Министерство образования Республики Беларусь  
Учреждение образования   
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №4-6

По дисциплине «Архитектура вычислительных систем»  
По теме «АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ С ЧИСЛАМИ

С ПЛАВАЮЩЕЙ ТОЧКОЙ»

Выполнил:  
студент гр. 153503  
Киселёва Елизавета Андреевна

Проверил:

Преподаватель

Калиновская Анастасия Александровна

Минск 2023

# 1 Цель работы

Изучить процессы выполнения арифметических операций над числами с плавающей точкой. Составить программу реализации алгоритма арифметических операций.

**2 Постановка задачи**

### ***Задание к лабораторной работе 4***

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

### ***Задание к лабораторной работе 5***

Написать программу эмулятора АЛУ, реализующего *операцию умножения с плавающей точкой* над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

### ***Задание к лабораторной работе 6***

Написать программу эмулятора АЛУ, реализующего *операцию деления с плавающей точкой* над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

# 3 Теоретические сведения

**3.1 Представление чисел в формате с плавающей точкой**

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.

При работе в десятичной системе счисления ученые давно нашли выход из положения, применяя для представления числовых величин так называемую научную нотацию. Так, число 976 000000 000 000 можно представить в виде 9.76x1014, а число 0,000000 000 000 0976 - в виде 9.76x10-14. При этом, фактически, разделительная точка динамически сдвигается в удобное место, а для того чтобы "уследить" за ее положением в качестве второго множителя - характеристики, - используется степень числа 10 (основания характеристики). Это позволяет с помощью небольшого числа цифр (т.е. чисел с ограниченной разрядностью) с успехом представлять как очень большие, так и очень малые величины.

**3.2 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.

**3.3 Арифметические операции над числами с плавающей точкой**

#### **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.

Блок-схема типового алгоритма представлена на рис. 9. Детальный пошаговый анализ этого алгоритма покажет, какие функции используются при выполнении операций сложения и вычитания чисел в формате с плавающей точкой. В дальнейшем для определенности будем считать, что используется формат, регламентированный стандартом IEEE 754. Перед началом выполнения операций операнды должны быть помещены в регистры АЛУ. Если в используемом формате с плавающей точкой предполагается неявный старший разряд мантиссы, этот разряд должен быть в явном виде включен в регистры операндов, и все операции с ним в дальнейшем будут проводиться точно так же, как и с остальными разрядами мантиссы.

#### **Умножение и деление**

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

Сначала рассмотрим алгоритм умножения (рис. К.10). Сразу же посче начала операции проверяется, не равен ли нулю один из сомножителей Если это так, то произведение также будет равно нулю. Следующий шаг — суммирование порядков. Поскольку, как правило, для хранения порядков используется смещенное представление, при суммировании двух смещенных представлений результат будет смещен дважды. Поэтому после суммирования кодов порядков из суммы вычитается значение смещения. При суммировании может возникнуть как переполнение порядка, так и потеря значимости. В обоих случаях формируется соответствующий сигнал. Если порядок произведения не выходит из диапазона, определенного форматом, далее перемножаются мантиссы сомножителей с учетом их знаков. Умножение мантисс выполняется по тому же алгоритму, что и умножение целых чисел в прямом коде, т.е. фактически перемножаются числа без знака, а затем произведению приписывается знак "плюс" или "минус" в зависимости от сочетания знаков сомножителей. Произведение мантисс имеет разрядность, вдвое большую, чем каждый из сомножителей. Лишние младшие разряды отбрасываются при округлении.

После того как будет получено произведение мантисс, результат нормализуется и округляется. Эти операции выполняются так же, как и при сложении или вычитании. Необходимо учесть, что при нормализации может возникнуть переполнение или потеря значимости порядка.

