**2022-2 CSED211 Lab02 Report**

학번 : 20210479

이름 : 이주현

**명예서약 (Honor Code)**

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

1. **문제별 코드 설명**
   * **Problem 01\_negate()**
     + 2의 보수를 이용해 not 연산을 구현했다. 각 bit를 바꾸어주는 ~ 연산자를 이용해 ~x + 1과 같이 표현해 주었다.
   * **Problem 02\_isLess()**
     + 우선 이 문제에는 두 가지 경우가 있다. x가 음수, y가 양수인 경우 또는 둘의 부호가 같고 y – x가 양인 경우이다.
     + x가 음수, y가 양수인 경우: 둘의 sign bit을 비교하면 된다. 이때 sign bit이 다르기만 하면 되는 것이 아니라 둘의 부호가 정해진 상태이므로 xor 연산자 대신 x & (~y)와 같이 나타내어 sign bit을 1로 만들어주었다. 문제에서 y가 더 큰 경우 1을 반환하라고 되어있는데, 이 반환을 위해서는 sign bit를 미리 1로 만들어 주는 것이 용이하다.
     + 둘의 부호가 같고 y – x가 양인 경우: 이 역시 sign bit을 1로 만들어 비교가 편하도록 x – y 항을 구현했다. 이때 – 연산자를 사용할 수 없으므로, 2의 보수를 이용하여 (x + (~y + 1))과 같이 표현했다. 또한 둘의 sign bit이 같은 상태라는 것을 표시하기 위해 xor 연산을 사용해 주었다. 이때 단순한 xor 연산만으로는 MSB 값이 0이 될 테니 ~을 붙여 MSB를 1로 만들어주었다. 1로 맞추어준 이유는 위와 같다. (반환에 용이)
     + 이렇게 두 경우로 나누어 구현을 마친 후, or operator(|)로 묶어 반환해주었다. 이때 MSB가 반환값을 결정하므로 (situation1 | situation2)로 묶은 뒤 >> 31 & 1 동작을 추가해 정확한 값을 반환하도록 만들었다.
   * **Problem 03\_float\_abs()**
     + 문제 해결에 앞서, 예외가 되는 상황을 찾아보았다. 이 문제에서는 NaN만 예외처리해주면 되었다. 예외가 발생한 경우 문제에 제시되어 있듯이 uf의 원래 값을 반환하였다.
     + 예외가 아닌 모든 경우에 대해, 0x7FFFFFFF과 and 연산을 적용하여 맨 앞의 sign bit을 0으로 만든 후 그 값을 반환하였다.
     + 이때, 원래 값인 uf의 경우 sign bit이 두 가지로 나뉘므로, 우선 uf에 0x7FFFFFFF과의 and 연산을 적용한 값을 abs라는 변수로 저장해놓고 그것을 이용해 NaN 예외 처리를 진행했다. 이렇게 abs를 사용할 시 더 적은 연산자를 이용해 효율적으로 예외를 처리할 수 있다.
   * **Problem 04\_float\_twice()**
     + 우선 입력받은 uf를 sign / exp / frac 비트로 나눠주었다. 그 후 예외를 처리했다. 이 문제에서도 NaN만 예외처리해주면 되므로, exp가 0xFF와 같다면 문제에 제시된 대로 원래의 uf 값을 반환하는 것으로 예외 처리를 마무리했다.
     + 나머지 일반적 경우에 대해서는 두 가지로 종류를 나누어 처리했다.
       - denormalized: exp가 0인 경우 아주 작은 수인 denormalized에 해당된다. 이 경우 frac을 1비트만큼 left shift하는 것을 통해 twice 연산을 수행했다. 연산 수행 후 sign, exp, frac을 다시 순서대로 합쳐 반환해 주었다.
       - normalized: exp에 1을 더하여 twice 연산을 수행할 수 있으므로, 원래 값인 uf에 (1 << 23)을 더해 반환해 주었다.
