EE2401 微算機系統Fall 2019

HW#2 (8051 Instruction set, I/O) (10/14/2019)

Due date: 10/31/2019. Severe penalty will be given to late homework.

107061112 王昊文

Note:

(a) The homework will be graded based on your **documentation** and **demonstration**.

(b) For all **(Software Design)** problems, you are required to us **MCU8051IDE**simulators to simulate and verify your programs.

(c) You are required to **type** your homework (first the problem then your solution) by using a **word processor** and submit in .doc (or .docx) format under a filename **EE2401f19-hw2-student_no-vn.doc(or .docx)**, where **student_no** is your student number, e.g., **107061xxx** and **vn** is your version number, e.g., **v3**. You should **upload your .doc file** in **iLMS** by the specified deadline whenever you have a newer version. Follow the iLMS upload homework process to upload your file.

(d) The homework will be graded based on your **latest version**. Old version(s) will be discarded.

(e) Each homework assignment will have full score of 100 points. 5 points will be deducted if you do not comply with the naming convention. Severe grade penalty will be given to late homework. 20 points will be taken off per day after deadline till zero point.

(f) Please treat the above requirements as a kind of training in writing a decent homework report. If you have any problem regarding this homework, please feel free to consult with TA or me. If you think the time is too short to accomplish this homework, please let me know in class.

1. (Software design) (25%)

In some applications, BCD (binary coded decimal) is used instead of binary. Each BCD will use 4 bits to represent. A two digits BCD is usually placed in a single byte. For example, a two digits decimal number 59 would be represented by the byte “0101 1001”, where the upper nibble (b_7~b_4) 0101 stands for “5” and lower nibble (b_3~b_0) 1001 stands for “9”.

Please write an 8051 assembly program that will add two 6-digit BCD numbers stored in internal data memory location say 30h-32h and 33h-35h, respectively and produce a 7 digits BCD result in 36h-39h. You should show at least two results of the BCD addition operation in MCU8051IDE (by print screen).

Answer:

```
MOV     A,      35H
ADD     A,      32H     ; add up the last two digits
DA      A               ; adjust to decimal
MOV     39H,    A       ; last two digits completed

MOV     A,      34H     ; same here for the middle digits
ADDC    A,      31H     ; consider the carry from previous
DA      A
MOV     38H,    A
```

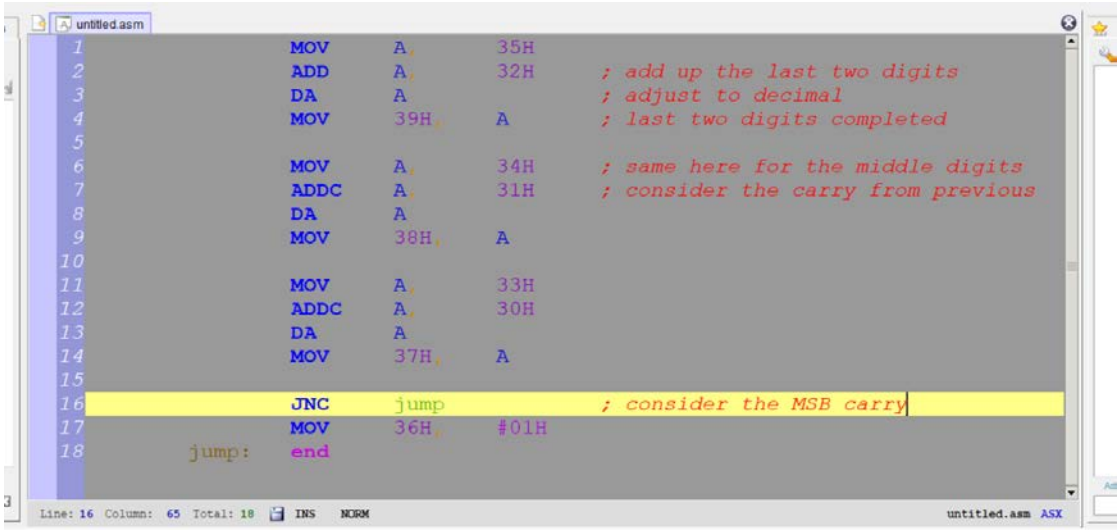
```
                MOV     A,      33H
                ADDC    A,      30H
                DA      A
                MOV     37H,    A

                JNC     jump            ; Consider the MSB carry
                MOV     36H     #01H
jump:           end

; The program terminates if no carry
```

For every two digits, we first move the numbers into the accumulators for adding, also we will check the carry flag while adding. Then, adjust the results into decimal for by using the DA instruction. Finally, consider the carry bit resulted by the MSB addition.

Schematic:



2. (Software design, simulation and instruction cycle counting) (25%)
8051 is commonly used as the core of a programmable logic controller (PLC) to perform logical manipulations on various incoming signals and issue proper controls accordingly, because it has Boolean instructions, internal bit-addressable bits for storing Boolean variables, and 4 8-bit I/O ports. The basic idea to implement a logic operation with 8051 is to enclose the logic operations in an infinite loop so that it can react to change of input persistently like hardware does. Assume that we want to implement the following logic operations:

P1.0 =(P1.1+P1.2 • !P1.3) ⊕ (P1.4 • P1.5+!P1.3) if P1.6 = 1 and
(!P1.1+P1.2 • P1.3) ⊕ (!P1.4+!P1.5 • P1.3) if P1.6 = 0.

(Note: ! means NOT, ⊕ means exclusive or)

(a) First construct the truth-table. Apply K-map to simplify the logic if you like.

