Microprocessor Lab-work #6

Multi-Interrupts at different priorities 100-11-14

[1] **Subject and goals**

(a) operations with two external interrupts and two timer interrupts

(b) built-in counter/timer setup and control for timing application

(c) time-up event detection by timer interrupt

(d) key entry event by external interrupt

(e) priority resolving among multi-interrupt

[2] **Preparations**

(a) **Refer to the ckt schematic diagram**:

(a.1) data path from 51CPU to the target LED modules

(a.2) data path from external interrupts to 51CPU

(b) **Datasheets reading**:

(b.1) 89c51 reference manual:

sections concerning the built-in timers

sections concerning the external interrupts

(c) **Readiness-evaluation:**

Can you or can you not

(c.1) check if the target module (e.g., any 7-seg LED digit) is working properly or not by manually wiring the circuitry?

(c.2) check if any of the switches in 1x4 and 4x4 units is working properly by manual operations?

(c.2) activate the timer for timing in any way as required?

(c.2) carry out trouble shooting along the data path from 89c51 CPU to the target module when the lab-work isn't going as expected?

How will you do that?

[3] **Lab-work for all:**

(a) **Operating Procedure**

TASK 1 (1.1) two switches in the 4x1 unit are used as external sources respectively for $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$, for controlling the activity of a discrete LED unit

(1.2) two timers using interrupts as time-out event indication are deployed for controlling the display of the 7-segment LED unit

(1.3) the 7-seg LED module is divided into two parts for timing display as required below

DIG6:5 varying from **00** to **59** based on TMR0

DIG4:1 varying from **00:00** to **59:99** based on TMR1

DIG6		DIG4		DIG1	
10^1 sec	10^0 sec	10^1 sec	10^0 sec	10^{-1} sec	10^{-2} sec

(1.4) a unit of discrete LEDs responds to $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$ interrupts as follows

normal state: ON-OFF ⇔ OFF-ON alternation between the higher and lower halves of the unit

on $\overline{\text{INT0}}$: the unit undertaking a left-to-right ripple display

on $\overline{\text{INT1}}$: the unit undertaking a right-to-left ripple display

(1.5) priority assignment (high to low): TMR1>INT1>INT0>TMR0

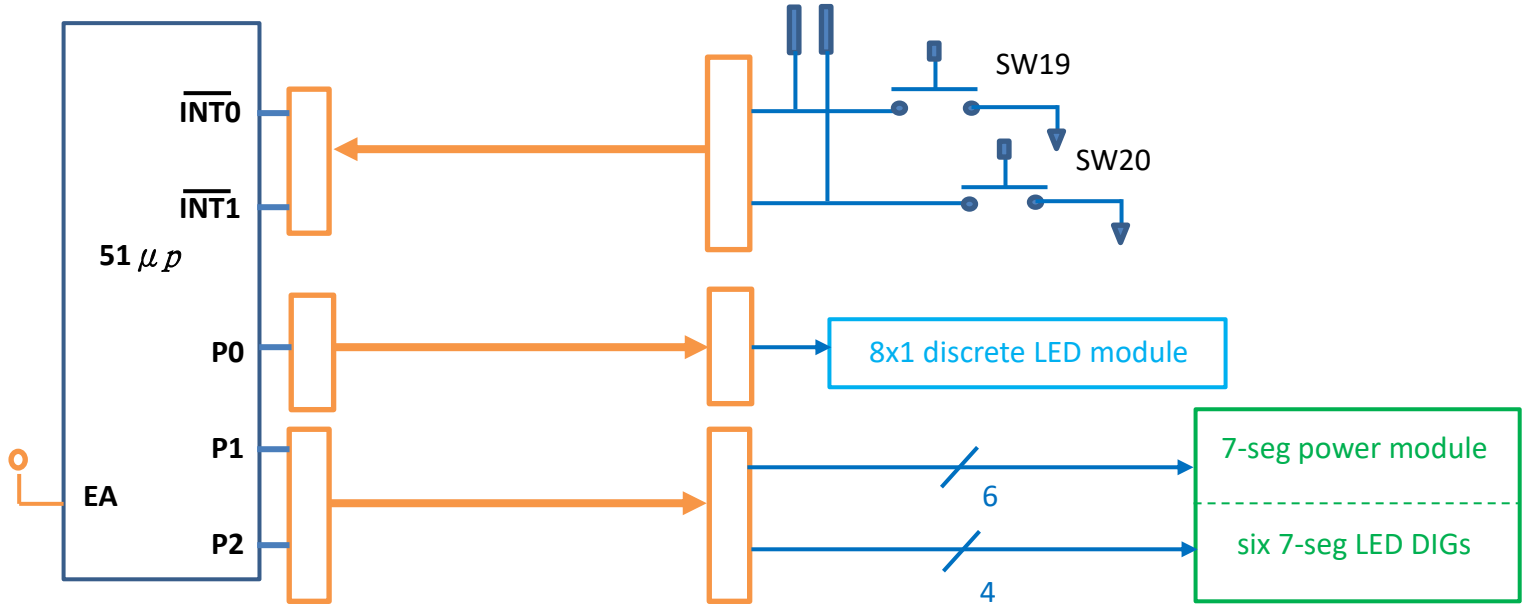
[is this a correct priority order achievable under 51-architecture???

how's the priority order arranged in the sample code???]

