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

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторным работам №№1-3

По дисциплине «Архитектура вычислительных систем»  
По теме «Арифметические операции с целыми числами»

Выполнил:  
студент гр. 953501  
 Гришаев Н.Г.

Проверил:

Старший преподаватель

Шиманский В.В.

Минск 2021

**Содержание**

[1. Цель работы 3](#_heading=h.gjdgxs)

[2. Постановка задачи 4](#_heading=h.30j0zll)

[3. Теоретические сведения 5](#_heading=h.1fob9te)

[3.1 Прямой код 5](#_heading=h.3znysh7)

[3.2 Дополнительный код 7](#_heading=h.2et92p0)

[3.3 Сложение и вычитание 8](#_heading=h.tyjcwt)

[3.4 Умножение и деление 9](#_heading=h.3dy6vkm)

[4. Программная реализация 12](#_heading=h.1t3h5sf)

[5. Выводы 16](#_heading=h.2s8eyo1)

[Список литературы 17](#_heading=h.17dp8vu)

[Приложение 1. Текст программы 18](#_heading=h.3rdcrjn)

# 1. Цель работы

Рассмотреть представление чисел в прямом, обратном и дополнительном кодах. Изучить процессы выполнения арифметических операций над целыми числами с фиксированной точкой. Рассмотреть различные алгоритмы выполнения арифметических операций. Сравнить различные варианты алгоритмов и определить наиболее подходящие для реализации в АЛУ.

# 2. Постановка задачи

***Задание к лабораторной работе 1***

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

***Задание к лабораторной работе 2***

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

***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

Существует ряд способов хранения и представления целых чисел с фиксированной точкой в памяти компьютера. Среди них можно выделить: *прямой код* (ориентирован на представление положительных целых чисел в двоичном коде)*, код со сдвигом, дополнительный код* (дополнение до единицы или до двух).

При выборе способа хранения целых чисел, следует ориентироваться на следующие условия:

* код не должен требовать усложнения архитектуры процессора для выполнения арифметических операций с отрицательными числами;
* не усложнял арифметические действия;
* хранил бы одинаковое количество положительных и отрицательных чисел;
* и др.

В рамках данной лабораторной работы рассматривается представление целых чисел в *прямом* и *дополнительном* кодах, а также создается программная реализация методов хранения информации и возможности произведения арифметических операций над ними (сложение, вычитание, умножение, деление).

## 3.1 Прямой код

Прямой код — способ представления [двоичных](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) [чисел](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE) с [фиксированной запятой](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%BA%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D1%8F%D1%82%D0%B0%D1%8F) в [компьютерной арифметике](https://ru.wikipedia.org/w/index.php?title=%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D0%BA%D0%B0&action=edit&redlink=1). Главным образом используется для записи [неотрицательных чисел](https://ru.wikipedia.org/w/index.php?title=%D0%9D%D0%B5%D0%BE%D1%82%D1%80%D0%B8%D1%86%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D1%87%D0%B8%D1%81%D0%BB%D0%B0&action=edit&redlink=1).

Для представления целого положительного (неотрицательного) числа в компьютере используется следующее правило:

* число переводится в двоичную систему;
* результат дополняется нулями слева в пределах выбранного формата;
* последний разряд слева является знаковым, в положительном числе он равен 0.

При записи числа в прямом коде старший разряд (старший бит) объявляется знаковым (signed) разрядом (знаковым битом). Если знаковый бит равен 0, число [положительное](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D0%BE%D0%BB%D0%BE%D0%B6%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE&action=edit&redlink=1), иначе — [отрицательное](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D1%80%D0%B8%D1%86%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE). В остальных разрядах записывается двоичное представление [модуля](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D0%BE%D0%BB%D1%8E%D1%82%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B0) числа.

Функция представления двоичных чисел (в том числе целых чисел и смешанных дробей) в прямом коде имеет вид:



Величина числа А в прямом коде определяется по следующей формуле:



где:  
 *i* - номер разряда числа; отрицательное число — номер разряда справа от запятой; положительное число — номер разряда слева от запятой;

*k* - количество разрядов справа от запятой (кол-во разрядов дробной части числа);

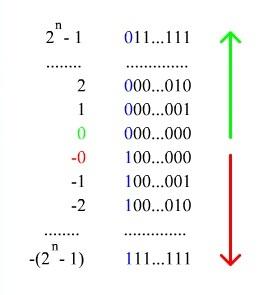
*n* - количество разрядов слева от запятой (кол-во разрядов целой части числа);

*p* - основание [системы счисления](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F); равно 2 для [двоичных](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) чисел, 10 — для [десятичных](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%81%D1%8F%D1%82%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), 16 — для [шестнадцатеричных](https://ru.wikipedia.org/wiki/%D0%A8%D0%B5%D1%81%D1%82%D0%BD%D0%B0%D0%B4%D1%86%D0%B0%D1%82%D0%B5%D1%80%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) и т. п.;

Например, число 6 в двоичном 8-разрядном представлении имеет вид: ***0000 0110***

Как и было сказано вначале – прямой код главным образом используется для записи и представления положительных (неотрицательных) чисел, потому что при использовании для чисел со знаком, у прямого кода есть два основных недостатка:

