1. float_neg

argument도 single-precision floating point value로 여겨진다고 문제에서 주어졌기 때문에 단순하게 부호를 나타내는 최상위 비트만 바꾸어주면 됩니다. 그런데 주의해야할 것은, NaN입니다. argument가 NaN인 경우 argument를 그대로 리턴해야 하므로 if문을 이용해서 NaN이라면 uf를 그대로 리턴하면 됩니다.

NaN인지 확인하기 위해서 0x7F800000과 uf를 &연산을 해줍니다. 이렇게하면 exp에 대응되는 8개의 비트를 제외하고는 모두 0이되고, 대응되는 8개의 비트는 uf의 대응되는 비트가 1인 경우만 1이 됩니다. 마찬가지로 0x007FFFFF와 uf를 &연산해주면 uf의 frac부분의 비트가 1인 경우만 1이되고 나머지는 0이됩니다.

NaN은 exp의 모든 비트가 1이고, frac이 0이 아니면 되므로 if문의 조건식을 저런식으로 작성해주면 됩니다.

그리고 NaN이 아닌 경우는 0x80000000와 uf에 ^를 해주면 uf의 최상위 비트가 1인 경우 0이, 0인 경우 1이 되고 나머지 비트는 바뀌지 않으므로 부호만을 바꿀 수 있게 됩니다.

2. float_i2f

int로 들어오는 argument를 single-precision floating point value로 바꾸는 문제입니다. 즉 주어진 정수 x에 대해 s, exp, frac를 얻어내면 됩니다. 먼저 s는 argument가 signed int이므로 s는 X의 MSB와 같습니다. 그래서 x & 0x80000000를 통해 바로 얻어낼 수 있습니다. 그리고 주어진 x가 음수이면 x를 –x로 바꾸어줍니다. 뒤에서 logical shift를 편하게 사용할 수 있기 때문입니다.

이제 exp를 구해봅시다. exp = E + Bias로 정의되므로 1.xxxxxxx…로 normalize하여 E를 얻어내야합니다. while문을 이용하여 1번비트, 2번비트, … 순으로 체크하며 1이 나타나는 가장 큰 비트를 찾습니다. single-precision floating point value의 Bias는 2^7-1=127이므로 Bias에 E를 더해서 exp를 얻어냅니다.

이제 남은 것은 frac을 얻어내는 것입니다. frac은 23비트를 가집니다. 따라서 E가 23이하인 경우는 frac에 수를 다 넣을 수 있기 때문에 rounding이 필요하지 않습니다. 그래서 이경우는 leftmost bit인 E번 비트를 기준으로 LSB 방향의 비트들을 frac에 저장해주는 것으로 frac이 구해집니다.

문제는 E가 24이상인 경우입니다. 이 경우는 rounding이 필요합니다. 일단 frac에 E번 비트 기준으로 LSB방향의 23개의 비트를 저장해줍니다. 그리고 G에 Guard bit인 frac & 1을 저장해줍니다. 그리고 R에 Round bit인 (x>>(E-24)) & 1을 저장해줍니다. 이제 Round bit 기준으로 LSB 방향의 모든 비트를 bitwise OR 연산으로 묶어 S에 저장해줍니다. 이렇게 하면 S에는 Sticky bit가 저장됩니다. 그리고 Round up condition인 R과 S가 1인 경우, S가 0이고 R과 G가 1인 경우이면 frac에

1을 더해주는 것으로 rounding을 해줍니다. 그런데 이때 frac의 23개의 비트가 모두 1인 경우는 frac의 범위를 넘어가기 때문에 이경우 exp의 값까지 바뀌게 됩니다. 따라서 frac에 1을 더했을 때 frac의 24번비트가 1인 경우, 즉, frac 범위를 초과하는 경우는 exp에 1을 더해주고 frac을 0으로 바꿔줍니다.

exp에 들어 있는 값은 exp에 대응되는 그냥 수이기 때문에 frac의 사이즈인 23만큼 왼쪽으로 shift를 해주고 bitwise OR로 s, exp<<23, frac을 묶어주면 답은 s | (exp<<23) | frac이 됩니다.

그런데 여기서 고려해주어야할 2가지 코너 케이스가 존재합니다. 첫번째는 0입니다. x가 0이면 return value도 0이어야 합니다. 하지만, exp가 127이 되어 return value가 0이 될 수 없습니다. 따라서 처음에 0을 리턴하도록 예외처리를 해줍니다.

