

## LESSON 9: WRITING PROGRAM4: 8051's INTERRUPTS

---

Objectives:

1. Study interrupts process, interrupt vector, how to use interrupt with 8051SBC.
2. Write a program that uses interrupt.

The 8051 compatible microcontroller provides interrupt sources generated by internal timer, the UART and external input pins. The interrupt vectors of these sources are predefined locations. These locations are located at low address space. Table below shows the interrupt vectors.

Interrupt Source	Vector Address
IE0 (External interrupt 0, INT0 pin)	0003H
TF0 (Timer0 interrupt)	000BH
IE1 (External interrupt 1, INT1 pin)	0013H
TF1 (Timer1 interrupt)	001BH
R1 or T1 (Receiver or Transmit interrupt)	0023H
TF2 or EXF2 (Timer2 or EXF2 interrupt)	002BH

All of above interrupts are maskable. The associated enable bits can be set to enable or cleared to disable. When RESET all of them is disabled.

Since our 8051SBC has ROM monitor located from 0000H to 7FFFH. To provide user defined interrupt service routine, these interrupt vectors are relocated to RAM space started from 8000H. Here is the new vector address.

Interrupt Source	Vector Address
IE0 (External interrupt 0, INT0 pin)	8003H
TF0 (Timer0 interrupt)	800BH
IE1 (External interrupt 1, INT1 pin)	8013H
TF1 (Timer1 interrupt)	801BH
R1 or T1 (Receiver or Transmit interrupt)	8023H
TF2 or EXF2 (Timer2 or EXF2 interrupt)	802BH

If we look the monitor code at reset address, we will see jump instructions of these vectors to the new location in RAM! We can say the OFFSET address for interrupt vector is 8000H. Let us try disassemble the code from address 0000H.

New memory location: 0				
ADDR:0000> Disassemble				
0000:	02	08	AA	LJMP 08AA
0003:	02	80	03	<b>LJMP 8003</b>

```

0006: 8E 82      MOV    DPL, R6
0008: 8F 83      MOV    DPH, R7
000A: 22         RET
000B: 02 80 0B   LJMP  800B
000E: AE 82      MOV    R6, DPL
0010: AF 83      MOV    R7, DPH
0012: 22         RET
0013: 02 80 13   LJMP  8013
0016: 74 2D      MOV    A, #2D
0018: 01 73      AJMP  0073
001A: 00         NOP
001B: 02 80 1B   LJMP  801B
001E: 11 73      ACALL 0073
0020: 01 71      AJMP  0071
0022: 00         NOP
0023: 02 80 23   LJMP  8023
0026: 11 16      ACALL 0016
0028: 01 71      AJMP  0071

ADDR: 002A>

```

### Interrupt Enable Register

The interrupt enable register IE is bit addressable register that used to enable the interrupt source.

(MSB)							(LSB)
EA	-	ET2	ES	ET1	EX1	ET0	EX0

EA (IE.7) is global enable bit. We must set it with any of enable bits to enable interrupt.

For example to enable EX0, external interrupt, we must set EA and EX0.

- ET2 (IE.5) is Timer2 interrupt enable bit.
- ES (IE.4) is Serial port interrupt enable bit.
- ET1 (IE.3) is Timer1 interrupt enable bit.
- EX1 (IE.2) is External Interrupt1 enable bit.
- ET0 (IE.1) is Timer0 interrupt enable bit.
- EX0 (IE.0) is External Interrupt0 enable bit.

### Interrupt Flag Bits

Table below shows associated interrupts flag. The flag is set when the CPU hardware meets appropriate condition. For example TF0 is set when timer0 is overflow.

The CPU polls these flags every machine cycle at state 5 Phase 2. If ET0 is set (timer0 interrupt was enabled), CPU will save current Program Counter to STACK memory and jump to address 000BH. After finished executing interrupt service routine, the RETI,

return from interrupt will retrieve top of STACK and load to Program Counter, so the CPU will return to main program. Thus we can say that the interrupt service routine is subroutine that was called when specific hardware condition was met.

<b>Interrupt</b>	<b>Flag</b>	<b>SFR and Bit position</b>
External 0	IE0	TCON.1
External 1	IE1	TCON.3
Timer 1	TF1	TCON.7
Timer 0	TF0	TCON.5
Serial Port	TI	SCON.1
Serial Port	RI	SCON.0
Timer 2	TF2	T2CON.7
Timer 2	EXF2	T2CON.6

### TIMER MODE control register

(MSB)				(LSB)			
GATE	C/T*	M1	M0	GATE	C/T*	M1	M0
<b>Timer1</b>				<b>Timer0</b>			

**GATE** Gating control bit. When set, Timer is enabled only INTx is high and TRx is set.

**C/T\*** When cleared, Timer operation (input from internal clock, XTAL/12) is enabled. Set for counter operation (input clock from T0 or T1 pin).