Truth table:

I	P1.1	P1.2	P1.3	P1.4	P1.5	P1.6	P1.0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1
2	0	0	0	0	1	0	0

3	0	0	0	0	1	1	1
4	0	0	0	1	0	0	1
5	0	0	0	1	0	1	1
6	0	0	0	1	1	0	1
7	0	0	0	1	1	1	1
8	0	0	1	0	0	0	0
9	0	0	1	0	0	1	0
10	0	0	1	0	1	0	0
11	0	0	1	0	1	1	0
12	0	0	1	1	0	0	0
13	0	0	1	1	0	1	0
14	0	0	1	1	1	0	1
15	0	0	1	1	1	1	1
16	0	1	0	0	0	0	0
17	0	1	0	0	0	1	0
18	0	1	0	0	1	0	0
19	0	1	0	0	1	1	0
20	0	1	0	1	0	0	1
21	0	1	0	1	0	1	0
22	0	1	0	1	1	0	1
23	0	1	0	1	1	1	0
24	0	1	1	0	0	0	0
25	0	1	1	0	0	1	0
26	0	1	1	0	1	0	0
27	0	1	1	0	1	1	0
28	0	1	1	1	0	0	0
29	0	1	1	1	0	1	0
30	0	1	1	1	1	0	1
31	0	1	1	1	1	1	1
32	1	0	0	0	0	0	1
33	1	0	0	0	0	1	0
34	1	0	0	0	1	0	1
35	1	0	0	0	1	1	0
36	1	0	0	1	0	0	0
37	1	0	0	1	0	1	0
38	1	0	0	1	1	0	0
39	1	0	0	1	1	1	0
40	1	0	1	0	0	0	1
41	1	0	1	0	0	1	1
42	1	0	1	0	1	0	1
43	1	0	1	0	1	1	1
44	1	0	1	1	0	0	1
45	1	0	1	1	0	1	1
46	1	0	1	1	1	0	0
47	1	0	1	1	1	1	0
48	1	1	0	0	0	0	1
49	1	1	0	0	0	1	0
50	1	1	0	0	1	0	1

51	1	1	0	0	1	1	0
52	1	1	0	1	0	0	0
53	1	1	0	1	0	1	0
54	1	1	0	1	1	0	0
55	1	1	0	1	1	1	0
56	1	1	1	0	0	0	0
57	1	1	1	0	0	1	1
58	1	1	1	0	1	0	0
59	1	1	1	0	1	1	1
60	1	1	1	1	0	0	0
61	1	1	1	1	0	1	1
62	1	1	1	1	1	0	1
63	1	1	1	1	1	1	0

After simplification:

P1.0 = (~P1.1&&~P1.2&&~P1.3&&P1.6) || (~P1.1&&~P1.3&&P1.4&&~P1.6) ||
(~P1.1&&P1.3&&P1.4&&P1.5) || (P1.1&&~P1.2&&~P1.4&&~P1.6) ||
(P1.1&&~P1.3&&~P1.4&&~P1.6) || (P1.1&&~P1.2&&P1.3&&~P1.5) ||
(P1.1&&P1.3&&~P1.4&&P1.6) || (P1.1&&P1.3&&~P1.5&&P1.6) ||
(P1.2&&P1.3&&P1.4&&P1.5&&~P1.6)

(b) Then draw a flow chart to describe your code for the logic operation.

Thoughts:

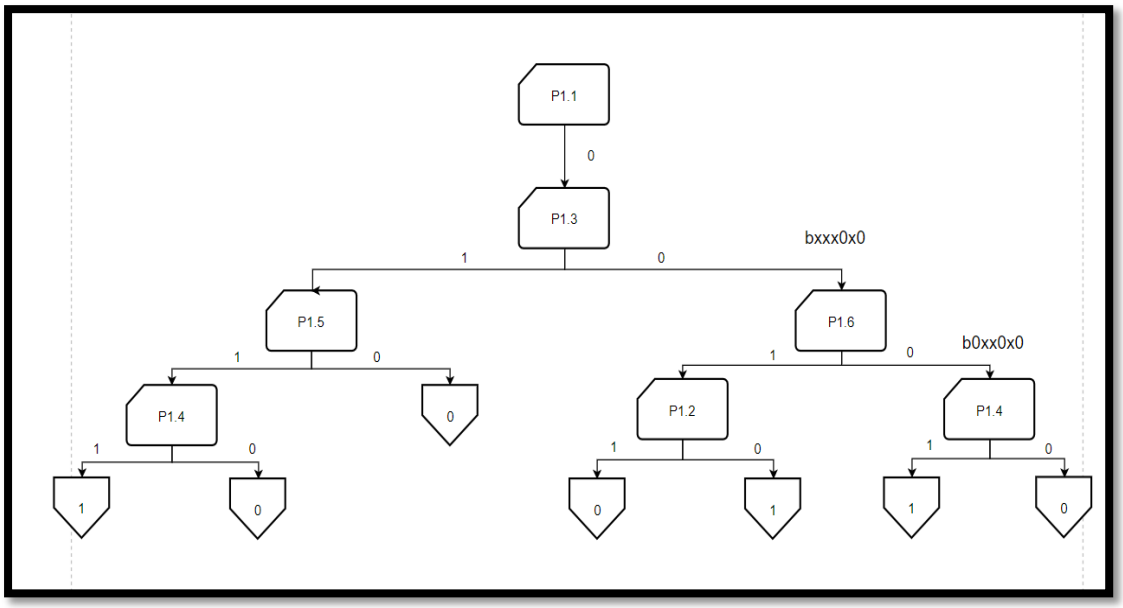
After we simplify the logic into SOP, the function consists of 9 terms. Based on the characteristics of AND and OR gate, since if any of the product term is true, then the whole logic will become true. Also, if any input in any product term is 0, then the product term will become false. In other words, our goal is to look at every input, and discover which product terms will be effected by the input. For a clearer explanation, I will give each product term a tag.

