Interrupt Programming

Instructor

Zhizheng Wu

吴智政

School of Mechatronic Engineering and Automation

INTERRUPT PROGRAMMING

- Introduction to 8051 Interrupt
- Interrupt Organization and Processing
- Timer Interrupt Programming
- External Interrupt Programming
- Serial Port Interrupt

Objectives:

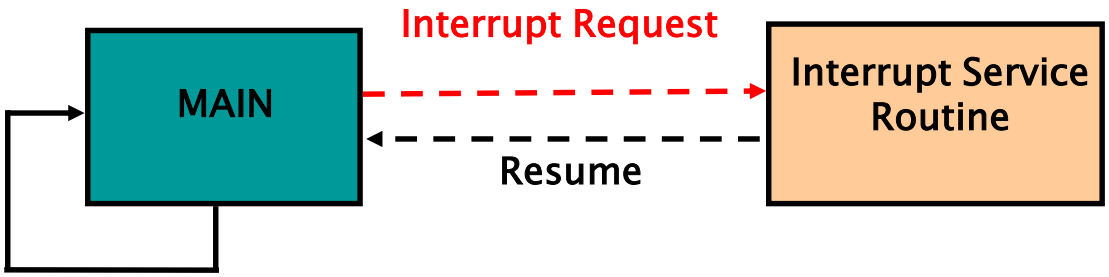
- Discuss various sources of interrupt and SFR registers used in the interrupt operations
- To explain interrupt processing mechanism

INTRODUCTION TO INTERRUPT

• Interrupt is an occurrence of a condition that causes a temporary suspension of a program while the condition is serviced by another program, which is Interrupt Service Routine (ISR).

• Interrupt allows a system to respond asynchronously to an event and deal with the event while another program is executing.

• The CPU cannot execute more than one instruction at a time, but it can temporarily suspend execution of a program, execute another, then return to the first program.



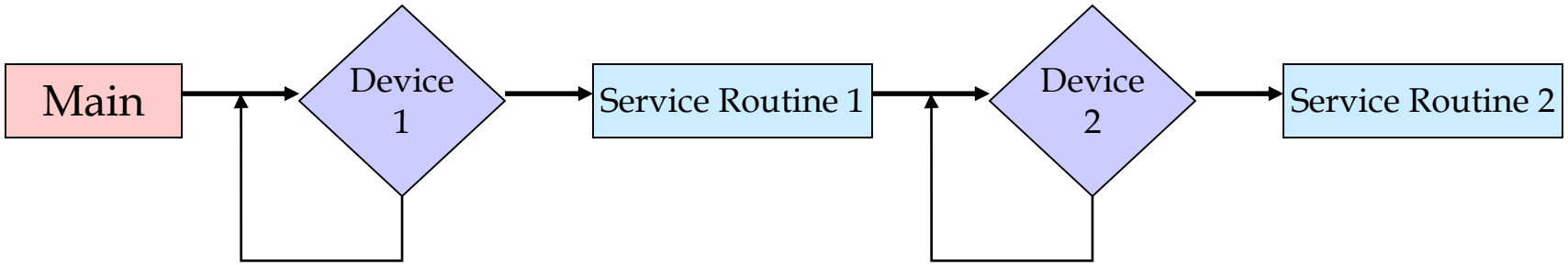
- **Interrupts vs. Polling**

- A single microcontroller can serve several devices.
- There are two ways to do that:
  1) interrupts
  2) polling.
- The program which is associated with the interrupt is called the ***interrupt service routine*** (ISR) or ***interrupt handler***.

INTERRUPT PROGRAMMING

• Interrupt is better than polling technique, where in polling technique, the CPU asks the devices periodically whether they need any service.

• Polling technique uses round robin fashion, where the devices need to wait for their turn, disregard if the device is ready or not. This method waste a lot of CPU time, which is significantly inefficient.



INTERRUPT PROGRAMMING

- Several advantages interrupt system over polling technique are:

i) In term of management

- With interrupt operation, each device can get the attention of the CPU based on the priority assigned to it and/or mask a device request for service.

- Priority based service is not possible in polling method because it works in round robin fashion.

ii) In term of efficiency

- Polling technique wastes much of the CPU processing power and time by polling devices.

- With interrupt operations, the CPU can execute the main program when no interrupt occur.

INTERRUPT ORGANIZATION

- Original 8051 has 6 sources of interrupts
  - ① Reset
  - ② Timer 0 overflow
  - ③ Timer 1 overflow
  - ④ External Interrupt 0
  - ⑤ External Interrupt 1
  - ⑥ Serial Port events (buffer full, buffer empty, etc)

☞ Each of the interrupt is individually enabled and disabled through Interrupt Enable register (IE), which is bit addressable SFR.

