Principle and Interface Techniques of Microcontroller

--8051 Microcontroller and Embedded Systems Using Assembly and C

LI, Guang (李光) **Prof. PhD, DIC, MIET**

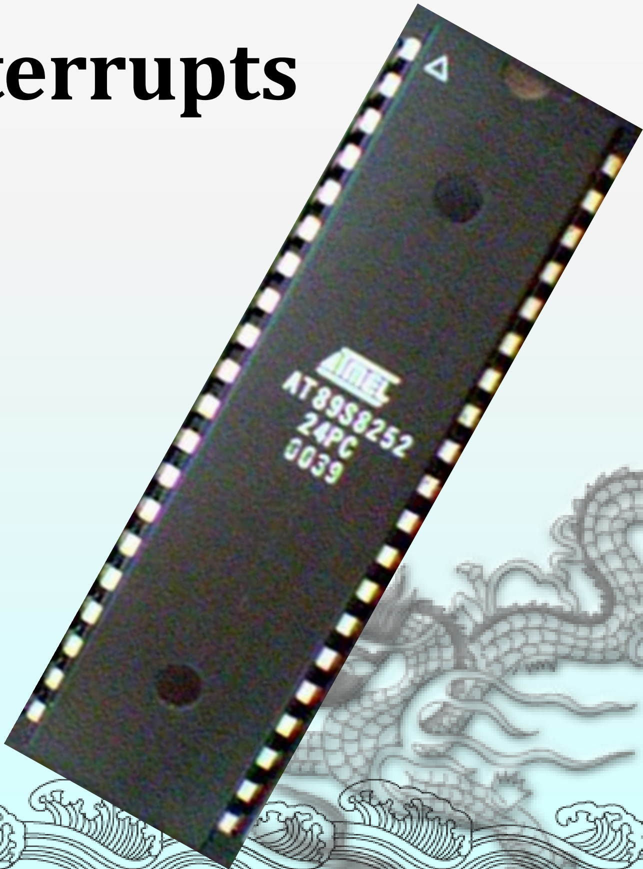
WANG, You (王酉) **PhD, MIET**

杭州 • 浙江大学 • 2014

Chapter 9
Interrupts Programming

Outline

- § 9-1 Interrupts of 8051
- § 9-2 External Hardware Interrupts
- § 9-3 Interrupt Priority
- § 9-4 Programming in C



§ 9-1 Interrupts of 8051

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 and Polling

Inte

Polling

➢Whenever any device needs its service, the device notifies the microcontroller by

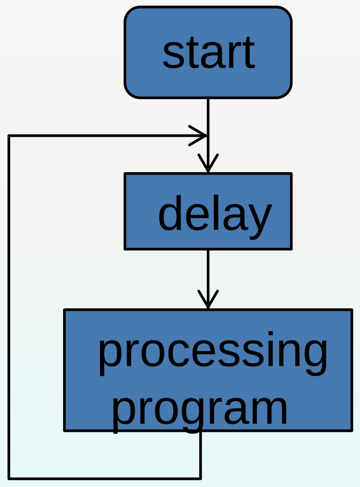
➢The microcontroller continuously monitors the status of a given device

➢When the conditions met, it performs the Service

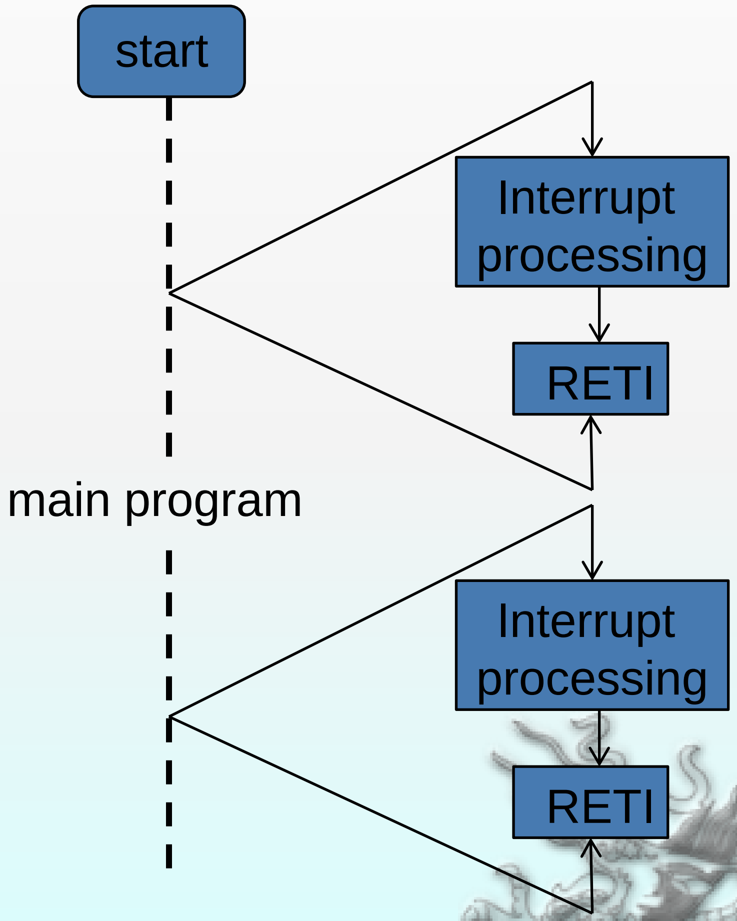
➢After that, it moves on to monitor the next device until every one is serviced