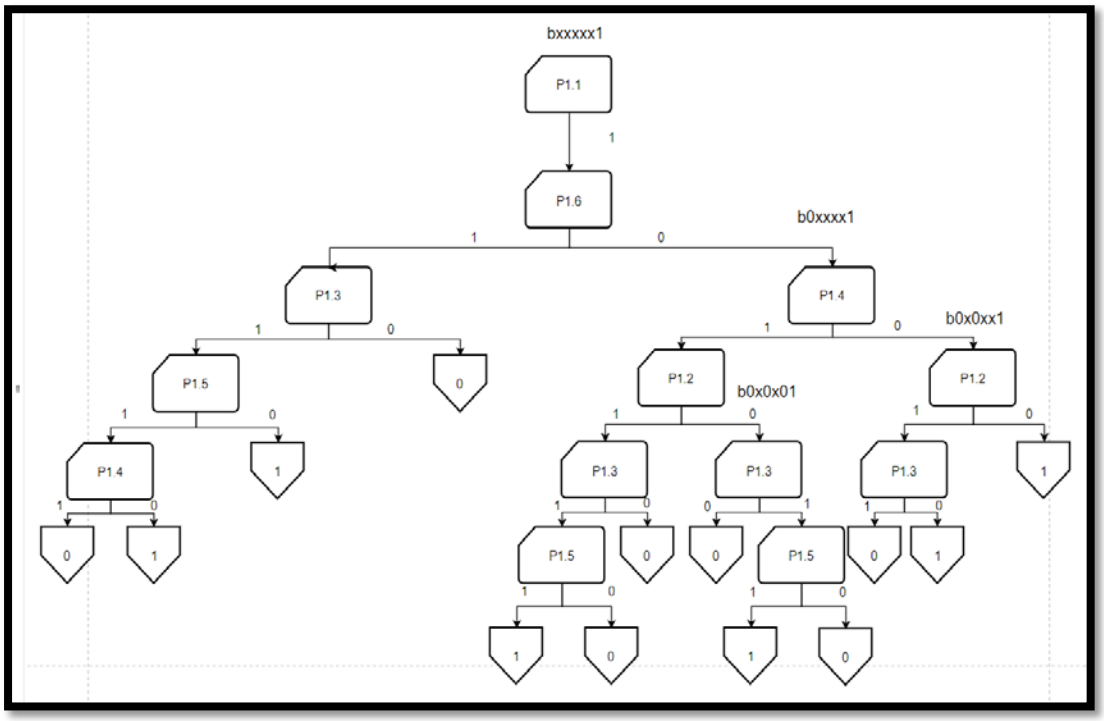
P1.0 = A(~P1.1&&~P1.2&&~P1.3&&P1.6) || B(~P1.1&&~P1.3&&P1.4&&~P1.6) ||
C(~P1.1&&P1.3&&P1.4&&P1.5) || D(P1.1&&~P1.2&&~P1.4&&~P1.6) ||
E(P1.1&&~P1.3&&~P1.4&&~P1.6) || F(P1.1&&~P1.2&&P1.3&&~P1.5) ||
G(P1.1&&P1.3&&~P1.4&&P1.6) || H(P1.1&&P1.3&&~P1.5&&P1.6) ||
I(P1.2&&P1.3&&P1.4&&P1.5&&~P1.6)

If P1.1 == 0 DEFGH term will become 0, hence we only have to consider ABCI, and so on……
If P1.1 == 1 ABC term will vanish
If P1.2 == 0 I term will vanish
If P1.2 == 1 ADF term will vanish
If P1.3 == 0 CFGHI term will vanish
If P1.3 == 1 ABE term will vanish
If P1.4 == 0 BCI term will vanish
If P1.4 == 1 DEG term will vanish
If P1.5 == 0 CI term will vanish
If P1.5 == 1 FH term will vanish
If P1.6 == 0 AGH term will vanish
If P1.6 == 1 BDEI term will vanish

After you check for 1 input, specific terms will vanish, and as you keep on checking, you will only have to check for only 1 product terms logic. That is the thinking flow of this program.

Design flow chart





Write an 8051 assembly program to implement the above logic operations. You should **verify your program in MCU8051IDE** by trying several input combinations according to the truth-table.

Answer:

The program simply keeps on comparing each input, deleting vanishing terms, only compare the terms that will affect the result.

Labels are marked in the flow chart. You can see which logic will be operated by looking at the flow chart.

```
start:      JB    P1.1,  bxxxxx1      ; start from looking at P1.1
            JNB   P1.3,  bxxx0x0
            JNB   P1.5,  p0equa0
            JNB   P1.4,  p0equa0
            SJMP  p0equa1

bxxxxx1:    JNB   P1.6,  b0xxxx1
            JNB   P1.3,  p0equa0
            JNB   P1.5,  p0equa1
            JNB   P1.4,  p0equa1
            SJMP  p0equa0

bxxx0x0:    JNB   P1.6,  b0xx0x0
            JNB   P1.2,  p0equa1
            SJMP  p0equa0

b0xx0x0:    JNB   P1.4,  p0equa0
            SJMP  p0equa1

b0xxxx1:    JNB   P1.4,  b0x0xx1
            JNB   P1.2,  b0x0x01
            JNB   P1.3,  p0equa0
            JNB   P1.5,  p0equa0
            SJMP  p0equa1

b0x0xx1:    JNB   P1.2,  p0equa1
            JNB   p1.3,  p0equa1
            SJMP  p0equa0

b0x0x01:    JNB   P1.3,  p0equa0
            JNB   P1.5,  p0equa0
            SJMP  p0equa1

p0equa1:    SETB  P1.0          ; assigns 1 to P1.0 and return back to start
            SJMP  start

p0equa0:    CLR   P1.0          ; assigns 0 to P1.0 and return back to start
            SJMP  start
            end
```

```
1 start:        JB      P1.1,   bxxxxx1
2               JNB     P1.3,   bxxx0x0
3               JNB     P1.5,   p0equa0
4               JNB     P1.4,   p0equa0
5               SJMP    p0equa1
6 bxxxxx1:      JNB     P1.6,   b0xxxx1
7               JNB     P1.3,   p0equa0
8               JNB     P1.5,   p0equa1
9               JNB     P1.4,   p0equa1
10              SJMP    p0equa1
11 bxxx0x0:     JNB     P1.6,   b0xx0x0
12              JNB     P1.2,   p0equa1
13              SJMP    p0equa0
14 b0xx0x0:     JNB     P1.4,   p0equa0
15              SJMP    p0equa1
16 b0xxxx1:     JNB     P1.4,   b0x0xx1
..              JNB     P1.2,   b0x0x01
17              JNB     P1.3,   p0equa0
```

(c) Estimate the best case and worst case propagation delay of your program from an input transition to output transition under 12 MHz operation:

The best case: (P1.5 == 0, P1.3 == 1, P1.1 == 0), (P1.6 == 1, P1.3 == 0, P1.1 == 1)

➔ 9μs

The worst case: (P1.6 == 0, P1.4 == 0, P1.3 == 1, P1.2 == 0, P1.1 == 1),
(P1.6 == 0, P1.4 == 0, P1.3 == 1, P1.2 == 1, P1.1 == 1)

➔ 15μs

3. (Data sheet study and software design/simulation in MCU8051IDE) (25%)

A 4-bit DIP switch and a **common-anode** 7-segment LED are connected to an 8051 as shown below. The 4-bit DIP switch is connected to P3.4-P3.7. When the DIP is in up position, the corresponding port pin would read in a "1", and a "0" for down position. To turn on an LED segment, the corresponding control signal input of the 7-segment, i.e., a~g and dp, should be logic zero. Which means the corresponding port pin should be at logic "1" since a 74LS240 is used in the middle.