* В прямом коде есть два варианта записи числа [0](https://ru.wikipedia.org/wiki/0_(%D1%87%D0%B8%D1%81%D0%BB%D0%BE)) (например, 00000000 и 10000000 в восьмиразрядном представлении). Второе представление называется «[отрицательный ноль](https://ru.wikipedia.org/wiki/%E2%88%920_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))»
* Использование прямого кода для представления отрицательных чисел в [памяти](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C) [компьютера](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) предполагает или выполнение арифметических операций [центральным процессором](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BD%D1%82%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) в прямом коде, или перевод чисел в другое представление (например, в [дополнительный код](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BF%D0%BE%D0%BB%D0%BD%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4_(%D0%BF%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0))) перед выполнением операций и перевод результатов обратно в прямой код (что неэффективно).

  
Рисунок 1. Нумерация двоичных чисел в двоичном представлении

## 3.2 Дополнительный код

Дополнительный код — наиболее распространённый способ представления [отрицательных целых чисел](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D1%80%D0%B8%D1%86%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) в [компьютерах](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80). Он позволяет заменить операцию вычитания на операцию сложения и сделать операции сложения и вычитания одинаковыми для знаковых и [беззнаковых](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%B7%D0%BD%D0%B0%D0%BA%D0%BE%D0%B2%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) чисел, чем упрощает архитектуру [ЭВМ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80).

Дополнительный код для отрицательного числа можно получить инвертированием его двоичного модуля (первое дополнение) и прибавлением к инверсии единицы (второе дополнение), либо вычитанием числа из нуля.

Двоичное 8-разрядное число *со знаком* в дополнительном коде может представлять любое целое в диапазоне от −128 до +127. Если старший разряд равен нулю, то наибольшее целое число, которое может быть записано в оставшихся 7 разрядах, равно 127.

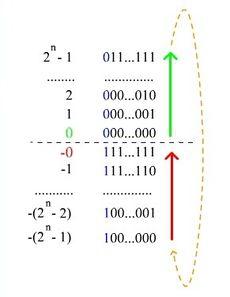
Достоинства представления чисел с помощью дополнительного кода:

* простое получение кода отрицательных чисел;
* из-за того, что 00 обозначает ++, коды положительных чисел относительно беззнакового кодирования остаются неизменными.
* количество положительных чисел равно количеству отрицательных.

Недостатки представления чисел с помощью кода с дополнением до единицы

* выполнение арифметических операций с отрицательными числами требует усложнения архитектуры центрального процессора;
* существуют два нуля: +0+0 и −0−0.

На рисунке 2 представлено изображение двоичных чисел в дополнительном коде:

  
Рисунок 2. Нумерация чисел в дополнительном коде

Например, переведём число −13 в двоичный 8-разрядный код. Прямой код модуля −13: 00001101. Инвертируем и получаем 11110010. Для получения из дополнительного кода самого числа достаточно инвертировать все разряды кода.

Так же важным с точки зрения работы с большими числами является понятие *расширения знака*:

Расширение знака — операция над двоичным числом, которая позволяет увеличить разрядность числа с сохранением знака и значения. Выполняется добавлением цифр со стороны старшего значащего разряда. Если число положительное (старший разряд равен 0), то добавляются нули, если отрицательное (старший разряд равен 1) — единицы.

Увеличение разрядности позволяет значительно увеличить диапазон значений числа. Например, в 16-битном представлении можно хранить 65 536 значений (формула была указана выше), в отличие от 256 в 8-битном представлении. Аналогичные преобразования для расширения разрядности используются и в *прямом коде*.

## 3.3 Сложение и вычитание

Рассмотрим операцию сложения на примере:

*Выполнить сложение чисел A = 53 и B = 14 в двоичной системе счисления.*

Переведем числа в двоичную систему счисления:

5310 **= 110101**2 ,  
1410 **= 1110**2 .

Запишем числа «A» и «B» столбиком, одно под другим, начиная с младших разрядов (нумерация разрядов начинается с нуля).

**Сложим** поразрядно числа «A» и «B» записывая результат в «C» начиная с младших разрядов. Весь процесс сложения наших чисел представлен в таблице 1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Разряды | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| А | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| B | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| C | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

Таблица 1. Сложение чисел 53 и 14

Теперь опишем как происходит описанный выше процесс сложения и вычитания чисел в нашем эмуляторе АЛУ:

Изучим блоки эмулятора АЛУ. Главный блок представляет собой модуль сложения двух чисел. Процесс сложения происходит по правилам сложения беззнаковых чисел. Вычитание чисел достигается инвертированием второго числа в противоположный знак. Данный блок назовем *сумматором*.

Результат передается либо в один из регистров слагаемых (этот вариант показан на схеме), либо в третий регистр результата. Кроме кода результата сумматор формирует сигнал переполнения, который фиксируется в битовом флаге переполнения. Значение флага интерпретируется следующим образом: 0 — переполнение отсутствует, 1 – присутствует. При выполнении операции вычитания код вычитаемого, хранящийся перед началом операции в регистре В, передается на схему, выполняющую операцию отрицания, а уже с выхода этой схемы код поступает на вход сумматора.