INTERRUPT ORGANIZATION

- Interrupt Enable Register (IE)

EA	--	ET2	ES	ET1	EX1	ET0	EX0
IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0

BIT	SYMBOL	DESCRIPTION
IE.7	EA	Global Enable/Disable
IE.6	--	Undefined
IE.5	ET2	Enable Timer 2 Interrupt (8052)
IE.4	ES	Enable Serial Port Interrupt
IE.3	ET1	Enable Timer 1 Interrupt
IE.2	EX1	Enable External 1 Interrupt
IE.1	ET0	Enable Timer 0 Interrupt
IE.0	EX0	Enable External 0 Interrupt

INTERRUPT ORGANIZATION

- Interrupt is disabled upon reset (by default).

- To enable interrupt operation, EA (IE.7) bit must be set. If EA bit is cleared, interrupt operation is disabled.

For example, the instruction to enable Timer 1 interrupt and Serial Port interrupt:

```
MOV   IE, #10011000B or,
MOV   IE, #98H
```

The instruction may also be written as:

```
SETB  IE.7    ; EA = 1, enable interrupt operation
SETB  IE.4    ; ES = 1, enable serial port interrupt
SETB  IE.3    ; ET1 = 1, enable Timer 1 interrupt
```

INTERRUPT ORGANIZATION

• Each interrupt source is individually programmed to one or two priority levels through Interrupt Priority register (IP), which is also bit-addressable SFR.

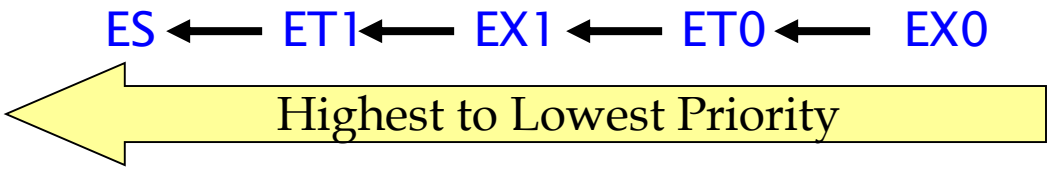
• Interrupt Priority Register (IP)

--	--	PT2	PS	PT1	PX1	PT0	PX0
IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0

BIT	SYMBOL	DESCRIPTION
IP.7	--	Undefined
IP.6	--	Undefined
IP.5	PT2	Priority for Timer 2 Interrupt (8052)
IP.4	PS	Priority for Serial Port Interrupt
IP.3	PT1	Priority for Timer 1 Interrupt
IP.2	PX1	Priority for External 1 Interrupt
IP.1	PT0	Priority for Timer 0 Interrupt
IP.0	PX0	Priority for External 0 Interrupt

INTERRUPT ORGANIZATION

- IP is cleared after a system reset to place all interrupts at the lower priority level by default



- This interrupt priority allow an ISR to be interrupted by an interrupt if the new interrupt has higher priority than the interrupt currently being serviced.

- If two interrupts with different interrupt priority occur simultaneously, the highest priority interrupt will be serviced first.

For example, instruction to prioritize Timer 1 interrupt and Serial Port interrupt is

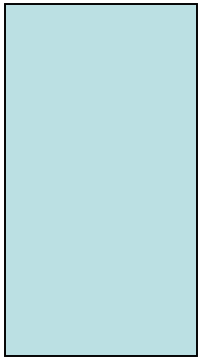
```
MOV     IP, #00011000B or,
MOV     IP, #18H
```

But if these two interrupt occur simultaneously, Timer 1 interrupt will be serviced first. (fixed polling sequence)

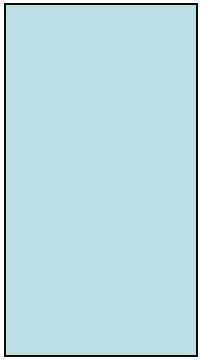
INTERRUPT ORGANIZATION

---	---	PT2	PS	PT1	PX1	PT0	PX0

- MOV IP , #00000100B or SETB IP.2 gives priority order
  1.
  2.
  3.
  4.
  5.



- MOV IP , #00001100B gives priority order
  1.
  2.
  3.
  4.
  5.



