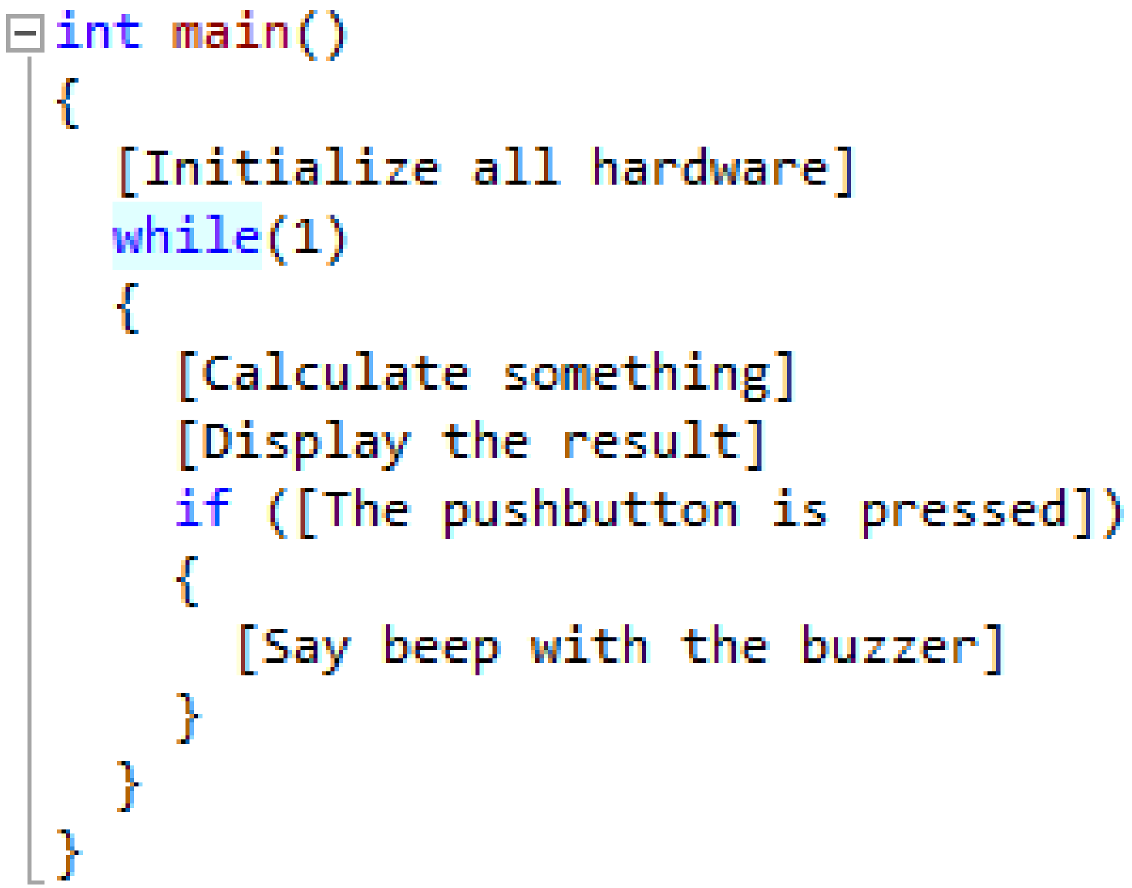
Example



* The computer continuously calculates "something", and then displays it.

* If you press the pushbutton, the buzzer must say "beep".

Solution <u>without</u> using interrupt



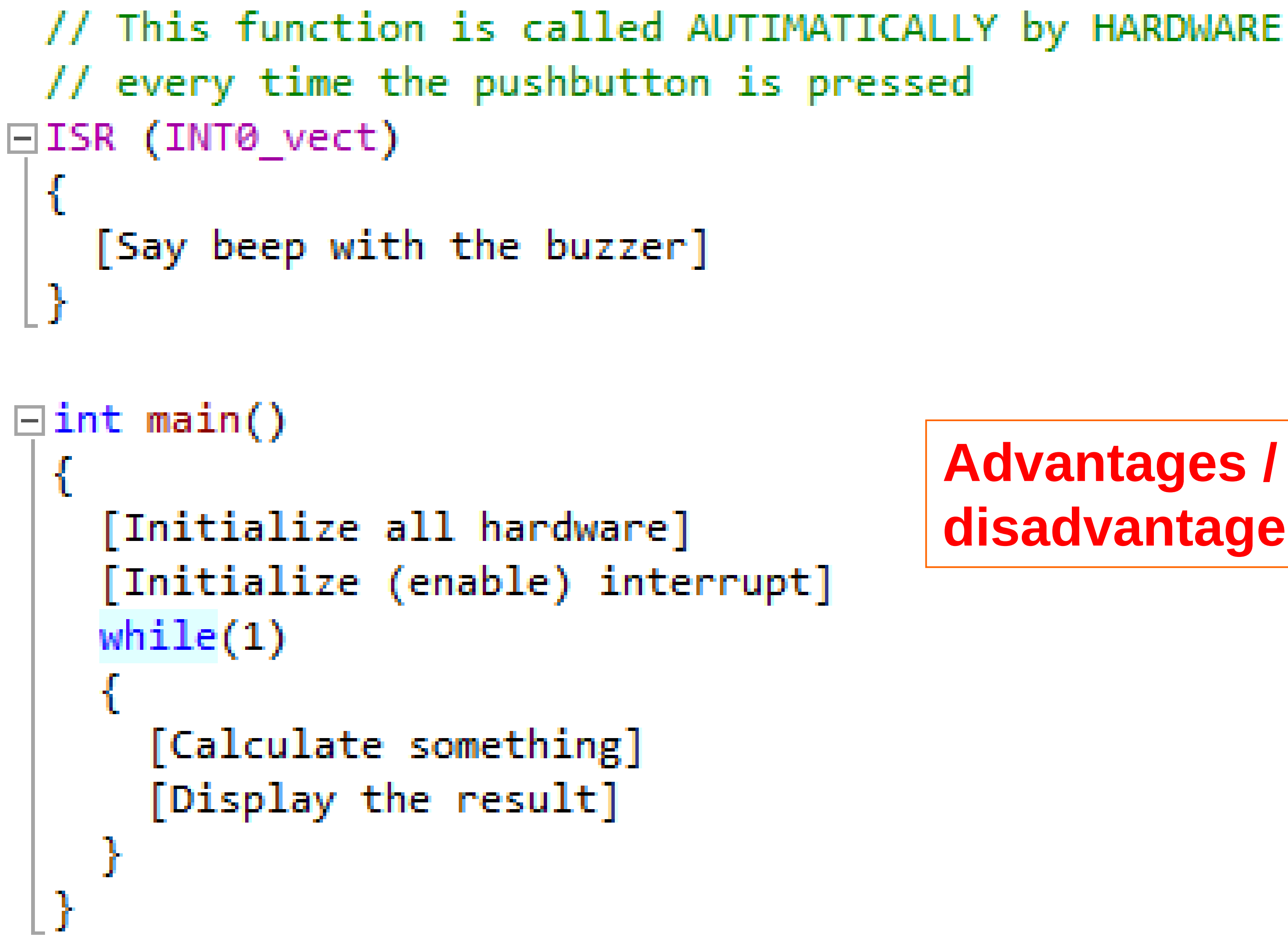
```
int main()
{
  [Initialize all hardware]
  while(1)
  {
    [Calculate something]
    [Display the result]
    if ([The pushbutton is pressed])
    {
      [Say beep with the buzzer]
    }
  }
}
```

What's the disadvantage ?

What are interrupts ?

- An interrupt is a temporary interruption of the ongoing program execution.
- Typically the interrupt is triggered via a hardware signal (an event).
- During the interrupt a special function is executed. The special function is called an interrupt routine (or interrupt service routine = ISR). This is to be written by the programmer.
- After the interrupt the interrupted program execution will be resumed.

