Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт по лабораторной работе №2

«Арифметические операции с числами с плавающей точкой»

Выполнил:

cтудент гр. 153503

Вергасов Вадим Михайлович

Руководитель:

Калиновская Анастасия Александровна

Минск 2023

1. **Цель работы**

Изучить принципы и особенности выполнения арифметических операций в двоичном коде с числами с плавающей точкой. Познакомиться со средствами, которые применяются в реализации арифметико-логического устройства. Получить практические навыки по программированию этих средств.

1. **Постановка задачи**

Написать программу эмулятора АЛУ, реализующего операции сложения, вычитания, умножения и деления с плавающей точкой над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

1. **Теоретические сведения**

Основные положения

В формате с фиксированной точкой, в частности в дополнительном коде, можно представлять положительные и отрицательные числа в диапазоне, симметричном на числовой оси относительно точки 0. Расположив воображаемую Разделяющую точку в середине разрядной сетки, можно в этом формате представлять не только целые, но и смешанные числа, а также дроби.

Однако такой подход позволяет представить на ограниченной разрядной сетке множество вещественных чисел в довольно узком диапазоне. Нельзя представить очень большие числа или очень маленькие. При выполнении деления двух больших чисел, как правило, теряется дробная часть частного.

Стандарт IEEE формата с плавающей точкой

Для унификации формата представления чисел с плавающей точкой, что является необходимым условием переносимости программного обеспечения, Институтом инженеров по электротехнике и радиоэлектронике IEEE разработан стандарт 754. В последнее десятилетие практически все процессоры и арифметические сопроцессоры проектируются с учетом требований этого стандарта.

Стандарт специфицирует два варианта формата: 32-битовый — обычной точности представления и 64-битовый — удвоенной точности представления. В первом формате поле порядка занимает 8 бит, а во втором - 11 бит. Стандарт регламентирует использование числа 2 в качестве неявно заданного значения основания характеристики. Помимо основных, в стандарте предусмотрены два расширенных варианта форматов обычной и удвоенной точности, конкретная спецификация которых зависит от реализации вычислительной системы. Расширенные форматы позволяют включать дополнительные биты в поле порядка (расширение диапазона представления) в поле мантиссы (повышение точности представления). Расширенные форматы предназначаются для промежуточных вычислений. За счёт повышения точности снижается вероятность появления ошибок округления, а при расширении диапазона снижается вероятность появления ошибки переполнения.

Сложение и вычитание

Алгоритмы выполнения операций сложения и вычитания в формате с плавающей точкой сложнее, чем аналогичные алгоритмы для чисел в формате с фиксированной точкой. Связано это, в первую очередь, с необходимостью выравнивания порядков операндов. Алгоритм включает четыре основных этапа.

1. Проверка на нуль.

2. Сдвиг мантисс для выравнивания порядков.

3. Суммирование или вычитание мантисс.

4. Нормализация результата.

Умножение и деление

При работе с числами в формате с плавающей точкой алгоритмы умножения и деления оказываются проще алгоритмов сложения и вычитания.

Дополнительные разряды

Перед началом выполнения любых арифметических операций операнды загружаются в регистры АЛУ. Регистры, предназначенные для работы с мантиссами, могут иметь большую разрядность, чем поле мантиссы, предусмотренное форматом представления, плюс один неявно заданный старший разряд. Как правило, дополнительные разряды размещаются справа, т.е. имеют вес, меньший веса самого младшего разряда представления. При загрузке регистров эти разряды заполняются кодом 0.

Округление

Результат любой операции над мантиссами операндов, как правило, формируется в регистре АЛУ, имеющем большую разрядность, чем предусмотрено форматом хранения. Поэтому при сохранении результата необходимо тем или иным способом выполнить его округление.

Особенности выполнения арифметических операций в соответствиисо стандартом IEEE

Стандартом IEEE 754 регламентируется не только формат с плавающей точкой для представления чисел, но и определенные правила выполнения арифметических операций над ними, что позволяет получать одинаковые результаты при реализации вычислительных алгоритмов на разных аппаратных и программных платформах. Один из аспектов стандарта, касающийся технологии округления, мы уже рассматривали выше. В этом разделе мы остановимся на трех других аспектах — операциях с бесконечно большими величинами, необычными числовыми величинами (типа NaN) и ненормализованными числами

Бесконечно большие величины

Арифметика бесконечных величин трактуется как частный случай арифметики вещественных чисел, в котором бесконечно большая величина интерпретируется следующим образом: -∞ < (любая конечная числовая величина ) < ∞ .

