IEEE 754 - стандарт двоичной арифметики с плавающей точкой

Автор: Яшкардин Владимир
www.softelectro.ru
Дата: 2009
Редакция: 04.06.2012
info@softelectro.ru

§1. Название стандарта.

Данный стандарт разработан ассоциацией IEEE (Institute of Electrical and Electronics Engineers) и используется для представления действительных чисел (чисел с плавающей точкой) в двоичном коде.
Наиболее используемый стандарт для вычислений с плавающей точкой, используется многими микропроцессорами и логическими устройствами, а также программными средствами.

Полное название стандарта в ассоциации IEEE:

- ***IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985)***
- ***IEEE стандарт для двоичной арифметики с плавающей точкой (ANSI/IEEE Std 754-1985)***

Название стандарта в международной электротехнической комиссии IEC:

- ***IEC 60559:1989, Binary floating-point arithmetic for microprocessor systems***
- ***IEC 60559:1989 двоичная арифметика с плавающей точкой для микропроцессорных систем***
- ***(IEC 559:1989 - старое обозначение стандарта)***

В 2008 года ассоциация IEEE выпустила стандарт IEEE 754-2008, который включил в себя стандарт IEEE 754-1985.

§2.Краткое описание стандарта.

Оригинальный выпуск стандарта:
IEEE Standard for Binary Floating-Point Arithmetic

The Institute of Electrical and Electronics Engineers, Inc
345 East 47th Street, New York, NY 10017, USA

Стандарт содержит 23 страницы текста в 7 секциях и одном приложении:

1.Scope(Область применения)
- 1.1 Implementation Objectives (Описывает цели стандарта)
- 1.2 Inclusions (Описывает что включено в стандарт)
- 1.3 Exclusions (Описывает что не определяет стандарт)

2.Definitions (Вводимы определения)
3.Formats (Форматы чисел)
- 3.1 Sets of Values (Наборы переменных для представления формата)
- 3.2 Basic Formats (Базовые форматы)
- 3.3 Extended Formats (Расширенные форматы)
- 3.4 Combinations of Formats (Комбинирование форматов)

4.Rounding (Округления)
- 4.1 Round to Nearest (Округление к ближайшему)
- 4.2 Directed Roundings (Прямое округление)
- 4.3 Rounding Precision (Точность округления)

5.Operations (Операции)
- 5.1 Arithmetic (Арифметика)
- 5.2 Square Root (Квадратный корень)
- 5.3 Floating-Point Format Conversions (Конверсия форматов с плавающей точкой)
- 5.4 Conversion Between Floating-Point and Integer Formats (Конверсия между форматами с плавающими точками и форматами целых чисел.)
- 5.5 Round Floating-Point Number to Integer Value (округление чисел с плавающей точкой в целые числа)
- 5.6 Binary <-> Decimal Conversion (Конверсия бинарного в десятичное)
- 5.7 Comparison (Сравнение)

6.Infinity, NaNs, and Signed Zero (Бесконечность, не числа, и знаковый ноль)
- 6.1 Infinity Arithmetic (Арифметические действия с бесконечностями)
- 6.2 Operations with NaNs (Операции с не числами)
- 6.3 The Sign Bit (Операции с знаковым бит)

7.Exceptions (Исключения)
- 7.1 Invalid Operation (Недопустимые операции)
- 7.2 Division by Zero (Деление на ноль)
- 7.3 Overflow (Переполнение)
- 7.4 Underflow (Нехватка разряда)

7.5 Inexact (Неточность)
8.Traps (Обнаружение недопустимых операций)
8.1 Trap Handler (Исполнитель обнаружения недопустимых операций)
8.2 Precedence (Первоочередность)
A.Recommended Functions and Predicates (Рекомендованные функции и утверждения)

К сожалению, организация IEEE превратилась из международной общественной инженерной организации (которой она была изначально) в торговую организацию.
Этой организации принадлежит авторское право на публикацию стандарта IEEE754-1985.
Поэтому если вы захотите ознакомиться, с оригиналом стандарта, вам придется купить его примерно за 80$.
Но, Российского законодательство разрешает мне в учебных целях комментировать данный стандарт.
Поэтому дальше я буду давать вольное изложение стандарта и выражать своё мнение о нём в учебных целях.

Стандарт IEEE 754-1985 определяет:
- как представлять нормализованные положительные и отрицательные числа с плавающей точкой
- как представлять денормализованные положительные и отрицательные числа с плавающей точкой
- как представлять нулевые числа
- как представлять специальную величину бесконечность (Infinity)
- как представлять специальную величину "Не число" (NaN или NaNs)
- четыре режима округления

IEEE 754-1985 определяет четыре формата представления чисел с плавающей запятой:
- с одинарной точностью (single-precision) 32 бита
- с двойной точностью (double-precision) 64 бита
- с одинарной расширенной точностью (single-extended precision) >=43 бит (редко используемый)
- с двойной расширенной точностью (double-extended precision) >= 79 бит (обычно используют 80 бит)

§3. Основные понятия в представлении чисел с плавающей точкой.

3.1 Представление числа в нормализованном экспоненциальном виде.

Возьмем, к примеру, десятичное число 155,625
Представим это число в нормализованном экспоненциальном виде : $1{,}55625\cdot10^{+2}=1{,}55625\cdot exp_{10}{}^{+2}$
Число $1{,}55625\cdot exp_{10}{}^{+2}$ состоит из двух частей: мантиссы M=1.55625 и экспоненты $exp_{10}=+2$
Если мантисса находится в диапазоне 1<=M<10, то число считается нормализованным.
Экспонента представлена основанием системы исчисления (в данном случае 10) и порядком (в данном случае +2).
Порядок экспоненты может иметь отрицательное значение, например число $0{,}0155625=1{,}55625\cdot exp_{10}{}^{-2}$.

3.2 Представление числа в денормализованном экспоненциальном виде.

Возьмем, к примеру, десятичное число 155,625
Представим это число в денормализованном экспоненциальном виде : $0{,}155625\cdot10^{+3}=0{,}155625\cdot exp_{10}{}^{+3}$
Число $0{,}155625\cdot exp_{10}{}^{+3}$ состоит из двух частей: мантиссы M=0,155625 и экспоненты $exp_{10}=+3$
Если мантисса находится в диапазоне 0,1<=M<1, то число считается денормализованным.
Экспонента представлена основанием системы исчисления (в данном случае 10) и порядком (в данном случае +3).
Порядок экспоненты может иметь отрицательное значение, например число $0{,}0155625=0{,}155625\cdot exp_{10}{}^{-1}$.

