16.317: Microprocessor Systems Design I

Fall 2014

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 0x01 and COUNTL is initially equal to 0x09? (All answers given below are in base 10, unless otherwise noted.)

i. ***9***

ii. 10

iii. 109

iv. 264 (which is 0x108 in hexadecimal)

v. 265

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

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

i. $x \neq y$

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

iii. $x = y$ and $C = 0$

iv. <u>x ≠ y and C = 1</u>

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

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

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

If `PORTC = 0xF6`, what value is in the working register when this function returns?

i. `B'00000001'`

ii. <u>B'00001111'</u>

iii. `B'00111100'`

iv. `B'11110000'`

v. `B'11111111'`

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. "It's about time we have a test that doesn't start at 8:00 AM."

iv. None of the above.

All of the above are "correct."

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

a. (3 points) Explain why interrupt flags need to be cleared in software.

Solution: *For most devices, if the interrupt flag (the bit that indicates that an interrupt has occurred) is not explicitly cleared, that device will continue to signal that it has triggered an interrupt. Therefore, in order to avoid repeated, invalid interrupts, the interrupt service routine must clear the flag bit for the device that triggered the interrupt.*

b. (3 points) On a microcontroller with multiple devices that can trigger interrupts, why should the global interrupt enable be set after all individual device interrupts are enabled? (Recall that the global interrupt enable is a bit that must be set for any interrupts to be allowed.)

Solution: *Once the global interrupt enable is set, any interrupt can occur. That bit should therefore be set last to ensure that all devices are properly set up before any interrupts from any devices are allowed to occur.*

c. (3 points) Explain why an interrupt-based delay loop is usually preferable to an instruction count-based delay loop.

Solution: *An instruction count-based delay loop forces the processor to execute instructions that essentially do nothing but wait. An interrupt-based delay loop frees the processor to execute other useful code while it is waiting for that interrupt to occur.*

d. (3 points) Explain why switches typically need to be debounced before their state can be accurately determined.

Solution: *Switches rarely transition smoothly from high to low voltage—noise in the signal can potentially signal that the switch has changed state when it hasn't actually done so. Debouncing essentially removes the noise to ensure that the state of the switch can be accurately determined.*

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 and change the state of `var`.
  - If `var` is 1, change it to 0; if `var` is 0, change it to 1.
- On a timer interrupt, display four bits from the ADC result on the LEDs. Note that your code should only change the lowest four bits of Port C—leave the upper bits unchanged.
  - If `var` is 1, display the upper four bits from ADRESH on the LEDs.
  - If `var` is 0, display the lower four bits from ADRESH on the LEDs.

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.

Students were responsible for bold, underlined, italicized code; other solutions may be valid.

```
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
          var = var ^ 1;                //   and change "var"
        }
    }
    if (INTCONbits.T0IF) {              // Timer 0 interrupt

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

        if (var == 1) {                 // Case to show upper four
          LATC = LATC & 0xF0;           //  bits of ADRESH on LEDs
          LATC = LATC | (ADRESH >> 4);
        }
        else {                          // Case to show lower four
          LATC = LATC & 0XF0;           //  bits of ADRESH on LEDs
          LATC = LATC | (ADRESH & 0x0F);
        }
    }
}
```

4. (50 points, 25 points per part) ***PIC assembly programming***

a. You have a 16-bit value, X, and an 8-bit value, P, specifying a bit position in X ($0 \leq P \leq 15$). You can access individual bytes within X—the low byte, XL, holds bit positions 0 to 7 (0 being the least significant), and the high byte, XH, holds bit positions 8 to 15.

Write code that will jump to location "L1" if bit P within the value X is set to 1. Do not change XH, XL, or P in your solution.

For example, if X = 0x0FF0 (XH = 0x0F, XL = 0xF0), your code should jump to L1 if P is between 4 and 11. This case is simply an example—do not assume X is always 0x0FF0.

