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

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторным работам №№4-6

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

Выполнил:
студент гр. 953501
Кременевский. В.С

Проверил:
Старший преподаватель
Шиманский В.В.

Минск 2021

Содержание

1. Цель работы 3
2. Постановка задачи....4
3. Теоретические сведения 5
Основные положения....5
Стандарт IEEE формата с плавающей точкой 7
Сложение и вычитание 9
Умножение и деление 10
4. Программная реализация 12
Успешное деление 2х чисел: 22
Особоые ситуации. которые могут вознакнуть в ходе операций: 22
5. Выводы....27
Список литературы 28
Приложение 1. Текст программы....29

1. Цель работы

Рассмотреть представление чисел с плавающей точной в двоичном коде. Изучить алгоритмы выполнения основных арифметических операций над действительными числами с плавающей точкой. Написать программу, реализующую такие алгоритмы.

2. Постановка задачи

Задание к лабораторной работе 4

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

Задание к лабораторной работе 5

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

Задание к лабораторной работе 6

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

3. Теоретические сведения

Основные положения

В формате с фиксированной точкой, в частности в дополнительном коде, можно представлять положительные и отрицательные числа в диапазоне, симметричном на числовой оси относительно точки 0. Расположив воображаемую Разделяющую точку в середине разрядной сетки, можно в этом формате представлять не только целые, но и смешанные числа, а также дроби.

Однако такой подход позволяет представить на ограниченной разрядной сетке множество вещественных чисел в довольно узком диапазоне. Нельзя представить очень большие числа или очень маленькие. При выполнении деления двух больших чисел, как правило, теряется дробная часть частного.

При работе в десятичной системе счисления ученые давно нашли выход из положения, применяя для представления числовых величин так называемую научную нотацию. Так, число 976 000000 000 000 можно представить в виде $9.76x10^{14}$, а число 0,000000 000 000 0976 - в виде $9.76x10^{-14}$. При этом, фактически, разделительная точка динамически сдвигается в удобное место, а для того чтобы "уследить" за ее положением в качестве второго множителя - характеристики, - используется степень числа 10 (основания характеристики). Это позволяет с помощью небольшого числа цифр (т.е. чисел с ограниченной разрядностью) с успехом представлять как очень большие, так и очень малые величины.

Этот же подход можно применить и в двоичной системе счисления. Число можно представить в виде

$$\pm S \cdot B^{\pm E}$$

Компоненты такого представления можно сохранить в двоичном слове, с, стоящем из трех полей:

- поле знака числа (плюс или минус)
- поле мантиссы S;
- поле порядка E;

Основание B подразумевается неявно и не сохраняется.

В крайнем левом бите слова хранится знак числа (0 – положительное, 1- отрицательное). В следующих 8 битах хранится значение порядка. Для представления порядка используется так называемый смещенный формат. Для получения действительного двоичного кода порядка из значения, сохраняемого в этом поле нужно вычесть фиксированное смещение. Как правило, смещение равно $(2^{k-1}-1)$, где k – разрядность поля порядка. В данном случае k = 8, и в поле порядка можно представить коды в диапазоне от 0 до 255. Если принять значение смещения 127, то действительное значение порядка чисел, представленных в таком формате может находится в интервале от -127 до +128. В данном примере считается, что основание характеристики совпадает с основанием системы счисления и равно 2.

Последнее поле в слове (23 бит) отводится для хранения значения мантиссы S. Теперь хочу обратить ваше внимание на следующий нюанс. Любое число можно представить в форме с плавающей точкой множеством способов. Так, приведенные ниже формы представления эквивалентны, если считать, что мантисса выражена в двоичной системе счисления:

$$0.110x2^5$$
$$110x2^2$$
$$0.0110x2^6$$

Для упрощения алгоритмов выполнения арифметических операций обычно принято нормализовать мантиссу. Нормализованная мантисса числа, отличного от нуля, имеет вид

$$0.1bbb...bx2^{\pm E},$$

где b представляет произвольную двоичную цифру (0 или 1). Это означает, что старший (левый) значащий разряд кода мантиссы всегда равен 1. Но если он всегда равен 1, его нет смысла хранить в составе числа, а можно просто учитывать этот факт при выполнении операций. Таким образом, в 23-битовом поле фактически хранится 24-разрядный код мантиссы, значение которой может был в диапазоне от 0.5_{10} до 1.0.

Ниже приведены примеры чисел в формате с плавающей точкой.

$0.11010001x2^{10100} = 0\ 10010011\ 10100010000000000000000$
$-0.11010001x2^{10100} = 1\ 10010011\ 10100010000000000000000$
$0.11010001x2^{-10100} = 0\ 01101011\ 10100010000000000000000$
$-0.11010001x2^{-10100} = 1\ 01101011\ 10100010000000000000000$

Обратите внимание на следующие особенности:

- знак сохраняется в старшем бите слова;
- первый разряд мантиссы всегда равен 1, и в поле мантиссы не хранится;
- к действительному значению порядка прибавляется смещение 127 и в поле порядка хранится эта сумма;
- основание характеристики равно 2.

Если в слове такой же длины хранить целые числа в дополнительном коде, то диапазон представления будет охватывать 2^{32} чисел от 2^{-31} до $2^{31}-1$ включительно. В формате с плавающей точкой с распределением полей, как показано на рис. 8.10, можно хранить:

отрицательные числа от $-(1-2^{-24})x2^{128}$ до $-0.5x2-1^{27}$,

положительные числа от $0.5x2-1^{27}$ до $(1-2^{-24})x2^{128}$.

Пять областей на числовой оси не включены в диапазон представления:

- отрицательные числа, меньшие, чем $-(1-2^{-24})x2^{128}$, эта область именуется отрицательной областью переполнения (negative overflow);

- отрицательные числа, большие, чем -0.5×2^{-127}, эта область именуется отрицательной областью потери значимости (negative underflow);
- нуль;
- положительные числа, меньшие, чем 0.5×2^{-127}, эта область именуется положительной областью потери значимости (positive underflow);
- положительные числа, большие, чем $(1-2^{24}) \times 2^{128}$, эта область именуется положительной областью переполнения (positive overflow).

Строго говоря, описанное выше представление не позволяет хранить число 0, но на практике код, состоящий из нулей во всех разрядах, считается допустимым и представляет число 0. В области переполнения можно попасть в том случае, если результат арифметической операции имеет абсолютную величину, превышающую ту, которая представляется нормализованным числом с порядком 128, а в область потери значимости – если результат операции имеет очень маленькую абсолютную величину. Потеря значимости как правило не представляет особой проблемы, поскольку в таких случаях с достаточной точностью результат можно считать равным 0.

Обращаю ваше внимание на то, что формат с плавающей точкой не позволяет представить больше отличающихся друг от друга числовых величин. Их по-прежнему 2^{32} (для слова в 32 бита). Интервал между соседними числами – переменный и зависит от абсолютной величины числа.. Фактически результаты округляются с точностью, определяемой имеющейся разрядной сеткой.

В рассматриваемом примере формата под порядок отводится 8 бит, а под мантиссу — 23 бит. Если увеличить длину поля порядка, диапазон представления расширится, но при этом придется сократить поле мантиссы, а значит точность представления снизится, т.е. уменьшится "плотность" размещения представляемых величин на числовой оси. Единственный способ увеличить диапазон, и точность представления — расширить разрядную сетку. В большинстве компьютеров имеются два формата с плавающей точкой: обычный и удвоенной точности. Например, типовой формат обычной точности занимает 32-битовое слово, а формат удвоенной точности занимает 64 бит.

Стандарт IEEE формата с плавающей точкой

Для унификации формата представления чисел с плавающей точкой, что является необходимым условием переносимости программного обеспечения, Институтом инженеров по электротехнике и радиоэлектронике IEEE разработан стандарт 754. В последнее десятилетие практически все процессоры и арифметические сопроцессоры проектируются с учетом требований этого стандарта.

	Обычная точность (32 бит)				Удвоенная точность (64 бит)			
	Знак мантиссы	Смещенный порядок	Мантисса	Значение	Знак мантиссы	Смещенный порядок	Мантисса	Значение
Положительный нуль	0	0	0	0	0	0	0	0
Отрицательный нуль	1	0	0	-0	1	0	0	-0
Плюс бесконечность	0	255 (все единицы)	0	$+\infty$	0	2047 (все единицы)	0	$+\infty$
Минус бесконечность	1	255 (все единицы)	0	$-\infty$	1	2047 (все единицы)	0	$-\infty$
Простое значение типа NaN	0 или 1	255 (все единицы)	$\neq 0$	NaN	0 или 1	2047 (все единицы)	$\neq 0$	NaN
Сигнализирующее значение типа NaN	0 или 1	255 (все единицы)	$\neq 0$	NaN	0 или 1	2047 (все единицы)	$\neq 0$	NaN
Положительное нормализованное число, отличное от 0	0	$0<e<255$	f	$2^{e-127}(1,f)$	0	$0<e<2047$	f	$2^{e-1023}(1,f)$
Отрицательное нормализованное число, отличное от 0	1	$0<e<255$	f	$-2^{e-127}(1,f)$	1	$0<e<2047$	f	$-2^{e-1023}(1,f)$
Положительное ненормализованное число	0	0	$f\neq 0$	$2^{e-126}(0,f)$	0	0	$f\neq 0$	$2^{e-1022}(0,f)$
Отрицательное ненормализованное число	1	0	$f\neq 0$	$-2^{e-126}(0,f)$	1	0	$f\neq 0$	$-2^{e-1022}(0,f)$

Таблица. Представление величин в формате с плавающей точкой в соответствии со стандартом IEEE 754