Solution using interrupt



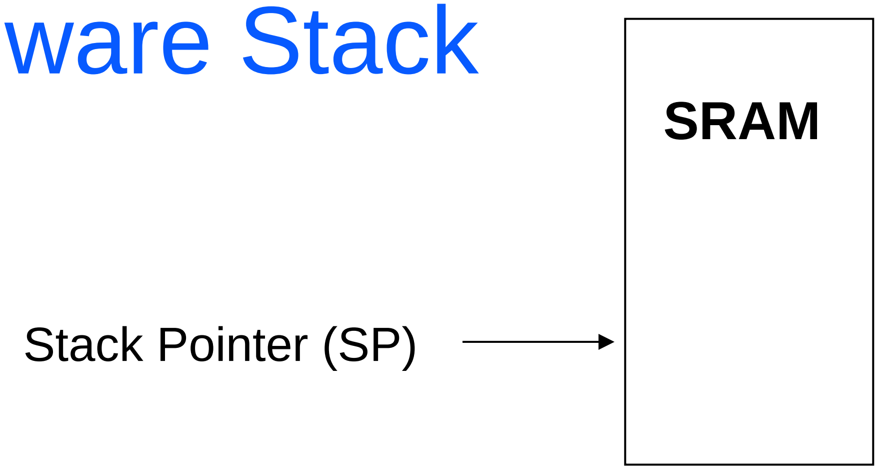
```
// This function is called AUTIMATICALLY by HARDWARE
// every time the pushbutton is pressed
ISR (INT0_vect)
{
  [Say beep with the buzzer]
}

int main()
{
  [Initialize all hardware]
  [Initialize (enable) interrupt]
  while(1)
  {
    [Calculate something]
    [Display the result]
  }
}
```

Advantages / disadvantages ?

The interrupt sequency

- The ongoing instruction must be executed.
- The program counter is saved on the stack.
  (4 clock cycles)
- Jumps to the interrupt routine.
  (3 clock cycles)
- The global interrupt flag is cleared.
  If ”sei” in the interrupt routine => possibility for ”nested interrupts”.
- RETI => The program counter is fetched from the stack, and the global interrupt flag is set again.
  (4 clock cycles).

Hardware Stack



- The stack pointer is an important register that keeps track of where the ”top of stack” is (e.g. next available location (address)).
- Each time the stack is written to the stack, the SP is **decremented**.
- Each time the stack is read from the stack, the SP is **incremented**.

Interrupts and the stack

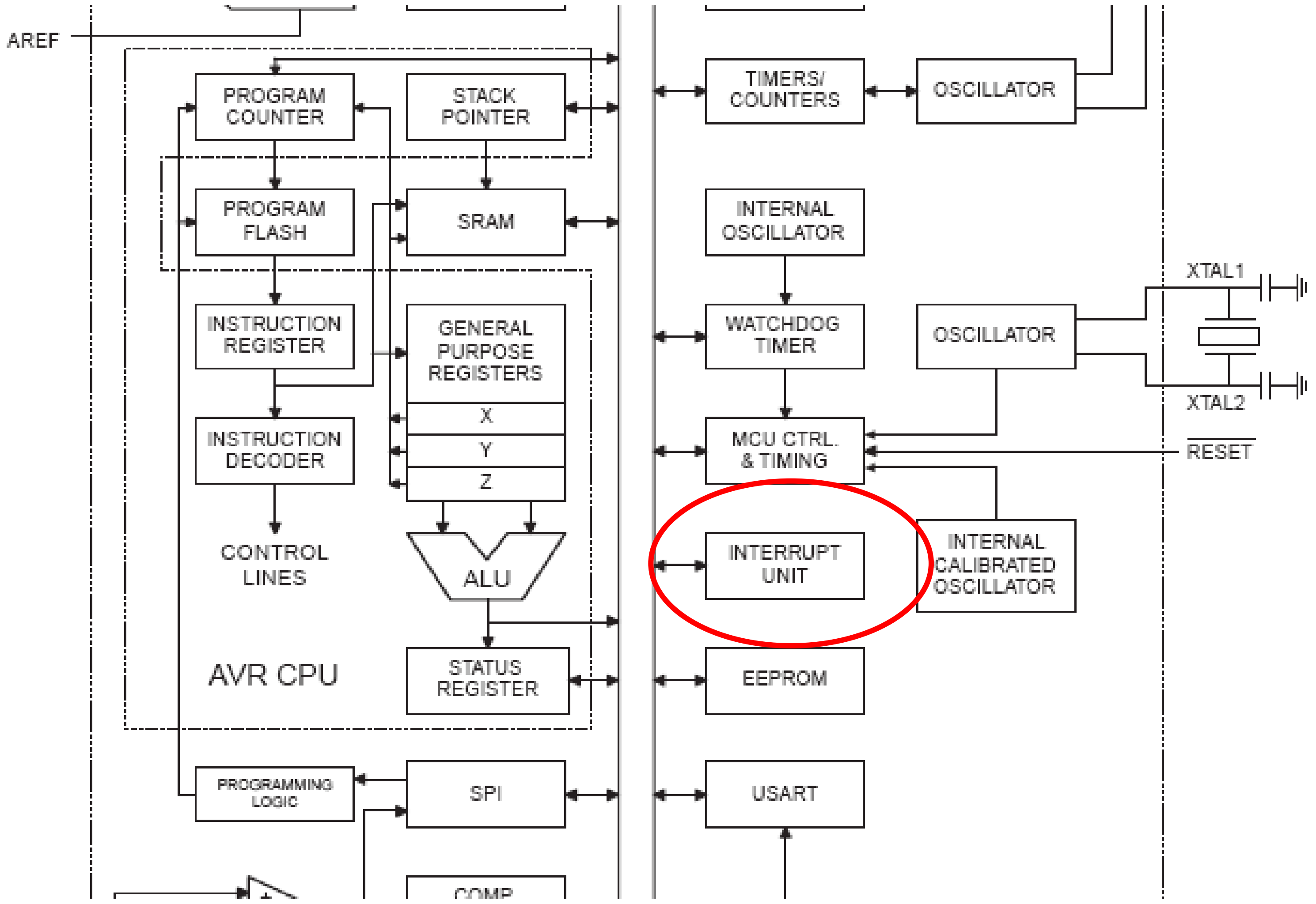
By HW interrupt the address of the next instruction is **automatically saved** on the stack ! SP is **de**cremented.

```
Interrupt:
          MOV R0,R4
          INC R7
          MOV R1,R3
          RETI
```

Program counter (PC) is loaded from the stack. SP is **in**cremented.

- By interrupts the stack is used automatically, so the program execution will be resumed afterwards.

