

# 16.317: Microprocessor Systems Design I

Spring 2014

Exam 3

May 6, 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: Reading PIC assembly	/ 16
Q3: PIC C programming	/ 24
Q4: PIC assembly programming	/ 40
<b>TOTAL SCORE</b>	<b>/ 100</b>
<b>EXTRA CREDIT</b>	<b>/ 10</b>

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. Which of the following operations **can** be done using only a single PIC16F1829 instruction?

- A. Two's complement negation of an 8-bit value (in other words,  $X = -X$ )
- B. Subtract a constant value from the working register, W
- C. One's complement (in other words, flip all bits) of the working register, W
- D. Rotate an 8-bit value to the right by one bit without rotating through the carry bit

- i. Only A
- ii. Only B
- iii. A and D
- iv. B and C
- v. All four operations (A, B, C, and D)

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

```
    btfsc    STATUS, C
    goto    END
    movlw   0x34
    subwf   x, F
    btfss   STATUS, Z
    goto    L1
```

END:

- i.  $C = 0$
- ii.  $C = 0$  and  $x \neq 0x34$
- iii.  $C = 0$  and  $x = 0x34$
- iv.  $C = 1$
- v.  $C = 1$  and  $x \neq 0x34$

1 (continued)

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

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

If  $PORTC = 0x0F$ , what value is in the working register when this function returns?

- i. B'00000010'
- 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. (16 points) **Reading PIC assembly**

Show the result of each PIC 16F1829 instruction in the sequences below. Be sure to show not only the state of updated registers, but also the carry (C) and zero (Z) bits.

a. cblock 0x70

    x  
    endc

    clrf    x

    incf    x, F

    movlw   0x0F

    xorwf   x, F

    swapf   x, F

    comf    x, W

    btfss   x, 1

    asrf    x, F

3. (24 points) **PIC C programming**

Complete each short function by writing the appropriate line 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.

a. (12 points)

Complete the interrupt service routine below so that if a timer interrupt has occurred, the LEDs will be updated to show the next value in the pattern stored in the array `st [ ]`, going back to the first value (0b0001) after showing the eighth (0b1001). If a switch interrupt has occurred, clear all LEDs and reset to the initial state. Assume the LEDs are wired to Port C, as on the development board used in HW 6, and that “SWITCH” and “DOWN” are appropriately defined.

Assume the use of the following global variables—`st [ ]` holds the list of values to be displayed on the LEDs, while `i` is the current index into that array. Assume `i` initially holds the value 0:

```
unsigned char st[8] = {0b0001, 0b0010, 0b0100, 0b0101,  
                      0b1010, 0b0100, 0b1000, 0b1001};
```

```
unsigned char i;
```

```
void interrupt ISR(void) {
```

```
    if (IOCAF) {                                     // SW1 was pressed  
  
        _____ // Clear flag in software  
        __delay_ms(5);                             // Delay for debouncing  
        if (SWITCH == DOWN) {                       // If switch still pressed  
  
            _____ // clear LEDs  
  
            i = 0;                                   // and reset i  
        }  
    }
```

```
    if (INTCONbits.T0IF) {                           // Timer 0 interrupt  
  
        _____ // Clear flag in software  
  
        _____ // Update LEDs to show  
                    // current st[] value  
  
        _____ // Increment i  
  
    if (_____ ) // If i exceeds max index  
        i = 0;                                       // reset i
```

```
    }  
}
```

3 (continued)

b. (12 points)

This function performs an analog to digital conversion and uses the least significant bits of the result, which are stored in ADRESL, to determine the operation of the program as follows:

- If the lowest two bits are 01, toggle the lowest LED, which is wired to the least significant bit (bit 0) of Port C.
  - That bit can be accessed either through the PORTC or LATC register; to access bit 0, use `PORTCbits.PORTC0` or `LATCbits.LATC0`
- If the lowest two bits are 10, toggle the second LED, which is wired to bit 1 of Port C.
- If the lowest two bits are 11, turn both the first and second LEDs on.

Assume the ADC is configured to produce a right-justified result, so the lowest bits of ADRESL are the least significant bits of the conversion result.

```
void read_adc(void) {
    unsigned char lobits;           // Variable to hold lowest 2
                                   // bits of ADC result
    __delay_us(5);                 // Wait for ADC cap to settle

    _____ // Start conversion
    while (GO) continue;           // Wait until conversion done

    _____ // lobits = lowest two bits
                                   // of ADC result

    if (lobits == 0b01) {           // In this case, toggle
                                   // lowest LED (FILL IN
                                   // SPACE TO LEFT WITH
                                   // APPROPRIATE CODE, WHICH
                                   // MAY USE MULTIPLE LINES)

    }
    else if (lobits == 0b10) {      // In this case, toggle
                                   // second LED (FILL IN
                                   // SPACE TO LEFT WITH
                                   // APPROPRIATE CODE, WHICH
                                   // MAY USE MULTIPLE LINES)

    }
    else if (lobits == 0b11) {      // In this case, turn first &
                                   // second LEDs on (FILL IN
                                   // SPACE TO LEFT WITH
                                   // APPROPRIATE CODE, WHICH
                                   // MAY USE MULTIPLE LINES)

    }
}
```

4. (40 points, 20 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 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 and Y are equal.

4 (continued)

b. Given two 8-bit variables, X and N, clear the lowest N bits of X (i.e., set the bits to 0). For example:

- If  $X = 0x0F$  and  $N = 0x02$ , clear the lowest two bits—bits 0 and 1.
  - $X = 0x0F = 0000\ 1111_2$  originally  $\rightarrow$  X will change to  $0000\ 1100_2 = 0x0C$
- If  $X = 0xFF$  and  $N = 0x06$ , clear the lowest six bits—bits 0 through 5.
  - $X = 0xFF = 1111\ 1111_2$  originally  $\rightarrow$  X will change to  $1100\ 0000_2 = 0xC0$

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. (*Original had typo saying you should not modify X.*)

4 (continued)

- 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 logical left shift:  $X = Y \ll ZL$ . (Note that, because the shift amount is no greater than 15, a single byte is sufficient to hold that value.) Do not change Y or ZL when performing this operation.