INTERRUPT PROCESSING

• When an interrupt occurs and is accepted by the CPU, the main program is interrupted. The following actions occur:

i) The current instruction completes execution
ii) The PC is saved on the stack
iii) The current interrupt status is saved internally
iv) Interrupts are blocked at the level of interrupt
v) The PC loaded with the vector address of the ISR
vi) The ISR executes

• The ISR finishes with a RETI instruction (return from interrupt). The following actions will be executed in order to resume the previous work

i) Retrieve the old PC from the stack
ii) Restore old interrupt status
iii) Continue execution of the main program from the point where it left off.

- Interrupt Vectors

➢ Interrupt vector is the address of the start of the ISR for the interrupting source.

➢ This vector address is loaded into PC when an interrupt is accepted.

INTERRUPT	FLAG	VECTOR ADDRESS
System reset	RST	0000H
External 0	IE0	0003H
Timer 0	TF0	000BH
External 1	IE1	0013H
Timer 1	TF1	001BH
Serial port	RI or TI	0023H

➢ When 'vectoring to an interrupt', the flag that caused the interrupt is automatically cleared by hardware, except for RI and TI.

INTERRUPT PROCESSING

- Since the interrupt vectors are at the bottom of code memory, the first instruction of the main program is often a jump above this area of memory, such as LJMP 0030H.

LCALL to ISR can be blocked by:

i) An interrupt of equal or higher priority level is already in progress.

ii) The current machine cycle in which the polling is done is not the final machine cycle in the execution of the instruction in progress.

iii) The instruction in progress is RETI or any write to IE or IP registers.

TIMER INTERRUPT PROGRAMMING

• Timer interrupts are caused by timer overflow, TF flag.

• When TFx = 1 (timer rolls over), and the interrupt is accepted by the CPU, the main program is interrupted and it vectors to the timer interrupt vectors. (000BH for T0 and 001BH for T1).

• Examples of timer interrupt programming:

EX.1) Write a program using Timer 0 and interrupts to create a 50% duty cycle 100Hz square wave on P1.0, and at the same time send the alternating values of ACH and 97H to Port 2 with a delay of 100µs of each transition. Assume crystal frequency to be 12MHz.

TIMER INTERRUPT PROGRAMMING

- **T = 1/f = 1/100 = 0.01s = <u>10000µs</u>**

- **Delay of each high and low portion = T x duty cycle**
  = 10000µs x 50%
  = <u>5000µs</u>

- **Machine Cycle = (1/12 x CF) $^{-1}$ = (1/12 x 12M) $^{-1}$**
  = <u>1µs</u>

- **No. of cycles = Delay/MC = 5000µs/1µs**
  = <u>5000</u> ← use mode 1

- **Load value = 65536 – 5000**
  = 60536
  = <u>EC78H</u> **; TH0 = ECH, TL0 = 78H**

```
        ORG     0000H
        LJMP    MAIN
        ORG     000BH
        LJMP    T0_ISR
        ORG     0030H
MAIN:   MOV     TMOD, #01H      ; configure timer
        MOV     TH0, #0ECH      ; load TH0
        MOV     TL0, #78H       ; load TH0
        SETB    TR0             ; start timer
        MOV     IE, #82H        ; Enable Timer 0 interrupt
HERE:   MOV     P2, #0ACH       ; send ACH to P2
        ACALL   DELAY           ; delay for 100µs
        MOV     P2, #97H        ; send 97H to P2
        ACALL   DELAY           ; delay for 100µs
        SJMP    HERE            ; repeat the process
DELAY:  MOV     R2, #49         ; delay subroutine
     XX: DJNZ   R2, XX
        RET
T0_ISR: CLR     TR0             ; stop timer
        MOV     TH0, #0ECH      ; load TH0
        MOV     TL0, #78H       ; load TL0
        SETB    TR0             ; start timer
        CPL     P1.0            ; complement bit
        RETI                    ; return from ISR
        END
```