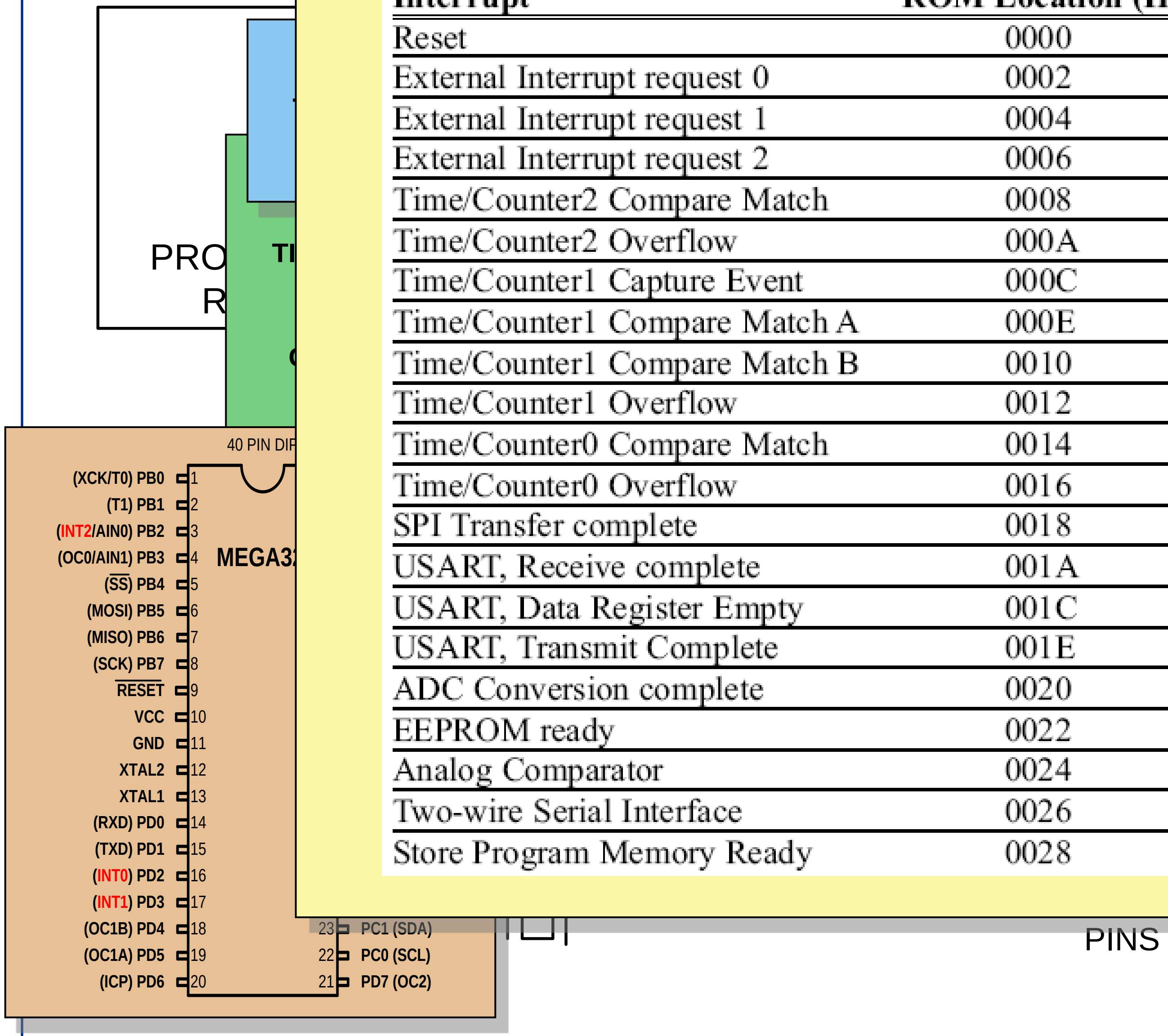
Mega32 interrupt unit



Global interrupt enable

- By a single bit (Global Interrupt Enable Flag) all interrups are enabled/disabled.
  Can be considered as the main switch of the interrupt system.
- The assembly instruction ”sei” enables interrupts.
- The assembly instruction ”cli” disables interrupts.

Interrupt unit



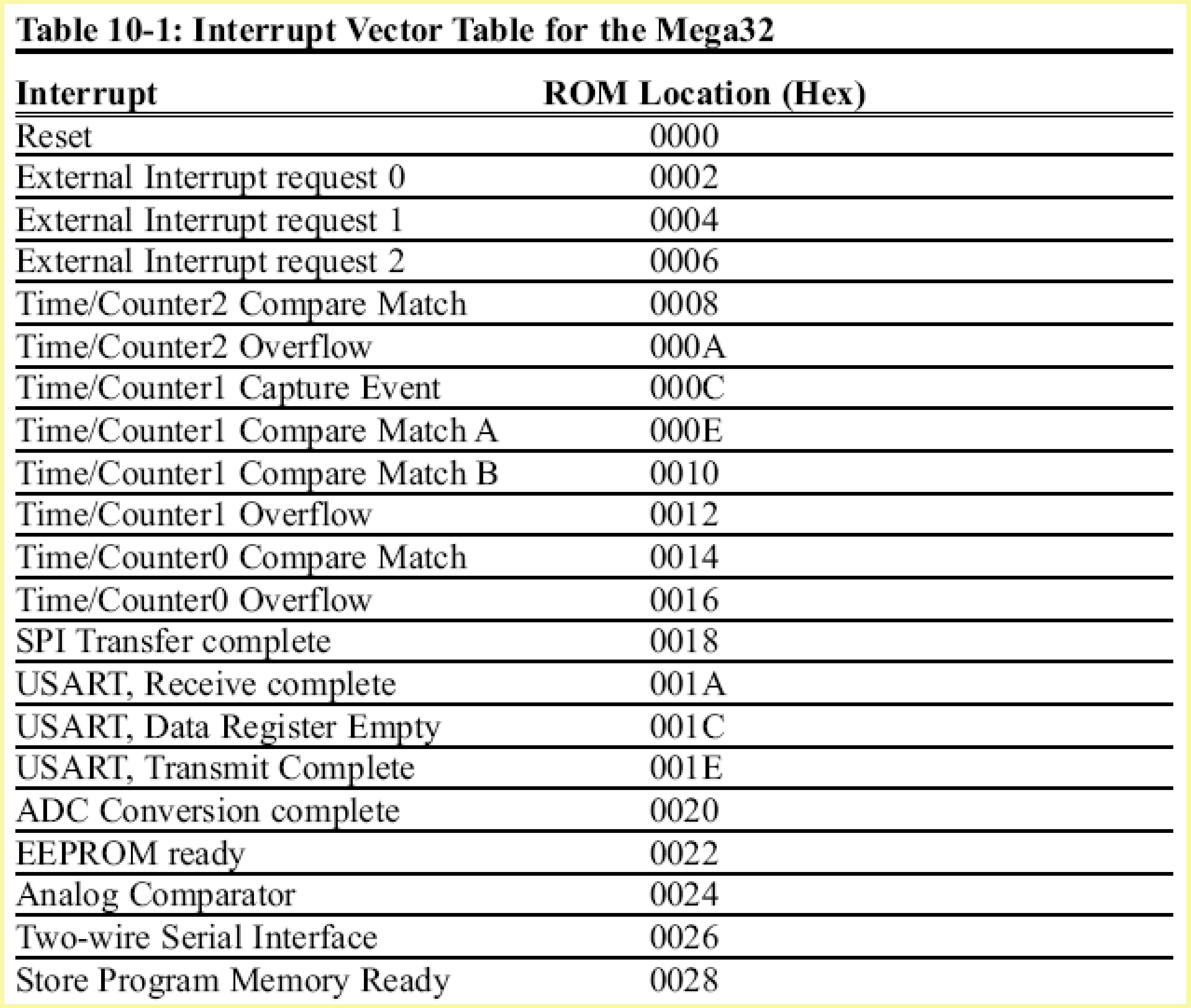


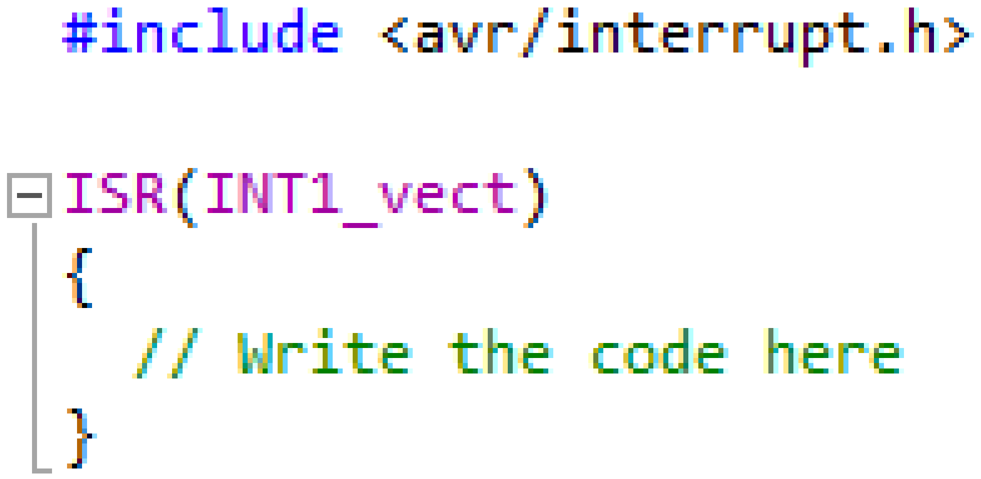
Table 10-1: Interrupt Vector Table for the Mega32

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

Code examples (AVR GCC)

```
// Global interrupt enable
sei();


// Global interrupt disable
cli();
```

Code example (ISR)



```
#include <avr/interrupt.h>

ISR(INT1_vect)
{
    // Write the code here
}
```