3.3 Преобразование десятичного числа в двоичное число с плавающей точкой.

Наша задача сводится к представлению десятичного числа с плавающей точкой, в двоичное число с плавающей точкой в экспоненциальном нормализованном виде.
Для этого разложим заданное число по двоичным разрядам:

$155{,}625 = 1\cdot2^{7}+0\cdot2^{6}+0\cdot2^{5}+1\cdot2^{4}+1\cdot2^{3}+0\cdot2^{2}+1\cdot2^{1}+1\cdot2^{0}+1\cdot2^{-1}+0\cdot2^{-2}+1\cdot2^{-3}$
$155{,}625 = 128 + 0 + 0 + 16 + 8 + 0 + 2 + 1 + 0{,}5 + 0 + 0{,}125$
$155{,}625_{10} = 10011011{,}101_{2}$ - число в десятичной и в двоичной системе с плавающей точкой

Приведем полученное число к нормализованному виду в десятичной и двоичной системе:
$1{,}55625\cdot exp_{10}{}^{+2} = 1{,}0011011101\cdot exp_{2}{}^{+111}$

В результате мы получили основные составляющие экспоненциального нормализованного двоичного числа:
Мантиссу M=1,0011011101
Экспоненту exp_2= +111

§4. Описание преобразования чисел по стандарту IEEE 754.

4.1 Преобразование двоичного нормализованного числа в 32 битный формат IEEE 754

Основное применение в технике и программирование получили форматы 32 и 64 бита.
Например, в VB используют типы данных single (32 бита) и double (64 бита).
В Си аналогично используют float (32 бита) и double (64 бит)
Рассмотрим преобразование двоичного числа 10011011,101 в формат single-precision (32 бита) стандарта IEEE 754.
Остальные форматы представления чисел в IEEE 754 являются увеличенной копией single-precision.

Чтобы представить число в формате single-precision IEEE 754 необходимо привести его к двоичному нормализованному виду.
В §3 мы проделали это преобразование над числом 155,625.
Теперь рассмотрим, как двоичное нормализованное число преобразуется к 32 битному формату IEEE 754.

Описание преобразования в 32 битный формат IEEE 754:

- Число может быть + или - .
  Поэтому отводится 1 бит для обозначения знака числа:
  0-положительное
  1-отрицательное
  Этот самый старший бит в 32 битной последовательности .
- Далее пойдут биты экспоненты, для этого выделяют 1 байт (8 бит).
  Экспонента может быть, как и число, со знаком + или -.
  Для определения знака экспоненты, чтобы не вводить ещё один бит знака, добавляют смещение к экспоненте в половину байта +127(0111 1111).
  То есть, если наша экспоната = +7 (+111 в двоичной), то смещенная экспонента = 7+127=134. А если бы, наша экспонента была -7 , то смещенная экспонета=127-7 =120.
  Смещенную экспоненту записывают в отведенные 8 бит.
  При этом, когда нам будет нужно получить экспоненту двоичного числа, мы просто отнимем 127 от этого байта.
- Оставшиеся 23 бита отводят для мантиссы.
  Но, у нормализованной двоичной мантиссы первый бит всегда равен 1, так как число лежит в диапазоне 1<=M<2.
  Нет смыла, записывать единицу в отведенные 23 бита, поэтому в отведенные 23 бита записывают остаток от мантиссы.

В таблице представлено десятичное число 155,625 в 32-х битном формате IEEE754:

1 бит	8 бит	23 бит	IEEE 754
0	1000 0110	001 1011 1010 0000 0000 0000	431BA000 (hex)
0(dec)	134(dec)	1810432(dec)	
знак числа	смещенная экспонента	остаток от мантиссы	число 155,625 в формате IEEE754

В результате десятичное число 155,625 представленное в IEEE 754 с одинарной точностью равно 431BA000 (hex).

4.2 Преобразования числа формата 32 бит IEEE 754 в десятичное число

Чтобы записать число в стандарте IEEE 754 или восстановить его, необходимо знать три параметра:

- S- бит знака (31-й бит)
- E- смещенная экспонента (30-23 биты)
- M - остаток от мантиссы (22-0 биты)

Это целые числа которые записанные в числе IEEE 754 в двоичном виде.

Приведём формулу для получения десятичного числа из числа IEEE754 одинарной точности:

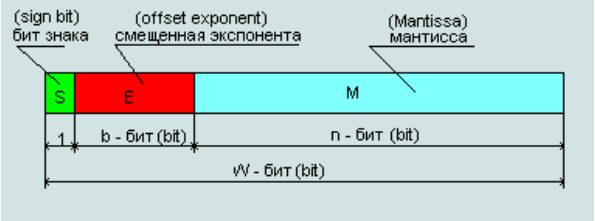
$$F = (-1)^S \ 2^{(E-127)}(1+M/2^{23})$$ где F - десятичное число

Проверяем наш пример:
$F = (-1)^0 \cdot 2^{(134-127)} \cdot (1+ 1810432 / 2^{23}) = 2^7 \cdot (1+0,2158203125) = 128 \cdot 1,2158203125 = 155,625$

Вывод этой формулы приводить не буду, всё видно и так.
Поясню только $(1+ M/2^{23})$ - это мантисса, единица в этой формуле- это та единица, которую мы выбросили из 23 бит, а остаток мантиссы в десятичном виде находим отношением двух целых чисел - остатка мантиссы к целому.

§5. Формальное представление чисел в стандарте IEEE 754 для любого формата точности.

