SFP.c 설명

2020314315 박지원

우선, 각 함수들에 대해 설명하기에 앞서 정의해둔 매크로 상수들에 대해 설명하겠습니다.

MAX: 0 11110 1111111111 으로, sfp로 나타낼 수 있는 가장 큰 수

MIN: 1 11110 1111111111 으로, sfp로 나타낼 수 있는 가장 작은 수

D_MAX: sfp가 표현할 수 있는 denormalized value중 가장 큰 값에 근사

D_MIN: sfp가 표현할 수 있는 denormalized value중 가장 작은 값에 근사

POS_INF: 0 11111 0000000000 으로, sfp에서의 $+\infty$

NEG_INF: 1 11111 0000000000 으로, sfp에서의 $-\infty$

NAN: 1 11111 1111111111 으로, NAN

TMAX: 32비트 2의 보수의 MAX(0x7fffffff)

TMIN: 32비트 2의 보수의 MIN(0x80000000)

BIAS: 15, sfp에서의 BIAS

1. sfp int2sfp(int input)

매개변수로 들어오는 input을 sfp로 바꾸는 함수입니다.

Input이 MAX보다 크거나 MIN보다 작다면, POS_INF와 NEG_INF을 반환해주면 됩니다.

Input이 MIN과 MAX사이의 값이 아니라면, NAN을 반환해주면 됩니다.

먼저, denormalized value를 처리해줘야 합니다.

Int의 denormalized value는 0밖에 없으므로 input이 0이면 0(0 00000 0000000000)을 반환해주면 됩니다. 이러한 과정을 거치면 normalized value를 제외한 모든 경우가 걸러집니다.

Input을 2진수로 나타냈을 때 몇자리로 나타내어지는지를 계산하여 cnt에 저장해줍니다.

이 때, $cnt-1$이 E이고 $E=exp-BIAS$이므로 exp는 $cnt-1+BIAS$로 나타낼 수 있습니다.

여기서 exp는 제일 오른쪽 5비트 안에 모두 나타내어지므로 왼쪽으로 10비트만큼 shift 하여 결과가 될 res에 기록해줍니다.

M은 implied 1을 포함하여 11비트로 구성되기 때문에 cnt가 11보다 크다면 $cnt-11$만큼 M을 오른쪽으로 shift 해주고(round to zero), cnt를 11로 바꿔줍니다.

초반에 normalized value가 아닌 값들을 모두 제외했으므로 이 수는 무조건 normalized 이므로 implied 1이 있습니다. 그런데 이것을 sfp에 기록할때는 말 그대로 implied이기 때문에 해당 비트를 삭제해줘야 합니다.

M = M & ~(1 << (--cnt)); 를 하여 cnt-1 번째 비트를 삭제해줍니다.

그러고 난 후 M 을 왼쪽으로 10 – cnt 비트만큼 shift 하여 res에 기록해주면 됩니다.

2. int sfp2int(sfp input)

매개변수로 들어오는 sfp를 int로 바꿔주는 함수입니다.

Input이 POS_INF이면 TMAX를, NEG_INF나 NAN이면 TMIN을 반환해줍니다.

Sfp의 가장 왼쪽 비트가 사인 비트이기 때문에 (input >> 15) & 1로 사인 비트를 확인할 수 있습니다. 나중에 E를 확인할 때의 편의를 위하여 음수라면 사인 비트를 0으로 바꿔줍니다.

Sfp의 M 부분을 담당하는 10개 비트의 값을 계산하기 위해 for문을 돌며 각 비트가 1일 경우에 해당 자리의 지수 값을 곱하여 더해줍니다. 그 후 implied 1 또한 계산해줘야 하기 때문에 1을 더해줍니다.

Input을 양수로 바꿔줬기 때문에 (input >> 10) 해주면 exp가 나옵니다.
E = exp – BIAS이므로 E = (input >> 10) – BIAS 입니다.
이제 부호와 M과 E를 모두 계산했으므로 부호 * M * Pow(2, E)를 반환해주면 됩니다.

3. sfp float2sfp(float input)

Int2sfp 함수의 초반부와 마찬가지로 MIN <= val <= MAX 범위를 벗어난 값이라면 부호에 알맞게 INF를 반환해주고, MIN <= val <= MAX에도 속하지 않는다면 NAN을 반환해줍니다.

Float가 매개변수로 들어왔기 때문에 해당 수는 이미 부동소수점으로 나타내어지고 있습니다. 따라서, 그 부동 소수점을 그대로 끌어와서 쓰면 되기 때문에 int* = (int*)&input으로 input의 값 비트들에 shift로 접근할 수 있도록 하였습니다. E를 구하기 전에, default 값으로 E를 1 – BIAS로 초기화 해뒀습니다. 이후, input으로 들어온 값이 D_MAX보다 크다면 해당 수는 normalized value이므로 float의 E를그대로 가져와줬습니다. Exp = E + BIAS이므로 exp를 구해준 후 exp를 왼쪽으로 10비트만큼 shift시켜 res에 기록해줍니다.

input / Pow(2, E) – (input > D_MAX ? 1 : 0); 이 input에서 fracVal입니다..
해당 값을 비트로 바꿔줘야 하기 때문에 소수를 이진수로 바꾸는 방식(한자리를 더할 때마다 fracVal의 값을 2씩 곱해가며 fracVal가 1이 넘어간다면 1을 Mantissa에 기록)을 통해 M을 11자리 이진수로 만들어줍니다. 이후, implied 1을 제거해준 다음에 res에 기록해주면 됩니다.