ISR names using AVR GCC

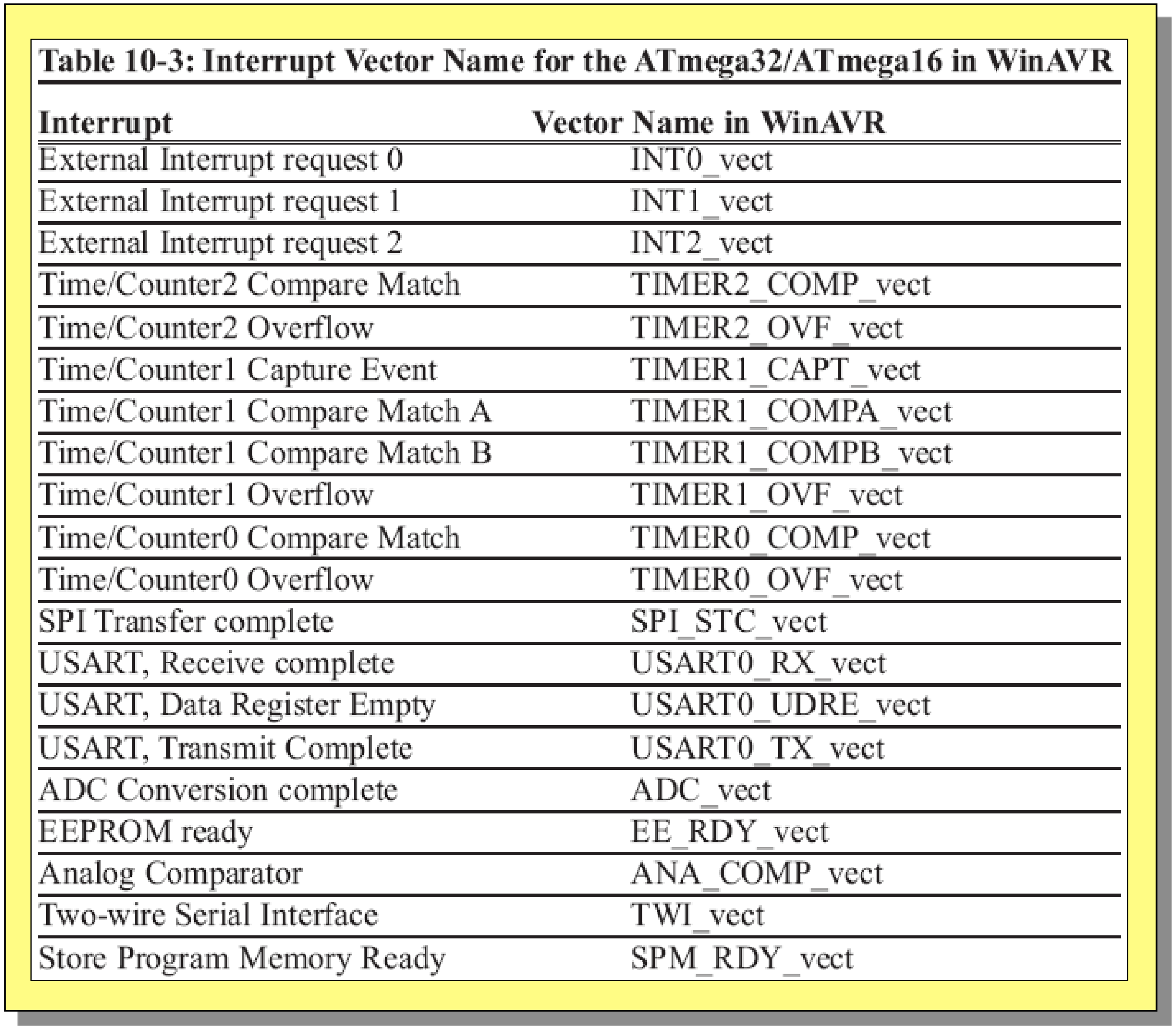


Table 10-3: Interrupt Vector Name for the ATmega32/ATmega16 in WinAVR

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

Local interrupt enables

- **The individual interrupt types** each have a **local** interrupt enable flag.

- **Only** if the local interrupt enable flag is set AND the global interrupt flag is set, the device will be able to start an interrupt !

Mega32: External interrupts

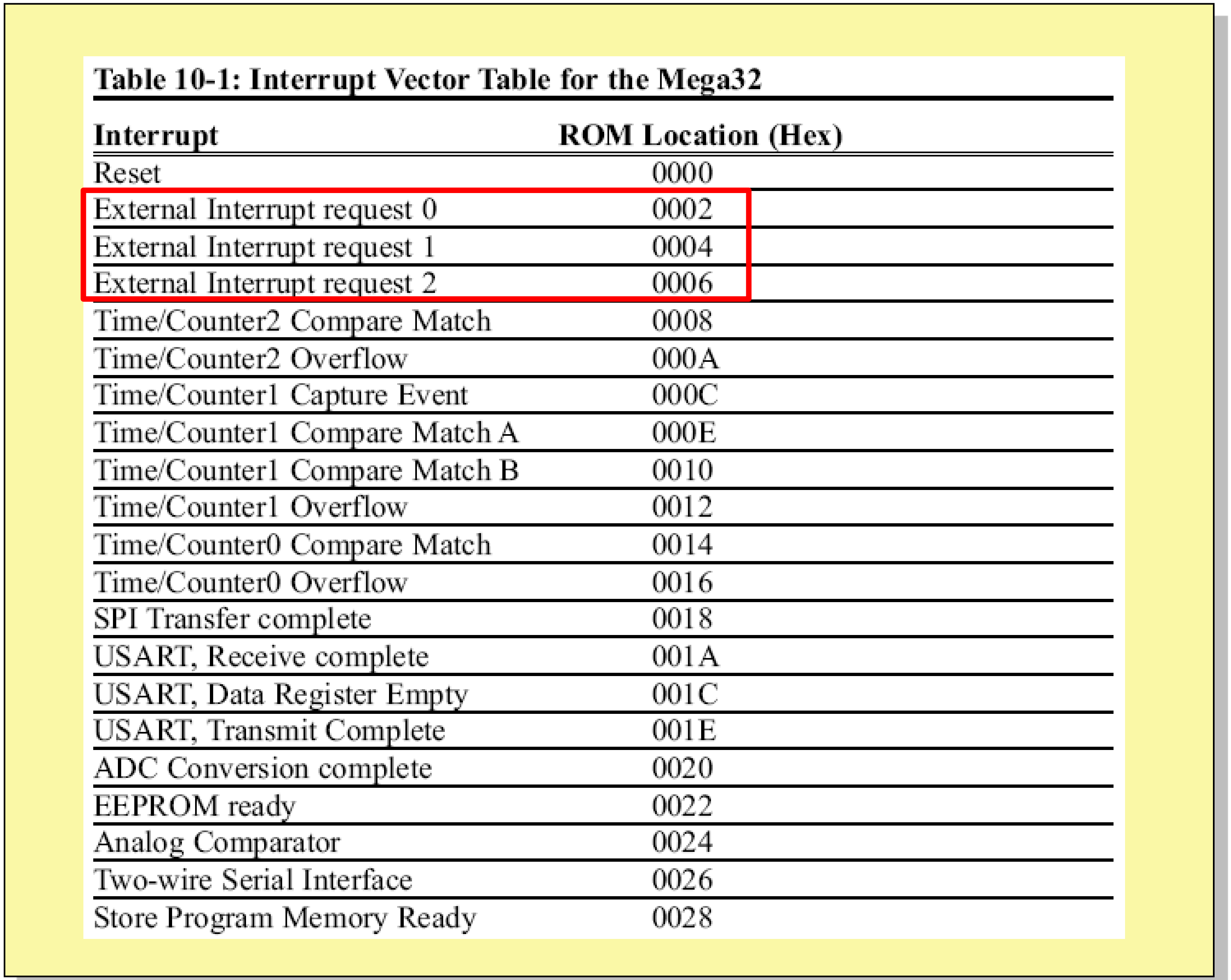
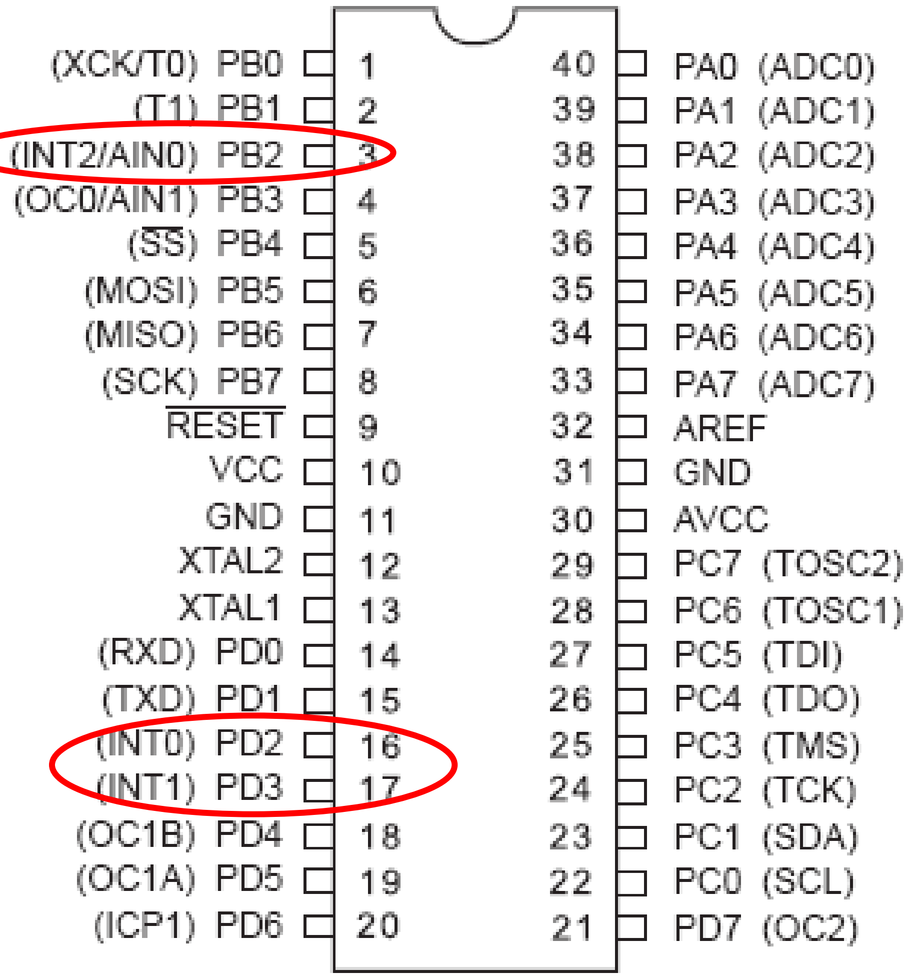


