**CSED211 Lab 2**

20210643 김현준

1. int negate

negate 함수는 two’s complement를 이용하여 간단하게 구현할 수 있었다. two's complement는 ~연산 후 1을 더해주면 되므로, 그대로 구현하였다.

2. int isLess

isLess 함수의 경우 x가 y보다 작으면 1을 리턴하므로, 두 가지 경우로 나누어 구현하면 간단할 것으로 생각하였다. 첫 번째 경우는 x, y의 부호가 다른 경우이다. 이 경우에는 x<0이고 y>=0일 때만 1이 리턴된다. 코드로는 x&(~y)로 표현할 수 있었다. 두 번째 경우는 x, y의 부호가 같은 경우이다. 부호가 같은 것은 코드로는 ~(x^y)로 표현하여, 부호가 같을 때 1이 나오게 할 수 있었다. 그리고, 부호가 같을 때에는 x-y가 음수여야 x<y이다. 따라서 x-(~y+1)로 x-y를 계산하여 앞서 ~(x^y)에 &연산해 주었다.

이렇게 두 경우로 나눠서 구현한 이유는 부호가 다른 경우는 overflow가 발생할 수 있기 때문이다. 두 경우로 나눠서 이러한 overflow를 피할 수 있었고, 결론적으로 첫 번째와 두 번째에서 나온 식을 31만큼 right shift하고 &1연산하여, x<y이면 1을 리턴하는 함수를 구현할 수 있었다. 이때 shift한 이유는, MSB에 두 경우에서 계산한 참, 거짓 여부가 있었기 때문이다.

3. float\_abs

float\_abs의 경우 single-precision floating point의 절댓값을 구하는 함수이다. Floating point에서 MSB는 sign bit이므로 절댓값 기능을 위해서는 MSB만 0으로 만들어 주면 될 것으로 생각하였다. 하지만 그 전에, NaN인 경우를 고려해야 했고, 따라서 frac와 exp를 각각 구하였다. Single-precision floating point의 성질에 따라 exp는 23만큼 right shift하여 &0xff를 해주어 구하고, fac는 &0x7fffff를 해주어 구하였다. 그리고 if를 사용하여 frac!=0이고 exp==0xff일때가 NaN이므로, 해당 상황에는 uf를 그대로 리턴할 수 있도록 구현하였다. 그리고 나서 NaN이 아닌 경우에는 그냥 MSB만 0이 될 수 있도록 &0x7fffffff를 해주어 절댓값을 리턴하였다. 이러한 방식으로 구현하니, floating point의 절댓값 기능을 구현할 수 있었다.

4. float\_twice

이 함수는 single-precision floating point를 두 배 해 주는 함수이다. 세 가지 경우를 나누어 생각해볼 수 있었고, 첫 번째는 NaN, 두 번째는 Denormalized인 경우, 세 번째는 Normalized인 경우였다. 우선, sign과 frac, exp를 각각 문제 3번과 같이 구해준 다음(sign은 MSB에 있으므로 31만큼 shift하여 &1 연산을 하여 구함), 문제 3번과 같이 frac!=0이고 exp==0xff일때가 NaN이므로 이때는 uf를 그대로 리턴하도록 하였다.

다음으로 Denormalized인 경우, 이때 exp는 0이므로 이때 if문이 작동되도록 하였고, frac를 left shift하여 두 배를 해 주었다. 그런데 이때 frac이 범위를 넘어갔을 수 있으므로 if문 하나를 더 사용하여 frac>>23이 0이 아닌 경우 exp가 1 증가되고 frac이 다시 &0x7fffff연산되어 세팅 되도록 하였다.

마지막으로 Normalized인 경우에는 exp를 1 증가시키고, infinity일 경우를 대비하여 exp가 0xff인 경우 frac=0이 되도록 하였다. 이렇게 세 가지 경우에 따라 각각 exp와 frac을 구했고, 마지막으로 처음에 right shift시켜준 만큼 다시 left shift하여 |연산으로 모두 합쳐준 값을 리턴하여 두 배 계산을 최종적으로 완료하였다.

5. float\_i2f

이 함수는 int를 float로 casting해 주는 함수이다. 구현을 위해 먼저 생각한 점은, exp=E-bias를 이용하여 계산하면 되겠다는 것과, 음수나 양수나 관계없이 절댓값이 같으면 frac과 exp 패턴은 동일하다는 것이었다. 그리고, while문을 사용하여 E를 구하고 exp를 구하면 될 것으로 생각하였고, 주어진 x를 leftshift함과 동시에 E를 증가시키면 될 것으로 생각하였다. 여기서 조금 더 편리성을 위하여 int e=158로 설정하여 놓고, while문이 한 번 돌아갈 때마다 e에서 1씩 빼는 것으로 구현하고자 하였다.

먼저, 구현을 위해 위에서 나온 e를 설정하였고 sign을 구하기 위한 mask bit를 1<<31로 설정하였다. 그리고 sign을 구해주고, exp와 frac은 선언만 해 두었다. 다음으로 가장 먼저 한 것은 예외 경우들을 처리하는 것이었다. X가 0인 경우와 Tmin인 경우 각각 0, 0xcf000000을 리턴하도록 하였다. 그리고 sign이 1인 경우 x를 양수로 바꾸어 두었다. 이는 절댓값이 같으면 exp와 frac이 다르지 않기 때문이다.

다음으로 while문을 사용하였는데, 조건은 !(x&mask)로 달아 x가 전부 다 shift될 때까지 지속되도록 하였다. While문 안에서는 x를 1만큼 left shift, e를 1씩 감소시켰다. while문을 거치고 나서 남은 e 값이 곧 exp가 되므로, exp는 e를 leftshift해주어 구하였다.

다음으로는 frac을 구하였다. Frac은 x를 !mask와 &연산하고 8만큼 right shift하여 구하였다. 그런데, 여기서 중요한 점은 frac을 8만큼 shift했으므로 필요한 경우에 rouding해주어야 한다는 것이었고, x&0x80인 경우와 ((x&0x7F) > 0)||frac&1)인 경우가 rounding이 필요하므로 이때 frac을 1 더해주도록 하였다. 마지막으로 sign과 exp와 frac을 모두 더하여 결과값을 리턴하였다. 이때 더한 이유는 frac에서 1 더했을 때 범위를 넘어가게 되면 바로 exp가 증가되도록 하여 간단하게 구현하기 위함이었다.

6. float\_f2i

이 함수는 5번에서 구현한 것의 반대 버전이었다. 우선 sign과 exp와 frac을 구하고, NaN과 infinity인 경우 0x80000000u를 리턴해 주었다. 그리고 나서 denormalized, normalized인 경우를 각각 따져서, denormalized인 경우 버리면 되므로 0을 리턴하고, Normalized인 경우 exp에서 bias만큼 빼주었다. 그리고 나서 overflow가 발생했는지 여부를 체크하고, exp에 따라 frac을 두 경우로 나눠 shift해 주었다. Exp가 22보다 큰 경우에는 exp-23, 아닌 경우에는 23-exp만큼 shift하였다. 마지막으로 frac 앞에 1이 떨어져 있는 상태이기 때문에 1<<exp를 더하여 1을 붙여 주었고, 음수인 경우를 고려하여 sign이 1인 경우 two’s complement 계산을 하였다. 마지막으로 코드 상에서 frac에는 single floating point가 int로 casting된 결과가 남아있게 되었다. 이런 식으로 코드를 구성하니, 원하는 기능을 하는 것을 확인할 수 있었다.