Рис. 1 Представление числа в формате IEEE 754



где:

- S - бит знака, если S=0 - положительное число; S=1 - отрицательное число

- E - смещенная экспонента двоичного числа;
  $exp_2 = E - (2^{(b-1)} - 1)$ - экспонента двоичного нормализованного числа с плавающей точкой
  $(2^{(b-1)} - 1)$ - заданное смещение экспоненты (в 32-битном ieee754 оно равно +127 см.выше)
- M - остаток мантиссы двоичного нормализованного числа с плавающей точкой

Формула вычисления десятичных чисел с плавающей точкой, из чисел представленных в стандарте IEEE754:

$$F = (-1)^S \ 2^{(E-2^{(b-1)}+1)}(1+M/2^n)$$ (Формула №1)

Используя формулу №1 вычислим формулы для нахождения десятичных чисел из форматов одинарной (32 бита) и двойной (64 бита) точности IEEE 754:

Рис.2 Формат числа одинарной точности (single-precision) 32 бита

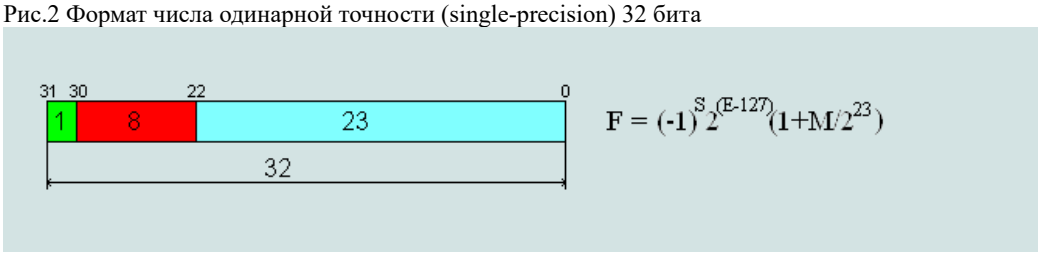
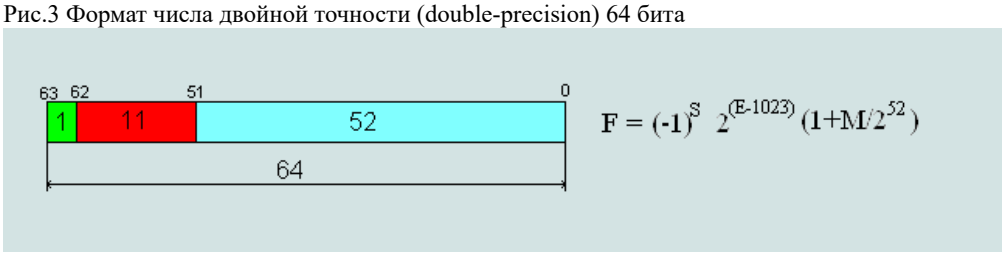


Рис.3 Формат числа двойной точности (double-precision) 64 бита



§6. Представление денормализованного числа и других чисел в формате IEEE 754

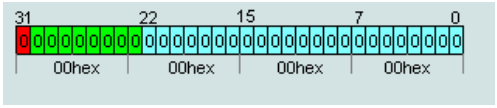
Если применить формулу №1 для вычисления минимального и максимального числа одинарной точности представленного в IEEE754, то получим следующие результаты:

- `00 00 00 00 hex= 5,87747175411144e-39 (минимальное положительное число)`
- `80 00 00 00 hex=-5,87747175411144e-39 (минимальное отрицательное число)`
- `7f ff ff ff hex= 6,80564693277058e+38 (максимальное положительное число)`
- `ff ff ff ff hex=-6,80564693277058e+38 (максимальное отрицательное число)`

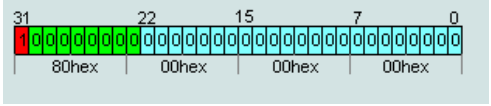
Отсюда видно, что невозможно представить число нуль или бесконечность в заданном формате.

Поэтому формула №1 не применяется в следующих случаях:

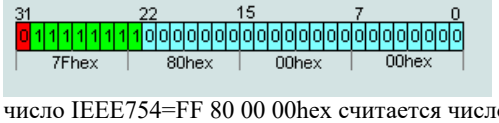
1. число IEEE754=00 00 00 00hex считается числом +0



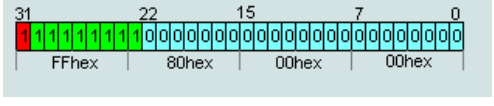
число IEEE754=80 00 00 00hex считается числом -0



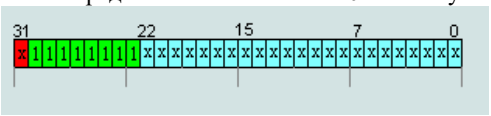
2. число IEEE754=7F 80 00 00hex считается числом +∞



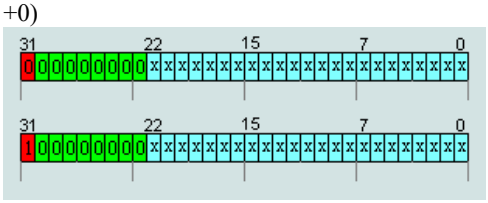
число IEEE754=FF 80 00 00hex считается числом -∞



3. числа IEEE754=FF (1xxx)X XX XXhex не считается числами (NAN), кроме случая п.2
числа IEEE754=7F (1xxx)X XX XXhex не считается числами (NAN), кроме случая п.2
Число представленное в битах с 0...22 могут быть любым числом кроме 0 (т.е.+∞ и -∞).



4. числа IEEE754=(x000) (0000) (0xxx)X XX XXhex считаются денормализованными числами, за исключением чисел п.1(то есть -0 и +0)



Формула расчета денормализованных чисел:

$$F = (-1)^S \; 2^{(E-2^{(b-1)}+2)} \; M/2^n$$

(Формула №2)

Пояснения к исключительным числам:

- С нулем понятно. Без него никак нельзя. Смущает только наличие двух нулей. Думаю, это было сделано для симметричности.
- - ∞/ +∞ , тоже понятно. Числа, которые больше границ диапазона представления чисел считаются бесконечными.
- Не числа NAN(No a Numbers). К ним относятся символы, или результаты недопустимых операций.
- Денормализованные числа. Это числа, мантиссы которых лежат в диапазоне 0.1 <= M <1.
  Денормализованные числа находятся ближе к нулю, чем нормализованные.
  Денормализованные числа как бы разбивают минимальный разряд нормализованного числа на некоторое подмножество.
  Сделано так потому, что в технической практике чаще встречаются величины близкие к нулю.

§7. Сведенья по числам одинарной и двойной точности, представленным в формате IEEE 754.

