16.317: Microprocessor Systems Design I

Spring 2015

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. We discussed the following delay loop in class:

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

How many times will the first instruction in this loop (`decfsz COUNTL,F`) execute if COUNTH is initially equal to 0x02 and COUNTL is initially equal to 0x01? (All answers given below are in base 10, unless otherwise noted.)

i. 1

ii. 101

iii. 201

iv. 257 (which is 0x101 in hexadecimal)

v. 258

1 (continued)

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

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

i. $x \neq y$

ii. $x = y$ and $C = 1$

iii. x = y and C = 0

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

v. $x \neq y$ and $C = 0$

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

```
F: movf    PORTC, W
   andlw   B'00000010'
   addwf   PCL, F
   retlw   B'11110000'
   retlw   B'00111100'
   retlw   B'00001111'
   retlw   B'11111111'
```

If `PORTC = 0xC3`, what value is in the working register when this function returns?

i. `B'00000001'`

ii. B'00001111'

iii. `B'00111100'`

iv. `B'11110000'`

v. `B'11111111'`

1 (continued)

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.

All of the above answers are "correct."

2. (12 points) ***General microcontroller programming***

a. (3 points) Explain how an interrupt service routine can prioritize one interrupt over another.

Solution: *Priority is implicitly created based on the order in which the ISR evaluates the interrupt flags—the first one tested will essentially be treated as the highest priority interrupt.*

b. (3 points) What are the different factors that determine the total amount of delay in an instruction count-based delay loop?

Solution: *The system clock cycle time, the number of clock cycles per instruction, and the number of instructions executed in the loop. You could also consider the initial values in the delay counter, which determines the number of instructions executed in the loop.*

c. (3 points) Given a 10-bit result from an analog to digital converter, as in the PIC 16F1829, explain a case in which you might prefer a left-justified result (upper 8 bits) over a right-justified result (lower 8 bits).

Solution: *Using a left-justified result essentially ignores the lowest two bits of the result, meaning that the result does not take small changes in voltage into account. You would likely prefer this type of result for any voltage measurement in which the voltage is human-controlled—for example, if the voltage is based on the position of a potentiometer—because people typically aren't capable of making small enough changes that you'd need the lowest bits of the ADC result.*

d. (3 points) Explain why button presses cannot accurately be detected by simply checking if the switch produces a low voltage.

Solution: *First, note that I determined this question was too vague and gave full credit to everyone who attempted it. Had I asked why presses couldn't accurately be counted, I think it would have been clearer. The solution below answers that question:*

Detecting and counting button presses accurately requires you to differentiate the point at which the button is pressed from points at which the button is simply being held down without having been released. The only way to detect the time when the button is pressed is to detect a falling edge (a high to low transition) on the button. Checking if the switch produces a low voltage simply allows you to recognize that the button is currently pressed and was pressed at some point before it was sampled.

3. (18 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 works with the analog-to-digital converter, as well as a global variable, `var`, which determines what is written to the LEDs. The ISR does the following:

- On a switch interrupt, start an analog-to-digital conversion.
- On a timer interrupt, evaluate the value in ADRESH and change the LEDs as follows:
  - If the value in ADRESH is in the upper half of possible values (which you can test by checking if any of the top 4 bits are set *(actually only the MSB—see below)*), increment the value on the LEDs. You must account for the rollover case when the LEDs go from 1111 to 0000.
  - Otherwise, decrement the value on the LEDs, accounting for the 0000 to 1111 case.

Assume the LEDs are wired to the lowest four bits of Port C, as on the development board used in HW 6, and that "SWITCH" and "DOWN" are appropriately defined.

NOTE: *The description above about detecting if ADRESH is in the upper half of possible values is incorrect. Any 8-bit register has 256 possible values in the range 0-255. The upper half of that range starts at 128. Therefore, all values in the upper half of the range have bit 7 set to 1.*

```
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
          GO = 1;                         //   start ADC
        }
    }
    if (INTCONbits.T0IF) {                // Timer 0 interrupt
        INTCONbits.T0IF = 0;              // Clear flag in software

        if (ADRESH & 0x80) {              // If A/D result is in
          LATC++;                         //  upper half of possible
          if (LATC == 0x10)               //  results (any of upper
            LATC = 0;                     //  four bits are set)
                                          //  increment LED value
                                          //  Must account for
                                          //    1111 → 0000 case
        }
        else {                            // Otherwise, decrement
          LATC--;                         //  LED value. Must
          if (LATC == 0xFF)               //  account for
            LATC = 0x0F;                  //     0000 → 1111 case
        }
    }
}
```

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 are certainly welcome to do so if you feel it will make your solution clearer to the instructor.

a. 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 less than 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).

Solution: *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*

```
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)
btfss   STATUS, C       ; If C == 0 (borrow occurred), then X < Y
goto    L1              ;   and program should jump to L1
```

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. Given two 8-bit variables, X and N, flip the lowest N bits of X (change 0 → 1, 1 → 0) while leaving the other bits unchanged. For example:

- If X = 0x00 and N = 0x02, flip the lowest two bits—bits 0 and 1.
  - X = 0x00 = 0000 0000_2 originally → X will change to 0000 $00\underline{11}_2$ = 0x03
- If X = 0x0C and N = 0x06, flip the lowest six bits—bits 0 through 5.
  - X = 0x0C = $0\underline{000\ 1100}_2$ originally → X will change to $0\underline{011\ 0011}_2$ = 0x33

Note that:

- Since X and N are not constants, you cannot use both values together in any PIC instruction (for example, btfsc X, N is not a valid instruction).
- Your code should not modify N.

Solution: *Other solutions may be acceptable. The key points are*

- *Using an XOR operation to flip the appropriate bit(s)*
- *Using the counter N to determine the number of bits to be flipped*

```
        movf    N, W            ; COUNT = N
        movwf   COUNT
        movlw   0x01            ; Initial bit mask for flipping—will start with 1
                                ;    in lowest bit position
L:      xorwf   X, F            ; Flip appropriate bit of x
        movwf   TEMP            ; TEMP = bit mask (1 << # iterations)
        lslf    TEMP, W         ; W = shifted bit mask (move everything to
                                ;  left, shifting 0 into lowest position)
        decfsz  COUNT, F        ; Decrement loop counter and return to
        goto    L               ;   start of loop if result is not 0
```

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 16-bit values, X and Y, and an 8 bit value, ZL. 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.

Perform a 16-bit left rotate without carry: X = Y rotated by ZL. (Note that, because the rotate amount is not greater than 15, a single byte is sufficient to hold that value.) Do not change Y or ZL when performing this operation.

At each step, note that the bit shifted out of the most significant bit should be shifted into the least significant bit. For example, if Y = 0xFF00 = 1111 1111 0000 0000_2 and ZL = 1, then the final result should have X = 1111 1110 0000 0001_2.

Solution:

```
        movf    YL, W       ; Copy YL to XL
        movwf   XL
        movf    YH, W       ; Copy YH to XH
        movwf   XH
        movf    ZL, W       ; Copy ZL to COUNT
        movwf   COUNT
L:      bcf     STATUS, C   ; Carry = 0
        btfsc   XH, 7       ; If MSB (bit to be rotated) is 1
        bsf     STATUS, C   ;   set carry to 1 before rotating
        rlf     XL, F       ; Rotate upper byte (C = bit to be shifted into XL)
        rlf     XH, F       ; Rotate lower byte
        decfsz  COUNT, F    ; Decrement loop counter and return to start
        goto    L           ;    of loop if there are more iterations.
```