EE2401 微算機系統Fall 2019

HW#4 (8051 applications on MCU8051IDE) (10/14/2019)

Due date: 11/14/2019. Severe penalty will be given to late homework.

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-hw4-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. (30%)

MCU8051IDE 中有提供一個 4 x 4 keypad 以及 1 個數字顯示的 7 段顯示器的 Virtual Hardware，請設計一個程式來讀入按鍵及顯示，一開始沒有任何按鍵時先不顯示 7 段顯示器，當開始有按鍵時，把該按鍵對應的文數字顯示在 7 段顯示器上，之後的按鍵就依此類推。

你最少必須提供你的系統架構(如下圖，但要標示 4x4keypad 及你所使用的 port pin)，流程圖(或 pseudo code)，解釋你的做法，你的程式碼，Virtual Hardware 儲存檔。可以讓助教驗證你的程式。

There are a total of 16 different buttons in virtual hardware, including special characters such as “*” and “#”, I will use E and F to replace them, repectively.

Pseudo code:

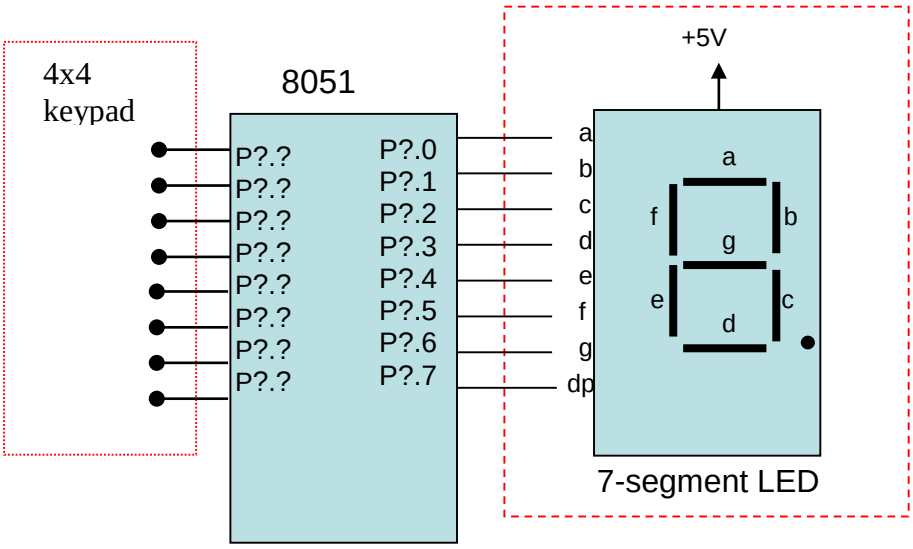
```
MAIN() {
    IN_HEX()              // get hex code
    Call OUTCHR()         // echo to VDT console
}


  IN_HEX() {
  BACK:
      [Debounce_count = 50]            // let R3 store debounce count
      FOR[R3 = 50, R3 > 0, R3++] {     // check if signal up for 50 sec, debounce
          [GET_KEY()]                  // Use flag C to check if key pressed
          IF[C == 0]                   // if key not pressed
              [Goto TOP]               // check again
      }
      [Push ACC]                       // save hex code to stack
```

```
BACK2:
        FOR[R3 = 50, R3 > 0, R3++] {     // check if signal up for 50 sec, debounce
            [GET_KEY()]                  // Use flag C to check if key pressed
            IF[C == 0]                   // if key not pressed
                [Goto BACK2]                 // check again
        }
        [POP ACC]                    // remove from stack when released
        [RETURN]
}

 GET_KEY() {
        [A = #0FEH]              // start from column 0
        FOR (R6 = 4; R6 > 0; R6--) {
            [P0 = A]
            [R7 = A]
            [A = P0]
            [A & #0F0H]              // isolate row lines (remain row data)
            IF [A != #0F0H] {        // if key hit
                [KEY_HIT()]
            ELSE
                [R7 = A]             // move to next
                [RL A]               // next line
        }
        [CLR C]
        RET
 }

 KEY_HIT() {
        [Convert A to Hex code]
        [C = 1]
        [R6 = A]                 // save hex code to R6
 }
 OUTCHR() This function simply turn the hex code to the configuration of the 8051 IDE
   ssd and set P1 to the output segments (As in homework 2)
;*****************************************************************
; KEYPAD INTERFACE EXAMPLE
;
; This program reads hexadecimal characters from a
; keypad attached to Port 1 and echoes keys pressed
; to the console.
;*****************************************************************

                HTOA        EQU     003CH       ; MON51 subroutines
                OUTCHR      EQU     01DEH       ;

                ORG     0000H       ; put main in reset ISR
MAIN:           CALL    IN_HEX      ; get hex code from key pad
                CALL    OUTCHR      ; echo to VDT console
                SJMP    MAIN        ; repeat
```

```
;*************************************************************************
; IN_HEX - input hex code from keypad with debouncing
; for key press and key release (50 repeat
; operations for each)
;*************************************************************************
IN_HEX:     MOV     R3,  #50  ; debounce count
BACK:       CALL    GET_KEY         ; key pressed? C = 1 yes, C = 0 No
            JNC     IN_HEX      ; no, check again
            DJNZ    R3,  BACK   ; yes, repeat 50 times for debouncing
            PUSH    ACC         ; save hex code

BACK2:      MOV     R3,  #50  ; wait for key up
BACK3:      CALL    GET_KEY         ; key still pressed?
            JC      BACK2       ; yes: keep checking
            DJNZ    R3,  BACK3  ; no, key released, repeat 50 times debouncing
            POP ACC             ; recover hex code and
            RET             ; return