M0 and M1 are operating MODE bits setting.

<b>M1</b>	<b>M0</b>	<b>Mode</b>	<b>Operating Mode</b>
0	0	0	13 bit timer/counter
0	1	1	16-bit timer/counter
1	0	2	8-bit Auto reload
1	1	3	Split Timer mode
1	1	3	Timer1 stopped

### TIMER CONTROL register

(MSB)				(LSB)			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

**TF1** TCON.7 Timer1 overflow flag. Set by hardware on overflow. Cleared by hardware when CPU jumps to service routine.

**TR1** TCON.6 Timer1 run control bit. Set/cleared by software to start/stop timer.

TF0 TCON.5 Timer0 overflow flag. Set by hardware on overflow. Cleared by hardware when CPU jumps to service routine.

TR0 TCON.4 Timer0 run control bit. Set/cleared by software to start/stop timer.

IE1 TCON.3 Interrupt 1 edge flag. Set by hardware when edge signal detected. Cleared by hardware when interrupt was processed.

IT1 TCON.2 Interrupt 1 signal type control bit. Set for falling edge, cleared for low level triggered.

IE0 TCON.1 Interrupt 0 edge flag. Set by hardware when edge signal detected. Cleared by hardware when interrupt was processed.

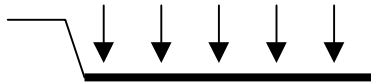
IT0 TCON.0 Interrupt 0 signal type control bit. Set for falling edge, cleared for low level triggered.

Note:

Falling Edge trigger signal



Low level trigger signal



### EXERCISE 9-1:

1. Test below program with LCD onboard.

```
$mod51  
  
led      equ P1.7  
cr       equ 13  
gpio1    equ 100h  
gpio2    equ 200h  
  
        dseg at 20h  
  
flag1:   ds 1  
  
        dseg at 50h  
  
tick:    ds 1  
sec100:  ds 1
```

```

sec:    ds 1
min:    ds 1
hour:   ds 1

        cseg at 8000h
        jmp start

        org 800Bh
        jmp service_timer0_interrupt

;---- initialization code -----
start:  orl tmod,#1 ; set timer0 to model

        mov flag1,#0
        mov sec,#0
        mov min,#59h
        mov hour,#17h
        call initlcd
        setb ea      ; set bit global interrupt
        setb et0    ; enable timer0 interrupt
        setb tr0    ; run timer0

;----- main loop running -----
main:   jmp $        ; wait here until timer0 overflow

service_timer0_interrupt:
        push acc
        push b
        push psw

        orl th0,#0dch ; reload timer0 with DC00H
        call update_clock
        call print_time_lcd

        pop psw
        pop b
        pop acc
        reti

update_clock:

        inc sec100
        mov a,sec100
        cjne a,#100,exit2
        mov sec100,#0

        setb flag1.0

        mov a,sec
        add a,#1
        da a

```

```

    mov sec,a
    cjne a,#60h,exit2
    mov sec,#0

    mov a,min
    add a,#1
    da a
    mov min,a
    cjne a,#60h,exit2
    mov min,#0

    mov a,hour
    add a,#1
    da a
    mov hour,a
    cjne a,#24h,exit2
    mov hour,#0
exit2:
    ret

print_time_lcd:

    jnb flag1.0,exit_print_time
    clr flag1.0

print_time1:
    mov a,#0
    mov b,#0
    call goto_xy ; set position (0,0)
    mov a,hour
    call print_A_lcd
    mov a,#':'
    call putch_lcd
    mov a,min
    call print_A_lcd
    mov a,#':'
    call putch_lcd
    mov a,sec
    call print_A_lcd

exit_print_time:

    ret

; print content of accumulator on LCD
; entry: BCD digit

print_A_lcd:
    push acc
    anl a,#0f0h
    swap a

```

```

        add a,#'0'
        call putch_lcd
        pop acc
        anl a,#0fh
        add a,#'0'
        call putch_lcd
        ret
$include(lcddrv.asm)

        end

```

2. Modify the code to print DATE-MONTH-YEAR.
3. Print text "Timer0 Interrupt" on second line of the LCD.

Above example is called interrupt driven tasks. The foreground task is running in the main loop. The background task is timer interrupt running. We see that background task is executing every 10ms with timer0 overflow.

