



Chapter 6

The 8051 Microcontroller

Interrupt

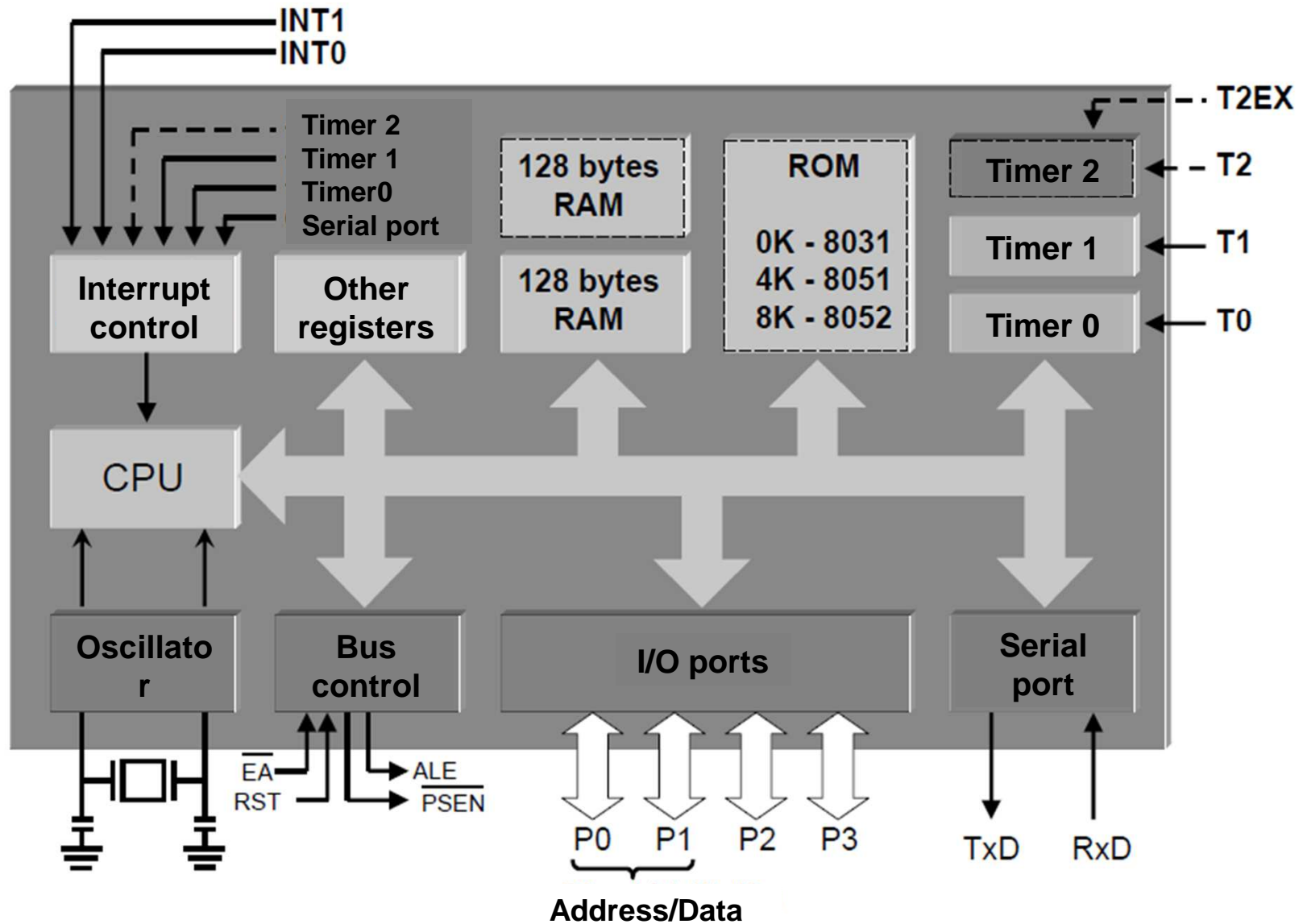


Faculty of Computer Science and Engineering
Department of Computer Engineering

Nguyen Quang Huy

huynguyen@cse.hcmut.edu.vn

Introduction



CPU

I/O

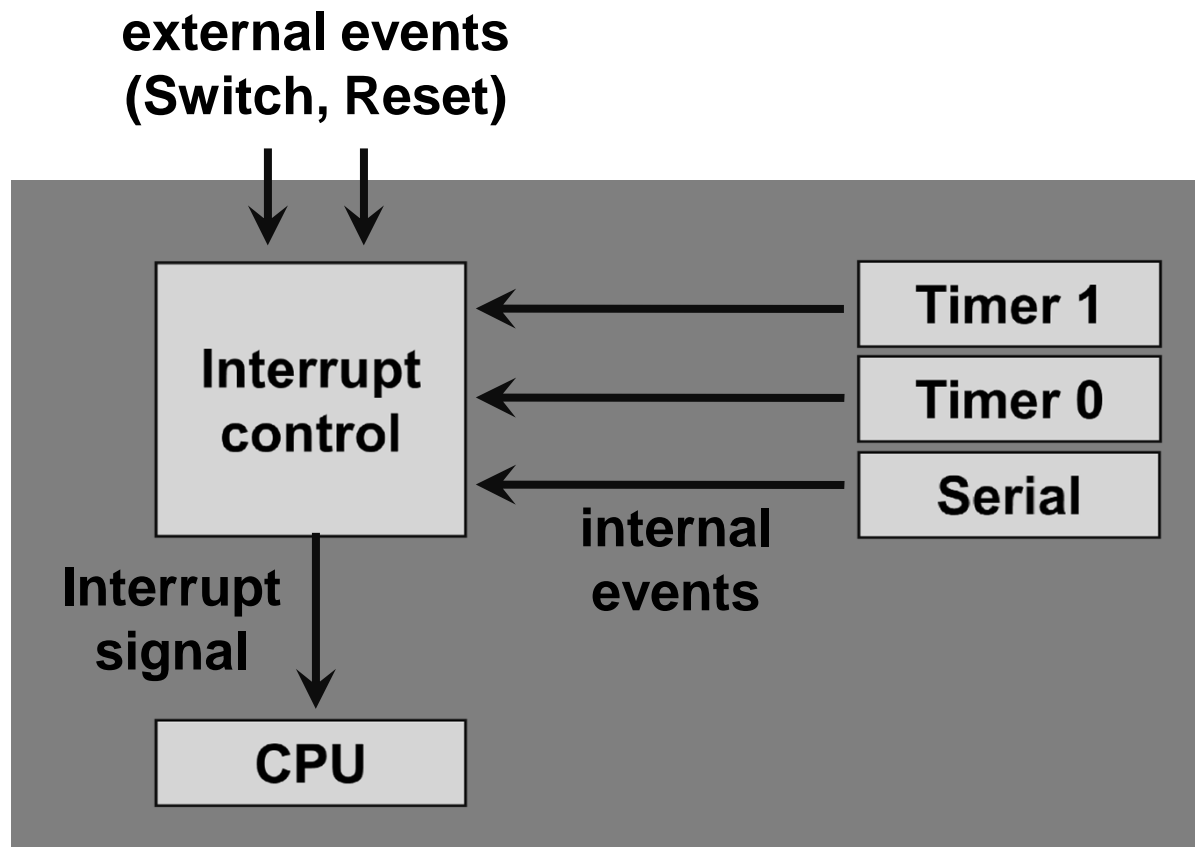
Timer

Interrupt

Serial

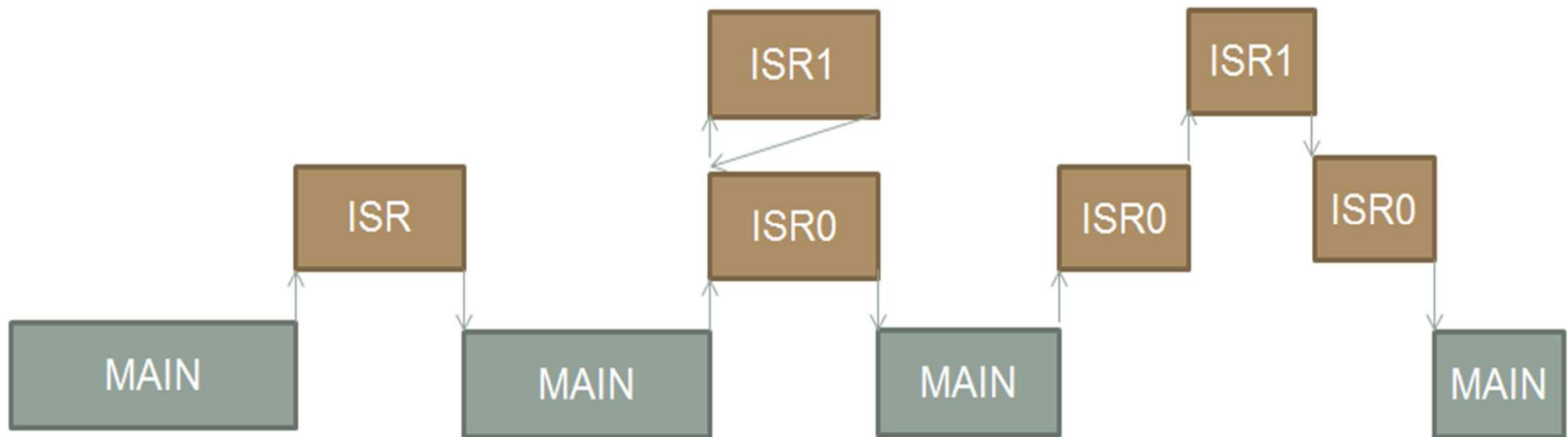
Introduction

- Interrupt : an external or internal event that interrupts the microcontroller to inform that a device needs its service.



