EECE.3170: Microprocessor Systems Design I
Spring 2016

Exam 3 Solution

1. (20 points, 5 points per part) ***Multiple choice***

For each of the multiple choice questions below, clearly indicate your response by circling or underlining the single choice you think best answers the question.

a. You are running a PIC16F1829 program that uses two I/O ports, Port A and Port C. If TRISA = 0x0F and TRISC = 0x19, how many I/O pins, in total, are configured as inputs?

i. 4

ii. ***7 (TRISx pin = 1 → I/O pin is input. 0x0F contains four 1s, 0x19 contains 3)***

iii. 9

iv. 15

v. 40

b. Under what conditions will the following code jump to the label L1?

```
   movf    x, W
   subwf   y, W
   btfsc   STATUS, Z
   btfss   STATUS, C
   goto    L1
END:
```

Both choices (i) and (v) were accepted because while this code technically tests the condition in choice (v), the carry flag will never determine the outcome of the jump—if x == y and the btfsc instruction doesn't skip, C must be 1, which means the btfss instruction will always skip.

i. ***$x \neq y$***

ii. x = y or C = 1

iii. x = y or C = 0

iv. $x \neq y$ or C = 1

v. ***$x \neq y$ or C = 0***

1 (continued)

c. You are given the following short PIC16F1829 assembly function:

```
F: movf    PORTC, W
   andlw   B'00000001'
   addwf   PCL, F
   retlw   B'11110000'
   retlw   B'00111100'
   retlw   B'00001111'
   retlw   B'11111111'
```

Which of the following PORTC values will cause this function to return `B'00001111'`?

i. `PORTC = 0xF0`

ii. `PORTC = 0x19`

iii. `PORTC = 0x86`

iv. `PORTC = 0x3F`

v. ***<u>None of the above</u>*** ***(In order for the function to return that value, the andlw instruction would have to produce the value 2, which would then be added to PCL to jump an extra 2 instructions ahead (the default behavior is to go to the next instruction). Since the andlw instruction will only produce the value 0 or 1, it's impossible for this function to execute the third or fourth retlw instructions.)***

d. Circle one (or more) of the choices below that you feel best "answers" this "question."

i. "Thanks for the free points."

ii. "I don't REALLY have to answer the last three questions, do I?"

iii. "This exam is the best final I've taken all day."

iv. None of the above.

<u>All of the above are "correct."</u>

2. (15 points) ***General microcontroller programming***
a. (4 points) You are given the following delay loop:

```
Ten_1
    decfsz    COUNTL,F       ; Inner loop
    goto      Ten_1
    decfsz    COUNTH,F       ; Outer loop
    goto      Ten_1
    return
```

If COUNTL is initially 100, COUNTH is initially 10, the clock frequency is 500 kHz, and each instruction takes 4 clock cycles, how long does the whole delay loop take? **Show your work for full credit.**

Solution: *The total amount of delay is the product of four factors: the clock cycle time, the number of cycles per instruction, the number of instructions per loop iteration, and the number of iterations. To determine each of these:*

- *Clock cycle time = 1 / (clock frequency) = 1 / 500 kHz = 2 x 10^{-6} sec = 2 µs*
- *Cycles per instruction is given as 4, which means each instruction takes 4 * 2 = 8 µs*
- *The number of instructions per iteration and number of loop iterations can be considered together to determine the total number of instructions:*
    - *For each decfsz/goto pair, the goto instruction is executed one fewer time than the decfsz because the goto is skipped the last time through the loop.*
    - *The number of iterations is based on the initial value of the variable being decremented:*
        - *For the outer loop, the decfsz executes 10 times and the goto executes 9.*
        - *The inner loop works differently on the first iteration than on all others:*
            - *1st time: decfsz executes 100 times, goto executes 99*
            - *2nd through 10th times: decfsz executes 256 times (since COUNTL is initially 0), goto executes 255.*
    - *The total number of instructions can be broken down as shown below, with the total instruction count for each instruction shown in red:*

```
Ten_1
    decfsz    COUNTL,F   100 + 9 * 256 = 2404
    goto      Ten_1      99 + 9 * 255 = 2394
    decfsz    COUNTH,F   10
    goto      Ten_1      9
    return               1
              TOTAL:     2404 + 2394 + 10 + 9 + 1 = 4818
```

The total delay is therefore (4818 instructions) * (8 µs per instruction) = 38544 µs = 38.544 ms.

2 (continued)

b. (3 points) In the blinking LED state machine program we covered in class, the following instructions isolate the lowest three bits of PORTD and copy them into W:

```
movf    PORTD, W
andlw   B'00000111'
```

Why might it be necessary to isolate the lowest three bits and mask out the upper five bits of PORTD?