Теперь рассмотрим алгоритм деления (рис. К.11). Как и ранее, первый этап — анализ операндов на равенство нулю. Если нулю равно делимое, то результату сразу присваивается значение 0. Если же нулю равен делитель, то в зависимости от конкретной реализации АЛУ результату может быть присвоено значение "бесконечность" с соответствующим знаком или сформирован сигнал арифметической ошибки.

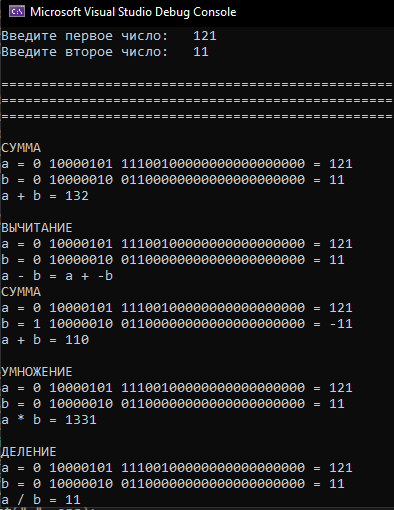
Следующий этап — вычитание кода порядка делителя из кода порядка делимого. При этом получится несмещенный код разности, который нужно скорректировать — сложить с кодом смещения. После завершения операций с порядком результата проверяется, не возникло ли переполнение порядка или потеря значимости.

Следующий этап — деление мантисс. За ним следуют обычные операции нормализации и округления.

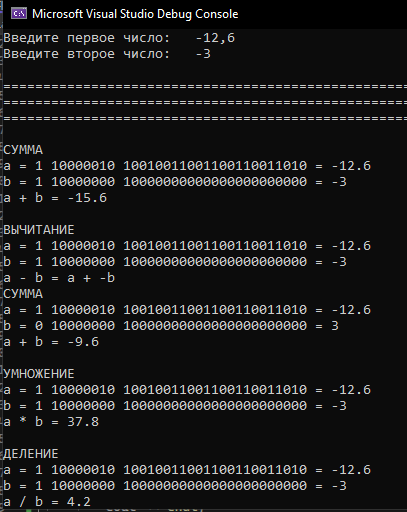
# 4 Пример работы программы

# 

# Тестовый пример 1



Тестовый пример 2



# Тестовый пример 3

# 5 Выводы

В ходе выполнения лабораторной работы была написана программа, реализующая арифметические операции над числами с плавающей точкой в дополнительном коде.

# Приложение 1. Текст программы

#include <iostream>

#include <bitset>

#include <algorithm>

#include <limits>

#include <functional>

#include <regex>

using std::cin;

using std::cout;

using std::string;

using std::endl;

using std::swap;

using std::bitset;

using std::count;

using std::numeric\_limits;

class ALU

{

public:

static float Float(string f) {

unsigned int x = bitset<32>(f).to\_ulong();

return \*((float\*)&x);

}

static string String(float f) {

return bitset<32>(\*((int\*)&f)).to\_string();

}

static void Print(float f) {

cout << String(f).insert(9, " ").insert(1, " ") << " = " << f << endl;

}

static void Print(string f) {

cout << f.insert(9, " ").insert(1, " ") << " = " << Float(f) << endl;

}

static void Print(string s, float f) {

cout << s << " = ";

Print(f);

}

static void Print(string s, string f) {

cout << s << " = ";

Print(f);

}

static bool SignBit(string f) {

return (f[0] == '1');

}

static int Sign(string f) {

return (SignBit(f) ? -1 : 1);

}

static void SetSign(string& f, bool s) {

f[0] = (s ? '1' : '0');

}

static void SetSign(string& f, int s) {

SetSign(f, ((s < 0) ? true : false));

}

static int Exp(string f) {

return (bitset<32>(f.substr(1, 8)).to\_ulong() - 127);

}

static void SetExp(string& f, int e) {

if ((e <= -127) || (128 <= e)) {

throw string("Порядок превышен");

}

f.replace(1, 8, bitset<8>(e + 127).to\_string());

