# INTRODUCTION TO 8051 INTERRUPTS

Interrupts are the events that temporarily suspend the main program, pass the control to the external sources and execute their task. It then passes the control to the main program where it had left off.

8051 has 5 interrupt signals, i.e. INT0, TFO, INT1, TF1, and RI/TI. Each interrupt can be enabled or disabled by setting bits of the IE register and the whole interrupt system can be disabled by clearing the EA bit of the same register.

In the 8051 microcontroller, interrupts are an essential feature that allows the microcontroller to respond to external events or internal conditions promptly. An interrupt is a signal that interrupts the normal execution of the program and diverts the microcontroller's attention to execute a specific set of instructions called an Interrupt Service Routine (ISR). The ISR handles the interrupt event and performs the necessary tasks before returning to the main program.

The 8051 microcontroller supports two types of interrupts: hardware interrupts and software interrupts.

**Hardware Interrupts:** The 8051 microcontroller has five hardware interrupt sources that can trigger an interrupt. These interrupts are:

1. External Interrupt 0 (INT0): This interrupt is triggered by an external signal on the INT0 pin (P3.2).
2. External Interrupt 1 (INT1): This interrupt is triggered by an external signal on the INT1 pin (P3.3).

**Software Interrupts:** The 8051 microcontroller also supports software interrupts, which are triggered by specific instructions in the program. The software interrupts are implemented using the instruction "INT" or "INTO".

1. .Timer 0 Interrupt (TF0): This interrupt is triggered when Timer 0 overflows.
2. Timer 1 Interrupt (TF1): This interrupt is triggered when Timer 1 overflows.

3. Serial Port Interrupt (RI/TI): These interrupts are triggered by specific events in the serial communication (receive interrupt - RI, transmit interrupt - TI).

To use interrupts in the 8051 microcontroller, you need to follow these steps:

1. **Enable Global Interrupts:** Set the "EA" (Enable All) bit in the Special Function Register (SFR) IE to enable interrupts globally. This allows the microcontroller to respond to interrupt requests.
2. **Enable Specific Interrupts:** Enable the specific interrupt sources you want to use by setting the corresponding bits in the IE SFR. For example, to enable the Timer 0 interrupt, set the "ET0" bit in IE.
3. **Write Interrupt Service Routines (ISRs):** Write the necessary ISRs for each enabled interrupt source. These routines handle the specific interrupt event and perform the required actions or tasks. Each ISR should end with a return instruction (RETI).
4. **Triggering and Execution:** When an interrupt event occurs, the microcontroller suspends the current program execution and transfers control to the corresponding ISR. After executing the ISR, the microcontroller returns to the point where it left off in the main program.

| Interrupt Number | Interrupt Description | Address |
|---|---|---|
| 0 | EXTERNAL INT 0 | 0003h |
| 1 | TIMER/COUNTER 0 | 000Bh |
| 2 | EXTERNAL INT 1 | 0013h |
| 3 | TIMER/COUNTER 1 | 001Bh |
| 4 | SERIAL PORT | 0023h |

Interrupts in the 8051 microcontroller allow for timely handling of external events or internal conditions, ensuring the system can respond quickly to critical tasks. By

effectively utilizing interrupts, you can achieve efficient multitasking, real-time event handling, and improved system performance in your 8051-based applications.

## IE (Interrupt Enable) Register

This register is responsible for enabling and disabling the interrupt. EA register is set to one for enabling interrupts and set to 0 for disabling the interrupts. Its bit sequence and their meanings are shown in the following figure.

| EA | - | - | ES | ET1 | EX1 | ET0 | EX0 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

| | | |
|---|---|---|
| EA | IE.7 | It disables all interrupts. When EA = 0 no interrupt will be acknowledged and EA = 1 enables the interrupt individually. |
| - | IE.6 | Reserved for future use. |
| - | IE.5 | Reserved for future use. |
| ES | IE.4 | Enables/disables serial port interrupt. |
| ET1 | IE.3 | Enables/disables timer1 overflow interrupt. |
| EX1 | IE.2 | Enables/disables external interrupt1. |
| ET0 | IE.1 | Enables/disables timer0 overflow interrupt. |
| EX0 | IE.0 | Enables/disables external interrupt0. |

### IP (Interrupt Priority) Register

We can change the priority levels of the interrupts by changing the corresponding bit in the Interrupt Priority (IP) register as shown in the following figure.

