

micro peripherals and interfaces robbie young

by Robbie Young

Submission date: 28-Nov-2022 02:39PM (UTC+0000)

Submission ID: 190323095

File name:

32858_Robbie_Young_micro_peripherals_and_interfaces_robbie_young_466098_1295613068.docx (1.19M)

Word count: 4268

Character count: 18703



Use of 1st Person Voice

In class I produced the count project, this runs a common anode seven segment. A common anode will light up a segment when a 0 is on the output of the microchip. (More information on the two types of seven segments and how they work can be found in figure 1). The count project is designed to counts upwards in binary and display it on the seven-segment, wait one second, repeat until it reaches 255, where it will then reset to zero subsequently repeating indefinitely.

```
*main.asm
1 ; FILE: count.asm
2 ; AUTHOR: Robbie Young
3 ; DATE: 26/11/22
4 ; DESCRIPTION: Count to 255 at 1 second intervals and display on a seven
5 ; segment display in binary.
6 ; The pause subroutine takes apx. 0.2 sec.
7 ; Pause is called 5 times
8
9 list p=16F84 ; select processor type
10 radix hex ; define radix as hexadecimal
11 #include "p16f84.inc"
12
13 ;-----
14 ; cpu equates (memory map)
15 portB equ 0x06 ;port B address
16 trisB equ 0x06 ;trisB address in register bank 1
17
18 status equ 0x03 ;status register
19 RP0 equ 5 ;bit 5 of status register
20 RP1 equ 6 ;bit 6 of status register
21
22 count equ 0x0c ; counter 0 to 255
23 nCount equ 0x0d ; pause routine inner loop counter
24 mCount equ 0x0e ; pause routine outer loop counter
```



(Figure 2)

Figure 2 is the first 24 lines of code in the program count. Lines 1 to 12 have no effect on the program, they just set up a few things and give a description of the programme including author and date. The two important lines of code are number 9 and 10. Number 9 selects the microcontroller used, this is very important because the software used needs to identify the corresponding compiler it needs to use, so it can translate the code correctly. Line 10 is used to define radix as a hexadecimal. figure 3 gives you more information about what radix does.

Lines 13 to 24 creates a memory map. Lines 21,22,23 are variables which will be used later in the code. The variable 'count' is the most important as this is what is outputted to the seven segment and is what will be displayed. Ncount and mcount are both used for timings later in the code.

The final few lines in figure two (14-19) are being used to set up addresses, so they can be found later when trying to set up portb to do the correct function.

```

25 ;-----
26
27     org     0x000
28
29 start
30     bcf     status,RP0      ; clear bank selection bits in the status reg
31     bcf     status,RP1
32     bsf     status,RP0      ; select bank 1
33     movlw   0x00           ; Load w with 0x00
34     movwf   trisb          ; Load it into trisb to make portb all outputs
35     bcf     status,RP0      ; reselect bank 0
36     clrf    portB          ; clear all lines Low
37     clrf    count          ; initialize counter to 0
38
39 get_count
40     movf    count, w        ; move count to W
41     movwf   portB           ; move W to port B to update the display
42     addlw   1               ; add +1 to W
43     movwf   count          ; save W in count
44     call    pause           ; delay 200ms by subroutine
45     call    pause           ; five pause executions equals ~ 1 second
46     goto    get_count       ; repeat forever
47
48 ;-----

```

(Figure 4)

Figure 4 is the next few lines of code. Line 29 is the start process. It is a subroutine that will start the code, initialises a few things and clears portb and count, then starts the process of 'get_count'. 'Start' is a subroutine that can be called anytime in the code, but because its job is to initialise things, 'Start' should never be called. It would only be called if something were to change later in the code.

Lines 30, 31, use the command bcf (bit clear F) this will clear (means set to 0) file register (f). Line 30 is clearing bit RP0 of the status register and 31 is doing the same but to RP1. Line 32 is using bsf (bit set F) this is setting (set to 1) file register (f). It sets RP0 of the status register back to a 1 so that portb can be set up to do this, line 33 is using the command movlw this sets the working register to 00000000 then line 34 then loads it into trisb. Once ran all of portb is an output. Portb can only change its state because RP0 is set to 1, this allows for trisb to be edited. If a logical 1 is set in trisb that means whichever corresponding bit in portb will be an input, if it is a 0 it will be an output. Line 35 clears RP0 making trisb unchangeable therefore portb will always be an output to the seven-segment display. Lines 36 and 37 are clrf commands this means that instead of clearing each bit it clears the whole byte so both portb and count are set to 0.

Now we get to the main loop within the code, 'get_count', this is a subroutine that can never be exited, therefore will repeat until it crashes. The first line of code within 'get_count' is movf this retrieves count out of the memory and moves it to the working register. Line 41 is movwf this moves whatever is in the working register to file (f). In this case moving 'count' to portb subsequently, printing the current value of 'count' to the seven-segment display. Line 42 uses Addlw this is a simple command that just add 1 to the working register, this currently is the variable 'count' so that it will increase by one. Line 43 then saves 'count' back to itself in the memory therefore updating itself. The next lines are 'call pause' there is meant to be 5. The reason for this is because each one pauses the program for 200ms. I have only got two in figure 4 because whilst testing the program I didn't want to wait 1 second each time for the program to update itself, so I cut it down to just 0.4 seconds. The call command is used to call in subroutines instead of going to them, the difference is you are not breaking out of the main subroutine you are just adding a few extra steps in. Return command is used to call you back to the place in the main code. The last line of the of figure 4 is the goto command, 'goto' command goes to the subroutine called and not return. This is why the code

produced will repeat.

```
48 ;-----
49 ;subroutine pause: Lets waste some time...
50 ;
51 ;inputs: nothing
52 ;outputs: nothing
53 ;
54 ;does: nothing but waste some time
55 ;corrupts: register W
56 ;
57 ;description: two loops using "mCount" and "nCount" register files
58 ;               "mCount" is the outer loop counter (255 down to zero)
59 ;               "nCount" is the inner loop counter (255 down to zero)
60 ;
61 ;-----
```