Solution: *Because other devices could be connected to those upper five bits, and their state shouldn't affect the operation of the rest of the "blink" function.*

c. (4 points) Say your program uses a 10-bit analog-to-digital converter (ADC) with positive/negative reference voltages of 5 V and 0 V, respectively. If the ADC output is 256, approximately what voltage was input to the converter? **Show your work for full credit.**

Solution: *A ten-bit ADC will produce* $2^{10} = 1024$ *possible values, between 0 and* $2^{10} - 1 = 1023$*. In class, we discussed the formula: ADC value =* $(V / V_{REF}) * 1023$*—in other words, determine what the ratio of the input voltage to the positive reference voltage (assuming a negative reference voltage of 0), and multiply that by the maximum ADC output. We can manipulate that formula to determine the input voltage, V:*

$$(\text{ADC value}) / 1023 = V / V_{REF}$$
$$((\text{ADC value}) * V_{REF}) / 1023 = V$$

Plugging in the values that are given:

$$(256 * 5\text{ V}) / 1023 = 1.25122\text{ V} \approx 1.25\text{ V}$$

d. (4 points) Say you are writing a program using two interrupts. One interrupt is triggered when a timer overflows, while the other is triggered when a switch is pressed. Describe a situation in which you would want to prioritize the timer interrupt (and therefore handle the timer interrupt first if both types of interrupt occur simultaneously), and describe a situation in which you would want to prioritize the switch interrupt.

Solution: *If a precise interval between events is required, then the timer interrupt (which presumably determines that interval) should be prioritized.*

If timing is not as important and the button press is used to trigger an important event, then that interrupt should be prioritized.

3. (15 points) ***PIC C programming***

Complete the short function below by writing the appropriate line(s) of C code into each of the blank spaces. The purpose of each line is described in a comment to the right of the blank.

This interrupt service routine detects both switch and timer interrupts and works with the analog-to-digital converter (ADC). The ISR does the following:

- On a timer interrupt, start an analog-to-digital conversion.
- On a switch interrupt, check if the ADC is done. If the ADC is done, assign the lowest four bits of the result to the LEDs, making sure to only change the lowest four bits of Port C while leaving the upper four bits the same.

Assume the LEDs are wired to the lowest four bits of Port C (as on the board used in HW 8) the ADC result is right-justified, and that “SWITCH” and “DOWN” are appropriately defined.

```
void interrupt ISR(void) {

   if (IOCAF) {                        // SW1 was pressed
      IOCAF = 0;                       // Clear flag in software
      __delay_ms(5);                   // Delay for debouncing
      if (SWITCH == DOWN) {            // If switch still pressed

         if (GO == 0) {                //   check if ADC is done

            LATC = LATC & 0xF0;        // If done, assign lowest 4
            LATC = LATC | (ADRESL & 0x0F); // result bits to LEDs,
                                          // making sure to only
                                          // change correct bits

         }

      }
   }
   if (INTCONbits.T0IF) {              // Timer 0 interrupt

      INTCONbits.T0IF = 0;             // Clear flag in software

      GO = 1;                          // Start ADC
   }
}
```

4. (50 points, 25 points per part) ***PIC assembly programming***
For each of the following complex operations, write a sequence of PIC 16F1829 instructions—**not C code**—that performs an equivalent operation. **CHOOSE ANY TWO OF THE THREE PARTS** and fill in the space provided with appropriate code. **You may complete all three parts for up to 10 points of extra credit, but must clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**

Assume that 8-bit variables "TEMP" and "COUNT" have been defined for cases where you may need extra variables.

Finally, please note that you are not required to write comments describing each instruction. You may certainly do so if you feel comments will make your solution clearer to the instructor.

a. You are given two 8-bit variables, P and Q. Write a sequence of instructions that takes the bit sequence in P and stores its reverse in Q. For example:

- If P = 0xF0 = 11110000_2, Q = 00001111_2 = 0x0F
- If P = 0x15 = 00010101_2, Q = 10101000_2 = 0xA8

Your solution should not change P.

Solution: *Other answers may be acceptable. The general idea behind this solution is to shift each bit of P into the carry flag, then rotate that bit from the carry flag into Q. As long as these operations are performed in opposite directions, Q will hold the same bits in reverse order.*

You could argue that this solution doesn't follow directions—it does, after all, "change" P—but saving the original value of P before starting the loop allows you to restore it when the loop is done, giving the appearance that P never changed.

```
        clrf      Q           ; Start with Q == 0
        movlw     8           ; Set loop counter to 8 since
        movwf     COUNT       ;   there are 8 bits in each register
        movf      P, W        ; Copy P into the working register so its
                              ;   original value can be restored at the end
L:      lsrf      P, F        ; Shift the least significant bit of P
                              ;   into the carry bit
        rlf       Q, F        ; Rotate the carry flag (the bit shifted out of P)
                              ;   into Q, moving the other bits to the left
        decfsz    COUNT, F    ; Decrement COUNT and return to the
        goto      L           ;   start of the loop if COUNT != 0
        movwf     P           ; Restore the original value of P
                              ;   once the loop is done
```

4 (continued)

Remember, you can assume that 8-bit variables "TEMP" and "COUNT" have been defined for cases where you may need extra variables.

b. You have a 16-bit variable, X, and an 8-bit variable, P. You can access individual bytes within X—the low byte, XL, holds bit positions 0 to 7, and the high byte, XH, holds bit positions 8 to 15.

