CSCI-UA.0201-003

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) If your program consists of 5 C files, the output of the linker consists of :
[✓ 1. one file] 2. five files 3. depends on the compiler
4. depends on whether we have 32-bit or 64-bit machines

(B) Assume a signed int x, what does the following expression present:
(1 + (x<<3) + ~x + x)
a. 7x [✓ b. 8x] c. 8x + 1 d. 7x + 1 e. none of the above

(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 an undefined variable. 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] State one advantage and one disadvantage of MACROs.

[Advantage]

The execution is fast because it does not involve any stack management.

[Disadvantage]

The code can be bigger because each occurrence of the macro name in the code will be replaced by the whole code of the macro.

4. Suppose we have the following decimal number: -23

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

+23→ in binary = 00010111 → getting the 2's complement: <u>11101001</u> = -23

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

11101001 → 1110 1001 → 0xE9

5. [6 points] Suppose x is an integer. Write **one C statement** that multiplies x by 51 **without using any multiplication operation**. That is x *= 51; or x = x * 51; are not accepted. Also x = x+x+x+ … 51 times and the like will not be accepted (hint: think in terms of shifting and addition operations).

x = (x<<5) + (x <<4) + (x<<1) + x;

Explanation: (x<<5) = 32x
(x<<4) = 16x
(x<<1) = 2x

6. [10 points]The following C code initializes an array of 500 elements with the numbers from 0 to 499. 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. (Note: some of these mistakes may NOT make the compiler complain)

```
#include < stdio.h>

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

    y = malloc(500*sizeof(int));
    for( x =  0; x < 500; x++)
        *(y + x ) = x;

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

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

    --some other function calls not relevant to the problem ---

    return 0;
}
```

Mistake description	**Correct code**
The range of char cannot reach 500	int x; /* unsigned chat x works too */
sum is not initialized	sum = 0; /* Anywhere before the second loop */
Missing typecasting in malloc	y = (int *)malloc(500*sizeof(int));
The second loop goes over 500	for(x = 0; x < 500; x++)
malloc without free	free(y); /* Just after the printf */

7. [2 points] We say that the memory is designed such that each byte has an address. Why don't we simplify the memory design and give one address to every 4 bytes for example?

If every 4 bytes have an address then it will be very hard and wasteful to reserve memory for types as char and short which have sizes less than 4 bytes.

8. Suppose we have the variable x declared as char and initialized to some value.

a) [2 points] Write one C statement that sets the least significant bit to 1, leaving all other bits intact.

x |= 1;

b) [2 points] Based on what you did in a) did x become even number? Odd number? Or it has no relation.

x became an odd numbers because even numbers have least significant bit = 0 (try yourself and check 2, 4, 6, 8, … in binary).

c) [2 points] Write an if-statement in C that puts x to zero if its most significant bit is 1.

if(x & 0x80) x = 0;

d) [2 points] Repeat question c) above but use a different condition in if-statement to check for the same condition.

if(x < 0) x = 0;

9. [4 points] We saw that we can specify addresses in x86 as D(Ra, Rb, S)

a) Why such complicated form is needed and not just use the simpler (Register) ?

To be able to translate complicated actions in high-level languages (HLL) such as array elements access, 2D array access, array of structures access, etc in assembly. With that complicated memory form, those sophisticated accesses in HLL will require many assembly instructions to accomplish.

b) S can only take the values 1, 2, 4, and 8. Why is that?

To implement correct pointer arithmetic for the basic types: char, short, int, and long which have sizes 1, 2, 4, and 8.