1. C++, g++ (GCC) 13.2.1 20230801

2. Тесты проходит

3.

Округление:

У меня округление к ближайшему, целому.

Я беру число x, которое мне нужно округлить, и хочу округлить его на последние k битов(удалить k битов и округлить последний бит получившегося числа).

Ну, тогда результат, округленный к нулю это:
res = x » k
Дальше выпишу последние k битов, чтобы понять, куда мне нужно округлять.
toRound = x & ((1 « k) — 1)

Если toRound > (1 « (k — 1)), то я хочу округлить число к ближайшему вверх(по модулю) и вернуть res + 1
Если toRound < (1 « (k — 1)), то я хочу округлить число к ближайшему вниз(по модулю) и вернуть res
Иначе я смотрю на последний бит res, если он равен 0, то возвращаю res, иначе res + 1

Это округление в двоичной системе, округление при выводе в десятичной реализовано аналогично.

Fixed:

Я перевожу числа из 16-ричной системы в десятичную и складываю их в long long.

Операция сложения:
Просто складываю a + b, биты сложатся по правилам математики, если переполнение из дробной части произойдет в целую ничего не сломается, потому что биты целой части идут сразу после битов дробной, после операции обрезаю все биты, кроме последних A+B (убираю переполнение)

Операция вычитания:
Пусть мы хотим сделать a — b это равносильно a + (-b) .
Заменяю b на его дополнительный код и делаю сложение.

Операция умножения:
Пусть мы хотим перемножить x * y.

Sign = sign(x) ^ sign(y)
Считаю знак, как ксор знаков исходных чисел. Если число отрицательное, то заменяю его на положительное(дополнение)

Форматы исходных чисел A:B, после перемножения мы получим число, формат которого 2A:2B, последние B бит я отбрасываю с округлением, затем первые A отбрасываю как переполнение.

Так как мы перемножали модуля, то в конце нужно добавить знак к результату, для этого я смотрю на sign и, если он равен одному, заменяю ответ на его дополнение.

Операция деления:
Считаю sign аналогично умножению.
Заменяю a и b на дополнение, если они отрицательные.

Заметим, что a // b возвращает целую часть от деления a / b.
Если сдвинуть a на 1 бит влево, то последний бит a // b будет равен первому биту, который должен был получиться после точки при делении a / b.

В общем случае, если сдвинуть a на k битов влево, то последние k битов a // b будут равны k битам после точки при делении a на k

Мы хотим сделать a / b и получить B битов после точки, для этого просто сдвинем a на B битов влево. (в реализации я сдвигаю a на b + 16 битов, а затем убираю последние 16 битов и округляю число, чтобы получить неплохую точность).

Потом вывожу результат в нужном формате.

Floating:

Я перевожу числа из 16-ричной системы и записываю их в long long.
Когда мне нужно как-то работать с числом, я разбиваю его на 3 числа: знак, экспоненту и мантиссу. Если число денормализованное, приведем его к нормализованному, экспонента в процессе вычислений может стать меньше экспоненты нормализованных чисел.

Операция сложения:
Разделяю числа на знак, экспоненту, мантиссу.

Дальше я передаю 2 числа в функцию, которая приводит числа к одной экспоненте. Если разность в экспонентах меньше 23 для single и 10 для half, то мы можем просто подвинуть мантиссу числа, экспонента которого больше влево, чтобы экспоненты стали равны и вернуть эти числа в функцию сложения. Если разность в экспонентах больше 23(10 для халф), очевидно, что после сложения мантисс результат нужно будет привести к 23(10) знакам и мантисса числа с меньшей экспонентой просто отбросится, а мантисса числа с большей экспонентой округлится, если нужно (просто проверяю, что разность экспонент меньше, например 30, если меньше, то могу привести к одной экспоненте, округлить и явно сложить, если нет,).

Теперь я имею 2 числа с одинаковой экспонентой.

