Алгоритм умножения float point чисел

**Оглавление**

[Базовые понятия 1](#_Toc1321153905)

[Введение 2](#_Toc1201259857)

[Умножение 2](#_Toc1032915010)

[Перемножение мантисс 3](#_Toc387407195)

[Суммирование экспонент 3](#_Toc1689205972)

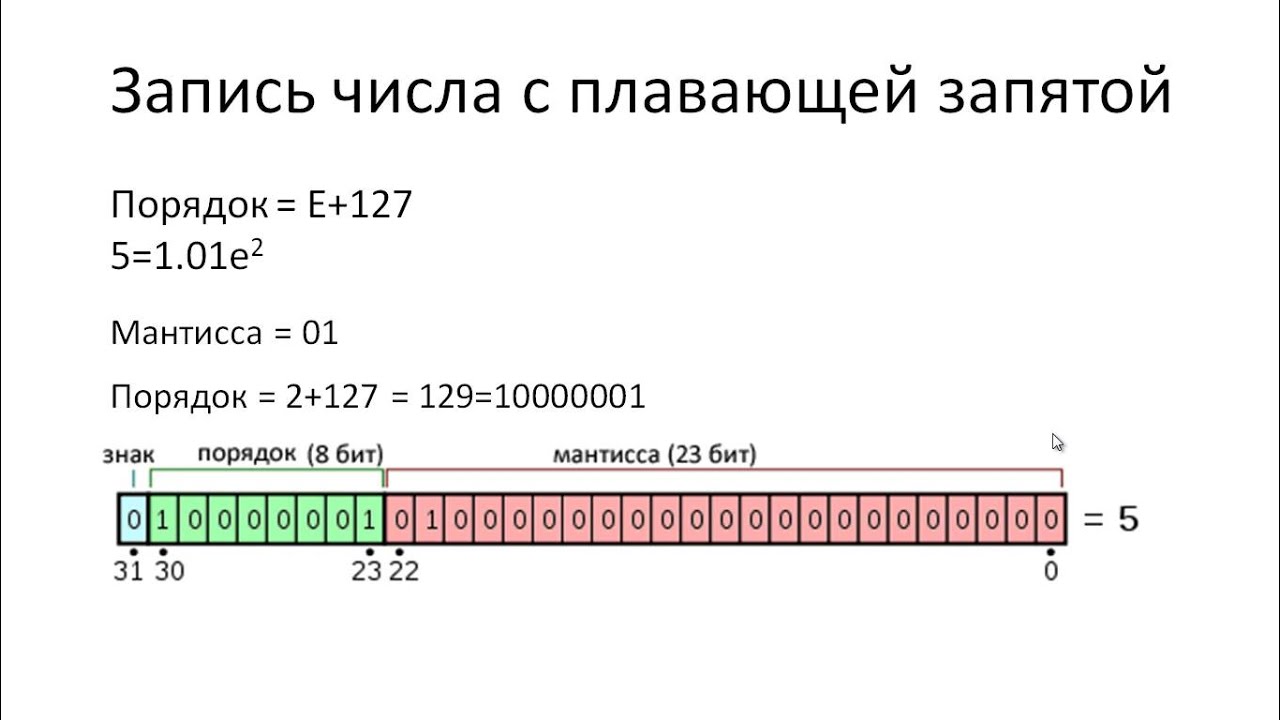
[Определение знака 4](#_Toc2076118339)

[Запись в результат 4](#_Toc1785373249)

[Краткий алгоритм 4](#_Toc368375346)

# Базовые понятия

**Число с плавающей запятой** — экспоненциальная форма представления вещественных (действительных) чисел, в которой число хранится в виде мантиссы и порядка (показателя степени).



**IEEE 754** — широко используемый стандарт IEEE, описывающий формат представления чисел с плавающей точкой. Используется в программных (компиляторы разных языков программирования) и аппаратных (CPU и FPU) реализациях арифметических действий.

# Введение

Cтандарт **IEEE 754**, описывает как должны быть записаны числа с плавающей запятой и как выполнять арифметические операции над ними. Его можно найти в открытом доступе.

Оригинал документа: [IEEE Std 754™-2008 (Revision of IEEE Std 754-1985), IEEE Standard for Floating-Point Arithmetic (andrewt0301.github.io)](https://andrewt0301.github.io/hse-acos-course/part1ca/07_FP/ieee-754-2008.pdf)

Для удобного представления float point чисел, можно использовать онлайн конвертер: [IEEE-754 Floating Point Converter (h-schmidt.net)](https://www.h-schmidt.net/FloatConverter/IEEE754.html)

# Умножение

Для умножения двух чисел с плавающей запятой нужно: перемножить их мантиссы; суммировать экспоненты; выполнить операцию **xor** над знаками. В зависимости от размера мантиссы и экспоненты числа алгоритм не меняется.