Простые и сигнализирующие значения типа NaN

Значения типа NaN представляют, по существу, не числа, а символы, "втиснутые" в формат с плавающей точкой. Различается два подтипа NaNзначений — сигнализирующее и простое. Появление в качестве операнда арифметической операции сигнализирующего NaN-значення генерирует исключительную ситуацию. Сигнализирующими NaN-значениями можно заполнять неинициализированные переменные и использовать их в специализированных арифметических расширениях, не вписывающихся в стандарт.

Ненормализованные числа

В формате с плавающей точкой, регламентированном стандартом IEEE 754,предусматривается возможность использования ненормализованной мантиссы для того, чтобы можно было обрабатывать ситуации потери значимости порядка. Когда порядок результата оказывается очень маленьким (большой по абсолютной величине порядок со знаком "минус"), этот результат можно денормализовать, сдвигая мантиссу вправо и одновременно увеличивая значение порядка до тех пор, пока число не будет "втиснуто" в разрядную сетку формата хранения.

1. **Программная реализация**

В качестве средства для написания программы использовался язык программирования C++.

Далее представлено несколько примеров работы программы

.

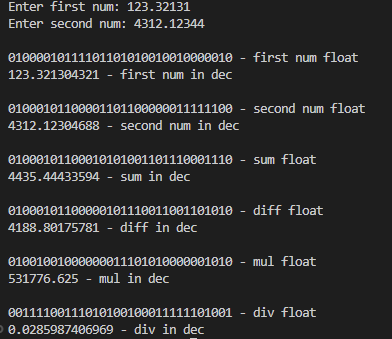


Рисунок 1 - Положительные числа

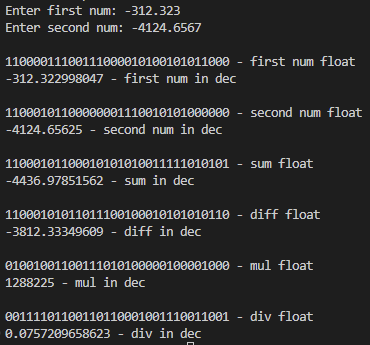


Рисунок 2 - Отрицательные числа

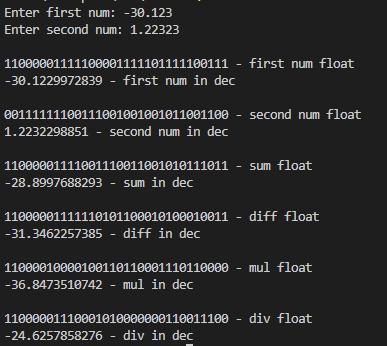


Рисунок 3 – Числа разных знаков

1. **Выводы**

В ходе лабораторной работы были изучены принципы реализации и свойства арифметических операций в двоичном коде с числами с плавающей точкой. Были изучены различные методы сложения, вычитания, умножения и деления для дробных чисел. На практике был разработан эмулятор арифметико-логического устройства (АЛУ), который пошагово выполняет операции сложения, вычитания, умножения и деления на числах с плавающей точкой.

**Приложение 1. Исходный код программы.**

#include <algorithm>

#include <bitset>

#include <cmath>

#include <iostream>

#include <sstream>

std::string bin\_diff(std::string num1\_s, std::string num2\_s,

bool is\_print = false) {

std::string res;

int carry = 0, tmp;

std::reverse(num1\_s.begin(), num1\_s.end());

std::reverse(num2\_s.begin(), num2\_s.end());

for (size\_t i = 0; i < num1\_s.size(); i++) {

if (num1\_s[i] == '.') {

res += '.';

continue;

}

tmp = (num1\_s[i] - num2\_s[i] - carry);

if (tmp < 0) {

tmp = std::abs(tmp);

carry = 1;

res += (tmp % 2) + '0';

} else {

res += tmp + '0';

carry = 0;

}

if (is\_print) {

std::cout << res << std::endl;

}

}

std::reverse(res.begin(), res.end());

return res;