Solution: *Other solutions may be possible; the key points of the solution are:*

- *Determine which byte to check (if $0 \leq P \leq 7$, you're looking at XL, otherwise, it's XH).*
- *If using XH, the bit position to test is P-8, since bits 8-15 of "X" are bits 0-7 of XH.*
- *Since P is in a register, you can't use a bit test instruction to look at the bit, so you need something that shifts value to be tested until you can look at the appropriate bit.*
    - *This solution copies XH or XL into TEMP and the position (P or P-8) into COUNT, so that general code can be used for both cases after that point.*

```
        movlw   8               ; W = 8
        subwf   P, W            ; W = P – 8 (subwf is used to compare W to 8)
        btfsc   STATUS, Z       ; P is less than 8 if Z = 0 and C = 0
        goto    testXH          ;   If either of those conditions are false, P >= 8,
        btfsc   STATUS, C       ;   so test XH
        goto    testXH          ; Otherwise, test XL (starting with next instruction)
        movf    XL, W           ; W = XL
        movwf   TEMP            ; TEMP = W = XL
        movf    P, W            ; W = P
        movwf   COUNT           ; COUNT = W = P
        goto    bitTest         ; Goto bit testing code
testXH:
        movwf   COUNT           ; Since W is already P–8, move that value to COUNT
        movf    XH, W           ; W = XH
        movwf   TEMP            ; TEMP = W = XH
bitTest:
        movf    COUNT, F        ; Check for COUNT == 0
        btfsc   STATUS, Z       ; If it is, don't shift
        goto    testEnd
testLoop:
        lsrf    TEMP, F         ; Shift TEMP right, COUNT times
        decfsz  COUNT, F        ; End result—bit you want to test is in bit 0
        goto    testLoop
testEnd:
        btfsc   TEMP, 0         ; Look at desired bit—skip jump if it's 0
        goto    L1
```

4 (continued)

b. Given two unsigned 8-bit values, X and Y, produce the 16-bit product X * Y, storing the low byte in a register called LO and the high byte in a register called HI.

One possible solution mimics a hardware multiplier, which handles multiplication as follows:

- Copy Y into the low byte of the product.
- Create a loop that iterates once for each bit in Y. In each iteration:
  - Test the least significant bit of LO.
    - If that bit is 1, add X to HI.
    - If that bit is 0, don't change HI.
  - After that, shift the entire 16-bit product (HI and LO) to the right by one bit. Ensure the least significant bit of HI shifts into the most significant bit of LO.
- Once the loop is done, the HI/LO registers together hold the final product.

Your solution should not change X or Y.

Solution: *Again, other solutions may be valid. The key points are outlined above.*

```
        movf    Y, W            ; W = Y
        movwf   LO              ; LO = W = Y
        movlw   8               ; W = 8
        movwf   COUNT           ; COUNT = W = 8
        clrf    HI              ; Initialize HI to 0 (which isn't described above)
        movf    X, W            ; Set W = X for addition
multLoop:
        btfsc   LO, 0           ; If lowest bit of LO is 0
        addwf   HI, F           ;   skip this add (HI = HI + X)
        lsrf    HI, F           ; Shift high byte to right by 1 (could also use asrf)
                                ;   since problem doesn't say if values are signed
        rrf     LO, F           ; Shift low byte to right by 1, shifting in bit from HI
        decfsz  COUNT, F        ; Decrement loop counter; if 0, multiplication is done
        goto    multLoop        ; Return to start of loop
```

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. Assume you have 8-bit values X, Y, and DIFF, where DIFF = X – Y. Your code will set a separate variable, OVFL, to 1 if overflow occurred in this subtraction and 0 otherwise.

   One possible algorithm uses the fact that overflow in subtraction only occurs when the numbers have different signs. If X is positive and Y is negative, DIFF should be positive. If X is negative and Y is positive, DIFF should be negative. Therefore, to check for overflow:

   - Check the signs of X and Y. If they match, overflow can't occur (set OVFL = 0 and skip the remaining steps).
   - If the signs of X and Y are different, check the signs of X and DIFF.
     - If those signs match, there is no overflow (set OVFL = 0).
     - If those signs do not match, overflow occurred (set OVFL = 1).

   Your solution should not change X, Y, or DIFF.

Solution: *Again, other solutions may be valid. Key points are outlined above.*

```
        clrf    OVFL        ; Clear OVFL
        movf    X, W        ; Compare signs of X and Y
        xorwf   Y, W
        andlw   0x80        ; Result will be non-zero if signs different
        btfsc   STATUS, Z   ; If result non-zero, check for overflow
        goto    ovchkdone   ; Otherwise, no overflow
        movf    X, W        ; Sign of X should match
        xorwf   DIFF, W     ;   sign of result if no overflow
        andlw   0x80
        btfss   STATUS, Z   ; If result non-zero,
        bsf     OVFL, 0     ;  overflow occurred
ovchkdone:                  ; End of sequence
```