Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра информатики

Отчёт по лабораторной работе №1
«Арифметические операции с целыми числами»

Выполнил:
студент гр. 153505
Савончик Е. В.

Проверила:
Калиновская А.А.

МИНСК 2023

Содержание

1 Цель работы

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

2 Постановка задачи

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

3 Теоретические сведения

Представление информации в компьютере

В двоичной системе счисления числа представляются с помощью комбинации единиц и нулей, знака «минус» и знака разделяющей точки между целой и дробной частью числа. Например, десятичное число -1.312510 в двоичном виде будет выглядеть как -1001.01012. Но в компьютере мы не можем хранить и обрабатывать символы знака и разделяющей точки — для "машинного" представления чисел могут использоваться только двоичные цифры (0 и 1). Если операции выполняются только с неотрицательными числами, то формат представления очевиден. В машинном слове из 8 бит можно представить числа в интервале от 0 до 255.

Прямой код

При записи числа в прямом коде старший разряд является знаковым разрядом. Если его значение равно нулю, то представлено положительное число или положительный ноль, если единице, то представлено отрицательное число или отрицательный ноль. В остальных разрядах (которые называются цифровыми) записывается двоичное представление модуля числа.

Дополнительный код

Как и в прямом, в дополнительном коде старший разряд в разрядной сетке отводится для представления знака числа. Остальные разряды интерпретируются не так, как в прямом коде. В табл. 1 перечислены основные свойства дополнительного кода и правила выполнения арифметических операций в дополнительном коде, которые мы рассмотрим в этом и следующем разделах.

Представление с фиксированной точкой

И наконец, следует остановиться еще на одном нюансе. Описанные выше форматы объединяются часто одним термином — формат с фиксированной точкой. Суть его в том, что положение разделительной точки между целой и дробной частями числа неявно фиксируется на разрядной сетке. В настоящее время принято фиксировать точку справа от самого младшего значащего разряда. Программист может использовать аналогичное представление для работы с двоичными дробными числами, мысленно фиксируя точку перед старшим значащим разрядом и соответственно масштабируя результаты преобразований, выполняемых стандартными программными или аппаратными средствами.

Сложение и вычитание двоичных чисел

Поскольку в двоичной арифметике используется позиционная система записи чисел, сложение и вычитание могут быть выполнены поразрядно.

Умножение

Алгоритмы выполнения умножения значительно сложнее, причем в современных вычислительных системах можно встретить как аппаратную его реализацию, так и программную. Существует много вариантов этих алгоритмов, причем многие из них имеют не только теоретический, но и практический интерес, и выбор одного из многих может быть произведен только с учетом специфики применения конкретной системы. В данном разделе мы ставили перед собой задачу дать читателю общее представление о подходе, на основе которого такие алгоритмы nроектируются. Начнем с простой задачи перемножения двух чисел без знака (т.е. неотрицательных чисел), а затем рассмотрим один из наиболее широко известных алгоритмов умножения целых чисел со знаком, представленных в двоичном коде.

Деление

По сравнению с умножением операция деления выполняется несколько сложнее, хотя соответствующие алгоритмы основываются на тех же принципах поразрядного анализа операндов. Исходный алгоритм, как и при умножении, — тот, который используется при вычислении вручную, карандашом на бумаге. Алгоритм состоит из повторяющейся последовательности шагов элементарных сдвигов и сложений или вычитаний.