Introduction

- An interrupt causes a change in the normal flow of instruction execution
- The microcontroller can handle many interrupts at a time or ignore some interrupts (based on priority)



Why Interrupt?

- A single microcontroller can serve several devices
 - These devices occasionally need CPU service
 - But we cannot predict when
 - Keep the CPU busy between events
- Need a way for CPU to find out devices need attention
- Polling
 - Interrupt

Polling vs. Interrupt

- Polling: CPU periodically checks each device to see if it needs service
 - Takes CPU time even when no requests pending
 - Impossible to assign device's priority (round-robin)
 - Impossible to ignore a device request for service
 - Overhead may be reduced at expense of response time
 - Can be efficient if events arrive rapidly

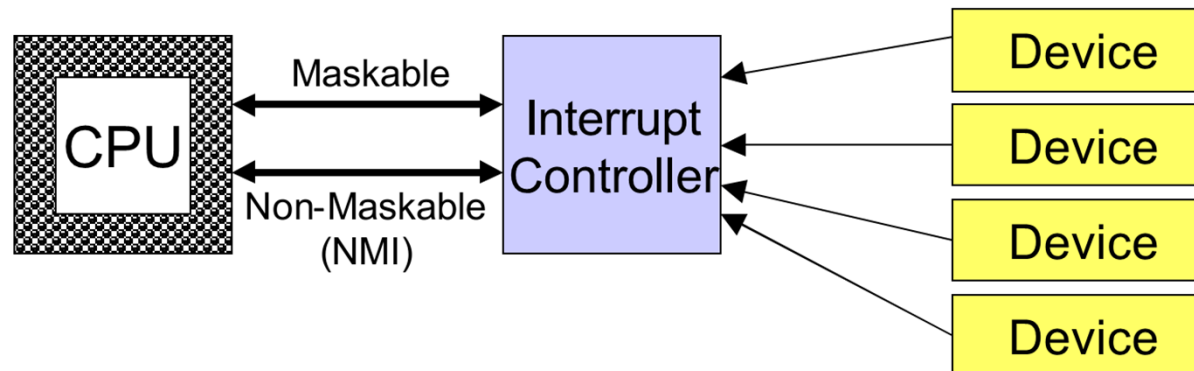
“Polling is like picking up your phone every few seconds to see if you have a call”