routine(ISR) or interrupt handler

Interrupts vs. Polling



Polling



Interrupts

Interrupts vs. Polling

- 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
- 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

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

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

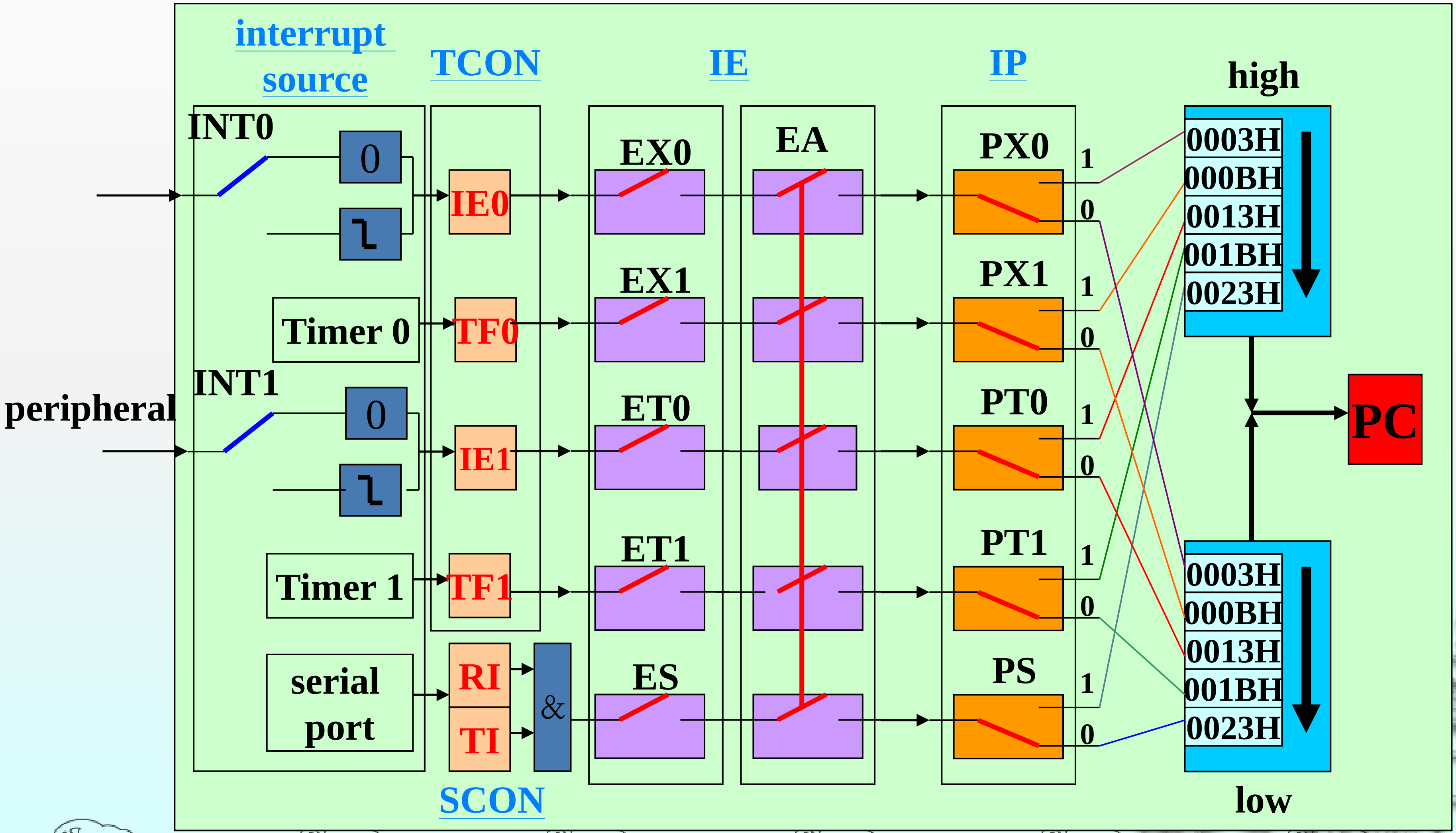
Steps in Executing an Interrupt

- 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

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

Interrupt system structure chart



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	