TASK2 Refer to the sample codes for task1, write the codes to achieve the same performances without using interrupt mechanism; small details of the task specifications will be lab-team-dependent which will be announced on site.

(a.1) jumper-wiring for a 4x1 key unit and LED modules setup

Refer to the schematic circuit diagram, do all jumper-wiring necessary for setting up the circuitry as required below.



(a.2) code preparation:

** edit the following sample 51 assembly codes,

** get the code ready for execution under IDE51

```
; TASK1    using interrupt mechanisms

    ; ============================
    ; (1) 4 interrupt events in action
    ; (2) 1x8 LEDs
    ;       normal ON-OFF ⇔ OFF-ON
    ;       INT0 → L2R scanning
    ;       INT1 → R2L scanning
    ; (3) 7-seg LEDs
    ;       DIG6-5: timing by second (TF1)
    ;       DIG4-1: timing by 10ms (TF0)
    ; (4) priority
    ;       priority order of the 4
    ;       interrupts    ???
    ; ============================
    ; port0: 1x8 discrete LEDs control
    ; port1: power-SW control for
    ;            7-seg LED module
    ; port 2: pattern control for
    ;          timing display on 7-seg LEDs
    ; 30H:     TMR1 10-2sec-register
    ; 31H:     TMR0 sec-register
    ; 32H:     TMR0 10-2sec-register
    ; 33H:     TMR0 10-2sec time-up flag
    ; 34H:     TMR1 10-2sec time-up flag
    ; 35H:     TMR1 counter: 0-19
    ; 36H:   normal pattern for 1x8 LEDs
    ; BITMAP 0H:     INT0 flag
    ; BITMAP 1H:     INT1 flag
    ; BITMAP 2H:     left-to-right Cy
    ; BITMAP 3H:     right-to-left Cy
    ; 40H:     1x8 LED pattern for INT0
    ; 41H:     1x8 LED pattern for INT1
            org     0
            jmp     start
            org     03H
            jmp     INT0
            org     0BH
            jmp     TMR0
            org     13H
            jmp     INT1
            org     1BH
            jmp     TMR1
start:
            mov     sp, #50H
            mov     TMOD, #11H
            mov     TH0, #>(65536-10000)
            mov     TL0, #<(65536-10000)
            mov     TH1, #>(65536-50000)
            mov     TL1, #<(65536-50000)
            mov     IE, #8FH
            mov     IP, #0AH
            mov      30H, #0
            mov      31H, #0
            mov      32H, #0
            mov      33H, #0
            mov      34H, #0
            mov      35H, #0
            clr      0H
            clr      1H
            mov      40H, #0FFH
            mov      41H, #0FFH

            setb     TR0
            setb     TR1
            ;mov      P1, #0H
```

```
        ;mov    P2, #0H
        clr     C
        mov     2H, C
        mov     3H, C
display_go:

normcycle:
        mov     36H, #0FH
        mov     R4, #2
        mov     R5, #14
halfcycle:
        mov     P0, 36H
int_in_session:
        mov     R6, #30
DIG6:   mov     A, 30H
        mov     B, #10
        div     A, B
        mov     P1, #0DFH
        mov     P2, A
        call    delay
DIG5:   mov     A, B
        mov     P1, #0EFH
        mov     P2, A
        call    delay
DIG4:   mov     A, 31H
        mov     B, #10
        div     A, B
        mov     P1, #0F7H
        mov     P2, A
        call    delay
DIG3:   mov     A, B
        mov     P1, #0FBH
        mov     P2, A
        call    delay
Dig2:   mov     A, 32H
        mov     B, #10
        div     A, B
        mov     P1, #0FDH
        mov     P2, A
        call    delay
DIG1:   mov     A, B
        mov     P1, #0FEH
        mov     P2, A
        call    delay
        djnz    R6, DIG6
        ;mov    36H, #0F0H
int1test:
        jnb     1H, int0test
        mov     A, 41H
        push    PSW
        mov     C, 3H
        rrc     A
        mov     3H, C
        pop     PSW
        mov     P0, A
        mov     41H, A
        jb      3H, int1_in_session
        clr     1H
int0test:
        jnb     0H, norm_cont
        mov     A, 40H
        push    PSW
        mov     C, 2H
        rlc     A
        mov     2H, C
        pop     PSW
        mov     P0, A
        mov     40H, A
        jb      2H, int0_in_session
```

```
            clr       0H
            jmp       norm_cont
int1_in_session:
int0_in_session:
            jmp       int_in_session
norm_cont:
            djnz      R5, midway
            mov       36H, #0F0H
            mov       R5, #14
            ;djnz      R4, halfcycle    ; Q0

            djnz      R4, midway       ; Q1
            jmp       display_go
midway:
            jmp       halfcycle        ; Q2

   delay:   ; ? how long should the
            ; delay be for visual
            ; satisfaction?
            ret
TMR0:       push      PSW
            push      A
          mov       TH0, #>(65536-10000)
          mov       TL0, #<(65536-10000)
            setb      TR0
            inc       32H
            mov       A, 32H
            cjne      A, #100, ext1
            mov       32H, #0
            inc       31H
            mov       A, 31H
            cjne      A, #60, ext1
            mov       31H, #0
 ext1:      pop       A
            pop       PSW
            reti

TMR1:       push      PSW
            push      A
          mov       TH1, #>(65536-50000)
          mov       TL1, #<(65536-50000)
            setb      TR1
            inc       35H
            mov       A, 35H
            cjne      A, #20, ext2
            mov       35H, #0
            inc       30H
            mov       A, 30H
            cjne      A, #60, ext2
            mov       30H, #0
  ext2:     pop       A
            pop       PSW
            reti
INT0:       jb        0H, intext0
            push      A
            push      PSW
            setb      0H
            pop       PSW
            pop       A
intext0:
            reti
INT1:       jb        1H, intext1
            push      A
            push      PSW
            setb      1H
            pop       PSW
            pop       A
intext1:
            reti
```

```
                end
; =========================

; TASK2: using S/W polling for event-
;              detection
;    coding on your own
, . . .GOOD LUCK!
```