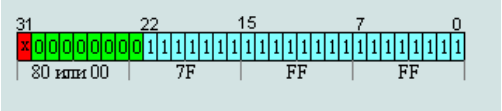
7.1 Вычисление границ диапазона для чисел одинарной точности IEEE 754

Зная формат чисел с одинарной точностью стандарта IEEE 754 можно посчитать границы диапазона представления действительных чисел в этом формате.
Для этого подставим значения максимальных и минимальных абсолютных чисел IEEE 754 в формулы №1 и №2.

Минимальное нормализованное число по (абсолютное)

$00\ 80\ 00\ 00 = 2^{-126}\cdot(1+0/2^{23}) = 2^{-126} \approx 1{,}17549435\cdot e^{-38}$
$80\ 80\ 00\ 00 = -2^{-126}\cdot(1+0/2^{23}) = -2^{-126} \approx -1{,}17549435\cdot e^{-38}$

Максимальное денормализованое число (абсолютное)



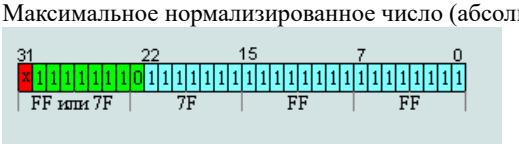
$00\ 7F\ FF\ FF = 2^{-126}\cdot(1-2^{-23}) \approx 1{,}17549421\cdot e^{-38}$
$80\ 7F\ FF\ FF = -2^{-126}\cdot(1-2^{-23}) \approx -1{,}17549421\cdot e^{-38}$
Отсюда видно что минимальное нормализированное число граничит с максимальным денормализированным.

Минимальное денормализованное число (абсолютное)

$00\ 00\ 00\ 01 = 2^{-126}\cdot 2^{-23} = 2^{-149} \approx 1{,}40129846\cdot e^{-45}$

80 00 00 01 = $-2^{-126} \cdot 2^{-23} = 2^{-149} \approx -1{,}40129846 \cdot e^{-45}$
Это число граничит с нулем.

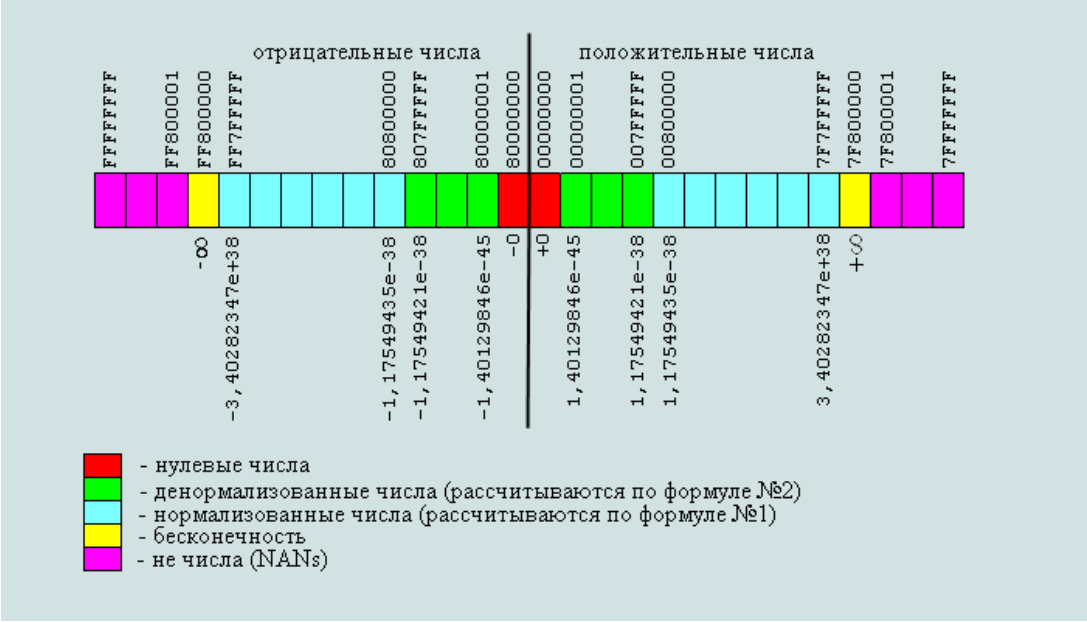
Максимальное нормализированное число (абсолютное)



7F 7F FF FF = $2^{127} \cdot (2-2^{-23}) \approx 3{,}40282347 \cdot e^{+38}$
FF 7F FF FF = $-2^{127} \cdot (2-2^{-23}) \approx -3{,}40282347 \cdot e^{+38}$
Это число граничит с бесконечностью.

