2023 Fall CSED 211 Lab Report

Lab number: 1

Student number: 20220302

Name: 김지현

1. Introduction
   - Lab 1 aims to learn bit-level representations of integers, exploring various fundamental bitwise operations such as bitwise NOR, checking for zero values, detecting overflow conditions, finding the absolute value of integers, and performing logical shifts.
2. System Design/ Algorithm
   1. Lab 1-1 "bitNor"
      - This function is to determine the bitwise NOR of the inputs x and y.
        - NOR is true only when both the inputs x and y are 0's.
      - On the instruction, it is shown that the bitNor function is ~(x|y). When you apply De Morgan's Formula for the above function, you will get ~x & ~y.
        - De Morgan's Formula: not (A or B) = not A and not B
   2. Lab 1-2 "isZero"
      - This function returns 1 when x is zero and 0 when x is not zero.
      - Using ! logical operation, we can determine whether x is zero or not.
      - By returning !x (when x is the input), if x is 0, the function will return 1 (true), and if x is not 0, the function will return 0 (false).
   3. Lab 1-3 "addOK"
      - This function determines whether x + y is allowed without causing any overflow.
      - We can think of three cases:
        - If the input x and y have different signs. In extreme cases, we could add Tmin and T max, x and y respectively, but there won't be any overflow.
        - If the input x and y have the same signs, and the sum of x and y also have the same sign, then, there isn't any overflow.

    - However, if the input x and y have the same signs, and the sum of x and y has different sign, there must be an overflow.
        - For example, think of adding -7 and -8 in 4 bits. -7 will be 1001 and -8 will be 1000. If you add 1001 and 1000, you will get (1)0001. If you ignore the carry, the answer comes out to be 1, while it should be -15. This is because – 15 is greater than -8 which is the smallest number that a 4bits can represent.
  - Therefore, in the code, one should determine signs for x, y, and the sum of x and y. Then, using XOR, we determine the possibility of overflow.
    - If, x and y have different signs, then either (x_sign ^ sum_sign) or (y_sign ^ sum_sign) will be 0. And the bitwise AND will be 0. Finally, if you do logical ! of 0, the result will be 1.
    - If x and y have the same signs and the sum has the same sign, then both (x_sign ^ sum_sign) or (y_sign ^ sum_sign) will be 0. Similarly, as in the first case, the result will be 1.
    - If x and y have the same signs and the sum has a different sign, then both (x_sign ^ sum_sign) or (y_sign ^ sum_sign) will be 1. The result of bitwise AND will be 1, and the result of logical ! will be 0.

4. Lab 1-4 "absVal"
    - This function returns the absolute value of x.
    - x_sign will be 111…1 if x is negative and 000…0 if x is positive.
    - By adding x_sign and x, you will get the positive counterpart of a negative number by adding two's complement of 1 and keep the positive number x the same.
    - Finally, XOR of the above result and x_sign will convert the above result bit-wisely if x is negative and keep unchanged if x is positive.
5. Lab 1-5 "logicalShift"
    - The mask will have n number of 1's in the right and 0's in the rest of the bits.
    - By doing the AND operation for the ~mask and x shifted to the right by n, you can keep the n bits on the MSB side as 0 because ~mask will be n 0's on the MSB side and 1's on the rest of the bits.