[Assignment1] Small Floating Point

2019312582최은서

`sfp getNaN()` returns NaN value in sfp.

`sfp getinfinitP()` returns plus infinity value in sfp.

`sfp getinfinitM()` returns minus infinity value in sfp.

`int getSign(sfp input)` returns 0 or 1 according to the input value's sign.

`int getExp(sfp input)` returns the exponents in integer.

`sfp getFrac(sfp input)` returns frac bits.

`void getInfo(sfp input, int *sign, int *exp, int *E)` sets the sign, exp, E value.

`int specialCheck(sfp input)` checks the input value's special case.

`void applySign(sfp* input, int sign)` apply the sign value at MSB.

`void applyExp(sfp* input, int exp)` apply the exponent at following 5 bits.

`void applyFrac(sfp* input, sfp c)` apply the frac at last 10 bits.

`sfp int2sfp(int input)`

1. input이 0인 경우 자료형만 sfp로 하여 0을 그대로 리턴한다.
2. maxsfp = 65504 이고 minsfp = -65504이므로 인풋값이 이들보다 크거나 작으면 각각 +infinity 혹은 -infinity 리턴한다.
3. 정수에서 변환하는 경우이므로 0이 아닌 demormalized인 경우는 없다.
4. 음수인 경우에는 2의 보수법을 취해서 양수로 바꾼 후, 부호 적용, 양수인 경우는 생략한다.
5. E의 값과 exp 의 값을 구한 후 적용한다.
6. frac 부분에서 leading 1인 경우 해당 부분을 빼주고 result에 적용한다.

`int sfp2int(sfp input)`

1. Tmax와 Tmin을 구하고, 인풋값이 special case인 경우 해당 값을 리턴한다.
2. Special case가 아닌 경우, 인풋값의 exp를 구해서 15보다 작은 경우는 0을 리턴한다. integer로 변환하는 것이기 때문에 frac을 고려하지 않는다.
3. normalcase인 경우, result에 인풋값의 frac 부분을 대입해준다.
4. 마지막으로 부호를 확인하는데, -일 경우 2의 보수법을 통해 정확한 값을 리턴한다.

`sfp float2sfp(float input)`

1. float인 인풋값의 비트를 unsigned integer save 에 우회하여 저장한다.
2. maxsfp와 minsfp의 값을 각각 설정해주고, 인풋값이 해당 수보다 크거나 같을 때 각각 +infinity 혹은 -infinity를 반환한다.
3. 그 외의 경우, float의 비트를 분석해 float exp를 구하고, 이 값을 이용해서 sfp의 exp도 구한 다음, result 에 적용한다.
4. frac부분은 float의 frac부분을 앞에서 10비트를 잘라서 대입해준다.
5. 마지막으로 부호를 체크하여 input값이 minus인 경우 MSB에 1을 넣어주고 리턴한다.

`float sfp2float(sfp input)`

1. 리턴할 result 변수를 선언하고, unsigned integer save를 통해 우회하여 32비트를 디자인한다.
2. 먼저 부호를 확인하여 save의 MSB에 적용한다.
3. input의 exp와 E를 구하고, float일 경우의 exp를 새로 구한 후 save의 MSB 다음 8비트 부분에 적용한다.
4. input의 frac 부분을 float의 마지막 23비트 부분에 적용한다. float는 sfp의 범위를 모두 포함하기 때문에 infinity값이나 NaN의 값이 들어와도 적용 가능하다.

`sfp sfp_add(sfp a, sfp b)`

1. specialCheck 함수를 통해서 input값들이 각각 special case에 해당하는지 확인한 후, 그에 알맞는 값을 return한다.
2. Special case가 아닌 경우, input a와 input b의 sign, exp, E를 각각 getInfo함수를 통해 구하고, frac값을 계산할 calA, calB, sums를 선언한다.
3. a와 b의 frac을 각각 구하고, 만약 leading 1일 경우 적절한 곳을 1로 바꾸어 leading 1을 표현한다.

4. exp가 다를경우와 같은경우를 나누어서 계산한다. 그 안에서는 값의 부호를 생각하여 계산한다. a의 exp가 더 클 경우는 다음과 같다.

   4-1. , b의 exp를 키우고, frac을 right shift연산을 한다. 이곳에서는 값을 잃지 않기 위해 a의 frac을 left shift 하였다.(calA)

   4-2. calA와 calB를 더한 값을 Sum에 저장하고, leading 1인 경우 알맞은 부분의 비트를 0으로 바꿔준 후 result에 대입한다. exp는 a의 것을 대입하되, 정규화 할때 바뀌는 경우가 있다면 고려해주어야 한다.

   4-3. 두 값의 부호가 같으면 둘다 음수인 경우에만 부호비트를 적용시켜준다. 다른 경우에는 값을 비교하여 적절한 MSB를 적용한다.

   b의 exp가 더 큰 경우도 같은 방식이다.

5. a와 b의 exp가 같은 경우는 exp를 같게 맞출 필요가 없으므로 그대로 frac부분을 연산해준 후 정규화하여 result에 적용하고 부호를 맞춰준다.

`sfp sfp_mul(sfp a, sfp b)`

1. specialCheck 함수를 통해서 input값들이 각각 special case에 해당하는지 확인한 후, 그에 알맞는 값을 return한다.
2. Special case가 아닌 경우, input a와 input b의 sign, exp, E를 각각 getInfo함수를 통해 구하고, frac값을 계산할 calA, calB, Mul을 선언한다.
3. a와 b의 frac을 각각 구하고, 만약 leading 1일 경우 적절한 곳을 1로 바꾸어 leading 1을 표현한다.
4. 두 input의 부호가 다른 경우 MSB에 1을 적용해주고, 같은 경우에는 양수이므로 0이다.
5. calA와 calB를 곱한 값인 Mul을 leading 1을 고려하여 result에 적용시킨다.
6. 정규화를 고려하여 exp를 적절히 수정시킨 후 result에 적용시킨다.

`char* sfp2bits(sfp result)`

1. sfp가 16비트이므로, return할 변수인 bitStream에 17바이트 메모리를 할당한다.
2. input을 비트 단위로 분석하여 각 비트가 1인 경우에는 문자 '1'을, 0인 경우에는 문자 '1'을 bitStream의 각 인덱스에 저장하고, 마지막 인덱스에는 끝을 나타내는 '\0'을 적용한다.
3. bitStream을 return 한다.