(Figure 5)

Figure 5 is lines 48 to 61 this does nothing as it is just comments. However, it does explain a few things.

```
61 ;-----
62 pause
63     movlw    0xff          ; set w = 255 decimal
64     movwf    mCount        ; mCount = w: outer loop counter
65 loadN
66     movlw    0xff          ; set w = 255 decimal
67     movwf    nCount        ; nCount = w: inner loop counter
68 decN
69     decfsz   nCount, f      ; nCount--
70     goto     decN          ; if nCount != 0 then repeat nCount--
71
72     decfsz   mCount, f      ; else decrement Count
73     goto     loadN         ; if mCount != 0 then
74                             ; reload nCount to 255 and decrement
75     return                ; else exit subroutine
76
77     end
78
79 ;-----
```

(Figure 6)

Figure 6 is the last part of the count project. Its purpose is for timing, when pause is called on line 44 this is where the code will go to. Lines 63 and 64 are used to load 255 into the variable mcount. Loadn does the same thing as pause but does it to the variable ncount. Decn is the main timer of the pause loop. Line 69 uses decfsz (Decrement f, Skip if 0) this command decreases (f) by one and when it reaches 0 it will skip the next command so presuming it starts at 255 it will do this command 255 times before it skips the goto command which forces its repeat.

7 seg	hex
0	0x40
1	0x79
2	0x24
3	0x30
4	0x19
5	0x12
6	0x02
7	0x78
8	0x00
9	0x10
A	0x08
B	0x03
C	0x46
D	0x21
E	0x06
F	0x0E

(Figure 7)

Figure 7 is a table. on the left you have what would be displayed on a common anode seven segment display and on the right is what hex value is need to be outputted by a microchip, this would later be used by me to make the seven segment display go from 0 to F.

```

;-----
;      cpu equates (memory map)
portB equ 0x06      ;port B address
trisb equ 0x06      ;trisB address in register bank 1

status equ 0x03      ;status register
RP0 equ 5            ;bit 5 of status register
RP1 equ 6            ;bit 6 of status register

count equ 0x0c       ; counter 0 to 255
nCount equ 0x0d      ; pause routine inner loop counter
mCount equ 0x0e      ; pause routine outer loop counter
;-----

      org 0x000

start
    bcf status,RP0    ; clear bank selection bits in the status reg
    bcf status,RP1
    bsf status,RP0    ; select bank 1
    movlw 0x00        ; load w with 0x00
    movwf trisb       ; load it into trisb to make portb all outputs
    bcf status,RP0    ; reselect bank 0
    clrf portB        ; clear all lines low
    clrf count        ; initialize counter to 0

```

(Figure 8)

Figure 8 is my code for section 3 part 1, and as you can see I didn't change anything from the original count program. The only thing I need to change in the count project was the subroutine get_count, so it would display 0 to F on the seven segment display.

```
62 get_count
63     movlw    0x40
64     movwf    portB
65     call     half
66     movlw    0x79
67     movwf    portB
68     call     half
69     movlw    0x24
70     movwf    portB
71     call     half
72     movlw    0x30
73     movwf    portB
74     call     half
75     movlw    0x19
76     movwf    portB
77     call     half
78     movlw    0x12
79     movwf    portB
80     call     half
81     movlw    0x02
82     movwf    portB
83     call     half
84     movlw    0x78
```

(Figure 9)

Figure 9 is the start of the new 'get_count' this displays values from figure 7 and then waits half a second and slowly goes down the line. It uses call half instead of call pause because pause is 0.2 seconds and half is set to half a second.

```

84      movlw    0x78
85      movwf    portB
86      call     half
87      movlw    0x00
88      movwf    portB
89      call     half
90      movlw    0x10
91      movwf    portB
92      call     half
93      movlw    0x08
94      movwf    portB
95      call     half
96      movlw    0x03
97      movwf    portB
98      call     half
99      movlw    0x46
100     movwf    portB
101     call     half
102     movlw    0x21
103     movwf    portB
104     call     half
105     movlw    0x06
106     movwf    portB
107     call     half
108     movlw    0x0E
109     movwf    portB
110     call     half
111     goto     get_count

```

(Figure 10)

Figure 10 is the rest of 'get_count' and as you can see it just displays a value and waits. The last line of code goto get_count makes sure it repeats and goes back to the top of 'get_count' ensuring a repeating process.