## 3.4 Умножение и деление

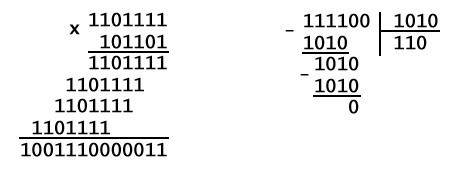
Умножение и деление двоичных чисел практически не отличается от умножения и деления чисел, записанных в десятичной системе счисления. Единственным отличием является то, что при умножении в столбик не приходится находить произведение первого множителя на значения последовательных разрядов второго множителя, так как значение этих разрядов 1 или 0. А при делении в столбик не нужно подбирать неполное делимое, так как учитывая специфику двоичных чисел, неполное делимое можно определить просто, посмотрев на делимое.

Введем правила умножение отдельных битов на таблице 2.

|  |  |  |
| --- | --- | --- |
| Первый | Второй | Результат |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Таблица 2. Умножение отдельных битов

Пример умножения и деления чисел введем на рисунке 3. Как видно, правила визуально коррелируют с визуальным представ сложением

  
Рисунок 3. Правила умножения и деления двоичного представления чисел

Реализация алгоритма умножения техническими или программными средствами позволяет несколько повысить его эффективность по сравнению с тем вариантом, который мы традиционно используем при вычислении в столбик вручную.

  Конструкторы АЛУ предпочитают способ, который не требует выполнения дополнительного преобразования после завершения умножения. Одним из таких способов является *алгоритм Бута* *(Booth).* Этот алгоритм также позволяет ускорить выполнение операции. Сомножители размещаются в регистрах Q (множитель) и М (множимое). Кроме них имеется одноразрядный регистр Q-1 который связан с младшим разрядом (Q0) регистра Q. Произведение формируется в регистрах А и Q. В исходном состоянии в регистрах А и Q, записаны нули. Как и ранее, схема управления анализирует разряды множителя, но на сей раз анализируется пара соседних разрядов — основной и тот, который находится справа от него. Если оба разряда имеют одинаковые значения (11 или 00), все разряды регистров A, Q и Q-1 сдвигаются на 1 разряд вправо. Если соседние разряды имеют отличающиеся коды, то выполняется сложение или вычитание кода, множимого из содержимого регистра А. Сложение, выполняется при комбинации кодов в соседних разрядах 01, а вычитание — при комбинации 10. Вслед за сложением или вычитанием выполняется сдвиг кодов в регистрах на один разряд вправо. Сдвиг выполняется таким образом, что старший разряд (крайний левый) регистра А (разряд Аn-1) сохраняется, хотя и переписывается в разряд Аn-2. Это необходимо для сохранения знака кода в регистрах А и Q. Такую операцию принято называть сдвигом с сохранением знака или арифметическим сдвигом

Для выполнения деления двоичных знаковых чисел также существует алгоритм, значительно отличающийся от ручного способа, но дающий преимущества в реализации в АЛУ. Ниже описан один из вариантов алгоритма деления чисел, представленных в дополнительном коде.

a.     Загрузить делитель в регистр М, а делимое — в регистры А и Q. Делимое должно иметь формат 2n -разрядного дополнительного кода. Например, 4-разрядное число 0111 должно быть представлено в формате 00000111, а число 1001 должно быть преобразовано в 11111001.

b.     Сдвинуть содержимое регистров А и Q на один разряд влево.

c.     Если коды в регистрах М и А имеют одинаковые знаки, вычесть из содержимого А содержимое М и оставить результат в А. В противном случае добавить к содержимому А код из М.

d.     Предыдущая операция считается успешной, если знак кода в регистре А не изменился в результате ее выполнения.

·Если операция была успешной или содержимое регистров А и Q равно нулю, то установить в младшем разряде частного (Q0) код 1.

· Если операция не увенчалась успехом и содержимое одного из регистров А и Q (или обоих) отлично от нуля, то установить в младшем разряде частного (Q0) код 0 и восстановить прежнее значение в регистре А.

e.     Повторять операции, указанные в пунктах 2 и 4, столько раз, сколько разрядов в регистре Q.

f.     После завершения операции прочесть значение остатка в регистре А. Если знаки делимого и делителя одинаковы, значение частного извлечь из регистра Q; в противном случае значение частного равно содержимому регистра Q с обратным знаком (операция отрицания должна быть выполнена по правилам для дополнительного кода).

# 4. Программная реализация

В качестве языка для реализации эмулятора АЛУ был выбран язык C++.

Для работы с операциями сложения, вычитания, умножения и деления предусмотрено хранение двоичных чисел в 16-битном и сдвоенном 32-битном регистрах, что предусматривает диапазон в 65536 (216) и 4294967296 (232) значений соответственно. Предусмотрена возможность обработки ситуаций переполнения с выводом информации о проблеме.

На рисунках представлен результат работы программы. Выполняется сложение и вычитание двух чисел с фиксированной точкой. Демонстрируются операции умножения двух целых чисел и деления с восстановлением остатка. Показана обработка переполнений при операциях сложения, отрицания и деления.