Сложение и вычитание

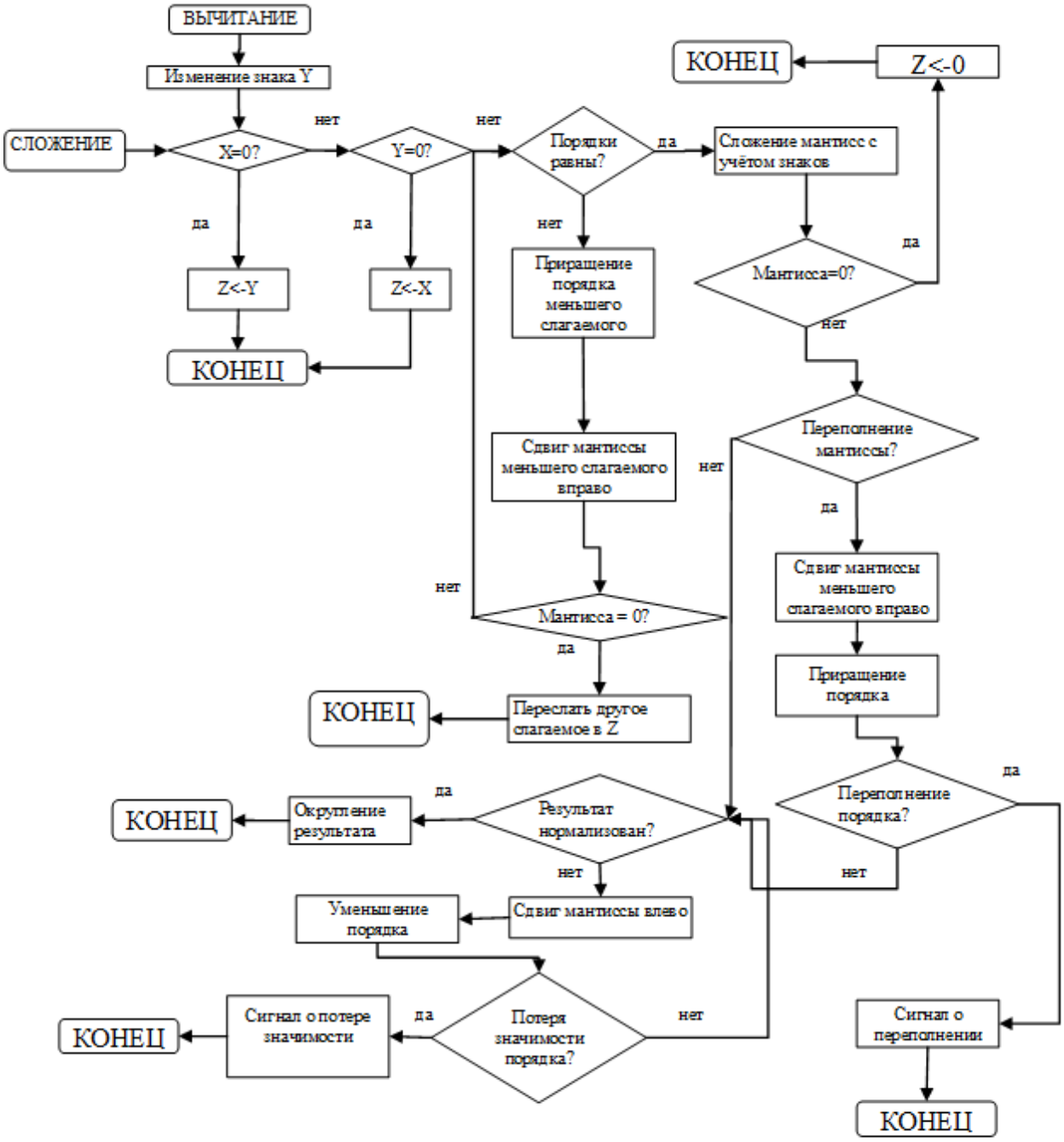

Рис. К.9. Схема алгоритма сложения и вычитания чисел в формате с плавающей точкой (Z <- X ± Y)

Умножение и деление

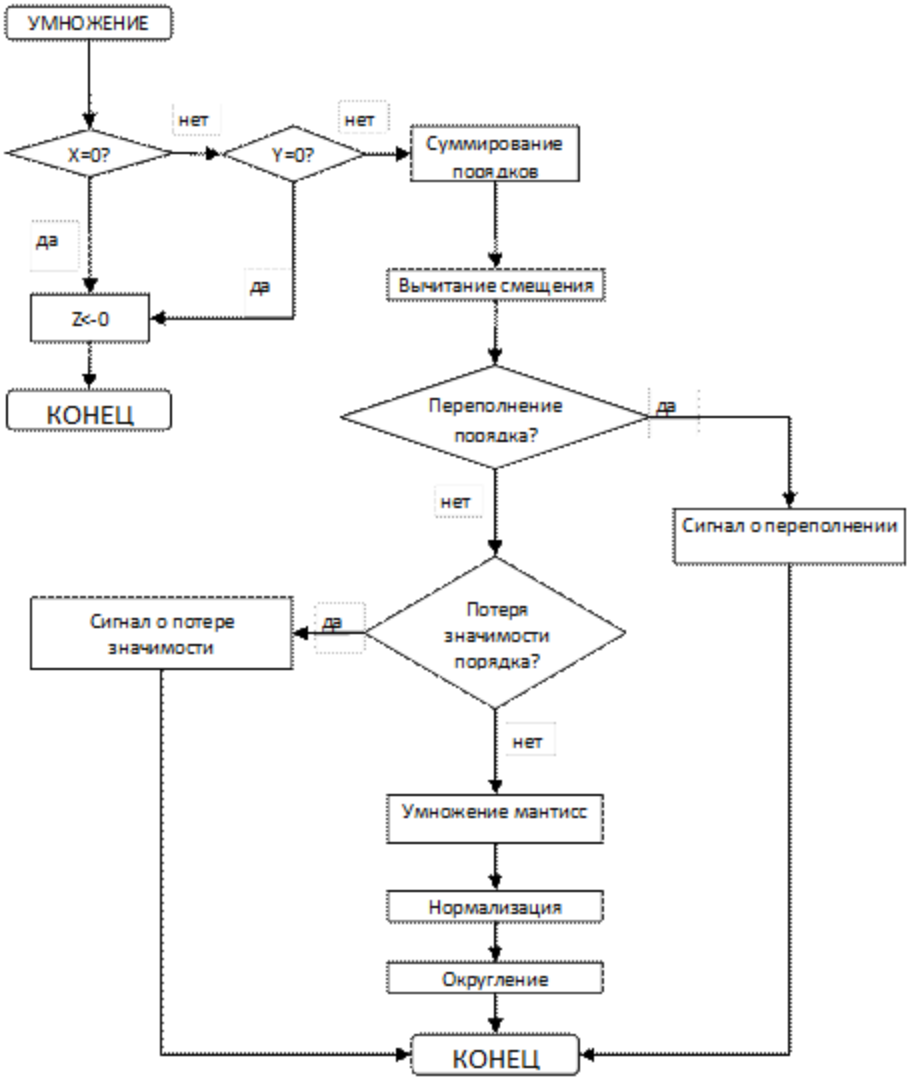

Рис. 10. Схема алгоритма умножения чисел в формате с плавающей точкой (Z<- X x Y)

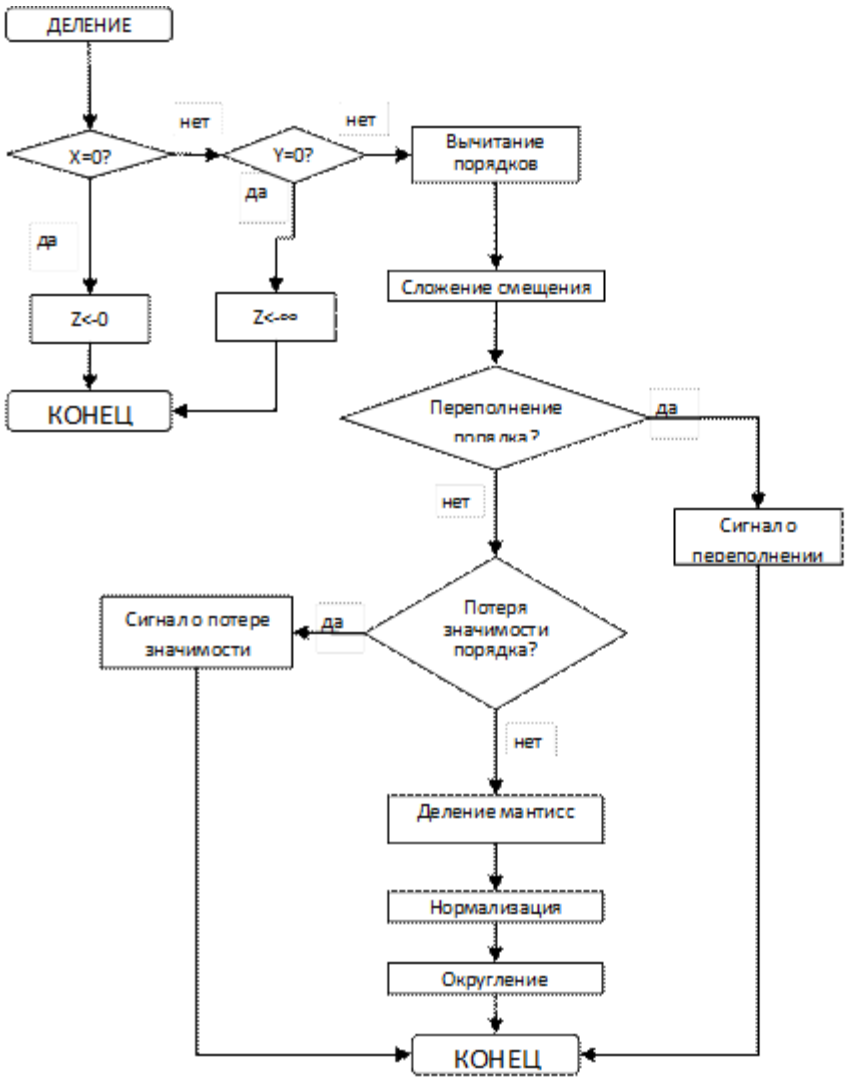

Рис. 11. Схема алгоритма деления чисел в формате с плавающей точкой (Z<- X / Y)

4. Программная реализация

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

Для работы с операциями сложения, вычитания, умножения и деления используется хранение вещественных чисел в 32-битном формате, что предусматривает 4294967296 (2^{32}) различных значения, находящихся приближённо в диапазоне $[-3.4 \cdot 10^{38} ; -1.2 \cdot 10^{-38}] \cup [1.2 \cdot 10^{-38} ; 3.4 \cdot 10^{38}]$.