(a.3) task execution:

** prepare the sample code, start the execution and observe circuit behaviors under IDE51 emulation.

** start trouble-shooting if necessary

Key: (1) checking along the data path in a stage by stage manner, from the start: inside of 89c51 to the end: the target module

(2) test the codes module by module, by masking irrelevant modules out, when things get out of hands, e.g.,

**) test TMR1 alone first

**) test TMR0 alone secondly

**) test both timers simultaneously then

**) test INT0 alone

**) test INT1 alone

**) test both external interrupts simultaneously

**) test all four subtasks all together

(b) **Observations**

(b.1) Is the sample code running well? If not, congratulate you that you have a chance for getting more experience in trouble-shooting.

(b.2) Please describe the execution flow structure if you would and could.

(b.3) Describe the purpose of bit 0, 1, 2, and 3 of the BITMAP; and the purpose of 40H and 41H.

(b.4) By the sample codes,

** right-to-left 1x8 LEDs scanning due to INT0 will not be taken over by left-to-right 1x8 LED scanning due to INT1;
the latter could only commence after the former is done.

** left-to-right 1x8 LED scanning due to INT1 will be taken over by right-to-left 1x8 LEDs scanning due to INT0;
only when the latter is done could the former resume scanning from where it was interrupted.

Make sure that you perfectly understand how this is controlled by the sample codes.

(b.5) By the sample codes, can INT0 interrupts the 1x8 LED scanning process due to INT1 for more than once?

(b.6) By the sample codes, can INT1 interrupts the 1x8 LED scanning process due to INT1 for more than once?

(b.7) By the sample codes, can INT0 interrupts the 1x8 LED scanning process due to INT0 for more than once?

(b.8) In the sample codes, the instruction lines handling DIG6-DIG1 display look pretty much the same except that control and pattern sources are different. Could you rewrite the code using a looping body of 6 rounds to do the same job?

(b.9) Is the push-pop pair for PSW in TMR0/1 interrupt handler necessary? Why or why not?

(b.10) In TASK1, several jobs have to be done so that visually all displays are simultaneously taken care:

** the display of six 7-seg digits with different patterns, under circuitry constraints

** the display of 1x8 discrete LEDs under different types of requests

** the restart of the timers

** timing record adjustment

** setting and clearing the flags for external interrupts

As can be seen, the jobs are taken care of either in the main code body or in the interrupt handler. How would you say about such

an allotment? Any better way of arrangement?

[4] **Comprehension evaluation**

(a) If not exploiting 89c51 interrupt mechanisms of INT0/1 and TMR0/1, how would time-up event be handled? How would the activities of the two switches be detected? How are the load balance among all subtasks (detection of 4 events, control over 2 types of LEDs, etc.) to be achieved? While writing the codes for the very same timing task without using timer interrupt, what kind of awkwardness may be encountered if any?