**What are interrupts?**

**Interrupts** are basically events that require immediate attention by the microcontroller. When an interrupt event occurs the microcontroller pause its current task and attend to the interrupt by executing an **Interrupt Service Routine (ISR)** at the end of the ISR the microcontroller returns to the task it had pause and continue its normal operations.

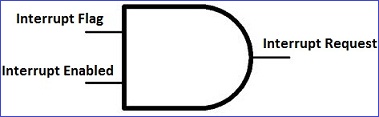
In order for the microcontroller to respond to an interrupt event the interrupt feature of the microcontroller must be enabled along with the specific interrupt. This is done by setting the **Global Interrupt Enabled** bit and the**Interrupt Enable** bit of the specific interrupt.

*Interrupt Service Routine or Interrupt Handler*  
An **Interrupt Service Routine (ISR)** or **Interrupt Handler** is a piece of code that should be execute when an interrupt is triggered. Usually each enabled interrupt has its own ISR. In AVR assembly language each ISR **MUST**end with the **RETI** instruction which indicates the end of the ISR.

*Interrupt Flags and Enabled bits*  
Each interrupt is associated with two (2) bits, an **Interrupt Flag Bit** and an **Interrupt Enabled Bit**. These bits are located in the I/O registers associated with the specific interrupt:

* The **interrupt flag** bit is set whenever the interrupt event occur, whether or not the interrupt is enabled.
* The **interrupt enabled** bit is used to enable or disable a specific interrupt. Basically is tells the microcontroller whether or not it should respond to the interrupt if it is triggered.

In summary basically both the **Interrupt Flag** and the **Interrupt Enabled** are required for an interrupt request to be generated as shown in the figure below.



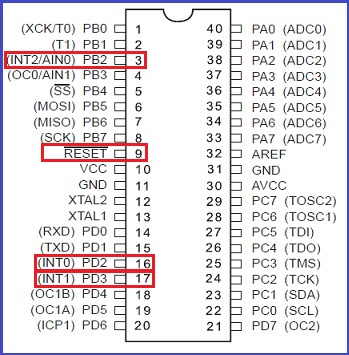
*Global Interrupt Enabled Bit*  
Apart from the enabled bits for the specific interrupts the global interrupt enabled bit **MUST** be enabled for interrupts to be activated in the microcontoller.

For the AVR 8-bits microcontroller this bit is located in the **Status I/O Register (SREG)**. The Global Interrupt Enabled is bit 7, the **I** bit, in the SREG.  
AVR Microcontroller SREG

**Interrupt sources provided with the AVR microcontroller**

The AVR 8-bits microcontroller provide both internal and external interrupt sources. The internal interrupts are associated with the microcontroller's peripherals. That is the **Timer/Counter, Analog Comparator, etc**. The external interrupts are triggered via external pins. The figure below shows the pins, on which the external interrupts can be triggered, for an AVR 8-bit microcontroller. On this microcontroller there are four (4) external interrupts:

1. The **RESET** interrupt - Triggered from pin 9.
2. **External Interrupt 0 (INT0)** - Triggered from pin 16.
3. **External Interrupt 1 (INT1)** - Triggered from pin 17.
4. **External Interrupt 2 (INT2)** - Triggered from pin 3.



**Very Important**  
When writing assembly codes for your AVR microcontroller utilizing the interrupt feature the following **MUST** be observed:

* The interrupt **MUST** be enabled by setting its enabled bit in the appropriate I/O register.
* The Global Interrupt bit, the **I** bit, in the microcontroller's status register (SREG) **MUST** also be enabled.
* The stack **MUST** be initialized. When an interrupt is being service the microcontroller need to store critical information on the stack and so it must be initialized.
* The Interrupt Service Routine (ISR) **MUST** end with the **RETI** instruction, which indicates the end of the ISR. The microcontroller needs to know when it reaches the end of the ISR so it can return to its previous task.

**Steps taken in servicing an interrupt**

Upon the triggering of an interrupt the following sequence is followed by the microcontroller providing that the both the specific interrupt and global interrupts are enabled in the microcontroller:

1. The microcontroller completes the execution of the current instruction, clears the I bit and stores the address of the next instruction that should have been executed (the content of the PC) on the stack.
2. The interrupt vector of the triggered interrupt is then loaded in the PC and the microcontroller starts execution from that point up until is reaches a **RETI** instruction.
3. Upon the the execution of the **RETI** instruction the address that was stored on the stack in **step 1** is reloaded in the PC and the I bit is re-enabled.
4. The microcontroller then start executing instructions from that point. That is the point that it left off when the interrupt was triggered.

**Important Notes:**  
As it relates to AVR microcontrollers - An **interrupt vector** is the memory address of an interrupt handler. The interrupt vector for each interrupt provided by the AVR microcontrollers can be found in its datasheet. The table below is an extract from the [interrupt section](http://www.avr-tutorials.com/sites/default/files/ATMega8515%20Interrupts.pdf) of the ATMega16 datasheet and gives the interrupt vectors for the interrupts provided with this microcontroller.

Please note here that the interrupt vectors are apart of the microcontroller's program memory. As such when utilizing interrupts this section of memory should be reserved to store pointers to interrupt handlers and not to store regular programs. For the ATMega16 microcontroller to ensure that regular programs are not stored in this section of program memory insert the following line is your AVR assembly code.

|  |
| --- |
| **ATMega16 Interrupt Vector Table** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector #** | **Program Memory Address** | **Source** | **Definition** |
| 1 | $000 | RESET | External Pin, Power-on Reset, Brown-out  Reset, Watchdog Reset, and JTAG AVR  Reset |
| 2 | $002 | INT0 | External Interrupt Request 0 |
| 3 | $004 | INT1 | External Interrupt Request 1 |
| 4 | $006 | TIMER2 COMP | Timer/Counter2 Compare Match |
| 5 | $008 | TIMER2 OVF | Timer/Counter2 Overflow |
| 6 | $00A | TIMER1 CAPT | Timer/Counter1 Capture Event |
| 7 | $00C | TIMER1 COMPA | Timer/Counter1 Compare Match A |
| 8 | $00E | TIMER1 COMPB | Timer/Counter1 Compare Match B |
| 9 | $010 | TIMER1 OVF | Timer/Counter1 Overflow |
| 10 | $012 | TIMER0 OVF | Timer/Counter0 Overflow |
| 11 | $014 | SPI,STC | Serial Transfer Complete |
| 12 | $016 | USART, RXC | USART, Rx Complete |
| 13 | $018 | USART, UDRE | USART Data Register Empty |
| 14 | $01A | USART, TXC | USART, Tx Complete |
| 15 | $01C | ADC | ADC Conversion Complete |
| 16 | $01E | EE\_RDY | EEPROM Ready |
| 17 | $020 | ANA\_COMP | Analog Comparator |
| 18 | $022 | TWI | Two-wire Serial Interface |
| 19 | $024 | INT2 | External Interrupt Request 2 |
| 20 | $026 | TIMER0 COMP | Timer/Counter0 Compare Match |
| 21 | $028 | SPM\_RDY | Store Program Memory Ready |

AVR Tutorials hope this AVR interrupt tutorial was benificial to you and looks forward to your continued visit for all your microcontroller tutorial needs.

(This tutorial is taken from [link](http://www.avr-tutorials.com/interrupts/about-avr-8-bit-microcontrollers-interrupts))