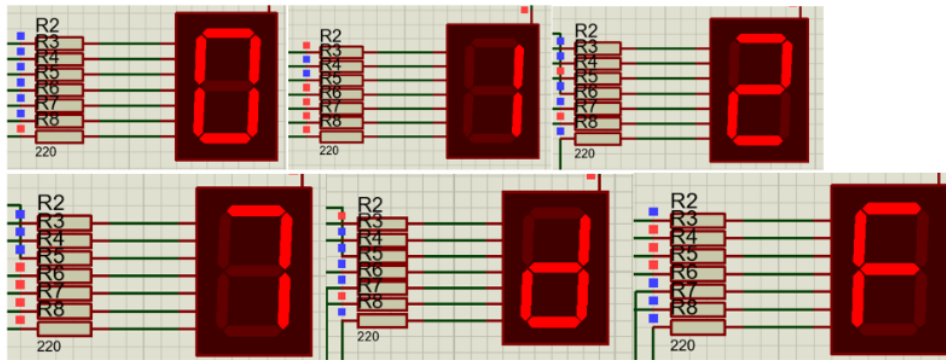
```

88  ;-----
89  pause
90      movlw    0xff      ; set w = 255 decimal
91      movwf    mCount    ; mCount = w: outer loop counter
92  loadN
93      movlw    0xff      ; set w = 255 decimal
94      movwf    nCount    ; nCount = w: inner loop counter
95  decN
96      decfsz   nCount, f  ; nCount--
97      goto     decN      ; if nCount != 0 then repeat nCount--
98
99      decfsz   mCount, f  ; else decrement Count
100     goto     loadN      ; if mCount != 0 then
                            ; reload nCount to 255 and decrement
101
102     return          ; else exit subroutine
103
104  half
105      call     pause
106      call     pause
107      return
108
109
110  end

```

(Figure 11)

figure 11 shows you the last part of the code and as you can see I didn't change the pause subroutine so it still a 0.2 second delay. The subroutine I added for this was 'half' this is currently only 0.4 seconds instead of the 0.5. The reason it was left like this, is because wanted to test my code to ensure it worked. I later added 0.1 seconds by using nops. Nop is a command that does nothing but waste time I didn't use them in the screenshot because I didn't want to wait whilst testing.



(Figure 12)

Figure 12 shows the seven segment display with 0,1,2,7,d and displayed on it these are just some of screen shots the code ran.

```

32      bsf      status,RP1
33      movlw    0x03
34      movwf    trisa
35      bcf      status,RP1

```

(Figure 13)

Figure 13 is what I did to change my code so that I could do section 3 part 2. I needed to read the buttons as inputs, to do this, I added lines 32 to 35 these make AO0 and AO1 inputs allowing for the buttons to be read.

```

45 get_count
46     btfsc     porta, 0
47     goto      but1
48     btfsc     porta, 1
49     goto      but2
50     goto      get_count
51
52 but1
53     clrf      portB
54     movlw     0x40
55     movwf     portB
56     goto      get_count
57
58 but2
59     clrf      portB
60     movlw     0x79
61     movwf     portB
62     goto      get_count

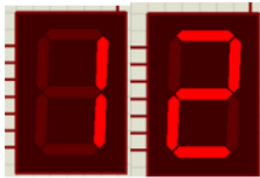
```

(Figure 14)

Once I could use the two inputs, the code I designed to do part 2 of section 3 would work as I could now get 'get_count' subroutine to read the two button inputs. If either one has a logical one on the input it will go to its respective button 6 times, to make that happen I used the Btfsc (bit test f, skip if clear) command. It works by reading the value on porta pin 0, if there is a logical 0 it will skip the next command. If there is a logical 1 it will do the next command taking it to the corresponding function. Line 50 will ensure it repeats until either button is pressed.

Subroutine but1 starts by clearing portb. This will make all lights turn on very momentarily, but does ensure that no legacy lights will be on. When the next command of movlw and then movwf are executed. Movlw sets 0x40 which is hex for 1 in this case. Next it gets moved from the working register to port b where it is displayed. The last line of code is the goto command, this will make sure the code goes back to get_count where it will then read the button inputs again.

Subroutine but2 does almost the same thing the only difference is the hex value is changed to 0x79 this is the hex value for 2 when displayed on to the seven segment.



(Figure 15)

Figure 15 shows both 1 and 2 being printed to the seven segment display.

```
45 get_count
46     btfsc     porta, 0
47     goto but1
48     btfsc     porta, 1
49     goto but2
50     goto get_count
51
52 but1
53     clrf      portB
54     movlw     0xFF
55     movwf     portB
56     goto     get_count
57
58 but2
59     clrf      portB
60     goto     get_count
61
```

(Figure 16)

Figure 16 shows the modifications I made to the code for section 3 part 3. The 2 changes I made to the code were: to start with line 54 where I changed the 0x40 to 0xFF. This changes what is outputted so instead of one lighting up it would be none of the lights would light up. The second change I did was removing the two lines under line 59 this made it so it wouldn't show two it would just leave all the lights on.

```

29 start
30     bcf      status,RP0      ; clear bank selection bits in the status reg
31     bcf      status,RP1
32     bsf      status,RP1
33     movlw    0x03
34     movwf    trisa
35     bcf      status,RP1
36
37     bsf      status,RP0      ; select bank 1
38     movlw    0x00           ; Load w with 0x00
39     movwf    trisb          ; Load it into trisb to make portb all outputs
40     bcf      status,RP0      ; reselect bank 0
41
42     clrf     portB           ; clear all lines Low
43     clrf     count          ; initialize counter to 0
44
45 get_count                                ;get_count is used to cheak the button inputs
46     btfsc    porta, 0        ;checks button one skips the line under if it is clear
47     goto     but1            ;if porta pin 0 is high then got to but1
48     btfsc    porta, 1        ;checks button two skips the line under if it is clear
49     goto     but2            ;if porta pin 1 is high then got to but1
50     goto     get_count        ;goes back to the start of the cycle
51
52 but1                                ;subroutine but1 used decrease the seven segment display
53     movf     count, w        ;moves count into the working registor
54     decfsz   count, w        ;decreases count by one skips if it is 0
55     goto     number          ;goes to the subroutine number to find the value of count
56     goto     pzo             ;count is 0 so goto print 0
57