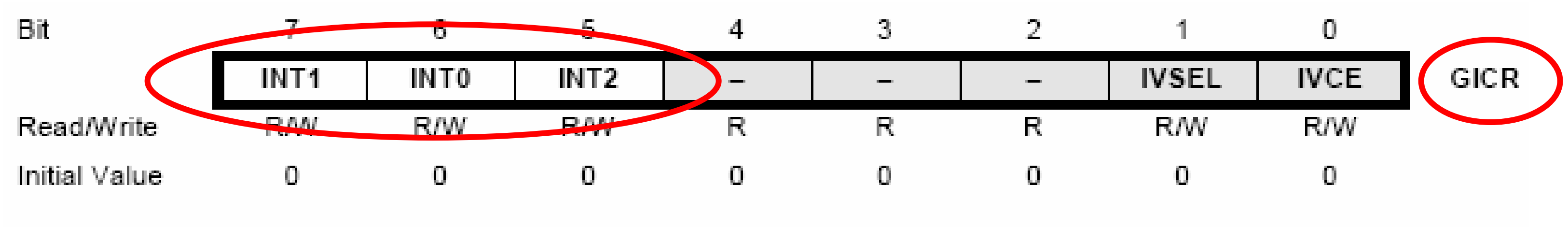
Table 10-1: Interrupt Vector Table for the Mega32

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

Mega32: External interrupt pins



External interrupt enable bits



Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- 0 : Disabled.
- 1 : Enabled.

- Code example:
  GICR = GICR | 0b11000000;
  -- or --
  GICR |= 0b11000000;

External interrupts (INT0,INT1,INT2)

- INT0 and INT1 :
  User decides whether to trig on positive or negative edges or to level trig (active low).
  In addition, “trig on any change” is an option.

- INT2 is always edge triggered (positive or negative).

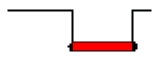
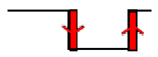
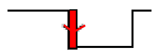
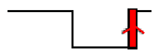
- Can be triggered even if the pin is configured as output (can be used as ”SW interrupt”).

Options for triggering, INT0 and INT1

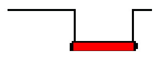
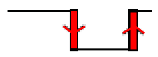
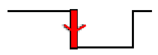
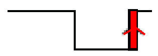
MCUCR							
SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00

ISC01, ISC00 (Interrupt Sense Control bits) These b
external INT0 pin that activates the interrupt, as shown i

```
LDI     R20,0x02    ;falling
OUT     MCUCR,R20
```

ISC01	ISC00		
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.

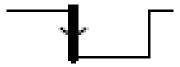
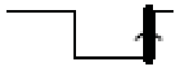
ISC11, ISC10 These bits define the level or edge that activates the INT1 pin.

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.

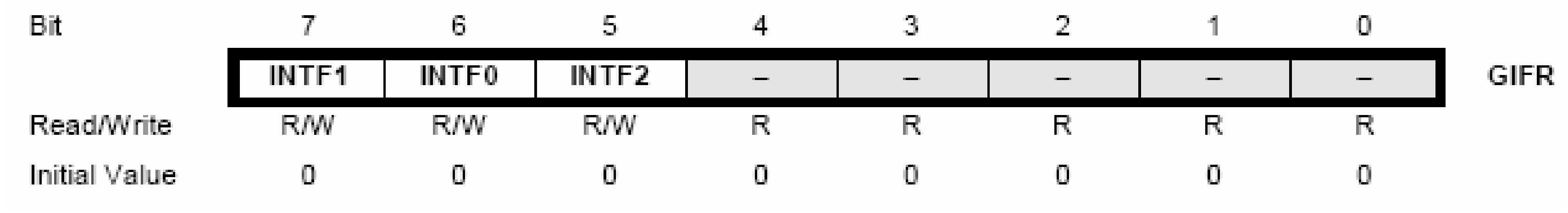
Options of triggering, INT2

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.

(External interrupts: Flags)



Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- Is set when the interrupt is triggered (FLIP-FLOP).
- If global interrupt enabled and locally enabled: Interrupt is started.
- The flag is cleared autimatically in the interrupt routine (or by SW by writing a 1 to the actual bit).
- It is rare that we need to use this register (when the interrupt is enabled).

Test ("socrative.com": Room = MSYS)

We want to enable the Mega32 external interrupts INT0 and INT1 (INT2 shall be disabled).
Moreover both INT0 and INT1 shall trig at "falling edge".
Which initialization is correct ?

- A: GICR = 0b11000000;
  MCUCR = 0b00000101;
- B: GICR = 0b11100000;
  MCUCR = 0b00001010;
- C: GICR = 0b11000000;
  MCUCR = 0b00001010;
- D: GICR = 0b11100000;
  MCUCR = 0b00000101;

Interrupt priority

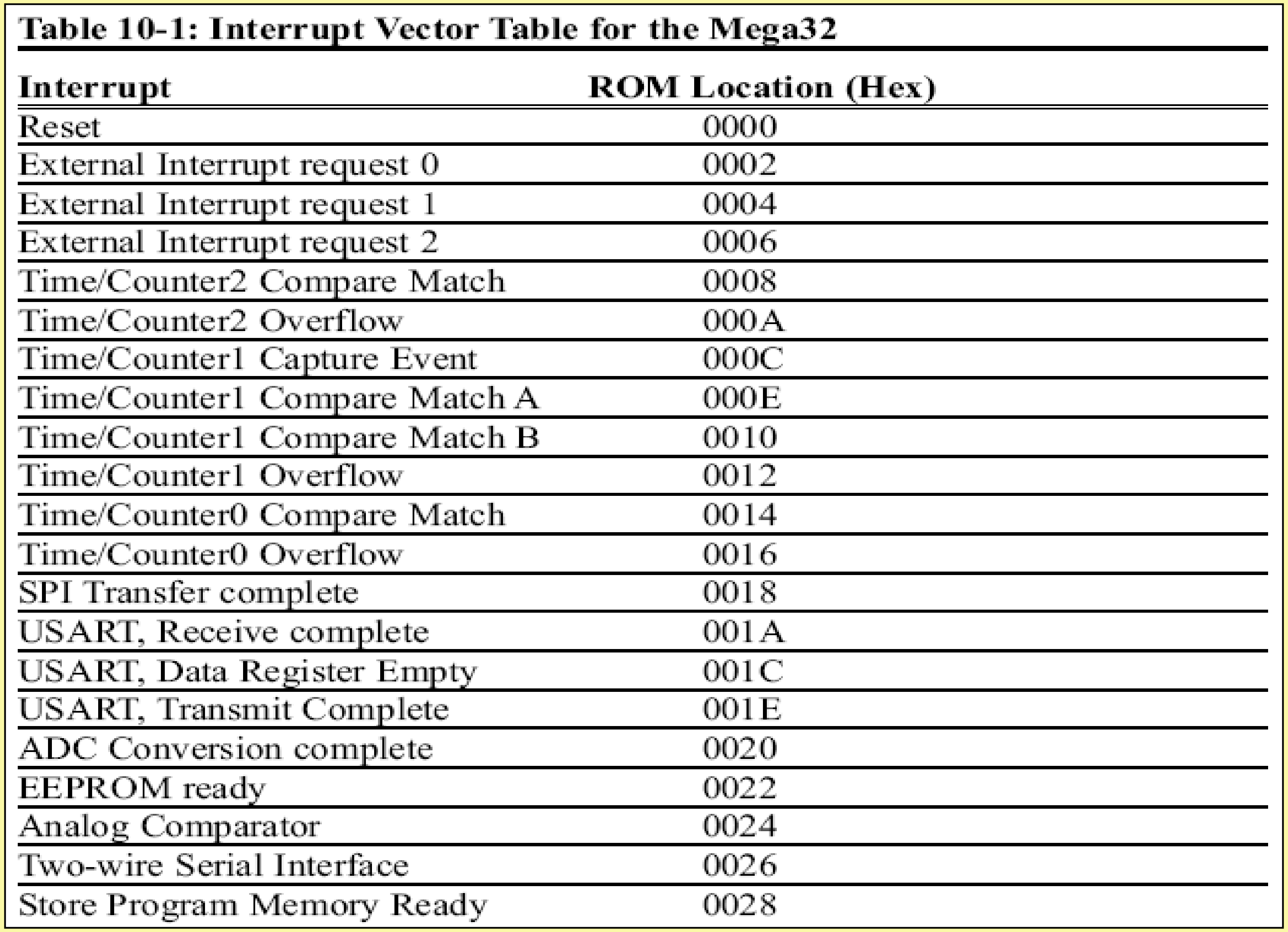
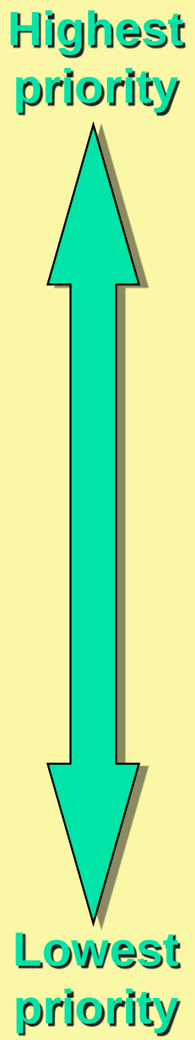