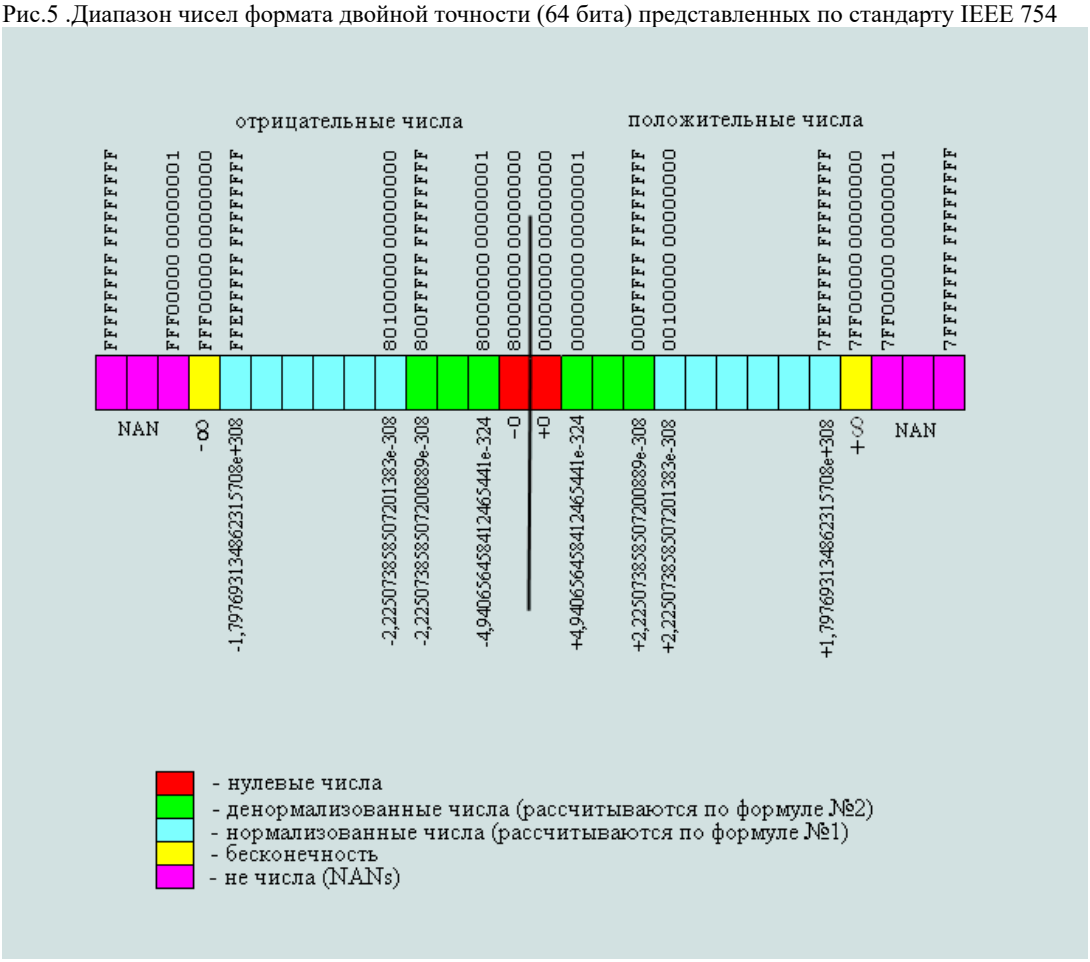
7.2 Полный диапазон чисел одинарной точности (32 бит) по стандарту IEEE754

Рис.4 .Диапазон чисел формата одинарной точности (32 бита) представленных по стандарту IEEE 754



7.3 Полный диапазон чисел двойной точности (64 бит) по стандарту IEEE754

Рис.5 .Диапазон чисел формата двойной точности (64 бита) представленных по стандарту IEEE 754



7.4 Точность представления вещественных чисел в формате IEEE754.

Числа представленные в формате IEEE754 представляют конечное множество, на которое отображается бесконечное множество вещественных чисел.
Поэтому исходное число может быть представлено в формате IEEE754 с ошибкой.

Рис.6 Функция ошибки точности представления числа в IEEE754

Абсолютная максимальная ошибка для числа в формате IEEE754 равна в пределе половине шага чисел.
Шаг чисел удваивается с увеличением экспоненты двоичного числа на единицу.
То есть, чем дальше от нуля, тем шире шаг чисел в формате IEEE754 по числовой оси.

Шаг денормализованных чисел равен $\mathbf{2^{(E-149)}}$ (Single) и $\mathbf{2^{(E-1074)}}$ (Double).
Соответственно предел макс. абсолютной ошибки будет равен 1/2 шага числа: $\mathbf{2^{(E-150)}}$ (Single) и $\mathbf{2^{(E-1075)}}$ (Double).
Относительная ошибка в % будет равна: $\mathbf{(2^{(E-150)}/F)*100\%}$(Single) и $\mathbf{(2^{(E-1075)}/F)*100\%}$ (Double).

Шаг нормализованных чисел равен $\mathbf{2^{(E-150)}}$ (Single) и $\mathbf{2^{(E-1075)}}$ (Double).
Соответственно предел макс. абсолютной ошибки будет равен 1/2 шага числа: $\mathbf{2^{(E-151)}}$ (Single) и $\mathbf{2^{(E-1076)}}$ (Double).
Относительная ошибка в % будет равна: $\mathbf{(2^{(E-151)}/F)*100\%}$(Single) и $\mathbf{(2^{(E-1076)}/F)*100\%}$ (Double).

Максимальная относительная ошибка для денормализованного числа(single/double):

$$\frac{2^{(E-150)}}{2^{(E-126)}\frac{M}{2^{23}}} = \frac{1}{2M}$$

Максимальная относительная ошибка нормализованного числа(single):

$$\frac{2^{(E-151)}}{2^{(E-127)}(1+\frac{M}{2^{23}})} = \frac{1}{2^{24}+2M}$$

Максимальная относительная ошибка нормализованного числа(double):

$$\frac{2^{(E-1076)}}{2^{(E-1023)}(1+\frac{M}{2^{52}})} = \frac{1}{2^{53}+2M}$$

Таблица 1. Максимальная возможная ошибка для чисел Single

IEEE754, hex	число, dec	абсолютная ошибка, dec	относительная, %
00000001	2^{-149} ≈1,401298e-45	2^{-150}≈0,700649e-45	=50
00000002	2^{-148} ≈2,802597e-45	2^{-150}≈0,700649e-45	=25
00000032	≈7,00649e-44	2^{-150}≈0,700649e-45	=1
007FFFFF	≈1,175494e-38	2^{-150}≈0,700649e-45	≈5,96e-6
00800001	≈1,175494e-38	2^{-149} ≈1,401298e-45	≈11,9209e-6
0DA24260	≈1,0e-30	2^{-123} ≈9,4039e-38	≈9,4039e-6
1E3CE508	≈1,0e-20	2^{-90} ≈8,0779e-28	≈8,0779e-6
2EDBE6FF	≈1,0e-10	2^{-57} ≈6,9389e-18	≈6,9389e-6
3F800000	≈1,0	2^{-23} ≈1,192e-7	≈11,9209e-6
41200000	≈10,0	2^{-20} ≈9,5367e-7	≈9,5367e-6
42C80000	≈1,0e+2	2^{-17} ≈7,6294e-6	≈7,62939e-6
501502F9	≈1,0e+10	2^{10} ≈1,024e+3	≈10,24e-6
60AD78EC	≈1,0e+20	2^{43} ≈8,7961e+12	≈8,7961e-6
7149F2CA	≈1,0e+30	2^{76} ≈7,5558e+22	≈7,5558e-6
7F7FFFFF	≈+3,40282e+38	2^{104} ≈2,02824e+31	≈5,96e-6

