Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Архитектура вычислительных систем

ОТЧЁТ
к практической работе
на тему

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ С ЧИСЛАМИ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Выполнил: студент группы 153501
Тимофеев Кирилл Андреевич

Проверила:
Калиновская Анастасия Александровна

МИНСК 2023

СОДЕРЖАНИЕ

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

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 $9.76x10^{14}$, 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.

Этот же подход можно применить и в двоичной системе счисления. Число можно представить в виде

$$\pm SxB^{\pm E}$$

Компоненты такого представления можно сохранить в двоичном слове, с, стоящем из трех полей:

- поле знака числа (плюс или минус)
- поле мантиссы S;
- поле порядка E;

Основание B подразумевается неявно и не сохраняется.

Принципы представления лучше пояснить на примерах. В крайнем левом бите слова хранится знак числа(0 – положительное, 1- отрицательное). В следующих 8 битах хранится значение порядка. Для представления порядка используется так называемый смещенный формат. Для получения действительного двоичного кода порядка из значения, сохраняемого в этом поле нужно вычесть фиксированное смещение. Как правило, смещение равно (2^{k-1}-1), где k – разрядность поля порядка.. В данном случае k = 8, и в поле порядка можно представить коды в диапазоне от 0 до 255. Если принять значение смещения 127, то действительное значение порядка чисел, представленных в таком формате может находится в интервале от -127 до +128. 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.

Таблица 3. Параметры форматов, регламентированные стандартом IEEE 754

Параметр	*Формат*			
	Обычная точность	*Расширенный обычной точности*	*Удвоенная точность*	*Расширенный удвоенной точност и*
Размер слова (бит)	*32*	*≥43*	*64*	*≥79*
Поле порядка (бит)	*8*	*≥11*	*11*	*≥15*
Смещение порядка	*127*	*Не регламентируется*	*1023*	

Максимальное значение порядка	*127*	*≥1023*	*1023*	*≥16383*
Минимальное значение порядка	*-126*	*≤-1022*	*-1022*	*≤-16382*
Диапазон представления (по основанию 10)	10^{-38}, 10^{+38}	*Не регламентируется*	10^{-308}, 10^{+308}	*Не регламентируется*
Поле мантиссы (бит)	*23*	*≥31*	*52*	*≥63*
Количество значений порядка	*254*	*Не регламентируется*	*2046*	*Не регламентируется*
Количество значений мантиссы	2^{23}	*Не регламентируется*	2^{52}	*Не регламентируется*
Количество отличающихся представимых величин	$1.98x2^{31}$	*Не регламентируется*	$1.99x2^{63}$	*Не регламентируется*

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.

Поскольку операции сложения и вычитания отличаются только тем, что при вычитании предварительно изменяется знак второго операнда (вычитаемого), эта операция включена в ветвь ВЫЧИТАНИЕ на схеме алгоритма, после чего обе ветви сливаются. Далее анализируется, не равен ли один из операндов нулю. Если это так, то результат — значение второго операнда.

Следующий этап — изменение кодов операндов таким образом, чтобы значения их порядков стали равны. Чтобы понять, зачем это нужно, рассмотрим следующий пример в десятичной системе счисления:

$$(123x10^0) + (456x10^{-2}).$$

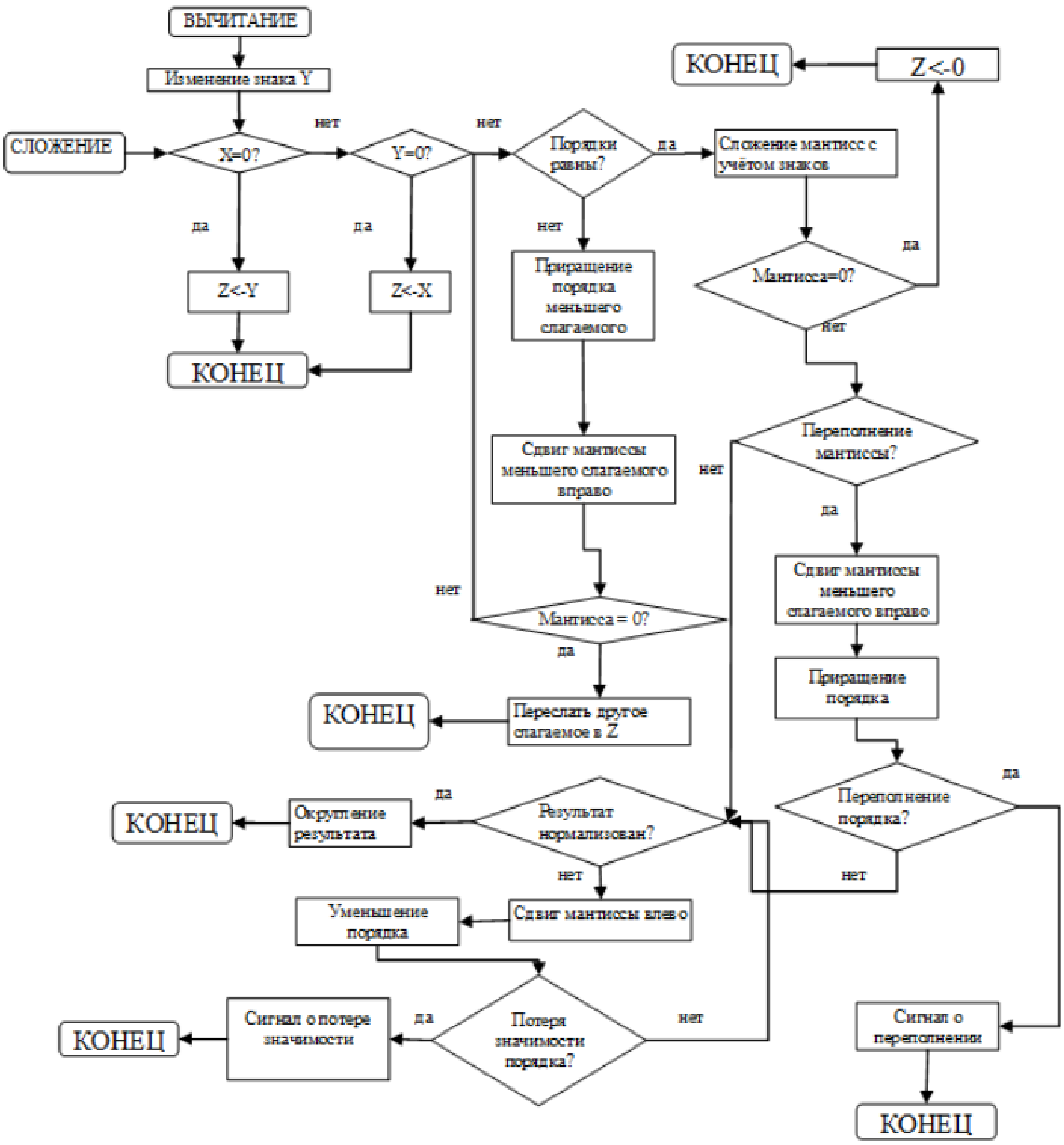
Очевидно, что нельзя просто сложить мантиссы этих двух чисел. Сначала нужно выровнять разрядные сетки обеих мантисс так, чтобы соответственные разряды (разряды с равным весом) занимали одинаковые позиции, т.е. цифра 4 второго числа находилась в той же позиции, что и цифра 3 первого числа. При этом порядки обоих чисел будут равны. Равенство порядков и есть, с точки зрения математики, условие, позволяющее складывать мантиссы обоих чисел в такой форме представления. Следовательно,

$$(123x10^0) + (456x10^{-2}) - (123x10^0) + (4.56x10^0) = 127.56x10^0.$$

Выравнивание выполняется за счет сдвига мантиссы меньшего числа вправо или мантиссы большего числа — влево. Поскольку в любом варианте теряются цифры операнда, выполняется сдвиг мантиссы меньшего числа вправо, что приводит к утере ее младших разрядов. Одновременно со сдвигом мантиссы вправо порядок меньшего числа увеличивается. Сдвиги выполняются до тех пор пока значения порядков обоих чисел не станут равны.