;*********************************************************************
; GET_KEY - get keypad status
; - return with C = 0 if no key pressed
; - return with C = 1 and hex code in ACC if
; a key is pressed
;*********************************************************************
GET_KEY:    MOV     A,   #0FEH    ; start with column 0 "1111 1110"
            MOV     R6,  #4   ; use R6 as column counter

TEST:       MOV     P0,  A        ; activate column line
            MOV     R7,  A        ; save column info in ACC
            MOV     A,   P0       ; read back Port 0
            ANL     A,   #0F0H    ; isolate row lines (column turn into 0)
            CJNE    A, #0F0H, KEY_HIT     ; row line active?
            MOV     A,   R7       ; no: move to next
            RL      A
            DJNZ    R6,  TEST     ; keep checking until all columns check

            CLR     C             ; no key pressed
            SJMP    EXIT          ; return with C = 0
; if found a key down
KEY_HIT:    MOV     R7,  A    ; save scan code in R7
            MOV     A,   #4   ; prepare to convert to hex code
            CLRC          ; column weighting
            SUBB    A,   R6   ; 4 - R6 = column number 0-3
            MOV     R6,  A    ; save in R6
            MOV     A,   R7   ; restore scan code
            SWAP    A         ; put in low nibble
            MOV     R5,  #4   ; use R5 as counter
AGAIN:      RRC     A         ; rotate for row num until 0
            JNC     DONE          ; done when C = 0
```

```
                INC     R6          ; add 4 to keycode to goto next
                INC     R6          ; row until active low found
                INC     R6
                INC     R6
                DJNZ    R5,  AGAIN
DONE:           SETB    C           ; C = 1 (key passed)
                MOV     A,   R6     ; hex code in A
EXIT:           RET
;*********************************************************************
; SSD
; This function will convert the lcd pad signal to seven segment display
;*********************************************************************
OUTCHR:
; First get the DIP input
start:          JB      A.0,        bxxx1
                JB      A.1,        bxx10
                JB      A.2,        bx100
                JB      A.3,        HEX7        ; this indicates the input is 8, and so on…
                SJMP    HEX1
bxxx1:          JB      A.1,        bxx11
                JB      A.2,        bx101
                JB      A.3,        HEX8
                SJMP    HEX2
bxx10:          JB      A.2,        bx110
                JB      A.3,        HEX9
                SJMP    HEX3
bxx11:          JB      A.2,        bx111
                JB      A.3,        HEXC
                SJMP    HEXA
bx100:          JB      A.3, HEXE
                SJMP    HEX4
bx101:          JB      A.3,        HEX0
                SJMP    HEX5
bx110:          JB      A.3,        HEXF
                SJMP    HEX6
bx111:          JB      A.3,        HEXD
                SJMP    HEXB
; Lookups
; These are the output to the seven segments
HEX0:           MOV     P1,  #0C0H
                RET
HEX1:           MOV     P1,  #0F9H
                RET
HEX2:           MOV     P1,  #0A4H
                RET
HEX3:           MOV     P1,  #0B0H
                RET
HEX4:           MOV     P1,  #99H
                RET
HEX5:           MOV     P1,  #92H
```

```
                RET
HEX6:           MOV     P1,  #82H
                RET
HEX7:           MOV     P1,  #0F8H
                RET
HEX8:           MOV     P1,  #80H
                RET
HEX9:           MOV     P1,  #90H
                RET
HEXA:           MOV     P1,  #88H
                RET
HEXB:           MOV     P1,  #83H
                RET
HEXC:           MOV     P1,  #0C6H
                RET
HEXD:           MOV     P1,  #0A1H
                RET
HEXE:           MOV     P1,  #86H
                RET
HEXF:           MOV     P1,  #08EH
                RET
 end
```