```

(Figure 17)

Figure 17 is the start of my code for section 3 part 4. Nothing changed in the start-up process, this is because I already had everything setup previously so didn't need to change anything. Again, I used the same 'get_count' subroutine as there was no changes needed there either.

The first changes I made to make my code work was to but1, instead of it turning everything off it now counts down as decfsz (Decrement f, Skip if 0) decreases 'count' by one. When 'count' is 0 the next line is skipped therefore goto number is used and goto print zero is only used if 'count' is 0, meaning that 0 will be printed to the seven segment display. When 'count' is anything but 0 the code will move to number.

```

57
58 but2                                ;subroutine but2 is used them button 2 is pressed and iwill increment count by 1
59     movf     count, w        ;moves count into the working registor
60     incf     count, count    ;increment count by 1 and then save it back on its self
61     btfsc    count, 0x10     ;checks bit number 5 of count
62     goto     reset           ;if it is a one reset count
63     goto     number          ;if bit 5 is a 0 find the value of count
64
65 reset                                ;reset subroutine used to reset count
66     movlw    0x0F           ;moves 00001111 into the working register
67     movwf    count          ;saves it back over count
68     goto     get_count        ;goes back to the button cheak subroutine
69
70 number                                ;checks what number needs to be printed
71     movf     count, w        ;moves count into the working registor
72     btfss    count, 0x08     ;checks bit 4 of count
73     goto     nse-zo          ;will be skiped if if bit four is set if it clear then goto nse-zo
74     btfss    count, 0x04     ;checks bit 3 of count
75     goto     nei-a           ;will be skiped if if bit four is set if it clear then goto nei-a
76     btfss    count, 0x02     ;checks bit number 2 of count
77     goto     nc-d            ;will be skiped if if bit four is set if it clear then goto nc-d
78     btfss    count, 0x01     ;checks bit number 1 of count
79     goto     pe              ;will redirect the code to pe
80     goto     pf              ;will redirect the code to pf
81

```

(Figure 18)

Figure 18 is the next part of my code and has 3 subroutines in it but2, reset and number. But2 is the subroutine for when button 2 is pressed, it counts upwards by adding 1 to 'count' and then it will check bit 5 to see if there is a logic 1 there, this check happens because if bit five is a 1 then 'count'

has gone to high and would loop back on itself printing 0. If bit 5 is a 1 then we go to the reset function, this function resets 'count' to its previous value making sure the same value is shown on the seven-segment display.

The last subroutine is 'number' this is just the start process in finding out what the value of 'count' is. It does this by first checking bit number 4 in the sequence. If bit number 4 is a logic one that means that the value of 'count' is 8 or higher if there is a 0 in place for the one that means 7 or lower. Once bit 4 is read if it is a 0 then goto nse-zo (number 7 to 0) if 1 continue on and find out the next bit (3). If bit 3 is a 0 then we move on to the eight to A check, if it is a 1, move on to bit 2 where we look for a one again, if it is not there we are in the range of c or d but if a one is present look at the last bit to find out if we are looking at e or f.

```

81
82 nse-zo          ;nse-zo (number seven to zero)
83      movf       count, w          ;moves count into the working register
84      btfss      count, 0x04       ;checks bit number 3 in count
85      goto       nth-zo           ;if bit number 3 is 0 then goto nth-zo
86      btfss      count, 0x02       ;checks bit 2 of count
87      goto       nfi-fo           ;if bit 2 is 0 then goto nfi-fo
88      btfss      count, 0x01       ;checks bit 1 of count
89      goto       psi              ;if 0 then goto psi
90      goto       pse              ;if one then goto pse
91
92 nth-zo          ;nth-zo (numbers three to zero)
93      movf       count, w          ;moves count to the working register
94      btfss      count, 0x02       ;checks bit 2 of count
95      goto       nzo-on           ;if bit 2 is 0 then goto nzo-on
96      btfss      count, 0x01       ;checks bit 1
97      goto       ptw              ;if bit one is 0 then goto print two
98      goto       pth              ;if bit 1 is 1 then goto print three
99
100 nzo-on          ;nzo-on (numbers zero to one)
101      movf       count, w          ;moves count to the working register
102      btfss      count, 0x01       ;checks bit 1 of count
103      goto       pze              ;if 0 then goto print zero
104      goto       pon              ;if 1 then goto print one
105
106 nfi-fo          ;nfi-fo (numbers five to four)
107      movf       count, w          ;moves count to the working register
108      btfss      count, 0x01       ;checks bit 1 of count
109      goto       pfo              ;if bit one is a 0 then goto print four
110      goto       pfi              ;if bit one is 1 then goto print five
111