4 Код программы

```
#include <iostream>
#include <algorithm>
#include <string>
#include <sstream>
#include <bitset>
#include <cmath>

int size_out;

std::string int_part_to_binary(std::string num_s) {
    std::stringstream cont;
    int num, size;
    std::string res;

    cont << num_s;
    cont >> num;
    if (num == 0) {
        size = 1;
    } else {
        size = std::ceil(std::log2(num));

        if (std::ceil(std::log2(num)) == std::log2(num))
            size++;
    }

    if (size > size_out)
        size_out = size;

    res = std::bitset<64>(num).to_string();

    return res;
}

std::string complete_fractional(std::string num_s){
    auto pos_it = std::find(num_s.begin(), num_s.end(), '.');
    // size_t k;

    if (pos_it == num_s.end()) {
        num_s += '.';

        for (size_t i = 0; i < 8; i++)
            num_s += '0';
    } else {
        int k = num_s.size() - std::distance(num_s.begin(), pos_it)-1;
        for (size_t i = 0; i < 8 - k; i++)
            num_s += '0';
    }

    return num_s;
}

std::string convert(std::string num_s) {
    for (auto &x:num_s) {
        if (x == '1')
            x = '0';
        else if (x == '0')
            x = '1';
    }


    int tmp, carry = 1;
    std::reverse(num_s.begin(), num_s.end());
```

```
    for (auto &x:num_s) {
        if (x == '.')
            continue;

        tmp = x-48;
        tmp += carry;

        if (tmp > 1) {
            carry = 1;
            x = 48;
        } else {
            carry = 0;
            x = tmp+48;
        }
    }

    if (carry == 1)
        num_s += '1';

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

    return num_s;
}

std::string decimal_to_binary(std::string num_s) {
    bool is_neg = false;

    if (num_s[0] == '-') {
        is_neg = 1;
        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_binary(num_s);
        if (is_neg)
            res = convert(res);
        res = complete_fractional(res);
        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_binary(int_part_s);
        std::stringstream cont;
        cont << fract_part_s;
        cont >> fract_part;
        res += '.';

        for (size_t i = 0; i < 8; i++) {
            fract_part *= 2;
            if (fract_part == 0)
                break;

            if (fract_part >= 1) {
                res += '1';
                fract_part -=1;
            } else {
                res += '0';
            }
```

```
        }

        if (is_neg)
            res = convert(res);
        res = complete_fractional(res);

        return res;
    }
}

std::string binary_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(res.begin(), res.end());

    return res;
}

std::string binary_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) + 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 binary_mul(std::string num1_s, std::string num2_s, bool
is_print = false) {
    std::string res, tmp_res, null_string;

    num1_s.erase(64, 1);
    num2_s.erase(64, 1);

    for (size_t i = 0; i < 146; i++)
        null_string += '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] = num1_s[71];
        }
        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 = binary_sum(res, tmp_res);
        std::reverse(res.begin(), res.end());
        if (is_print)
            std::cout << res << std::endl;
    }

    res.insert(16, ".");
    std::reverse(res.begin(), res.end());

    return res;
}

std::string binary_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;
    Q.erase(64, 1);
    M.erase(64, 1);

    for (size_t i = 0; i < 8; i++) {
        Q += '0';
        M = M[0] + M;
    }
```

```
    for (size_t i = 0; i < Q.size(); i++)
        A += Q[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 = binary_diff(A, M);
        else
            A = binary_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 << Q << std::endl;

    }

    if (A[0] != M[0])
        Q = convert(Q);

    Q.insert(Q.size() - 8, ".");

    return Q;
}

double binary_to_decimal(std::string num_s) {
    int pos = num_s.find(".");
    double res = 0;
    bool is_neg = 0;

    if (num_s[0] == '1') {
        is_neg = 1;
        num_s = convert(num_s);
    }

    for (int i = pos - 1; i >= 0; i--) {
        res += std::pow(2, pos - i -1) * (num_s[i] - '0');
    }

    for (int i = pos + 1; i < num_s.size(); i++) {
        res += std::pow(2, -(i - pos)) * (num_s[i] - '0');
    }

    if (is_neg)
        res *= -1;

    return res;
}
```

```
int main()
{
    std::string num1, num2, num1_b, num2_b, sum, diff, mul, div;

    std::cout << "enter the first num:";
    std::cin >> num1;
    std::cout << "enter the second num:";
    std::cin >> num2;

    num1_b = decimal_to_binary(num1);
    num2_b = decimal_to_binary(num2);

    std::cout << num1_b << " - num1 in binary" << std::endl << num2_b << "
- num2 in binary" << std::endl << std::endl;

    sum = binary_sum(num1_b, num2_b);
    sum.erase(0, sum.size() - 9 - size_out - 2);
    diff = binary_diff(num1_b, num2_b);
    diff.erase(0, diff.size() - 9 - size_out - 2);
    mul = binary_mul(num1_b, num2_b);
    mul.erase(0, mul.size() - 17 - size_out * 2);
    div = binary_div(num1_b, num2_b);
    div.erase(0, div.size() - 9 - size_out - 2);

    std::cout << sum << " - binary summary" << std::endl;
    std::cout << binary_to_decimal(sum) << " - decimal summary" <<
std::endl;
    std::cout << diff << " - binary difference"<<  std::endl;
    std::cout << binary_to_decimal(diff) << " - decimal difference" <<
std::endl;
    std::cout << mul << " - binary multiplication" << std::endl;
    std::cout << binary_to_decimal(mul) << " - decimal multiplication" <<
std::endl;
    std::cout << div << " - binary division" << std::endl;
    std::cout << binary_to_decimal(div) << " - decimal division" <<
std::endl;

    return 0;
}
```