После того, как порядки будут выровнены, наступает этап сложения мантисс с учетом их знаков. Поскольку слагаемые могут иметь разные знаки, их алгебраическая сумма может оказаться равной нулю. Не исключено и появление переполнения — переноса из старшего разряда суммы. В этом случае необходимо выполнить дополнительный сдвиг результата вправо и одновременно увеличить на 1 значение порядка. Но тогда возможно появление переполнения порядка, которое расценивается как аварийная ситуация и влечет за собой прекращение операции.

После сложения мантисс наступает этап нормализации результата. Нормализация представляет собой серию сдвигов кода мантиссы влево (в сторону старших разрядов) с одновременным уменьшением значения порядка до тех пор, пока значение старшей цифры мантиссы не станет отличным от нуля. Я специально оговариваю цифры, поскольку в случае, когда основанием характеристики является число 16, шестнадцатеричной цифрой мантиссы будет служить 4-разрядный двоичный код. Последняя операция – округление результата.



Умножение и деление

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

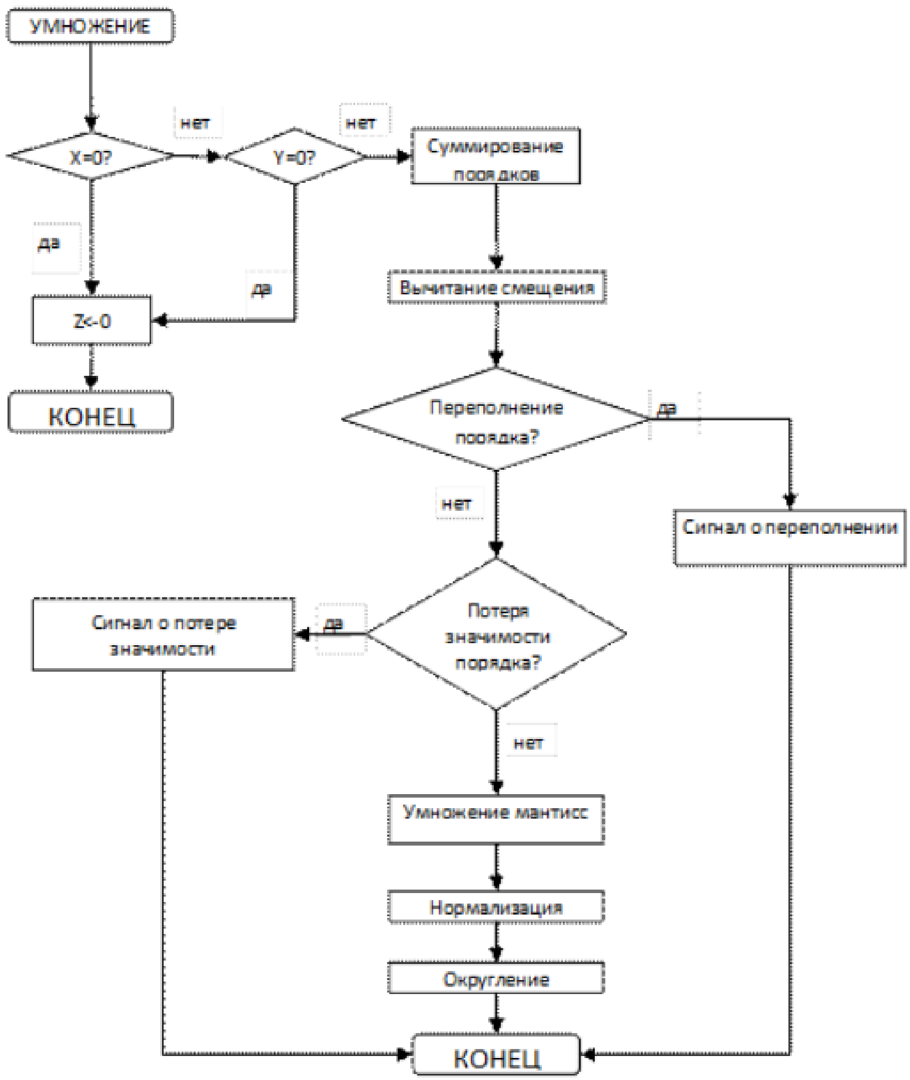


Рис. 10. Схема алгоритма умножения чисел в формате с плавающей точкой (Z<- X x Y)

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.