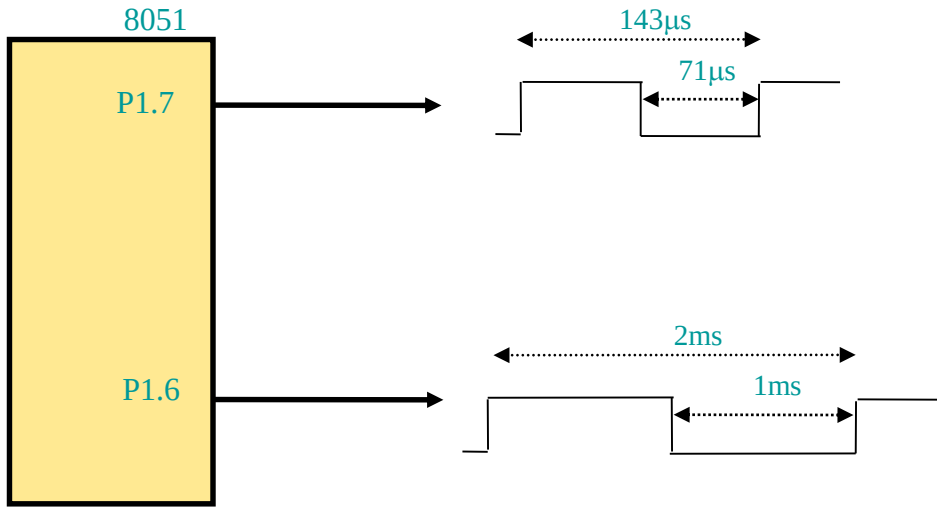
EX. 2) Write a program using Timer 1 and interrupts to create a 50% duty cycle100kHz square wave on P1.6, and at the same time send the incoming data from Port 2 to Port 3 continuously. Assume crystal frequency to be 12MHz.

- **T = 1/f = 1/100k = 10µs**
- **Delay of each high and low portion = T x duty cycle**
  = 10µs x 50%
  = 5µs
- **Machine Cycle = (1/12 x CF) $^{-1}$ = (1/12 x 12M) $^{-1}$**
  = 1µs
- **No. of cycles = Delay/MC = 5µs/1µs**
  = 5 ← use mode 2
- **Load value = 256 – 5**
  = 251
  = FBH ; TH1 = FBH

TIMER INTERRUPT PROGRAMMING

```
        ORG     0000H
        LJMP    MAIN
        ORG     001BH
        LJMP    T1_ISR

        ORG     0030H
MAIN:   MOV     IE, #10001000B ; enable Timer 1 interrupt
        MOV     P2, #0FFH      ; make P2 as input port
        MOV     TMOD, #20H     ; configure timer mode
        MOV     TH1, #0FBH     ; timer load value
        SETB    TRI            ; start timer

HERE:   MOV     P3, P2         ; send data from P2 to P3
        SJMP    HERE           ; do it continuously

T1_ISR: CPL     P1.6           ; compliment port bit
        RETI                   ; return from interrupt

        END
```

EX. 3) Write a program using interrupts to simultaneously create 7 kHz and 500 Hz square waves on P1.7 and P1.6.



TIMER INTERRUPT PROGRAMMING

```
        ORG     0000H
        LJMP    MAIN
        ORG     000BH
        LJMP    T0ISR
        ORG     001BH
        LJMP    T1ISR
        ORG     0030H
MAIN:   MOV     TMOD,#12H
        MOV     TH0,#-71
        SETB    TR0
        SETB    TR1
        MOV     IE,#8AH
        SJMP    $
T0ISR:  CPL     P1.7
        RETI
T1ISR:  CLR     TR1
        MOV     TH1,#HIGH(-1000)
        MOV     TL1,#LOW(-1000)
        SETB    TR1
        CPL     P1.6
        RETI
        END
```

- Timer ISR
  - ☞ Notice that
    - ✓ There is no need for a "CLR TFx" instruction in timer ISR
    - ✓ 8051 clears the TF internally upon jumping to ISR
  - ☞ Notice that
    - ✓ We must reload timer in mode 1
    - ✓ There is no need on mode 2 (timer auto reload)

EXTERNAL INTERRUPT PROGRAMMING

• External interrupts occur as a result of a low-level or negative edge on the INT0 at P3.2 or INT1 at P3.3 pin on the 8051.

• Flags that generate these interrupts are bits IE0 and IE1 in TCON.

• These flags will be be activated (generating interrupt signal) by two activation levels, level triggered and edge triggered.

EXTERNAL INTERRUPT PROGRAMMING

i) Level triggered

- In level triggered mode, INT1 and INT0 pins are normally high.
- If a low-level signal is applied to them (INTx = 0), it triggers the interrupt.
- If the interrupt is accepted by the CPU, the microcontroller finishes the current instruction and jump to the vector address to service the interrupt.
- The low level signal at the INTx pin must be removed before the execution of RETI. Otherwise, another interrupt will be generated.

ii) Edge triggered

- INTx pin can be edge triggered by setting the bit IT0 or IT1 in TCON
- In edge triggered mode, when a high-to-low signal is applied to the INTx pin, interrupts signal will be generated and forced to jump to the vector address.

