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. 153503

Вергасов В. М.

Проверила:

Калиновская А. А.

Минск 2023

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.

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. **Программная реализация**

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

На рисунках представлен результат работы программы с разными входными данными.

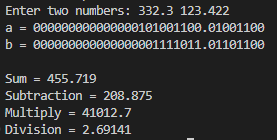


Рис. 1 – Пример (два положительных числа).

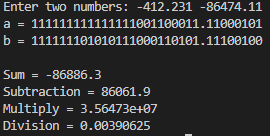


Рис. 2 – Пример (два отрицательных числа).

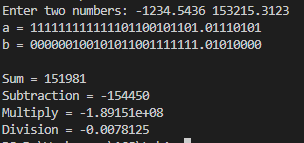


Рис. 3 – Пример (числа разного знака).

**5. 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.

**Приложение 1. Исходный код программы.**

#include <algorithm>

#include <bitset>

#include <cmath>

#include <iostream>

#include <string>

std::string get\_2s\_complement(std::string num) {

for (auto& c : num) {

if (c == '1') {

c = '0';

} else if (c == '0') {

c = '1';

}

}

int tmp, carry = 1;

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

for (auto& c : num) {

if (c == '.') {

continue;

}

tmp = c - 48;

tmp += carry;

if (tmp > 1) {

carry = 1;

c = 48;

} else {

carry = 0;

c = tmp + 48;

}

}

if (carry == 1) {

num += '1';

}

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

return num;

}

int const integer\_bits = 24;

int const fractional\_bits = 8;

std::string to\_fixed\_point\_str(float number) {

std::string binary;

bool is\_negative = number < 0;

int integer\_part = static\_cast<int>(number);

if (is\_negative) {

integer\_part = std::abs(integer\_part);

}

binary = std::bitset<integer\_bits>(integer\_part).to\_string();

float fractional\_part = std::abs(number) - integer\_part;

binary += ".";

for (int i = 0; i < fractional\_bits; i++) {

fractional\_part \*= 2;

int bit = static\_cast<int>(fractional\_part);

binary += std::to\_string(bit);

fractional\_part -= bit;

}

if (is\_negative) {

binary = get\_2s\_complement(binary);

}

return binary;

}

std::string add(std::string a, std::string b) {

int n = a.length();

int m = b.length();

int max\_len = std::max(n, m);

std::string sum = "";

int carry = 0;

if (n < max\_len) {

a = std::string(max\_len - n, '0') + a;

} else if (m < max\_len) {

b = std::string(max\_len - m, '0') + b;

}

for (int i = max\_len - 1; i >= 0; i--) {

if (a[i] == '.') {

sum = "." + sum;

continue;

}

if (a[i] == '1' && b[i] == '1') {

if (carry == 1) {

sum = "1" + sum;

} else {

sum = "0" + sum;

}

carry = 1;

} else if (a[i] == '0' && b[i] == '0') {

if (carry == 1) {

sum = "1" + sum;

} else {

sum = "0" + sum;

}

carry = 0;

} else {

if (carry == 1) {

sum = "0" + sum;

carry = 1;

} else {

sum = "1" + sum;

carry = 0;

}

}

}

return sum;

}

std::string mul(std::string a, std::string b) {

std::string res, tmp\_res;

std::string null\_string((integer\_bits + fractional\_bits) << 1, '0');

a.erase(integer\_bits, 1);

b.erase(integer\_bits, 1);

res = null\_string;

if (a[0] == '0' && b[0] == '1') {

std::swap(a, b);

}

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

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

for (size\_t i = 0; i < b.size(); i++) {

if (b[i] == '0') {

continue;

}

tmp\_res = null\_string;

for (size\_t j = i; j < tmp\_res.size(); j++) {

tmp\_res[j] = a.back();

}

for (size\_t j = i; j < i + a.size(); j++) {

tmp\_res[j] = a[j - i];

}

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

std::reverse(tmp\_res.begin(), tmp\_res.end());

res = add(res, tmp\_res);

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

}

res = res.insert(fractional\_bits << 1, ".");

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

return res;

}

std::string sub(std::string a, std::string b) {

std::string res;

int carry = 0, tmp;

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

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

for (size\_t i = 0; i < a.size(); i++) {

if (a[i] == '.') {

res += '.';

continue;

}

tmp = (a[i] - b[i] - carry);

if (tmp < 0) {

tmp = std::abs(tmp);

carry = 1;

res += (tmp % 2) + 48;

} else {

res += tmp + 48;

carry = 0;

}

}

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

return res;

}

std::string div(std::string a, std::string b) {

if (b.find("1") == std::string::npos) {

throw std::invalid\_argument("division by zero occurred!");

}

std::string res, tmp, A, Q, old\_a;

Q = a;

Q.erase(integer\_bits, 1);

b.erase(integer\_bits, 1);

for (size\_t i = 0; i < 8; i++) {

Q += '0';

b = b[0] + b;

}

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] == b[0]) {

A = sub(A, b);

} else {

A = add(A, b);

}

if (A[0] == old\_a[0]) {

Q[Q.size() - 1] = '1';

} else {

Q[Q.size() - 1] = '0';

A = old\_a;

}

}

if (A[0] != b[0]) {

Q = get\_2s\_complement(Q);

}

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

return Q;

}

float bin\_to\_dec(std::string num) {

int pos = num.find(".");

float res = 0;

bool is\_neg = 0;

if (num[0] == '1') {

is\_neg = 1;

num = get\_2s\_complement(num);

}

for (int i = pos - 1; i >= 0; i--) {

res += std::pow(2, pos - i - 1) \* (num[i] - '0');

}

for (int i = pos + 1; i < static\_cast<int>(num.size()); i++) {

res += std::pow(2, -(i - pos)) \* (num[i] - '0');

}

if (is\_neg) {

res \*= -1;

}

return res;

}

float perform\_add(float a, float b) {

return bin\_to\_dec(add(to\_fixed\_point\_str(a), to\_fixed\_point\_str(b)));

}

float perform\_sub(float a, float b) {

b = -b;

return perform\_add(a, b);

}

float perform\_mul(float a, float b) {

int sign = a \* b < 0 ? -1 : 1;

return sign \* bin\_to\_dec(mul(to\_fixed\_point\_str(std::abs(a)), to\_fixed\_point\_str(std::abs(b))));

}

float perform\_div(float a, float b) {

int sign = a \* b < 0 ? -1 : 1;

return sign \* bin\_to\_dec(div(to\_fixed\_point\_str(std::abs(a)), to\_fixed\_point\_str(std::abs(b))));

}

int main() {

std::cout << "Enter two numbers: ";

float a, b;

while (!(std::cin >> a >> b)) {

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cout << "Invalid input, enter two numbers: ";

}

std::cout << "a = " << to\_fixed\_point\_str(a) << "\n";

std::cout << "b = " << to\_fixed\_point\_str(b) << "\n\n";

std::cout << "Sum = " << perform\_add(a, b) << "\n";

std::cout << "Subtraction = " << perform\_sub(a, b) << "\n";

std::cout << "Multiply = " << perform\_mul(a, b) << "\n";

try {

std::cout << "Division = " << perform\_div(a, b) << "\n";

} catch (const std::exception& e) {

std::cout << e.what() << '\n';

}

return 0;

}