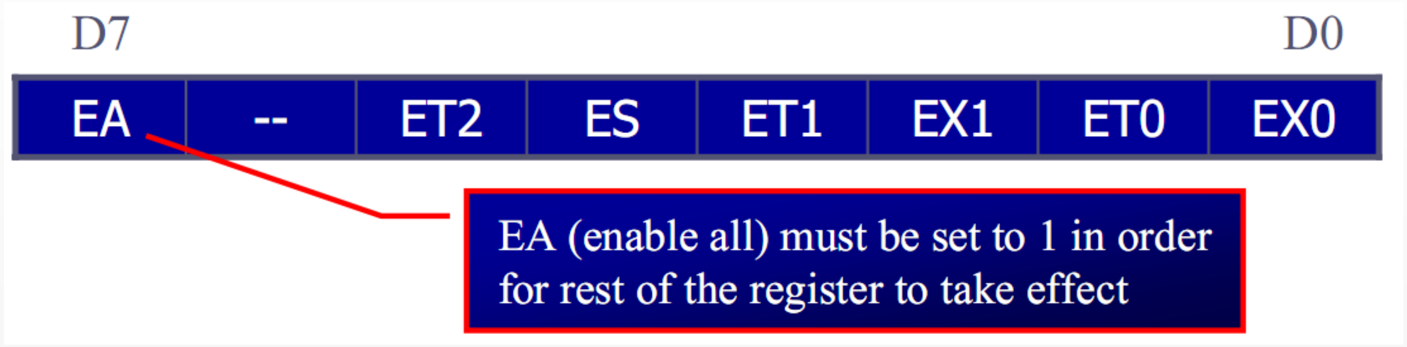
```
        ORG  0       ;wake-up ROM reset location
        LJMP MAIN    ;by-pass int. vector table
;-----  the wake-up program
        ORG  30H
MAIN:
        ....
        END
```

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.

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

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

- 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

- 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.

```
(a)     MOV  IE,#10010110B     ;enable serial, timer 0, EX1
```

