Министерство образования Республики Беларусь  
Учреждение образования   
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №3

По дисциплине «Архитектура вычислительных систем»  
По теме «ОПЕРАЦИЯ ДЕЛЕНИЯ С ФИКСИРОВАННОЙ ТОЧКОЙ»

Выполнил:  
студент гр. 153503  
Киселёва Елизавета Андреевна

Проверил:

Преподаватель

Калиновская Анастасия Александровна

Минск 2023

# 1 Цель работы

Рассмотреть представление чисел в прямом, обратном и дополнительном кодах. Изучить процессы выполнения деления над целыми числами с фиксированной точкой. Составить программу реализации алгоритма деления.

**2 Постановка задачи**

***Задание к лабораторной работе 3***

Написать программу эмулятора АЛУ, реализующего операцию деления *с фиксированной точкой*над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

# 3 Теоретические сведения

**3.1 Прямой код**

Существует несколько соглашений о едином формате представления как положительных, так и отрицательных чисел. Всех их объединяет то, что старший бит слова (с точки зрения европейца — самый левый, или бит, которому при представлении числа без знака должен быть приписан самый большой вес) является битом хранения знака или знаковым разрядом. Все последующие биты слова представляют значащие разряды числа, которые в каждом формате интерпретируются по-своему. Значение 1 в знаковом разряде интерпретируется как представление всем словом отрицательного числа.

|  |  |
| --- | --- |
| 0001001 = | +18 |
| 10010010 = | -18 |

Формат представления чисел в прямом коде неудобен для использования в вычислениях. Во-первых, сложение и вычитание положительных и отрицательных чисел выполняется по-разному, а потому требуется анализировать знаковые разряды операндов. Во-вторых, в прямом коде числу 0 соответствуют две кодовых комбинации:

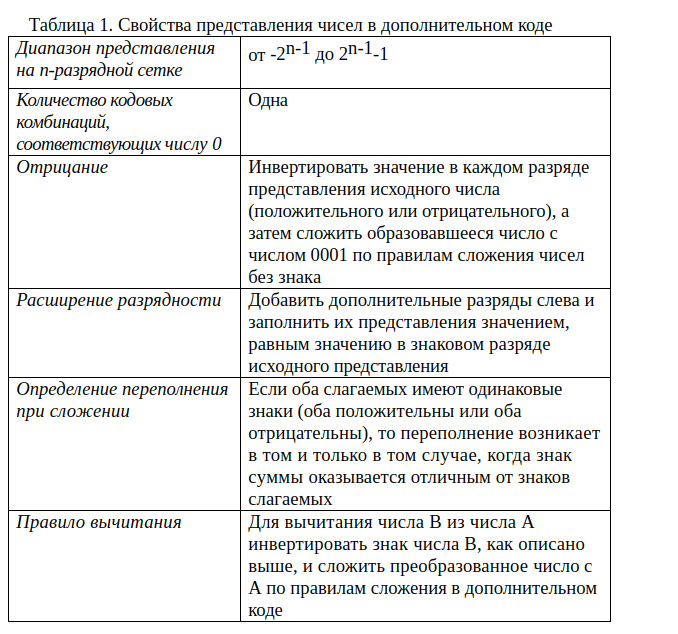
|  |  |
| --- | --- |
| 0000000= | +010 |
| 10000000= | -010 |
|  |  |

Это также неудобно, поскольку усложняется анализ результата на равенство нулю, а такая операция в программах встречается очень часто.

Из-за этих недостатков прямой код практически не применяется при peaлизации в АЛУ арифметических операций над целыми числами. Вместо этого более широкое применение находит другой формат, получивший наименование дополнительного кода.

## **3.2 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.



В большинстве описаний дополнительного кода основное внимание уделяется технике формирования представления отрицательного числа по представлению соответствующего положительного, причем не приводится формальное доказательство работоспособности описанной схемы. Мы решили нарушить эту традицию, и в данном разделе, а также в следующем будем основываться на описании, в котором это представление рассматривается в терминах взвешенной суммы значений разрядов. Такой способ мы уже использовали выше при описании представления чисел без знака и целых чисел со знаком в прямом коде. Преимущество такой методики в том, что она не оставляет ни малейших сомнений в справедливости излагаемых правил выполнения арифметических операций в любых частных случаях.