(a) What is common-anode 7-segment LED? Why do we need the 220Ω resistor and the 74LS240? What's the function and purpose of 74LS240 in this design? You should search the web to find out relevant data sheets of 74LS240 and common-anode 7-segment LED, read them, and then try to answer the question.

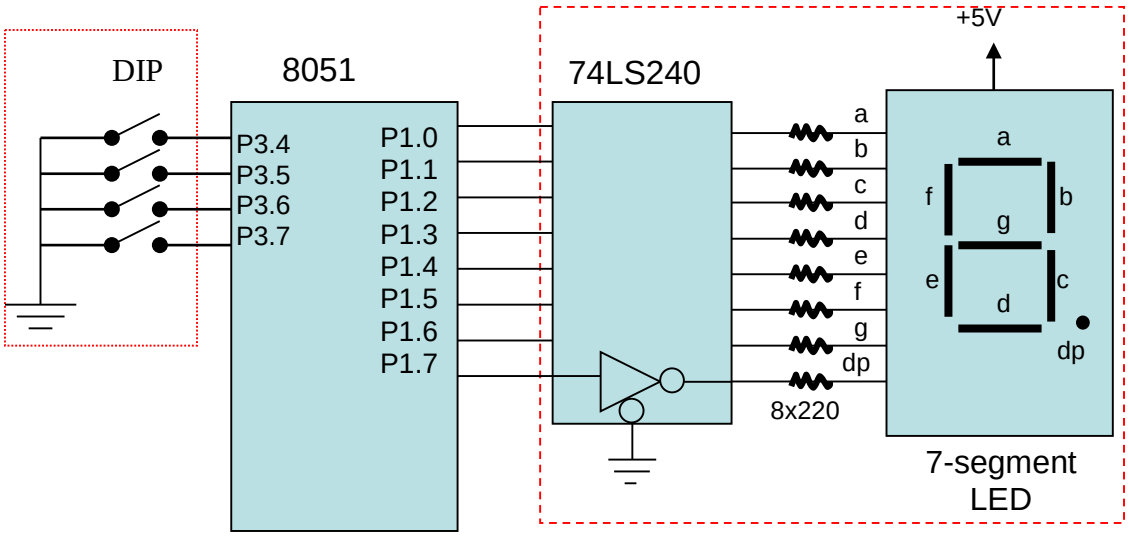
Answer：

A **common-anode** is a way to implement the seven segment, a common anode has all the anodes of 7-segments connected together. When working with a CA seven segment display, power must be applied externally to the anode of the anode connection that is common to all the segments. Then applying a low signal (Logic 0) to a particular segment, the specific segment will light up. The additional **220Ω** resistor should be added to limit the amount of current flowing through each LED segment.

The **74LS240** is an **octal-three state buffer**. The word "three state" indicates that the buffer allows an output port to assume a high impedance state, effectively removing the output from the circuit, in addition to the 1 and 0 logic levels. It allows multiple circuits to share the same output line. The word "octal" indicates that there is a total of eight inputs at a time, by using the buffer can stabilize the input signal and prevent glitches.

Also, in this example we are using a CA seven segment display, hence we will be using an inverted three state output version of the buffer.

(b) Design an 8051 program which can continuously read the DIP position and display the 7-segment LED correspondingly. Test your program **in MCU8051IDE** and use the **simple keypad** and **LED display** virtual HW to demonstrate your program (In this case you don't need the 74240 buffer). You should capture at least two examples of the MCU8051IDE simulation results by print screen and put them in your written report.



Design thoughts:

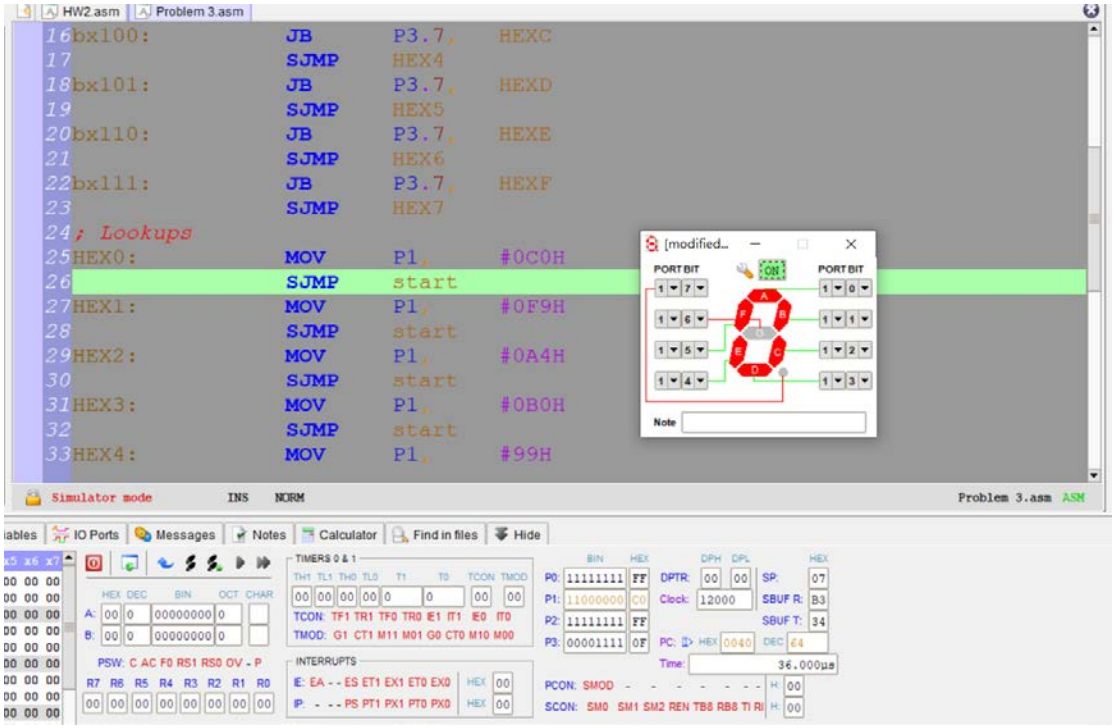
The first half of the program, we have to determine what is the DIP input. According to the input we have to jump the particular display number.