Polling vs. Interrupt

- Interrupt

- Upon interrupt signals, CPU interrupts its operation and serves the device
- CPU serves devices based on assigned priority
- CPU can ignore (mask) a device request for service
- No overhead when no requests pending
- Can do other works between events

“Interrupts are like waiting for the phone to ring”



Interrupt Service Routine (ISR)

- For every interrupt, there must be an *interrupt service routine* (ISR) or *interrupt handler*
- When an interrupt is invoked, the MCU runs the ISR
- For every interrupt, there is a fixed location in memory that holds the address of ISR
 - Each ISR must start at a specific memory address.
 - The group of memory locations set aside to hold the addresses of ISRs is called *interrupt vector table*

Interrupt Sources

| Interrupt | ROM Location (hex) | Pin |
|------------------------|--------------------|-----------|
| Reset | 0000 | 9 |
| External HW (INT0) | 0003 | P3.2 (12) |
| Timer 0 (TF0) | 000B | |
| External HW (INT1) | 0013 | P3.3 (13) |
| Timer 1 (TF1) | 001B | |
| Serial COM (RI and TI) | 0023 | |

ROM Memory

| | | |
|-------|--|---------|
| 0000H | LJMP MAIN | ← Reset |
| 0003H | Ex. INT0 ISR | |
| 000BH | Timer 0 ISR | |
| 0013H | Ex. INT1 ISR | |
| 001BH | Timer 1 ISR | |
| 0023H | Serial ISR | |
| 0030H | MAIN: <i>Main program should be here</i> | |

Interrupt Sources

| Interrupt | Name | Numbers |
|----------------------|-------------|----------------|
| External Interrupt 0 | (INT0) | 0 |
| Timer Interrupt 0 | (TF0) | 1 |
| External Interrupt 1 | (INT1) | 2 |
| Timer Interrupt 1 | (TF1) | 3 |
| Serial Communication | (RI + TI) | 4 |
| Timer 2 (8052 only) | (TF2) | 5 |