Чтобы отследить, где будет заканчиваться мантисса суммы, я создаю 2 переменные:

x = mantA ^ (1 « lenMant)
y = mantB ^ (1 « lenMant)

В них первый бит — единица целой части.

Так как вычисления происходят в знаковом типе, то если одно из чисел отрицательное, я заменяю его мантиссу на отрицательное число и просто складываю мантиссы.

Если на данном шаге результат получился 0, я возвращаю 0, чтобы потом не дописать единицу целой части к нулю.

В результате я смотрю, где находится первая единица — бит целой части. И ксорю его с единицей, чтобы в ответе осталась только мантисса. Так как я знаю позицию первой единицы(целой части), то я знаю и сколько лишних знаков у меня получилось в мантиссе. Я снова округляю результат на последние max(0, firstOne — mantLength) битов, чтобы осталась мантисса длины 23 (10 для халф). Это мантисса результата сложения.

Экспонента результата это экспонента чисел, когда мы привели их к одной экспоненте + firstOne(res) — max(firstOne(x), firstOne(y))
Почему такая формула для экспоненты работает:
Посмотрим на первый единичный бит складываемых чисел. Если перепонление не произошло, то он равен первому единичному биту результата, если переполнение произошло, то первый единичный бит результата > первого единичного бита максимального из исходных чисел, значит число переполнилось и надо его сдвинуть на firstOne(res) — firstOne(max(x,y)) битов вправо.

Знак результата это знак при мантиссе т. к. мы складывали в знаковом типе, а саму мантиссу заменяем на llabs(mant).

Дальше просто возвращаю из знака, новой экспоненты и новой мантиссы собираю число, как в формате ввода и записываю его в long long.

Операция вычитания:
Просто ксорю бит знака второго числа с единицей. Если число было положительным, оно станет отрицательным, если оно было отрицательным, то станет положительным. И просто выполняю сложение (сложение с отрицательным числом мы делать научились)

Операция умножения:
Я разбиваю числа на знак, экспоненту мантиссу.

Чтобы отследить, где будет заканчиваться мантисса произведения, я создаю 2 переменные:

x = mantA ^ (1 « lenMant)
y = mantB ^ (1 « lenMant)

В них первый бит — единица целой части.

Тогда
(Можно представить x и y как числа с фиксированной точкой и форматом A:23 или A:10, тогда умножение для них будет работать как для обычных фикседов)

mantRes = mantX * mantY

Экспонента — это сумма экспонент исходных чисел.

Теперь я аналогично сложению ищу первую единицу и убираю ее — это целая часть.

Аналогично сложению рассмотрим первый единичный бит результата, посмотрим на то, как он изменился относительно первого единичного бита максимального из чисел, увеличим экспоненту на firstOne(res) — max(firstOne(x), firstOne(y))

Знак произведения мы можем посчитать как ксор исходных знаков.

В результате я передаю знак, экспоненту и мантиссу в функцию, которая собирает из них число из 32 (16 для халф) бит. Так же она проверяет, что экспонента не стала слишком большой и, если это произошло, возвращает бесконечность.

Полученное этой функцией число я возвращаю на выход.

Операция деления:
Почти всё аналогично умножению.
Ввожу x и y, дальше, как и в фикседах, сдвигаю a на длину мантиссы + 16 битов для точности и делаю

res = x / y

Экспонента ответа — это разность экспонент исходных чисел.

Затем я удаляю лишние биты, которые добавлял для точности и округляю на них результат. Так же убираю бит целой части.

Аналогично сложению и умножению рассмотрим первый единичный бит результата, посмотрим на то, как он изменился относительно первого единичного бита максимального из чисел, увеличим экспоненту на firstOne(res) — max(firstOne(x), firstOne(y))

Обрабатываю случаи по типу деления на 0, на бесконечность и.т.д. и, если нашелся какой-то особый случай, то возвращаю ответ.

Знак это ксор знаков исходных чисел.
Дальше восстанавливаю ответ по знаку, экспоненте и новой мантиссе и возвращаю его.