

Micros Peripherals and Interfaces - coursework - 2025 -2317250 *by Edward Bulpitt*

Submission date: 13-Jan-2025 03:58PM (UTC+0000)

Submission ID: 248288011

File name: 52085_Edward_Bulpitt_Micros_Peripherals_and_Interfaces_-_coursework_-_2025_-2317250_621295_11523273.docx (305.8K)

Word count: 3111

Character count: 15807

Micros Peripherals and Interfaces

Edward Bulpitt, UWTSD, 2317250

Contents

Section 1	3
Circuit Description and Component Functions.....	3
Section 2	4
Planning Pseudocode:	4
Section 3	1
1. Modify "Count.asm" so that the seven segment display counts from 0 to F in hexadecimal numbers in 0.2 second increments.....	1
2. "Write a program so that when you press one of the two buttons, it displays "L" on the 7 segment LED display, then when you press the other one, it displays "r".	2
3. "Write another program, so one button causes all of the LED segments to light, and the other switch turns them off. (the switch activations should be 'latching' i.e. you don't have to hold the button down for the on/off effect to remain)"	3
4. "Then combine all of these techniques together, by writing a new program where you start by displaying '5' but you can count up (to a maximum of 9) if you press one button, and count down (to a minimum of 0) if you press the other."	5
References:	7

SECTION 1

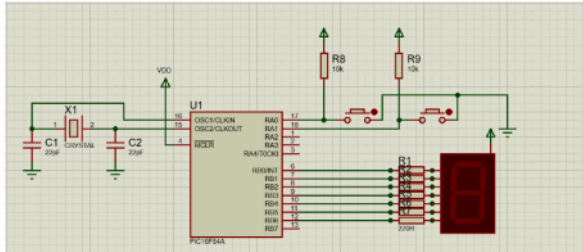


Figure 1 PIC16F84 circuit with seven segment display made in proteus

Circuit Description and Component Functions

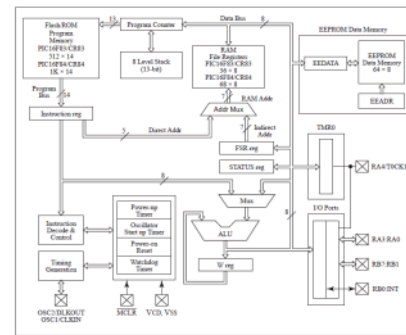
This circuit uses a PIC16F84A microcontroller to control a 7-segment display in Proteus. The microcontroller sends signals to the display via PORTB, turning specific segments on or off to show numbers or characters. Resistors limit the current to each segment, and a crystal oscillator with capacitors ensures stable operation. The circuit is powered by a 5V supply, and buttons or switches connected to PORTA can be used to adjust the displayed value.



Figure 2 PIC16F84A

1. PIC16F84A Microcontroller (U1):

This PIC is an 8 bit microcontroller that is part of the RISC (Reduced Instruction Set Computer) Family it is a low cost easy to use microcontroller due to its small instruction set of only 35, i will be using these commands along with ports RB0 through RB6 to output binary data to the 7 segment display, the reason this microcontroller is 8 bits is because the data bus is 8 bits wide, all processing, storage and retrieval is accomplished using these 8 bits (1 byte). The A in the PIC16f84A notates that this version is capable of running at 20Mhz instead of 10Mhz [1]



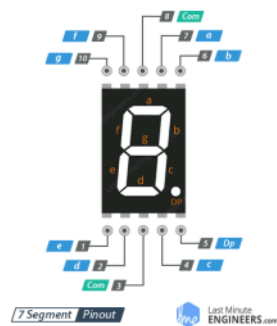


Figure 4. 7 segment display illustration

2. 7-Segment Display:

There are two main types of 7 segment displays: cathode and anode, [3] cathode is connected to ground and lights up each segment by bringing the specific pin high whereas anode (used in my circuit) is connected to VDD and the pins are brought low to illuminate the segments, to display a number we would need to turn off or on specific segments, this is done with an 8 bit pattern, it being 8 due to a decimal which can also be used. For example, The number 1 we would need only segment b and c illuminated being 11111001 in binary. 1 being off and 0 being on due to anode configuration. In hex this would be 0xF9 these are stored in a table in the code, and used to display each number and letter on the display based on which port is high or low

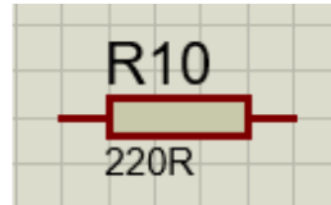


Figure 5 Resistor R10 at 220 ohm

3. 220 Ω Resistors (R1-R7):

These resistors limit the current flowing through each segment of the 7-segment display. An LED segment typically needs around 10–20 mA for proper illumination, and driving it directly from a microcontroller I/O pin without a resistor would risk burning out both the LED segment and possibly the microcontroller pin.