Interrupt Service Routine (ISR)

```
void ext0_isr(void) interrupt 0 { ... }
```

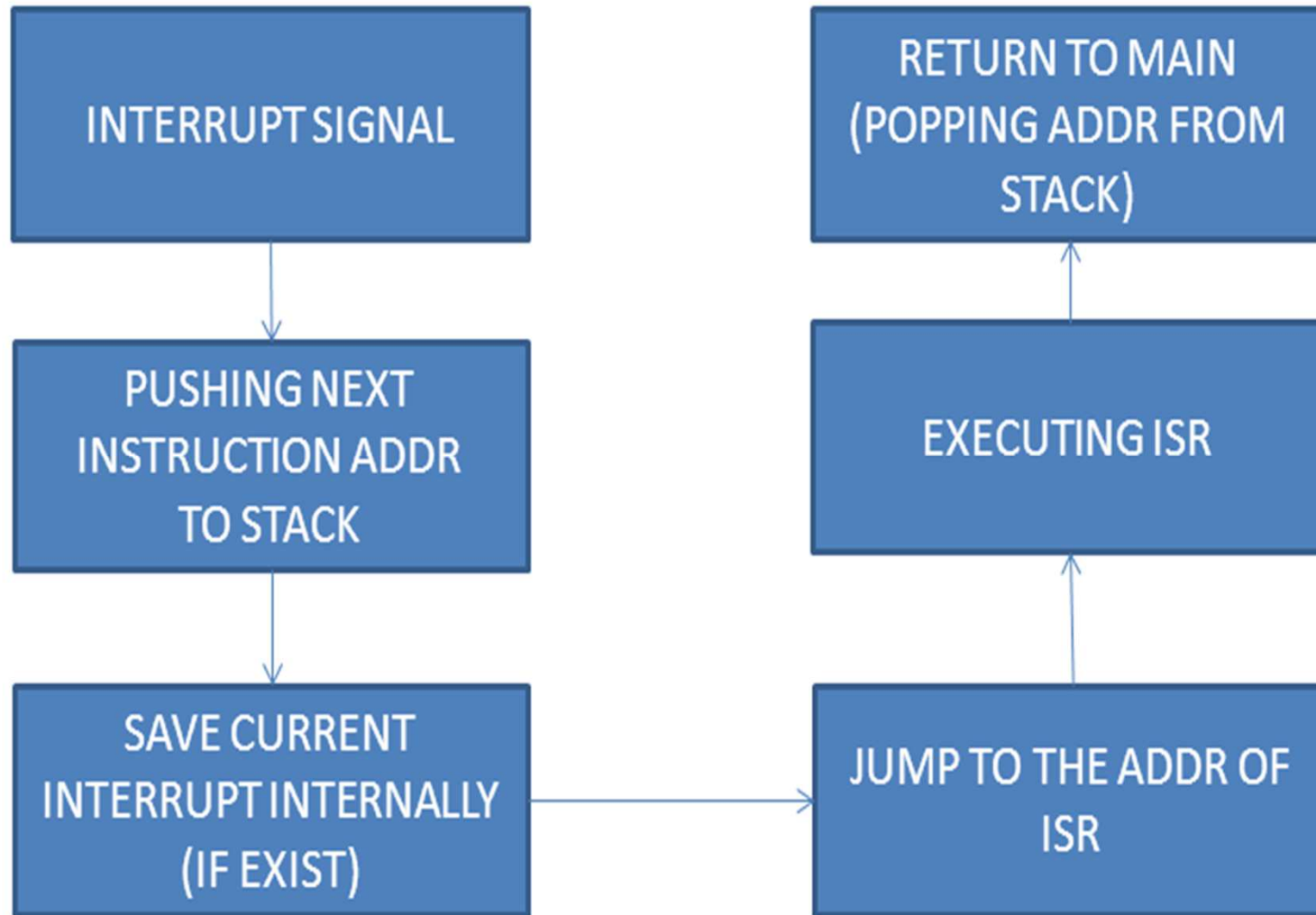
```
void timer0_isr(void) interrupt 1 { ... }
```

```
void ext1_isr(void) interrupt 2 { ... }
```

```
void timer1_isr(void) interrupt 3 { ... }
```

```
void serial_isr(void) interrupt 4 { ... }
```

Steps in Executing an Interrupt



Enable/Disable Interrupts



EA (enable all) must be set to 1 in order for rest of the register to take effect

| | | |
|-----|------|--|
| EA | IE.7 | Disables all interrupts |
| -- | IE.6 | Not implemented, reserved for future use |
| ET2 | IE.5 | Enables or disables timer 2 overflow or capture interrupt (8952) |
| ES | IE.4 | Enables or disables the serial port interrupt |
| ET1 | IE.3 | Enables or disables timer 1 overflow interrupt |
| EX1 | IE.2 | Enables or disables external interrupt 1 |
| ET0 | IE.1 | Enables or disables timer 0 overflow interrupt |
| EX0 | IE.0 | Enables or disables external interrupt 0 |

External Interrupt

- The 8051 has two external hardware interrupts
 - Pin 12 (P3.2) and pin 13 (P3.3) are designated as INT0 and INT1
 - Bits ET0 / ET1 must be set to enable external interrupt.
 - Priority: 0 (Ext0) & 2 (Ext1)
 - There are two activation levels for the external hardware interrupts
 - Level triggered
 - Edge triggered

External Interrupt

- Timer Control Register (TCON)