Table 10-1: Interrupt Vector Table for the Mega32

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 (“nesting”)

- The global interrupt flag is cleared (auto-matically), when the ISR is entered. Therefore interrupts are disabled.
- The global interrupt flag is set (auto-matically) when the RETI is executed. Therefore interrupt is re-enabled.
- If you want to enable interrupt nesting, you can – in the beginning of the ISR – execute sei().

Interrupts and resource conflicts

- What can go wrong here ?

```
.INCLUDE "M32DEF.INC"
.ORG        0x0         ;location for reset
            JMP         MAIN
.ORG        0x14        ;Timer0 compare match
            JMP         T0_CM_ISR
;---------main program-----------------------------
.ORG        0x100
MAIN:       LDI         R20,HIGH(RAMEND)
            OUT         SPH,R20
            LDI         R20,LOW(RAMEND)
            OUT         SPL,R20  ;set up stack
            SBI         DDRB,5   ;PB5 = output
            LDI         R20,160
            OUT         OCR0,R20
            LDI         R20,0x09
            OUT         TCCR0,R20
            LDI         R20,(1<<OCIE0)
            OUT         TIMSK,R20
            SEI
            LDI         R20,0xFF
            OUT         DDRC,R20
            OUT         DDRD,R20
            LDI         R20, 0
HERE:       OUT         PORTC,R20
            INC         R20
            JMP         HERE
;-------------------------ISR for Timer0
T0_CM_ISR:
            IN          R20,PIND
            INC         R20
            OUT         PORTD,R20
            RETI
```

Solution 1: Use different registers

- Select other registers (hassle).

```
.INCLUDE "M32DEF.INC"
.ORG        0x0         ;location for reset
            JMP         MAIN
.ORG        0x14        ;Timer0 compare match
            JMP         T0_CM_ISR
;---------main program-----------------------------
.ORG        0x100
MAIN:       LDI         R20,HIGH(RAMEND)
            OUT         SPH,R20
            LDI         R20,LOW(RAMEND)
            OUT         SPL,R20  ;set up stack
            SBI         DDRB,5   ;PB5 = output
            LDI         R20,160
            OUT         OCR0,R20
            LDI         R20,0x09
            OUT         TCCR0,R20
            LDI         R20,(1<<OCIE0)
            OUT         TIMSK,R20
            SEI
            LDI         R20,0xFF
            OUT         DDRC,R20
            OUT         DDRD,R20
            LDI         R20, 0
HERE:       OUT         PORTC,R20
            INC         R20
            JMP         HERE
;-------------------------ISR for Timer0
T0_CM_ISR:
            IN          R21,PIND
            INC         R21
            OUT         PORTD,R21
            RETI
```

Solution 2: “Context saving” (PUSH/POP)

- PUSH the used registers in the beginning of the ISR. POP them back just before the IRET.

```
.INCLUDE "M32DEF.INC"
.ORG      0x0       ;location for reset
          JMP       MAIN
.ORG      0x14      ;Timer0 compare match
          JMP       T0_CM_ISR
;---------main program----------------------------
.ORG      0x100
MAIN:     LDI       R20,HIGH(RAMEND)
          OUT       SPH,R20
          LDI       R20,LOW(RAMEND)
          OUT       SPL,R20  ;set up stack
          SBI       DDRB,5   ;PB5 = output
          LDI       R20,160
          OUT       OCR0,R20
          LDI       R20,0x09
          OUT       TCCR0,R20
          LDI       R20,(1<<OCIE0)
          OUT       TIMSK,R20
          SEI
          LDI       R20,0xFF
          OUT       DDRC,R20
          OUT       DDRD,R20
          LDI       R20, 0
HERE:     OUT       PORTC,R20
          INC       R20
          JMP       HERE
;-------------------------ISR for Timer0
T0_CM_ISR:
          PUSH      R20      ;save R20
          IN        R20,PIND
          INC       R20
          OUT       PORTD,R20
          POP       R20      ;restore R20
          RETI
```

In assembly: Remember to save SREG

- In assembly: Always PUSH/POP the SREG in the ISR, if the flags are changed in the ISR.

```
PUSH    R20
IN      R20,SREG
PUSH    R20
...
POP     R20
OUT     SREG,R20
POP     R20
```

Timer 0 interrupts

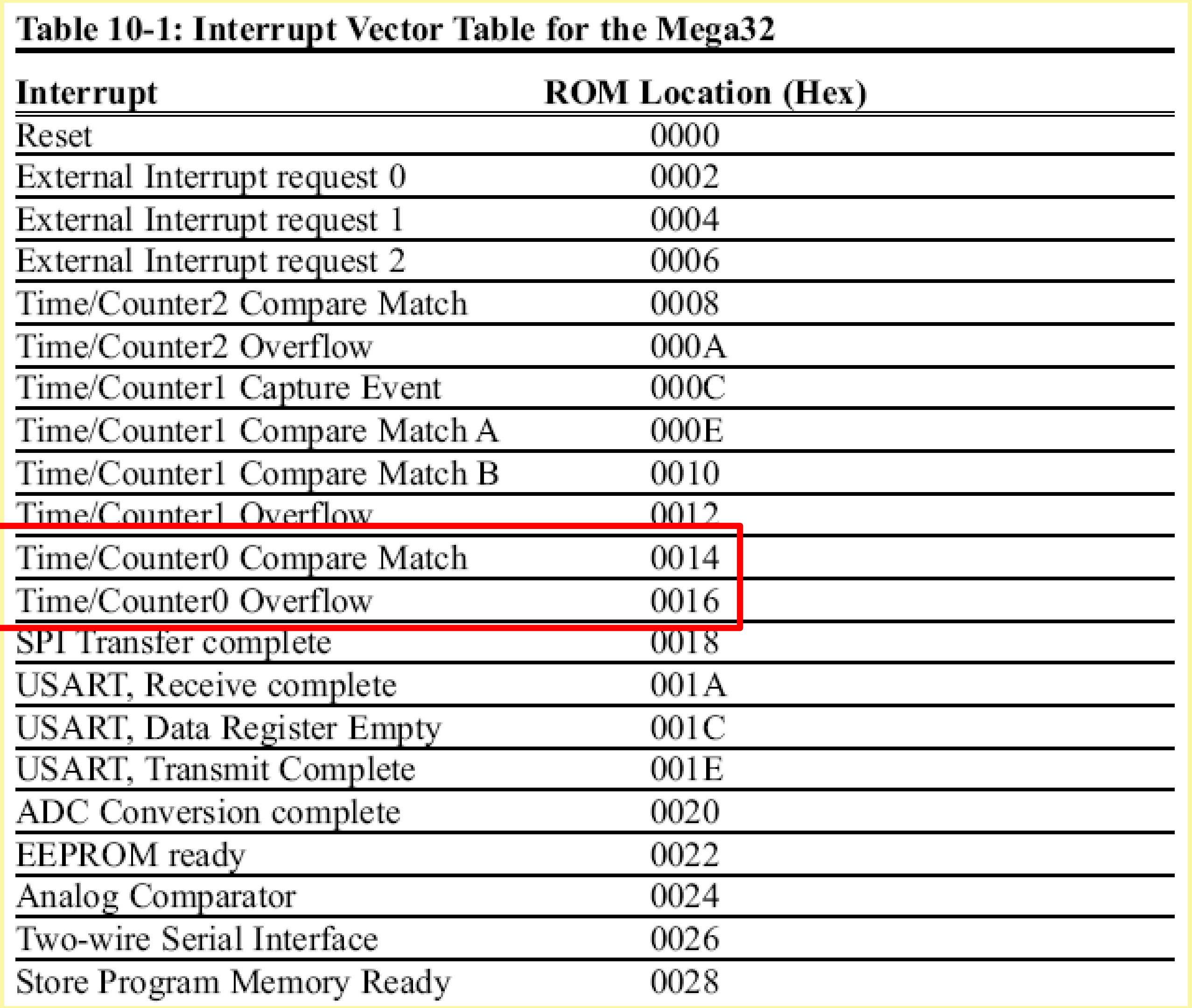
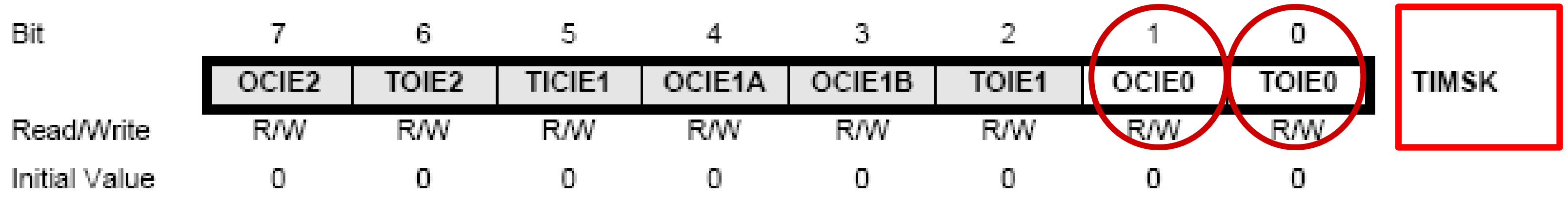