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
(c)     CLR  IE.7   ;disable all interrupts
```

§ 9-2 External Hardware Interrupts

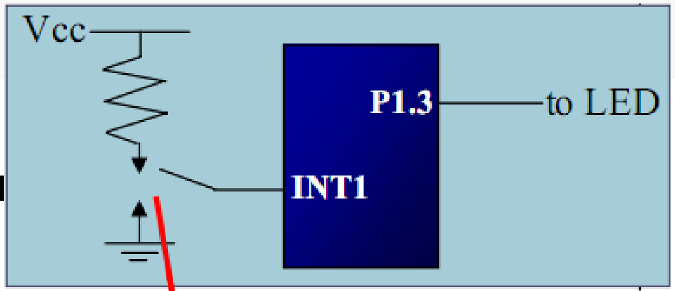
- The 8051 has two external hardware interrupts
  - Pin 12 (P3.2) and pin 13 (P3.3) of the 8051, designated as INT0 and INT1, are used as external hardware interrupts
    - ✓ The interrupt vector table locations 0003H and 0013H are set aside for INT0 and INT1
  - There are two activation levels for the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

Level-Triggered Interrupt

- In the level-triggered mode, INT0 and INT1 pins are normally high
  - If a low-level signal is applied to them, it triggers the interrupt
  - Then the microcontroller stops whatever it is doing and jumps to the interrupt vector table to service that interrupt
  - The low-level signal at the INT pin must be removed before the execution of the last instruction of the ISR, RETI; otherwise, another interrupt will be generated
- This is called a level-triggered or level-activated interrupt and is the default mode upon reset of the 8051

Example 11-2

Assume that the INT1 pin is connected to a switch that is normally high. Whenever it goes low, it should turn on an LED. The LED isconnected to P1.3 and is normally off. When it is turned on it should stay on for a fraction of a second. As long as the switch is pressed low, the LED should stay on.



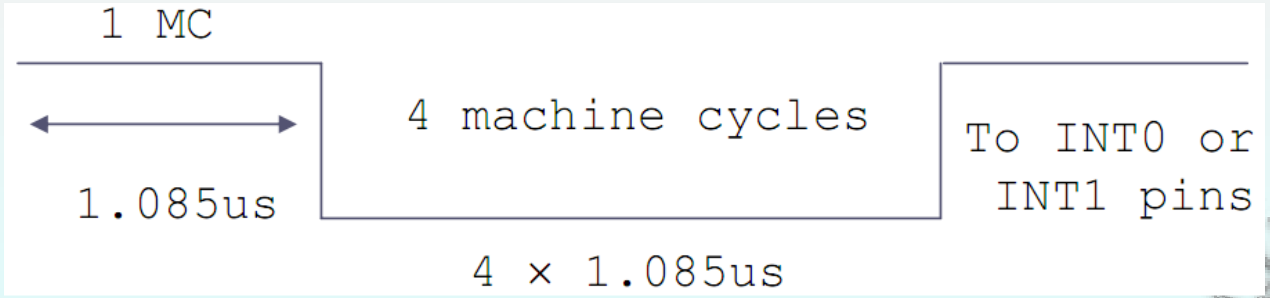
```
          ORG  0000H
          LJMP MAIN        ;by-pass interru
;----  ISR for INT1 to turn on LED
          ORG  0013H       ;INT1 ISR
          SETB P1.3        ;turn on LED
          MOV  R3,#255
BACK:     DJNZ R3,BACK     ;keep LED on for a while
          CLR  P1.3        ;turn off the LED
          RETI             ;return from ISR
;----  MAIN program for initialization
          ORG  30H
MAIN:     MOV  IE,#10000100B ;enable external INT 1
HERE:     SJMP HERE          ;stay here until get interrupted
          END
```

Pressing the switch will cause the LED to be turned on. If it is kept activated, the LED stays on

Sampling Low Level-Triggered Interrupt

- Pins P3.2 and P3.3 are used for normal I/O unless the INT0 and INT1 bits in the IE register are enabled
  - After the hardware interrupts in the IE register are enabled, the controller keeps sampling the INTn pin for a low-level signal once each machine cycle
  - According to one manufacturer's data sheet,
    - The pin must be held in a low state until the start of the execution of ISR
    - If the INTn pin is brought back to a logic high before the start of the execution of ISR there will be no interrupt
    - If INTn pin is left at a logic low after the RETI instruction of the ISR, another interrupt will be activated after one instruction is executed

- To ensure the activation of the hardware interrupt at the INTn pin, make sure that the duration of the low-level signal is around 4 machine cycles, but no more
  - ✓ This is due to the fact that the level-triggered interrupt is not latched
  - ✓ Thus the pin must be held in a low state until the start of the ISR execution



Note: On reset, IT0 (TCON.0) and IT1 (TCON.2) are both low, making external interrupt level-triggered

Edge-Triggered Interrupt

- To make INT0 and INT1 edge-triggered interrupts, we must program the bits of the TCON register
  - The TCON register holds, among other bits, the IT0 and IT1 flag bits that determine level- or edge-triggered mode of the hardware interrupt
    - IT0 and IT1 are bits D0 and D2 of the TCON register
    - They are also referred to as TCON.0 and TCON.2 since the TCON register is bit-addressable

TCON (Timer/Counter) Register (Bit-addressable)

D7							D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when timer/counter1 overflows. Cleared by hardware as the processor vectors to the interrupt service routine
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when timer/counter 0 overflows. Cleared by hardware as the processor vectors to the interrupt service routine
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 on/off

IE1	TCON.3	External interrupt 1 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt
IE0	TCON.1	External interrupt 0 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt

Assume that pin 3.3 (INT1) is connected to a pulse generator, write a program in which the falling edge of the pulse will send a high to P1.3, which is connected to an LED (or buzzer). In other words, the LED is turned on and off at the same rate as the pulses are applied to the INT1 pin.

