Interrupts

Chapter 11

Lecture 6

Bilal Habib

DCSE, UET

INTERRUPTS

Interrupts vs. Polling

- An *interrupt* is an external or internal event that interrupts the microcontroller to inform it that a device needs its service
- A single microcontroller can serve several devices by two ways
  - Interrupts
    - Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal
    - Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device
    - The program which is associated with the interrupt is called the *interrupt service routine* (ISR) or *interrupt handler*

INTERRUPTS

Interrupts vs. Polling (cont')

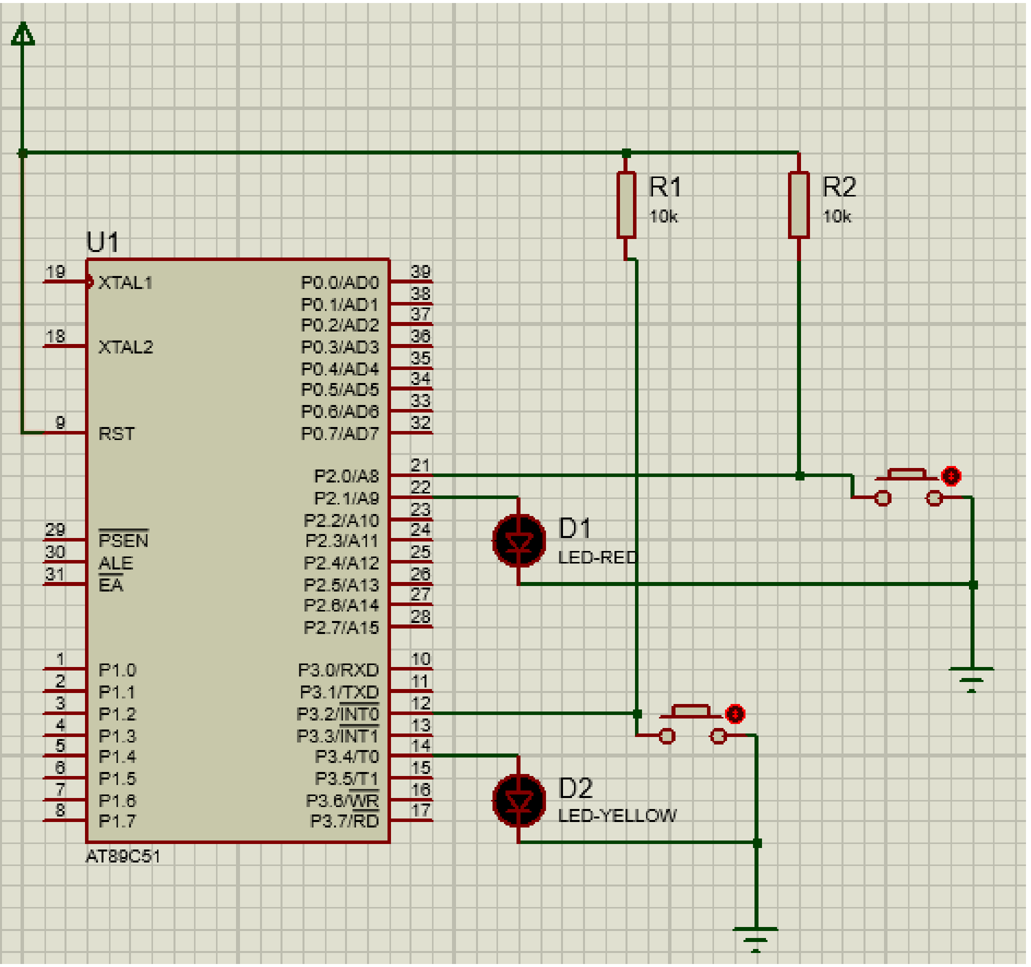
- (cont')
  - Polling
    - The microcontroller continuously monitors the status of a given device
    - When the conditions met, it performs the service
    - After that, it moves on to monitor the next device until every one is serviced
- Polling can monitor the status of several devices and serve each of them as certain conditions are met
  - The polling method is not efficient, since it wastes much of the microcontroller's time by polling devices that do not need service
  - ex. `JNB TF,target`

INTERRUPTS

Interrupts vs. Polling
(cont')

- The advantage of interrupts is that the microcontroller can serve many devices (not all at the same time)
  - Each devices can get the attention of the microcontroller based on the assigned priority
  - For the polling method, it is not possible to assign priority since it checks all devices in a round-robin fashion
- The microcontroller can also ignore (mask) a device request for service
  - This is not possible for the polling method

Polling VS Interrupts




```c
#include <reg51.h>
#include <stdio.h>
sbit polling_button = P2^0;
sbit LED_red = P2^1;
sbit interrupt_button = P3^2;
sbit LED_yellow = P3^4;
int i= 0;

void ext_int_0() interrupt 0 // ISR for External Interrupt 0 (INT0)
{
    for( i = 0; i< 1000; i++);   // To take care of debouncing
    LED_yellow ^= 1; // LED_yellow = LED_yellow ^ 1;
}

void main()
{
   interrupt_button = 1; // Configure the INT0 pin as Inputs
   polling_button = 1; // Configure P2.0 as input
   EX0 = 1;     // Enable INT0, IE register
   EA  = 1;     // Enable Global Interrupt bit, IE register
   while(1)
   {
    if(polling_button == 0)
        LED_red = 1;
    else
        LED_red = 0;
    }
}
```

