Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Кафедра информатики

Отчет по практическому занятию №2

Арифметические операции с числами с плавающей точкой

Выполнила:
студент гр. 153505
Власенко Т. П.

Проверила:
Калиновская А. А.

Минск 2023

1. Цель работы

Рассмотреть представление чисел с плавающей точной в двоичном коде. Изучить алгоритмы выполнения основных арифметических операций над действительными числами с плавающей точкой. Написать программу, реализующую такие алгоритмы.

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

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.

4. Программная реализация

В качестве средств для написания программы используются язык C++ и среда разработки Visual Studio Code.

Результаты:

```
Enter a: 44.678
Enter b: -9.21

a:
01000010001100101011011001000101
44.678

b:
11000001000100110101110000101000
-9.21

SUM:
01000010000011011101111100111011
35.468

SUB:
01000010010101111000110101001111
53.888

MUL:
11000011110011011011110111111110
-411.484

DIV:
11000000100110110011101110100110
-4.85103
```

```
Enter a: -455.211
Enter b: -2.9998

a:
11000011111000111001101100000010
-455.211

b:
11000000001111111111110010111001
-2.9998

SUM:
11000011111001010001101011111011
-458.211

SUB:
11000011111000100001101100001001
-452.211

MUL:
01000100101010101011000101010111
1365.54

DIV:
01000011000101111011111101000011
151.747
```

Рисунок 1. Пример работы программы

5. Выводы

В ходе выполнения лабораторной было подробно изучено хранение дробных чисел в памяти компьютера. Так же была рассмотрена работа арифметико-логического устройства.

Приложение 1. Исходный код программы

```
#include <iostream>
#include <sstream>
#include <bitset>
```

```
#include <cmath>
#include <algorithm>

std::string bin_sub(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) + 48;
        }
        else
        {
            res += tmp + 48;
            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_sub(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] - 48 + carry;
        num_s[i] = tmp % 2 + 48;
        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 - 48 - carry;
        if (tmp < 0)
        {
            carry = 1;
            x = tmp * (-1) % 2 + 48;
        }
        else
        {
            x = tmp + 48;
            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, size;
    std::string res;

    cont << num_s;
    cont >> num;
    size = std::ceil(std::log2(num));

    while (num)
    {
        res += num % 2 + 48;
        num /= 2;
    }

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

    return res;
}

std::string compl_fract(std::string num_s, size_t zero_count)
{
    auto pos_it = std::find(num_s.begin(), num_s.end(), '.');
    size_t k;
    std::string zero_str(zero_count, '0');

    num_s += '.' + zero_str;

    return num_s;
}

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 = compl_fract(res, 24 - res.size());
        return res;
    }
    else
    {
        std::string int_part_s, fract_part_s, res;
        double fract_part;

        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 + 48;

    return res;
}

std::string conv(std::string num_s)
{
    int carry = 1, tmp;

    for (auto &x : num_s)
    {
        if (x != '.')
            x = (x - 48) ^ 1 + 48;
        std::cout << x << ' ';
```

```
    }

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

    for (auto &x : num_s)
    {
        tmp = x - 48 + carry;
        x = tmp % 2 + 48;
        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_sub(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;

    if (num1_s[0] != num2_s[0])
    {
        num2_s[0] = (num2_s[0] - 48) ^ 1 + 48;
        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_sub(mant1, mant2);

        if (mant.find('1') == std::string::npos)
            return std::string(33, '0');

        while (!(mant[0] - 48))
        {
```

```
            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_sub(mant1, mant2);

            if (mant.find('1') == std::string::npos)
                return std::string(33, '0');

            while (!(mant[0] - 48))
            {
                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';
        }
        return float_sub(num1_s, 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;

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 << "perepolnenie poriadka";
            exit(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_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 << "perepolnenie poriadka";
                    exit(0);
                }
            }
        }
    }

    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] - 48) ^ (num2_s[0] - 48) + 48;
    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_sub(exp, std::bitset<9>(127).to_string());
        exp.erase(0, 1);
    }
    else
    {
        exp = bin_sub(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 (size_t i = 0; i < size - size1 - size2; i++)
            inc(exp);
    }
    else if (size < size1 + size2)
    {
        for (size_t 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] - 48) ^ (num2_s[0] - 48) + 48;
    int pres = 24;

    exp = bin_sum(exp1, std::bitset<9>(127).to_string());
    exp = bin_sub(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 < 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 < mant.size() - 1; i++)
    {
        res += (mant[i] - 48) * std::pow(2.0, 0 - i);
    }

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

    if (num_s[0] - 48)
        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, sub, mul, div;
    std::cout << "Enter a: ";
    std::cin >> num1;
    std::cout << "Enter b: ";
    std::cin >> num2;
    std::cout << "\n";

    std::cout << "a:\n";
    print_float(dec_to_float(num1));
    std::cout << "\n";
    std::cout << float_to_dec(dec_to_float(num1));
    std::cout << "\n\n";

    std::cout << "b:\n";
    print_float(dec_to_float(num2));
    std::cout << "\n";
    std::cout << float_to_dec(dec_to_float(num2));
    std::cout << "\n\n";

    std::cout << "SUM:\n";
    sum = float_sum(dec_to_float(num1), dec_to_float(num2));
    print_float(sum);
    std::cout << "\n";
    std::cout << float_to_dec(sum);
    std::cout << "\n\n";

    std::cout << "SUB:\n";
    sub = float_sub(dec_to_float(num1), dec_to_float(num2));
    print_float(sub);
    std::cout << "\n";
    std::cout << float_to_dec(sub);
    std::cout << "\n\n";

    std::cout << "MUL:\n";
    mul = float_mul(dec_to_float(num1), dec_to_float(num2));
    print_float(mul);
    std::cout << "\n";
    std::cout << float_to_dec(mul);
    std::cout << "\n\n";

    std::cout << "DIV:\n";
    div = float_div(dec_to_float(num1), dec_to_float(num2));
    print_float(div);
    std::cout << "\n";
    std::cout << float_to_dec(div);
    std::cout << "\n";

    return 0;
}
```