By adding a 220 Ω resistor in each segment line, we ensure that the current stays within safe limits the LED segments.



Figure 6. Push Button Switch in Proteus

4. Push Buttons (connected via R8 and R9, 10 k Ω pull-up resistors):

The two push to provide the user input for the PIC

R8 and R9 are 10 k Ω resistors connected to VDD. The switches are connected to RA0 and RA1 on the PIC. Because the circuit is arranged so that the pins normally see VDD (through the pull-ups), a not-pressed button reads as logic high or 1. When the button is pressed, the pin is pulled to ground, resulting in a logic low or 0.

In the code, we check these bits to detect a button press. Specifically:

RA0 (the “left” button) triggers a decrement if pressed.

RA1 (the “right” button) triggers an increment if pressed.

There’s a small software delay/debounce routine in the code to prevent false multiple triggers if the button’s contacts bounce.



Figure 7. Crystal Oscillator 1Mhz

5. Crystal Oscillator (X1) and 22 pF Capacitors (C1, C2):

The purpose of the crystal Oscillator is to provide the PIC micro controller with pulses which act as the clock of the microcontroller and controls the operation speed, for this design i chose to use a 1Mhz Oscillator thus giving the PIC a clock speed of 1Mhz. It generates this clock speed by using a small quartz crystal located next to an electrode, when a voltage is applied to that electrode the crystal will oscillate giving the clock speed it was manufactured for, however there is a need for 2 capacitors in this setup to ensure that it stays at its stable operating frequency, this is called load capacitance and the data sheet of the specific crystal oscillator should show which to use however, in very sensitive electronics you may need to take into account the parasitic capacitance of things like PCB traces and pins to get the correct values. The data sheet states that each instruction is 4 cycles [2] giving the pic an operating speed of 250,000 operations per second. This could be changed to 4Mhz to make better use of the PIC’s speed.

SECTION 2

PLANNING PSEUDOCODE:

To display the hexadecimal characters (0-F) on the 7 segment display, we need to map each hexadecimal to its corresponding 7-segment display pattern. The program will increment the counter from 0 to F (0-15) and for each value, it converts it to a 7-segment display pattern which is used to control the ports.

Microcontroller Ports:

Set PORTB as outputs to control the 7-segment display.

PORTA as inputs for the 2 buttons

A counter:

Start with the counter at 0 and increment or decrement

Main Loop:

Convert the current count (0-15) to a 7-segment display pattern using a lookup table.

Send the display pattern to PORTB.

Increment the count, and if it reaches 16, reset it back to 0.

Hex to 7-Segment Conversion:

Each hexadecimal digit (0-F) corresponds to a specific 7-segment display pattern.

For example, the digit 0 is represented by 0xC0 all but the middle segment will be on.

This in binary would be 1000 0000

We can use a table to store these.

Pause:

A delay loop used to slow down the counter increments, and also software debouncing for the buttons presses.