```

(Figure 19)

Figure 19 is the next lines of code, which contain 4 more checkers to find the value of 'count' they all work almost identical. The only thing is the destination as seen in nse-zo, it is looking at a large range of 7 to 0. It slowly gets filtered until the correct value for 'count' is found. Once found the code will be sent to the corresponding print command execute cycle, for example lines 88 to 90 find the last bit value and subsequently send it of to either print six or print seven depending if it bit 1 is a 0 or 1.

```

111
112 nc-d                                ;nc-d (numbers c to d)
113     movf    count, W                ;moves count to the working register
114     btfss   count, 0x01             ;checks bit 1 of count
115     goto    pc                      ;if 0 then goto print c
116     goto    pd                      ;if 1 then goto print d
117
118 nei-a                                ;nei-a (numbers eight to a)
119     movf    count, W                ;moves count to the working register
120     btfss   count, 0x02             ;checks bit 2
121     goto    nei-ni                 ;if 0 then goto eight to nine
122     btfss   count, 0x01             ;if 1 check bit number 1
123     goto    pa                      ;if 0 goto print a
124     goto    pb                      ;if 1 goto print b
125
126 nei-ni                               ;nei-ni (numbers eight to nine)
127     movf    count, W                ;moves count to the working register
128     btfss   count, 0x01             ;checks bit number 1
129     goto    pei                    ;if 0 then goto print eight
130     goto    pni                    ;if 1 goto print nine
131
132 pze                                  ;print 0
133     movlw   0x40                    ;moves hex of 0 into the wroking register
134     movwf   portB                   ;prints the working register to portb
135     goto    get_count               ;goes back to button selection subroutine
136
137 pon                                  ;print 1
138     movlw   0x79                    ;moves hex of 1 into the wroking register
139     movwf   portB                   ;prints the working register to portb
140     goto    get_count               ;goes back to button selection subroutine
141

```

(Figure 20)

Figure 20 has the last few checkers in it and the a few print commands. The last few checkers work the same as before, but all are in the eight and above categories. The more interesting part of figure 20 is pzo and pon subroutines, they will print either a 0 or 1 to the seven segment where it is then displayed. To do this they both use the same method, if you look at lines 133 to 135 we can see how they do it, the first line (133) moves 0x40 in to the working register, this is the hex value needed to print 0, the next line then move 0x40 from the working register to portb, meaning that 0x40 will be sent to the seven segment display and printed, lighting up the number 0. The last line in the subroutine is the goto command this will send the code back to get_count this is where we find the input of the buttons therefore completing the cycle.

```

141
142 ptw                                  ;print 2
143     movlw   0x24                    ;moves hex of 2 into the wroking register
144     movwf   portB                   ;prints the working register to portb
145     goto    get_count               ;goes back to button selection subroutine
146
147 pth                                  ;print 3
148     movlw   0x30                    ;moves hex of 3 into the wroking register
149     movwf   portB                   ;prints the working register to portb
150     goto    get_count               ;goes back to button selection subroutine
151
152 pfo                                  ;print 4
153     movlw   0x19                    ;moves hex of 4 into the wroking register
154     movwf   portB                   ;prints the working register to portb
155     goto    get_count               ;goes back to button selection subroutine
156
157 pfi                                  ;print 5
158     movlw   0x12                    ;moves hex of 5 into the wroking register
159     movwf   portB                   ;prints the working register to portb
160     goto    get_count               ;goes back to button selection subroutine
161
162 psi                                  ;print 6
163     movlw   0x02                    ;moves hex of 6 into the wroking register
164     movwf   portB                   ;prints the working register to portb
165     goto    get_count               ;goes back to button selection subroutine
166
167 pse                                  ;print 7
168     movlw   0x78                    ;moves hex of 7 into the wroking register
169     movwf   portB                   ;prints the working register to portb
170     goto    get_count               ;goes back to button selection subroutine
171

```

(Figure 21)

Figure 21 is similar to figure 20 it just contains more print commands.

```

171
172 pei                                ;print 8
173     movlw    0x08                    ;moves hex of 8 into the wroking register
174     movwf    portB                  ;prints the working register to portb
175     goto     get_count               ;goes back to button selection subroutine
176
177 pni                                ;print 9
178     movlw    0x10                    ;moves hex of 9 into the wroking register
179     movwf    portB                  ;prints the working register to portb
180     goto     get_count               ;goes back to button selection subroutine
181
182 pa                                  ;print a
183     movlw    0x08                    ;moves hex of a into the wroking register
184     movwf    portB                  ;prints the working register to portb
185     goto     get_count               ;goes back to button selection subroutine
186
187 pb                                  ;print b
188     movlw    0x03                    ;moves hex of b into the wroking register
189     movwf    portB                  ;prints the working register to portb
190     goto     get_count               ;goes back to button selection subroutine
191
192 pc                                  ;print c
193     movlw    0x46                    ;moves hex of c into the wroking register
194     movwf    portB                  ;prints the working register to portb
195     goto     get_count               ;goes back to button selection subroutine
196
197 pd                                  ;print d
198     movlw    0x21                    ;moves hex of d into the wroking register
199     movwf    portB                  ;prints the working register to portb
200     goto     get_count               ;goes back to button selection subroutine
201

```

(Figure 22)

Figure 22 is again very similar to figure 21 and 20 as it is just more print commands.

```

201
202 pe                                ;print e
203     movlw    0x06                    ;moves hex of e into the wroking register
204     movwf    portB                  ;prints the working register to portb
205     goto     get_count               ;goes back to button selection subroutine
206
207 pf                                ;print f
208     movlw    0x0E                    ;moves hex of f into the wroking register
209     movwf    portB                  ;prints the working register to portb
210     goto     get_count               ;goes back to button selection subroutine
211

```

(Figure 23)

This is the last few lines of my code and as you can see it is just print commands which once completed go back to the button reading subroutine.