На рисунках представлен результат работы программы. Выполняется сложение и вычитание двух чисел с плавающей точкой. Демонстрируются операции умножения и деления двух вещественных чисел. Показана обработка особых ситуаций при операциях сложения, умножения и деления.

*Ниже приведен результат программы при **суммировании** 2х чисел.*

Пример 1.

```
--Сложение--
Первое число: 0 (Порядок: 0 )
0 00000000 00000000000000000000000
Второе число: 0 (Порядок: 0 )
0 00000000 00000000000000000000000
Так как одно из чисел равно 0, то результат равен другому числу!
Ответ: 0
```

Пример 2.

```
--Сложение--
Первое число: -13.5 (Порядок: 3 )
1 10000010 10010000000000000000000393216
Второе число: 0 (Порядок: 3 )
0 00000000 00000000000000000000000
Так как одно из чисел равно 0, то результат равен другому числу!
Ответ: -13.5
```

Пример 3.

```
--Сложение--
Первое число: 0 (Порядок: 0 )
0 00000000 00000000000000000000000
Второе число: 10.789 (Порядок: 0 )
0 10000010 01011001001111111011111393217
Так как одно из чисел равно 0, то результат равен другому числу!
Ответ: 10.789
```

Пример 4.

```
--Сложение--
Первое число: -13.5 (Порядок: 3 )
1 10000010 10110000000000000000000917504
Второе число: -10.5 (Порядок: 3 )
1 10000010 01010000000000000000000

После уравнивания порядков:
1 10000010 10110000000000000000000917504
1 10000010 01010000000000000000000
Результирующая мантисса: 10000000000000000000000
Результат: -13.5 + -10.5 = -24
1 10000011 10000000000000000000000
```

Пример 5.

```
--Сложение--
Первое число: 1.005 (Порядок: 0 )
0 00000000 00000001010001111101011393217
Второе число: 101.5 (Порядок: 0 )
0 10000101 10010110000000000000000

После уравнивания порядков:
0 10000101 00000001010001111101011393217
0 10000101 10010110000000000000000
Результирующая мантисса: 11001101000001010001111
Результат: 1.005 + 101.5 = 102.505
0 10000101 10011010000001010001111
```

Пример 6.

```
--Сложение--
Первое число: -13.5 (Порядок: 3 )
1 10000010 10110000000000000000000
Второе число: 2.5 (Порядок: 3 )
0 10000000 01000000000000000000000

После уравнивания порядков:
1 10000010 10110000000000000000000
0 10000010 01000000000000000000000
Вычитаем мантиссы:
11011000000000000000000 -
00101000000000000000000
---------------------------------
10110000000000000000000

Результат: -13.5 + 2.5 = -11
1 10000010 01100000000000000000000
```

Пример 7.

```
--Сложение--
Первое число: 4.5 (Порядок: 2 )
0 10000001 00100000000000000000000
Второе число: 6.5 (Порядок: 2 )
0 10000001 10100000000000000000000

После уравнивания порядков:
0 10000000 00100000000000000000000
0 10000001 10100000000000000000000
Результирующая мантисса: 01100000000000000000000
Результат: 4.5 + 6.5 = 11
0 10000010 01100000000000000000000
```

Рис.1 Успешное суммирование двух чисел

*Ниже приведен результат программы при **вычитании** 2х чисел.*

Пример 1.

```
---Вычитание---
-5.85 - 2.15
Первое число: -5.85 (Порядок: 2 )
1 10000001 01110110011001100011001393217
Второе число: 2.15 (Порядок: 2 )
0 10000000 00010011001100110011010
Вычитание = сложение меняя знак у второго числа
Его новое представление:
1 10000000 00010011001100110011010
--Сложение--
Первое число: -5.85 (Порядок: 2 )
1 10000001 01110110011001100011001393217
Второе число: -2.15 (Порядок: 2 )
1 10000000 00010011001100110011010

После уравнивания порядков:
1 10000001 01110110011001100011001393217
1 10000001 00010011001100110011010
Результирующая мантисса: 11111111111111111111110
Результат: -5.85 - 2.15 = -8
1 10000001 11111111111111111111110
```

Пример 2.

```
---Вычитание---
-0.85 - 1.25
Первое число: -0.85 (Порядок: -1 )
1 01111110 10110011001100110011001101393217
Второе число: 1.29688 (Порядок: -1 )
0 00000000 01000000000000000000000393216
Вычитание = сложение меняя знак у второго числа
Его новое представление:
1 00000000 01000000000000000000000393216
--Сложение--
Первое число: -0.85 (Порядок: -1 )
1 01111110 10110011001100110011001101393217
Второе число: -1.29688 (Порядок: -1 )
1 00000000 01000000000000000000000393216

После уравнивания порядков:
1 00000000 10110011001100110011001101393217
1 00000000 01000000000000000000000393216
Результирующая мантисса: 00001100110011001100110
Результат: -0.85 - 1.25 = -2.1
1 10000000 00001100110011001100110
```

Пример3.

```
---Вычитание---
-101.999 - -99.999
Первое число: -101.999 (Порядок: 6 )
1 10000101 10010111111111101111100
Второе число: -99.999 (Порядок: 6 )
1 10000101 10001111111111101111100
Вычитание = сложение меняя знак у второго
Его новое представление:
0 10000101 10001111111111101111100
--Сложение--
Первое число: -101.999 (Порядок: 6 )
1 10000101 10010111111111101111100
Второе число: 99.999 (Порядок: 6 )
0 10000101 10001111111111101111100

После уравнивания порядков:
1 10000101 10010111111111101111100
0 10000101 10001111111111101111100
Вычитаем мантиссы:
11001011111111110111110053982823411000111
---------------------------------
00000100000000000000000

Результат: -101.999 - -99.999 = -2
1 10000000 00000000000000000000000
```

Пример 4.

```
---Вычитание---
8.85 - 2.35
Первое число: 8.85 (Порядок: 3 )
0 10000010 00011011001100110011101393217
Второе число: 2.35 (Порядок: 3 )
0 10000000 00101100110011001100110
Вычитание = сложение меняя знак у второго числа
Его новое представление:
1 10000000 00101100110011001100110
--Сложение--
Первое число: 8.85 (Порядок: 3 )
0 10000010 00011011001100110011101393217
Второе число: -2.35 (Порядок: 3 )
1 10000000 00101100110011001100110

После уравнивания порядков:
0 10000010 00011011001100110011101393217
1 10000010 00101100110011001100110
Вычитаем мантиссы:
10001101100110011001101393217 -
00100101100110011001001
---------------------------------
01101000000000000000010

Результат: 8.85 - 2.35 = 6.5
0 10000001 10100000000000000000100
```

Пример 5.

```
---Вычитание---
5.5 - 3.5
Первое число: 5.5 (Порядок: 2 )
0 10000001 01100000000000000000000
Второе число: 3.5 (Порядок: 2 )
0 10000000 11000000000000000000000
Вычитание = сложение меняя знак у второго числа
Его новое представление:
1 10000000 11000000000000000000000
--Сложение--
Первое число: 5.5 (Порядок: 2 )
0 10000001 01100000000000000000000
Второе число: -3.5 (Порядок: 2 )
1 10000000 11000000000000000000000

После уравнивания порядков:
0 10000001 01100000000000000000000
1 10000001 11000000000000000000000
Вычитаем мантиссы:
10110000000000000000000 -
01110000000000000000000
---------------------------------
01000000000000000000000

Результат: 5.5 - 3.5 = 2
0 10000000 00000000000000000000000
```

