EECE.3170: Microprocessor Systems Design I

Fall 2016

Exam 2
November 4, 2016

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, 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. If a file register, `x`, holds the value `0x03`, which of the following instruction sequences will set `x = 0xFD`?

i.
```
comf x, F
```

ii.
```
comf x, W
```

iii.
```
comf x, F
incf x, W
```

iv.
```
comf x, F
incf x, F
```

v. None of the above

b. Which of the following code snippets will not jump to the label L if `x = 0xF0`?

A.
```
btfss   x, 4
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 B

iii. A and C

iv. B and D

v. A, C, and D

1 (continued)

c. If a file register, `x`, is equal to `0x37`, and the working register, `W`, is equal to `0x91`, what values do those two registers hold after executing the instruction `swapf x, W`?

i. `x = 0x91, W = 0x37`

ii. `x = 0x73, W = 0x91`

iii. `x = 0x19, W = 0x91`

iv. `x = 0x37, W = 0x19`

v. `x = 0x37, W = 0x73`

d. If a file register, `z`, is equal to `0x03`, and the working register, `W`, is equal to `0x05`, which of the following instructions will set `W = 0x02`?

i. `sublw  0x07`

ii. `subwf  z, W`

iii. `addlw  0x02`

iv. `decf  z, F`

v. 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 the state of the carry (C) bit for any shift or rotate operations.

```
cblock 0x7F
   z
endc
clrf    z
```

```
decf    z, W
```

```
incf    z, F
```

```
andlw   0xC3
```

```
iorwf   z, F
```

```
asrf    z, F
```

```
btfsc   z, 7  (Show how this instruction determines if the next instruction is skipped)
```

```
bsf     z, 1
```

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

a. (8 points) The code below represents the body of a function (the code after the function prologue and before the function epilogue). Given this function body, explain what registers should be saved on the stack in the function prologue and why they should be saved:

```
mov  ecx, [ebp+8]
mov  esi, [ebp+12]
lea  esi, [esi+4*ecx]
imul DWORD PTR [esi]
```

b. (6 points) Explain why function arguments and variables are typically accessed using base pointer-relative addressing (for example, `-4[ebp]` or `[ebp+4]`) as opposed to stack pointer-relative addressing.

3 (continued)

c. (6 points) What is the minimum size for a stack frame in an x86 subroutine using the calling convention we discussed in class? (Recall that a calling convention determines how arguments are passed, values are returned, local variables are declared, and registers are saved in a function.) Explain your answer for full credit.

d. (8 points) You are writing a function that takes four signed 32-bit integer arguments. The function should return the average of the four values. Write only the body of the function—do not write any code that adds data to or removes data from the stack frame.

   For full credit, write the function body without using any local variables or any registers that would have to be saved on the stack in the function prologue.

4. (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 loop. You may assume "X" and "Y" are 16-bit variables in memory that can be accessed by name (for example, `MOV AX, X` would move the contents of variable "X" to the register `AX`). 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)

b. Implement the following conditional statement. As in part (a), assume that "X", "Y", and "Z" 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 (AX < X) {
   Z = AX + BX;
   if (Z > Y) {
      Y = X - AX;
      X = X + BX;
   }
   else if (Z < Y) {
      Y = X + AX;
      X = X - BX;
   }
}
```

4 (continued)

c. Implement the following loop. As in part (a), assume "X", "Y", and "Z" 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 (a)—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;
}
```