INTERRUPTS

Interrupt Service Routine

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

INTERRUPTS

Steps in Executing an Interrupt

- Upon activation of an interrupt, the microcontroller goes through the following steps
  1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack
  2. It also saves the current status of all the interrupts internally (i.e: not on the stack)
  3. It jumps to a fixed location in memory, called the interrupt vector table, that holds the address of the ISR

INTERRUPTS

Steps in Executing an Interrupt (cont')

(cont')

4. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it
   - It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt)
5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted
   - First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC
   - Then it starts to execute from that address

INTERRUPTS

Six Interrupts in 8051

- Six interrupts are allocated as follows
  - Reset – power-up reset
  - Two interrupts are set aside for the timers: one for timer 0 and one for timer 1
  - Two interrupts are set aside for hardware external interrupts
    - P3.2 and P3.3 are for the external hardware interrupts INT0 (or EX1), and INT1 (or EX2)
  - Serial communication has a single interrupt that belongs to both receive and transfer

INTERRUPTS

Six Interrupts in 8051 (cont')

Interrupt vector table

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	

```
        ORG   0       ;wake-up ROM reset location
        LJMP  MAIN    ;by-pass int. vector table
;---- the wake-up program
        ORG   30H
MAIN:
        ....
        END
```

Only three bytes of ROM space assigned to the reset pin. We put the LJMP as the first instruction and redirect the processor away from the interrupt vector table.

INTERRUPTS

Enabling and Disabling an Interrupt

- Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated
- The interrupts must be enabled by software in order for the microcontroller to respond to them
  - There is a register called IE (interrupt enable) that is responsible for enabling (unmasking) and disabling (masking) the interrupts

INTERRUPTS

Enabling and Disabling an Interrupt (cont')

IE (Interrupt Enable) Register

D7 … D0

EA	--	ET2	ES	ET1	EX1	ET0	EX0

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

INTERRUPTS

Enabling and Disabling an Interrupt (cont')

- To enable an interrupt, we take the following steps:
  1. Bit D7 of the IE register (EA) must be set to high to allow the rest of register to take effect
  2. The value of EA
     - If EA = 1, interrupts are enabled and will be responded to if their corresponding bits in IE are high
     - If EA = 0, no interrupt will be responded to, even if the associated bit in the IE register is high

Enabling and Disabling interrupts

Example 11-1

Show the instructions to (a) enable the **serial** interrupt, **timer 0** interrupt, and **external** hardware interrupt 1 (**EX1**),and (b) disable (mask) the timer 0 interrupt, then (c) show how to disable all the interrupts with a single instruction.

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

Solution:

(a) MOV IE,#10010110B ;enable serial,

;timer 0, EX1

IE = 0x 96;

Another way to perform the same manipulation is

SETB IE.7 ;EA=1, global enable

SETB IE.4 ;enable serial interrupt

SETB IE.1 ;enable Timer 0 interrupt

SETB IE.2 ;enable EX1

(b) CLR IE.1 ;mask (disable) timer 0

;interrupt only, IE = 0xFD= 1111 1101__;

(c) CLR IE.7 ;disable all interrupts

IE = 0x7F;//It will disable all interrupts

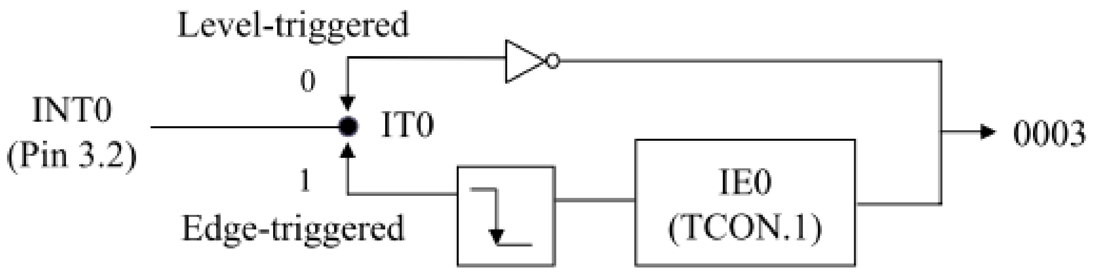
EXTERNAL HARDWARE INTERRUPTS

- The 8051 has two external hardware interrupts
  - Pin 12 (P3.2) and pin 13 (P3.3) of the 8051, designated as INT0 and INT1, are used as external hardware interrupts
    - The interrupt vector table locations 0003H and 0013H are set aside for INT0 and INT1
  - There are two activation levels for the external hardware interrupts
    - Level trigged
    - Edge trigged

triggered

EXTERNAL HARDWARE INTERRUPTS (cont')

Activation of INT0



Activation of INT1