**3.3 Деление**

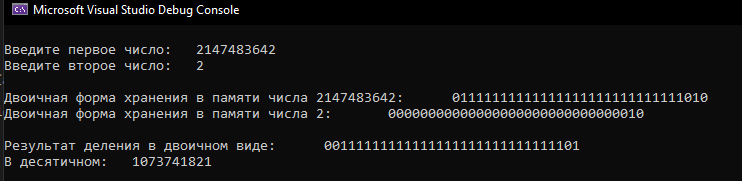
По сравнению с умножением операция деления выполняется несколько сложнее, хотя соответствующие алгоритмы основываются на тех же принципах поразрядного анализа операндов. Исходный алгоритм, как и при умножении, — тот, который используется при вычислении вручную, карандашом на бумаге. Алгоритм состоит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

На рис. К.6 показан пример "длинного" деления целых чисел без знака. Чрезвычайно полезно детально рассмотреть последовательность шагов этого алгоритма. Сначала разряды делимого анализируются слева направо до тех пор, пока последовательность разрядов не будет представлять число, большее, чем делитель, или равное ему. При делении этого числа на делитель получается частное, которое больше или равно 1. До тех пор, пока такое число не будет обнаружено при просмотре делимого(событие превышения), в частное заносятся коды 0, которые последовательно сдвигаются слева направо. Когда возникает событие превышения, в частное заносится 1, а делитель вычитается из частичного делимого. Результат этой операции называется частичным остатком. Далее выполняется циклическая процедура. В каждом цикле в частичный остаток добавляется очередной разряд делимого, и так до тех пор, пока при анализе частичного остатка не наступит событие превышения. В этом случае делитель вычитается из частичного остатка и формируется новый частичный остаток, а в очередной разряд частного заносится код 1. Процесс продолжается до тех пор, пока не будут проанализированы все разряды делимого.

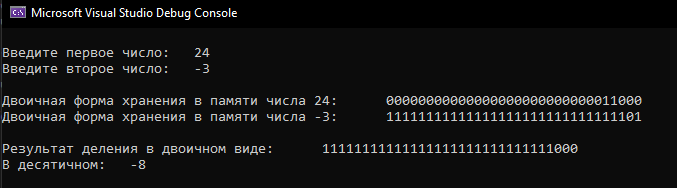
# 4 Пример работы программы

# 

# Тестовый пример 1



# Тестовый пример 2



# Тестовый пример 3

# 5 Выводы

В ходе выполнения лабораторной работы была написана программа, реализующая операции деления над числами с фиксированной точкой в дополнительном коде.

# Приложение 1. Текст программы

#include <iostream>

#include <functional>

#include <regex>

using namespace std;

int ram\_a[33];

int reg\_m[33];

int reg\_a[33];

int reg\_q[33];

int number\_1[33];

int number\_2[33];

int result[33];

int mask[33];

long long a1, a2;

void convert\_to\_binary(int mask[], long long n) {

int j = 31;

while (n > 0) {

mask[j--] = n % 2;

n /= 2;

}

}

void print\_array(int a[]) {

for (int i = 0; i < 32; ++i)

cout << a[i];

}

void to\_additional(int a[]) {

for (int i = 0; i < 32; ++i)

a[i] = (a[i] + 1) % 2;

int j = 31;

a[j]++;

while (a[j] == 2) {

a[j--] %= 2;

a[j]++;

}

}

int toDec(int a[]) {

int result = -(a[0] \* pow(2, 32));

for (int i = 1; i < 32; ++i)

result += a[i] \* pow(2, 31 - i);

return result;

}

void sum(int a[], int b[], int code) {

int j = 31, ram = 0;

while (j >= 0) {

result[j] = a[j] + b[j] + ram;

ram = result[j] / 2;

result[j] %= 2;

j--;

}

}

void shift() {

int ram = reg\_q[0];

for (int i = 0; i < 31; ++i)

reg\_q[i] = reg\_q[i + 1];

reg\_q[31] = 0;

for (int i = 0; i < 31; ++i)

reg\_a[i] = reg\_a[i + 1];

reg\_a[31] = ram;