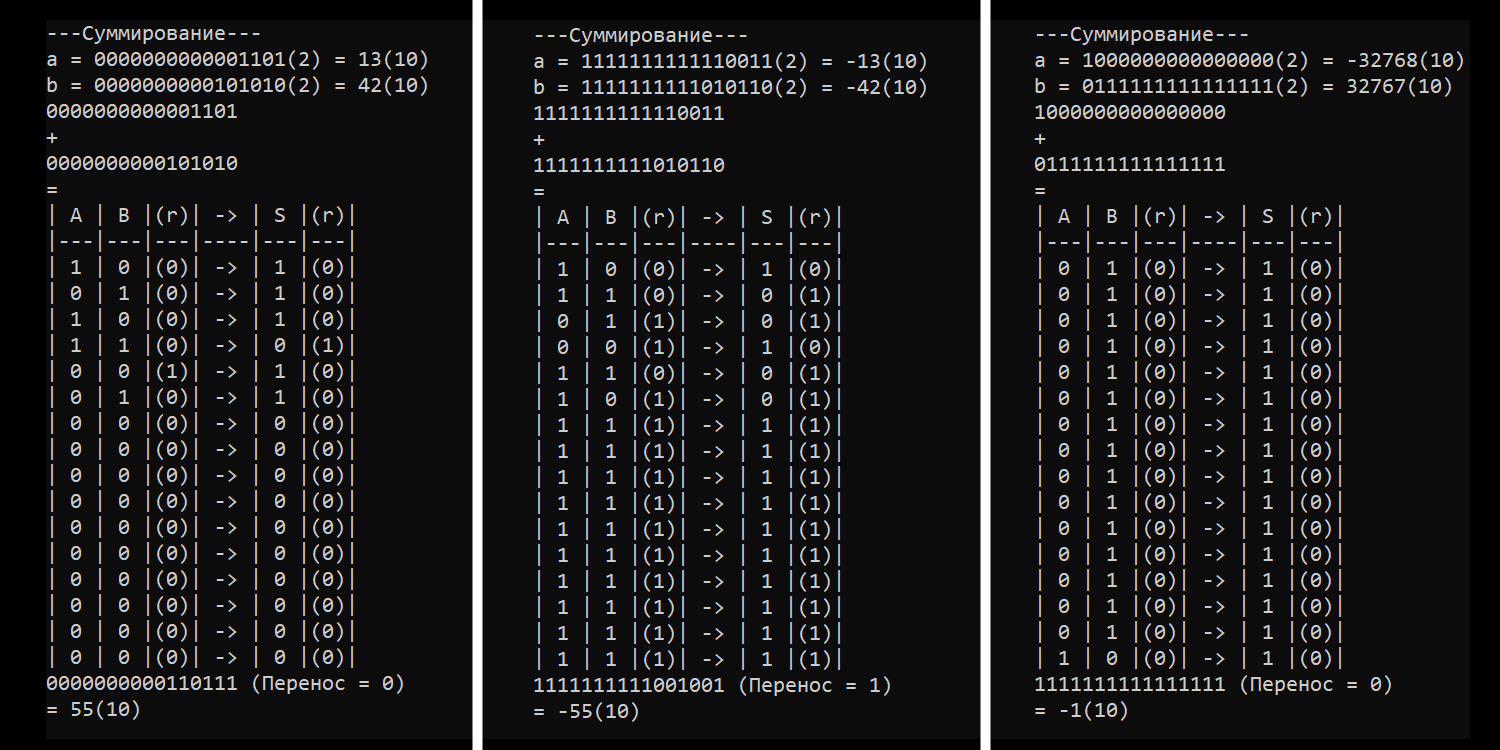


Рис.1 Успешное суммирование двух чисел

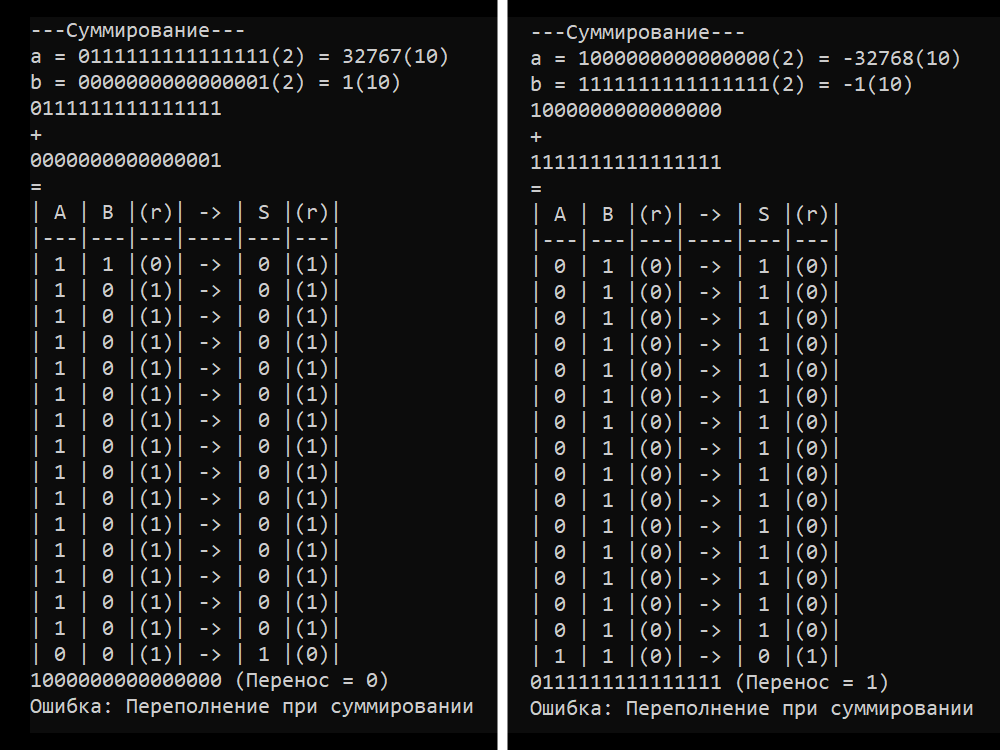


Рис.2 Обработка переполнения при суммировании

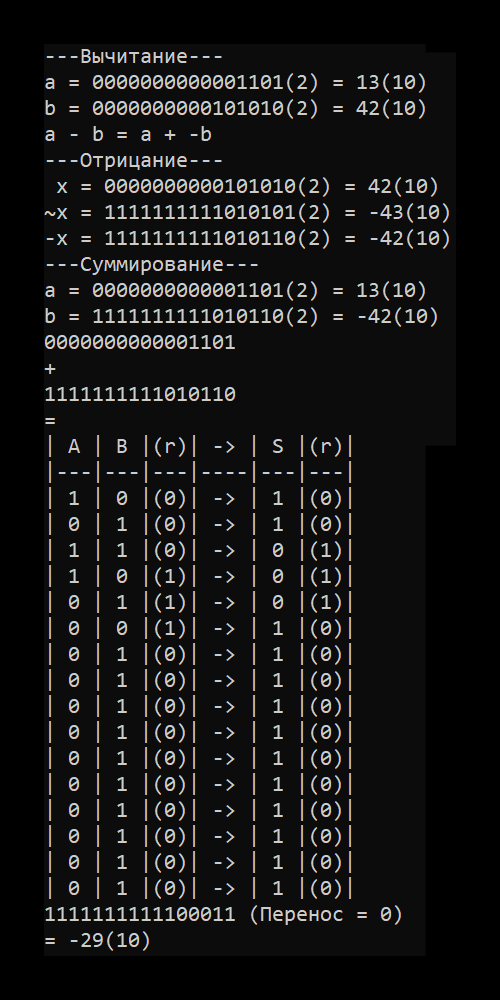


Рис.3 Успешное вычитание двух чисел