Рис.3 Успешное вычитание двух чисел

Ниже приведен результат программы при умножении 2х чисел.

```
---Умножение---
1.5 * 2.5
Первое число: 1.5 (Порядок: 0 )
0 00000000 10000000000000000000000393216
Второе число: 2.5 (Порядок: 0 )
0 10000000 01000000000000000000000
Переменожение мантис:
11000000000000000000000393216 *
101
------------------------
11110000000000000000000000
Порядок первого: 0 (00000000 )
Порядок второго: 1 (10000000 )
Результирующий порядок (после нормализации) : 1 (10000000 )
Ответ: 1.5 * 2.5 = 3.75
0 10000000 11100000000000000000000
3.75
```

```
---Умножение---
8.5 * 2
Первое число: 8.5 (Порядок: 3 )
0 10000010 00010000000000000000000393216
Второе число: 2 (Порядок: 3 )
0 10000000 00000000000000000000000
Переменожение мантис:
10001000000000000000000393216 *
1
------------------------
10001000000000000000000
Порядок первого: 3 (10000010 )
Порядок второго: 1 (10000000 )
Результирующий порядок (после нормализации) : 4 (10000011 )
Ответ: 8.5 * 2 = 17
0 10000011 00010000000000000000000
17
```

```
---Умножение---
8 * 7
Первое число: 8 (Порядок: 3 )
0 10000010 00000000000000000000000
Второе число: 7 (Порядок: 3 )
0 10000001 11000000000000000000000
Переменожение мантис:
1 *
111
-----------------------
111
Порядок первого: 3 (10000010 )
Порядок второго: 2 (10000001 )
Результирующий порядок (после нормализации) : 5 (10000100 )
Ответ: 8 * 7 = 56
0 10000100 11000000000000000000000
56
```

```
---Умножение---
0.05 * 10.5
Первое число: 0.05 (Порядок: -5 )
0 01111010 10011001100110011103932160000
Второе число: 10.875 (Порядок: -5 )
0 10000010 01010000000000000000000393216
Переменожение мантис:
11001100110011001103932l6 *
10101000000000000000000393216
-----------------------
11001100110011001110000110011001100111101000
Порядок первого: -5 (01111010 )
Порядок второго: 3 (10000010 )
Результирующий порядок (после нормализации) : -2 (01111101 )
Ответ: 0.05 * 10.5 = 0.400001
0 01111101 10011001100110011100001
0.400001
```

```
---Умножение---
2.1 * 10.5
Первое число: 2.1 (Порядок: 1 )
0 10000000 00001100110011001100110
Второе число: 10.5 (Порядок: 1 )
0 10000010 01010000000000000000000
Переменожение мантис:
10000110011001100110011 *
10101
-----------------------
10110000011001100110011111
Порядок первого: 1 (10000000 )
Порядок второго: 3 (10000010 )
Результирующий порядок (после нормализации) : 4 (10000011 )
Ответ: 2.1 * 10.5 = 22.05
0 10000011 01100000110011001100111
22.05
```

Рис.4 Успешное перемножение двух чисел

Успешное деление 2х чисел:

```
---Деление---
a = 0 01111111 00000000000000000000000 = 1
b = 1 10000000 10000000000000000000000 = -3
a / b =
Порядок a = 01111111
Порядок b = 10000000
Порядок a/b = 00000000 01111110
Мантисса a = 8388608
Мантисса b = 12582912
Мантисса a/b = 5592405
Знак a/b = -1
Порядок a/b = -1
Мантисса a/b = 00000000 0 10101010101010101010101
Нормализация ответа: <-
Знак a/b = -1
Порядок a/b = -2
Мантисса a/b = 00000000 1 01010101010101010101010
  = 1 01111101 01010101010101010101010 = -0.333333

---Деление---
a = 0 00000000 00000000000000000000000 = 0
b = 0 10000010 10100000000000000000000 = 13
a / b =
a = 0.0, b != 0.0 -> a / b = 0
  = 0 00000000 00000000000000000000000 = 0
```

Особоые ситуации. которые могут вознакнуть в ходе операций:

```
---Умножение---
a = 0 00000001 00000000000000000000000 = 1.17549e-038
b = 0 00000001 00000000000000000000000 = 1.17549e-038
a * b =
Порядок a = 00000001
Порядок b = 00000001
Порядок a*b = 11111111 10000011
Ошибка: Потеря значимости порядка

---Умножение---
a = 0 00000001 00000000000000000000000 = 1.17549e-038
b = 0 01111110 11111111011111001110111 = 0.999
a * b =
Порядок a = 00000001
Порядок b = 01111110
Порядок a*b = 00000000 00000000
Ошибка: Потеря значимости порядка
```

```
---Деление---
a = 0 11111110 11111111111111111111111 = 3.40282e+038
b = 0 00000001 00000000000000000000000 = 1.17549e-038
a / b =
Порядок a = 11111110
Порядок b = 00000001
Порядок a/b = 00000001 01111100
Ошибка: Переполнение порядка
```

```
---Деление---
a = 0 10000100 01010000000000000000000 = 42
b = 0 00000000 00000000000000000000000 = 0
a / b =
b = 0.0 ->
Ошибка: Деление на ноль
```

```
---Деление---
a = 0 00000001 00000000000000000000000 = 1.17549e-038
b = 0 01111111 00000000100000011000101 = 1.001
a / b =
Порядок a = 00000001
Порядок b = 01111111
Порядок a/b = 00000000 00000001
Мантисса a = 8388608
Мантисса b = 8396997
Мантисса a/b = 8380227
Знак a/b = +1
Порядок a/b = -126
Мантисса a/b = 00000000 0 11111111101111101000011
Нормализация ответа: <-
Ошибка: Потеря значимости порядка
```

```
---Деление---
a = 0 00000001 00000000000000000000000 = 1.17549e-038
b = 0 11111110 11111111111111111111111 = 3.40282e+038
a / b =
Порядок a = 00000001
Порядок b = 11111110
Порядок a/b = 11111111 10000010
Ошибка: Потеря значимости порядка
```

```
---Умножение---
a = 0 00000001 00000000000000000000000 = 1.17549e-038
b = 0 00000001 00000000000000000000000 = 1.17549e-038
a * b =
Порядок a = 00000001
Порядок b = 00000001
Порядок a*b = 11111111 10000011
Ошибка: Потеря значимости порядка

---Умножение---
a = 0 00000001 00000000000000000000000 = 1.17549e-038
b = 0 01111110 11111111011111001110111 = 0.999
a * b =
Порядок a = 00000001
Порядок b = 01111110
Порядок a*b = 00000000 00000000
Ошибка: Потеря значимости порядка
```

```
---Умножение---
a = 0 11111110 11111111111111111111111 = 3.40282e+038
b = 0 11111110 11111111111111111111111 = 3.40282e+038
a * b =
Порядок a = 11111110
Порядок b = 11111110
Порядок a*b = 00000001 01111101
Ошибка: Переполнение порядка

---Умножение---
a = 0 11111110 11111111111111111111111 = 3.40282e+038
b = 0 01111111 00000000010000011000101 = 1.001
a * b =
Порядок a = 11111110
Порядок b = 01111111
Порядок a*b = 00000000 11111110
Мантисса a = 16777215
Мантисса b = 8396997
Мантисса a*b = 16793992
Знак a*b = +1
Порядок a*b = 127
Мантисса a*b = 00000001 0 00000001000000110001000
Нормализация ответа: ->
Ошибка: Переполнение порядка
```

```
---Суммирование---
a = 0 00000010 00000000000000000000000 = 2.35099e-038
b = 1 00000001 10000000000000000000000 = -1.76324e-038
a + b =
a = 1 00000001 10000000000000000000000 = -1.76324e-038
b = 0 00000010 00000000000000000000000 = 2.35099e-038
a + b =
Порядки чисел не равны.
a = - 00000010 0.01100000000000000000000
b = + 00000010 0.10000000000000000000000
Мантисса a = -6291456
Мантисса b = 8388608
Мантисса a+b = 2097152
Знак a+b = +1
Порядок a+b = -125
Мантисса a+b = 00000000 0 01000000000000000000000
Нормализация ответа: <- <-
Ошибка: Потеря значимости порядка
```

```
---Суммирование---
a = 0 11111110 11111111111111111111111 = 3.40282e+038
b = 0 11111110 11111111111111111111111 = 3.40282e+038
a + b =
Порядки чисел равны.
a = + 11111110 0.11111111111111111111111
b = + 11111110 0.11111111111111111111111
Мантисса a = 16777215
Мантисса b = 16777215
Мантисса a+b = 33554430
Знак a+b = +1
Порядок a+b = 127
Мантисса a+b = 00000001 1 11111111111111111111110
Нормализация ответа: ->
Ошибка: Переполнение порядка
```

5. Выводы

Стандарт IEEE 754 описывает унифицированный формат хранения чисел с плавающей точкой в 32-битном формате: 1 бит выделяется для знака, 8 бит – для порядка, 23 бита – для мантиссы.

Для нахождения суммы двух вещественных чисел необходимо уравнять порядки чисел, сложить их мантиссы с учётом знака, и нормализовать полученный результат.

При выполнении суммирования чисел необходимо проверить равенство каждого из них нулю, обнуление младшего числа при выравнивании порядков, обнуление мантиссы при суммировании, переполнение мантиссы и порядка в процессе вычислений.

Для нахождения произведения двух вещественных чисел необходимо суммировать порядки чисел, перемножить их мантиссы с учётом знака, и нормализовать полученный результат.

При выполнении умножения чисел необходимо проверить равенство каждого из них нулю, переполнение порядка и потерю значимости в процессе вычислений.

Для нахождения частного двух вещественных чисел необходимо вычесть порядки чисел, поделить их мантиссы с учётом знака, и нормализовать полученный результат.

При выполнении деления чисел необходимо проверить равенство каждого из них нулю, переполнение порядка и потерю значимости в процессе вычислений.