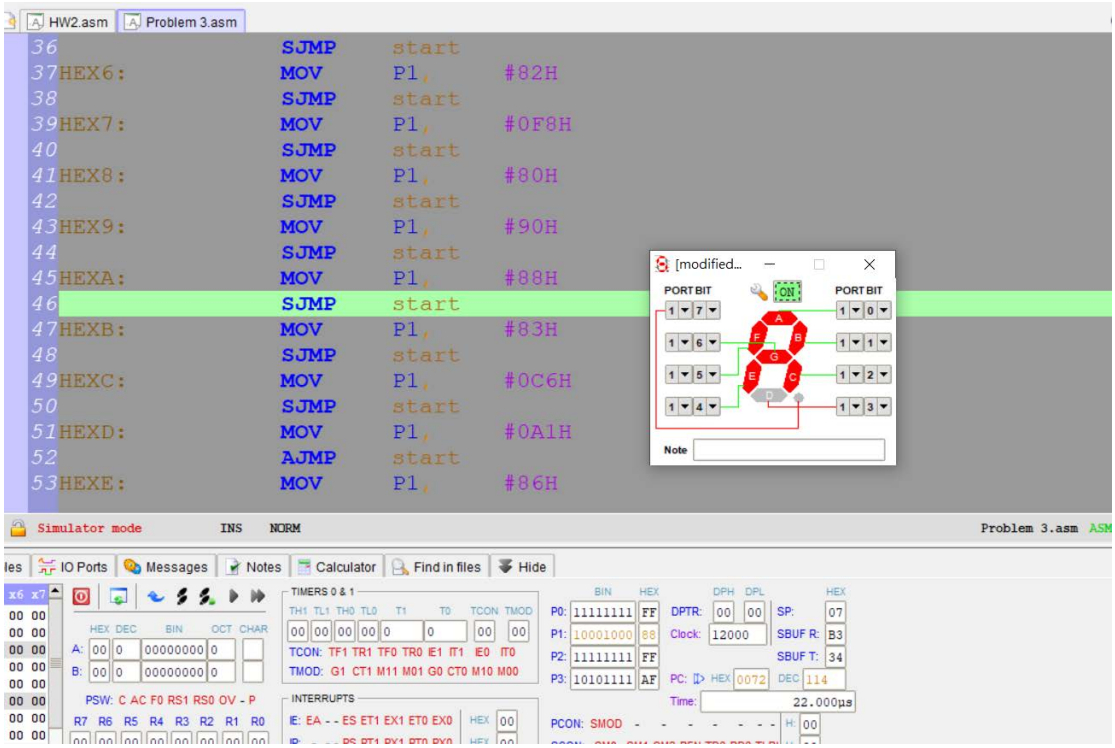
Code:

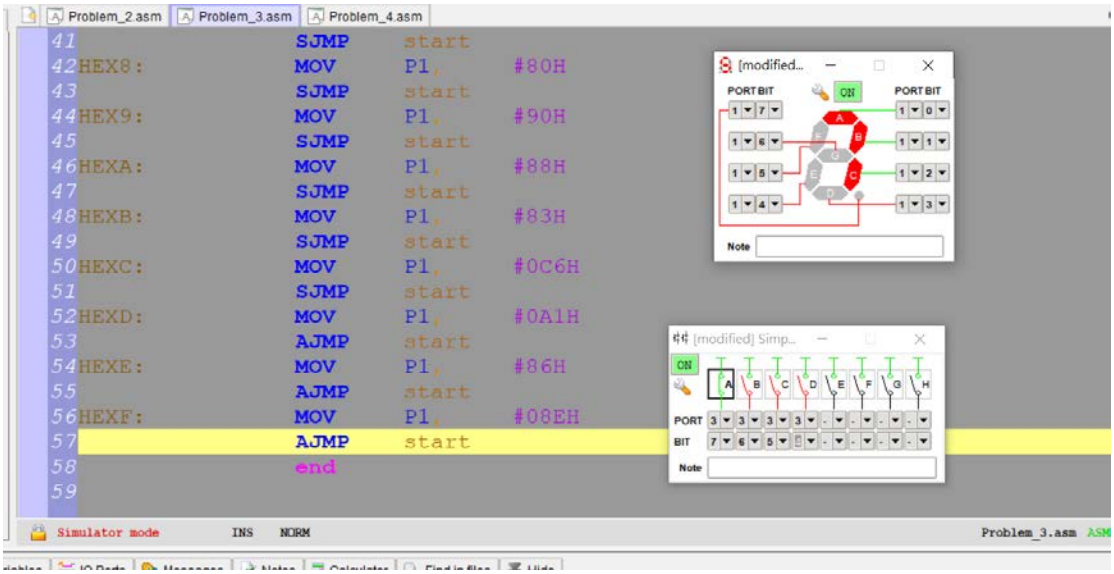
```
; First get the DIP input
start:      JB      P3.4,   bxxx1
            JB      P3.5,   bxx10
            JB      P3.6,   bx100
            JB      P3.7,   HEX8        ; this indicates the input is 8, and so on…
            SJMP    HEX0
bxxx1:      JB      P3.5,   bxx11
            JB      P3.6,   bx101
            JB      P3.7,   HEX9
            SJMP    HEX1
bxx10:      JB      P3.6,   bx110
            JB      P3.7,   HEXA
            SJMP    HEX2
bxx11:      JB      P3.6,   bx111
            JB      P3.7,   HEXB
            SJMP    HEX3
bx100:      JB      P3.7,   HEXC
            SJMP    HEX4
bx101:      JB      P3.7,   HEXD
            SJMP    HEX5
bx110:      JB      P3.7,   HEXE
            SJMP    HEX6
bx111:      JB      P3.7,   HEXF
```