To conclude section 3 part 4 I feel as if there is a better way to produce this code, I would try and use an array of some kind with a pointer that moves up and down. I feel as if you save a lot of space with the code it would also make it more practical and easier to understand what is going on. Whist looking in to this i found a blog post  8 helped me understand and if I was to do this again I would mostly use something similar to figure 24. Whist righting this section I have thought of a more efficient way of doing this code by using the decfsz command. I would use it by taking count and decrementing it until it reaches 0 where I would print however many times it went through ,the process.

For section 3 part 5 I would like to use an interrupt I have done some research into this using the information gathered for figure 25. I would make the  9 `press an interrupt`, this would ensure that whist being pressed it will activate the buzzer and when released would stop. The reason this is better than using a check command cycle is because it would happen almost instantly and I could

monitor the interrupt a lot better than just doing a button check every so often. However, with this knowledge I still cant get the buzzer to work as intended.

For all of this code I used the Microchip instruction set found in figure 26



[1]

Electronics Tutorials, "7-segment Display and Driving a 7-segment Display," Basic Electronics Tutorials, Feb. 16, 2018. <https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html>

[2]

Robbie Young "Screenshot of proteus"

[3]

"What is Radix?," www.computerhope.com.

<https://www.computerhope.com/jargon/r/radix.htm#:~:text=Updated%3A%2007%2F12%2F2017> (accessed Nov. 28, 2022).

[4]

Robbie Young "Screenshot of proteus"

[5]

Robbie Young "Screenshot of proteus"

[6]

Robbie Young "Screenshot of proteus"

[7]

Robbie Young "Screenshot of proteus"

[8]

Robbie Young "Screenshot of proteus"

[9]

Robbie Young "Screenshot of proteus"

[10]

Robbie Young "Screenshot of proteus"

[11]

Robbie Young "Screenshot of proteus"

[12]

Robbie Young "Screenshot of proteus"

[13]

Robbie Young "Screenshot of proteus"

[14]

Robbie Young "Screenshot of proteus"

[15]

Robbie Young "Screenshot of proteus"

[16]

Robbie Young "Screenshot of proteus"

[17]

Robbie Young "Screenshot of proteus"

[18]

Robbie Young "Screenshot of proteus"

[19]

Robbie Young "Screenshot of proteus"

[20]

Robbie Young "Screenshot of proteus"

[21]

Robbie Young "Screenshot of proteus"

[22]

Robbie Young "Screenshot of proteus"

[23]

Robbie Young "Screenshot of proteus"

[24]

"Pointer in Assembly??? | Microchip," www.microchip.com.

<https://www.microchip.com/forums/m237683.aspx> (accessed Nov. 28, 2022).

[25]

GRTCMore, "Interrupt Routine in Assembly Language Using 8-bit PIC," *Instructables*.

<https://www.instructables.com/Interrupt-Routine-in-Assembly-Language-Using-8-bit/> (accessed Nov. 28, 2022).

[26]

1997 Microchip Technology Inc. "Section 29. Instruction set"

<http://ww1.microchip.com/downloads/en/DeviceDoc/31029a.pdf> (accessed Nov. 28, 2022).

Word count without this is 2701 there are 1356 words below but is the code in word format

; FILE: count.asm

; AUTHOR: JMY

; DATE: 24th March 2005

; DESCRIPTION: Count to 255 at 1 second intervals and display on a seven

; segment display in binary.

; The pause subroutine takes apx. 0.2 sec.

; Pause is called 5 times

list p=16F84 ; select processor type

radix hex ; define radix as hexadecimal

; #include "p16f84.inc"

;

; cpu equates (memory map)

portB equ 0x06 ;port B address

trisb equ 0x06 ;trisB address in register bank 1

status equ 0x03 ;status register

RP0 equ 5 ;bit 5 of status register

RP1 equ 6 ;bit 6 of status register

count equ 0x0c ; counter 0 to 255

nCount equ 0x0d ; pause routine inner loop counter

mCount equ 0x0e ; pause routine outer loop counter

;------

org 0x000

start

bcf status,RP0 ; clear bank selection bits in the status reg

bcf status,RP1

bsf status,RP1

movlw 0x03

movwf trisa

bcf status,RP1

bsf status,RP0 ; select bank 1

movlw 0x00 ; load w with 0x00

movwf trisb ; load it into trisb to make portb all outputs

bcf status,RP0 ; reselect bank 0

```
clrf    portB        ; clear all lines low
clrf    count        ; initialize counter to 0
```