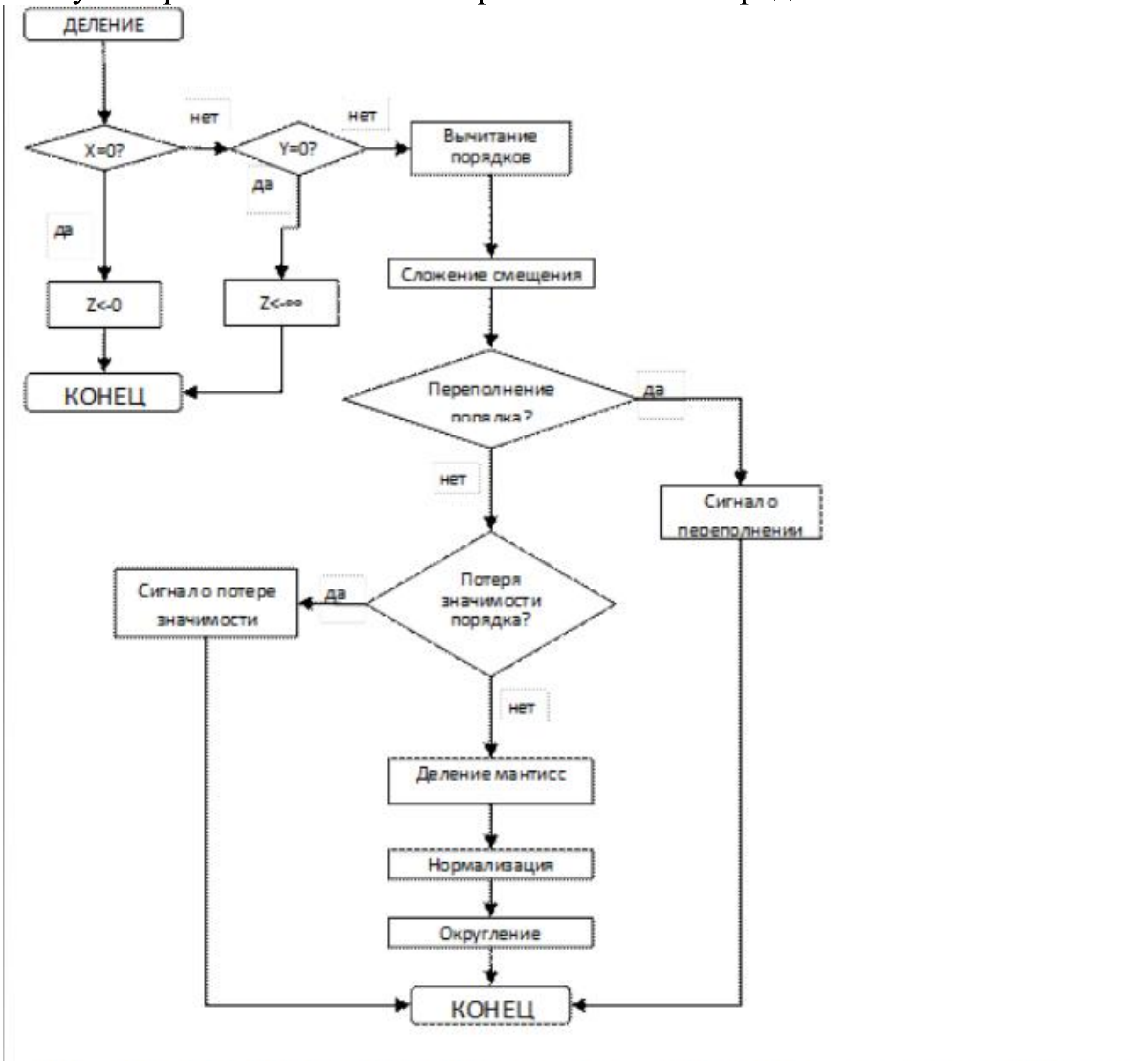


Рис. 11. Схема алгоритма деления чисел в формате с плавающей точкой ($Z \leftarrow X / Y$)

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.