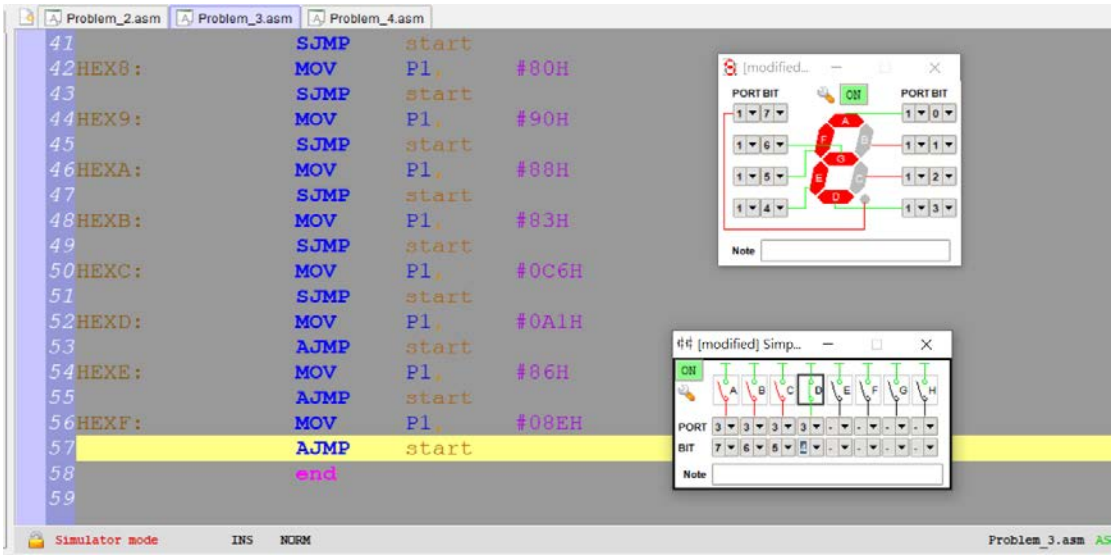
```
                SJMP  HEX7
; Lookups
; These are the output to the seven segments
HEX0:           MOV   P1,     #0C0H
                SJMP  start
HEX1:           MOV   P1,     #0F9H
                SJMP  start
HEX2:           MOV   P1,     #0A4H
                SJMP  start
HEX3:           MOV   P1,     #0B0H
                SJMP  start
HEX4:           MOV   P1,     #99H
                SJMP  start
HEX5:           MOV   P1,     #92H
                SJMP  start
HEX6:           MOV   P1,     #82H
                SJMP  start
HEX7:           MOV   P1,     #0F8H
                SJMP  start
HEX8:           MOV   P1,     #80H
                SJMP  start
HEX9:           MOV   P1,     #90H
                SJMP  start
HEXA:           MOV   P1,     #88H
                SJMP  start
HEXB:           MOV   P1,     #83H
                SJMP  start
HEXC:           MOV   P1,     #0C6H
                SJMP  start
HEXD:           MOV   P1,     #0A1H
                AJMP  start
HEXE:           MOV   P1,     #86H
                AJMP  start
HEXF:           MOV   P1,     #08EH
                AJMP  start
                end
```

Screenshots:







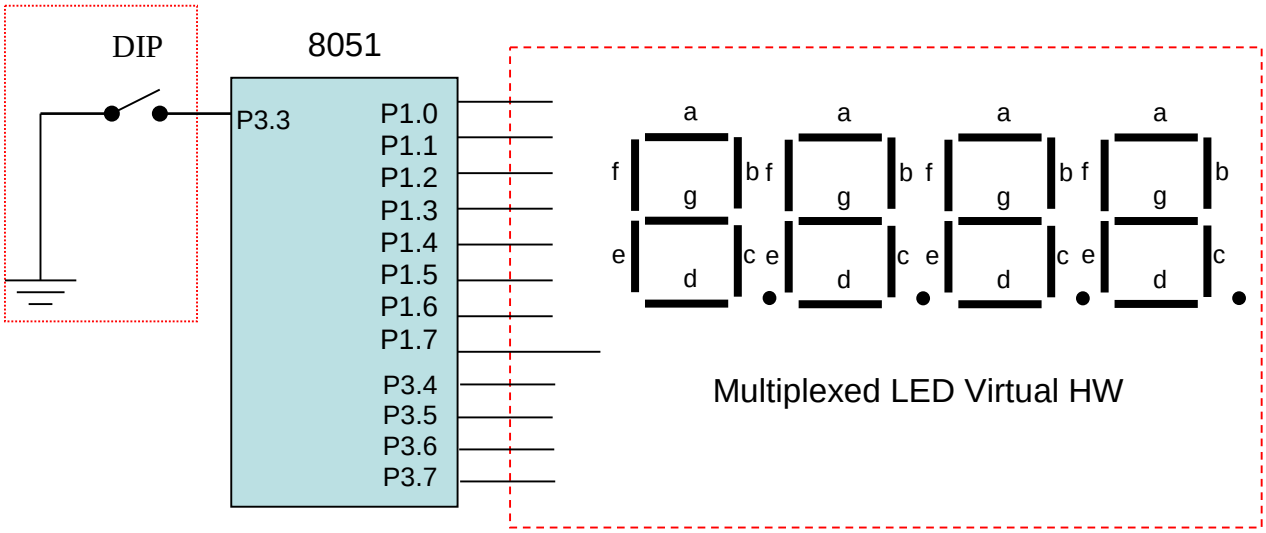


4. (Multiplexed LED virtual hardware simulation in MCU8051IDE) (25%)
Design a count-down program that can count down from 9999 to 0000 using the multiplexed LED virtual hardware to display the 4 digits. When DIP switch toggles (H→L→H), start the count-down from 9999. Use a software delay loop to control the count-down speed. Display each digit change on the 4-seven-segment LED virtual hardware using P1 and P3.4~P3.7 for its control. Stop the count-down when it reached at 0000.
You should first draw a flowchart (or pseudo code) for your program. Then demo your program under MCU8051IDE by print screen and put them in your written report.

Thought process:

Since 9999 is quite large, it is more convenient to store the four digits into 4 registers. We use four registers to count and store. Since only one LED will be

lightened up one at a time, we will store the digit that should be lightened up into accumulator A, and use it for our output. Using data pointer and lookup table for the display.