`get_count` ;`get_count` is used to check the button inputs

<i>btjsc</i>	<i>porta, 0</i>	<i>;checks button one skips the line under if it is clear</i>
--------------	-----------------	---

goto but1 ;if porta pin 0 is high then got to but1

```
btjsc    porta, 1    ;checks button two skips the line under if it is clear
```

goto but2 ;if porta pin 1 is high then got to but1

goto get_count ;goes back to the start of the cycle

```
but1 ;subroutine but1 used decrease the seven segment display
```

```
movf    count, w           ;moves count into the working register
```

<i>decfsz</i>	<i>count, w</i>	<i>;decreases count by one skips if it is 0</i>
---------------	-----------------	---

gotonumber ;goes to the subrutine number to find the value of count

```
gotopzo           ;count is 0 so goto print 0
```

but2 ;subroutine but2 is used them button 2 is pressed and iwill

increment count by 1

```
movf    count, w           ;moves count into the working register
```

```
incf count, count      ;increment count by 1 and then save it back on its self
```

<i>btfs</i>	<i>count, 0x10</i>	<i>;checks bit number 5 of count</i>
-------------	--------------------	--------------------------------------

gotoreset ;if it is a one reset count

gotonumber ;if bit 5 is a 0 find the value of count

```
;reset subroutine used to reset count
```

```
movlw    0x0F                ;moves 00001111 into the working register
movwfm   count               ;saves it back over count
gotoget_count                ;goes back to the button check subroutine
```

```
number                                ;checks what number needs to be printed
movf     count, w              ;moves count into the working register
btfss    count, 0x08          ;checks bit 4 of count
goto     nse-zo               ;will be skipped if if bit four is set if it clear then goto nse-zo
btfss    count, 0x04          ;checks bit 3 of count
gotonei-a                                ;will be skipped if if bit four is set if it clear then goto nei-a
btfss    count, 0x02          ;checks bit number 2 of count
gotonc-d                                ;will be skipped if if bit four is set if it clear then goto nc-d
btfss    count, 0x01          ;checks bit number 1 of count
gotope                                ;will redirect the code to pe
gotopf                                ;will redirect the code to pf
```

```
nse-zo                                ;nse-zo (number seven to zero)
movf     count, w              ;moves count into the working register
btfss    count, 0x04          ;checks bit number 3 in count
gotonth-zo                            ;if bit number 3 is 0 then goto nth-zo
btfss    count, 0x02          ;checks bit 2 of count
gotonfi-fo                            ;if bit 2 is 0 then goto nfi-fo
btfss    count, 0x01          ;checks bit 1 of count
gotopsi                                ;if 0 then goto psi
gotopse                                ;if one then goto pse
```

<i>nth-zo</i>			<i>;nth-zo (numbers three to zero)</i>
<i>movf</i>	<i>count, w</i>	<i>;moves count to the working register</i>	
<i>btfss</i>	<i>count, 0x02</i>	<i>;checks bit 2 of count</i>	
<i>gotonzo-on</i>		<i>;if bit 2 is 0 then goto nzo-on</i>	
<i>btfss</i>	<i>count, 0x01</i>	<i>;checks bit 1</i>	
<i>gotoptw</i>		<i>;if bit one is 0 then goto print two</i>	
<i>gotopth</i>		<i>;if bit 1 is 1 then goto print three</i>	
<i>nzo-on</i>			<i>;nzo-on (numbers zero to one)</i>
<i>movf</i>	<i>count, w</i>	<i>;moves count to the working register</i>	
<i>btfss</i>	<i>count, 0x01</i>	<i>;checks bit 1 of count</i>	
<i>gotopze</i>		<i>;if 0 thrn goto print zero</i>	
<i>gotopon</i>		<i>;if 1 then goto print one</i>	
<i>nfi-fo</i>			<i>;nfi-fo (numbers five to four)</i>
<i>movf</i>	<i>count, w</i>	<i>;moves count to the wroking register</i>	
<i>btfss</i>	<i>count, 0x01</i>	<i>;checks bit 1 of count</i>	
<i>gotopfo</i>		<i>;if bit one is a 0 then goto print four</i>	
<i>gotopfi</i>		<i>;if bit one is 1 then goto print five</i>	
<i>nc-d</i>			<i>;nc-d (numbers c to d)</i>
<i>movf</i>	<i>count, w</i>	<i>;moves count to the working register</i>	
<i>btfss</i>	<i>count, 0x01</i>	<i>;checks bit 1 of count</i>	
<i>gotopc</i>		<i>;if 0 then goto print c</i>	
<i>gotopd</i>		<i>;if 1 then goto print d</i>	

```

nei-a                                ;nei-a (numbers eight to a)

    movf    count, w                ;moves count to the working register

    btfss   count, 0x02             ;checks bit 2

    goto nei-ni                    ;if 0 then goto eight to nine

    btfss   count, 0x01             ;if 1 cheack bit number 1

    gotopa                                     ;if 0 goto print a

    gotopb                                     ;if 1 goto print b


nei-ni                                ;nei-ni (numbers eight to nine)

    movf    count, w                ;moves count to the working register

    btfss   count, 0x01             ;checks bit number 1

    gotopei                            ;if 0 then goto print eight

    gotopni                            ;if 1 goto print nine


pze                                    ;print 0

    movlw   0x40                    ;moves hex of 0 into the wroking register

    movwf   portB                   ;prints the working register to portb

    goto get_count                  ;goes back to button selection subroutine


pon                                    ;print 1

    movlw   0x79                    ;moves hex of 1 into the wroking register

    movwf   portB                   ;prints the working register to portb

    goto get_count                  ;goes back to button selection subroutine


ptw                                    ;print 2

    movlw   0x24                    ;moves hex of 2 into the wroking register