}

void push(int a[], int b[]) {

if (a[0] == 1)

for (int i = 0; i < 32; ++i)

reg\_a[i] = 1;

if (a[0] == 0)

for (int i = 0; i < 32; ++i)

reg\_a[i] = 0;

for (int i = 0; i < 32; ++i)

reg\_q[i] = a[i];

for (int i = 0; i < 32; ++i)

reg\_m[i] = b[i];

}

void substraction(int a[], int b[]) {

to\_additional(b);

sum(a, b, 2);

for (int i = 0; i < 32; ++i)

a[i] = result[i];

}

std::pair<bool, bool> check\_zero() {

int s1 = 0;

int s2 = 0;

for (int i = 0; i < 32; ++i) {

s1 += reg\_a[i];

s2 += reg\_q[i];

}

return std::make\_pair(s1 == 0, s2 == 0);

}

void divide(int a[], int b[]) {

push(a, b);

if (a2 == 0) {

std::cout << "\n\nДЕЛЕНИЕ НА НОЛЬ\n";

exit(0);

}

if ((a1 == a2) && (a1 < 0 || a2 < 0)) {

for (int i = 0; i < 33; ++i)

reg\_q[i] = 1;

return;

}

if (a1 == 0) {

for (int i = 0; i < 33; ++i)

reg\_q[i] = 0;

return;

}

if (a1 != 0)

for (int k = 0; k < 32; ++k) {

shift();

bool minus = reg\_a[0];

for (int i = 0; i < 32; ++i)

ram\_a[i] = reg\_a[i];

if (reg\_m[0] == reg\_a[0])

substraction(reg\_a, reg\_m);

else {

sum(reg\_a, reg\_m, 1);

for (int i = 0; i < 32; ++i)

reg\_a[i] = result[i];

}

bool new\_znak = reg\_a[0];

bool exp = false;

if (minus == new\_znak)

exp = true;

auto zero = check\_zero();

if (exp == true || (zero.first == true && zero.second == true))

reg\_q[31] = 1;

else if (exp == false && (zero.first == false && zero.second == false)) {

reg\_q[31] = 0;

for (int i = 0; i < 32; ++i)

reg\_a[i] = ram\_a[i];

}

}

if (a1 / abs(a1) != a2 / abs(a2)) {

to\_additional(reg\_q);

}

}

template<typename ReturnValue>

ReturnValue CorrectInput(const string& variable\_name, const regex& pattern,

function<ReturnValue(string& input)> converter) {

string value;

do {

cout << "Введите " << variable\_name << ":\t";

cin >> value;

} while (!regex\_match(value, pattern));

return converter(value);

}

int main() {

setlocale(LC\_ALL, "rus");

std::cout << '\n';

a1 = CorrectInput<long long>(std::string("первое число"), std::regex("-?[0-9]\*"), [](std::string& s) {return atoi(s.c\_str()); });

a2 = CorrectInput<long long>(std::string("второе число"), std::regex("-?[0-9]\*"), [](std::string& s) {return atoi(s.c\_str()); });

convert\_to\_binary(number\_1, abs(a1));

convert\_to\_binary(number\_2, abs(a2));

if (a1 < 0 && a2 < 0) {

int n1[33];

int n2[33];

for (int i = 0; i < 33; ++i) {

n1[i] = number\_1[i];

n2[i] = number\_2[i];

}

if (a1 < 0)

to\_additional(n1);

if (a2 < 0)

to\_additional(n2);

std::cout << "\nДвоичная форма хранения в памяти числа " << a1 << ":\t";

print\_array(n1);

std::cout << "\nДвоичная форма хранения в памяти числа " << a2 << ":\t";

print\_array(n2);

}

if (a1 > 0 || a2 > 0) {

if (a1 < 0)

to\_additional(number\_1);

if (a2 < 0)

to\_additional(number\_2);

std::cout << "\nДвоичная форма хранения в памяти числа " << a1 << ":\t";

print\_array(number\_1);

std::cout << "\nДвоичная форма хранения в памяти числа " << a2 << ":\t";

print\_array(number\_2);

}

divide(number\_1, number\_2);

int res = toDec(reg\_q);

std::cout << "\n\nРезультат деления в двоичном виде:\t";

print\_array(reg\_q);

std::cout << "\nВ десятичном:\t" << res << '\n';

return 0;

}