**Lab2 report**

20210774 김주은

2-1. negate

negate를 구하는 함수로, x에 ~연산을 취한 후 1을 더하여 결과값을 return한다.

2-2. isLess

x가 y보다 작으면 1을 return하는 함수이다. Less를 greater or equal의 반대라고 생각하고 문제를 해결하였다. x>>31 & 1이라는 expression을 통해 right shift를 한 후 1과 &연산을 하여 x의 부호를 구하고 마찬가지의 방법으로 y의 부호도 구한다.

x\_sign과 y\_sign에 ^연산을 취하여 두 변수의 부호가 다른지 확인한다.

x-y를 (-y+1)+x를 통해 계산하고 이를 shift right하여 sum의 부호를 계산한다.

여기서 greater or equal은 x >= y를 말하는데 이를 계산해본다.

sign\_dif & y\_sign : 부호가 다른 경우에는 y의 부호가 음수일 때 항상 true이다.

!(sign\_dif | sum) : 부호가 같고, sum이 양수이면 항상 true이다.

그러므로, sign\_dif & y\_sign과 !(sign\_dif | sum) 을 | 연산을 취한 후 마지막에 ! 연산을 하여 greater or equal 이 아닐 때, 즉 less 일 때 1을 return 하도록 한다.

2-3. float\_abs

먼저 각각 sign exp frac 부분을 계산한다.

sign은 uf 를 31만큼 shift right하여 구하고, exp는 23만큼 shift right, frac은 0X7FFFFF와 & 연산을 통해 구한다.

exp와 0xFF가 같고, frac과 0이 다르다면 난수이므로 난수 일 때, uf를 return하도록 한다.

nan이 아닌 경우에는 sign에 0을 대입하여 양수를 만들고 나머지 비트끼리 or 연산을 통해 return한다.

2-4. float\_twice

먼저 각각 sign exp frac 부분을 계산한다.

sign은 uf 를 31만큼 shift right하여 구하고, exp는 23만큼 shift right, frac은 0X7FFFFF와 & 연산을 통해 구한다.

exp와 0xFF가 같고, frac과 0이 다르다면 난수이므로 난수 일 때, uf를 return하도록 한다.

Nan이 아닌 경우에는, 먼저 exp가 0인 경우에는 denormalized인 case이므로 frac을 left shift 1만큼 진행하고, 만약 left shift했을 때 범위를 벗어나는 경우에는 exp 를 1만큼 증가해준다. 여기서, 범위를 벗어나는 경우는 frac>>23이 0과 다르다는 것으로 판별을 한다.

그리고 exp가 0xFF보다 작은 경우에는 exp 를 1만큼 증가시켜준다.

이후 exp와 frac 등을 다 설정한 후에는 sign과 exp, frac을 알맞은 비트 수만큼 shift해준 후 | 연산을 취하여 return 한다.

2-5. float\_i2f

먼저 bias 변수를 0x7F로 초기화한다. (127)

exp 변수는 bias에 31을 더한 값인 158을 대입해준다.

sign\_mask 변수에는 1<<31을 한 값을 넣어서 sign을 판별할 때 쓰도록 한다. 즉, msb만 1인 bit pattern을 가진 셈이다.

sign 변수는 x의 sign 정보를 담는 변수로, x와 sign\_mask를 &연산을 통해 msb가 1인지 0인지 저장한다.

추가적으로 뒤에 계산에서 쓰일 frac와 round\_up을 선언해준다.

<예외처리>

먼저, x가 0일 때는 0이므로 그대로 0을 return한다.

두번째로는 INT\_MIN일 경우에는 따로 계산해준다. sign\_mask가 즉 INT\_MIN이므로 이를 통해 판단을 하고 해당되면 sign\_mask와 exp << 23 를 | 연산을 통해 계산하여 return한다.

<계산>

sign이 1인 경우, 즉 음수인 경우에는 양수로 바꾸어 계산한다.

먼저 x를 계속하여 1씩 shift left를 하여 msb가 1이 될 때 까지 반복하고 반복할 때 마다 exp 를 감소시켜주며 exp를 계산한다. exp는 기존 158이었기 때문에 반복하는 수만큼 감소하게 될 것이다.

이후 shift left된 x를 ~sign\_mask, 즉 msb를 제외하고 모두 1인 bit pattern과 & 연산을 취하여 8만큼 shift right 해주며 frac을 계산한다.

여기서 frac을 구해주는 과정에서 소수점 아래에 내려가는 bit들이 생길 수 있기에 이를 고려하여 올림을 해주어야 한다. 위에서 선언한 round\_up 변수를 활용하여 계산하는데,

x를 먼저 0XFF와 &연산을 취하여 lsb쪽의 8개의 bit들을 추출하고, 마지막 8비트가 0X80보다 크거나, 0X80임과 동시에 frac의 lsb가 1이면 round\_up에 1을 대입한다. 이는 rounding하는 규칙과 연관이 있는데, lsb아래의 10, 11을 체크한후 11이면 무조건 올림, 10이면 lsb가 1일 때 올림임을 반영한 결과이다.

round\_up인 경우에는 frac에 1을 더하고, 더한 frac이 범위를 벗어나는 경우, 즉 frac을 23만큼 shift right했을 때 1인 경우에는 frac을 모두 0으로 만들어주고 exp를 1만큼 더해준다.

이렇게 총 계산된 결과로 sign과 exp와 frac을 각각 알맞게 shift한 후 |연산을 통해 결과값을 return한다.

2-6. float\_f2i

나중에 계산하여 return할 result 변수를 선언해준다.

sign,exp, frac 부분을 각각 계산하여 (위 문제들에서와 같은 방식으로 shift) 각 변수에 대입한다.

bias를 선언하고 0X7F를 대입해준다.

exp – bias 계산하여 e 값에 대입해준다.

<예외처리>

exp 가 0XFF이면 nan이거나 infinite이므로 0x80000000을 return해준다.

e가 0보다 작으면 결국 2^e가 1보다 작으므로 정수로는 0으로 변환하여 return한다.

마지막으로 e가 31보다 클 경우에는 overflow가 발생하므로 0x80000000를 return한다.

<result 계산>

e가 23보다 작으면 frac을 23-e만큼 shift right하고, 그 외의 경우에는 e-23만큼 shift right한다.

본래 M = 1+frac이므로 result에 1을 e만큼 shift left한 수를 더해준다.

이후 sign이 1일 때, 즉 음수일 때는 계산된 result를 음수로 바꾸어준다.

최종 계산된 결과인 result 값을 return한다.