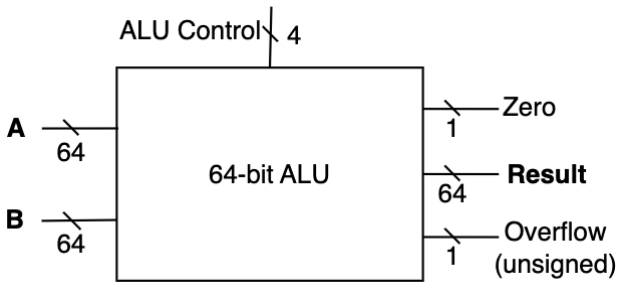
Department of Computer Science
National Tsing Hua University
EECS403000 Computer Architecture
Spring 2023 Homework 3

Deadline: 2023/4/16 23:59

1. (15 points) In class, we have described a 64-bit ALU whose control input ALU Control is composed of 1-bit Ainvert, 1-bit Bnegate, and 2-bit Operation from left to right. Re-design the ALU to meet the following specification. Note that (1) addition, subtraction, and set less than operations are all performed on unsigned integers, and (2) overflow happens when an addition result is too large (i.e., larger than 2^{64}-1) or a subtraction result is too small (i.e., smaller than 0) to represent. You need to draw the circuit diagrams of each 1-bit ALU and the 64-bit ALU; for each 1-bit ALU, you can use only one full adder to perform an addition or a subtraction operation in a manner similar to that shown in class.



ALU Control	Function
0001	and
0011	or
0010	add unsigned (addu)
0110	subtract unsigned (subu)
0100	set less than unsigned (sltu)
1111	nand

2. (16 points) Consider two unsigned binary numbers: M = 0111 and N = 1101.
   (a) (8 points) Write down each step of M × N according to version 1 of the multiply algorithm.
   (b) (8 points) Write down each step of M × N according to version 2 of the multiply algorithm.

3. (20 points) Consider two signed binary numbers: M = 0111 and N = 0110.
   (a) (10 points) Write down each step of M ÷ N according to version 1 of the divide algorithm.
   (b) (10 points) Write down each step of M ÷ N according to version 2 of the divide algorithm.

4. (20 points) Consider two decimal numbers: X = 0.3125 and Y = -15.96875.
   (a) (10 points) Write down X and Y in the IEEE 754 single-precision format. You must detail how you get the answers, or you will receive 0 points.
   (b) (5 points) Assuming X and Y are given in the IEEE 754 single precision format, show all the steps to perform X + Y and write the result in the IEEE 754 single precision format.
   (c) (5 points) Assuming X and Y are given in the IEEE 754 single precision format, show all the steps to perform X × Y and write the result in the IEEE 754 single precision format.

5. (20 points) Consider a new floating-point number representation that has only 8 bits containing the sign bit, exponent, and fraction from left to right. The leftmost bit is still the sign bit, the exponent is 3 bits wide and has a bias of 3, and the faction is 4 bits long. A hidden 1 to the left of the binary point is assumed. In this representation, any 8-bit binary pattern having 000 in the exponent field and a non-zero fraction indicates a denormalized number: $(-1)^{Sign} \times (0 + Fraction) \times 2^{-2}$. Write the answers of (a), (b), and (c) in scientific notation, e.g., 1.0101×2^2.
   (a) (3 points) What is the largest negative "normalized" number, denoted as a0?
   (b) (6 points) What is the smallest negative "denormalized" number, denoted as a1? What is the second smallest negative "denormalized" number, denoted as a2?
   (c) (4 points) Find the differences between a0 and a1, and between a1 and a2. Also, describe what you observe and any implications from them.
   (d) (3 points) What decimal number does the Hexadecimal pattern 0x5C represent?
   (e) (4 points) Let U be the nearest representation of the decimal number -5.7; that is, U has the smallest approximation error. What is the approximation error between U and -5.7? What is the actual decimal number represented by U?

6. (9 points) John wrote the C code below, where each "float" number is internally stored in the IEEE 754 single precision format. Under the assumptions that (1) each "float" number is not NaN and (2) the "float" numbers +0 and -0 are

considered equal, John would like to claim that the function greater_or_equal (float x, float y) will return 1 if $x \geq y$, and will return 0 if $x < y$. Is John's claim correct? Why?

```
unsigned float_to_unsigned(float x) {
    return *(unsigned*)&x;
}

//0 greater_or_equal
// return 1 if x >= y
// return 0 if x < y
/*
 * Assumptions:
 * 1. input x and y are NOT NaN.
 * 2. +0, and -0 are considered equal.
 */
int greater_or_equal(float x, float y) {
    unsigned ux = float_to_unsigned(x);
    unsigned uy = float_to_unsigned(y);

    unsigned sx = ux >> 31;
    unsigned sy = uy >> 31;


    return (ux << 1 == 0 && uy << 1 == 0) ||
    (!sx && sy) ||
    (!sx && !sy && ux >= uy) ||
    (sx && sy && ux <= uy);
}
```