SECTION 3

1. MODIFY "COUNT.ASM" SO THAT THE SEVEN SEGMENT DISPLAY COUNTS FROM 0 TO F IN HEXADECIMAL NUMBERS IN 0.2 SECOND INCREMENTS.

```

;=====
; Main.asm: Program for an auto-incrementing counter (0 to F)
; Increments every 0.2 seconds
; Processor: PIC16F84A
;=====

#include p16f84a.inc

; Register definitions
portB equ 0x06          ; Register for output (PORTB)
trisb equ 0x06          ; Register for configuring PORTB direction
status equ 0x03         ; Status register
RP0 equ 5               ; Bank select bit

; Variable definitions
count equ 0x0C          ; Holds the current counter value
nCount equ 0x0D         ; Inner loop counter for delay
mCount equ 0x0E         ; Outer loop counter for delay

; Reset vector
RST code 0x0            ; Reset starts at address 0x0

    goto Start          ; Jump to the main program

; Main program code
PGM code
Start
    ; Configure ports
    bsf status, RP0     ; Switch to Bank 1 (for configuration)
    movlw 0x00          ; Load 0x00 into working register
    movwf trisb         ; Set all PORTB pins as output
    bcf status, RP0     ; Switch back to Bank 0

    ; Initialize counter
    clrf count          ; Set counter value to 0

main_loop
    movf count, W        ; Load current counter value
    call table          ; Convert counter value to 7-segment pattern
    movwf portB         ; Send the pattern to PORTB
    call pause          ; Wait for 0.2 seconds
    incf count, F       ; Increment the counter
    movlw 0x10          ; Load the maximum value (16, or 0x10)
    subwf count, W      ; Check if counter equals 16
    btfsc STATUS, Z     ; If zero flag is set (count == 16), skip next
    clrf count          ; Reset counter to 0 if it reached 16
    goto main_loop      ; Repeat the main loop

; Delay routine for 0.2 seconds
pause
    movlw 0xC8          ; Load the outer loop count (200)
    movwf mCount        ; Initialize outer loop counter

loadN
    movlw 0xC8          ; Load the inner loop count (200)
    movwf nCount        ; Initialize inner loop counter

decN
    decfsz nCount, F    ; Decrement inner loop counter; skip if zero
    goto decN          ; Repeat inner loop until zero
    decfsz mCount, F    ; Decrement outer loop counter; skip if zero
    goto loadN         ; Repeat outer loop until zero
    return             ; End of delay routine

; Lookup table for 7-segment display patterns (0 to F)

```

```

table
    addwf PCL                ; Offset lookup based on counter value
    retlw 0xC0               ; Pattern for 0
    retlw 0xF9               ; Pattern for 1
    retlw 0xA4               ; Pattern for 2
    retlw 0xB0               ; Pattern for 3
    retlw 0x99               ; Pattern for 4
    retlw 0x92               ; Pattern for 5
    retlw 0x82               ; Pattern for 6
    retlw 0xF8               ; Pattern for 7
    retlw 0x80               ; Pattern for 8
    retlw 0x90               ; Pattern for 9
    retlw 0x88               ; Pattern for A
    retlw 0x83               ; Pattern for b
    retlw 0xC6               ; Pattern for C
    retlw 0xA1               ; Pattern for d
    retlw 0x86               ; Pattern for E
    retlw 0x8E               ; Pattern for F
END

```

2. "WRITE A PROGRAM SO THAT WHEN YOU PRESS ONE OF THE TWO BUTTONS, IT DISPLAYS "L" ON THE 7 SEGMENT LED DISPLAY, THEN WHEN YOU PRESS THE OTHER ONE, IT DISPLAYS "R".

```

;=====
; Main.asm: Program to display "L" or "R" on button press
; Processor: PIC16F84A
;=====

#include p16f84a.inc

; Register definitions
portB equ 0x06                ; Register for output (PORTB)
trisB equ 0x06                ; Register for configuring PORTB direction
portA equ 0x05                ; Register for input (PORTA)
trisA equ 0x05                ; Register for configuring PORTA direction
status equ 0x03               ; Status register
RP0 equ 5                     ; Bank select bit

; Variable definitions
dCount equ 0x0C               ; Temporary variable for delay counter

; Reset vector
RST code 0x0                  ; Reset starts at address 0x0
    goto Start                ; Jump to the main program

; Main program code
PGM code
Start
    ; Configure ports
    bsf status, RP0           ; Switch to Bank 1 (for configuration)
    movlw 0x00                ; Load 0x00 into working register
    movwf trisB                ; Set all PORTB pins as output
    movlw 0x0F                ; Load 0x0F into working register
    movwf trisA                ; Set lower 4 bits of PORTA as input
    bcf status, RP0           ; Switch back to Bank 0

main_loop
    movf portA, W              ; Read the state of PORTA (buttons)

    btfss portA, 0             ; Check if the left button (bit 0) is pressed
    call show_L                ; If pressed, display "L"

```

```

    btfss portA, 1          ; Check if the right button (bit 1) is pressed
    call show_R            ; If pressed, display "R"

    goto main_loop         ; Repeat the loop to continuously check buttons

show_L
    movlw 0x47             ; Load pattern for "L" into working register
    movwf portB            ; Send the pattern to PORTB to display "L"
    call delay             ; Add a delay for debounce
    return                ; Return to the main loop

show_R
    movlw 0x4E             ; Load pattern for "R" into working register
    movwf portB            ; Send the pattern to PORTB to display "R"
    call delay             ; Add a delay for debounce
    return                ; Return to the main loop

; Debounce delay routine
delay
    movlw 0xFF             ; Load the maximum count (255) into working register
    movwf dCount           ; Initialize the delay counter
delay_loop
    decfsz dCount, F       ; Decrease the counter and skip next if zero
    goto delay_loop       ; Repeat until the counter reaches zero
    return                ; Return after the delay

END