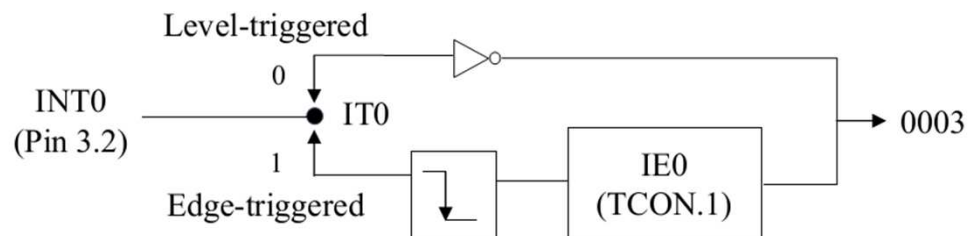


External interrupt edge flag

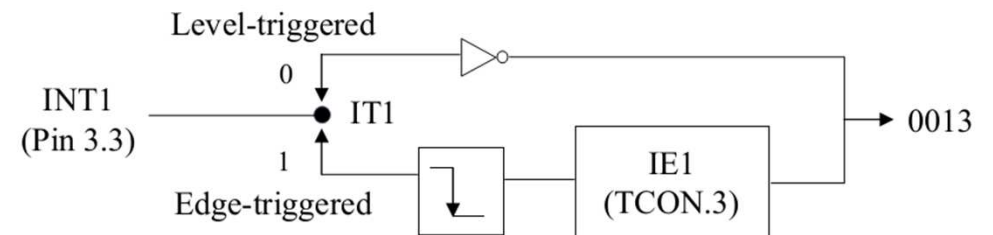
- **Set** by CPU when the external interrupt edge (**H-to-L**) is detected
- **Clear** by CPU when finishes the ISR

IT = 0 for **level**-triggered (**default**)
IT = 1 for **edge**-triggered

Activation of INT0

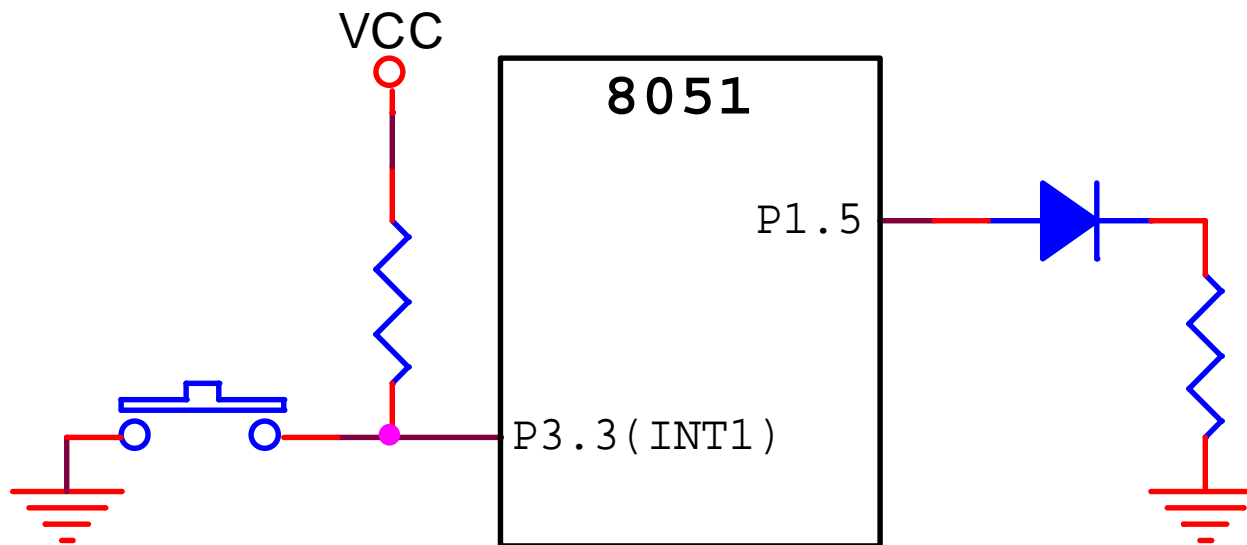


Activation of INT1



External Interrupt - Example

- Write a program to turn ON the led when switch is pressed LOW. Compare among three methods
 - Polling
 - Level-triggered
 - Edge-triggered



Polling Method

```
#include <REG51.h>

sbit SW = P3^3;
sbit LED = P1^5;

void main(){
    SW = 1;           // SW as input
    while (1){
        if (SW == 0) // SW pressed
            LED = 1;  // LED ON
        else
            LED = 0;  // LED OFF
    }
}
```