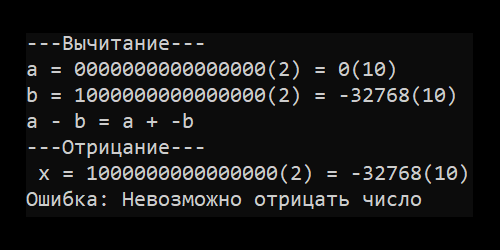


Рис.4 Обработка невозможности выполнения вычитания

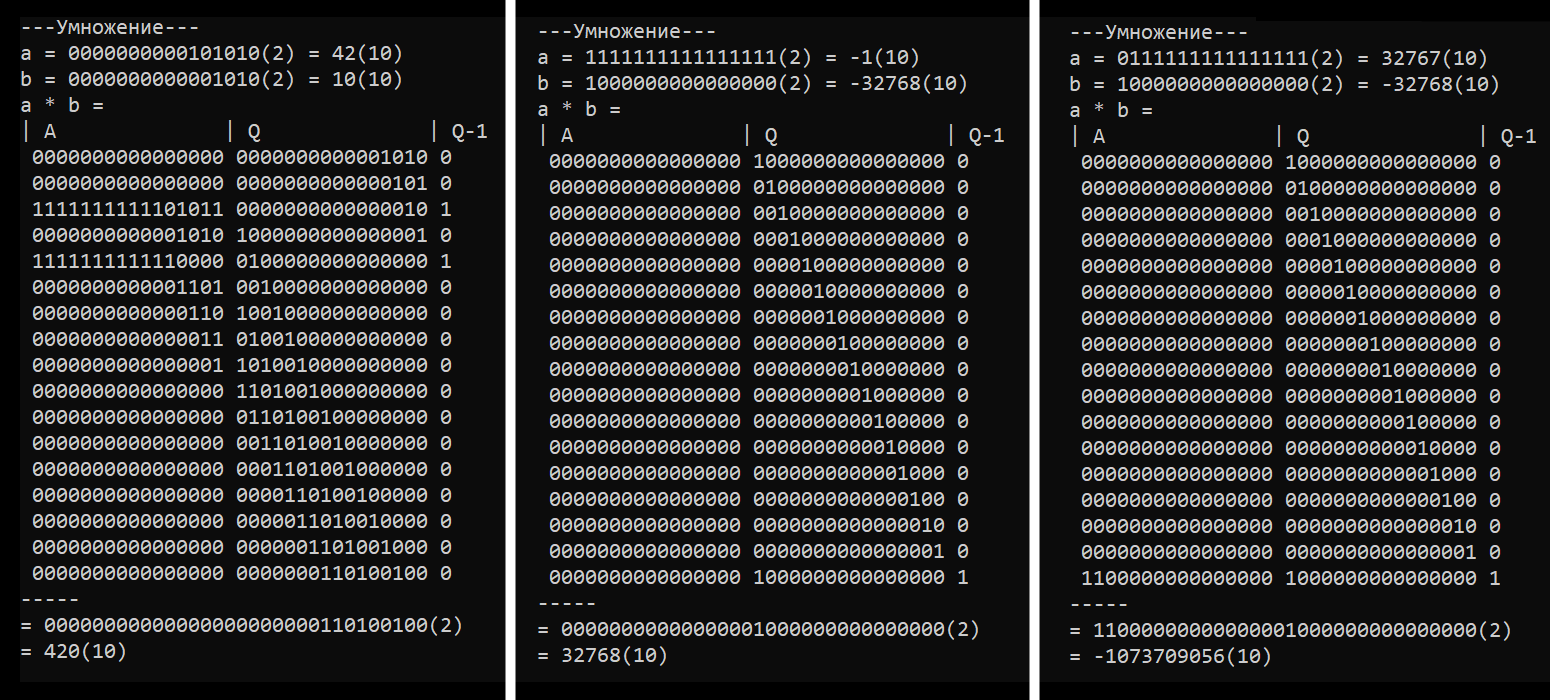


Рис.5 Успешное перемножение двух чисел

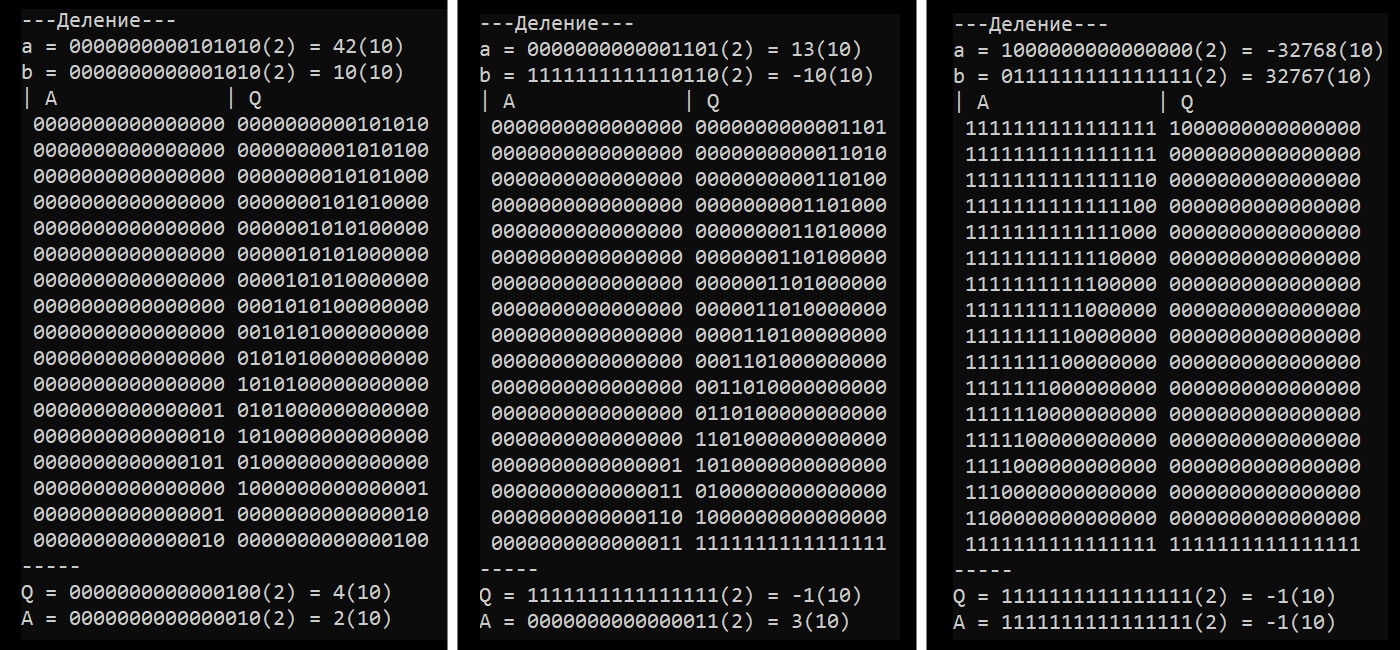


Рис.6 Успешное деление двух чисел

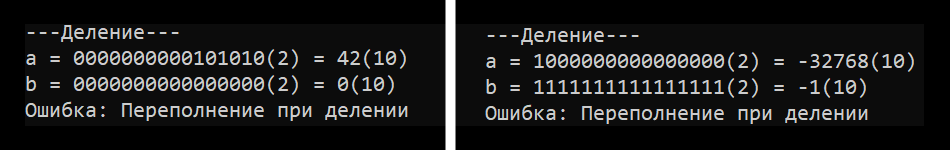


Рис.7 Обработка переполнения при делении

# 5. Выводы

Дополнительный код позволяет хранить знаковые целые числа с фиксированной точкой и проводить над ними арифметические операции.

