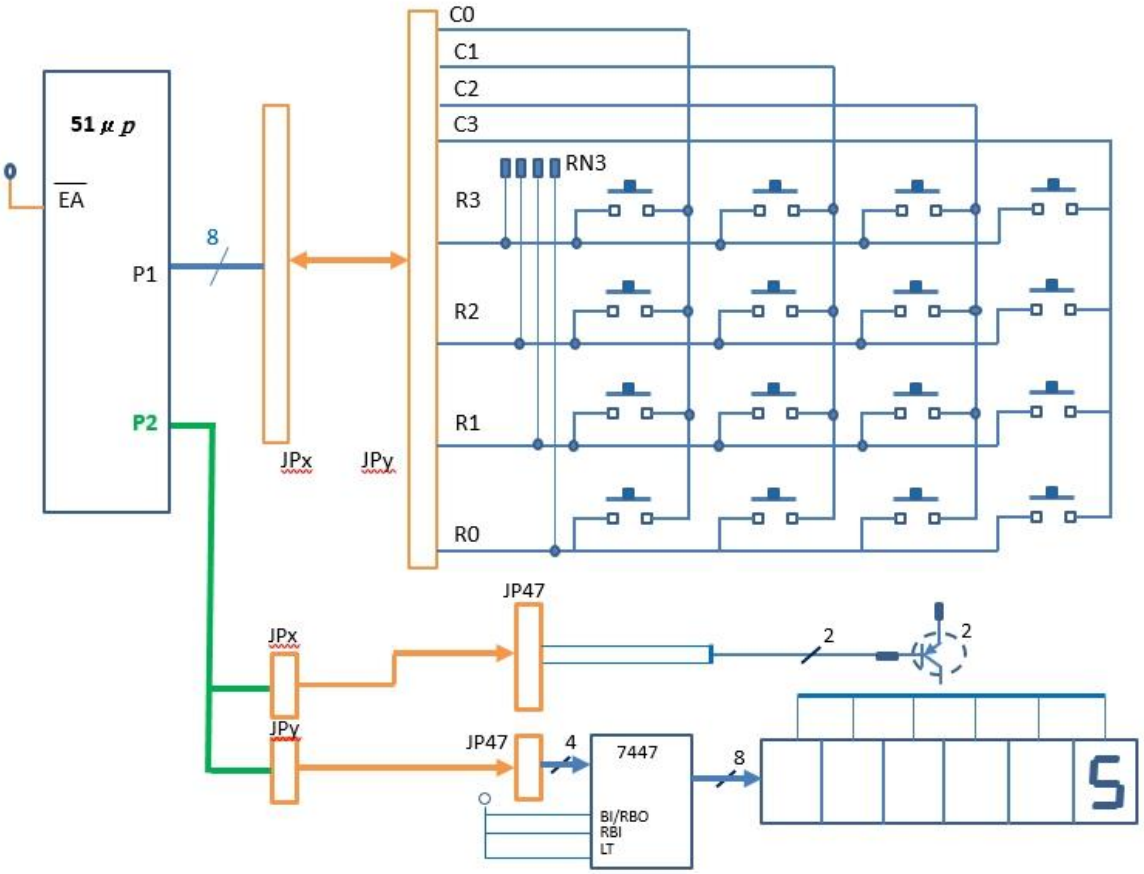
Microprocessor Lab 3 Report

0416106 彭敬樺
0416109 周才錢

Subject and Goal:

This lab is about using µ-Vision 51IDE residing on MegaWin82G516 to:

- Interconnect structure of a 4x4 keypad, together with scanning and encoding process
- Balance among several IO-control tasks that have to be executed in order repetitively.



Preparations:

- Power cable and required connection from the output to the led input is established. The color of LED-module is controlled by the output of port P2 that is connected to JP47. Port P1 will control the switch input key by connection it to JP28
- Check the correctness and check if there are any defective on the board by activating all 7-segment LED module using static/dynamic pattern display.

Operating Procedure:

- Jumper-wiring for circuit setup
- Check the 7-segment LED module to see if it's working or not by running code to turn all the light on.
- Code preparation

- Task execution:
  - Start IDE51 emulation,
  - Start execution and troubleshooting if necessary.

