CSCI-UA.0201-001

Computer Systems Organization

Midterm Exam Fall 2017 (time: 60 minutes)

Last name: First name: NetID:

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.**
- **The exam consists of 4 pages, 9 questions, and a total of 50 points.**

---

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 compiler is:

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

(B) By seeing the number: 0xFA00700B we know for sure that it is:

1. signed int 2. unsigned int
3. single precision floating point (4. We do not know for sure)

(C) Suppose we have a 32-bit machine. The size of "short int" is:

1. 4 bytes 2. 8 bytes
(3. 2 bytes) 4. Depends on the compiler.

(D) Suppose we have a 64-bit machine. The size of "short int" is:

1. 4 bytes 2. 8 bytes
(3. 2 bytes) 4. Depends on the compiler.

(E) If we write a C program that includes a function call to a function that does not exist in the libraries and was not defined in our code. Then:

1. the compiler will complain 2. the assembler will complain
(2. the linker will complain) 3. the loader will complain

2. [4 points] Floating point variables take 4 bytes, same size as int variables. Yet, the range of numbers that float can present is much bigger than int. So, why don't we just get rid of int in programming languages and just use float. (Your answer must not exceed 2 sentences).

Because floating point operations are much slower and power hungry than integer operations.

3. [6 points] When we say we have a 64-bit machine. What does this mean? (State 3 implications to get full credit):

- **Addresses are 64-bit wide**.

- **Each register can store 64 bits.**

- **Data bus can carry 64 bits of data at once between the processor and memory.**

4. Suppose we have the following decimal number: -11

a) [3 points] Write that number in an 8-bit binary number. To get full credit, show all the steps.

+11 → to binary = 00001011 → to get -11 we need 2s complement → <u>11110101</u>

b) [2 points] Translate the number you calculated in a) above to hexadecimal.

11110101 → 1111 0101 → 0xF5

5. [6 points] Suppose x is an integer. Write **one C statement** that multiplies x by 67 **without using any multiplication operation**. That is x *= 67; or x = x * 67; are not accepted. Also x = x+x+x+ … 67 times and the like will not be accepted (hint: think in terms of shifting and addition operations).

x = (x << 6) + (x<<1) + x;

Explanation: (x<<6) → 64x
(x<<1) → 2x

6. [10 points]The following C code initializes an array of 1000 elements with the numbers from 0 to 999. That is, first element of the array is initialized to 0, second element to 1, etc. Then it adds these numbers and puts the result in sum. Finally, it prints sum on the screen. The code has 5 mistakes. Indicate those mistakes and show how you can fix them. Put your answer in the table below the code.

```
#include < stdio.h>

int main()
{
    unsigned char x;
    unsigned int * y;
    unsigned char sum;

    y = malloc(1000*sizeof(long));
    for( x =  0; x < 1000; x++)
        *(y + x ) = x;

    for( x = 0; x < 1000; x ++)
       sum + = y[x];

    printf("sum = %d\n", sum);

    free(y);

    return 0;
}
```

Mistake description	**Correct code**
1000 is bigger than the range of unsigned char	unsigned int x;
Unsigned char is not enough for anticipated result in sum	unsigned int sum;
Missing typecasting before malloc	y = (int *)malloc(sizeof(long));
Wrong type used in malloc	y = (int *)malloc(sizeof(unsigned int));
sum is not initialized	sum = 0; /* Anywhere before the second loop */

7. [2 points] In x86 assembly language, why can't we move data (1, 2, 4, or 8 bytes) from one memory location to another memory location in one instruction?

Because moving is an instruction; and instructions are executed by the CPU not the memory. Therefore, the CPU must be involved. Memory cannot move data by itself.

8. Suppose we have the variable x declared as unsigned char and initialized to some value.
   a) [2 points] Write one C statement that sets the most significant bit to 1 without affecting the other bits.

      x |= 0x80;

   b) [4 points] Based on what you did in a) did x become a negative number? Justify

      No, because x is *unsigned*, as indicated in the problem.

   c) [2 points] Assume x had its initial value (i.e. its value before you did step a above), write an if-statement in C that puts x to zero if its most significant bit is 1.

      if(x & 0x80) x = 0;

9. [4 points] Since pointers in 64-bit machines have the same size (8 bytes) no matter what they are pointing to, then why do we need to specify the type that the pointer is pointing to? State two reasons to get the full credit:

- **So that the compiler can generate the correct pointer arithmetic when accessing data structures like arrays.**

- **So that when you use the pointer to put a number in memory, the compiler knows how many bytes to use.**