}

std::string bin\_sum(std::string num1\_s, std::string num2\_s,

bool is\_print = false) {

std::string res;

int carry = 0, tmp;

std::reverse(num1\_s.begin(), num1\_s.end());

std::reverse(num2\_s.begin(), num2\_s.end());

for (size\_t i = 0; i < num1\_s.size(); i++) {

if (num2\_s[i] == '.') {

res += '.';

continue;

}

tmp = (num1\_s[i] + num2\_s[i] + carry) - 96;

carry = tmp / 2;

res += std::to\_string(tmp % 2);

if (is\_print) {

std::cout << res << std::endl;

}

}

std::reverse(num1\_s.begin(), num1\_s.end());

std::reverse(num2\_s.begin(), num2\_s.end());

if (carry) {

res += '1';

}

std::reverse(res.begin(), res.end());

return res;

}

std::string bin\_mul(std::string num1\_s, std::string num2\_s,

bool is\_print = false) {

std::string res, tmp\_res, null\_string(num1\_s.size() \* 2, '0');

res = null\_string;

if (num1\_s[0] == '0' && num2\_s[0] == '1') {

std::swap(num1\_s, num2\_s);

}

std::reverse(num1\_s.begin(), num1\_s.end());

std::reverse(num2\_s.begin(), num2\_s.end());

for (size\_t i = 0; i < num2\_s.size(); i++) {

if (num2\_s[i] == '0') {

continue;

}

tmp\_res = null\_string;

for (size\_t j = i; j < tmp\_res.size(); j++) {

tmp\_res[j] = '0';

}

for (size\_t j = i; j < i + num1\_s.size(); j++) {

tmp\_res[j] = num1\_s[j - i];

}

std::reverse(res.begin(), res.end());

std::reverse(tmp\_res.begin(), tmp\_res.end());

res = bin\_sum(res, tmp\_res);

std::reverse(res.begin(), res.end());

if (is\_print) {

std::cout << res << std::endl;

}

}

std::reverse(res.begin(), res.end());

return res;

}

std::string bin\_div(std::string num1\_s, std::string M, bool is\_print = false) {

if (M.find("1") == std::string::npos) {

return "divide by zero";

}

std::string res, tmp, A, Q, old\_a;

Q = num1\_s;

for (size\_t i = 0; i < 8; i++) {

M = M[0] + M;

}

for (size\_t i = 0; i < Q.size(); i++) {

A += '0';

}

for (size\_t i = 0; i < Q.size(); i++) {

for (size\_t j = 0; j < A.size() - 1; j++) {

A[j] = A[j + 1];

}

A[A.size() - 1] = Q[0];

for (size\_t j = 0; j < Q.size() - 1; j++) {

Q[j] = Q[j + 1];

}

old\_a = A;

if (A[0] == M[0]) {

A = bin\_diff(A, M);

} else {

A = bin\_sum(A, M);

}

if (A[0] == old\_a[0]) {

Q[Q.size() - 1] = '1';

} else {

Q[Q.size() - 1] = '0';

A = old\_a;

}

if (is\_print) {

std::cout << std::to\_string(i) + ": " << A << ' ' << Q << std::endl;

}

}

return Q;

}

void inc(std::string &num\_s) {

int carry = 1, tmp;

std::reverse(num\_s.begin(), num\_s.end());

for (size\_t i = 0; i < num\_s.size(); i++) {

tmp = num\_s[i] - '0' + carry;

num\_s[i] = tmp % 2 + '0';

carry = tmp / 2;

}

if (carry == 1) {

num\_s += '1';

}

std::reverse(num\_s.begin(), num\_s.end());

}

void dec(std::string &num\_s) {

int carry = 1, tmp;

std::reverse(num\_s.begin(), num\_s.end());

for (auto &x : num\_s) {

tmp = x - '0' - carry;

if (tmp < 0) {

carry = 1;

x = tmp \* (-1) % 2 + '0';

} else {

x = tmp + '0';

carry = 0;

}

}

std::reverse(num\_s.begin(), num\_s.end());

}

std::string int\_part\_to\_bin(std::string num\_s) {

std::stringstream cont;

int num;

std::string res;

cont << num\_s;

cont >> num;

while (num) {

res += num % 2 + '0';

num /= 2;

