Microprocessors and Microcontrollers

CSE 315

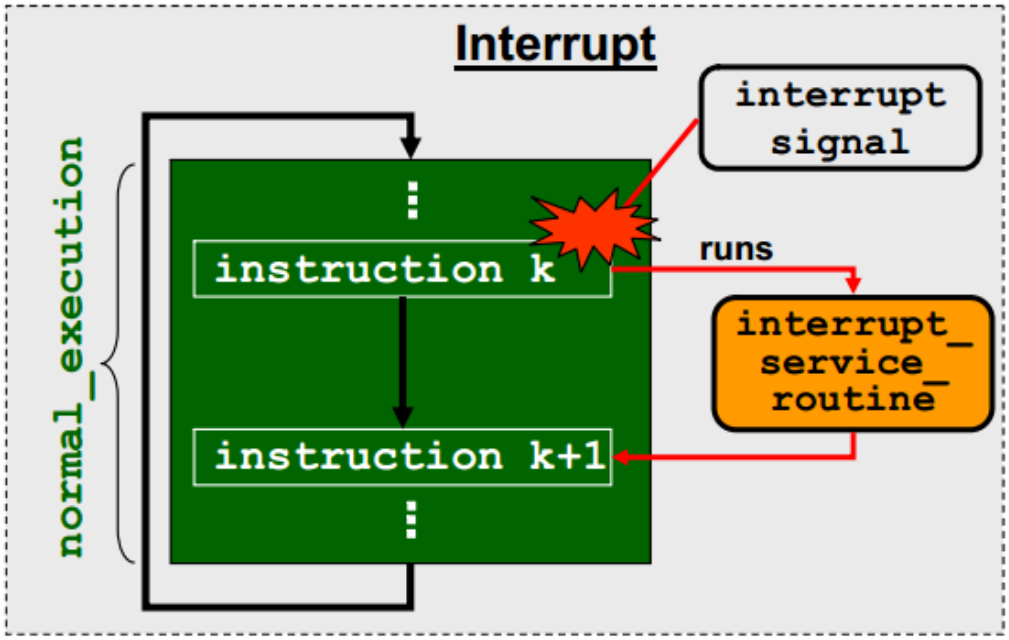
Md. Iftekharul Islam Sakib

INTERRUPT

Interrupts vs. Polling

Polling

```
while (1){
    get_device_status;
    if (service_required){
        service_routine;
    }
    normal_execution;
}
```



Interrupts vs. Polling

- Using polling
  - the CPU must continually check the device's status.
- Using interrupt
  - A device will send an interrupt signal when needed.
  - In response, the CPU will perform an interrupt service routine, and then resume its normal execution.

Interrupts vs. Polling

- Efficiency
- Monitoring several devices
- Priority

Interrupt execution sequence

A device issues an interrupt

CPU finishes the current instruction

CPU acknowledges the interrupt

CPU saves its states and PC onto stack

CPU loads the address of ISR onto PC

CPU executes the ISR

CPU retrieves its states and PC from stack

Normal execution resumes

ATmega16 interrupt subsystem

- The ATmega16 has total 21 interrupts
- We focus on 16 of them
  - 3 external interrupts
  - 8 timer interrupts
  - 3 serial port interrupts
  - 1 ADC interrupt
  - 1 SPI interrupt

ATmega16 interrupt subsystem

- The ATmega16 has total 21 interrupts

- There are 5 others
  - 1 reset interrupt
  - 1 analogue comparator interrupt
  - 1 TWI interrupt
  - 2 memory interrupts