```
        ORG  0000H
        LJMP MAIN
;---ISR for hardware interrupt INT1 to turn on LED
        ORG  0013H      ;INT1 ISR
        SETB P1.3       ;turn on LED
        MOV  R3,#255
BACK:   DJNZ R3,BACK    ;keep the buzzer on for a while
        CLR  P1.3       ;turn off the buzzer
        RETI            ;return from ISR
;------MAIN program for initialization
        ORG  30H
MAIN:   SETB TCON.2            ;make INT1 edge-triggered int.
        MOV  IE,#10000100B ;enable External INT 1
HERE:   SJMP HERE              ;stay here until get interrupted
        END
```

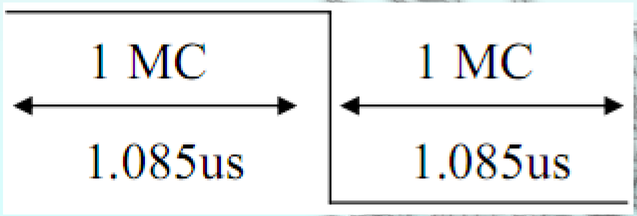
The on-state duration depends on the time delay inside the ISR for INT1

When the falling edge of the signal is applied to pin INT1, the LED will be turned on momentarily.

Sampling Edge-Triggered Interrupt

- In edge-triggered interrupts
  - The external source must be held high for at least one machine cycle, and then held low for at least one machine cycle
  - The falling edge of pins INT0 and INT1 are latched by the 8051 and are held by the TCON.1 and TCON.3 bits of TCON register
    - ✓ Function as interrupt-in-service flags
    - ✓ It indicates that the interrupt is being serviced now and on this INTn pin, and no new interrupt will be responded to until this service is finished

Minimum pulse duration to detect edge-triggered interrupts XTAL=11.0592MHz



Sampling Edge-Triggered Interrupt

- Regarding the IT0 and IT1 bits in the TCON register, the following two points must be emphasized
  - When the ISRs are finished (that is, upon execution of RETI), these bits (TCON.1 and TCON.3) are cleared, indicating that the interrupt is finished and the 8051 is ready to respond to another interrupt on that pin
  - During the time that the interrupt service routine is being executed, the INTn pin is ignored, no matter how many times it makes a high-to-low transition
    - RETI clears the corresponding bit in TCON register (TCON.1 or TCON.3)
    - There is no need for instruction CLR TCON.1 before RETI in the ISR associated with INT0

- Example 11-3

What is the difference between the RET and RETI instructions? Explain why we can not use RET instead of RETI as the last instruction of an ISR.

Solution:

Both perform the same actions of popping off the top two bytes of the stack into the program counter, and marking the 8051 return to where it left off.

However, RETI also performs an additional task of clearing the interrupt-in-service flag, indicating that the servicing of the interrupt is over and the 8051 now can accept a new interrupt on that pin. If you use RET instead of RETI as the last instruction of the interrupt service routine, you simply block any new interrupt on that pin after the first interrupt, since the pin status would indicate that the interrupt is still being serviced. In the cases of TF0, TF1, TCON.1, and TCON.3, they are cleared due to the execution of RETI.

§ 9-3 Interrupt Priority

- When the 8051 is powered up, the priorities are assigned according to the following
  - In reality, the priority scheme is nothing but an internal polling sequence in which the 8051 polls the interrupts in the sequence listed and responds accordingly

Interrupt Priority Upon Reset

Highest To Lowest Priority	
External Interrupt 0	(INT0)
Timer Interrupt 0	(TF0)
External Interrupt 1	(INT1)
Timer Interrupt 1	(TF1)
Serial Communication	(RI + TI)

- Example 11-4

Discuss what happens if interrupts INT0, TF0, and INT1 are activated at the same time. Assume priority levels were set by the power-up reset and the external hardware interrupts are edge-triggered.

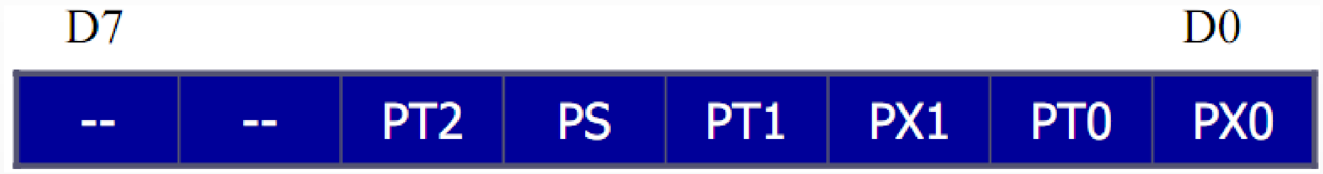
Solution:

If these three interrupts are activated at the same time, they are latched and kept internally. Then the 8051 checks all five interrupts according to the sequence listed in Table 11-3. If any is activated, it services it in sequence. Therefore, when the above three interrupts are activated, IE0 (external interrupt 0) is serviced first, then timer 0 (TF0), and finally IE1 (external interrupt 1).