Операция отрицания состоит из двух последовательных команд: инвертирования всех битов числа и добавления единицы. Из-за ограниченности диапазона значений в дополнительном коде, невозможно отрицать минимальное число (число формата 100…002), так как в результате выполнения команд получится то же самое число.

Операция сложения целых чисел в дополнительном коде представляет собой обычное суммирование «в столбик», начиная с младшего разряда, при этом перенос из старшего разряда игнорируется.

При выполнении сложения чисел с одинаковыми знаками результат может не поместиться в используемую разрядную сетку. Такое явление расценивается как переполнение, и для его обнаружения используется следующее правило: Если знаки слагаемых совпадают, то переполнение возникает, если знак суммы отличается от знака слагаемых.

Операция вычитания выполняется по следующему правилу: Для вычитания одного числа (вычитаемого) из другого (уменьшаемого) необходимо предварительно выполнить операцию отрицания над вычитаемым, а затем добавить результат к уменьшаемому по правилам сложения в дополнительном коде.

Одним из способов умножить два числа в дополнительном коде является алгоритм Бута, который позволяет обойтись меньшим количеством операций сложения и вычитания, чем простейший алгоритм, а также не требует выполнения дополнительного преобразования после завершения умножения.

Для выполнения деления двоичных знаковых чисел в дополнительном коде также существует алгоритм, значительно отличающийся от ручного способа, но дающий преимущества в реализации в АЛУ. Этот алгоритм также вычисляет остаток от деления.

При выполнении операции деления может возникнуть ситуация переполнения, если осуществляется попытка деления на ноль или в случае, если частное или остаток превышают размерность регистра.

В ходе выполнения лабораторной работы была написана программа, реализующая основные арифметические операции над числами в дополнительном коде. Также осуществлена проверка на возникновение исключительных ситуаций в процессе вычислений.

# Список литературы

1. Волорова Н. А. Лабораторный практикум по курсу «Архитектура вычислительных систем» для студентов специальности «Информатика» / 93-444-487-2- Мн.: БГУИР, 2003. – 32с.:ил.

# Приложение 1. Текст программы

#include <iostream>

#include <iomanip>

using std::cin;

using std::cout;

using std::string;

using std::endl;

using std::pair;

class Register

{

#define WordSize 16

#define Number long

bool bits[WordSize];

public:

static const Number Minimum = -(1 << (WordSize - 1));

static const Number Maximum = (1 << (WordSize - 1)) - 1;

bool& operator [] (int i)

{

return bits[i];

}

bool SignBit()

{

return bits[WordSize - 1];

}

Register() {

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

bits[i] = 0;

}

}

Register(Number x) {

\*this = x;

}

Register& operator = (Register& x) {

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

bits[i] = x[i];

}

return \*this;

}

Register& operator = (Number x) {

if ((x < Minimum) || (Maximum < x)) {

throw string("Присваиваемое число не помещается в регистр");

}

Register ans;

bool neg = x < 0;

x = abs(x);

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

ans[i] = x % 2;

x /= 2;

}

if (neg) {

ans.Inverse().Inc();

}

\*this = ans;

return \*this;

}

Register& Inverse() {

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

bits[i] = !bits[i];

}

return \*this;

}

Register& Inc() {

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

if (bits[i]) {

bits[i] = 0;

} else {

bits[i] = 1;

break;

}

if (i + 1 == WordSize) {

//cout << "IncOverflow" << endl;

}

}

return \*this;

}

Register& Shl(int x) {

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

bits[i] = bits[i - x];

}

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

bits[i] = 0;

}

}

Register& Shr(int x) {

for (int i = 0; i + x < WordSize; i++) {

bits[i] = bits[i + x];

}

for (int i = WordSize - x; i < WordSize; i++) {

bits[i] = 0;

}

}

Register& Sar(int x) {

bool sign = SignBit();

for (int i = 0; i + x < WordSize; i++) {

bits[i] = bits[i + x];

}

for (int i = WordSize - x; i < WordSize; i++) {

bits[i] = sign;

}

}

operator string() {

string s = "";

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

s = (bits[i] ? "1" : "0") + s;

}

return s;

}

operator Number() {

Register t = \*this;

Number ans = 0;

int sign = 1;

if (t.SignBit()) {

sign = -1;

t.Inverse().Inc();

}

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

ans = (ans << 1) + t[i];

}

ans \*= sign;

return ans;

}

friend std::ostream& operator<< (std::ostream& out, Register& x) {

out << string(x);

return out;

}

bool operator == (Register& x) {

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

if (bits[i] != x[i]) {

return false;

}

}

return true;

}

};

#define DoubleRegister pair<Register, Register>

bool DEBUG = true;

class ALU

{

public:

static void print(string s, Register x) {

cout << s << " = " << x << "(2) = " << int(x) << "(10)" << endl;

}

static Number StrToNum(string s) {

Number ans = 0;

int sign = 1;