두번째는 tmin, 즉 0x80000000입니다. 이경우 s를 구하고 x의 부호를 바꿔주는 과정에서 이 값은 부호를 바꿔줄 수 없기 때문에(|tmin| = |tmax|+1) 전처리를 통해 미리 답을 구해주어야 합니다. tmin의 경우 음수이므로 s=1이고, E=31이므로, exp는 127+31=158이 됩니다. 또, normalize했을 때 1.000000000...이 되기 때문에 frac은 0이됩니다. 158은 2진수로 10011110이 되므로 이 케이스의 리턴 벨류는 2진수로 11001111000000000000000000000000, 즉, 16진수로 0xCF000000이 됩니다.

3. float_twice

single-precision floating point value는 exp값에 따라 normalized value(exp를 나타내는 비트가 모두 0이거나 모두 1이 아닌 경우), denormalized value(exp를 나타내는 비트가 모두 0인 경우), special value(exp를 나타내는 비트가 모두 1인 경우)로 분류할 수 있습니다. 따라서 3개의 케이스로 나눠 계산해주면 됩니다.

먼저 normalized value인 경우를 생각해봅시다. 이때는 exp = E + Bias로 정의됩니다. 그런데 2를 곱한다는 것은 결국 E값을 +1해준다는 것과 같습니다. 따라서 exp에 1을 더해주는 것으로 *2를 표현할 수 있습니다.

이제 denormalized value인 경우를 생각해봅시다. 2진수로 표현된 어떤 수에서 *2를 한다는 것은 결국 소수점을 오른쪽으로 한번 옮기겠다는 뜻입니다. 그렇다는 말은 결국 원래 어떤 수가 차지하던 비트가 n번비트~LSB까지였다고 가정하면 n+1번비트~LSB+1번비트까지로 바뀐다는 뜻입니다. 즉, 부호비트인 MSB를 제외하고 left shift를 1번 수행해주는 것으로 *2를 표현할 수 있습니다.

마지막으로 special value의 경우를 생각해봅시다. 이 경우는 frac이 모두 0이면 infinity, 그렇지 않으면 NaN을 나타냅니다. 그런데 NaN인 경우는 그대로 NaN를 리턴해야합니다. 또, infinity의 경우는 *2를 해도 그대로 infinity여야 하므로(부호변화x) 결국 special value인 경우는 argument인 uf를 그대로 리턴해주면 됩니다.

4. float_abs

이 문제는 매우 간단합니다. 부호비트만 0으로 바꿔주는 것으로 문제가 해결됩니다. 단, 주의해야 할 것은 역시 코너 케이스입니다. uf가 NaN이라면 uf를 그대로 리턴해주어야합니다. 따라서 if문을 이용하여 NaN의 조건인 exp의 모든 비트가 1이고 frac이 0이 아닌 경우를 예외처리해줍니다. 나머지 케이스는 부호비트가 0이고 나머지는 모두 1인 수 0x7FFFFFFF와 bitwise and를 해주어 부호비트만을 0으로 바꿔 리턴해주면 됩니다.

5. float_half

float_twice를 풀때처럼 케이스를 나눠주면 됩니다. special value의 경우 NaN은 그대로 리턴해야하고 infinity는 0.5를 곱해도 infinity여야하므로 uf를 그대로 리턴해줍니다.

그리고 normalized와 denormalized인 경우를 처리해주어야 하는데 이 부분이 float_twice와 조금 다릅니다. normalized중에서 exp가 2진수 00000001보다 큰 경우는 단순히 exp에서 1을 빼주는 것으로 해결할 수 있습니다.

그런데 exp가 2진수 00000001인 경우와 denormalized, 즉 exp가 0인 경우는 rounding을 해주어야 합니다. 먼저 exp의 마지막 비트를 frac으로 옮깁니다. 그리고 frac의 최하위 두 비트가 모두 1인 경우(frac & 3 == 3) frac에 1을 더하여 rounding을 해줍니다. 그다음 right shift를 해줍니다. 즉, 0.5를 곱해줍니다. 이렇게 하면 exp가 2진수 00000001인 경우와 denormalized value인 경우도 해결할 수 있습니다.