Name Steinhof, Michael Date: 9 March 2022

NYU ID: MS12944

Section: 007

**Lab 1**

**Total in points** (100 points total):

**Professor’s Comments:**

**Affirmation of my Independent Effort: Michael Steinhof**

**(Sign Here)**

Explanations for Each Function

**Bit xor**

Essentially, the solution states that both digits cannot both be 1 and they both cannot be 0. If they are, we know the two integers are equal (outputting a 0).

**All odd bits**

This solution works by taking the input and doing an and comparison with a bit mask of 85. The bit mask contains ones in all its odd bits, so “anding” the two and checking if it matches 85 determines if all odd bits are equal to 1.

**Is ACII digit**

To find if a number is greater than another, we take the difference of the two numbers and get the sign bit of the result. We can do “subtraction” by taking the two’s complement and adding the two values together. By bit-shifting to the right by 7, we can then extract the sign bit. Then, by doing two comparisons this way and “anding” the result, we can find if the given value was within the range 0x30 <= x <= 0x39.

**Conditional**

First, we convert x to a 0 or 1, and then using the least significant bit, if the bit is 1 then we output y but if the bit is 0 we output z.

**Logical negation**

By comparing the most significant bit (using an or operation) of the integer to the most significant bit of the two’s complement, we are left with a -1 or a 0. -1 occurs when the two’s complement of the integer is non-zero. Conversely, the result is 0 when the integer is 0. By adding 1 to this result, we are left with the answer.

**T min**

The bit value of 1 is 00000001. Shifting this by 7 yields 10000000. Negating this therefore leaves us with 01111111. Finally, by adding 1 we are left with 10000000. Effectively, we take the maximum positive value that can be stored in 8 bits and add 1 to overflow to the minimum negative value.

**Is t max**

We compare the negation of the input to the minimum two’s complement value. If the input is the maximum value, then the negation will be the minimum value. The final step is to compare this value to our minimum value found in tmin (i.e., ~(1 << 7) + 1)).

**Negate**

To get the negative of a number, we simply take the two’s complement of that number.

**Is less or equal**

There are three case to be checked, and if any of them is true then we know that x <= y. The first [trivial] case is where x is the minimum two’s complement value. In this case, x <= y no matter what. The second case is where the subtraction (really, addition of y and the two’s complement of x) of the two values overflows and y is positive. The third case is when the value did not overflow and the sum of the two values is positive. If any of these three cases are true, then x <= y.

**How many bits**

I could not design a solution to this problem. I spent quite a while brainstorming with several peers on how to approach this problem (though any coding and write-up here is completely my own), but ultimately everything we tried seemed to fail or not meet the requirements in some major way. I would surely consider this the most difficult problem of the set but would like to say that I tried to complete it.

**Float twice**

This solution works by decomposing the floating-point representation of the integer into its three components: sign bit, exponent, and mantissa. Then, to do 2\*f, this is as simple as incrementing the exponent by 1 and re-assembling the floating-point representation.

**Float i2f**

This problem requires creating the three components of an floating-point representation of a number: the sign bit, exponent, and mantissa (fraction). The sign is already essentially given, as the integer is a signed integer (meaning it already has a sign bit). Following that, we use the sign bit to determine if we should take the two’s complement of our integer; if the sign bit is 1, we already handled that bit and can take the two’s complement to no longer have it. Next, we compute the exponent by repeatedly multiplying by 2 (using shifts). Finally, the mantissa is found. Last, any remainder is incorporated into the mantissa. Then, we return the 3 values combined while accounting for the bias in the exponent.

**Float f2i**

This is, broadly speaking, the same as the previous question. I found this direction (f2i) to be more challenging than i2f, but by breaking the problem down beforehand on paper and discussing it abstractly with peers, it largely resulted in the same process we have done before. Having the ability to use conditionals made the problem much more approachable, at the very least.