§ 9-3 Interrupt Priority

- We can alter the sequence of interrupt priority by assigning a higher priority to any one of the interrupts by programming a register called IP (interrupt priority)
  - To give a higher priority to any of the interrupts, we make the corresponding bit in the IP register high
  - When two or more interrupt bits in the IP register are set to high
    - While these interrupts have a higher priority than others, they are serviced according to the sequence of Table 11-13

Interrupt Priority Register (Bit-addressable)



D7							D0
--	--	PT2	PS	PT1	PX1	PT0	PX0

--	IP.6	Reserved
--	IP.7	Reserved
PT2	IP.5	Timer 2 interrupt priority bit (8052 only)
PS	IP.4	Serial port interrupt priority bit
PT1	IP.3	Timer 1 interrupt priority bit
PX1	IP.2	External interrupt 1 priority bit
PT0	IP.1	Timer 0 interrupt priority bit
PX0	IP.0	External interrupt 0 priority bit

Priority bit=1 assigns high priority
Priority bit=0 assigns low priority

- ## Example 11-4

(a) Program the IP register to assign the highest priority to INT1(external interrupt 1), then

(b) discuss what happens if INT0, INT1, and TF0 are activated at thesame time. Assume the interrupts are both edge-triggered.

Solution:

(a) MOV IP,#00000100B ;IP.2=1 assign INT1 higher priority.

The instruction SETB IP.2 also will do the same thing as the above line since IP is bit-addressable.

(b) The instruction in Step (a) assigned a higher priority to INT1 than the others; therefore, when INT0, INT1, and TF0 interrupts are activated at the same time, the 8051 services INT1 first, then it services INT0, then TF0. This is due to the fact that INT1 has a higher priority than the other two because of the instruction in Step (a). The instruction in Step (a) makes both the INT0 and TF0 bits in the IP register 0. As a result, the sequence in Table 11-3 is followed which gives a higher priority to INT0 over TF0

Example 11-5

Assume that after reset, the interrupt priority is set the instruction MOV IP,#00001100B. Discuss the sequence in which the interrupts are serviced

Solution:

The instruction "MOV IP #00001100B" (B is for binary) and timer 1 (TF1)to a higher priority level compared with the reset of the interrupts. However, since they are polled according to Table, they will have the following priority.

Highest Priority	External Interrupt 1	(INT1)
	Timer Interrupt 1	(TF1)
	External Interrupt 0	(INT0)
	Timer Interrupt 0	(TF0)
Lowest Priority	Serial Communication	(RI+TI)

Interrupt inside an Interrupt

- In the 8051 a low-priority interrupt can be interrupted by a higher-priority interrupt but not by another low-priority interrupt
  - Although all the interrupts are latched and kept internally, no low-priority interrupt can get the immediate attention of the CPU until the 8051 has finished servicing the high-priority interrupts

Triggering Interrupt by Software

- To test an ISR by way of simulation can be done with simple instructions to set the interrupts high and thereby cause the 8051 to jump to the interrupt vector table
  - ex. If the IE bit for timer 1 is set, an instruction such as SETB TF1 will interrupt the 8051 in whatever it is doing and will force it to jump to the interrupt vector table
    - We do not need to wait for timer 1 go roll over to have an interrupt

§ 9-4 Programming in C

- The 8051 compiler have extensive support for the interrupts
  - They assign a unique number to each of the 8051 interrupts

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

  - It can assign a register bank to an ISR
    - ✓ This avoids code overhead due to the pushes and pops of the R0 – R7 registers

两个按键分别控制LED灯的开关, P3.2接口的按键按下时开灯, P3.3接口的按键按下时关灯。

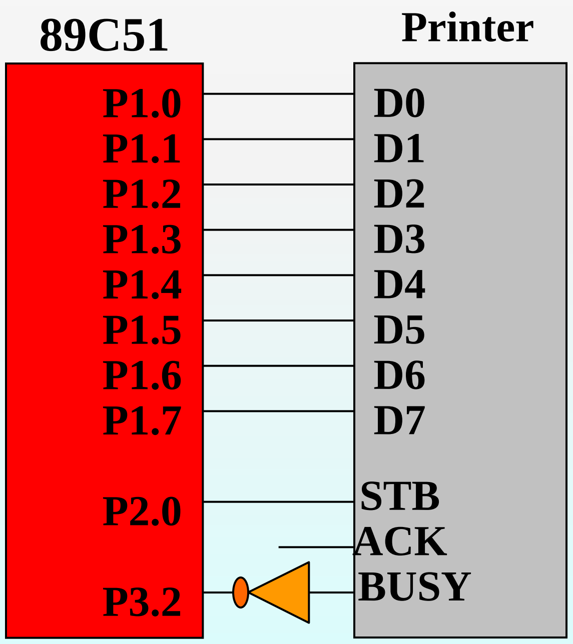
```
void  extern0( )   interrupt 0{}
void  timer0( )   interrupt 1 {}
void  extern1( )   interrupt 2{}
void  timer1( )   interrupt 3 {}
void  serial0( )   interrupt 4 {}
```