Таблица 2. Максимальная возможная ошибка для чисел Double

IEEE754, hex	число, dec	абсолютная ошибка, dec	относительная, %
00000000 00000001	2^{-1074} ≈4,940656e-324	2^{-1075}≈2,470328e-324	=50
00000000 00000002	2^{-1073} ≈9,881313e-324	2^{-1075}≈2,470328e-324	=25
00000000 00000032	≈2,470328e-322	2^{-1075}≈2,470328e-324	=1
000FFFFF FFFFFFFF	≈2,225073e-308	2^{-1075}≈2,470328e-324	≈1,110223e-14
00100000 00000001	≈2,225074e-308	2^{-1074} ≈4,940656e-324	≈2,220446e-14
2B2BFF2E E48E0530	≈1,0e-100	2^{-385} ≈1,268971e-116	≈1,268971e-14
3FF00000 00000000	=1,0	2^{-52} ≈2,220446e-16	≈2,220446e-14
54B249AD 2594C37D	≈1,0e+100	2^{280} ≈1,942669e+84	≈1,942669e-14
6974E718 D7D7625A	≈1,0e+200	2^{612} ≈1,699641e+184	≈1,699641e-14
7FEFFFFF FFFFFFFF	≈1,79769e+308	2^{971} ≈1,99584e+292	≈1,110223e-14

Из выше приведенного следует, что основная масса чисел в формате IEEE754 имеет стабильную небольшую относительную погрешность:

Максимально возможная относительная погрешность для числа Single составляет 2^{-23}*100% =11,920928955078125e-6 %

Максимально возможная относительная погрешность для числа Double составляет 2^{-52}*100% =2,220446049250313080847263336181 6e-14 %

7.5 Общие сведенья для чисел одинарной и двойной точности стандарта IEEE 754.

Таблица 3. Сведенья о формате 32/64 бит в стандарте ANSI/IEEE Std 754-1985

наименование формата	single-precision	double-precision
длина числа, бит	32	64
смещенная экспонента (E), бит	8	11
остаток от мантиссы (M), бит	23	52
смещение	127	1023
двоичное денормализованое число	$(-1)^S \cdot 0,M \cdot exp_2^{-127}$,где M-бинарное	$(-1)^S \cdot 0,M \cdot exp_2^{-1023}$,где M-бинарное
двоичное нормализованое число	$(-1)^S \cdot 1,M \cdot exp_2^{(E-127)}$,где M-бинарное	$(-1)^S \cdot 1,M \cdot exp_2^{(E-1023)}$,где M-бинарное
десятичное денормализованное число	$F = (-1)^S \cdot 2^{(E-126)} \cdot M/2^{23}$	$F = (-1)^S \cdot 2^{(E-1022)} \cdot M/2^{52}$
десятичное нормализованное число	$F = (-1)^S \cdot 2^{(E-127)} \cdot (1 + M/2^{23})$	$F = (-1)^S \cdot 2^{(E-1023)} \cdot (1 + M/2^{52})$
Абс. макс. возм. погрешность числа	$2^{(E-150)}$	$2^{(E-1075)}$
Отн. макс. возм. погрешность денорм. числа	1/(2M)	1/(2M)
Отн. макс. возм. погрешность норм. числа	$1/(2^{24}+2M)$	$1/(2^{53}+2M)$
минимальное число	$\pm 2^{-149} \approx \pm 1,40129846 \cdot e^{-45}$	$\pm 2^{-1074} \approx \pm 4,94065646 \cdot e^{-324}$
максимальное число	$\pm 2^{127} \cdot (2-2^{-23}) \approx \pm 3,40282347 \cdot e^{+38}$	$\pm 2^{1023} \cdot (2-2^{-52}) \approx \pm 1,79769313 \cdot e^{+308}$

§8. Округление чисел в стандарте IEEE 754.

Стандарт IEEE754 предусматривает четыре способа округления чисел.

Способы округления чисел по стандарту IEEE 754:

- Округление стремящееся к ближайшему целому.
- Округление стремящееся к нулю.
- Округление стремящееся к +∞
- Округление стремящееся к -∞

Таблица 3. Примеры округления чисел до десятых

исходное число	к ближ. целому	к нулю	к +∞	к -∞
1,33	1,3	1,3	1,4	1,3
-1,33	-1,3	-1,3	-1,3	-1,4
1,37	1,4	1,3	1,4	1,3
-1,37	-1,4	-1,3	-1,3	-1,4
1,35	1,4	1,3	1,4	1,3
-1,35	-1,4	-1,3	-1,3	-1,4

Как происходит округление показано на примерах в таблице 3.
При преобразовании чисел необходимо выбрать один из способов округления.
По умолчанию это первый способ -округление к ближайшему целому.
Часто в различных устройствах используют второй способ - округление к нулю.
При округлении к нулю нужно просто отбросить не значащие разряды числа, поэтому этот способ самый легкий в аппаратной реализации.

§9. Проблемы компьютерных вычислений, вызванные использованием стандарта IEEE754.

Стандарт IEEE 754 широко применяется в технике и программировании.
Большинство современных микропроцессоров изготовляются с аппаратной реализацией представления вещественных переменных в формате IEEE754.
Язык программирования и программист не могут изменить эту ситуацию, иного преставления вещественного числа в микропроцессоре не существует.
Когда создавали стандарт IEEE754-1985 представление вещественной переменной в виде 4 или 8 байт казалось очень большой величиной, так как объём оперативной памяти MS-DOS был равен 1 Мб.
А, программа в этой системе могла использовать только 0,64 Мб.
Для современных ОС размер в 8 байт является ничтожным, тем не менее переменные в большинстве микропроцессоров продолжают представлять в формате IEEE754-1985.

Рассмотрим ошибки компьютерных вычислений, вызванные применением чисел в формате IEEE754.

9.1 Ошибки связанные с точностью представления вещественных чисел в формате IEEE754. Опасная редукция.

Данная ошибка всегда присутствую в компьютерных вычислениях.
Причина её возникновения описана в п.7.4.