}

std::reverse(res.begin(), res.end());

return res;

}

std::string fract\_to\_bin(std::string fract\_part\_s, size\_t digit\_count) {

std::string res;

std::stringstream cont;

double fract\_part;

cont << fract\_part\_s;

cont >> fract\_part;

for (size\_t i = 0; i < digit\_count; i++) {

fract\_part \*= 2;

if (fract\_part >= 1) {

res += '1';

fract\_part -= 1;

} else {

res += '0';

}

}

return res;

}

std::string dec\_to\_bin(std::string num\_s) {

std::stringstream tmp\_cont;

double tmp\_num;

tmp\_cont << num\_s;

tmp\_cont >> tmp\_num;

if (tmp\_num == 0) {

num\_s = "0";

}

if (num\_s[0] == '-') {

num\_s.erase(0, 1);

}

size\_t pos = std::distance(num\_s.begin(),

std::find(num\_s.begin(), num\_s.end(), '.'));

if (pos == num\_s.size()) {

std::string res = int\_part\_to\_bin(num\_s);

res = res + "." + std::string(24 - res.size(), '0');

return res;

} else {

std::string int\_part\_s, fract\_part\_s, res;

int\_part\_s = num\_s.substr(0, pos);

fract\_part\_s = num\_s.substr(pos);

res = int\_part\_to\_bin(int\_part\_s);

std::stringstream cont;

res += '.';

res += fract\_to\_bin(fract\_part\_s, 25 - res.size());

return res;

}

}

std::string bin\_to\_float(std::string bin) {

int pos =

std::distance(bin.begin(), std::find(bin.begin(), bin.end(), '.')),

exp;

std::string exp\_s, mant, res;

if (pos == 1) {

if (bin[0] == '0') {

pos -= std::distance(bin.begin(),

std::find(bin.begin(), bin.end(), '1'));

exp = 127 + pos;

exp\_s = std::bitset<8>(exp).to\_string();

pos = std::distance(bin.begin(),

std::find(bin.begin(), bin.end(), '1'));

bin.erase(pos, 1);

mant = bin.substr(pos - 1);

} else {

exp = 127;

exp\_s = std::bitset<8>(exp).to\_string();

bin.erase(1, 1);

mant = bin;

}

} else if (pos == 0) {

exp = 127;

exp -= bin.find('1');

bin.erase(0, bin.find('1'));

exp\_s = std::bitset<8>(exp).to\_string();

mant = bin;

} else {

exp = 127 + pos - 1;

exp\_s = std::bitset<8>(exp).to\_string();

bin.erase(pos, 1);

mant = bin;

}

res = "0" + exp\_s + mant;

return res;

}

std::string dec\_to\_float(std::string num\_s) {

bool is\_neg = 0;

std::string num\_b, res;

if (num\_s[0] == '-') {

is\_neg = 1;

num\_s.erase(0, 1);

}

num\_b = dec\_to\_bin(num\_s);

res = bin\_to\_float(num\_b);

res[0] = is\_neg + '0';

return res;

}

std::string conv(std::string num\_s) {

int carry = 1, tmp;

for (auto &x : num\_s) {

if (x != '.') x = ((x - '0') ^ 1) + '0';

std::cout << x << ' ';

}

std::reverse(num\_s.begin(), num\_s.end());

for (auto &x : num\_s) {

tmp = x - '0' + carry;

x = tmp % 2 + '0';

carry = tmp / 2;

}

std::reverse(num\_s.begin(), num\_s.end());

return num\_s;

}

std::string float\_sum(std::string num1\_s, std::string num2\_s);

std::string float\_diff(std::string num1\_s, std::string num2\_s) {

std::string exp1 = num1\_s.substr(1, 8), exp2 = num2\_s.substr(1, 8),

mant1 = num1\_s.substr(9), mant2 = num2\_s.substr(9);

std::string mant;

char sign\_bit = 0;

if (num1\_s[0] != num2\_s[0]) {

num2\_s[0] = ((num2\_s[0] - '0') ^ 1) + '0';

return float\_sum(num1\_s, num2\_s);

}

if (num1\_s.find('1') == std::string::npos) {

return num2\_s;

} else if (num2\_s.find('1') == std::string::npos) {

return num1\_s;