   * **Problem 05\_float\_i2f()**
     + 우선 이 문제에서는 예외가 두 가지 있었다. 0과 TMin이다. 0인 경우에는 그냥 0을 출력해 주었고, TMin인 경우에는 IEEE rule에 맞게 변환한 constant인 0xCF000000을 반환해 주는 것으로 예외 처리를 마무리했다.
     + 일반적인 경우에 대해서는 다음과 같은 방법으로 IEEE rule을 구현했다.
       - sign: 정수가 음수인 경우, - 연산을 한번 거쳐 양수 형태로 표현되도록 해주었다. 그래야 제대로 exp와 frac을 표시해줄 수 있다.
       - exp: while문을 이용해서 x가 1.xxx … 로 표현될 수 있는 가장 큰 shift count를 구해 shift라는 변수에 저장해 주었다. 그리고 while문을 돌며 마지막에 1이 한번 더 더해지므로 while문을 빠져나온 즉시 shift에서 1을 빼 정확한 값을 저장하도록 했다. 그리고 그것을 bias(127)와 더해 exp bit를 표시해 주었다.
       - frac: x를 (31 – shift)만큼 left shift 하고, exp bit수(8비트)만큼 right shift해 구해 주었다. 이때 shift가 23보다 클 경우 rounding이 필요하므로 따로 조건을 빼내어 rounding을 진행해 주었다.
       - rounding: shift가 23보다 큰 경우에 대해 진행했다. 우선 rounding이 필요한 자리 바로 뒤의 자리들을 round라는 변수에 저장해 주었다. 그리고 근사 조건에 따라 근사가 필요할 경우 frac에 1을 더해주었다. 이때 frac에 overflow가 발생했을 경우 exp에 1을 더해주었다.
     + IEEE rule 구현 후, 위 문제와 같이 sign, exp, frac을 다시 순서대로 합쳐 반환해 주었다.
   * **Problem 06\_f2i()**
     + 우선 이 문제에서는 예외가 두 가지 있었다. NaN과 infinity이다. 문제에 제시된대로, 이러한 숫자가 입력되었을 경우에는 0x80000000을 반환하도록 해 예외 처리를 마무리했다.
     + 일반적인 경우는 다음과 같이 나누어 구현했다.
       - denormalized: exp가 bias, 즉 0x7F보다 작을 경우 denormalized이므로 0을 반환해주도록 하였다.
       - normalized: 우선 exp에서 0x7F를 빼 실제 E를 구해주었다. 그리고 이때 overflow가 발생했을 경우 이 역시 예외로 처리해주었다. 그 후 exp가 23보다 밑인지 위인지 파악해 각각의 경우에 따라 frac을 right/left shift해주었다. 이때 이 수들을 저장하기 위해 answer라는 변수를 사용해 주었다.
       - 그리고 answer에 (1 << exp)를 더해 E를 표시해주었다. 이때 원래 숫자의 sign이 1이었다면 정수도 음수로 표현되어야 하므로 그 경우를 if 문으로 따로 설정해 – 연산을 거치도록 했다.
       - 이렇게 각 단계를 거쳐 완성된 answer를 반환해 함수 구현을 마무리했다.
2. **Lab02에서 배운 점**

* Lab02를 해결하며 bitwise operator의 작용을 복습할 수 있었다. 앞문제들로 Lab01을 복기하고, 그것을 이용해 Lab02에서 다루게 된 floating point 문제들도 해결할 수 있었다.
* 조금 더 많은 연산자와 조건문, 반복문 등의 사용법을 복습했다.
* floating point single precision을 복습할 수 있었다. sign, exp, frac의 비트 수와 각각의 표현법 등 수업에서 배운 내용을 복습하고 해당 개념을 적용해 문제를 풀며 미처 이해하지 못했던 부분까지 알고 공부하는 기회를 가질 수 있었다.