В ходе выполнения лабораторной работы была написана программа, реализующая основные арифметические операции над числами с плавающей точкой. Также осуществлена проверка на возникновение особых 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

Файл Float.h

```
#ifndef LAB456_FLOAT_H
#define LAB456_FLOAT_H
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>




#define show_add true
#define show_sub true
#define show_mul true
#define show_div false


class Float {
   float float_num;
   const static int m_size = 23;
   const static int e_size = 8;
   int zero_exp = 127;
   const static int N = m_size + e_size + 1;
   std::vector<int> number;
   std::vector<int> mantissa;
   std::vector<int> exp;

   int expInt;
   int sign;

   static std::vector<int> truncMantissa(std::vector<int> m);
   std::vector<int> addBinaryMul(const std::vector<int>& v1,std::vector<int>& v2);
   static std::vector<int> addBinary(const std::vector<int>& v1, const std::vector<int>& v2);
   static std::vector<int> subBinary(const std::vector<int>& v1, const std::vector<int>& v2);

   static std::pair<std::vector<int>, int> addBinary_s(const std::vector<int>& v1, const std::vector<int>&
v2);

   static std::vector<int> rMove(const std::vector<int>& v);
   static std::vector<int> lMove(const std::vector<int>& v);

   static std::vector<int> buildBinary(int num, int size);
   static std::vector<int> entireToBin(int entire);
   static std::vector<int> buildExp(int exponent, int exp_size);
   static std::vector<int> fractionToBin(float fraction, int size=16);
   static int maxInt(int exp_size);

   static int binEntireToInt(const std::vector<int>& v);
   static int entireFromMantissa(const std::vector<int>& mantissa, int exp);
   static float fractionFromMantissa(const std::vector<int>& mantissa, int exp);
   static std::vector<int> trimV(const std::vector<int>& v_1, const std::vector<int>& v_2, int size);
```

```
    static std::vector<int> binFromMantissa(std::vector<int>& m);
    static std::vector<int> mantissaFromBin(std::vector<int>& bin);
    static std::vector<int> mantissaFromRandBin(std::vector<int>& bin);


    friend std::ostream& operator<<(std::ostream& out, const std::vector<int>& v);
    friend std::ostream& operator<<(std::ostream& out, const Float& floating);


//    Float operator>>(int num) const;

    static std::vector<int> make2c(std::vector<int>& bin);




public:
    Float();
    explicit Float(float num);
    Float(const Float& num);
    Float operator+(Float& floating_2);
    Float operator-(Float& floating_2);
    Float operator*(Float& floating_2);
    Float operator/(Float& floating_2);
    Float& operator=(const Float& floating_2);

    explicit operator float() const;

};



class Register{

    static const int N = 48;
    int number;
    std::vector<int> binary;

    struct Flags{
        int OF = 0;
    } flags;


    void printBits() const;

    static std::vector<int> makeBinary(int num);
    static void reverseArrBits(std::vector<int> &arr);
    static void twos_complement(std::vector<int>& bin);
    static int toInt(const Register& reg);


public:

    Register();
    explicit Register(int num);
```

```
    explicit Register(const std::vector<int> &bitsArr);
    Register(const Register& reg);

    void setBinary(const std::vector<int> & binArr);
    void setNumber(const int num);
    void setRegister(const Register& reg);

    static Register reverseBits(const Register &reg);
    void reverseBits();
    static int getBigNum(const Register& reg_1, const Register& reg_2);

    Register& operator=(const Register& reg);

private:
    int& operator[](const int index);
    const int& operator[](const int index) const;

public:
    void setIndexedVal(int index, int value);
    Register& operator++();
    Register operator++(int);

    Register operator+(const Register &reg_2);
    Register operator+(int& num);
    friend Register operator+(int& num, Register& reg_2);

    Register operator-(const Register& reg_2);
    Register operator-(int& num);
    friend Register operator-(int& num, Register& reg_2);

    std::vector<int> operator*(Register& reg_2);
    Register operator/(Register& reg_2);

    friend bool operator==(const Register& reg_1, const Register& reg_2);
    friend bool operator==(const Register& reg_1, const int& num);

    Register operator<<(const Register& reg2) const;
    Register operator<<(int num) const;
    Register operator>>(const Register& reg_2) const;
    Register operator>>(int num) const;

    friend std::ostream& operator<< (std::ostream &out, const Register& reg);
    friend std::istream& operator>> (std::istream &in, Register& reg);

    explicit operator int() const;
};



#endif //LAB456_FLOAT_H
```