Complete List

Vector No.	Program Address	Interrupt vector name	Description
1	$000	RESET_vect	Reset
2	$002	INT0_vect	External Interrupt Request 0
3	$004	INT1_vect	External Interrupt Request 1
4	$006	TIMER2_COMP_vect	Timer/Counter2 Compare Match
5	$008	TIMER2_OVF_vect	Timer/Counter2 Overflow
6	$00A	TIMER1_CAPT_vect	Timer/Counter1 Capture Event
7	$00C	TIMER1_COMPA_vect	Timer/Counter1 Compare Match A
8	$00E	TIMER1_COMPB_vect	Timer/Counter1 Compare Match B
9	$010	TIMER1_OVF_vect	Timer/Counter1 Overflow
10	$012	TIMER0_OVF_vect	Timer/Counter0 Overflow
11	$014	SPI_STC_vect	Serial Transfer Complete
12	$016	USART_RXC_vect	USART, Rx Complete
13	$018	USART_UDRE_vect	USART Data Register Empty
14	$01A	USART_TXC_vect	USART, Tx Complete
15	$01C	ADC_vect	ADC Conversion Complete
16	$01E	EE_RDY_vect	EEPROM Ready
17	$020	ANA_COMP_vect	Analog Comparator
18	$022	TWI_vect	2-wire Serial Interface
19	$024	INT2_vect	External Interrupt Request 2
20	$026	TIMER0_COMP_vect	Timer/Counter0 Compare Match
21	$028	SPM_RDY_vect	Store Program Memory Ready

Complete List

Vector No.	Program Address	Interrupt vector name	Description
1	$000	RESET_vect	Reset
2	$002	INT0_vect	External Interrupt Request 0
3	$004	INT1_vect	External Interrupt Request 1
4	$006	TIMER2_COMP_vect	Timer/Counter2 Compare Match
5	$008		
6	$00A		
7	$00C		
8	$00E		
9	$010		
10	$012		
11	$014		
12	$016		
13	$018		
14	$01A		
15	$01C		
16	$01E		
17	$020		
18	$022	TWI_vect	2-wire Serial Interface
19	$024	INT2_vect	External Interrupt Request 2
20	$026	TIMER0_COMP_vect	Timer/Counter0 Compare Match
21	$028	SPM_RDY_vect	Store Program Memory Ready

Vector No.

- An interrupt with a lower 'Vector No' will have a higher priority.
- INT0 has a higher priority then INT1 and INT2

Complete List

Vector No.	Program Address	Interrupt vector name	Description
1	$000	RESET_vect	Reset
2	$002	INT0_vect	External Interrupt Request 0
3	$004	INT1_vect	External Interrupt Request 1
4	$006	TIMER2_COMP_vect	Timer/Counter2 Compare Match
5	$008	TIMER2_OVF_vect	Timer/Counter2 Overflow
6	$00A	TIMER1_CAPT_vect	Timer/Counter1 Capture Event
7	$00C		
8	$00E		
9	$010		
10	$012		
11	$014		
12	$016		
13	$018		
14	$01A		
15	$01C		
16	$01E		
17	$020		
18	$022		
19	$024		
20	$026		
21	$028		

Program Address.

- The fixed memory location for a given interrupt handler.
- E.g., in response to interrupt INT0, CPU runs instruction at $002.
- Usually the instruction is JMP address (address of ISR)

Complete List

Vector No.	Program Address	Interrupt vector name	Description
1	$000	RESET_vect	Reset
2	$002	INT0_vect	External Interrupt Request 0
3	$004	INT1_vect	External Interrupt Request 1
4	$006	TIMER2_COMP_vect	Timer/Counter2 Compare Match
5	$008	TIMER2_OVF_vect	Timer/Counter2 Overflow
6	$00A	TIMER1_CAPT_vect	Time
7	$00C	TIMER1_COMPA_vect	Time
8	$00E	TIMER1_COMPB_vect	Time
9	$010	TIMER1_OVF_vect	Time
10	$012	TIMER0_OVF_vect	Time
11	$014	SPI_STC_vect	Se
12	$016	USART_RXC_vect	USA
13	$018	USART_UDRE_vect	USA
14	$01A	USART_TXC_vect	USA
15	$01C	ADC_vect	ADC
16	$01E	EE_RDY_vect	EEP
17	$020	ANA_COMP_vect	Analog Comparator
18	$022	TWI_vect	2-wire Serial Interface
19	$024	INT2_vect	External Interrupt Request 2
20	$026	TIMER0_COMP_vect	Timer/Counter0 Compare Match
21	$028	SPM_RDY_vect	Store Program Memory Ready