}

if (num1\_s[0] == num2\_s[0] && (exp1 + mant1 > exp2 + mant2)) {

sign\_bit = num1\_s[0];

} else if (num1\_s[0] == num2\_s[0] && (exp1 + mant1 < exp2 + mant2)) {

sign\_bit = num2\_s[0];

}

if (exp1 + mant1 < exp2 + mant2) {

std::swap(exp1, exp2);

std::swap(mant1, mant2);

}

if (exp1 == exp2) {

mant = bin\_diff(mant1, mant2);

if (mant.find('1') == std::string::npos) {

return std::string(33, '0');

}

while (!(mant[0] - '0')) {

mant.erase(0, 1);

mant += '0';

dec(exp1);

}

if (mant.find('1') == std::string::npos) {

return std::string(33, '0');

}

} else {

if (exp1 < exp2) {

std::swap(exp1, exp2);

std::swap(num1\_s, num2\_s);

std::swap(mant1, mant2);

}

while (exp1 != exp2 && mant2.find('1') != std::string::npos) {

inc(exp2);

mant2 = '0' + mant2;

mant2.erase(mant2.size() - 1, 1);

}

if (mant2.find('1') == std::string::npos) {

return num1\_s;

} else {

mant = bin\_diff(mant1, mant2);

if (mant.find('1') == std::string::npos) {

return std::string(33, '0');

}

while (!(mant[0] - '0')) {

mant.erase(0, 1);

mant += '0';

dec(exp1);

}

}

}

return sign\_bit + exp1 + mant;

}

std::string float\_sum(std::string num1\_s, std::string num2\_s) {

if (num1\_s[0] != num2\_s[0]) {

if (num1\_s[0] == '1') {

std::swap(num1\_s, num2\_s);

num2\_s[0] = '0';

} else {

num2\_s[0] = '0';

}

std::string result = float\_diff(num1\_s, num2\_s);

result[0] = ((num2\_s[0] - '0') ^ 1) + '0';

return result;

}

std::string exp1 = num1\_s.substr(1, 8), exp2 = num2\_s.substr(1, 8),

mant1 = num1\_s.substr(9), mant2 = num2\_s.substr(9);

std::string mant;

if (num1\_s.find('1') == std::string::npos) {

return num2\_s;

} else if (num2\_s.find('1') == std::string::npos) {

return num1\_s;

}

if (exp1 == exp2) {

mant = bin\_sum(mant1, mant2);

if (mant.find('1') == std::string::npos) {

return std::string(33, '0');

}

if (mant.size() > 24) {

mant.erase(mant.size() - 1, 1);

inc(exp1);

if (exp1.size() > 8) {

std::cout << "Exponent overflow";

exit(1);

}

}

} else {

if (exp1 < exp2) {

std::swap(exp1, exp2);

std::swap(num1\_s, num2\_s);

std::swap(mant1, mant2);

}

while (exp1 != exp2 && mant2.find('1') != std::string::npos) {

inc(exp2);

mant2 = '0' + mant2;

mant2.erase(mant2.size() - 1, 1);

}

if (mant2.find('1') == std::string::npos) {

return num1\_s;

} else {

mant = bin\_sum(mant1, mant2);

if (mant.find('1') == std::string::npos) {

return std::string(33, '0');

}

if (mant.size() > 24) {

mant.erase(mant.size() - 1, 1);

inc(exp1);

if (exp1.size() > 8) {

std::cout << "Exponent overflow";

exit(1);

}

}

}

}

return num1\_s[0] + exp1 + mant;

}

std::string float\_mul(std::string num1\_s, std::string num2\_s) {

std::string exp1 = '0' + num1\_s.substr(1, 8),

exp2 = '0' + num2\_s.substr(1, 8), mant1 = num1\_s.substr(9),

mant2 = num2\_s.substr(9);

char sign\_bit = ((num1\_s[0] - '0') ^ (num2\_s[0] - '0')) + '0';

std::string mant, exp;

int pos1 = mant1.rfind('1'), pos2 = mant2.rfind('1');

mant1.erase(pos1 + 1);

mant2.erase(pos2 + 1);

mant1 = std::string(24 - mant1.size(), '0') + mant1;

mant2 = std::string(24 - mant2.size(), '0') + mant2;

exp = bin\_sum(exp1, exp2);

if (exp.size() == 9) {

exp = bin\_diff(exp, std::bitset<9>(127).to\_string());

exp.erase(0, 1);