4. float sfp2float(sfp input)

Sfp2int의 초반부와 마찬가지로, POS_INF이면 TMAX를, NEG_INF나 NAN이라면 TMIN을 반환해줍니다.

이 함수에서도 sfp2int와 같은 방식으로 사인 비트를 찾아줘서 부호를 저장할 수 있습니다. 이 함수에서 또한 나중에 E를 확인할 때의 편의를 위하여 음수라면 사인 비트를 0으로 바꿔주었습니다.

또한, M부분의 값 또한 sfp2int와 같은 방식으로 구할 수 있습니다. 그러나, int로 바꿔주는 것이

아니라 float로 바꿔주는 것이기 때문에 input이 1023(0 00000 1111111111: denormalized value 중 0이 아닌 가장 작은 값)보다 클 경우에만 implied 1을 M에 더해줘야 합니다.

마지막으로, input이 1023보다 작다면, input은 denormalized value를 나타내는 sfp이므로, E = 1 – BIAS입니다. 그러나, 1023보다 input이 클 때는 normalized value인데, 이 때의 E는 (input >> 10) – BIAS 입니다.

이렇게 부호와 M과 E를 모두 구해줬으면, 부호 * M * Pow(2, E)를 반환해주면 됩니다.

Add와 Mul을 구현하기에 앞서, add와 mul에는 round to even 방식의 반올림을 적용해야 하기 때문에 shift_rte(unsigned long long m, int n)이라는 함수를 정의해줬습니다. 해당 함수는 round to even을 적용하면서 m을 오른쪽으로 n비트만큼 shift해주는 함수입니다.

---

5. sfp sfp_add(sfp a, sfp b)

일단, a와 b로부터 s, exp, m을 각각 가져와야 합니다.

S는 (sfp >> 15) & 1을 통하여 확인할 수 있고,
exp는 sfp의 사인 비트를 지우고 오른쪽으로 10비트만큼 shift 해주면 구할 수 있습니다.
마지막으로 m은 a를 왼쪽으로 6비트만큼 shift(s와 exp를 없애줍니다.) 한 후에 오른쪽으로 다시 6비트만큼 shift 하면 됩니다.

이렇게 가져온 exp가 0이라면 해당 수는 denormalized value로 1 – BIAS를 E로 가지고, 0 이 아니라면 normalized value로 exp – BIAS를 E로 가집니다.

부동소수점의 덧셈을 위하여 E가 더 작은 수의 m을 오른쪽으로 E_BIG – E_SMALL 만큼 shift 하여 두 수의 E를 같은 수로 맞춰줍니다.
이후 두 수의 부호가 같다면 m1과 m2를 더해 덧셈 연산을 진행해주고, 두 수의 부호가 같지 않다면 m_big – m_small을 통해 덧셈 연산을 진행해줍니다.

각 연산이 끝난 후에는 앞서 정의한 shift_rte 함수를 이용해 1.xxxxx형식을 갖도록 normalization 해줍니다. 이 과정에서 E는 당연히 변하게 됩니다.

이렇게 구한 E가 15보다 크다면, 부호에 따라서 POS_INF 또는 NEG_INF를 반환해줍니다.

E가 1 – BIAS 이하라면, m1을 오른쪽으로 (1 – BIAS) – E 만큼 shift(round to even이기 때문에 shift_rte를 사용해줘야 합니다.) 한 후, 사인 비트와 M을 res에 기록하여 반환해줍니다.

E가 15보다 크지도 않고 1 – BIAS 이하도 아니라면, Exp = E + BIAS를 구해주고, res에 사인 비트와 M, Exp를 모두 기록하여 반환해주면 됩니다.

6. sfp sfp_mul(sfp a, sfp b)

Sfp_add에서와 똑같이 S, exp, shift를 구해주고,

결과값의 E를 구하는 것은 아주 쉽습니다. 그냥 E1 + E2를 해주면 끝입니다.

곱셈의 편의를 위하여 m을 둘 다 1.xxxxx꼴로 바꿔줍니다(이후 최소 몇 자리 수가 나오고 최대 몇 자리 수가 나올지 예측 가능합니다.). 그 후 m1 * m2를 하여 결과값의 M을 구해줍니다. 그런데, 11자리 2진수 두개를 곱한 것이기 때문에 오른쪽으로 최소 10비트 만큼 shift, 최대 14비트 만큼 shift해줘야 합니다. (이 과정에서 당연히 E는 변하게 됩니다.)

이렇게 구한 E가 15보다 크다면 부호에 따라 POS_INF나 NEG_INF를 반환해주면 됩니다.

S를 기록해준 후,

E < −14보다 작고 M이 0보다 클동안 M을 오른쪽으로 계속 shift 해주며 E를 더해줍니다.

이렇게 한 뒤에도 E가 −14보다 작다면 이 수는 sfp로 나타내기에는 너무 작은 수이므로 0을 반환해줍니다.

E가 −14이고 M >> 10이 1이라면 이 수는 denormalized value로, M을 res에 기록해주고 반환해주면 됩니다.

위의 경우에 모두 해당하지 않는다면 이 수는 normalized value로, 앞서 구한 E를 통해 Exp를 구해준 뒤 M과 E를 기록하여 반환해주면 됩니다.