Vector names to be used with ISR

Complete List

Vector No.	Program Address	Interrupt vector name	Description
1	$000	RESET_vect	Reset
2	$002	INT0_vect	External Interrupt Request 0
3	$004	INT1_vect	External Interrupt Request 1
4	$006	TIMER2_COMP_vect	Timer/Counter2 Compare Match
5	$008	TIMER2_OVF_vect	Timer/Counter2 Overflow
6	$00A	TIMER1_CAPT_vect	Timer/Counter1 Capture Event
7	$00C	TIMER1_COMPA_vect	Timer/Counter1 Compare Match A
8	$00E	TIMER1_COMPB_vect	Timer/Counter1 Compare Match B
9	$010	TIMER1_OVF_vect	Timer/Counter1 Overflow
10	$012	TIMER0_OVF_vect	Timer/Counter0 Overflow
11	$014	SPI_STC_vect	Serial Transfer Complete
12			USART, Rx Complete
13			USART Data Register Empty
14			USART, Tx Complete
15			ADC Conversion Complete
16			EEPROM Ready
17			Analog Comparator
18			2-wire Serial Interface
19			External Interrupt Request 2
20			Timer/Counter0 Compare Match
21	$028	SPM_RDY_vect	Store Program Memory Ready

Description

Steps to program an interrupt in C

1. Include header file <avr\interrupt.h>.
2. Use C macro ISR() to declare the interrupt handler and update IVT.
3. Configure details about the interrupt by setting relevant registers.
4. Enable the specific interrupt.
5. Enable the interrupt subsystem globally using sei().