☞ By default, INTx pins are level triggered interrupts

EXTERNAL INTERRUPT PROGRAMMING

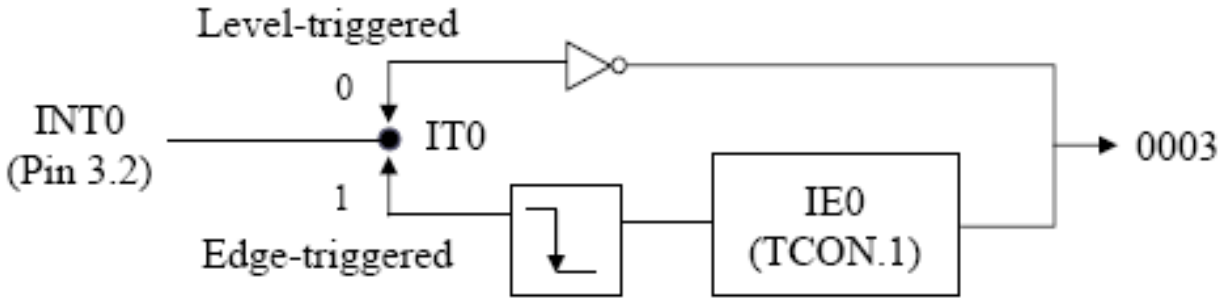
- By low nibble of Timer control register TCON
- IE0 (IE1): External interrupt 0(1) edge flag
  - set by CPU when external interrupt edge (H-to-L) is detected
  - Does not affected by H-to-L while ISR is executed
  - Cleared by CPU when RETI executed
  - does not latch low-level triggered interrupt
- IT0 (IT1): interrupt 0 (1) type control bit
  - Set/cleared by software
  - IT=1 edge trigger
  - IT=0 low-level trigger

(LSB)

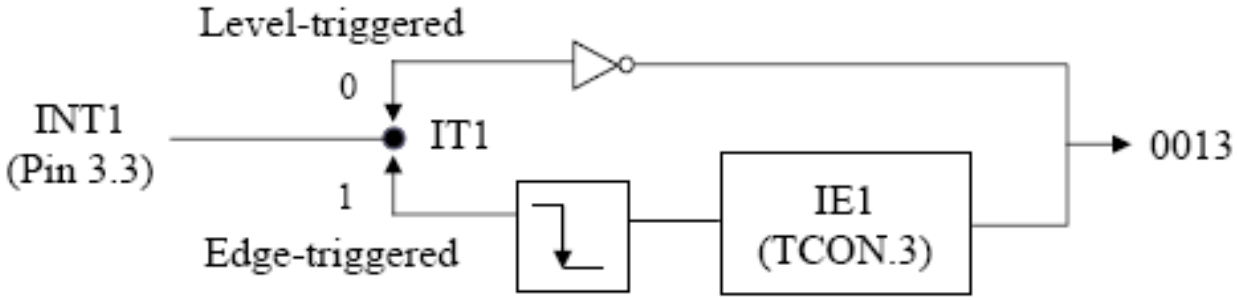
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Timer 1		Timer0		for Interrupt			

EXTERNAL INTERRUPT PROGRAMMING

Activation of INT0

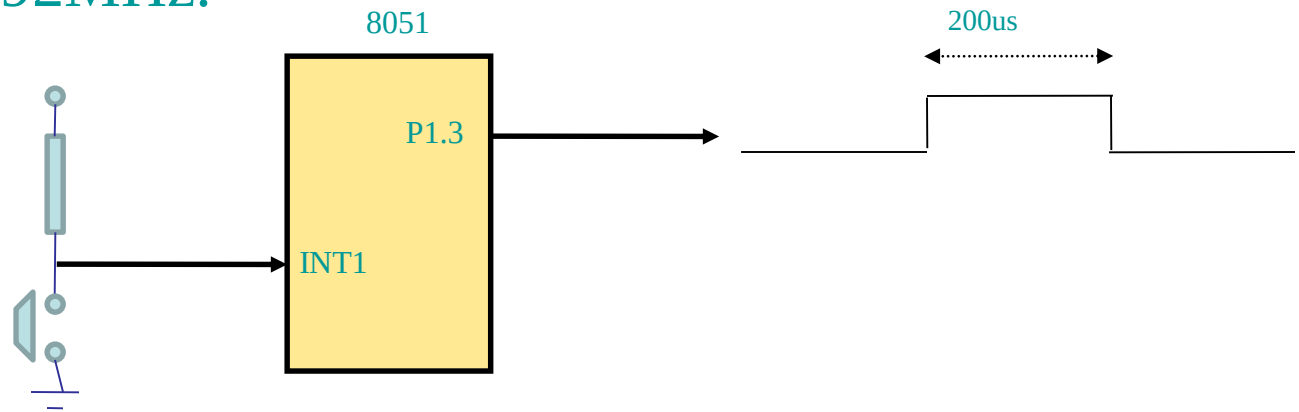


Activation of INT1



Example 1 (Example 11-5): Level-triggered interrupt

Assume that the INT1 pin is connected to a switch that is normally high. Whenever it goes low, it should turn on an LED connected at P1.3. When it turn on it should stay on for 200µs (use the software to create the delay). As long as the switch is pressed low, the LED should stay on. Write a program to implement the task. Assume the crystal frequency to be 11.0592MHz.