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

sign = -1;

for (int i = 0; i < s.size(); i++) {

s[i] = '0' + ('1' - s[i]);

}

for (int i = s.size() - 1; i >= 0; i--) {

s[i] = '0' + ('1' - s[i]);

if (s[i] == '1') {

break;

}

}

}

for (int i = 0; i < s.size(); i++) {

ans = (ans << 1) + (s[i] - '0');

}

return sign \* ans;

}

static Register JustNeg(Register x) {

if (x == Register::Minimum) {

if (DEBUG) {

throw string("Невозможно отрицать число");

}

}

return x.Inverse().Inc();

}

static Register Negate(Register x) {

cout << "---Отрицание---" << endl;

Register ans = x;

print(" x", ans);

if (ans == Register::Minimum) {

throw string("Невозможно отрицать число");

}

ans.Inverse();

print("~x", ans);

ans.Inc();

print("-x", ans);

return ans;

}

static Register JustAdd(Register a, Register b) {

Register ans;

bool r = 0;

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

ans[i] = a[i] ^ b[i] ^ r;

r = (a[i] & b[i]) | (r & (a[i] | b[i]));

}

if ((!(a.SignBit() ^ b.SignBit())) & (ans.SignBit() ^ a.SignBit())) {

if (DEBUG) {

throw string("Переполнение при суммировании");

}

}

return ans;

}

static Register Addition(Register a, Register b) {

cout << "---Суммирование---" << endl;

print("a", a);

print("b", b);

cout << a << endl;

cout << "+" << endl;

cout << b << endl;

cout << "=" << endl;

Register ans;

bool r = 0;

cout << "| A | B |(r)| -> | S |(r)|" << endl;

cout << "|---|---|---|----|---|---|" << endl;

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

cout << "| " << a[i] << " | " << b[i] << " |(" << r << ")|";

ans[i] = a[i] ^ b[i] ^ r;

r = (a[i] & b[i]) | (r & (a[i] | b[i]));

cout << " -> | " << ans[i] << " |(" << r << ")|" << endl;

}

cout << ans << " (Перенос = " << r << ")" << endl;

if ((!(a.SignBit() ^ b.SignBit())) & (ans.SignBit() ^ a.SignBit())) {

throw string("Переполнение при суммировании");

}

cout << "= " << int(ans) << "(10)" << endl;

cout << endl;

return ans;

}

static Register JustSub(Register a, Register b) {

return JustAdd(a, JustNeg(b));

}

static Register Subtraction(Register a, Register b) {

cout << "---Вычитание---" << endl;

print("a", a);

print("b", b);

cout << "a - b = a + -b" << endl;

return Addition(a, Negate(b));

}

static DoubleRegister Multiplication(Register a, Register b) {

cout << "---Умножение---" << endl;

print("a", a);

print("b", b);

cout << "a \* b = " << endl;

Register ans;

bool Q = 0;

cout << "| A" << std::setw(WordSize+1) << " | Q" << std::setw(WordSize+3) << " | Q-1" << endl;

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

cout << " " << ans << " " << b << " " << Q << endl;

if (Q ^ b[0]) {

ans = JustAdd(ans, (Q ? a : JustNeg(a)));

}

Q = b[0];

b.Shr(1);

b[WordSize-1] = ans[0];

ans.Sar(1);

}

cout << " " << ans << " " << b << " " << Q << endl;

cout << "-----" << endl;

cout << "= " << ans << b << "(2)" << endl;

cout << "= " << StrToNum(string(ans) + string(b)) << "(10)" << endl;

cout << endl;

DEBUG = true;

return DoubleRegister(ans, b);

}

static DoubleRegister Division(Register a, Register b) {

cout << "---Деление---" << endl;

print("a", a);

print("b", b);

if ((b == 0) | ((a == Register::Minimum) & (b == -1))) {

throw string("Переполнение при делении");

}

DEBUG = false;

Register M = b, A = -a.SignBit(), Q = a;

bool sign = a.SignBit() ^ b.SignBit();

cout << "| A" << std::setw(WordSize+1) << " | Q" << endl;

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

cout << " " << A << " " << Q << endl;

A.Shl(1);

A[0] = Q.SignBit();

Q.Shl(1);

Register oldA = A;

if (A.SignBit() ^ M.SignBit()) {

A = JustAdd(A, M);

} else {

A = JustSub(A, M);

}

if ((!A.SignBit() ^ oldA.SignBit()) | ((A == 0) & (Q == 0))) {

Q[0] = 1;

} else {

Q[0] = 0;

A = oldA;

}

}

if (sign) {

Q = JustNeg(Q);

}

cout << " " << A << " " << Q << endl;

cout << "-----" << endl;

print("Q", Q);

print("A", A);

cout << endl;

DEBUG = true;

return DoubleRegister(Q, A);

}

};

#define Test(f) \

try { f; } catch (string ex) { cout << "Ошибка: " << ex << endl << endl; }

int main()

{

setlocale(LC\_ALL, "Russian");

int a, b;

cin >> a >> b;

Test(ALU::Addition(a, b));

Test(ALU::Subtraction(a, b));

Test(ALU::Multiplication(a, b));

Test(ALU::Division(a, b));

return 0;

}