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.
2. Постановка задачи.
3. Теоретические сведения.
4. Код программы.
5. Результаты работы программы.
6. Вывод.
7. **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.

1. **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.

1. **Теоретические сведения**

***Представление информации в компьютере***

* 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 cистемы. В данном разделе мы ставили перед собой задачу дать читателю общее представление о подходе, на основе которого такие алгоритмы npoeктируются. 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.

1. **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;

}

1. **Результаты работы программы**

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

На рисунках представлен результат работы программы с разными входными данными:

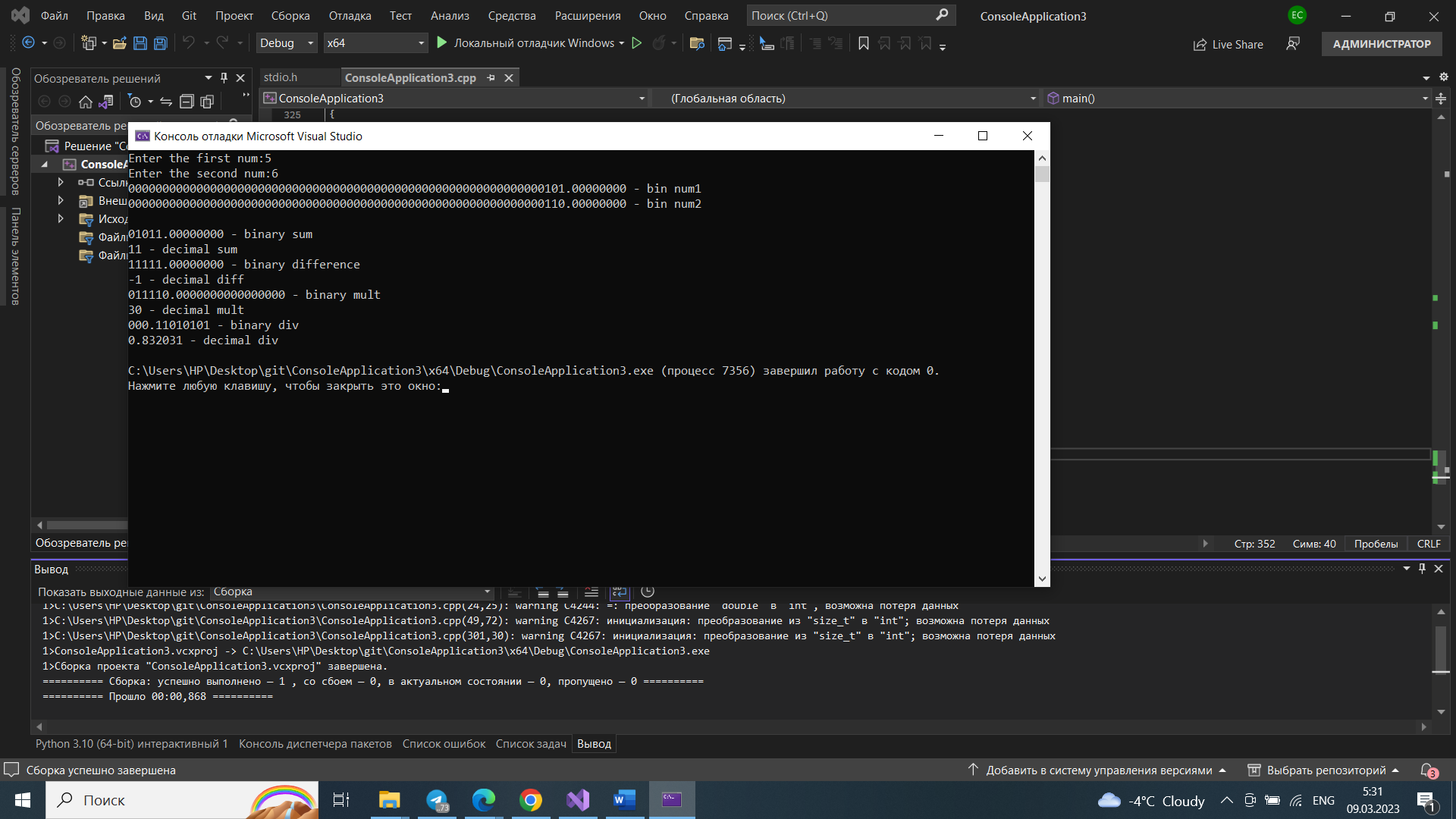


Рис. 1 – Пример (два положительных числа).

