CSCI-UA.0201-001

Computer Systems Organization

Makeup Midterm Exam Fall 2016 (time: 60 minutes)

Last name: First name:

Notes:

- **If you perceive any ambiguity in any of the questions, state your assumptions clearly.**
- **Questions vary in difficulty; it is strongly recommended that you do not spend too much time on any one question.**

---

1. (5 points) Circle the correct answer among the choices given. If you circle more than one answer, you will lose the grade of the corresponding question.

(A) The assembler is:

1. machine dependent 2. language dependent 3. both 4. none

(B) The following number: 0x1F000001 can be interpreted as:

1. signed int
2. unsigned int
3. single precision floating point
4. all of them

(C) The instruction *leal (%eax), %ebx* accesses the memory:

1. once 2.twice 3. 0 times 4. depends on whether 32-bit or 64-bit

(D) Regarding instruction set architecture, backward compatibility means

1. executing old instructions on new hardware
2. executing new instructions on old hardware
3. both 1 and 2
4. none of the above

(E) Which of the following pointers has a larger size (in terms of bytes)?

1. pointer in a 32-bit machine
2. pointer in a 64-bit machine
3. pointer in a 32-bit machine pointing to an array of 100 integers

2. (2 points) assign a value to x and a value to y (you can specify them in binary) such that

a. (x && y) is evaluated to true and (x & y) is evaluated to false

One possible example is: x = 00…01 and y = 00…10
Of course both x and y are of the same type.

b. How about the other way around?

Cannot be done because for the first expression to be false at least one variable must be 0, which will automatically make the second expression false too.

3. [2 points] In C, like in many other languages, we need to declare a variable before we can use it. For instance, we have to declare *int x;* before we can use x. Why is that (state two reasons)?

- **So that the compiler knows how many bytes to tell the OS to reserve and/or which type of registers to use (e.g. rax vs eax vs ax for example)**
- **For the pointer arithmetic to be done correctly.**
- **To give a name to a memory location so that we can use it in our program.**

4. [2 points] Suppose we have the following decimal number: -10

a) Write that number in an 8-bit binary number. To get full credit, show all the steps.

First, let's get +10 → 0000 1010
Then we get the 2's complement which will bring it to -10
→ 2's complement(0000 1010) = <u>1111 0110</u>

b) Translate the number you calculated in a) above to hexadecimal.

1111 0110 → 0xF6

5. [2 points] Suppose x is an integer. We want to test whether the two most significant bits of x are 1 or not (i.e. the two left most bits), so we wrote the C expression:

if(….)
{ *tests successful and the two bits are 1* }
else
{*means at least one bit of the two most significant bits is 1*}

What will you put between the parenthesizes in order to test that condition?

To get the correct condition you must ensure that:
(The most significant bit is 1) AND (The bit next to it is also 1)
So

sol 1: if((x & 0x80000000) && (x & 0x40000000))

sol 2: if((x & 0xc0000000) == 0xc0000000)

6. Suppose that we have the following number: 0xAA
   a) [1 point] Write this number in binary:

      1010 1010

   b) [2 points] Suppose that this number is interpreted as unsigned number, what is the decimal equivalent (note: you don't have to write a final decimal number, you can leave it in the format of 2^x+2^y+ …). To get full score, show all the steps.

      We have 8 bits. So each bit will be multiplied by 2^x where x depends on the position of the bit. We start form the far right (least significant bit) where x =0 till we reach the far left (most significant bit) where x = 7.
      $2^7 + 2^5 + 2^3 + 2^1$

   c) [2 points] Suppose that this number is interpreted as signed number, what is the decimal equivalent (note: you don't have to write a final decimal number, you can leave it in the format of 2^x+2^y+ …). To get full score, show all the steps.

      The most significand bit is 1 so the number is negative.
      So, we need to get the 2's complement of 1010 1010 → 0101 0110
      The decimal equivalent is thus:
      $-(2^6 + 2^4 + 2^2 + 2^1)$ Note the negative sign.

      Another solution: $-2^7 + 2^5 + 2^3 + 2^1$

7. [2 points] Suppose “a” is a pointer to unsigned integer
(i.e. it was declared as *unsigned int * a;*) and points to the following array of unsigned integers: {1,1,2,2,3}.
How many times the body of the following loop will be executed? Justify

while((*a++) & 0x1) { .... loop body …. }

That loop checks whether the corresponding array element is odd. If it is odd, it will advance to the next element (a is incremented and, by pointer arithmetic, will move to the next element). Therefore, the loop body will be executed <u>twice</u>.