ISR

- Basic Construct

```
ISR(interrupt vector name)
{
//to do logic
}
```

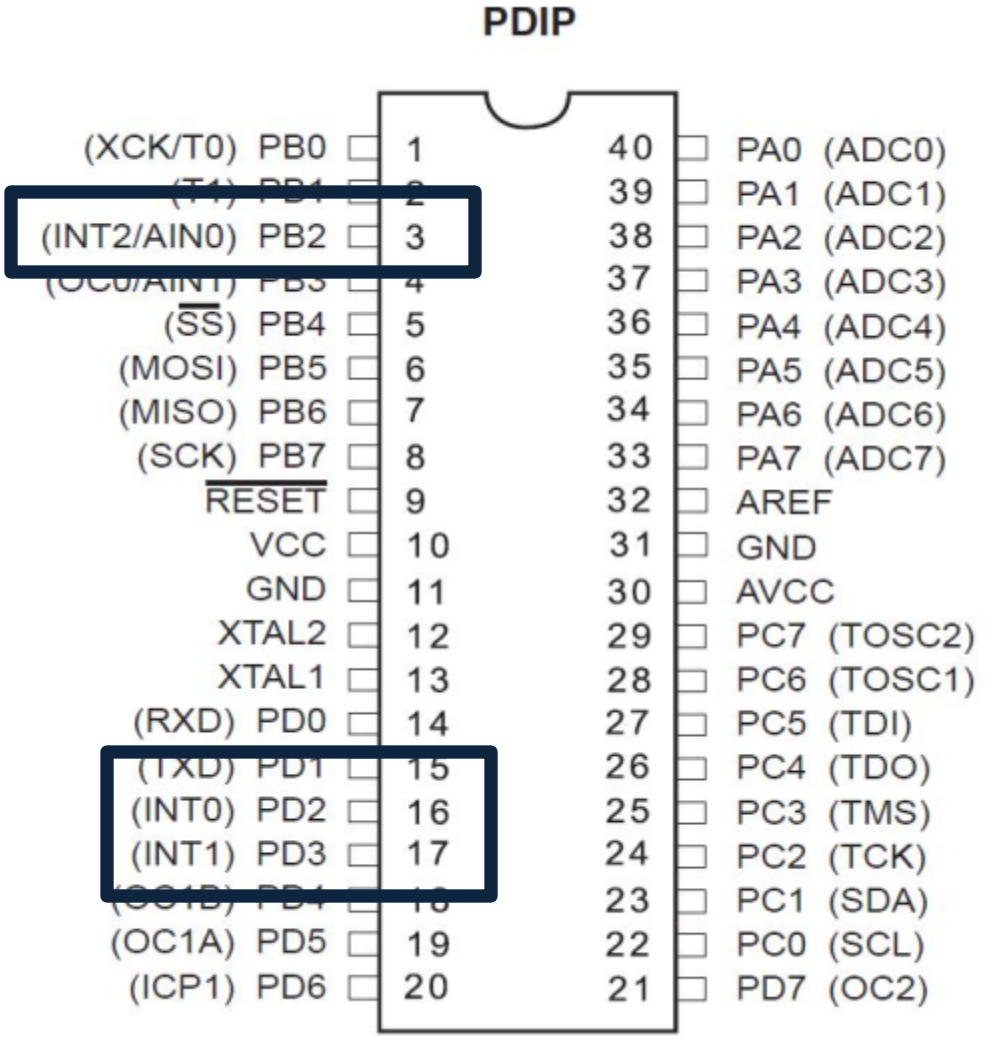

ISR

- To handle external interrupt 1

```
ISR(INT1_vect)
{
    //to do logic
}
```


External Interrupts

- Three external interrupts
  - INT 0
  - INT 1
  - INT 2



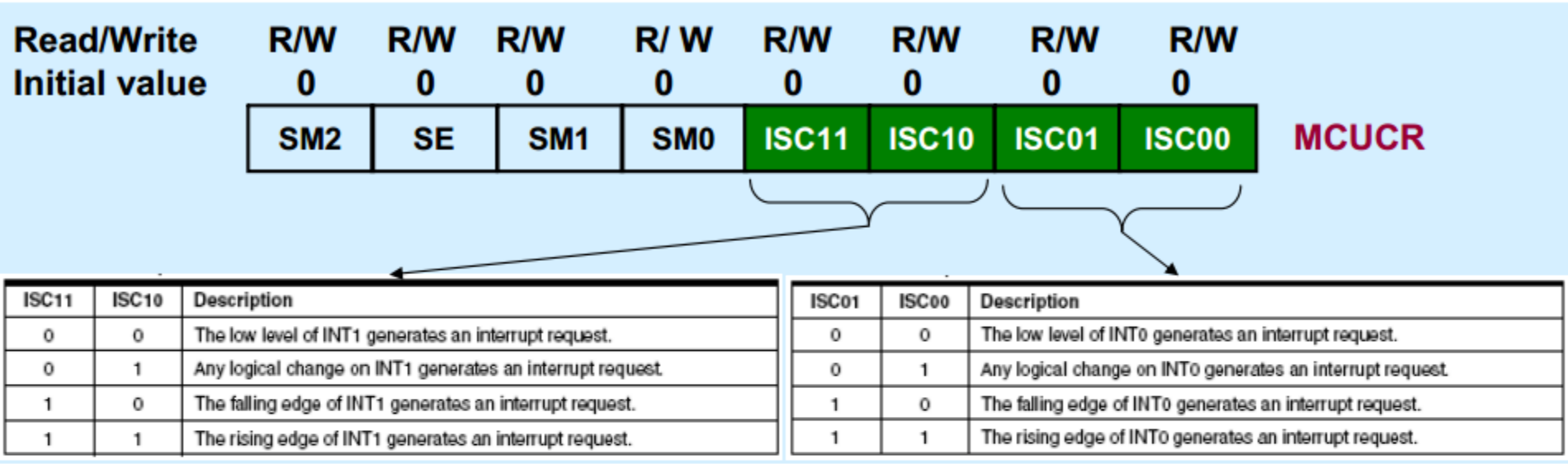
External Interrupts

- Key steps in using external interrupts.
  - Specifying what types of event will trigger the interrupt (Step 3)
  - Enabling the interrupt (Step 4)