A low priority interrupt can only be interrupted by the high priority interrupt, but not interrupted by another low priority interrupt.

If two interrupts of different priority levels are received simultaneously, the request of higher priority level is served.

If the requests of the same priority levels are received simultaneously, then the internal polling sequence determines which request is to be serviced.

| - | - | PT2 | PS | PT1 | PX1 | PT0 | PX0 |
|---|---|---|---|---|---|---|---|
| bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | |

| | | |
|---|---|---|
| - | IP.6 | Reserved for future use. |
| - | IP.5 | Reserved for future use. |
| PS | IP.4 | It defines the serial port interrupt priority level. |
| PT1 | IP.3 | It defines the timer interrupt of 1 priority. |
| PX1 | IP.2 | It defines the external interrupt priority level. |
| PT0 | IP.1 | It defines the timer0 interrupt priority level. |
| PX0 | IP.0 | It defines the external interrupt of 0 priority level. |

# Interrupt programming in 8051

Timer Interrupt Programming: In microcontroller Timer 1 and Timer 0 interrupts are generated by time register bits TF0 AND TF1. This timer interrupts programming by C code involves:

1. Selecting the configuration of TMOD register and their mode of operation.
2. Enables the IE registers and corresponding timer bits in it.

3. Choose and load the initial values of TLx and THx by using appropriate mode of operation.
4. Set the timer run bit for starting the timer.
5. Write the subroutine for a timer and clears the value of TRx at the end of the subroutine.

**Let's see the timer interrupt programming using Timer0 model for blinking LED using interrupt method:**

```
#include< reg51 .h>
sbit Blink Led = P2^0;  // LED is connected to port 2 Zeroth pin
void timer0_ISR (void) interrupt 1  //interrupt no. 1 for Timer0
 {
Blink Led=~Blink Led;   // Blink LED on interrupt
TH0=0xFC;   // loading initial values to timer
TL0=0x66;
}
void main()
{
TMOD=0x0l;      // mode 1 of Timer0
TH0 = 0xFC:    // initial value is loaded to timer
TL0 = 0x66:
ET0 =1;         // enable timer 0 interrupt
TR0 = 1;        // start timer
while (1);      // do nothing
 }
```

**External Hardware Interrupt Programming**

Microcontroller 8051 is consisting of two external hardware interrupts: INT0 and INT1 as discussed above. These interrupts are enabled at pin 3.2 and pin 3.3. It can be level triggered or edge triggered. In level triggering, low signal at pin 3.2 enables the interrupt, while at pin 3.2 high to low transition enables the edge triggered interrupt.

1. Let us see the programmable feature of 8051 microcontroller is:
2. Enables the equivalent bit of external interrupt in Interrupt Enable (IE) register.
3. If it is level triggering, then write subroutine appropriate to this interrupt, or else enable the bit in TCON register corresponding to the edge triggered interrupt.

**Consider the edge triggered external hardware interrupt programming is:-**

```
void main()

{

IT0 = 1;    // Configure interrupt 0 for falling edge on INT0

EXO = 1;    // Enabling the EX0 interrupt

EA =1;      // Enabling the global interrupt flag

}

void ISR_ex0(void) interrupt 0

{

<body of interrupt>

}
```

**Serial Communication Interrupt Programming**

It is used when there is a need to send or receive data. Since one interrupt bit is used for both Transfer Interrupt (TI) and Receiver Interrupt (RI) flags, Interrupt Service Routine (ISR) must examine these flags for knowing the actual interrupt. By the logical OR operation of RI and TI flags causes the interrupt and it is clear by the software alone. Consider the steps involved in serial communication interrupt programming are:-

1. Configure the Interrupt Enable register for enabling serial interrupt.
2. Configure the SCON register for performing transferring and receiving operation.
3. Write a subroutine for given interrupt with appropriate function.

Let's see the program for sending 'E' through serial port with 9600 baud rate using Serial Interrupt:

```
void main()
{
TMOD = 0x20:
TH1= 0xFD;      // baud rate for 9600 bps
SCON = 0x50;
TR1=1;
EA=l;
whlle(l);
}
void ISR_Serial(void) interrupt 4
```

```
{
if(TI==l)
{
SBUF= ?E?;
TI=0;
}
else
RI =0;
}
```