Радует только то, что величина относительной ошибки имеет размерность для single 10^{-6} для double 10^{-14}

Величины абсолютных ошибок могут быть значительными, максимально для single 10^{31} и для double 10^{292}, что может вызывать определённые проблемы вычислений.

```
//Пример 1. Ошибка вызванная точностью представления числа в формате IEEE754
		#include "stdio.h"

		int
		main(int argc, char *argv[])
		{
			float a, b, f;
			a=123456789;
			b=123456788;
			f=a-b;
			printf("Result: %f\n", f);
			return 0;

		}
		Result: 8.000000  (Ответ должен быть 1.000000)
```

Если пример посчитать на бумажке, то ответ будет 1. Абсолютная ошибка равна +7.
Почему ответ получился неправильным?
Число 123456789 в single=4CEB79A3hex(ieee)=123456792(dec) абсолютная ошибка представления равна +3
Число 123456788 в single=4CEB79A2hex(ieee)=123456784(dec) абсолютная ошибка представления равна -4
Относительная погрешность исходных чисел приблизительно равна 3,24e-6%
В результате одной операции относительная погрешность результата стала 800%, т.е. увеличилась в 2,5e+8 раз.
Это я называю **"Опасной редукцией"**, т.е. катастрофическим понижением точности вычислений в операция где абсолютное значение результата много меньше любого из входного значения переменных.

На самом деле ошибки точности представления числа наиболее безобидные в компьютерных вычислениях, и обычно многие программисты на них не обращают никакого внимания.
Тем не менее они вас могут сильно огорчить.

9.2 Ошибки связанные с неправильным приведением типов данных. Дикие ошибки.

Эти ошибки вызваны тем, что исходное число представленное в формате single и в формате double обычно не равны друг другу.
Например: исходное число 123456789,123456789
Single: 4CEB79A3=+123456792,0(dec)
Double: 419D6F34547E6B75=+123456789,123456791043281555175781 25
Разница между Single и Double составит:2,87654320895671844482421875

Вот пример на VB:

```
	Private Sub Command1_Click()
			Dim a As Single
			Dim b As Double
			Dim c As Double

			a = 123456789.123457
			b = 123456789.123457
			c = a - b
			Text1.Text = c

		End Sub
	Результат: 2,87654320895672 (должен быть 0)
```

Относительная погрешность результата равна: ∞ (бесконечности)
Такую ошибку называют "грязным нулем".
Если переменные привести к одному типу, то этой ошибки не будет.

```
		Private Sub Command1_Click()
			Dim a As Single
			Dim b As Single
			Dim c As Single
```

```
                a = 123456789.123457
                b = 123456789.123457
                c = a - b
                Text1.Text = c

        End Sub
Результат: 0.0
```

Поэтому переменные и промежуточные результаты компьютерных вычислений должны быть приведены к одному типу данных.
Например, требование приведения данных к одному типу изложено в стандарте на язык Си ISO/IEC 9899:1999.

Обратите внимание на то, что недостаточно просто привести все исходные данные к одному типу.
Необходимо привести также результаты промежуточных операций к одному типу.
Вот пример с ошибкой в промежуточном результате:

```
'Пример 1 ошибка в приведении промежуточных данных на VB (Visual Studio)
        Private Sub Command1_Click()
                Dim a As Single
                Dim b As Single
                Dim c As Single

                a = 1
                b = 3
                c = a / b
                c = c - 1 / 3
                Text1.Text = c

        End Sub
Результат: 9,934108E-09 (Должен быть 0.0)
```

Здесь ошибка возникает потому, что промежуточный результат 1/3 в строке c=c-1/3 будет иметь тип double, а не single.
Чтобы избавиться от ошибки вам надо привести промежуточный результат к типу single с помощью оператора приведения типов CSng.

```
'Пример 2 приведение промежуточных данных на VB (Visual Studio)
        Private Sub Command1_Click()
                Dim a As Single
                Dim b As Single
                Dim c As Single

                a = 1
                b = 3
                c = a / b
                c = c - CSng(1 / 3)
                Text1.Text = c

        End Sub
Результат: 0.0
```

Пример приведения типа данных для GNU C, присланный Григорием Ситкаревым :

```
        //Вариант 1 с не приведенным промежуточным результатом:
        #include "stdafx.h"
        #include "stdlib.h"
        #include "stdio.h"
        #include "math.h"

        int
        main(int argc, char *argv[])
        {
         float a, b, c, d;
                a = 1.0;
         b = 3.0;
                c = a / b;
                d = (c - 1.0/3.0) * 1.0e9;//результат деления 1/3 имеет к тип double
         printf("Result: %f\n", d);
                return 0;
        }
        Result: 9.934108 (Должен быть 0.0)
```

```
//Вариант 2 с приведенным промежуточным результатом:
#include "stdafx.h"
#include "stdlib.h"
#include "stdio.h"
#include "math.h"

int
main(int argc, char *argv[])
{
        float a, b, c, d;
        a = 1.0;
        b = 3.0;
        c = a / b;
        d = (c - 1.0f/3.0f) * 1.0e9f;//результат деления 1/3 приведен к типу float
 printf("Result: %f\n", d);
        return 0;
}
Result: 0.0
```

Во втором варианте вы видите, что деление констант в промежуточном результате приведено к типу "float" (одиночная точность в Си).
Данные варианты компилировались и исполнялись с помощью "GNU C".
Если компилировать и исполнить показанные выше варианты на VC++ (Visual Studio), то результаты будут обратными.
То есть вариант 2 будет иметь результат -9.934108, а вариант 1 Result: 0.000000.
Отсюда можно сделать неутешительный вывод, что результат вычислений может зависеть от типа и версии компилятора.
В данном случае можно предположить, что компилятор VC++ автоматически приводит типы переменных, и попытка их насильственного приведения к одному типу заканчивается ошибкой.

