EECE.3170: Microprocessor Systems Design I

Spring 2016

Exam 2
March 30, 2016

Name: _____

Section (circle 1): **201** *(MWF 9-9:50)* **202** *(MWF 10-10:50)*

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, tablets) 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.**

Note also that your solutions to Question 4 will be short sequences of code, not subroutines. **You do not have to write any code to deal with the stack when solving Question 4.**

You will be provided with seven pages (4 double-sided sheets) of reference material for the exam: a list of the x86 instructions and condition codes we have covered thus far, a description of subroutine calling conventions, and a list of the PIC 16F1829 instructions we have covered thus far. You do not have to submit these pages when you turn in your exam.

You will have 50 minutes to complete this exam.

Q1: Multiple choice	/ 16
Q2: Reading PIC assembly	/ 16
Q3: Subroutines; HLL → assembly	/ 28
Q4: Conditional instructions	/ 40
TOTAL SCORE	/ 100
EXTRA CREDIT	/ 10

1. (16 points, 4 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.

Please note that all of the multiple choice questions deal with PIC 16F1829 instructions.

a. Which of the following instructions can always be used to complement the working register, W? (Note: remember that a complement operation simply flips all the bits of a register—it is not the same as negating a register.)

i. `comf x, W`

ii. `addlw -1`

iii. `sublw 0`

iv. `xorlw 0xFF`

v. `iorwf x, W`

b. Which of the following code snippets will jump to the label L if `x = 0x01`?

A.
```
btfss   x, 0
goto    L
```

B.
```
btfsc   x, 7
goto    L
```

C.
```
decfsz  x, F
goto    L
```

D.
```
incfsz  x, F
goto    L
```

i. Only A

ii. Only D

iii. A and B

iv. B and C

v. A, B, and C

1 (continued)

c. Which of the following instructions will set the carry bit (C) to 1 if the file register `x` is equal to 0xF0, the working register is equal to 0x20, and the carry bit is initially 0?

A. `subwf  x, F`

B. `lslf  x, F`

C. `rrf  x, F`

D. `addwf  x, F`

i. A and B

ii. B and C

iii. A, B, and C

iv. A, B, and D

v. A, B, C, and D

d. Which of the following instructions has the same effect as rotating a file register, `x`, by four bits, without including the carry?

i. `rrf  x, F`

ii. `rlf  x, F`

iii. `lslf  x, F`

iv. `asrf  x, F`

v. `swapf  x, F`

2. (16 points) ***Reading PIC assembly***

Show the result of each PIC 16F1829 instruction in the sequences below. Be sure to show the state of the carry (C) bit for any shift or rotate operations. You may assume C is initially 0.

a.
```
cblock 0x70
   x
endc
clrf    x



comf    x, W



sublw   0x10



incf    x, F



lslf    x, F



iorwf   x, F



xorlw   0x3C



addwf   x, W
```

3. (28 points) ***Subroutines; HLL → assembly***

The following questions (parts a-c) deal with the register and memory contents shown below. Note that:

- These values represent the state of some registers and memory locations immediately after the stack frame has been set up for the current function.
- The entire stack frame for the current function is shown, but there may be some additional data stored in the given address range—do not assume that the values shown in memory represent only the contents of the current stack frame.
- For parts a-c of this problem, you can assume that the stack frame for the current function starts at address 0x12580020.

EAX:	0x0000ABBA
EBX:	0x00001400
ECX:	0x09090909
EDX:	0xFF000000
ESI:	0x11340550
EDI:	0x11340590
ESP:	0x12580008
EBP:	0x12580014

Address	
0x12580000	0x00000005
0x12580004	0xCAE11600
0x12580008	0x09090909
0x1258000C	0x00001400
0x12580010	0x00000000
0x12580014	0x12580040
0x12580018	0x31700050
0x1258001C	0xFF000000
0x12580020	0x0000ABBA

a. (5 points) Assuming each argument uses 4 bytes, how many arguments does this function take? Explain your answer.

b. (4 points) Can you determine how many bytes of data the function called before the current function uses for local variables and saved registers? If so, explain how many bytes that function uses; if not, explain why not.

3 (continued)

c. (4 points) If we assume that the function uses the stack to save every register it overwrites, what registers does this function overwrite? Explain your answer.

d. (15 points) A partially completed x86 function is written below. Complete the function by writing the appropriate instructions in the blank spaces provided. The comments next to each blank or instruction describe the purpose of that instruction. Assume that the function takes two arguments (`v1` and `v2`, in that order) and contains a single local integer variable, `x`.

```
f  PROC                              ; Start of function f
   push    ebp                       ; Save ebp
   mov     ebp, esp                  ; Copy ebp to esp

   sub     esp, 4                    ; Create space on stack for x

   mov     ebx, DWORD PTR 8[ebp]     ; ebx = v1


   _____________________________     ; ebx = v1 + v2


   _____________________________     ; x = ebx = v1 + v2 (copy ebx
                                     ;    to memory location for x)
   sra     ebx, 2                    ; ebx = ebx >> 2 = x >> 2


   _____________________________     ; ebx = ebx + x = (x >> 4) + x


   _____________________________     ; Clear space allocated for
                                     ;     local variable
   pop     ebp                       ; Restore ebp

   _____________________________     ; Return from subroutine
f  ENDP
```

e. (40 points) ***Conditional instructions***

For each part of this problem, write a short x86 code sequence that performs the specified 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.**

Note also that your solutions to this question will be short sequences of code, not subroutines. **You do not have to write any code to deal with the stack when solving these problems.**

a. Implement the following conditional statement. You may assume that "X" and "Y" refer to 16-bit variables stored in memory, which can be directly accessed using those names (for example, `MOV AX, X` would move the contents of variable "X" to the register AX). Your solution should not modify AX or BX.

```
if (X < 10) {
   Y = X + AX;
}
else if (Y > BX) {
   X = Y - AX;
}
else {
   X = Y * 4;
   Y = X / 4;
}
```

4 (continued)

b. Implement the following loop. As in part (a), assume "X" and "Y" are 16-bit variables in memory that can be accessed by name. Assume that ARR is an array of 32-bit values, and that the loop does not go outside the bounds of the array. The starting address of this array is in the register SI when the loop starts—you can use that register to help you access values within the array. Your solution should not modify X, Y, or EAX.

```
for (i = X; i < Y; i = i + 3) {
   ARR[i+1] = ARR[i] + ARR[i+2];
   ARR[i] = ARR[i+2] – EAX;
}
```

4 (continued)

c. Implement the following loop. As in part (a), assume "X" and "Y" are 16-bit variables in memory that can be accessed by name. Recall that a while loop is a more general type of loop than the for loop seen in part (b)—a while loop simply repeats the loop body as long as the condition tested at the beginning of the loop is true. Your solution should not modify AX or BX.

```
while ((X < AX) || (Y > BX)) {
   X = X - Z;
   Y = Y + X;
}
```