Specifying Events that Trigger Interrupt (Step 3)

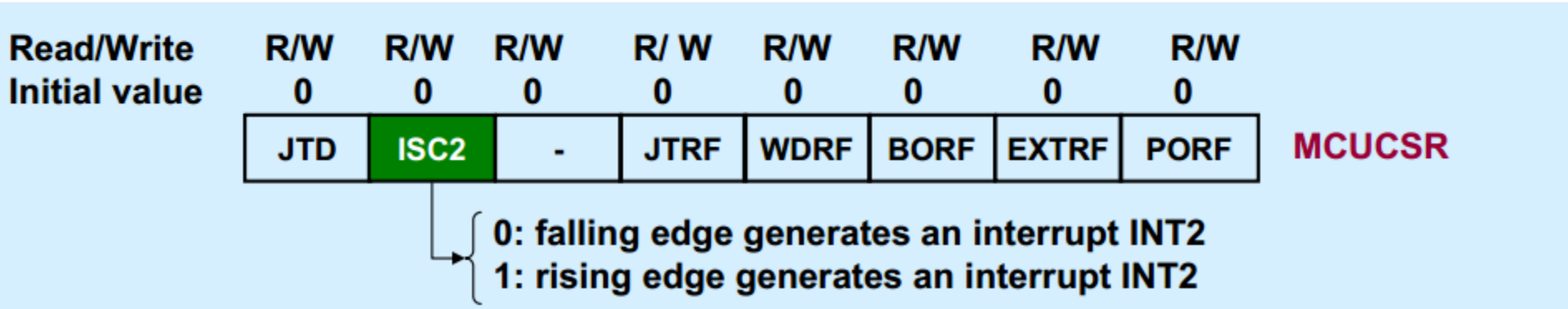
- 2 registers
  - MCU Control Register (For INT0 and INT 1)
  - MCU Control and Status Register (For INT2)

Specifying Events that Trigger Interrupt (Step 3)

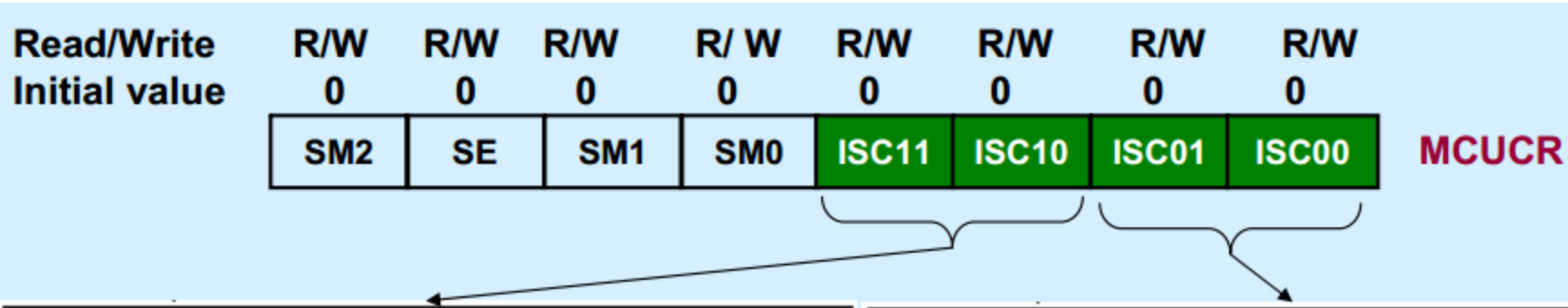


ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

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.