Level-Triggered Interrupt

```
#include <REG51.h>

sbit SW = P3^3;
sbit LED = P1^5;

void ext_isr(void) interrupt 2{
    LED = 1;    // LED ON
}

void main(){
    IT1 = 0;    // Level-triggered enable
    IE = 0x84;  // Enable EX.INT1
    while (1){
        LED = 0; // LED OFF
    }
}
```

Edge-Triggered Interrupt

```
#include <REG51.h>

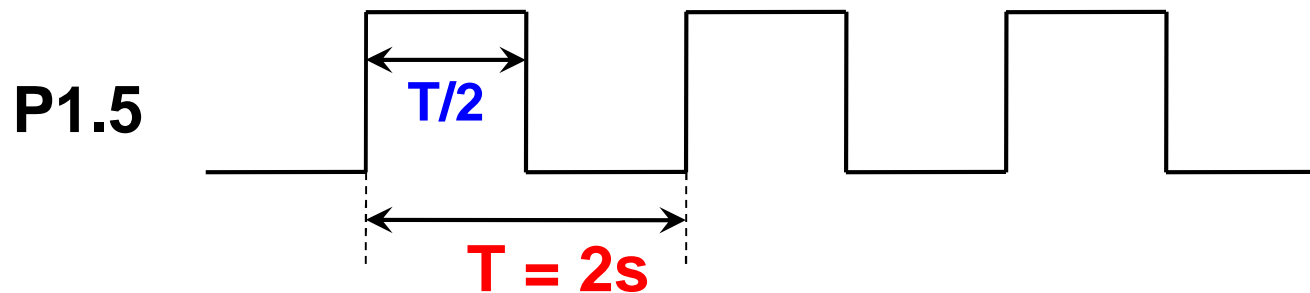
sbit SW = P3^3;
sbit LED = P1^5;

void ext_isr(void) interrupt 2{
    LED = 1;    // LED ON
}

void main(){
    IT1 = 1;    // Edge-triggered enable
    IE = 0x84;  // Enable EX.INT1
    while (1){
        LED = 0; // LED OFF
    }
}
```

Timer Interrupt

- Write a program to generate a **square wave of 0.5Hz frequency** on pin 1.5. Assume that XTAL = 12 MHz



- Solution
 - $T = 1/f = 1/0.5 = 2s \rightarrow T/2 = 1s = 50ms \times 20$
 - We need a subroutine which generates a 50ms time delay
 \rightarrow **Delay value** = $50ms/1\mu s = 50\,000$ (clocks)

Polling Program

```
#include <REG51.h>
sbit out = P1^5;
void delay_50ms(void){
    TH0 = (-50000/255);
    TL0 = (-50000%255);
    TR0 = 1;    // Start timer 0
    while (TF0==0);    //Polling
    TR0 = 0;
    TF0 = 0;
}
void delay_1s(void){
    unsigned char cnt;
    for (cnt=0; cnt<20; cnt++)
        delay_50ms();
}
```

```
void main(){
    TMOD = 0x01;
    out = 1;
    while (1){
        delay_1s();
        out = ~out;
    }
}
```

Interrupt Program

```
#include <REG51.h>
sbit out = P1^5;
unsigned char cnt = 20;
void timer0_isr(void) interrupt 1{
    TR0 = 0;
    TH0 = (-50000/255);
    TL0 = (-50000%255);
    if (cnt == 0) {
        cnt = 20; out = ~out;
    }
    else
        cnt--;
    TF0 = 0;
    TR0 = 1;
}
```

```
void main(void) {
    TMOD = 0x01;
    IE = 0x82;
    TH0 = (-50000/255);
    TL0 = (-50000%255);
    TR0 = 1; /* Start
               timer */
    out = 1;
    while (1){
        P2 = P3;
        P3 = 0xFF;
        ... // Everything
    }
}
```

Reference

- “*The 8051 Microcontroller and Embedded Systems Using Assembly and C – 2nd*” - Muhammad Ali Mazidi, Janice Gillispie Mazidi, Rolin D.McKinlay
- “*The 8051 Microcontroller - 2nd*” - I. Scott Mackenzie, Prentice-Hall 1995