}

static int Mant(string f) {

return bitset<32>("1" + f.substr(9, 23)).to\_ulong();

}

static void SetMant(string& f, int e) {

if ((e < 0) || ((1 << 24) <= e)) {

throw string("Мантисса превышена");

}

if ((e & (1 << 23)) == 0) {

throw string("Мантисса не начинается с 1");

}

f.replace(9, 23, bitset<23>(e ^ (1 << 23)).to\_string());

}

static bool IsZero(string f) {

return count(f.begin() + 1, f.end(), '0') == 31;

}

static string Negate(string f) {

string ans = f;

ans[0] = '0' + '1' - ans[0];

return ans;

}

static void Addition(string a, string b, float af, float bf) {

cout << "СУММА" << endl;

Print("a", a);

Print("b", b);

cout << "a + b = " << af + bf << '\n';

return;

}

static void Addition(float a, float b) {

Addition(String(a), String(b), a, b);

}

static void Subtraction(string a, string b, float af, float bf) {

cout << "ВЫЧИТАНИЕ" << endl;

Print("a", a);

Print("b", b);

cout << "a - b = a + -b" << endl;

Addition(a, Negate(b), af, -bf);

}

static void Subtraction(float a, float b) {

Subtraction(String(a), String(b), a, b);

}

static void Multiplication(string a, string b, float af, float bf) {

cout << "УМНОЖЕНИЕ" << endl;

Print("a", a);

Print("b", b);

cout << "a \* b = " << af \* bf << '\n';

return;

if (IsZero(a)) {

cout << "a = 0.0 -> a \* b = 0" << endl;

Print(" ", 0);

return;

}

if (IsZero(b)) {

cout << "b = 0.0 -> a \* b = 0" << endl;

Print(" ", 0);

return;

}

int SignA = Sign(a), SignB = Sign(b);

int ExpA = Exp(a), ExpB = Exp(b);

int MantA = Mant(a), MantB = Mant(b);

cout << "Порядок a = " << a.substr(1, 8) << endl;

cout << "Порядок b = " << b.substr(1, 8) << endl;

int ms = SignA \* SignB;

int me = ExpA + ExpB;

long long mm = ((long long)MantA \* MantB) >> 23;

cout << "Порядок a\*b = " << bitset<16>(me + 127).to\_string().insert(8, " ") << endl;

if (me >= 128) {

throw string("Переполнение порядка");

}

if (me <= -127) {

throw string("Потеря значимости порядка");

}

cout << "Мантисса a = " << MantA << endl;

cout << "Мантисса b = " << MantB << endl;

cout << "Мантисса a\*b = " << mm << endl;

if (mm == 0) {

throw string("Что-то пошло не так");

}

cout << "Знак a\*b = " << (ms < 0 ? "-1" : "+1") << endl;

cout << "Порядок a\*b = " << me << endl;

cout << "Мантисса a\*b = " << bitset<32>(mm).to\_string().insert(9, " ").insert(8, " ") << endl;

cout << "Нормализация ответа:";

while (mm >= (1 << 24)) {

cout << " ->";

mm >>= 1;

me++;

if (me >= 128) {

cout << endl;

throw string("Переполнение порядка");

}

}

if ((mm & (1 << 23)) == 0) {

throw string("Что-то пошло не так");

}

cout << endl;

cout << "Знак a\*b = " << (ms < 0 ? "-1" : "+1") << endl;

cout << "Порядок a\*b = " << me << endl;

cout << "Мантисса a\*b = " << bitset<32>(mm).to\_string().insert(9, " ").insert(8, " ") << endl;

string ans(32, '0');

SetSign(ans, ms);

SetExp(ans, me);

SetMant(ans, mm);

Print(" ", ans);

return;

}

static void Multiplication(float a, float b) {

Multiplication(String(a), String(b), a, b);