EXTERNAL INTERRUPT PROGRAMMING

Machine Cycle = (1/12 x crystal frequency) $^{-1}$

= (1/12 x 11.0592M) $^{-1}$

= 1.085µs

TCN = 200µs/1.085µs

= 184

TCN = 1 + (2 x R0)

184 = 1 + (2R0)

R0 = 92

EXTERNAL INTERRUPT PROGRAMMING

```
        ORG     0000H
        LJMP    MAIN            ; jump to main program
        ORG     0013H           ; external 1 interrupt vector address
        LJMP    INT1_ISR        ; jump to ISR

        ORG     0030H
MAIN:   MOV     IE, #10000100B  ; enable external 1 interrupt
        SJMP    $               ; wait until switch pressed low

INT1_ISR: SETB  P1.3            ; on the LED
        MOV     R3, #92
BACK:   DJNZ    R3, BACK        ; delay for 200μs
        CLR     P1.3            ; off the LED
        RETI                    ; return from interrupt

        END
```

EXTERNAL INTERRUPT PROGRAMMING

Example 2:

Edge-triggered interrupt

```
        ORG 0000H
        LJMP MAIN
;interrupt service routine (ISR)
;for hardware external interrupt INT1
        ORG 0013H
        SETB P1.3
        MOV R3,#255
BACK:   DJNZ R3,BACK
        CLR P1.3
        RETI
;main program for initialization
        ORG 30H
MAIN:   SETB TCON.2          ;on negative edge of INT1
        MOV IE,#10000100B
HERE:   SJMP HERE
        END
```

What is the above program for ? (**Example 11-6**)

Example 3:

• Using interrupts, design an 8051 furnace controller that keeps a building around 20 ^{0}C. The furnace ON/OFF solenoid is connected to P1.7 such that

• P1.7 = 1 for solenoid engaged (furnace ON)

• P1.7 = 0 for solenoid disengaged (furnace OFF).

• Temperature sensors are connect to INT0 and INT1 and provide HOT and COLD signals, respectively. The program should turn on the furnace for T<19 ^{0}C and turn it off for T>21 ^{0}C.

EXTERNAL INTERRUPT PROGRAMMING

Example 3:

