16.317: Microprocessor Systems Design I

Fall 2014

Exam 3
December 15, 2014

Name: _____ **ID #:** _____

For this exam, you may use a calculator and one 8.5" x 11" double-sided page of notes. All other electronic devices (e.g., cellular phones, laptops, PDAs) are prohibited. If you have a cellular phone, please turn it off prior to the start of the exam to avoid distracting other students.

The exam contains 4 questions for a total of 100 points. Please answer the questions in the spaces provided. If you need additional space, use the back of the page on which the question is written and clearly indicate that you have done so.

Please note that Question 4 has three parts, but you are only required to complete two of the three parts. You may complete all three parts for up to 10 points of extra credit. If you do so, **please clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**

You will be provided with two pages (1 double-sided sheet) of reference material for the exam that contain the PIC16F1829 instruction set. You do not have to submit this sheet when you turn in your exam.

You will have three hours to complete this exam.

Q1: Multiple choice	/ 20
Q2: General microcontroller programming	/ 12
Q3: PIC C programming	/ 18
Q4: PIC assembly programming	/ 50
TOTAL SCORE	/ 100
EXTRA CREDIT	/ 10

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. $x \neq y$ and C = 1

v. $x \neq y$ and 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'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. `B'00001111'`

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.

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

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

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.)

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

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

```
void interrupt ISR(void) {

    if (_____________________________) {   // SW1 was pressed

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

          ___________________________________ //   start ADC

          ___________________________________ //   and change "var"

        }
    }
    if (_____________________________) {    // Timer 0 interrupt

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

        if (_____________________) {       // Case to show upper four
                                           //  bits of ADRESH on LEDs
                                           //  (FILL SPACE TO LEFT
                                           //  WITH NECESSARY LINE(S)
                                           //  OF CODE
        }
        else {                             // Case to show lower four
                                           //  bits of ADRESH on LEDs
                                           //  (FILL SPACE TO LEFT
                                           //  WITH NECESSARY LINE(S)
                                           //  OF CODE
        }
    }
}
```

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 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.

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 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.

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.