```

```
movwf    portB           ;prints the working register to portb  
gotoget_count           ;goes back to button selection subroutine
```

```
pth                               ;print 3  
  
movlw    0x30             ;moves hex of 3 into the wroking register  
movwf    portB           ;prints the working register to portb  
gotoget_count           ;goes back to button selection subroutine
```

```
pfo                               ;print 4  
  
movlw    0x19             ;moves hex of 4 into the wroking register  
movwf    portB           ;prints the working register to portb  
gotoget_count           ;goes back to button selection subroutine
```

```
pfi                               ;print 5  
  
movlw    0x12             ;moves hex of 5 into the wroking register  
movwf    portB           ;prints the working register to portb  
gotoget_count           ;goes back to button selection subroutine
```

```
psi                               ;print 6  
  
movlw    0x02             ;moves hex of 6 into the wroking register  
movwf    portB           ;prints the working register to portb  
gotoget_count           ;goes back to button selection subroutine
```

```
pse                               ;print 7  
  
movlw    0x78             ;moves hex of 7 into the wroking register  
movwf    portB           ;prints the working register to portb
```

gotoget_count ;goes back to button selection subroutine

pei ;print 8

movlw 0x00 ;moves hex of 8 into the wroking register

movwf portB ;prints the working register to portb

gotoget_count ;goes back to button selection subroutine

pni ;print 9

movlw 0x10 ;moves hex of 9 into the wroking register

movwf portB ;prints the working register to portb

gotoget_count ;goes back to button selection subroutine

pa ;print a

movlw 0x08 ;moves hex of a into the wroking register

movwf portB ;prints the working register to portb

gotoget_count ;goes back to button selection subroutine

pb ;print b

movlw 0x03 ;moves hex of b into the wroking register

movwf portB ;prints the working register to portb

gotoget_count ;goes back to button selection subroutine

pc ;print c

movlw 0x46 ;moves hex of c into the wroking register

movwf portB ;prints the working register to portb

gotoget_count ;goes back to button selection subroutine

```
pd                                ;print d
    movlw    0x21                ;moves hex of d into the wroking register
    movwf    portB              ;prints the working register to portb
    gotoget_count               ;goes back to button selection subroutine

pe                                ;print e
    movlw    0x06                ;moves hex of e into the wroking register
    movwf    portB              ;prints the working register to portb
    gotoget_count               ;goes back to button selection subroutine

pf                                ;print f
    movlw    0x0E                ;moves hex of f into the wroking register
    movwf    portB              ;prints the working register to portb
    gotoget_count               ;goes back to button selection subroutine
end
```

FINAL GRADE

GENERAL COMMENTS

Instructor

47 /100

PAGE 1



Comment 1

Where are sections 1 & 2?



Use of 1st Person Voice

When writing technical reports, do not use first person voice. Instead, use the third person passive voice.

You should not use words like "I", "you", "we" etc..



Comment 2

This is just the code you were given to start. "Count.asm" from Moodle

PAGE 2

PAGE 3

PAGE 4



Comment 3

Where do these values come from? Why does 0x12 produce the character 5 on the 7-segment display?

PAGE 5



Comment 4

This code, while functional, is not reusable. It would have been far better to use the INDR and FSR registers.

PAGE 6

PAGE 7



Comment 5

RA0 and RA1

PAGE 8



Comment 6

The inputs from the switches are active low i.e. they are high when the switch is not pressed and low when it is. This is the opposite way round from the way you described.

PAGE 9

PAGE 10



Comment 7

This seems a little over complicated. using the INDR and FSR registers would be a far better solution - it would create more simple reusable code

PAGE 11

PAGE 12



Comment 8

You haven't included figure 24, or figure 25



Comment 9

The PIC only supports interrupt on pin change for some IO on Port B, not on Port A

PAGE 13



Comment 10

no figure 26 in report



Comment 11

You cannot reference yourself like this - your screenshot of proteus is not published.

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

GRADING FORM: SMU GENERIC_SMOSEY

ROBBIE YOUNG

47

ENGAGEMENT WITH LITERATURE

including reading, referencing, academic conventions and academic honesty



No description of circuit given. Limited information about the 7-segment display given

8

KNOWLEDGE & UNDERSTANDING

Knowledge of the basic underlying concepts and principles of a subject.



Limited knowledge and understanding demonstrated.

8

COGNITIVE & INTELLECTUAL

Evaluate underlying concepts and principles of a subject and interpret qualitative and quantitative data in order to develop lines of argument and make sound judgements.



Some interesting solutions presented, but your code is not tight and only some sections are reusable. No flowcharts or pseudocode given to explain your assembly code.

11

PRACTICAL APPLICATION SKILLS

Different approaches to solving problems in particular contexts.



Minimum number of tasks completed. Some errors in the assembly e.g. switch tests for wrong logic level on pins.

11

TRANSFERABLE SKILLS

Communicate the results of their study/work accurately and reliably, and with structured and coherent arguments; the qualities needed for employment requiring the exercise of some personal responsibility within a structured and managed environment.



Structure could be far better. You should be using the 3rd person voice for formal academic writing. References missing information. Please see my comments in the text

9