} else {

exp = bin\_diff(exp, std::bitset<8>(127).to\_string());

}

mant = bin\_mul(mant1, mant2);

int pos = mant.find('1'), size1 = mant1.size() - mant1.find('1') - 1,

size2 = mant2.size() - mant2.find('1') - 1,

size = mant.size() - mant.find('1') - 1;

if (size > size1 + size2) {

for (int i = 0; i < size - size1 - size2; i++) {

inc(exp);

}

} else if (size < size1 + size2) {

for (int i = 0; i < size1 + size2 - size; i++) {

dec(exp);

}

}

mant.erase(0, pos);

if (mant.size() <= 23) {

mant += std::string(24 - mant.size(), '0');

} else {

mant.erase(24);

}

while (mant[0] != '1') {

dec(exp);

mant.erase(0, 1);

mant += '0';

}

return sign\_bit + exp + mant;

}

std::string float\_div(std::string num1\_s, std::string num2\_s) {

std::string exp1 = '0' + num1\_s.substr(1, 8),

exp2 = '0' + num2\_s.substr(1, 8), mant1 = num1\_s.substr(9),

mant2 = num2\_s.substr(9);

std::string exp, mant;

char sign\_bit = ((num1\_s[0] - '0') ^ (num2\_s[0] - '0')) + '0';

int pres = 24;

exp = bin\_sum(exp1, std::bitset<9>(127).to\_string());

exp = bin\_diff(exp, exp2);

mant1 += std::string(pres, '0');

mant2 = std::string(pres, '0') + mant2;

mant = bin\_div(mant1, mant2);

exp.erase(0, 1);

int pos = mant.find('1');

if (pos < static\_cast<int>(mant.size()) - 1 - pres) {

for (size\_t i = 0; i < mant.size() - 1 - pres - pos; i++) inc(exp);

} else {

for (size\_t i = 0; i < pos - (mant.size() - 1 - pres); i++) dec(exp);

}

mant.erase(0, pos);

if (mant.size() > 24) {

mant.erase(24);

}

return sign\_bit + exp + mant;

}

double float\_to\_dec(std::string num\_s) {

std::string exp = num\_s.substr(1, 8), mant = num\_s.substr(9);

double res = 1;

for (int i = 1; i < static\_cast<int>(mant.size()) - 1; i++) {

res += (mant[i] - '0') \* std::pow(2.0, 0 - i);

}

res \*= std::pow(2, (int(std::bitset<8>(exp).to\_ulong()) - 127));

if (num\_s[0] - '0') {

res \*= -1;

}

return res;

}

void print\_float(std::string float\_s) {

float\_s.erase(9, 1);

std::cout << float\_s;

}

int main() {

std::string num1, num2, num1\_d, num2\_d, sum, diff, mul, div;

std::cout << "Enter first num: ";

std::cin >> num1;

std::cout << "Enter second num: ";

std::cin >> num2;

std::cout << "\n";

print\_float(dec\_to\_float(num1));

std::cout << " - first num float\n"

<< float\_to\_dec(dec\_to\_float(num1)) << " - first num in dec\n\n";

print\_float(dec\_to\_float(num2));

std::cout << " - second num float\n"

<< float\_to\_dec(dec\_to\_float(num2)) << " - second num in dec\n\n";

sum = float\_sum(dec\_to\_float(num1), dec\_to\_float(num2));

print\_float(sum);

std::cout << " - sum float\n" << float\_to\_dec(sum) << " - sum in dec\n\n";

diff = float\_diff(dec\_to\_float(num1), dec\_to\_float(num2));

print\_float(diff);

std::cout << " - diff float\n"

<< float\_to\_dec(diff) << " - diff in dec\n\n";

mul = float\_mul(dec\_to\_float(num1), dec\_to\_float(num2));

print\_float(mul);

std::cout << " - mul float\n" << float\_to\_dec(mul) << " - mul in dec\n\n";

div = float\_div(dec\_to\_float(num1), dec\_to\_float(num2));

print\_float(div);

std::cout << " - div float\n" << float\_to\_dec(div) << " - div in dec\n";

return 0;

}