Файл Float.cpp

```
#include "Float.h"



Float::Float() {
   number = std::vector<int>(N, 0);
   mantissa = std::vector<int>(m_size, 0);
   exp = std::vector<int>(e_size, 0);
}


Float::Float(float num) : Float() {

   float_num = num;
   int entire = abs((int)num);
   float fraction = abs(num) - (float)entire;
   std::vector<int> entireBin = entireToBin(entire);
   std::vector<int> fractionBin = fractionToBin(fraction, 23);

   int offset = 0;
   if (entire > 0)
      offset = (int)entireBin.size() - 1;
   else if (entire == 0 && fraction != 0){
      for(int i = 0; fractionBin[i] != 1; ++i){
         offset--;
      }
      offset--;
   }


   int i = 0;
   if(num >= 0){
      sign = 0;
      number[i] = 0;
      ++i;
   }
   else{
      sign = 1;
      number[i] = 1;
      ++i;
   }

   zero_exp = maxInt(e_size);
   expInt = offset;
   int exponent = maxInt(e_size) + offset;
   exp = buildExp(exponent, e_size);
   while(i < e_size+1){
      number[i] = exp[i-1];
      ++i;
   }
```

```
//    Don't keep track first one in mantissa and skip it

   // combine two vectors and to mantisa

   std::vector<int> trimmed = trimV(entireBin, fractionBin, m_size);
   int k = 0;
   while(trimmed[k] != 1 && k < trimmed.size()){
      ++k;
   }
   ++k;
   for(int m = 0; i < N; ++k){

      number[i] = trimmed[k];
      mantissa[m] = trimmed[k];
      ++i;
      ++m;

   }

//    for(int k = 1; i < N && k < entireBin.size(); ++k){
//
//       number[i] = entireBin[k];
//       mantissa[m] = entireBin[k];
//       ++i;
//       ++m;
//    }
//
//    for(int k = 0; i < N && k < fractionBin.size(); ++k){
//       if (entire == 0 && fraction == 0){
//          break;
//       }
//       if (entire == 0 && k == 0){
//          while(fractionBin[k] != 1 && k < fractionBin.size()){
//             ++k;
//          }
//          continue;
//       }
//       number[i] = fractionBin[k];
//       mantissa[m] = fractionBin[k];
//       ++i;
//       ++m;
//    }
}


Float::Float(const Float& num) {
   this->expInt = num.expInt;
   this->number = num.number;
   this->exp = num.exp;
   this->mantissa = num.mantissa;
   this->sign = num.sign;
   this->float_num = num.float_num;
}
```

```
Float& Float::operator=(const Float &floating_2) {
   this->expInt = floating_2.expInt;
   this->number = floating_2.number;
   this->exp = floating_2.exp;
   this->mantissa = floating_2.mantissa;
   this->sign = floating_2.sign;
   this->float_num = floating_2.float_num;
   return *this;
}


std::vector<int> Float::buildBinary(int num, int size){
   std::vector<int> bin(size, 0);
   for(int i = 0; num > 0; ++i){
      bin[size-1-i] = num % 2;
      num /= 2;
   }
   return bin;
}


int Float::maxInt(int exp_size){
   return (int)pow(2, e_size - 1) - 1;
}


std::vector<int> Float::buildExp(int exponent, int exp_size){
   if (exponent == 127){
      return std::vector<int> (exp_size, 0);
   }
   std::vector<int> binExp(exp_size, 0);
   for(int i = 0; i <= exp_size; ++i){
      binExp[exp_size-i-1] = exponent % 2;
      exponent /= 2;
   }
//    std::reverse(binExp.begin(), binExp.end());
   return binExp;
}


std::vector<int> Float::entireToBin(int entire){
   std::vector<int> binArr(0,0);
   while(entire > 0){
      binArr.push_back(entire % 2);
      entire /= 2;
   }
   std::reverse(binArr.begin(), binArr.end());
   return binArr;
}


std::vector<int> Float::fractionToBin(float fraction, int size){
   std::vector<int> binArr(0, 0);
   for(int i = 0; i < size; ++i){
      fraction *= 2;
      if (fraction >= 1){
```

```
            binArr.push_back(1);
            fraction -= 1;
        }
        else
            binArr.push_back(0);
    }
    return binArr;
}


std::ostream& operator<<(std::ostream& out, const std::vector<int>& v){
    for(auto &i : v){
        out << i;
    }
    return out;
}


std::ostream& operator<<(std::ostream& out, const Float& floating){
    out << floating.sign << ' ';
    out << floating.exp << ' ';
    out << floating.mantissa << '\n';
    return out;
}


int Float::binEntireToInt(const std::vector<int> &v) {
    int number = 0;
    for(int i = 0; i < v.size(); ++i){
      number += v[v.size()-1-i] * (int)pow(2, i);
    }
    return number;
}


int Float::entireFromMantissa(const std::vector<int>& mantissa, int exp){
    int entire = 0;

    entire += pow(2, exp);
    for(int i = 0; i < exp; ++i){
        entire += mantissa[i] * (int)pow(2, exp-i-1);
    }
    return entire;
}


float Float::fractionFromMantissa(const std::vector<int>& mantissa, int exp){

    float fraction = 0;
    if (exp >= 0) {
        for (int i = exp, k = -1; i < m_size; ++i, --k) {
            fraction += mantissa[i] * (float) pow(2, k);
        }
    }
    else{
        fraction += pow(2, exp);
```

```
    for (int i = 0, k = exp-1; i < m_size; ++i, --k) {
        fraction += mantissa[i] * (float) pow(2, k);
    }
  }
  return fraction;
}


Float::operator float() const {
  float num = 0;
  num += (float)entireFromMantissa(mantissa, expInt);
  num += fractionFromMantissa(mantissa, expInt);
  if (sign == 1)
    num *= -1;
  return num;
}


std::vector<int> Float::addBinary(const std::vector<int>& v1, const std::vector<int>& v2) {
  std::vector<int> result(v1.size(), 0);
  int r = 0;
  for(int i = v1.size() - 1; i >= 0; --i){
    if (v1[i] + v2[i] + r == 0){
      result[i] = 0;
      r = 0;
    }
    if (v1[i] + v2[i] + r == 1){
      result[i] = 1;
      r = 0;
    }
    if (v1[i] + v2[i] + r == 2){
      result[i] = 0;
      r = 1;
    }
    if (v1[i] + v2[i] + r == 3){
      result[i] = 1;
      r = 1;
    }
  }

  return result;
}


std::vector<int> Float::addBinaryMul(const std::vector<int>& v1, std::vector<int>& v2) {
  int v2_size = v2.size();
  std::vector<int> newv2(v1.size(), 0);
  int k = 0;
  for(int i = newv2.size()-1; i >= 0; --i){
    if (k < v2_size){
      newv2[i] = v2[v2.size()-k-1];
      k++;
    }
  }
  v2 = newv2;
  std::vector<int> result(v1.size(), 0);
```

```
    int r = 0;
    for(int i = v1.size() - 1; i >= 0; --i){
        if (v1[i] + v2[i] + r == 0){
            result[i] = 0;
            r = 0;
        }
        if (v1[i] + v2[i] + r == 1){
            result[i] = 1;
            r = 0;
        }
        if (v1[i] + v2[i] + r == 2){
            result[i] = 0;
            r = 1;
        }
        if (v1[i] + v2[i] + r == 3){
            result[i] = 1;
            r = 1;
        }
    }
    std::vector<int> extra(v1.size() + 1, 0);
    extra[0] = 1;
    if (r == 1){
        for(int i = 1; i < extra.size(); ++i){
            extra[i] = result[i-1];
        }
        return extra;
    }
    return result;
}


std::pair<std::vector<int>, int> Float::addBinary_s(const std::vector<int>& v1, const std::vector<int>&
v2) {
    std::vector<int> result(v1.size(), 0);
    int r = 0;
    for(int i = v1.size() - 1; i >= 0; --i){
        if (v1[i] + v2[i] + r == 0){
            result[i] = 0;
            r = 0;
        }
        else if (v1[i] + v2[i] + r == 1){
            result[i] = 1;
            r = 0;
        }
        else if (v1[i] + v2[i] + r == 2){
            result[i] = 0;
            r = 1;
        }
        else if (v1[i] + v2[i] + r == 3){
            result[i] = 1;
            r = 1;
        }
    }

    return {result, r};
}
```

```
std::vector<int> Float::rMove(const std::vector<int>& v){
   std::vector<int> result(v.size(), 0);
   for(int i = (int)v.size()-1; i >= 1; --i){
      result[i] = v[i-1];
   }
   result[0] = 0;
   return result;
}


std::vector<int> Float::lMove(const std::vector<int>& v){
   std::vector<int> result(v.size(), 0);
   for(int i = 0; i < v.size() - 1; ++i){
      result[i] = v[i+1];
   }
   result[v.size()-1] = 0;
   return result;
}



//Float Float::operator>>(int num) const{
//    Float result(0);
//    while(num > 0) {
//        result.mantissa = rMove(mantissa);
//        result.exp = addBinary(exp, buildBinary(1, e_size));
//        result.expInt += 1;
//        --num;
//    }
//    return result;
//}



//Float Float::operator<<(int num) const{
//    Float result(0);
//    while(num > 0) {
//        result.mantissa = lMove(mantissa);
//        result.exp = addBinary(exp, buildBinary(1, e_size));
//        result.expInt += 1;
//        --num;
//    }
//    return result;
//}


std::vector<int> Float::subBinary(const std::vector<int> &v1, const std::vector<int> &v2) {
   std::vector<int> res(v1.size(), 0);
   int r = 0;
   for(int i = (int)v1.size() - 1; i >= 0; --i) {
      if (v1[i] - v2[i] - r == 1) {
         res[i] = 1;
         r = 0;
      }
      else if (v1[i] - v2[i] - r == 0) {
```

```
            res[i] = 0;
            r = 0;
        }
        else if (v1[i] - v2[i] - r == -1) {
            res[i] = 1;
            r = 1;
        }
        else if (v1[i] - v2[i] - r == -2) {
            res[i] = 1;
            r = 1;
        }
    }
    return res;
}


Float Float::operator+(Float &floating_2) {

//    Float fl2_copy(floating_2);
    if (show_add){
        std::cout << "--Сложение--\n";
    }

    if(show_add){
        std::cout << "Первое число: " << (float)float_num << " (Порядок: " << this->expInt  << " )" << '\n';
        std::cout << *this;
        std::cout << "Второе число: " << (float)floating_2 << " (Порядок: " << this->expInt  << " )" << '\n';
        std::cout << floating_2;
    }

    if(this->float_num == 0){
        if (show_add){
            std::cout << "Так как одно из чисел равно 0, то результат равен другому числу!\n";
            std::cout << "Ответ: " << floating_2.float_num << '\n';
        }
        return floating_2;
    }
    else if(floating_2.float_num == 0){
        if (show_add){
            std::cout << "Так как одно из чисел равно 0, то результат равен другому числу!\n";
            std::cout << "Ответ: " << this->float_num << '\n';


        }
        return *this;
    }

    int diff = this->expInt - floating_2.expInt;
    std::vector<int> newMantissa;
    int exponent = this->expInt > floating_2.expInt ? this->expInt : floating_2.expInt;

    std::vector<int> mant;
    std::vector<int> constMant;
    if (diff > 0){
        mant = binFromMantissa(floating_2.mantissa);
        constMant = binFromMantissa(this->mantissa);
```

```
    }
    else{
        mant = binFromMantissa(this->mantissa);
        constMant = binFromMantissa(floating_2.mantissa);
    }

    while(diff != 0) {
        if (diff > 0) {
            floating_2.expInt += 1;
            --diff;
        }
        else if (diff < 0) {
            this->expInt += 1;
            ++diff;
        }
        mant = rMove(mant);
    }

    this->exp = buildExp(exponent + zero_exp, e_size);
    floating_2.exp = buildExp(exponent + zero_exp, e_size);


    if (show_add){
        std::cout << "\nПосле уравнивания порядков:\n";
        std::cout << *this;
        std::cout << floating_2;
    }

    Float result(0);

    if (this->sign == floating_2.sign) {
        auto resMantissa = addBinary_s(mant, constMant);


        if (show_add) {
            std::cout << "Результирующая мантисса: " << resMantissa.first;
        }

        if (resMantissa.second == 1) {
            exponent += 1;
//          resMantissa.first = lMove(resMantissa.first);
//          while (resMantissa.first[0] != 1) {
//              exponent += 1;
//              resMantissa.first = lMove(resMantissa.first);
//          }
//          resMantissa.first = lMove(resMantissa.first);
            result.mantissa = resMantissa.first;

        }
        else{
            result.mantissa = mantissaFromBin(resMantissa.first);
        }


        result.expInt = exponent;
        result.exp = buildExp(exponent + result.zero_exp, e_size);
```

```
        result.float_num = (float) result;
        result.sign = this->sign;
    }
    else {
        std::vector<int> subRes;
//      if (floating_2.float_num > this->float_num){
            bool bigger_mant = false;
            for(int i = 0; i < mant.size(); ++i){
                if (mant[i] > constMant[i]){
                    bigger_mant = true;
                    break;
                }
                else if (mant[i] < constMant[i]){
                    bigger_mant = false;
                    break;
                }
            }

            if (bigger_mant){
                subRes = subBinary(mant, constMant);
                if(show_add){
                    std::cout << "Вычитаем мантиссы: "  << '\n';
                    std::cout << mant << ' - \n';
                    std::cout << constMant << '\n';
                    std::cout << "---------------------------------\n";
                    std::cout << subRes << '\n';
                }
            }
            else {
                subRes = subBinary(constMant, mant);
                if (show_add) {
                    std::cout << "Вычитаем мантиссы: " << '\n';
                    std::cout << constMant << " - \n";
                    std::cout << mant << '\n';
                    std::cout << "---------------------------------\n";
                    std::cout << subRes << '\n';
                }
            }

//      }

        while (subRes[0] != 1) {
            exponent -= 1;
            subRes = lMove(subRes);
        }
//      subRes = lMove(subRes);


        result.mantissa = mantissaFromBin(subRes);
        result.expInt = exponent;
        result.exp = buildExp(exponent + result.zero_exp, e_size);
        result.float_num = (float) result;
        if (this->float_num - floating_2.float_num > 0){
            result.sign = 0;
        }
        else
```

```
        result.sign = 1;

    }

    if (show_add){
        std::cout << "\nРезультат: " << this->float_num << " - " << floating_2.float_num << " = "
            << (float)result << '\n';
        std::cout << result;
    }

    return result;
}


std::vector<int> Float::make2c(std::vector<int> &bin) {
    std::vector<int> twosBin(bin.size(), 0);
    for(int i = 0; i < bin.size(); ++i) {
        bin[i] == 1 ? twosBin[i] = 0 : twosBin[i] = 1;
    }

    int i = (int)bin.size()-1;