2. (30%)

MCU8051IDE 中有提供一個 4 個數字顯示的 7 段顯示器的 Virtual Hardware，假設我們所要顯示的數字是存放在內部資料記憶體 30H, 31H(2 個 bytes 共 4 個 hex 數字)，請在 MCU8051IDE 環境下設計一個程式來不斷的顯示這 4 個數字在 7 段顯示器的 Virtual Hardware 上。

你最少必須提供你的系統架構，流程圖(或 pseudo code)，解釋你的做法，你的程式碼，Virtual Hardware 儲存檔。可以讓助教驗證你的程式。

Note that the delay time can be configured in the DELAY() function. For testing and debugging conveniences, We only delay for 2us.

```
[Clear P1]                              // clear output ports
[Clear P3]                              // clear display selects
[DPTR = #LOOKUP]                        // initial data pointer to lookup table
MAIN() {
    [Move 31H to A]                     // display what is inside 31H
    [Strip higher nibble of A]
    [R0 = A]                            // R0 displays 31H lower nibble
    [Reassign A 31H to A and strip lower nibble of A]
    [Swap A]            // to make the high bits come down to low bit to assign
    [R1 = A]                            // R1 displays 30H higher nibble
    [Move 30H to A]                     // display what is inside 30H
    [Strip higher nibble of A]
    [R2 = A]                            // R0 displays 30H lower nibble
    [Reassign A 30H to A and strip lower nibble of A]
    [Swap A]            // to make the high bits come down to low bit to assign
    [R3 = A]
    DISPLAY()
}
DISPLAY() {
    [A = R3]            // current displaying digit
    [A = A+@DPTR]       // to the lookup for display
    [P1 = A]            // to the output display port
    [CLR P3.4]          // activate display, for scanning
    [DELAY()]           // delay for scanning
    [SETB P3.4]         // deactivate display
    Same for the other 3 bit…..
    …….
    // When all digit finish
    [P3 = #0FFH]        // deactivate all select
    RET
}
DELAY() {
    ……
}
```

```
            MOV     P1,  #00H     ; clear port 1 for display
            MOV     P3,  #0FFH    ; clear port 3 for select
MAIN:       MOV     A,   31H
            ANL     A,   #0FH     ; strip the high nibble
            MOV     R0,  A        ; put the 31H low nibble into R0
            MOV     A,   31H      ; this time for the higher nibble
            ANL     A,   #0F0H    ; strip the lower nibble
            SWAP    A
            MOV     R1,  A        ; put the 30H high nibble into R1
            MOV     A,   30H
            ANL     A,   #0FH     ; strip the high nibble
            MOV     R2,  A        ; put the 30H low nibble into R2
            MOV     A,   30H      ; this time for the higher nibble
            ANL     A,   #0F0H    ; strip the lower nibble
            SWAP    A
            MOV     R3,  A        ; put the 32H high nibble into R3
```

```
                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
                CLRP3.5     ; 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
                CLRP3.7
                LCALL   DELAY
; all finished prepare to return
                MOV     P3, #0FFH   ; deactivate all select
                AJMP    MAIN
; 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
                DB  088H
                DB  083H
```

```
DB  0C6H
DB  0A1H
DB  086H
DB  08EH
end
```

3. (40%)

MCU8051IDE 中有提供一個 4 個數字顯示的 7 段顯示器的 Virtual Hardware 可以用來當作時鐘的顯示，左邊兩位數用來顯示”時”，右邊兩位數用來顯示”分”，採用 24 小時制。假設小時的兩位數字(00-23)存在內部資料記憶體 30H 處，分鐘的兩位數(00-59)存在 31H 處，它們的值可以隨時透過 MCU8051IDE 更改 30H 和 31H 位置內容來達到，同時你的程式在執行時也會透過你所設計的計時功能每 60 秒更新時-分的顯示。請在 MCU8051IDE 下設計一個程式使用此虛擬硬體來實現這個簡易的時鐘，包括每 60 秒分鐘的值要加 1 更新，小時亦然。