ЗАДАНИЯ И ВЫВОД ПРОГРАММЫ

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.

Вывод программы

```
Введите первое число
44
Введите второе число
12
a = 0 10000010 10000000000000000000000
b = 0 10000010 10000000000000000000000
3145728
4194304


a+b
0 10000100 11000000000000000000000
1.7500000e5 = 56
3145728
4194304


a-b
0 00000000 00000000000000000000000
1.0000000e-127 = 0


a*b
0 10001000 00001000000000000000000
1.0312500e9 = 528


a/b
0 10000000 11010101010101010101010
1.8281250e1 = 3.66667
```

ПРОГРАММНЫЙ КОД

```cpp
#include <iostream>
#include <bitset>
#include <string>

using namespace std;

const int SIZE = 32;
const int EN = 8;
const int MN = 23;
const int SHIFT = (1 << (EN - 1)) - 1;

typedef bitset<32> myfloat;
typedef bitset<EN> pow2;
typedef bitset<MN> mantis;

bitset<23> operator+(bitset<23> m1, bitset<23> m2)
{
    return bitset<23>(m1.to_ulong() + m2.to_ulong());
}

bool isZero(myfloat a)
{
    a[31] = 0;
    return a.none();
}

myfloat zero(bool sign = 0)
{
    myfloat rv = myfloat();
    rv.reset();
    rv[0] = sign;
    return rv;
}

pow2 getPow(myfloat a)
{
    pow2 rv = pow2();
    for (int i = 0; i < EN; ++i)
    {
        rv[i] = a[i + MN];
    }

    return rv;
}

mantis getMantis(myfloat a)
{
    mantis rv = mantis();
    for (int i = 0; i < MN; ++i)
    {
        rv[i] = a[i];
    }

    return rv;
}
```

```cpp
void setMantis(myfloat& a, long m)
{
    for (int i = 0; i < MN; ++i)
    {
        a[i] = m % 2;
        m >>= 1;
    }
}

void setPow(myfloat& a, unsigned long pow)
{
    for (int i = 31 - EN; i < 31; ++i)
    {
        a[i] = pow % 2;
        pow >>= 1;
    }
}

void SetZero(myfloat& a)
{
    a.reset();
}

myfloat operator+(myfloat a, myfloat b)
{
    if (isZero(a)) return b;
    if (isZero(b)) return a;

    if (getPow(a).to_ulong() < getPow(b).to_ulong()) swap(a, b);
    else if (getPow(a).to_ulong() == getPow(b).to_ulong())
    {
        if (getMantis(a).to_ulong() < getMantis(b).to_ulong()) swap(a, b);
    }
    unsigned long pow_a = getPow(a).to_ulong();
    unsigned long pow_b = getPow(b).to_ulong();

    cout << getMantis(a).to_ulong() << endl;
    cout << getMantis(b).to_ulong() << endl;
    long m_a = getMantis(a).to_ulong() + (1 << 23);
    long m_b = getMantis(b).to_ulong() + (1 << 23);


    while (pow_a != pow_b)
    {
        pow_b += 1;
        m_b >>= 1;
        if (m_b == 0) return a;
    }

    if (a[31] == 1) m_a *= -1;
    if (b[31] == 1) m_b *= -1;

    long res = m_a + m_b;
    bool sign = res < 0;
    if (sign) res = -res;
    if (res >= 1 << 24)
```

```cpp
    if (res >= 1 << 24)
    {
        res >>= 1;
        pow_a += 1;
    }
    res -= 1 << 23;

    myfloat rv = myfloat();
    if (pow_a > 254)
    {
        cout << "Сигнал о переполнении";
        return myfloat();
    }
    if (res == 0) return zero(sign);
    setMantis(rv, res);
    setPow(rv, pow_a);
    rv[31] = sign;
    return rv;
}

myfloat operator-(myfloat a, myfloat b)
{
    b[31].flip();
    return a + b;
}

myfloat operator*(myfloat a, myfloat b)
{
    unsigned long pow_a = getPow(a).to_ulong();
