Course: **Computer Architecture** (2020 Fall)

(Always write code in valid C99/C11 standard.)

(You are allowed to read lecture materials. But you shall not disclose your answer during the quiz.)

(09:10 ~ 10:00AM)

1. What is the hexadecimal representation for decimal **`-51`** encoded as an 8-bit two's complement number? [5 points]

2. Can the value of the sum of two 2's complement numbers **0xB3 + 0x47** be represented using an 8-bit 2's complement representation? If so, what is the sum in hex? If not, write NO. [5 points]

3. Suppose we have defined a linked list `struct` as follows. Assume `*lst` points to the first element of the list, or is NULL if the list is empty.

```
struct ll_node {
    int first;
    struct ll_node *rest;
}
```

(3a) [5 points] Implement `add_front`, which adds one new value to the front of the linked list. Function prototype: `void add_front(struct ll_node **lst, int value);`

(3b) [5 points] Implement `free_ll`, which frees all the memory consumed by the linked list. You must prevent users from writing to unusable memory after free.

Function prototype: `void free_ll(struct ll_node **lst);`

4. The IEEE 754 standard defines a binary representation for floating point values using three fields.

- The sign determines the sign of the number (0 for positive, 1 for negative).
- The exponent is in biased notation. For instance, the bias is `-127` which comes from -(2^{8-1} − 1) for single-precision floating point numbers.
- The significand or mantissa is akin to unsigned integers, but used to store a fraction instead of an integer.

The below table shows the bit breakdown for the single precision (32-bit) representation. The leftmost bit is the MSB and the rightmost bit is the LSB.

1	8	23
sign	Exponent	Mantissa/Significand/Fraction

For normalized floats:

$\texttt{Value} = (-1)^{\text{Sign}} * 2^{\text{Exp+Bias}} * 1.\text{significand}_2$

For denormalized floats:

$\texttt{Value} = (-1)^{\text{Sign}} * 2^{\text{Exp+Bias+1}} * 0.\text{significand}_2$

Exponent	Significand	Meaning
0	Anything	Denorm
1-254	Anything	Normal
255	0	Infinity
255	Nonzero	NaN

Note that in the above table, our exponent has values from `0` to `255`. When translating between binary and decimal floating point values, we must remember that there is a bias for the exponent.

(4a) [5 points] What is the largest finite positive value that can be stored using a single precision float? Express in details.

(4b) [5 points] What is the smallest positive value that can be stored using a single precision float? Express in details.

(4c) [5 points] What is the smallest positive normalized value that can be stored using a single precision float? Express in details.

5. Not every number can be represented perfectly using floating point. For example, 13 can only be approximated and thus must be rounded in any attempt to represent it. For this question, we will only look at positive numbers.

(5a) [5 points] What is the next smallest number larger than 4 that can be represented completely?

(5b) [10 points] If we are given a normalized number that is not the largest representable normalized number with exponent value `x` and with significand value `y`, what is the stepsize at that value? (There are 23 significand bits.)

6. [15 points] Implement C function which gets a number and returns if it is positive (x >0). if x **<=0** then return false, if x **>0** then return true.

example: 0 => 0/false, -12 => 0/false, 29 => 1/true

However, you CAN NOT use any flow control (e.g. `if, else, for, switch, case, goto, while, "?"`) nor logical operators (e.g. `&&, ||, >, <, ==, <=, >=`). Instead, you SHALL use operator `<<, >>, &, |, +, ^, ~, !` to implement.

Assume `sizeof(int) = 4`

Function prototype: `int is_positive(int x)`

7. Assume a new execution mode called "turbo mode" provides a 2.5x speedup to the sections of programs where it applies. What percentage of a program (measured by original execution time) must run in the "turbo mode" for an overall speedup of 10%? [5 points]

8. Following the bit-level floating-point coding rules, implement the function with the following prototype:

```
/*
 * Compute (int) f
 * If conversion causes overflow or f is NaN, return 0x80000000
 */
typedef unsigned float_bits;
int float_to_int(float_bits f);
```

For floating-point number f, this function computes `(int) f`. Your function should round toward zero. If f cannot be represented as an integer, then the function should return `0x80000000`. [20 points]

9. [10 points] Write code for a function with the following prototype:

```
/* Divide by power of two. Assume 0 <= k < w-1 */
int div_power2(int x, int k);
```

The function should compute $x/2^k$ with correct rounding.

10. [10 points] Write code for the function with the following prototype:

```
/*
 * Return 1 when x can be represented as an n-bit, 2's complement
 * number; 0 otherwise
 * Assume 1 <= n <= w
 */
int fits_bits(int x, int n);
```