Table 10-1: Interrupt Vector Table for the Mega32

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

Timer 0: Interrupt enables



Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit TOIE0:
  0 => Disable overflow interrupt.
  1 => Enable overflow interrupt.

- Bit OCIE0:
  0 => Disable Compare Match interrupt.
  1 => Enable Compare Match interrupt.

Timer 1 interrupts

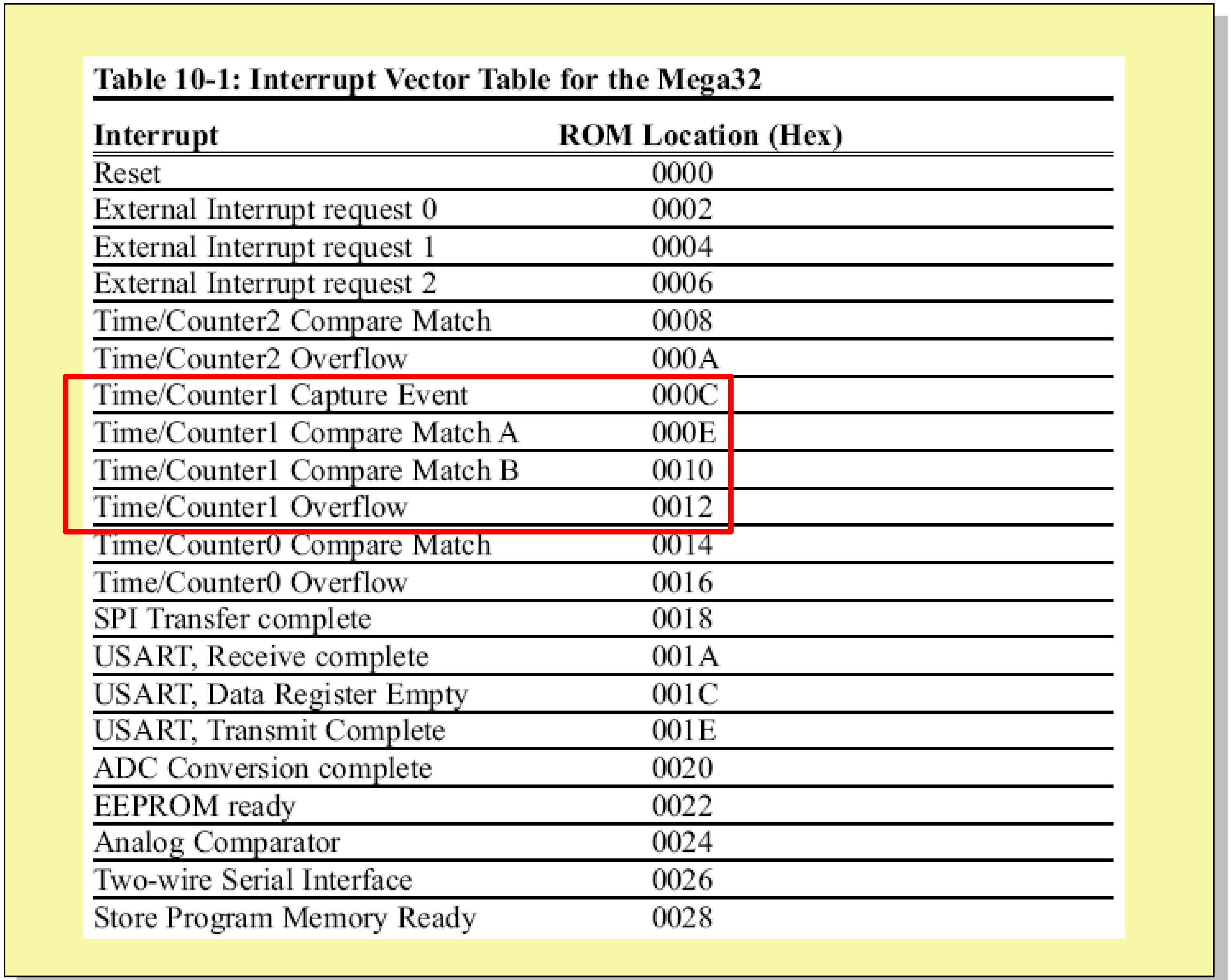
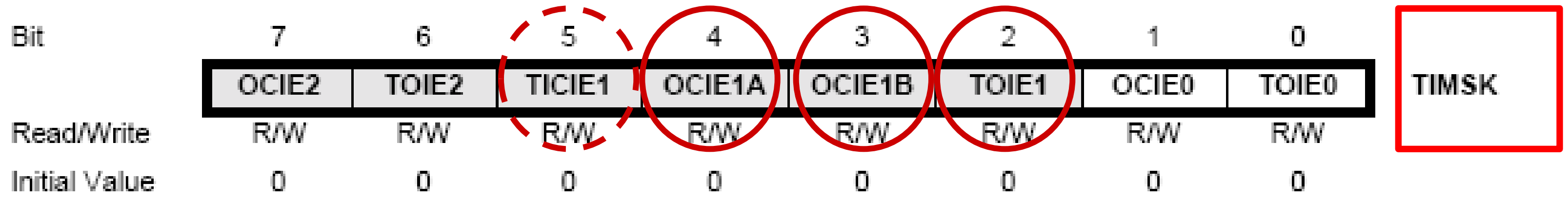


Table 10-1: Interrupt Vector Table for the Mega32

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

Timer 1: Interrupt enables



Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit TOIE1:
  0 => Disable overflow interrupt.
  1 => Enable overflow interrupt.
- Bit OCIE1A:
  0 => Disable Compare Match A interrupt.
  1 => Enable Compare Match A interrupt.
- Bit OCIE1B:
  0 => Disable Compare Match B interrupt.
  1 => Enable Compare Match B interrupt.