你最少必須提供你的系統架構，流程圖(或 pseudo code)，解釋你的做法，你的程式碼，Virtual Hardware 儲存檔。可以讓助教驗證你的程式。

```
[Clear P1]                      // clear output ports
[Clear P3]                      // clear display selects
[DPTR = #LOOKUP]                // initial data pointer to lookup table
MAIN() {
    [Move 31H to A]             // display what is inside minute
    [Move #10H to B]            // for division
    [A / B]
    [R0 = B]                    // store low digit of minute
    [R1 = A]                    // store high digit of minute
    [Move 30H to A]             // display what is inside hour
    [Move #10H to B]            // for division
    [A / B]
    [R2 = B]                    // R2 displays hour lower nibble
    [R3 = A]                    // R3 displays hour higher nibble
    DISPLAY()
    COUNTING()
}
DISPLAY() {
    [A = R3]            // current displaying digit
    [A = A+@DPTR]       // to the lookup for display
    [P1 = A]            // to the output display port
    [CLR P3.4]          // activate display, for scanning
    [DELAY()]           // delay for scanning
    [SETB P3.4]         // deactivate display
    Same for the other 3 bit…..
    …….
    // When all digit finish
    [P3 = #0FFH]        // deactivate all select
    RET
```

```
}
DELAY() {
      ……
}
COUNTING() {
      [Move 32H to A]                  // for later counting
      If [A != #59H]  {
            [32H += 1]
            [RET]
}
      ELSE {
            IF[R0 != #09H] {
                  [CLR 32H]
                  [31H += 1]
                  [RET]
            }
            ELSE {
                  IF [R1 != #05H] {                // ADD10MIN
                        [CLR 32H]
                        [31H += #07H]
                        [RET]
                  }
                  ELSE {
                        IF [R2 != #03H] {
                              [CLR 32H]
                              [CLR 31H]
                              [30H += #07H]
                              [RET]
                        ELSE {
                              IF [R3 != #02H] {
                                    [CLR 32H]
                                    [CLR 31H]
                                    [30H += #07H]
                                    [RET]
                              }
                              ELSE
                                    [Clear all digits]
                        }
                  }
            }
      }
}
```

I will use 30H to store hour, 31H to store minute, 32H to store second
Same as previous, Delay time can be configured inside the DELAY() function.

```
                MOV     P1,  #00H           ; clear port 1 for display
                MOV     P3,  #0FFH          ; clear port 3 for select
                MOV     DPTR,   #LOOKUP         ; load to the lookup table

MAIN:           MOV     A,   31H
                MOV     B,   #10H       ; 10, to divide
                DIV     AB          ; for minute display
                MOV     R0,  B      ; put the 31H low nibble into R0
                MOV     R1,  A      ; put the 30H high nibble into R1
                MOV     A,   30H
                MOV     B,   #10H       ; 10, to divide
                DIV     AB
                MOV     R2,  B      ; put the 30H low nibble into R2
                MOV     R3,  A      ; put the 32H high nibble into R3

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,  #01H           ; 1
DEL1:           MOV     R5,  #01H           ; 1
DEL2:           DJNZ    R5,  DEL2           ; 2
                DJNZ    R4,  DEL1           ; 1 * 1 * 2 = 2
                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 satisfy carry conditions (60 second 60 minutes 24 hour)
COUNTING:   MOV     A,  32H
            CJNE    A, #59D, ADD1SEC
            CJNE    R0,#09H, ADD1MIN    ; add 1 minute
            CJNE    R1,#05H, ADD10MIN; add 10 minutes
            MOV     A,  30H
            CJNE    R2,#03H, ADD1HOU
            AJMP    CHECKRES
            CJNE    R2,#09H, ADD1HOU    ; add 1 hour

CHECKRES:   CJNE    R3,#02H, ADD10HOU; add 10 hours
            AJMP    RES

ADD1SEC:    INC     32H
            AJMP    MAIN
ADD1MIN:    MOV     32H,    #00H    ; clear second digits
            INC     31H     ; increase 1 minute
            AJMP    MAIN

ADD10MIN:   MOV     32H,    #00H    ; clear second digits
            MOV     A,  31H ; increase 10 minutes
            ADD     A,  #07H
            MOV     31H,    A
            AJMP    MAIN
ADD1HOU:    MOV     32H,#00H
            MOV     31H,    #00H
            INC 30H
            AJMP    MAIN

ADD10HOU:   MOV     32H,    #00H
            MOV     31H,    #00H
            MOV     A,  30H
            ADD     A,  #07H
            MOV     30H,    A
            AJMP    MAIN

RES:        MOV     32H,    #00H
            MOV     31H,    #00H
```

```
MOV     30H,    #00H
AJMP    MAIN
end
```