   Write a sequence of instructions that sets P equal to the highest bit position within X that contains a 1. (This operation is similar to the x86 BSR instruction). If X = 0, P should be unchanged. For example:

   - If X = 0x61AB = $0\underline{1}10\ 0001\ 1010\ 1011_2$, P = 14 (underlined bit is "highest" 1)
   - If X = 0x0082 = $0000\ 0000\ \underline{1}000\ 0010_2$, P = 7

   Your solution should not change XL or XH.

<u>**Solution:**</u> *Other solutions may be acceptable. This solution does the following:*

- *Test each byte, starting with XH (since we're looking for highest 1). If both 0, leave P unchanged. Otherwise, move the byte to be tested into TEMP, and set P equal to its maximum possible value (15 if the byte is XH, 7 if the byte is XL).*
- *Using a bitmask that starts with a single 1 in the MSB (because, again, we're looking for the highest 1), test each bit of TEMP by ANDing the bitmask with TEMP.*
  - *If the result of the AND is non-zero, you've found the highest 1.*
  - *If the result of the AND is 0, shift the bitmask to the right (moving the 1 to the next lowest bit) and decrement P (decreasing the bit position).*
  - *Note that the "TestBits" loop doesn't need a variable to count iterations. If your code enters that loop, then at least one of the bytes is non-zero, and the goto instruction that jumps to "TestDone" will be executed when the first 1 is found.*

The code for this problem is written in full on the next page because (1) it required the detailed explanation above, and (2) the code itself turned out to be pretty long. My apologies to my Spring 2016 students!

<u>Solution (continued):</u>

```
        movlw       0x80        ; COUNT will be bitmask used to test register
        movwf       COUNT       ;   (either XH or XL); start with 1 in MSB
        movf        XH, W       ; Test to see if XH == 0; if not, highest
        btfsc       STATUS, Z   ;   non-zero bit is in XH
        goto        TestXL      ; Otherwise, test XL
        movwf       TEMP        ; Set TEMP = XH (XH is already in W)
        movlw       15          ; Set P = 15 to start, since bit position
        movwf       P           ;   will be between 15 and 8
        goto        TestBits    ; Jump to actual bit test
TestXL:
        movf        XL, W       ; Test to see if XL == 0; if not, exit sequence
        btfsc       STATUS, Z   ;   and don't change P at all
        goto        TestDone
        movwf       TEMP        ; Set TEMP = XL (XL is already in W)
        movlw       7           ; Set P = 7 to start, since bit position
        movwf       P           ;   will be between 7 and 0
TestBits:
        movf        TEMP, W     ; Set W = register to be tested
        andwf       COUNT, W    ; AND register with COUNT (bit mask)
        btfss       STATUS, Z   ; If result of AND != 0, tested bit must be 1 so
        goto        TestDone    ;    P = correct bit position and sequence is done
        decf        P, F        ; Decrement P and shift bit mask to
        lsrf        COUNT, F    ;    prepare to test next bit
        goto        TestBits
TestDone:                       ; End of sequence
```

4 (continued)

Remember, you can assume that 8-bit variables "TEMP" and "COUNT" have been defined for cases where you may need extra variables.

c. You are given two unsigned 16-bit values, X and Y. You can access individual bytes within each value—"X" contains bytes XH and XL (XL is the least-significant byte) and "Y" contains bytes YH and YL.

   Write a sequence of instructions that jumps to location "L1" if X is greater than <u>or</u> equal to Y. Your solution should not change any of the bytes of X or Y. Again, assume X and Y are unsigned (i.e., non-negative).

<u>Solution:</u> *Other solutions may be acceptable. The key points are:*

- *Performing a 16-bit subtraction in which the result ends up in the working register, so neither X nor Y is changed.*
- *Evaluating the carry flag to determine whether a borrow was used. If a borrow was <u>not</u> used (i.e., if C == 1), then X is greater than or equal to Y and the jump should be taken.*

```
movf     YL, W           ; W = YL
subwf    XL, W           ; W = XL – YL
movf     YH, W           ; W = YH
subwfb   XH, W           ; W = XH – YH – (~C) (account for borrow in lo byte)
btfsc    STATUS, C       ; If C == 1 (borrow occurred), then X >= Y
goto     L1              ;   and program should jump to L1
```