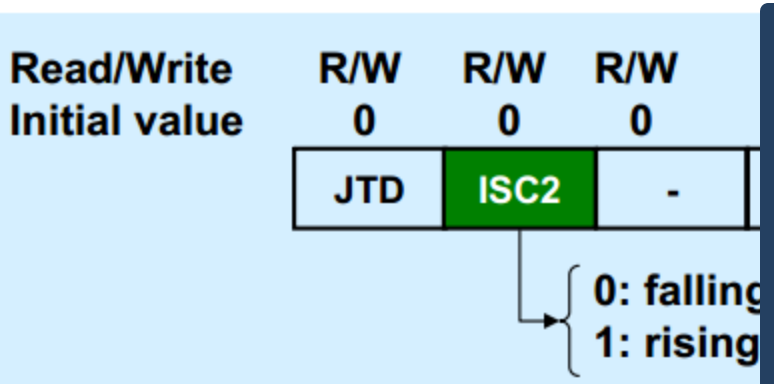


Specifying Events that Trigger Interrupt (Step 3)



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.

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.
	0	The falling edge of INT0 generates an interrupt request.
1		The rising edge of INT0 generates an interrupt request.



To specify that INT1 is triggered on any change in pin D.3

```
MCUCR = (1<<ISC10);
```

Enabling the interrupt (Step 4)

- GICR (General Interrupt Control Register) register is used to enable external interrupts
- To enable Interrupt 1
  - `GICR = (1<<INT1);`
- INT1 is defined in *io.h*

	INT1	INT0	INT2	-	-	-	IVSEL	IVSEL
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

TOGGLE THE CONTENT OF PORT B, WHENEVER A CERTAIN SENSOR CONNECTED TO YOUR SYSTEM GOES TO LOW STATE

C Code

```
#include <avr/io.h>
#include <avr/interrupt.h> //STEP1

ISR(INT1_vect)//STEP2
{
   PORTB = ~PORTB;
}
int main(void)
{
   DDRB = 0xFF;
   PORTB = 0b01010101;
   GICR = (1<<INT1); //STEP3
   MCUCR = MCUCR & 0b11110011;//STEP4
   sei();//STEP5
    while(1);
}
```


Disabling global interrupt

- You typically turn off interrupts when you are doing a task that should not be interrupted.
- One example is reading/writing 16-bit values like TCNT1. (Details will be discussed when we study Timer)
- The *cli()* macro is used to disable all interrupts by clearing the global interrupt mask.

Nested Interrupts

- The global interrupt is disabled by hardware when an interrupt has occurred
  - so by default nested interrupt is disabled in ATmega32
- Global interrupt is set again by the RETI instruction to enable subsequent interrupts. This is done automatically by the compiler.
- However to enable nested interrupts it can be enabled manually with the sei() in the ISR.

ISR Usage

- understand how often the interrupt occurs
- understand how much time it takes to service each interrupt
- make sure there is enough time to service all interrupts and to still get work done in the main loop
- Keep ISRs Short and Simple. (short = short time, not short code length)
  - Do only what has to be done. Long ISRs may preclude others from being run

Use of volatile

- As ISRs are not called from main or any other function, it can not take argument or return any values
- We have to use global variables
- We must use volatile to declare such variables
  - why ??

Use of volatile

- Compilers can optimize away some variables
- It does so when it sees that the variable cannot be changed within the scope of the code it is looking at
- variables changed by the ISR are outside the scope of main()
  - thus, they get optimized away

Use of volatile

```
volatile uint8_t tick; //keep tick out of regs!

ISR(TIMER1_OVF_vect){
tick++; //increment my tick count
}

main()
{
    while(tick == 0x00)
    {
        bla, bla, bla...
    }
}
```

Without volatile volatile modifier, -02 optimization removes tick because nothing in while loop can ever change tick

Resource

- ATMega Datasheet
- http://web.engr.oregonstate.edu/~traylor/ece473/lectures/interrupts.pdf
- http://www.avrfreaks.net/forum/nested-interrupts-2