Code Preparation:

```
; ================================
; version 2 key-code display on two 7-seg
;       digits
;         12  13  14  15
;         08  09  10  11
;         04  05  06  07
;         00  01  02  03
; ================================
; port 1 for keypad scanning
;   P1: 0-3 output for column scanning
;   P1: 4-7 input for row reading
; port 2 for using two digits of 7-seg LED
          org   0
          mov  sp, #50H
          mov  R7, #15
          mov  P2, #0FFH
    col0:    mov  R6, #0
mov  P1, #0FEH
          mov  A, P1
          cpl   A
          jnz   keycoding
          call   display   ; ==??==
    col1:    mov  R6, #1
mov  P1, #0FDH
          mov  A, P1
          orl    A, #0FH
          swap  A
          cpl   A
          jnz   keycoding
          call   display   ; ==??==
    col2:    mov   R6, #2
mov  P1, #0FBH
          mov  A, P1
          orl    A, #0FH
          swap  A
          cpl   A
          jnz    keycoding
          call   display    ; ==??==
    col3:    mov   R6, #3
mov  P1, #0F7H
          mov  A, P1
          orl    A, #0FH
          swap  A
          cpl   A
          jz    col0
kecoding:      ; A: 0-9,A,B,C,D,E F
rr     A
anl   #0FH
jnb    A.2, cont
mov   A, #3
    cont:    jz     cont2
  mov   R7, A
      clr     A
    cont1:   add    A, #4
          djnz    R7, cont1
    cont2:   add    A, R6
          mov   R7, A
          call    display  ; ==??==
          jmp    col0
    display:
          push    PSW
          push    A
          push    F0H  ; push B??
          mov    A, R7
          mov    B, #10
          div     A, B
          anl     A, #0FH
          orl     A, #0D0H
          mov    P2, A
          call     delay
          mov    A, B
          anl     A, #0FH
```

```
        orl     A, #0E0H
        mov     P2, A
        call    delay
        pop     F0H
        pop     A
        pop     PSW
        ret
delay:  push    2
        push    3
        mov     R2, #20
xxx:    mov     R3, #250
        djnz    r3, $
        djnz    r2, xxx
        pop     3
        pop     2
        ret

end
```

Observation:

- The code is running well. All the wanted sequence react to the correct button being pressed in the switch.
- When running version 1, there are 1 times for the 89c51 to access the 7-seg LED digit for each key-code display and there are no more any access into the LED digit in between two key entries.
- When handling keypad scanning and the 7-segment LED module together, there are many difficulties in doing so due to the limitation for the LED module that have to be initialized one by one. The easiest way is to separate the two functions and call them sequentially one after another to avoid more confusion.
- The function 'jmp' in 89c51 is not really suitable for very long range jump that will exceed an amount of maximum that it can handle. The "call" function in this case can handle it better and we can return to the caller easier rather than designating different return address if the function has different return address for different caller.
- The better way to implement the 'display' function call is by putting the function after all keyboard scanning has been handled. In this case, we can reduced the code size by a few line, and ensure that the value of the register responsible for handling the LED segment is kept initialized.
- Without RN3 pull-up resistor array, then the keypad module will not work at all the reason is that this module relies on the difference of voltages between the source and destination. If there are no pull-up resistor array, the voltage all around the keypad and the responsible port will not change whether if the keypad is pressed or not
- In version 2, the value "15" appears on the LED digits at first because we initialize it to do so in the code, if we instead initialize it to zero, then we will see "00" as the first number shown.
- When the delay is elongated by 10 times the initial value, then we will be able to see the LED digit flicker. This phenomenon is actually happening when the initial value is used, however due to the speed of the initial duration, then our eyes cannot realize the fast flickering light and the phenomenon is not observed. When the delay is elongated, then the phenomenon is being enlarged,

Comprehensive evaluation:

- Modified code with loop applied to the scanning body:

```
org   0
          mov  sp, #50H
          mov  P2, #0FFH
starting_loop:
                        mov  R6, #0
                        mov  P1, #0FEH
                        mov R4, #4
scanning_loop:
          mov  A, P1
          orl    A, #0FH
          swap  A
          cpl   A
          jnz   keycoding

                        push    0E0H ;push A
                        mov A, P1
                        rl A
                        mov P1, A
                        pop    0E0H ;pop A

                        inc R6
                        djnz                R4,
scanning_loop
                        jz    display

keycoding:      ; A: 0-9,A,B,C,D,E F
rr     A
;;;;;;;;;;;;;;;;;;;;;;;;;;
anl    A, #0FH
push 0E0H ;push A
anl             A, #04H
jz              cont
;jnb    A.2, cont;;;;;;;;;;;;;;;;;;;;
pop 0E0H ;pop A
mov   A, #3
jmp newly_created
    cont:    pop 0E0H ;pop A
      jz     cont2

    newly_created:
mov   R7, A
   clr     A
 cont1:   add    A, #4
        djnz    R7, cont1
 cont2:   add    A, R6
        mov   R7, A
        call    display  ; ==??==
        jmp    starting_loop
 display:
        push    PSW
        push    0E0H ;push A
        push    0F0H  ; push B??
        mov    A, R7
        mov    B, #10
        div     AB
        anl     A, #0FH
        orl     A, #0D0H
        mov    P2, A
        call     delay
        mov    A, B
        anl     A, #0FH
        orl     A, #0E0H
        mov    P2, A
        call     delay
        pop     0F0H ;pop B
        pop     0E0H ;pop A
        pop     PSW
                        mov P1, R7
        ret


 delay:  push    2
        mov   R2, #100
        djnz   r2, $
        pop   2
        ret

end
```

- Usually one typical key entry last in the time unit of millisecond. There are no significant problem in this condition for the CPU to register more than once for the same key press. However, if this could become a problem, then we can determine if it is a valid input or just a noise by waiting for the input to be registered longer until the typical duration of a key pressed. When a key has been pressed for that determined duration, then we can say that it is registered, and we can determine if it is released by the same method to.