    unsigned long pow_b = getPow(b).to_ulong();

    unsigned long pow_r = pow_a + pow_b - SHIFT;

    unsigned long long mant_a = getMantis(a).to_ulong() + (1 << 23);
    unsigned long long mant_b = getMantis(b).to_ulong() + (1 << 23);
    unsigned long long mant_r =  mant_a * mant_b;
    mant_r >>= 23;
    if (mant_r >= (1 << 24))
    {
        pow_r += 1;
        mant_r >>= 1;
    }
    mant_r -= (1 << 23);
    myfloat res = myfloat();
    setMantis(res, mant_r);
    setPow(res, pow_r);
    res[31] = a[31] != b[31];

    return res;
}

string toString(myfloat a)
{
    long p = getPow(a).to_ulong() - 127;
    mantis m = getMantis(a);
    string str = (a[31] == 1) ? "-" : "";
```

```cpp
    string str = (a[31] == 1) ? "-" : "";
    long long j = 5;
    unsigned long long f = 1;
    long long count = 1;

    for (int i = m.size() - 1; i >= 16; --i)
    {
        f *= 10;
        count *= 10;
        if (m[i] == 1)
        {
            f += j;
        }

        j *= 5;
    }

    string mstr = to_string(f);
    mstr.erase(mstr.begin());
    return str + "1." + mstr + "e" + to_string(p);
}


string format(string bs)
{
    bs.insert(1, 1, ' ');
    bs.insert(2 + EN, 1, ' ');

    return bs;
}
union fuck
{
    float f;
    unsigned long l;
};

myfloat operator/(myfloat a, myfloat b)
{
    pow2 p_a = getPow(a);
    pow2 p_b = getPow(b);

    mantis m_a = getMantis(a);
    mantis m_b = getMantis(b);

    long res_pow = p_a.to_ulong() - p_b.to_ulong() + SHIFT;

    unsigned long long ch = m_a.to_ullong() + (1 << 23);
    unsigned long long zn = m_b.to_ullong() + (1 << 23);

    ch <<= 23;
    unsigned long long res_m = ch / zn;
    if (!(res_m & (1 << 23)))
    {
        --res_pow;
        res_m <<= 1;
    }
```

```cpp
            res_m <<= 1;
        }

    res_m -= (1 << 23);
    myfloat res = myfloat();
    setMantis(res, res_m);
    setPow(res, res_pow);

    res[SIZE - 1] = a[31] != b[31];

    return res;
}

int main()
{
    setlocale(0, "");
    fuck f;
    cout << "Введите первое число\n";
    cin >> f.f;
    myfloat a = myfloat(f.l);
    cout << "Введите второе число\n";
    cin >> f.f;
    myfloat b = myfloat(f.l);

    cout << "a = " << format(bitset<32>(f.l).to_string()) << endl;
    cout << "b = " << format(bitset<32>(f.l).to_string()) << endl;

    auto r = a + b;
    f.l = r.to_ulong();
    cout << endl << endl << "a+b" << endl;
    cout << format(r.to_string()) << endl;
    cout << toString(r) << " = " << f.f << endl;

    r = a - b;
    f.l = r.to_ulong();
    cout << endl << endl << "a-b" << endl;
    cout << format(r.to_string()) << endl;
    cout << toString(r) << " = " << f.f << endl;

    r = a * b;
    f.l = r.to_ulong();
    cout << endl << endl << "a*b" << endl;
    cout << format(r.to_string()) << endl;
    cout << toString(r) << " = " << f.f << endl;

    r = a/b;
    f.l = r.to_ulong();
    cout << endl << endl << "a/b" << endl;
    cout << format(r.to_string()) << endl;
    cout << toString(r) << " = " << f.f << endl;
}
```

ВЫВОДЫ

В результате данной работы были изучены методы представления чисел с плавающей точкой и формат IEEE-754.