Name: Tong, Justin Date: 3/21/2022

NYU ID: jht9626

Section: 007

Lab 1

Total Points:\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Professor’s Comments:

Affirmation of my Independent Efforts: Justin Tong

https://github.com/justintong25/CSOLab1

<https://github.com/justintong25/CSOLab1>

bitXor: I set a as x & y and b as ~x & ~y returning ~a & ~b. ~a & ~b is 1 if and only if a is 0 and b is 0. If a is 0, x and y cannot be 1 at the same time and is b is 0, x and y cannot be 0 at the same time. The two equations are equal when x is not equal to y.

tMin: shift 1 to the left by 31 to get the minimum

IsTmax: You take the opposite of the one’s complement and then take the xor of the opposite of x + 1. Then if you to the opposite of the bitwise or operator of the opposite value and then the new x value it will tell if the x value is the max value two’s complement or not. If x is the Tmax, then it will return ~(tmax + 1)

allOddBits: The odd numbered bits are set to one in the hexadecimal representation of 0xAAAAAAAA and check this number with bitwise or shifted 8 and 16 places over. Use bitwise & with the input to find the same bits and then return if all the bits are equal, if they are return 1 otherwise return 0.

Negate: return the one’s complement of the number and add 1 to get the two's complement.

isAsciiDigit: Find the upper and lower past the characters 0 and 9. Then find the 32nd digit for upper and it should be out of the range. Do the same for lower but plus 1. If any of upper or lower variables return 1, then it is not true, if upper and lower both return 1 then it is in the range.

Conditional: check if x is a value other than 1, if it is then x =1 otherwise 0

isLessThanOrEqual: find the two’s complements of the X input, then add together the y and the negative x value. Then we can find the sign of this new added x and y value. Find the tMin by shifting 1 over by 31 places. Find both the x and y signs by using the & bitwise operator to see which bit values are both 1s. Find the Xor of the x and y sign then find the sign of that value. If the opposite of the sign of the xor of x and y and the opposite of the sign of the y added with the x values. Use bitwise or of that with the xor of x and y and the sign of x and return. If it is less than or equal to, return 1 otherwise 0.

logicalNeg: set x equal to the bitwise or of its two’s complement to get the negative of x. Then you shift it over to the first bit. Add 1 to negate the value.

howManyBits: Every bit is a power of two so each time you add 1 to x and you shift until there is one bit left, continue and then return n + 1

Float\_twice: Find the exponent and sign, check for edge cases where exp is max value or 0. Add 1 to the exp, if it is greater than the max value, return the max value with the sign, otherwise, return the exp shifted back with the bitwise or of the input and the largest float number with the input number which returns the flaot number multiplied bh 2

Float\_i2f: if x is 0 return 0 if x is the lowest integer case then return lowest integer + 1. Find the sign and if the sign is 1 make x negative. Shift x over until you find the leading digit. Then find the exponents where the exponent = bias + the numbers after the decimal point after shifting x. Then you have to reshift x to the left by 31 minus the shift from before. You find the fraction by shifting x over by 8 because it is how many bits are in the exponent with the bitwise and operator of the largest value. Then you. Combine the sign, exp, and frac and then shift to have everything the same size.

float\_f2i: Find the sign, exponent, and fraction. If the exponent is greater than 31 then it is an overflow and return max value, if its less than 0 it is not possible so it is also 0. If the input not is bitwise and with the maximum value return 0. If the exponent is greater than 23, then the fraction gets shifted over to the left by 23 – exponent otherwise shift the fraction to the right by 23 – exponent. This is to have the fraction in a 1.xxxx value to find the bias. Then you shift around the fraction to find the leading digit.