```

3. "WRITE ANOTHER PROGRAM, SO ONE BUTTON CAUSES ALL OF THE LED SEGMENTS TO LIGHT, AND THE OTHER SWITCH TURNS THEM OFF. (THE SWITCH ACTIVATIONS SHOULD BE 'LATCHING' I.E. YOU DON'T HAVE TO HOLD THE BUTTON DOWN FOR THE ON/OFF EFFECT TO REMAIN)"

```

;=====
; Main.asm: Program to turn all segments of a 7-segment display ON or OFF
; Processor: PIC16F84A
;=====

#include p16f84a.inc

; Register definitions
portB equ 0x06             ; Register connected to 7-segment display (output)
trisB equ 0x06             ; Direction register for PORTB
portA equ 0x05             ; Register connected to buttons (input)
trisA equ 0x05             ; Direction register for PORTA
status equ 0x03            ; Status register
RP0 equ 5                  ; Bit to select register bank

; Variables
dCount equ 0x0C            ; Counter for delay

; Reset vector
RST code 0x00              ; Reset starts at address 0x00
    goto Start             ; Jump to the main program

; Main code
PGM code
Start
    ; Configure ports
    bsf status, RP0        ; Switch to Bank 1 for configuration
    movlw 0x00             ; Set PORTB pins as output
    movwf trisB
    movlw 0x0F             ; Set lower 4 bits of PORTA as input
    movwf trisA
    bcf status, RP0        ; Switch back to Bank 0

main_loop

```

```

; Check if left button (RA0) is pressed
btfss portA, 0      ; skip if RA0 is not pressed
call all_on         ; if pressed, turn all segments ON

; Check if right button (RA1) is pressed
btfss portA, 1      ; skip if RA1 is not pressed
call all_off        ; if pressed, turn all segments OFF

goto main_loop      ; continue checking buttons

all_on
    movlw 0x00      ; load value to turn all segments ON
    movwf portB     ; Output to PORTB
    call delay      ; add debounce delay
    return          ; return to main loop

all_off
    movlw 0xFF      ; Load value to turn all segments OFF
    movwf portB     ; Output to PORTB
    call delay      ; Add debounce delay
    return          ; Return to main loop

; Debounce delay
delay
    movlw 0xFF      ; Load delay duration
    movwf dCount    ; Initialize delay counter
delay_loop
    decfsz dCount, F ; Decrease counter and skip if zero
    goto delay_loop ; Repeat until counter reaches zero
    return          ; End of delay routine

END

```

4. "THEN COMBINE ALL OF THESE TECHNIQUES TOGETHER, BY WRITING A NEW PROGRAM WHERE YOU START BY DISPLAYING '5' BUT YOU CAN COUNT UP (TO A MAXIMUM OF 9) IF YOU PRESS ONE BUTTON, AND COUNT DOWN (TO A MINIMUM OF 0) IF YOU PRESS THE OTHER."

```

;=====
; Main.asm: Program for an up/down counter (0-9, A-F) starting at 5
; Processor: PIC16F84A
; Description:
;   - A 7-segment display is connected to PORTB for output.
;   - Two buttons are connected to PORTA (RA0 and RA1):
;     * Press RA0 to decrement the counter (down).
;     * Press RA1 to increment the counter (up).
;   - The counter starts at 5, increments up to F, and decrements down to 0.
;   - The 7-segment display continuously updates to reflect the current count.
;=====

include p16f84a.inc

;-----
; Register definitions
;-----
portB      equ    0x06    ; Output register for the 7-segment display
trisb      equ    0x06    ; Data direction register for PORTB
portA      equ    0x05    ; Input register for buttons
trisa      equ    0x05    ; Data direction register for PORTA
status     equ    0x03    ; Status register
RP0        equ    5       ; Bank select bit for switching between memory banks

;-----
; Variable definitions
;-----
count      equ    0x0C    ; Stores the current count value (0-F)
lastState  equ    0x0D    ; Stores the last button state to detect changes
dCount     equ    0x0E    ; Loop counter used in the delay routine

;-----
; Reset vector: Starts program execution after a reset
;-----
RST        code    0x0000
           goto    Start    ; Jump to the main program entry point

;-----
; Main program
;-----
PGM        code
Start
;-----
; STEP 1: Configure Input/Output ports
;-----
        bsf      status, RP0    ; Switch to Bank 1 for configuration
        movlw    0x00           ; Set all PORTB pins as outputs
        movwf    trisb
        movlw    0x0F           ; Set lower 4 bits of PORTA as inputs (RA0, RA1)
        movwf    trisa
        bcf      status, RP0    ; Switch back to Bank 0 for normal operation

;-----
; STEP 2: Initialize variables
;-----
        movlw    0x05           ; Set the initial count to 5
        movwf    count
        movlw    0xFF           ; Initialize `lastState` to detect button changes
        movwf    lastState

;-----
; Main loop: Continuously check buttons and update the display
;-----
main_loop

