16.317: Microprocessor Systems Design I

Fall 2013

Exam 2

November 6, 2013

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 3 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 3 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 five pages (3 double-sided sheets) of reference material for the exam: a list of the x86 instructions and condition codes we have covered thus far, as well as a description of subroutine calling conventions. 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	/ 20
Q2: Subroutines; HLL → assembly	/ 40
Q3: Conditional instructions	/ 40
TOTAL SCORE	/ 100

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.

For all parts of the question, assume you have an x86 processor running in protected mode.

a. Which of the following selector values indicate that the given segment is in global memory?

   A. `SS = 0138h`
   B. `DS = FF13h`
   C. `GS = 118Bh`
   D. `ES = 5371h`

   i. A and C

   ii. B and D

   iii. A, B, and C

   iv. All of the above (A, B, C, and D)

   v. None of the above

b. Which of the following statements about local memory accesses in protected mode are **true**?

   A. The LDTR is a selector that points to a descriptor in the global descriptor table; that descriptor provides information about the current local descriptor table.

   B. The LDTR stores the base address and limit (maximum offset) for the current local descriptor table.

   C. The LDTR is changed every time a task makes an access to local memory.

   D. The LDTR cache stores the base address and limit for the current local descriptor table.

   i. Only A

   ii. Only B

   iii. A and D

   iv. B and C

   v. B, C, and D

1 (continued)
Again, assume you have an x86 processor running in protected mode. Parts (c) and (d) of the question refer to the following table, which shows the current local descriptor table. The descriptor shown at address 18800000h is the first descriptor in this table.

Memory	Address

Base = 11308100h
Limit = 0007h
	18800000h

Base = 1578AA00h
Limit = 0FFFh
	18800008h

Base = 33731700h
Limit = 0027h
	18800010h

Base = 54440000h
Limit = FFFFh
	18800018h

Base = 09AC2200h
Limit = 00FFh
	18800020h

Base = 35700100h
Limit = 007Fh
	18800028h

c. If DS = 000F, what is the starting address of the current data segment?

i. 000F0h

ii. 11308100h

iii. 1578AA00h

iv. 18800008h

v. DEADBEEFh

d. Say the selector SS breaks down as follows: requested priority level (RPL) = 3, table indicator (TI) = 1, index = 3. If ESP = 0000FFC0h, what linear address corresponds to the logical address SS:ESP?

i. 0000FFC0h

ii. 1880FFD8h

iii. 54440000h

iv. 5444FFC0h

v. ABAD1DEAh

2. (40 points) ***<u>Subroutines; HLL → assembly</u>***

The following questions deal with the simple C function shown below, which takes two integer arguments (`v1` and `v2`), contains one local variable (`x`) and returns the value shown:

```
int f(int v1, int v2) {
   int x = v1 – v2;
   return (x << 4) - x;
}
```

a. (16 points) Draw the stack frame for this function if it is called with 10 and 2 as its arguments (in other words, a program contains the function call `f(10, 2)`). Be as specific as possible—in particular:

- Show all known values—if, for example, the argument `v1` is equal to 20, write the value 20 in your diagram, not the argument name `v1`.
- For all arguments or variables with unknown values, write the argument or variable name.
- Clearly indicate where the stack pointer (`esp`) and base pointer (`ebp`) point in the current stack frame. You do not need to know the values of these registers.

Assume the function saves the register `ebx` on the stack, since it overwrites that register.

2 (continued)

b. (24 points) A partially completed x86 assembly version of this 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.

The C version of the function is provided below for your reference. Note that a variable of type `int` is a 32-bit signed integer.

```
int f(int v1, int v2) {
    int x = v1 – v2;
    return (x << 4) – x;
}
```

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

   _____________________________       ; Create space on stack for x

   _____________________________       ; Save ebx on the stack

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

   _____________________________       ; ebx = v1 - v2

   _____________________________       ; x = ebx = v1 - v2 (copy ebx
                                       ;    to memory location for x)
   sll     ebx, 4                      ; ebx = ebx << 4 = x << 4

   _____________________________       ; ebx = ebx – x = (x << 4) - x

   _____________________________       ; Restore ebx

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

   _____________________________       ; Return from subroutine
f  ENDP
```

3. (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.**

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 `AX`).

```
if (X > 10) {
   if (Y > 0) {
      DX = X;
   }
   else {
      DX = Y;
   }
}
else
   DX = 0;
```

3 (continued)

b. Implement the following loop. Assume that ARR is an array of 11 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 = 0; i <= 10; i++) {
    ARR[i] = ARR[i] + AX;
    AX = BX – ARR[i];
}
```

3 (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.

```
while (X < Y) {
   X = X + 1;
   Y = Y – 1;
}
```