Pseudo Code:

Toggle: jump if p3.3 not bit1 and jump if p3.3 bit 1.
Initial the four registers storing the initial value #09H
Initial Data pointer to lookup table:
// starting to display
This loop runs four times for every digit{
    Move the current working register to A, for display
    Let A store the display value by using the lookup table
    Move A, which is currently storing the display data, to P1 for display
    Activate the select for the first bit which is at P3.4, P3.5, P3.6, P3.7
    Call delay for several micro second for display
    Deactivate the select
}
Jump to the counting part
// Counting part
Check if each register is storing zero.
IF [last digit != 0]
    Last digit -= 1
    Jump back to the loop
ELSE IF [third digit != 0]
    Third digit -= 1
    Last digit = 9
    Jump back to the loop
ELSE IF [second digit != 0]
    Second digit -= 1
    Third digit = 9

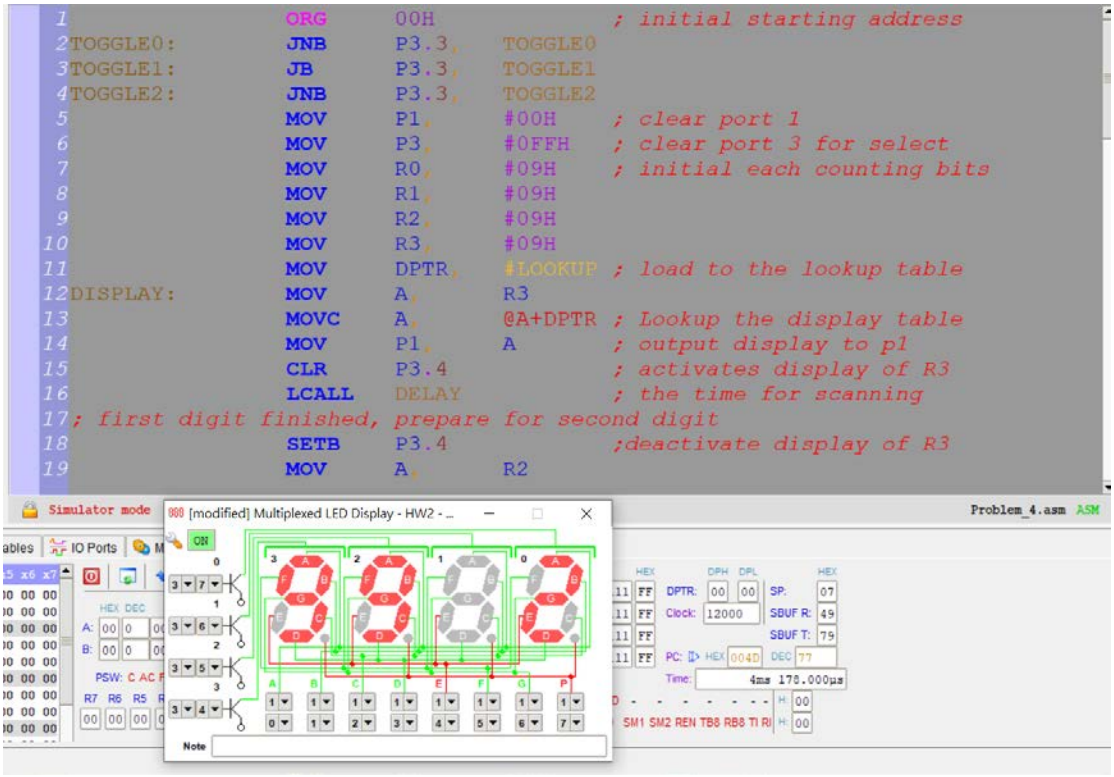
```
        Second digit = 9
        Last digit = 9
ELSE IF [first digit != 0]
        First digit -= 1
        Second digit = 9
        Third digit = 9
        Last digit = 9
ELSE
        end
```

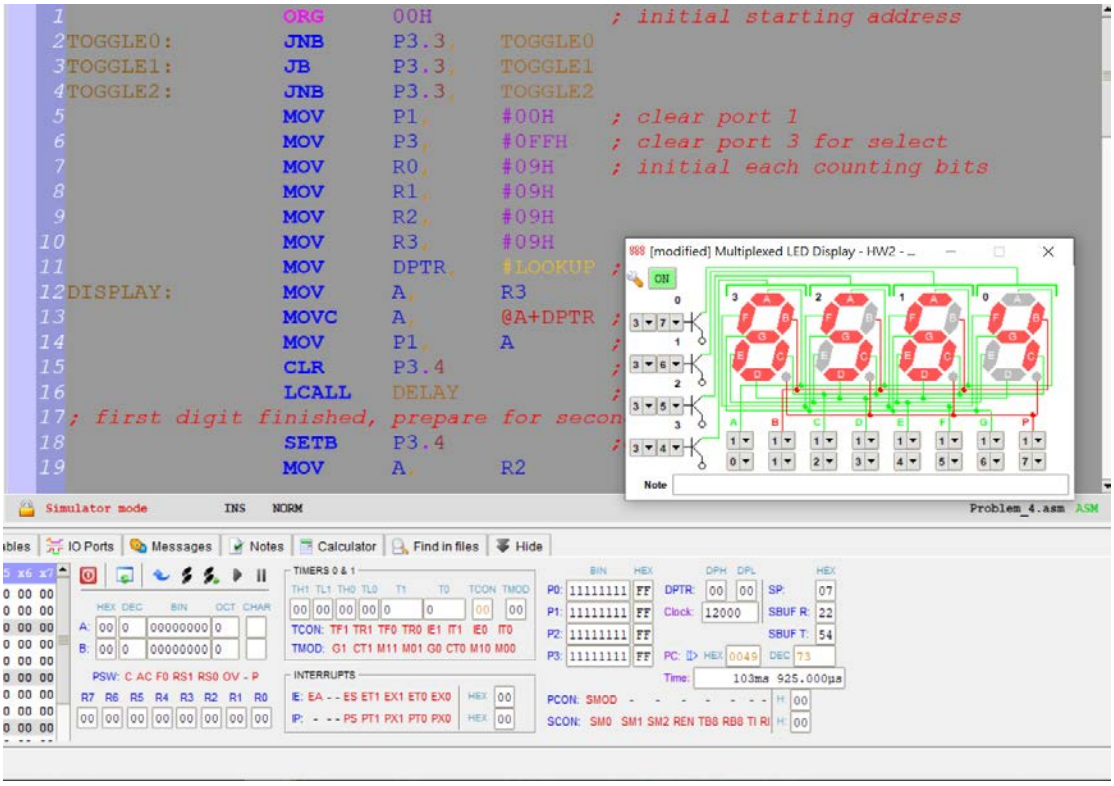
```
            ORG  00H                         ; initial starting address
TOGGLE0:    JNB  P3.3,  TOGGLE0              ; Toggle button required
TOGGLE1:    JB   P3.3,  TOGGLE1
TOGGLE2:    JNB  P3.3,  TOGGLE2
            MOV  P1,    #00H                 ; clear port 1 for display
            MOV  P3,    #0FFH                ; clear port 3 for select
            MOV  R0,    #09H          ; initial each counting bits 9999
            MOV  R1,    #09H
            MOV  R2,    #09H
            MOV  R3,    #09H
            MOV  DPTR, #LOOKUP           ; load to the lookup table
DISPLAY:    MOV  A,     R3
            MOVC        A,     @A+DPTR   ; Lookup the display table
            MOV  P1,    A                ; output display to p1
            CLR  P3.4                    ; activates display of R3
            LCALL       DELAY            ; the time for scanning
; first digit finished, prepare for second digit
            SETB P3.4                    ;deactivate display of R3
            MOV  A,     R2
            MOVC        A,     @A+DPTR   ; Lookup the display table
            MOV  P1,    A                ; output display to p1
            CLR  P3.5           ;         activates display of R2
            LCALL       DELAY            ; the time for scanning
; second digit finished, prepare for third digit
            SETB P3.5
            MOV  A,     R1
            MOVC        A,     @A+DPTR
            MOV  P1,    A
            CLR  P3.6
            LCALL       DELAY
; third digit finished, prepare for the last digit
            SETB P3.6
            MOV  A,     R0
            MOVC        A,     @A+DPTR
```