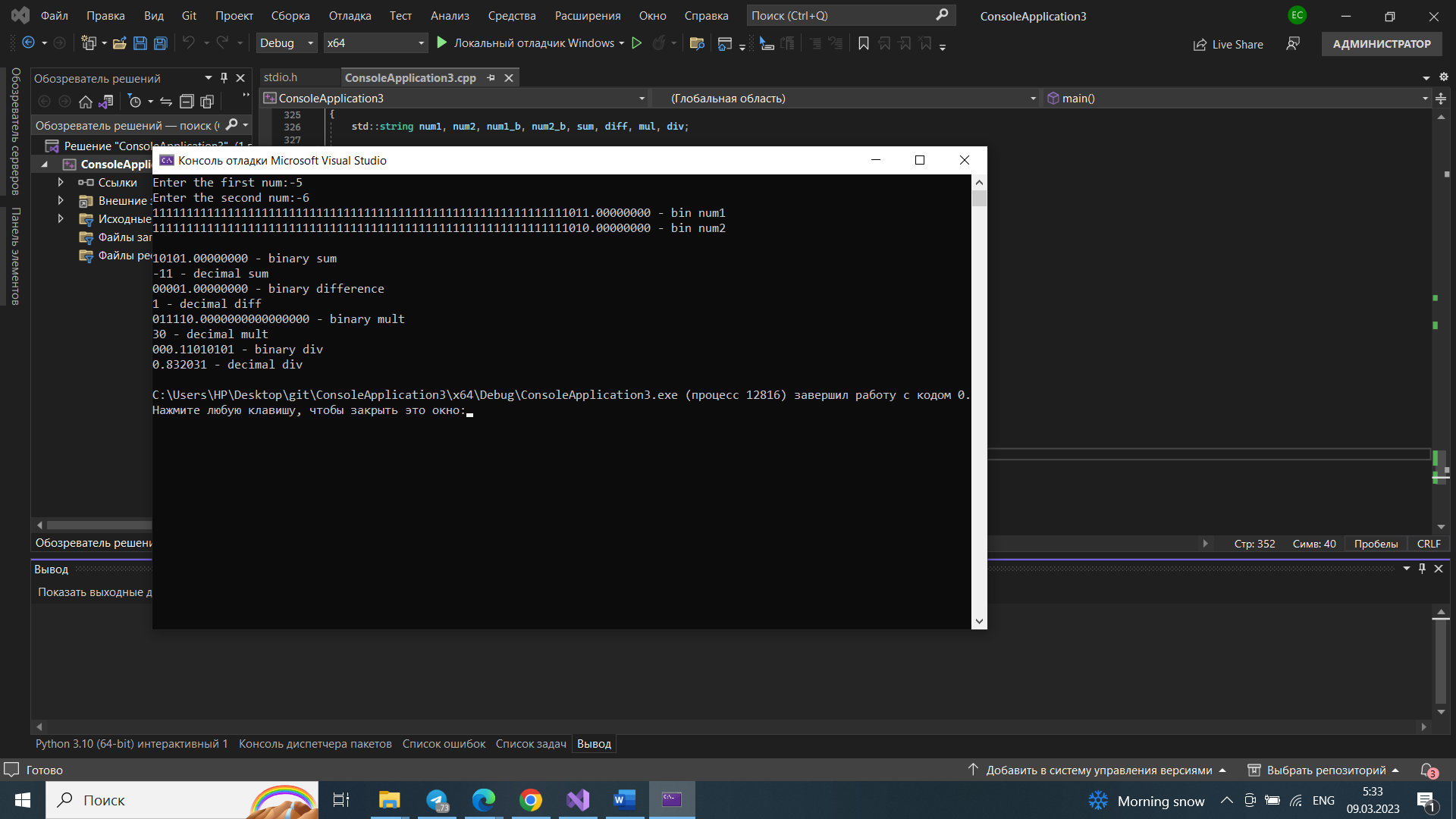


Рис. 2 – Пример (два отрицательных числа).