```

```

; STEP 3: Detect button state changes
movf    portA, W      ; Read current button states
xorwf   lastState, W  ; Compare with previous state
andlw   0x03          ; Mask unused bits (only care about RA0, RA1)
btfsc   STATUS, Z     ; Skip if no change detected
goto    display       ; If no change, proceed to update the display

movf    portA, W      ; Update lastState with current button states
movwf   lastState

; STEP 4: Handle button presses
btfss   portA, 0      ; Check RA0 (left button)
call    left_button   ; If pressed, decrement the count
btfss   portA, 1      ; Check RA1 (right button)
call    right_button  ; If pressed, increment the count

;-----
; STEP 5: Update the 7-segment display with the current count
;-----
display
    movf    count, W      ; Load the current count into W
    call    table         ; Convert the count to a 7-segment pattern
    movwf   portB        ; Output the pattern to PORTB
    call    delay         ; Add a delay for debounce and visibility
    goto    main_loop     ; Repeat the main loop

;-----
; Subroutines
;-----

; Increment the count, stopping at F (15 in decimal)
right_button
    movlw   0x09         ; Set the upper limit of 9
    subwf   count, W     ; Compare current count
    btfsc   STATUS, Z     ; If count == 15, skip increment
    return
    incf    count, F     ; Increment the count
    return

; Decrement the count, stopping at 0
left_button
    movf    count, W     ; Load the current count into W
    btfsc   STATUS, Z     ; If count == 0, skip decrement
    return
    decf    count, F     ; Decrement the count
    return

; Simple delay routine for debounce and display visibility
delay
    movlw   0xFF         ; Set loop counter to 255
    movwf   dCount

delay_loop
    decfsz  dCount, F     ; Decrement the counter, skip if zero
    goto    delay_loop   ; Repeat until counter reaches zero
    return

; Lookup table for 7-segment display patterns (0-9, A-F)
table
    addwf   PCL          ; Use 'count' to index into the table
    retlw   0xC0          ; 0
    retlw   0xF9          ; 1
    retlw   0xA4          ; 2
    retlw   0xB0          ; 3
    retlw   0x99          ; 4
    retlw   0x92          ; 5
    retlw   0x82          ; 6
    retlw   0xF8          ; 7
    retlw   0x80          ; 8
    retlw   0x90          ; 9

```

2317250

```
END                ; End of program
```

REFERENCES:

- [1] <https://www.electricalandcontrol.com/microcontroller-pic16f84/> - 11/01/2025
- [2] https://www.pcbheaven.com/picpages/Instruction_Cycle_Duration_and_Programmable_Delays/ - 13/01/2025
- [3] <https://www.mathaelectronics.com/interfacing-of-seven-segment-display-with-arduino/> - 13/01/2025

Micros Peripherals and Interfaces - coursework - 2025

-2317250

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

48/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

ENGAGEMENT WITH LITERATURE

 Only three reference sources cited - need to have much more than this and they should cover all aspects/components.

35

KNOWLEDGE AND UNDERSTANDING

 Despite the limited EWL, the actual information provided is reasonable. However the buzzer and voltage reg have not been included!

45

COGNITIVE AND INTELLECTUAL

 Of the four programming tasks attempted, these show a good level of flair and don't simply rely on the brute force approach.

65

PRACTICAL APPLICATION

 Only a part of the overall circuit has been simulated in proteus.
Its as if the bare minimum has been done to pass!

45

TRANSFERABLE SKILLS



More care/time could have been put into making the layout professional and consistent.

50