    while(i >= 0){
        if (twosBin[i] == 1) {
            twosBin[i] = 0;
        }
        else {
            twosBin[i] = 1;
            break;
        }
        --i;
    }
    return twosBin;
}


Float Float::operator-(Float &floating_2) {

    if (show_sub){
        std::cout << "---Вычитание---\n";
        std::cout << this->float_num << " - " << floating_2.float_num << '\n';

        std::cout << "Первое число: " << (float)float_num << " (Порядок: " << this->expInt  << " )" << '\n';
        std::cout << *this;
        std::cout << "Второе число: " << (float)floating_2 << " (Порядок: " << this->expInt  << " )" << '\n';
        std::cout << floating_2;

        std::cout << "Вычитание = сложение меняя знак у второго числа\n";
    }
    Float result(0);
    if (floating_2.sign == 1){
        floating_2.sign = 0;
    }
    else{
        floating_2.sign = 1;
    }
```

```
    if (show_sub) {
        std::cout << "Его новое представление: \n";
        std::cout << floating_2;
    }

    result = *this + floating_2;

//    if (show_sub){
//        std::cout << "--Вычитание--\n";
//    }
//
//    if(show_sub){
//        std::cout << "Первое число: " << this->float_num << " (Порядок: " << this->expInt  << " )" << '\n';
//        std::cout << *this;
//        std::cout << "Второе число: " << floating_2.float_num << " (Порядок: " << this->expInt  << " )" <<
'\n';
//        std::cout << floating_2;
//    }
//
//    if(this->float_num == 0){
//        return floating_2;
//    }
//    else if(floating_2.float_num == 0){
//        return *this;
//    }
//
//    int diff = this->expInt - floating_2.expInt;
//    std::vector<int> newMantissa;
//    int exponent = this->expInt > floating_2.expInt ? this->expInt : floating_2.expInt;
//
//    std::vector<int> mant;
//    std::vector<int> constMant;
//    if (diff > 0){
//        mant = binFromMantissa(floating_2.mantissa);
//        constMant = binFromMantissa(this->mantissa);
//    }
//    else{
//        mant = binFromMantissa(this->mantissa);
//        constMant = binFromMantissa(floating_2.mantissa);
//    }
//
//    while(diff != 0) {
//        if (diff > 0) {
//            floating_2.expInt += 1;
//            --diff;
//        }
//        else if (diff < 0) {
//            this->expInt += 1;
//            ++diff;
//        }
//        mant = rMove(mant);
//    }
//
//    this->exp = buildExp(exponent + zero_exp, e_size);
//    floating_2.exp = buildExp(exponent + zero_exp, e_size);
//
```

```
//
//    if (show_sub){
//        std::cout << "\nПосле уравнивания порядков:\n";
//        std::cout << *this;
//        std::cout << floating_2;
//    }
//
//    std::vector<int> inverse = make2c(mant);
//    auto resMantissa = addBinary_s(mant, constMant);
//
//
//    if (show_sub){
//        std::cout << "Результирующая мантисса: " << resMantissa.first;
//    }
//
////    if (resMantissa.second == 1){
////        exponent +=1;
////        resMantissa.first = lMove(resMantissa.first);
////        while(resMantissa.first[0] != 1){
////            exponent += 1;
////            resMantissa.first = lMove(resMantissa.first);
////        }
////        resMantissa.first = lMove(resMantissa.first);
////    }
//
//    Float result(0);
//    result.mantissa = mantissaFromBin(resMantissa.first);
//    result.expInt = exponent;
//    result.exp = buildExp(exponent + result.zero_exp, e_size);
//    result.float_num = (float)result;
//
//    if (show_sub){
//        std::cout << "\nРезультат: " << this->float_num << " + " << floating_2.float_num << " = "
//                  << result.float_num << '\n';
//        std::cout << result;
//    }
//
//    return result;

}


std::vector<int> Float::truncMantissa(std::vector<int> m) {
    std::vector<int> truncated(0, 0);
    int cnt_zer = 0;
    for(int i = m.size() - 1; i >= 0; --i){
        if (m[i] == 0) {
            ++cnt_zer;
        }else{
            break;
        }
    }

    for(int i = 0; i < m.size() - cnt_zer; ++i){
        truncated.push_back(m[i]);
    }
```

```
    return truncated;
}


Float Float::operator*(Float &floating_2) {

    if (show_mul) {
        std::cout << "---Умножение---\n";

        std::cout << this->float_num << " * " << floating_2.float_num << '\n';

        std::cout << "Первое число: " << (float) float_num << " (Порядок: " << this->expInt << " )" << '\n';
        std::cout << *this;
        std::cout << "Второе число: " << (float) floating_2 << " (Порядок: " << this->expInt << " )" << '\n';
        std::cout << floating_2;
    }

    auto mant_1 = binFromMantissa(this->mantissa);
    auto mant_2 = binFromMantissa(floating_2.mantissa);

    std::vector<int> mul(0, 0);


    mant_1 = truncMantissa(mant_1);
    auto truncated_2 = truncMantissa(mant_2);
    std::vector<int> sum(mant_1.size(), 0);

    for(int i = 0; i < truncated_2.size(); ++i){
        std::vector<int> num(mant_1.size() + i);
        for(int k = 0; k < num.size(); ++k){
            if (k < mant_1.size()) {
                num[k] = mant_1[k];
            }
            else{
                num[k] = 0;
            }
        }
        if (truncated_2[i] == 0){
            continue;
        }
        else{
            sum = addBinaryMul(num, sum);
        }
    }

    if(show_mul){
        std::cout << "Переменожение мантис:\n";
        std::cout << mant_1 << " * " <<'\n';
        std::cout << truncated_2 << '\n';
        std::cout << "-----------------------\n";
        std::cout << sum << '\n';
    }


    std::vector<int> newMant(m_size, 0);
    for(int i = 1; i < sum.size(); i++){
```

```
        newMant[i-1] = sum[i];
    }

    Float result(0);
    result.mantissa = newMant;


    int exponent = floating_2.expInt + this->expInt;
    result.exp = buildExp(zero_exp + exponent, e_size);
    if (this->sign != floating_2.sign){
        result.sign = 1;
    }
    else{
        result.sign = 0;
    }
    result.expInt = exponent;

    int i = 1;
    for(int k = 0; k < result.exp.size(); ++k){
        result.number[i] = result.exp[k];
    }
    for(int k = 0; k < result.mantissa.size(); ++k){
        result.number[i] = result.mantissa[k];
    }


    if (show_mul){
        std::cout << "Порядок первого: " << this->expInt << " (" << this->exp <<  " )" << '\n';
        std::cout << "Порядок второго: " << floating_2.expInt << " (" << floating_2.exp <<  " )" << '\n';
        std::cout << "Результирующий порядок (после нормализации) : " << result.expInt
        << " (" << result.exp <<  " )" << '\n';

        std::cout << "Ответ: " << this->float_num << " * " << floating_2.float_num << " = " << (float)result <<
'\n';
        std::cout << result;
    }

    return result;
}


//Float Float::operator/(Float& floating_2){
//    int resSign = 0;
//    if (this->sign + floating_2.sign == 1){
//        resSign = 1;
//    }
//    int exp = this->expInt - floating_2.expInt;
//
//    std::vector<int> mant_1 = binFromMantissa(this->mantissa);
//    std::vector<int> mant_2 = binFromMantissa(floating_2.mantissa);
//    Register m_1(mant_1);
//    Register m_2(mant_2);
```

```
//}



std::vector<int> Float::trimV(const std::vector<int> &v_1, const std::vector<int> &v_2, int size) {
   std::vector<int> trimmed(size, 0);
   int i = 0;
   int k = 0;
   while(i < size) {
      if (i < v_1.size()) {
         trimmed[i] = v_1[i];
      } else {
         trimmed[i] = v_2[k];
         ++k;
      }
      ++i;
   }
   return trimmed;
}


std::vector<int> Float::binFromMantissa(std::vector<int> &m) {
   std::vector<int> bin(m.size()+1, 0);
   for(int i = 0; i < bin.size(); ++i){
      if (i == 0){
         bin[i] = 1;
         continue;
      }
      bin[i] = m[i-1];
   }
   return bin;
}


std::vector<int> Float::mantissaFromRandBin(std::vector<int>& bin){
   std::vector<int> m(m_size, 0);
   for(int i = 1; i < m.size(); ++i){
      m[i-1] = bin[i];
   }
   return m;
}


std::vector<int> Float::mantissaFromBin(std::vector<int> &bin) {
   std::vector<int> m(bin.size()-1, 0);
   for(int i = 1; i < bin.size(); ++i){
      m[i-1] = bin[i];
   }
   return m;
}



void Register::setBinary(const std::vector<int> &binArr) {
```

```
    binary = binArr;
    number = (int)(*this);
}


void Register::setNumber(const int num){
    number = num;
    binary = makeBinary(abs(num));
    if (num < 0) {
        twos_complement(binary);
    }
}


void Register::setRegister(const Register& reg) {
    number = reg.number;
    binary = reg.binary;
}



void Register::reverseBits(){
    reverseArrBits(binary);
    ++(*this);
}


Register Register::reverseBits(const Register& reg){
    Register reversed(reg.binary);
    reversed.reverseBits();
    return reversed;
}


void Register::twos_complement(std::vector<int>& bin){
    reverseArrBits(bin);
    int i = N-1;
    while(i >= 0){
        if (bin[i] == 1) {
            bin[i] = 0;
        }
        else {
            bin[i] = 1;
            break;
        }
        --i;
    }
}


void Register::printBits() const{
    for(int i = 0; i < N; ++i) {
        std::cout << binary[i];
    }
}
```

```
std::vector<int> Register::makeBinary(int num) {

   std::vector<int> binArr(N, 0);
   int i = 0;
   while(num > 0) {
      binArr[N-1-i] = num % 2;
      num /= 2;
      ++i;
   }
   return binArr;
}


void Register::reverseArrBits(std::vector<int> &arr) {
   for(int i = 0; i < N; ++i) {
      arr[i] == 1 ? arr[i] = 0 : arr[i] = 1;
   }
}


int Register::toInt(const Register& reg){
   int intNum = 0;
   intNum -= reg.binary[0] * pow(2, N - 1);
   for(int i = N - 1; i > 0; --i){
      if (reg.binary[i] == 0){
         continue;
      }
      intNum += pow(2, N-i-1);
   }
   return intNum;
}


Register::Register() {
   binary = std::vector<int>(N, 0);
   number = 0;
}


Register::Register(int num) : Register() {
   setNumber(num);
}


Register::Register(const std::vector<int> &bitsArr) : Register() {
   setBinary(bitsArr);
}


Register::Register(const Register& reg){
   setNumber(reg.number);
}


Register& Register::operator=(const Register& reg){
```

```
    setRegister(reg);
    return *this;
}


int& Register::operator[](const int index){

    return this->binary[index];
}


const int& Register::operator[](const int index) const {
    return this->binary[index];
}


void Register::setIndexedVal(int index, int value){
    (*this)[index] = value;
    this->setBinary(this->binary);
}


Register& Register::operator++(){
    int i = N-1;
    while(i >= 0){
        if (binary[i] == 0){
            binary[i] = 1;
            break;
        }
        binary[i] = 0;
        --i;
    }
    return *this;
}


Register Register::operator++(int){
    Register temp(binary);
    ++(*this);
    return temp;
}


Register Register::operator+(const Register &reg_2){
    int r{};
    int r_prev{};
    Register result;
    if (show_add){
        std::cout << "---Сложение---\n";
        std::cout << "Первое число: " << int(*this) << " (" << this->binary << ")"<< '\n';
        std::cout << "Второе число: " << int(reg_2) << " (" << reg_2.binary << ")"<< '\n';


        std::cout << std::setw(5);
        std::cout << "|num_1|" << " |num_2|" << " |CF|" << " |res|\n";
    }
```

```
    for(int i = N-1; i >= 0; --i) {
        r_prev = r;
        if ((this->binary[i] + reg_2[i] + r) == 1) {
            r = 0;
            result[i] = 1;
        }
        else if ((this->binary[i] + reg_2[i] + r) == 2) {
            r = 1;
            result[i] = 0;
        }
        else if(this->binary[i] + reg_2[i] + r == 3) {
            r = 1;
            result[i] = 1;
        }
        else{
            r = 0;
            result[i] = this->binary[i] + reg_2[i];
        }
        if (show_add){
            std::cout << std::setw(4) << binary[i];
            std::cout << std::setw(8) << reg_2[i];
            std::cout << std::setw(6) << r_prev;
            std::cout << std::setw(6) << result[i] << '\n';
        }

    }
    if ((this->binary[0] == reg_2[0]) && (this->binary[0] != result[0])){
        result.flags.OF = 1;
        std::cout << "Переполнение\n";
    }

    if(show_add){
        std::cout << (int)*this << " + " << (int)reg_2 << " = " << (int)result << '\n';
        std::cout << "Результат: " << (int)result << " (" << result.binary << ")\n";
    }
    result.setBinary(result.binary);
    return result;
}


Register Register::operator+(int& num){
    Register reg_2(num);
    Register result = *this + reg_2;
    return result;
}


Register operator+(int& num, Register& reg_2){
    Register result = reg_2 + num;
    return result;
}


Register Register::operator-(const Register& reg_2){
    if (show_sub){
        std::cout << "---Вычитание---\n";
```

```
        std::cout << (int)*this << " - " << (int)reg_2 << '\n';
        std::cout << "Первое число: " << int(*this) << " (" << this->binary << ")"<< '\n';
        std::cout << "Второе число: " << int(reg_2) << " (" << reg_2.binary << ")"<< '\n';
    }

    Register reg_2_temp(reg_2);
    if (reg_2 == -32678){
        std::cout << "Невозможно выполнить отрицание для " << int(reg_2) << '\n';
        return Register(0);
    }

    reg_2_temp.reverseBits();
    if (show_sub){
        std::cout << "Инвертирование 2 числа:\n";
        std::cout << "~ " << reg_2.binary << " = " << reg_2_temp.binary << " (" << (int)reg_2_temp << ")\n";
    }
    Register result = *this + reg_2_temp;
    return result;
}


Register Register::operator-(int& num){
    Register reg_2(num);
    Register result = *this - reg_2;
    return result;
}


Register operator-(int& num, Register& reg_2){
    Register reg_1(num);
    Register result = reg_1 - reg_2;
    return result;
}


int Register::getBigNum(const Register& reg_1, const Register& reg_2){
    int num = 0;
    int j = 0;
    for(int i = 0; i < N; ++i){
        num += reg_2[N-i-1] * (int)pow(2, j);
        ++j;
    }
    for(int i = 0; i < N; ++i){
        num += reg_1[N-i-1] * (int)pow(2, j);
        ++j;
    }
    return num;
}


std::vector<int> Register::operator*(Register& reg_2){


    Register A;
    int Q1 = 0;
    int A1 = 0;
```