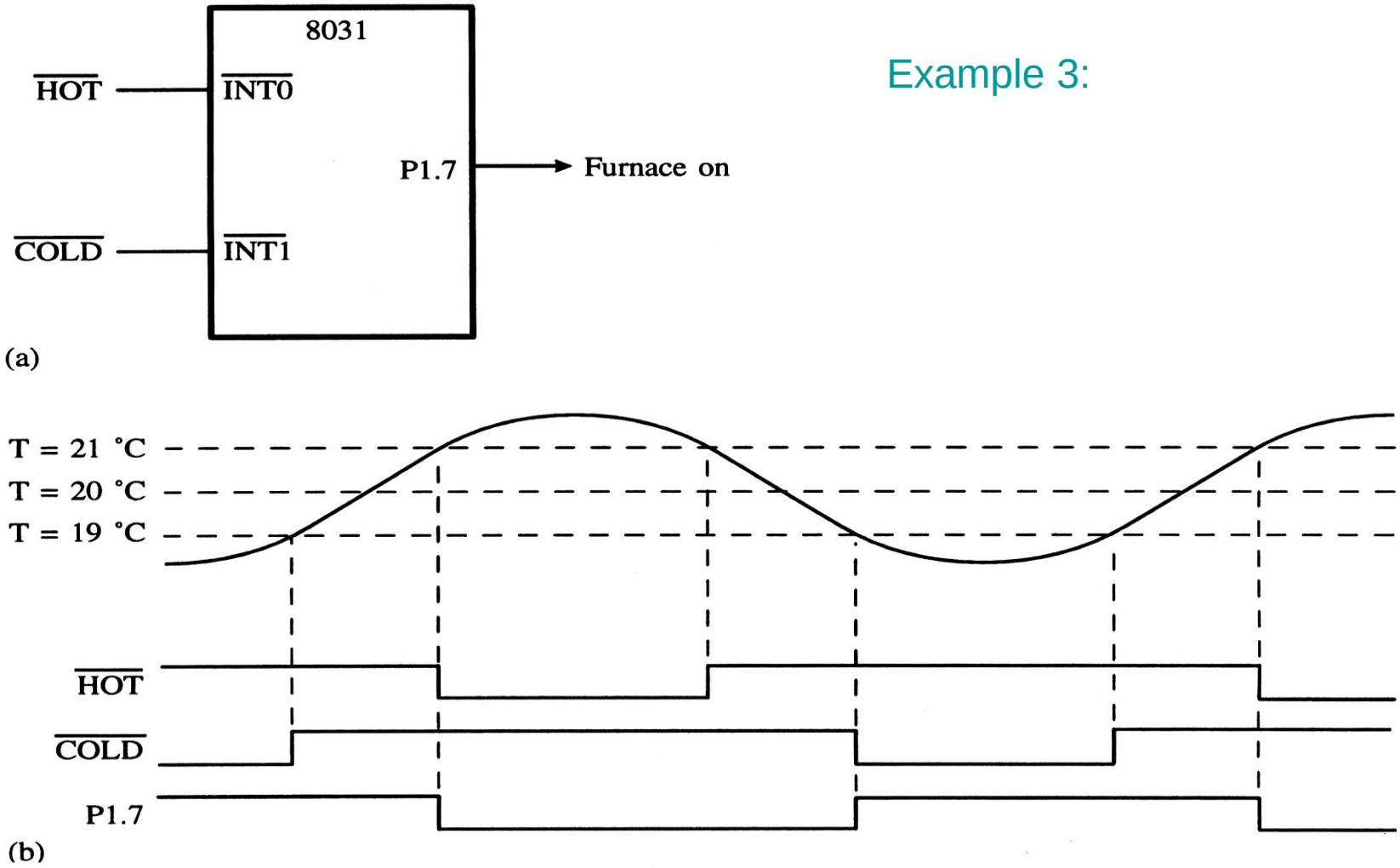


FIGURE 6–5
Furnace example. (a) Hardware connections (b) Timing.

EXTERNAL INTERRUPT PROGRAMMING

```
        ORG 0000H
        LJMP main
        ORG 0003h
x0isr:  CLR P1.7
        RETI
        ORG 0013H
x1isr:  SETB P1.7
        RETI
        ORG 0030H
main:   MOV IE, #85h
        SETB IT0
        SETB IT1
        SETB P1.7
        JB P3.2, skip
        CLR P1.7
skip:   SJMP $
        END
```

SERIAL PORT INTERRUPT PROGRAMMING

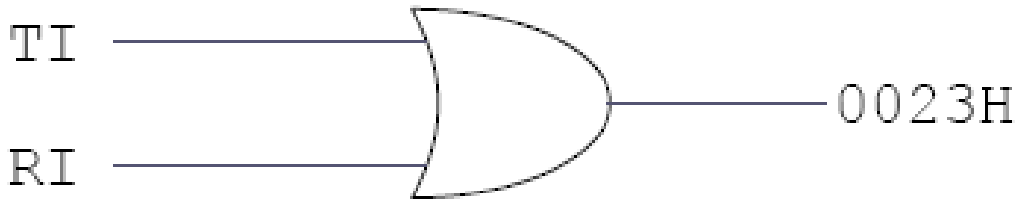
- In serial port interrupt, TI and RI will send an interrupt signal when one of them is raised.

- Before servicing the interrupt, the ISR must examine the TI and RI flags to see which one caused the interrupt and respond accordingly.

- TI and RI flags will not be cleared automatically in the ISR.

SERIAL PORT INTERRUPT PROGRAMMING

- In the 8051 there is only one interrupt set aside for serial communication
- This interrupt is used to both send and receive data
- If the interrupt bit in the IE register (IE.4) is enabled, when RI or TI is raised the 8051 gets interrupted and jumps to memory location 0023H to execute the ISR



☞ Need to check which one caused the interrupt

- Examples of serial port interrupt programming:

Example 11-8:

Write a program in which the 8051 reads data from P1 and writes it to P2 continuously while giving a copy of it to the serial COM port to be transferred serially. Assume that crystal frequency is 11.0592MHz and SMOD = 0. Set the baud rate at 9600.

