AVR INTERRUPTS

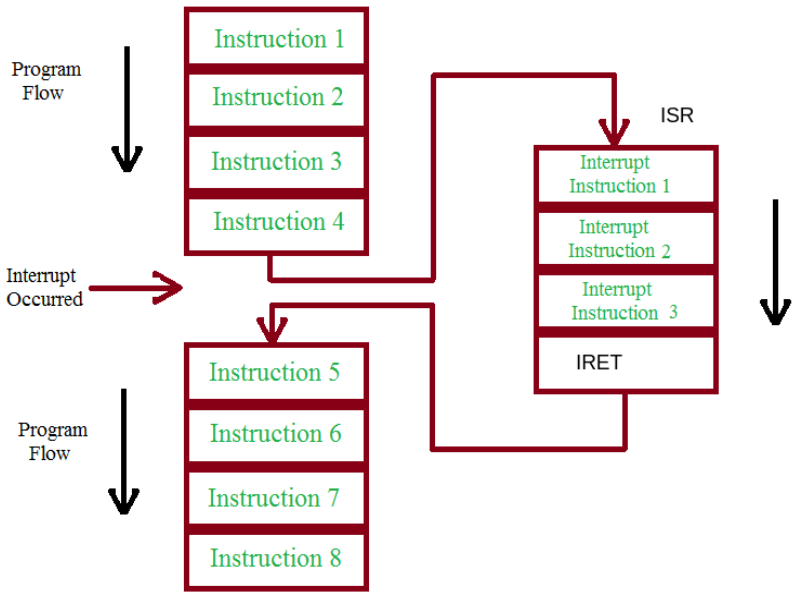
Interrupt is a signal that causes to temporarily stop the normal execution of a program and execute some other program associated with that interrupt signal. This signal can be generated from an external device or from some internal units of the processor/controller. The program associated with an interrupt signal is called Interrupt Service Routine (ISR) or interrupt handler. For every interrupt, there is a fixed location in memory to contain the starting address of its ISR. The group of memory locations to hold the addresses of ISRs is called the interrupt vector table (IVT). A JMP instruction is placed in the vector table to point to the address of the ISR. A part of IVT of ATMega32 is shown below:

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

Steps in executing an interrupt:

1. It finishes the instruction it is currently executing and saves the address of the next instruction (program counter) on the stack.
2. It jumps to the fixed location shown in the interrupt vector table. From there the controller goes to the ISR.
3. The microcontroller starts to execute the interrupt service subroutine until it reaches the last instruction, which is RETI (return from interrupt).
4. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted, by popping the PC from stack.
5. The microcontroller resumes to execute the previous program.

The following diagram illustrate the interrupt mechanism:



Sources of Interrupts:

The sources of interrupts can be external or internal. If the interrupt signal is arrived from other devices, it is called external interrupts. There are three external interrupts in ATmega32 – INT0, INT1 and INT2. They are located on pins PD2, PD3, and PB2, respectively. Devices can interrupt the microcontroller through this pins. Internal interrupt signals are triggered from the internal functional units of the microcontroller, such as timers, USART, ADC etc. Thus, some of the sources of interrupts are:

- Timer overflow interrupts
- Timer Compare match interrupts
- External hardware interrupts (INT0, INT1, INT2)
- USART Interrupts
- SPI (Serial Peripheral Interface) interrupts
- ADC (Analog-to-Digital Converter) interrupts

Enabling and disabling an interrupt:

Interrupts can be enabled or disabled. If an interrupt is disabled (masked), the controller will not respond to such interrupts. Each interrupt has an enable bit. For all the interrupts, there is a global enable bit. This global enable bit is the D7 bit (I bit) of SREG (Status) register. The microcontroller will respond to an interrupt only when both its individual enable bit and the global enable bit are set. In other words, if the global enable bit is not set, the controller will not respond to interrupts even when its individual enable bit is set. So to enable an interrupt:

1. Enable (set) the global interrupt enable bit.
2. Enable (set) the individual enable bit.

To enable/disable the global interrupt, we use the C functions:

- sei() : To enable global interrupt
- cli() : To disable global interrupted

To enable/disable individual interrupt, we have to know the registers where the particular bit is placed. For example, the Timer related interrupt bits are placed in TIMSK register, as shown below.

D7							D0
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

For example, to enable Timer0 overflow interrupt, we set the D0 bit by writing:

```
TIMSK=0X01;
```

Interrupt Priority in AVR

- If two interrupts are activated at the same time, the interrupt with the higher priority is served first.
- The priority of each interrupt is related to the address of that interrupt in the interrupt vector.
- The interrupt that has a lower address, has a higher priority.
- For example, the address of external interrupt 0 is 2, while the address of external interrupt 2 is 6; thus, external interrupt 0 has a higher priority, and if both of these interrupts are activated at the same time, external interrupt 0 is served first.

Interrupt latency:

- The time from the moment an interrupt is activated to the moment the CPU starts to execute the task is called the interrupt latency
- This latency is 4 machine cycle times
- During this time the PC register is pushed on the stack and the I bit of the SREG register clears, causing all the interrupts to be disabled

Interrupt Programming in C

To manage interrupts using C programs, follow the steps :

1. Include the interrupt header file: `#include <avr/interrupt.h>`
2. Enable the global interrupt using the function: `sei();`
3. Define the ISR. It has the following strutcture.

```
ISR(interrupt vector name)
 {
   //our program
 }
```

Each interrupt has a predefined vector name. We have to use that name for the particular interrupt. Following are some of the vector names.

Interrupt	Vector name
Timer/Counter0 Overflow	TIMER0_OVF_vect
Timer/Counter0 Compare Match	TIMER0_COMP_vect
Timer/Counter1 Overflow	TIMER1_OVF_vect
Timer/Counter2 Compare Match	TIMER2_COMP_vect
External Interrupt request 0	INT0_vect
External Interrupt request 1	INT1_vect
External Interrupt request 2	INT2_vect

For example, the following ISR is defined for the Timer0 compare match interrupt:

```
ISR (TIMER0_COMP_vect)
{
    //Code for the ISR
}
```

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

```
#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
}
```

Qn: 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.

```
#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
}
```