**Microprocessor Lab-work #7**

**Handling of Nested Interrupts** 100-11-14

[1] **Subject and goals**

(a) control for accepting further INT0 interrupts while an INT0-handler is at service.

(b) priority resolving among multi-interrupt

[2] **Preparations**

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

(a.1) 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 the 4x1 column is working properly by manual operations?

(c.3) activate the timer for timing in any way as required?

(c.4) figure out just how nested interrupting is to be realized? [By the design of 51-interrupt mechanism, an interrupt cannot interrupt itself; specifically, while the INT0 handler is at service, no further INT0 interrupts will be acknowledged by 51CPU.]

(c.5) 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** general requirements  (1.1) one switch in the 4x1 unit is used as external source for INT0  (1.2) each time the INT0 handler is invoked, it should  \*\* sustain for 10 seconds  \*\* allow further INT0 interrupts to get in(i.e., creating the situation of nested-interrupts) till a depth of 10 levels is reached  (1.3) using TMR0 interrupt as time-out event indication for 10-sec timing  (1.4) DIG1: time to go in the current INT0 interrupt level, counting down from 9 to 0 sec.  DIG2: depth of the current INT0 interrupt,  \* starting from 0 and increasing by 1 each time an INT0 occurs, no further increasing when 9 is reached,  \* decreasing by 1 from current depth of INT0 if no further new occurrence of INT0 when the down counting of DIG1 reaches 0.  DIG6 DIG5 DIG2 DIG1  level 100  # sec  (1.5) the 1x8 discrete LEDs exhibit ON-OFF ⬄ OFF-ON display at 1 Hz as the system routine.  \*\* wiring for proper setup of all modules \*\* code preparation \*\* code executing with ICE51  **TASK2** It is noted that as INT0 interrupts started getting nested, the routine display of the 1x8 LED unit got stalled. Refer to the sample codes for task1 and see how the defects could be removed. Modify the code lines, use extra data structures, deploy facilities supported by 51CPU, or even design an entirely new execution structure; do whatever you can. |

(a.1) jumper-wiring for the target module setup

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

[ **spared operation:** use one channel of LA-probe for grabbing INT0-signal due to activity of SW19]

SW19

**INT0**

1 SW20

**INT1**

**51μ*p***

**P0** 8x1 discrete module

7-seg power module

**EA P1** 4

six 7-seg LED DIGs

4

(a.2) code preparation:

\*\* edit the following sample 51 assembly codes, get the code ready for execution under IDE51

|  |
| --- |
| ; TASK1 nested INT0 interrupts  ; ============================  ; creating the context of nested  ; interrupt by relieving the at-service  ; flag during the interrupt cycle of  ; 51CPU  ; =============================  ; P1(7-4): power-SW control for  ; 7-seg LED module  ; P1(3-0): pattern control for  ; timing display on 7-seg LEDs  ; 30H: depth level for nested INT0  ; 31H: time left for current INT0  ; handler  ; 32H: TMR0 50ms counts  ; 33H: INT0 op. stack at 40H  ; 3AH: main stack at 60H  ; BITMAP 0H: flag of **level-1** INT0  org 0  jmp start  org 03H  jmp INT0  org 0BH  jmp TMR0  start:  mov sp, #60H  mov 3AH, SP  mov TMOD, #01H  ;mov TH0, #>(65536-50000)  ;mov TL0, #<(65536-50000)  mov IE, #83H  mov IP, #2H ;  orl TCON, #1  mov 30H, #0 ; depth  ;mov 31H, #10 ; time  ;mov 32H, #0 ; ms-cnt  mov 33H, #40H ; stack  setb 0H ; 1st INT0 arrv.  ;setb TR0  display\_go:  mov R1, #100  mov P0, #0F0H  call delay1  djnz R1, $-3  mov R0, #100  mov P0, #0FH  call delay1  djnz R0, $-3  jmp display\_go  TMR0: push IE  ANL IE, #80H ; ???  push PSW  push A  mov A, 30H  jz ext1  mov TH0, #>(65536-50000)  mov TL0, #<(65536-50000)  setb TR0  inc 32H  mov A, 32H  cjne A, #20, ext1  mov 32H, #0  mov A, 31H  jz ext1  ;jz treset\_10  dec 31H ;  ext1: pop A  pop PSW  pop IE  reti  ;treset\_10:  ; mov 31H, #10  ; jmp ext1  INT0:  call bounce\_wait ;  push A  push PSW  mov A, 30H  jz INT0\_1  ; ============== A  anl IE, #0FDH  mov 3AH, SP  mov SP, 33H ; stack40H  push 31H ; cnt\_dwn  push 32H ; 20cnt  push TH0 ; time left  push TL0  mov 33H, SP  mov SP, 3AH  orl IE, #2  ; ============== EOA  INT0\_1: inc 30H  mov 31H, #10 ; cnt\_dwn  mov 32H, #0 ; 20\_cnt  mov TH0, #>(65536-50000)  mov TL0, #<(65536-50000)  setb TR0  clr 0H  call fake\_exit ;  cntdwn:  mov A, 30H ; depth  orl A, #0D0H  mov P1, A  call delay  mov A, 31H ; cnt\_dwn  dec A  orl A, #0E0H  mov P1, A  call delay  mov A, 31H  jnz cntdwn  anl IE, #0FEH  dec 30H ; depth  mov A, 30H  jz INT0\_exit  ; ================= B  anl IE, #0FDH  mov 3AH, SP  mov SP, 33H  pop TL0 ;  pop TH0  pop 32H  pop 31H  mov 33H, SP  mov SP, 3AH  setb TR0 ; WHY?  orl IE, #2H  ; ================ EOB  INT0\_exit:  pop PSW  pop A  orl IE, #1  fake\_exit:  reti  delay:  push 0  mov R0, #200  djnz R0, $  pop 0  ret  delay1:  bounce\_wait:  push 0  push 1  mov R0, #20  dd1: mov R1, #250  djnz R1, $  djnz r0, dd1  pop 1  pop 0  ret  end  ; TASK2 [hints]  ; \* using TMR1 to trigger discrete  ; LEDs display during INT0 nesting  ; process  ; \* TMR1 with higher IP than INT0  ; lower IP than TMR0  ; \* TMR1 clocking at the rate the same  ; as that of pattern-switching on  ; discrete LEDs  ; \* all amendment in codes and data  ; structure thus incurred should be  ; taken into account  ; GOOD LUCK! |