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

      if (Q1 == 0 && reg_2[N-1] == 1){
         A = A - *this;
      }
      else if (Q1 == 1 && reg_2[N-1] == 0){
         A = A + *this;
      }


//       shifting
      A1 = A.binary[N-1];
      A = A >> 1;
      Q1 = reg_2.binary[N-1];
      reg_2 = reg_2 >> 1;
      reg_2.setIndexedVal(0, A1);
//       reg_2[0] = A1;


   }
   return reg_2.binary;
//    return getBigNum(A, reg_2);
}


Register Register::operator/(Register& reg_2) {

   if (reg_2 == 0){
      std::cout << "Ошибка при делении! Не предусмотрено деление на 0!";
      return Register(0);
   }

   if (*this == -32768 && reg_2 == -1) {
      std::cout << "Переполнение при делении!";
      return Register(0);
   }


   Register A;
   if (this->binary[0] == 1){
      A.setNumber(-1);
   }
   Register M(reg_2);
   Register reg_1(*this);




   int Qn{};
   for (int i = N - 1; i >= 0; --i) {
      Qn = reg_1[0];
      A = A << 1;
      reg_1 = reg_1 << 1;
```

```
        A[N - 1] = Qn;



        int A_sign = A[0];

        if (M[0] == A[0]) {
            A = A - M;

        } else {
            A = A + M;

        }

        if (A_sign == A[0] || (A == 0 && reg_1 == 0)) {
            reg_1.setIndexedVal(N - 1, 1);

        } else {
            if (M[0] == A_sign) {
                A = A + M;
            }
            else{
                A = A - M;
            }
            reg_1.setIndexedVal(N - 1, 0);

        }


    }

    if ((*this)[0] != M[0]) {
        Register saved_reg_1(reg_1);
        reg_1.reverseBits();


    }


    return reg_1;
}


bool operator==(const Register& reg_1, const Register& reg_2){
    return reg_1.binary == reg_2.binary;
}

bool operator==(const Register& reg_1, const int& num){
    return (int)reg_1 == num;
}

Register Register::operator<<(const Register& reg2) const{
    Register shifted;
    shifted.setNumber((int)(*this) << reg2.number);
    return shifted;
}
```

```
Register Register::operator<<(int num) const{
   Register shifted;
   shifted.setNumber((int)(*this) << num);
   return shifted;
}


Register Register::operator>>(const Register& reg_2) const{
   Register shifted;
   shifted.setNumber((int)(*this) >> reg_2.number);
   return shifted;
}


Register Register::operator>>(int num) const{
   Register shifted;
   shifted.setNumber(this->number >> num);
   return shifted;
}


Register::operator int() const {
   int intNum = 0;
   intNum -= this->binary[0] * (int)pow(2, N - 1);
   for(int i = N - 1; i > 0; --i){
      if (this->binary[i] == 0){
         continue;
      }
      intNum += pow(2, N-i-1);
   }
   return intNum;
}


std::ostream& operator<< (std::ostream &out, const Register& reg) {
   for(int i = 0; i < Register::N; ++i){
      out << reg[i] << ' ';
      if (i == 7){
         out  << ' ';
      }
   }
   out << '\n';
   return out;
}




std::istream& operator>> (std::istream &in, Register& reg){
   int num;
   in >> num;
   Register newRegister(num);
   reg = newRegister;
```

```
   return in;
}


} static void Division(string a, string b) {
cout << "---Деление---" << endl;
Print("a", a);
Print("b", b);
cout << "a / b =" << endl;
if (IsZero(b)) {
cout << "b = 0.0 ->" << endl;
throw string("Деление на ноль");
}
if (IsZero(a)) {
cout << "a = 0.0, b != 0.0 -> a / b = 0" << endl;
Print(" ", 0);
return;
}
bool SignBitA = SignBit(a), SignBitB = SignBit(b);
int SignA = Sign(a), SignB = Sign(b);
int ExpA = Exp(a), ExpB = Exp(b);
int MantA = Mant(a), MantB = Mant(b);
cout << "Порядок a = " << a.substr(1, 8) << endl;
cout << "Порядок b = " << b.substr(1, 8) << endl;
int ds = SignA * SignB;
int de = ExpA - ExpB;
long long dm = ((long long) MantA << 23) / MantB;
cout << "Порядок a/b = " << bitset<16>(de + 127).to_string().insert(8, " ") << endl;
if (de >= 128) {
throw string("Переполнение порядка");
}
if (de <= -127) {
throw string("Потеря значимости порядка");
}
```

```
cout << "Мантисса a = " << MantA << endl;
cout << "Мантисса b = " << MantB << endl;
cout << "Мантисса a/b = " << dm << endl;
if (dm == 0) {
throw string("Что-то пошло не так");
}
```

out << "Знак a/b = " << (