```
#include <reg51.h>
sbit   LED = P1 ^ 0;
void INT_init (void){
            EA = 1;
            EX1 = 1;
            EX0 = 1;
            IT1 = 1;     //1:   falling edge-triggered
            IT0 = 1;
}
void INT_1 (void) interrupt 2  //using 2
{

            LED = 1;   // turn off the light
}

void INT_0 (void) interrupt 0  //  using 0
{

            LED = 0;   //turn the light on
}
void main(void){
            INT_init();  //extern interrupt initialization
            while(1){
                          //other program
            }
```

Q 1 Output the value of RAM 30H ~ 60H to printer, using interrupt to finish the program.

STB: Start signal, a edge trigger will start a print.

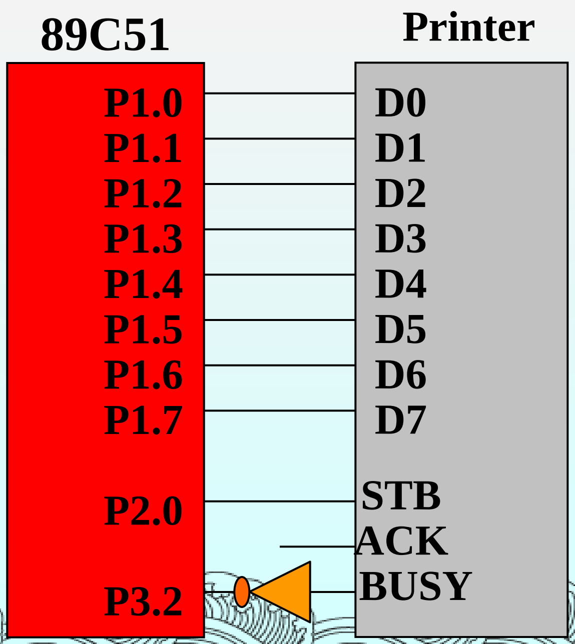
BUSY: Output of printer. It will be ‘H’ when printer is ready, it will be ‘L’ when printer is busy.



A 1

```
        ORG   0000H
        LJMP  MAIN
        ORG   0003H
        LJMP  AINT0
MAIN：  MOV   SP， #60H
        SETB  EA
        SETB  EX0
        SETB  IT0
        MOV   R0,#30H
        MOV   P1,@R0
        SETB  P2.0
        CLR   P2.0
        SJMP  $
```

```
AINT0: CJNE R0,#60H, NEQ
EQ:    SJMP AINT00
NEQ:   INC  R0
       MOV  P1,@R0
       SETB P2.0
       CLR  P2.0
AINT00: RETI
```



THANK YOU!!