Now suppose we want to use onboard keypad for time adjusting. We can place it in foreground task with less priority. The highest priority still is timer0 interrupt.

### EXERCISE 9-2:

1. Edit and assemble below program.

```

$mod51

led      equ P1.7
cr       equ 13
gpio1    equ 100h
gpio2    equ 200h

        dseg at 20h

flag1:   ds 1

        dseg at 50h

tick:    ds 1
sec100:  ds 1
sec:     ds 1
min:     ds 1
hour:    ds 1

        cseg at 8000h

```

```

    jmp start

    org 800Bh
    jmp service_timer0_interrupt

;---- initialization code -----
start:  orl tmod,#1 ; set timer0 to model

    mov flag1,#0
    mov sec,#0
    mov min,#59h
    mov hour,#17h
    call initlcd
    setb ea      ; set bit global interrupt
    setb et0    ; enable timer0 interrupt
    setb tr0    ; run timer0

;----- main loop running -----
main:   call scan_key
        jmp main

service_timer0_interrupt:
    push acc
    push b
    push psw

    orl th0,#0dch ; reload timer0 with DC00H
    call update_clock
    call print_time_lcd

    pop psw
    pop b
    pop acc
    reti

update_clock:

    inc sec100
    mov a,sec100
    cjne a,#100,exit2
    mov sec100,#0

    setb flag1.0

    mov a,sec
    add a,#1
    da a

```

```

    mov sec,a
    cjne a,#60h,exit2
    mov sec,#0

    mov a,min
    add a,#1
    da a
    mov min,a
    cjne a,#60h,exit2
    mov min,#0

    mov a,hour
    add a,#1
    da a
    mov hour,a
    cjne a,#24h,exit2
    mov hour,#0
exit2:
    ret

print_time_lcd:

    jnb flag1.0,exit_print_time
    clr flag1.0

print_time1:
    mov a,#0
    mov b,#0
    call goto_xy ; set position (0,0)
    mov a,hour
    call print_A_lcd
    mov a,#':'
    call putch_lcd
    mov a,min
    call print_A_lcd
    mov a,#':'
    call putch_lcd
    mov a,sec
    call print_A_lcd

exit_print_time:

    ret

; print content of accumulator on LCD
; entry: BCD digit

```

```

print_A_lcd:
    push acc
    anl a,#0f0h
    swap a
    add a,#'0'
    call putch_lcd
    pop acc
    anl a,#0fh
    add a,#'0'
    call putch_lcd
    ret

adjust_min:  mov a,min
             add a,#1
             da a
             mov min,a
             cjne a,#60h,adjust1
             mov min,#0
adjust1:    call print_time1
             ret

debounce:  mov r7,#0
             djnz r7,$
             ret

; foreground task
scan_key:  call check_key
             jnc scan_key ; still pressed?
             call debounce ; debounce

until_press:
    call check_key
    jc until_press
    call debounce
    call check_key

    cjne a,#0efh,key1 ; S3 key?
    call adjust_min
    ret

key1:
    ret

; read GPIO2
; exit: C=0 key pressed, A=key value
;       C=1 no key pressed

```

```

check_key:  mov  dptr,#gpio2
            movx a,@dptr
            orl  a,#0fh    ; maskout low nibble
            cjne a,#0ffh,key_pressed
            setb c
            ret           ; no key pressed

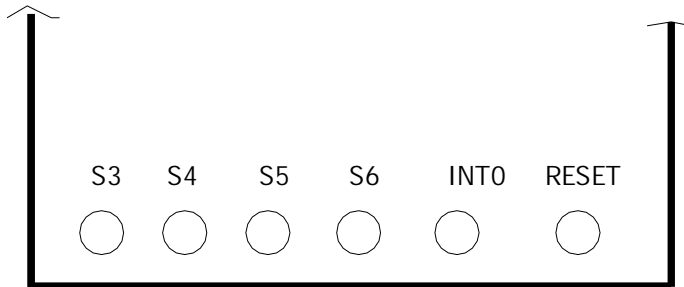
key_pressed: clr  c        ; return key and c=0
            ret

$include(lcddrv.asm)

            end

```

3. Add more keys to adjust HOUR, says S4 for HOUR adjusting.



3. The 8051SBC has onboard INT0 key, next to RESET key. Add the service routine that responds external interrupt 0 with falling-edge trigger. The service routine will clear LCD. See above table for external interrupt enable bit.