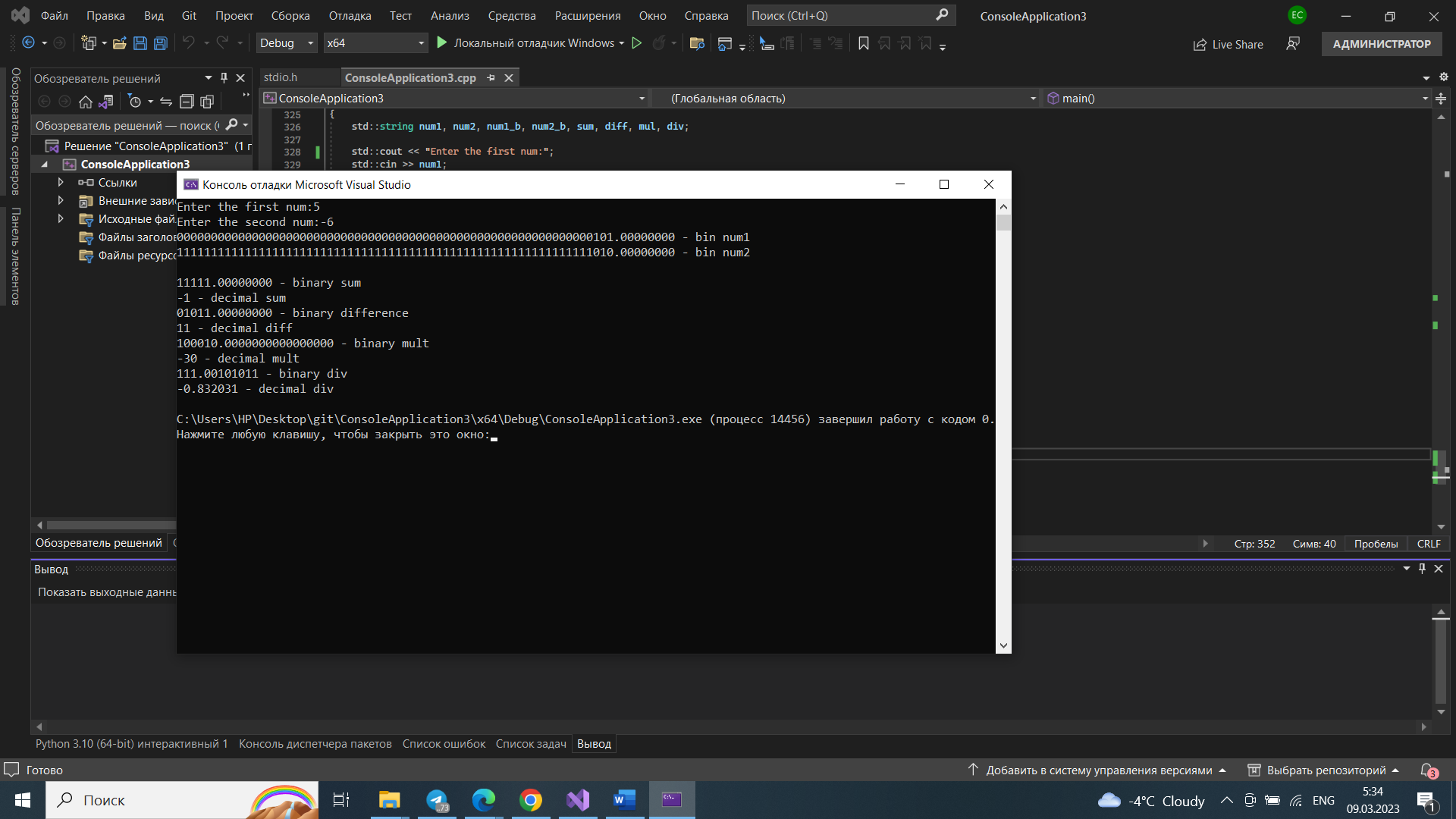


Рис. 3 – Пример (числа разного знака).

1. **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.