5 Результаты работы программы

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

На рисунках представлен результат работы программы с разными входными данными:

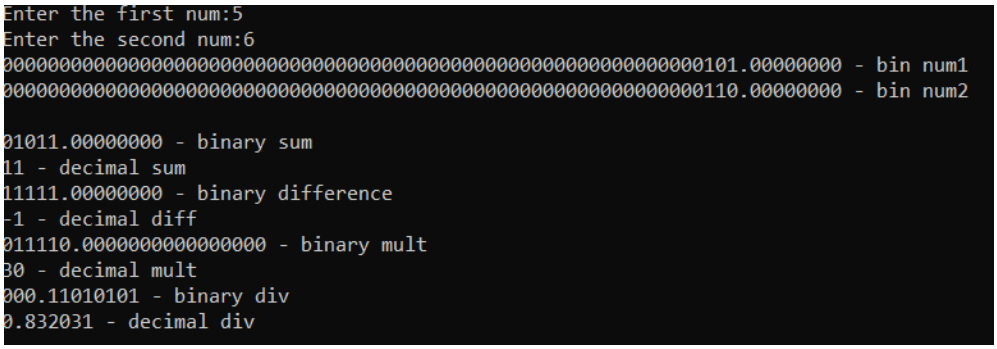

Рис. 1 – Пример (два положительных числа).

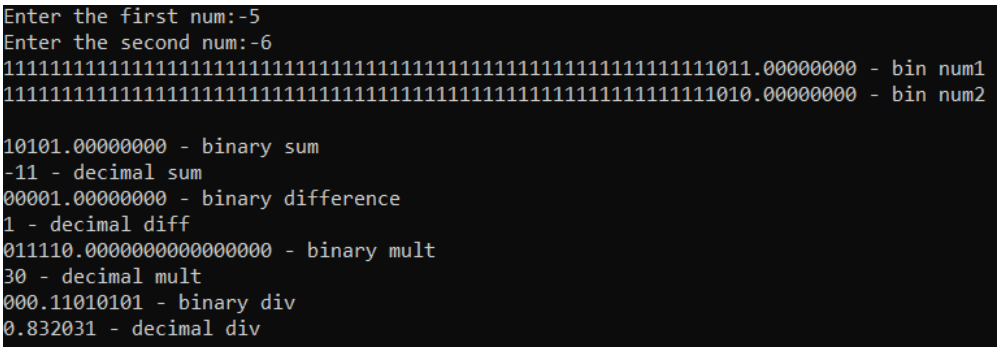

Рис. 2 – Пример (два отрицательных числа).

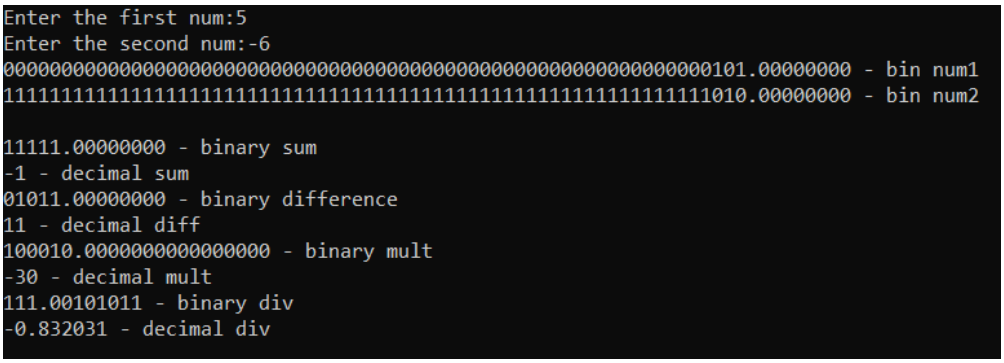

Рис. 3 – Пример (числа разного знака).

6 Выводы

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