$$\text{TH1} = 256 - \frac{2^{SMOD} \times CF}{384 \times Baud}$$

$$= 256 - \frac{11.0592\text{MHz}}{384 \times 9600}$$

$$= 253/\text{FDH}$$

```
        ORG     0000H
        LJMP    MAIN            ; jump to main program
        ORG     0023H           ; serial port interrupt vector add
        LJMP    SP_ISR

        ORG     0030H
MAIN:   MOV     IE, #10010000B  ; enable serial port interrupt
        MOV     P1, #0FFH       ; make P1 as input
        MOV     TMOD, #20H      ; configure timer
        MOV     TH1, #0FDH      ; load the reload value
        MOV     SCON, #50H      ; configure serial port mode
        SETB    TR1             ; start timer

HERE:   MOV     A, P1           ; send data to A
        MOV     SBUF, A         ; send data to COM port
        MOV     P2, A           ; send data to P2
        SJMP    HERE            ; do it continuously

SP_ISR: JB      TI, TRANS       ; check if TI = 1
        MOV     A, SBUF         ; put the received data in ACC
        CLR     RI              ; clear flag
        RETI                    ; return from interrupt
TRANS:  CLR     TI              ; clear flag
        RETI                    ; return from interrupt
        END
```

Example 11-9:

Write a program in which the 8051 gets data from P1 and sends it to P2 continuously while incoming data from the serial port is send to P0. Assume crystal frequency to be 11.0592MHz and SMOD = 1. Set the baud rate at 4800.

$$\text{TH1} = 256 - \frac{2^{SMOD} \times CF}{384 \times Baud}$$

$$= 256 - \frac{2 \times 11.0592\text{MHz}}{384 \times 4800}$$

$$= \text{F4H}$$

SERIAL PORT INTERRUPT PROGRAMMING

```
        ORG     0000H
        LJMP    MAIN            ; jump to main program
        ORG     0023H           ; serial port interrupt vector
                                  address
        LJMP    SP_ISR

        ORG     0030H
MAIN:   MOV     IE, #10010000B  ; enable serial port interrupt
        MOV     P1, #0FFH       ; make P1 as input
        MOV     TMOD, #20H      ; configure timer
        MOV     TH1, #0F4H      ; load the reload value
        MOV     SCON, #50H      ; configure serial port mode
        SETB    TRI             ; start timer
HERE:   MOV     P2, P1          ; put data from P1 to P2
        SJMP    HERE            ; do it continuously

SP_ISR: JB      TI, TRANS       ; check if TI =1
        MOV     P0, SBUF        ; put the receive data in P0
        CLR     RI              ; clear flag
        RETI                    ; return from interrupt
TRANS:  CLR     TI              ; clear flag
        RETI                    ; return from interrupt
        END
```

Example 11-10:

Write a program using interrupts to do the following:

(a) Receive data serially and sent it to P0

(b) Have P1 port read and transmitted serially, and a copy given to P2

(c) Make timer 0 generate a square wave of 5kHz frequency on P0.1

Assume that XTAL=11.0592MHz. Set the baud rate at 4800.

SERIAL PORT INTERRUPT PROGRAMMING

```
        ORG 0000H
        LJMP MAIN
        ORG 000BH          ;ISR for timer 0
        CPL P0.1           ;toggle P0.1
        RETI               ;return from ISR
        ORG 0023H
        LJMP SERIAL        ;jump to serial interrupt ISR
        ORG 30H
MAIN:   MOV P1, #0FFH      ;make P1 an input port
        MOV TMOD, #22H     ;timer 1,mode 2(auto reload)
        MOV TH1, #0F6H     ;4800 baud rate
        MOV SCON, #50H     ;8-bit, 1 stop, ren enabled
        MOV TH0, #-92      ;for 5kHZ wave
        MOV IE, 10010010B  ;enable serial int.
        SETB TR1           ;start timer 1
        SETB TR0           ;start timer 0
BACK:   MOV A, P1          ;read data from port 1
        MOV SBUF, A        ;give a copy to SBUF
        MOV P2, A          ;send it to P2
        SJMP BACK          ;stay in loop indefinitely
```

SERIAL PORT INTERRUPT PROGRAMMING

```
;-----------------SERIAL PORT ISR
            ORG 100H
SERIAL:JB TI, TRANS          ;jump if TI is high
            MOV A, SBUF      ;otherwise due to receive
            MOV P0, A        ;send serial data to P0
            CLR RI           ;clear RI since CPU doesn't
            RETI             ;return from ISR
TRANS: CLR TI                ;clear TI since CPU doesn't
            RETI             ;return from ISR
            END
```