(a.3) task execution:

\*\* 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, when things are getting somewhat ugly and messy, by masking irrelevant modules out, e.g., \*\*) test TMR0 alone

\*\*) test INT0 alone

\*\*) test joint-operation of INT0 and TMR0

1. **Observations**

(b.1) Is the sample code running well? If so, it’s too bad that nothing you could learn about from it.

If not, congratulate you that you have a chance for getting more experience in trouble-shooting.

(b.2) For each SW19 entry, the nested-level of INT0 should increase by 1, and yet at times it is seen on DIG2 LED a quantum leap of the level index. Check the INT0-signal recorded by the logic analyzer at the same moment; can you explain the interconnection between the two phenomena?

Is the subroutine “bounce\_wait” working at all? If the code line “call bounce\_wait” is removed, will the leaping of level-index be getting worse?

(b.3) Please describe the execution flow structure if you will.

(b.4) In the sample codes, a second STACK at 40H was used for stocking timing data for each level of INT0. Was it necessary? Can these data be stocked using the same STACK staring at 60H?

(b.5) In the sample codes, the depth limit of interrupt nesting was claimed to be 10; there was no precaution taken in the codes for preventing the nesting from more than 10 levels, though. How would you do that if the protection is to be enforced?

(b.6) Refer to the sample codes and consider the resources of 51CPU given, would you expect that 10-level of INT0-nesting attainable? Why or why not? And what is your estimation on the limit of INT0-nesting?

(b.7) By the sample codes, the execution proceeds as follows.

\*\* initially the 1x8 discrete LEDs perform routine display of ON-OFF ⬄ OFF-ON alternation, and DIG2 and DIG1 of the 7-seg unit shows none; it remains so as long as no INT0 occurs

\*\* when the 1st INT0 occurs,

\*\*) DIG2 shows level 1 and DIG1 starts counting down from 9 to 0

\*\*) the 1x8 LEDs do routine work as are

\*\*) if no further INT0 came in during the 9-to-0 countdown, it resumes the initial state

\*\* when an INT0 occurs during the down counting process of the current INT0,

\*\*) DIG2 shows level number incremented by 1 and DIG1 starts counting down from 9 to 0

\*\*) the 1x8 LEDs do routine work as are

\*\*) if no further INT0 came in during the 9-to-0 countdown, it resumes the previous state where

DIG2 shows the previous level number and DIG1 resumes counting down from where it was interrupted.

As can be seen, the routine display of 1x8 LEDs is stalled whenever an INT0 interrupt is being served. How would you remedy the problem?

[4] **Comprehension evaluation**

(a) Try rewriting the codes so that it is shorter and cleaner in code structure and less complicated in execution than the sample codes.

(b) What are done during the initiation stage of the codes?

(c) In the initiation stage of the codes, TMR0 setting and initiation was omitted, together with the initialization of time down-counting variable for INT0 . Why? (hint: when will TMR0 timing be needed?)

(d) In TMR0 handler, the instructions concerning the reset of down-counting variable was omitted. Why? What would be the consequence if they are back in actions?

(e) In TMR0 handler, was the disabling of INT0 necessary? Can that instruction line be removed without any harm to the task?

(f) In INT0 handler, how was further interrupts of INT0 being allowed? And when were they allowed during the execution of the handler?

(g) During certain portion of time when INT0 is running, further interrupts of INT0 will not be acknowledged. Why?

(h) In INT0 handler, TMR0 interrupt (of higher priority than INT0) was masked when there are STACK-switching between the main-stack and the timing-stack to be done. Why? (hints: what if TMR0 interrupt was not masked?)

(i) In INT0 handler, TMR0 will have to be restarted in certain cases. Explain why the line marked with **; ???** in the code block B is needed and why it is needed right there?