# Краткий алгоритм

1. Переводим числа в формат **IEEE 754**
2. Разделяем их на: знак, экспоненту и мантиссу
3. Перемножаем мантиссы
   1. Добавляем мнимые единицы
   2. Выполняем операцию умножения
   3. Проверяем на *переполнение*
      1. Если переполнение есть – сдвигаем мантиссу на 1 бит вправо, чтобы слева от точки не было лишнего разряда
      2. Запоминаем, что нужно будет увеличить порядок (добавить единицу к результирующей экспоненте)
   4. Проверяем на округление
      1. Округляем, если выполняется условие: (i(24) and i(23)
      2. Округляем, если выполняется условие: (i(23) and (i(22-0))
   5. Удаляем лишние нули и мнимую единицу
   6. Сохраняем результат
4. Суммируем экспоненты
   1. Суммируем экспоненты
   2. Если было *переполнение* в пункте 3.3 добавляем единицу
   3. Отнимаем максимальную экспоненту (для fp32 это - 127)
   4. Сохраняем результат
5. Находим знак
   1. Выполняем операцию xor между знаками
   2. Сохраняем результат
6. Соединяем сохраненные результаты
7. Готово

# Пример

Возьмем два числа: 5,5 \* 5,5 = 30,25.

Переводим в формат IEEE 754, fp32:

5,5 = 101,1 = 1,011e2;

Мантисса = 011;

Экспонента = 2;

Результат в формате **IEEE 754:**



## Перемножение мантисс

Это самый трудоемкий и сложный для понимания этап умножения. Мантиссы операндов: 011 и 011 соответственно. Согласно стандарту IEEE 754, добавим к каждой мантиссе мнимую единицу, а также расширим на требуемое количество разрядов:

1.011000000000000000000000

Перемножив мантиссы, получим результат, разрядность которого в два раза больше исходной:

1.11100100000000000000000 000000000000000000000000

Проверим результат на *переполнение*. При умножении 5,5 \* 5,5 его нет. Однако, если умножить мантиссы чисел 12 и 11,5, то получим результат:

1.10000000000000000000000 \* 1.01110000000000000000000 = 10.001010000000000000000000 0000000000000000000000.

В **этом** примере *переполнение* есть, поэтому требуется нормализовать результирующую мантиссу, сдвинув ее на 1 бит вправо:

( 10.001010... >> 1 ) = 1.000101...

Если была *нормализация*, в будущем, не забудем *инкрементировать* экспоненту.

При вычислении 5,5 \* 5,5, получим результат:

1.1110010000000000000000000 0000000000000000000000,

Выполним округление по условиям, где i – мантисса:

1. (i(24) and i(23))

2. (i(23) and (i(22-0))

Если одно из условий **истинно**, то делаем сдвиг вправо на размер мантиссы и *инкрементируем* результат. Если условие **ложно** — делаем только сдвиг.

При вычислении 5,5 \* 5,5 условие **ложно**:

1) Если оба бита равны 1, то округляем:

1.1110010000000000000000**0 0**00000000000000000000000

2) Если самый левый, выделенный бит равен 1 и хотя бы один бит правее от него равен 1, то округляем:

1.11100100000000000000000 **000000000000000000000000**

Пример **выполнения** условий на иных числах:

1) Если оба бита равны 1, то округляем:

1.01001111111001110110110**1** **1**0000000000000000000000

2) Если самый левый, выделенный бит равен 1 и хотя бы один бит правее от него равен 1, то округляем:

1.010011111110011101101100 **1 0101110010101100000010**

При вычислении 5,5 \* 5,5 условие **ложно**, поэтому делаем сдвиг:

1.1110010000000000000000000 0000000000000000000000 >> 23 = 1.1110010000000000000000000

Удаляем мнимую единицу и получаем результат:

1110010000000000000000000



## Суммирование экспонент

Экспонента числа 5,5 = 10000001, в десятичном формате = 129. Суммируем: 129 + 129 = 258;

Отнимаем максимальный размер экспоненты, для fp32 - это = 127, выполняем: 258 - 127 = 131.

Формула максимального размера экспоненты

(Если была *нормализация* мантиссы, *инкрементируем* результат)

Переведем результат в двоичный формат: 131 = 10000011.

## Определение знака

Выполняем операцию xor между нашими знаками: 0 xor 0 = 0.

## Запись в результат

Соединяем результаты: 0 10000011 11100100000000000000000.



Проверка полученного результата:

1.111001e4 = 11110,01 = 30,25 = (5,5 \* 5,5)