}

static void Division(string a, string b, float af, float bf) {

cout << "ДЕЛЕНИЕ" << endl;

if (bf == 0) {

cout << "\nДеление на ноль!\n";

return;

}

Print("a", a);

Print("b", b);

cout << "a / b = " << af / bf << '\n';

return;

if (IsZero(b)) {

cout << "b = 0.0 ->" << endl;

throw string("Деление на ноль");

}

if (IsZero(a)) {

cout << "a = 0.0, b != 0.0 -> a / b = 0" << endl;

Print(" ", 0);

return;

}

bool SignBitA = SignBit(a), SignBitB = SignBit(b);

int SignA = Sign(a), SignB = Sign(b);

int ExpA = Exp(a), ExpB = Exp(b);

int MantA = Mant(a), MantB = Mant(b);

cout << "Порядок a = " << a.substr(1, 8) << endl;

cout << "Порядок b = " << b.substr(1, 8) << endl;

int ds = SignA \* SignB;

int de = ExpA - ExpB;

long long dm = ((long long)MantA << 23) / MantB;

cout << "Порядок a/b = " << bitset<16>(de + 127).to\_string().insert(8, " ") << endl;

if (de >= 128) {

throw string("Переполнение порядка");

}

if (de <= -127) {

throw string("Потеря значимости порядка");

}

cout << "Мантисса a = " << MantA << endl;

cout << "Мантисса b = " << MantB << endl;

cout << "Мантисса a/b = " << dm << endl;

if (dm == 0) {

throw string("Что-то пошло не так");

}

cout << "Знак a/b = " << (ds < 0 ? "-1" : "+1") << endl;

cout << "Порядок a/b = " << de << endl;

cout << "Мантисса a/b = " << bitset<32>(dm).to\_string().insert(9, " ").insert(8, " ") << endl;

cout << "Нормализация ответа:";

while (dm >= (1 << 24)) {

throw string("Что-то пошло именно так");

}

if ((dm & (1 << 23)) == 0) {

cout << " <-";

dm <<= 1;

de--;

if (de <= -127) {

cout << endl;

throw string("Потеря значимости порядка");

}

}

cout << endl;

cout << "Знак a/b = " << (ds < 0 ? "-1" : "+1") << endl;

cout << "Порядок a/b = " << de << endl;

cout << "Мантисса a/b = " << bitset<32>(dm).to\_string().insert(9, " ").insert(8, " ") << endl;

string ans(32, '0');

SetSign(ans, ds);

SetExp(ans, de);

SetMant(ans, dm);

Print(" ", ans);

return;

}

static void Division(float a, float b) {

Division(String(a), String(b), a, b);

}

};

#define Test(f) \

try { f; cout << endl; } catch (string ex) { cout << "Ошибка: " << ex << endl << endl; }

template<typename ReturnValue>

ReturnValue CorrectInput(const string& variable\_name, const std::regex& pattern,

std::function<ReturnValue(string& input)> converter) {

string value;

do {

cout << "Введите " << variable\_name << ":\t";

cin >> value;

} while (!regex\_match(value, pattern));

return converter(value);

}

int main() {

setlocale(LC\_ALL, "RUS");

float a = CorrectInput<float>(std::string("первое число"), std::regex("-?[0-9]+,?[0-9]\*"), [](std::string& s) {return atof(s.c\_str()); });

float b = CorrectInput<float>(std::string("второе число"), std::regex("-?[0-9]+,?[0-9]\*"), [](std::string& s) {return atof(s.c\_str()); });

cout << '\n';

cout << "========================================================================================================================";

cout << '\n';

cout << "========================================================================================================================";

cout << '\n';

cout << "========================================================================================================================";

cout << '\n';

cout << '\n';

Test(ALU::Addition(a, b));

Test(ALU::Subtraction(a, b));

Test(ALU::Multiplication(a, b));

Test(ALU::Division(a, b));

return 0;

}