

16.317: Microprocessor Systems Design I

Spring 2015

Exam 2
April 1, 2015

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.

a. Which of the following statements about interrupts and exceptions are true?

- A. If an interrupt occurs, the program must use its next instruction to call the interrupt service routine (ISR).
- B. In a system where multiple devices share a single interrupt line, the interrupt service routine can poll all external devices to determine which device caused an interrupt.
- C. Only the instruction pointer and stack pointer are saved when an interrupt occurs on an x86 processor.
- D. An interrupt vector is the first instruction of an interrupt service routine.

- i. Only A
- ii. Only B
- iii. A and C
- iv. B and D
- v. All of the above (A, B, C, and D)

b. If a file register, x , is set to $0x01$, what is the result of the instruction `comf x, W`?

- i. $W = 0xFF$
- ii. $x = 0xFF$
- iii. $W = 0xFE$
- iv. $x = 0xFE$

1 (continued)

c. Which of the following instructions will set the carry bit (C) to 1 if the file register x is equal to $0xFF$, the working register is equal to $0x01$, and the carry bit is initially 0?

A. `addwf x, F`

B. `lslf x, F`

C. `rrf x, F`

D. `bsf x, 0`

i. Only A

ii. Only B

iii. A and B

iv. A, B, and C

v. A, B, C, and D

d. Which of the following instructions can always be used to flip (in other words, change 0s to 1s and 1s to 0s) the lower four bits of the working register, W, while leaving the upper four bits of the register unchanged?

i. `clrw`

ii. `sublw 0x0F`

iii. `iorlw 0xF0`

iv. `xorlw 0x0F`

v. `andlw 0xF0`

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.

a. cblock 0x70

 x

 endc

 movlw 0xC6

 addlw 0xFD

 movwf x

 swapf x, W

 iorwf x, W

 asrf x, F

 btfsc STATUS, C

 subwf x, F

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

The following questions 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.
- The last four instructions executed before entering the body of the current function (which are not the last four instructions executed to set up the stack frame) are:

```
push ebx
push ecx
push edx
call f
```

EAX: 0x0000ABBA
 EBX: 0x00001400
 ECX: 0x09090909
 EDX: 0xFF000000
 ESI: 0x11340550
 EDI: 0x11340590
 ESP: 0x40120154

Address	
0x40120150	0x00000005
0x40120154	0x0000000A
0x40120158	0xFFFF0000
0x4012015C	0x40120200
0x40120160	0x3170F000
0x40120164	0x00001400
0x40120168	0x09090909
0x4012016C	0xFF000000
0x40120170	0x192610AA

a. (5 points) What is the return address for this function? Explain your answer.

b. (4 points) What value does the base pointer (EBP) hold in this function? Explain your answer.

3 (continued)

c. (4 points) If we assume that each local variable uses four bytes, how many local variables are declared in this function? 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 one argument, `a1`, and contains one local integer variable, `v1`.

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

  _____                          ; Create space on stack for v1
                                          ;   and v2
  mov     eax, DWORD PTR 8[ebp]        ; eax = a1

  add     eax, 10                       ; eax = eax + 10 = a1 + 10

  _____                          ; v1 = eax = a1 + 10 (copy eax
                                          ;   to memory location for v1)

  _____                          ; v1 = v1 - 20 = a1 - 10

  _____                          ; eax = eax / v1
                                          ;   = (a1 + 10) / (a1 - 10)
                                          ; (use signed division; ignore
                                          ; remainder)

  mov     esp, ebp                    ; Clear space allocated for
                                          ;   local variable
  _____                          ; Restore ebp

  ret                                     ; Return from subroutine
f ENDP
```

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 conditional statement. You may 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).

```
if ((AX > 10) || (BX < 30) {
    X = AX + BX;
    if (X == Y)
        Z = 0;
    else
        Z = 1;
}
else
    Z = 2;
```

4 (continued)

- b. Implement the following loop. Assume that `ARR` is an array of twenty 16-bit values. 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.

```
for (i = 19; i > 0; i = i - 1) {  
    ARR[i] = ARR[i-1] - AX;  
    AX = ARR[i-1] + 0x1234;  
}
```


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 (b)—a while loop simply repeats the loop body as long as the condition tested at the beginning of the loop is true.

```
while (Y != X) {  
    Y = X - AX;  
    X = Z + AX;  
    Z = Z - 2;  
}
```