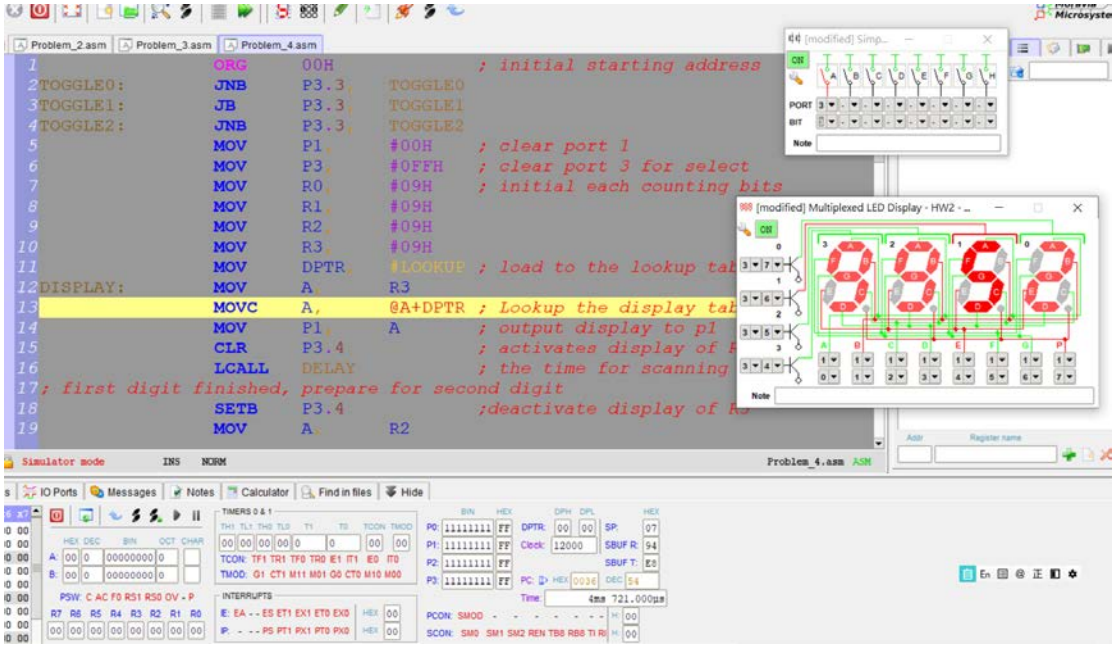
```
                MOV P1,     A
                CLR  P3.7
                LCALL       DELAY
; all finished prepare to return
                MOV P3,     #0FFH           ; deactivate all select
                SJMP COUNTING               ; jump to counting part
; delay for 8 us for scanning
DELAY:          MOV R4,     #02H            ; 2
DEL1:           MOV R5,     #02H            ; 2
DEL2:           DJNZ R5,    DEL2            ; 2
                DJNZ R4,    DEL1            ; 2 * 2 * 2 = 8
                RET
; lookup table
LOOKUP:
                DB    0C0H          ; 0
                DB    0F9H          ; 1
                DB    0A4H          ; 2
                DB    0B0H          ; 3
                DB    099H          ; 4
                DB    092H          ; 5
                DB    082H          ; 6
                DB    0F8H          ; 7
                DB    080H          ; 8
                DB    090H          ; 9
; count check if the lower bit is zero
COUNTING:CJNE R0,   #00H,       DEC1        ; decrease 1
                CJNE  R1,   #00H,       DEC10       ; decrease 10
                CJNE  R2,   #00H,       DEC100
                CJNE  R3,   #00H,       DEC1000
                SJMP end
DEC1:           DEC  R0
                AJMP DISPLAY
DEC10:          DEC  R1
                MOV R0,     #09H
                AJMP DISPLAY
DEC100:         DEC  R2
                MOV R1,     #09H
                MOV R0,     #09H
                AJMP DISPLAY
DEC1000:        DEC  R3
                MOV R2,     #09H
                MOV R1,     #09H
                MOV R0,     #09H
                AJMP DISPLAY
```

end: end

Screenshot:






Problem_2.asm
Problem_3.asm
Problem_4.asm
1 ORG 00H ; initial starting address
2TOGGLE0: JNB P3.3, TOGGLE0
3TOGGLE1: JB P3.3, TOGGLE1
4TOGGLE2: JNB P3.3, TOGGLE2
5 MOV P1, #00H ; clear port 1
6 MOV P3, #0FFH ; clear port 3 for select
7 MOV R0, #09H ; initial each counting bits
8 MOV R1, #09H
9 MOV R2, #09H
10 MOV R3, #09H
11 MOV DPTR, #LOOKUP ; load to the lookup tal
12DISPLAY: MOV A, R3
13 MOVC A, @A+DPTR ; Lookup the display tab
14 MOV P1, A ; output display to p1
15 CLR P3.4 ; activates display of F
16 LCALL DELAY ; the time for scanning
17; first digit finished, prepare for second digit
18 SETB P3.4 ;deactivate display of F
19 MOV A, R2
[modified] Simp...
[modified] Multiplexed LED Display - HW2 - ...
Simulator mode
INS
NORM
Problem_4.asm ASM
IO Ports
Messages
Notes
Calculator
Find in files
Hide