Timer 2 interrupts

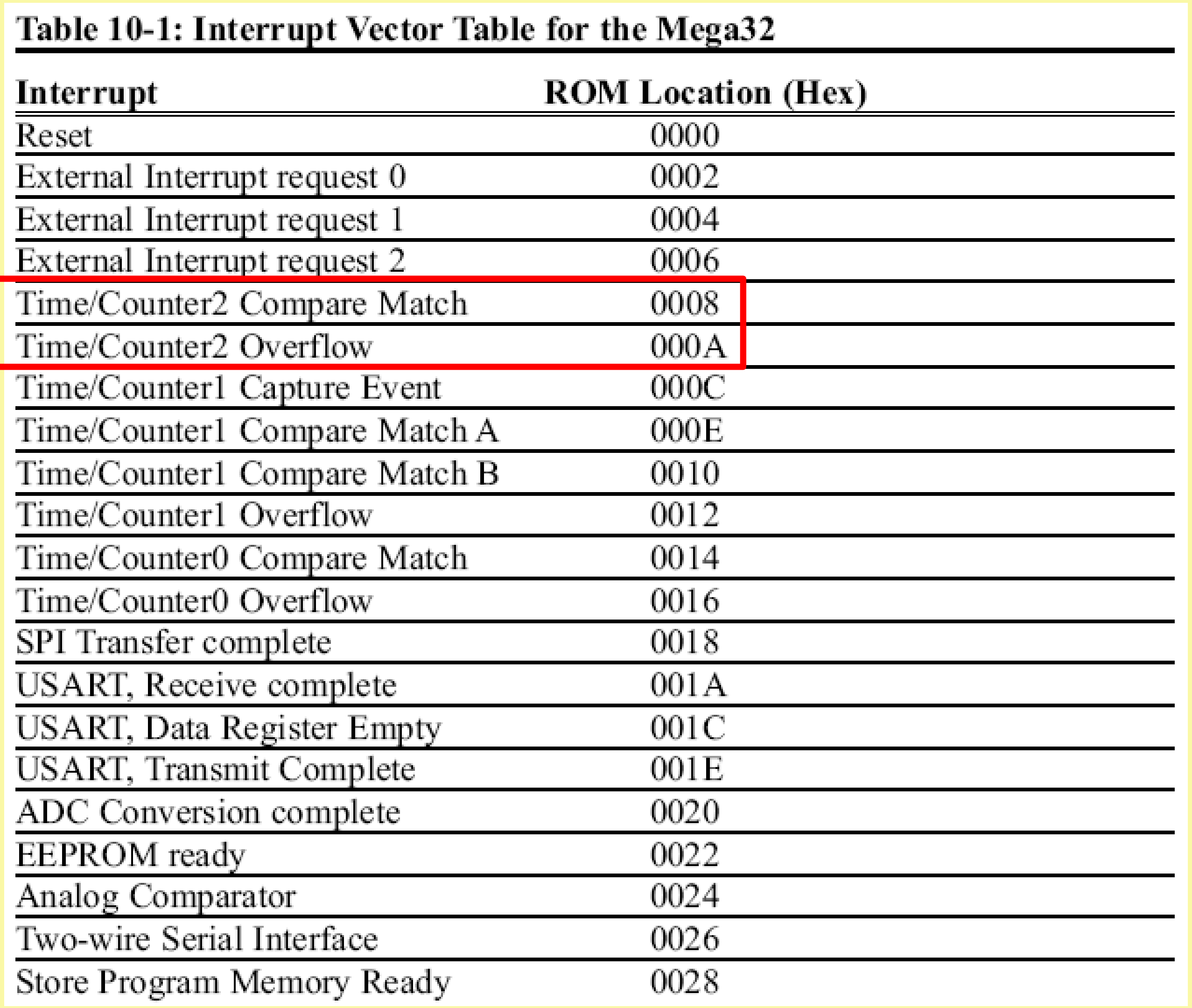


Table 10-1: Interrupt Vector Table for the Mega32

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

Timer 2: Overflow interrupt enable

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit TOIE2:
  0 => Disable overflow interrupt.
  1 => Enable overflow interrupt.

- Bit OCIE2:
  0 => Disable Compare Match interrupt.
  1 => Enable Compare Match interrupt.

Test ("socrative.com": Room = MSYS)

How to enable Timer 1 overflow interrupts ?

- A: TIMSK |= 0b00000001;
  sei();
- B: TIMSK |= 0b00010000;
  sei();
- C: TIMSK |= 0b00000010;
  sei();
- D: TIMSK |= 0b00000100;
  sei();

C: Example 1

- Use Timer0 to generate a square wave signal at pin PORTB.5 – while data at PORTC is transferred to PORTD.

```
#include "avr/io.h"
#include "avr/interru
int main ()
{
        DDRB |= 0x20;
        TCNT0 = -32;
        TCCR0 = 0x01;
        TIMSK = (1<<TO
        sei ();
        DDRC = 0x00;
        DDRD = 0xFF;
        while (1)
          PORTD = PINC
}
ISR (TIMER0_OVF_vect)
{
        TCNT0 = -32;
        PORTB ^= 0x20;
}
```

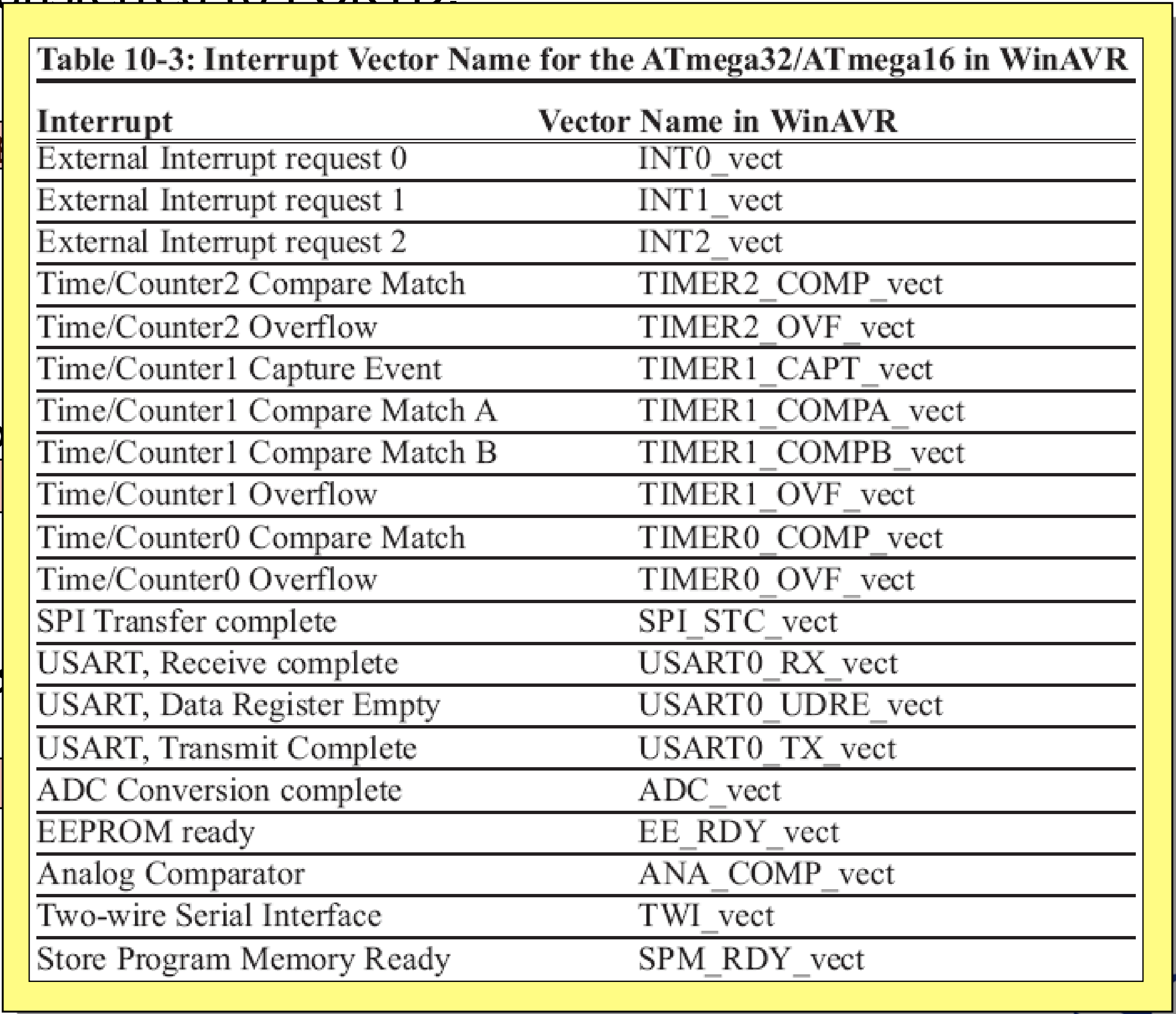


Table 10-3: Interrupt Vector Name for the ATmega32/ATmega16 in WinAVR

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: Example 2

- Assume that pin INT0 is connected to a pushbutton, normally being HIGH. Write a program that toggles PORTC.3, every time the INT0 pin goes LOW. Use the external interrupt in “level-triggered mode”.

```
#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
       sei ();                     //enable interrupts

       while (1);                  //wait here
}

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

Test (”socrative.com”: Room = MSYS)

The external interrupt INT0 is initialized like this:

```
GICR |= 0b01000000;
MCUCR = 0b00000011;
sei( );
```

What signal at the INT0 pin will start an interrupt ?

- A: When the signal is low (0 volt)
- B: When the signal changes state
- C: When the signal changes from high to low
- D: When the signal changes from low to high

End of lesson 16