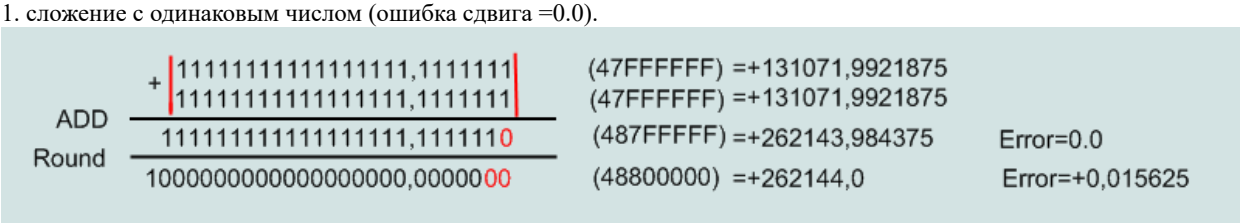
Если Вариант 1 (без приведения типов) выполнить с переменными двойной точности (double), то ошибки приведения данных не будет и Result=0.000000
Поэтому в большинстве случаев чтобы избавиться от приведения типов данных надо просто использовать тип данных double и забыть о существовании типа single(float).
Ошибки вычислений вызванные не приведением типов данных я называю **"Дикими ошибками"**, так как они связаны с незнанием стандартов и теории программирования (т.е. с плохим фундаментальным образованием)

9.3 Ошибки вызванные сдвигом мантисс. Циклические дыры.

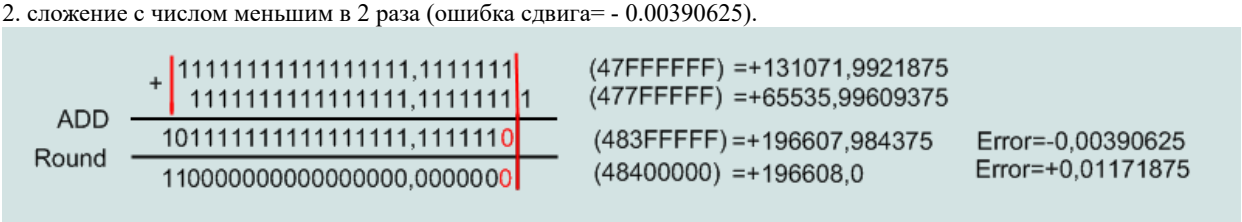
Эти ошибки связаны с потерей точности результата при неполном пересечении мантисс чисел на числовой оси.
Если мантиссы чисел не пересекаются на числовой оси, то операции сложения и вычитания между этими числами невозможны.
К примеру возьмём число Single: 47FFFFFF = +131071,9921875(dec)
В двоичной системе это число выглядит так: +11111111111111111,1111111

Покажем несколько компьютерных операций сложений и этим числом в формате Single
Значащих цифр в мантиссе двоичного числа в формате Single не более 24
Красным цветом показаны цифры выходящие за этот предел и не участвующие в формате Single

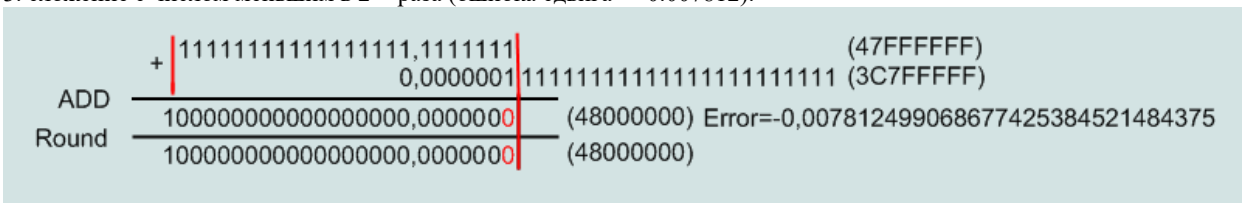
1. сложение с одинаковым числом (ошибка сдвига =0.0).



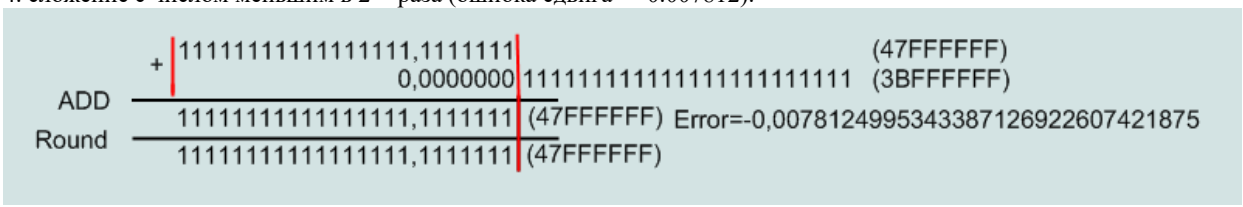
2. сложение с числом меньшим в 2 раза (ошибка сдвига= - 0.00390625).



3. сложение с числом меньшим в 2^{23} раза (ошибка сдвига= - 0.007812).



4. сложение с числом меньшим в 2^{24} раза (ошибка сдвига= - 0.007812).



В последнем примере мантиссы чисел разошлись, и арифметические операции с этими числами становятся бессмысленными.

Как видно из приведенных примеров ошибка сдвига возникает если у исходных нормализованных чисел различаются экспоненты.
Если числа отличаются более чем в 2^{23} (для single) и 2^{52} (для double), то операции сложения и вычитания между этими числами не возможны.
Максимальная относительная погрешность результата операции равна примерно 5,96e-6%, что не превышает относительную погрешность представления числа (см.п.9.1).

Хотя с относительной погрешностью здесь всё в порядке, здесь есть другие проблемы.
Во-первых, работать с числами можно только в узком диапазоне числовой оси, где мантиссы пересекаются.
Во-вторых, для каждого исходного числа существует предел выполнения цикла называемый **"Циклической дырой"**.
Поясню, если существует цикл в котором исходное число суммируется к сумме, то существует численный предел суммы для этого числа.
То есть сумма достигнув определённой величины перестает увеличиваться от сложения её с исходным числом.

Приведу пример циклической дыры в автоматической системе управления:
Имеется фармацевтический цех, производящий таблетки весом 10мг.
Состоящий из: формовочной машины, накопительного бункера на 500 кг, фасовочной машины, автоматической системы управления.
Формовочная машина подает в бункер по 10 таблеток одновременно.
Фасовочная машина забирает по одной таблетке.
Автоматическая система управления учитывает таблетки поступившие в бункер от формовочной машины и взятые из бункера фасовочной машиной.
То есть существует программа, которая показывает заполнения бункера продукцией в кг.
Когда в бункере будет более 500 кг продукции, формовочная машина встает на паузу , она включиться кода в бункере станет 200 кг продукции.
Фасовочная машина остановиться если в бункере будет менее 10 кг и запустится когда в бункере будет более 100 кг продукции.
Обе машины могут периодически останавливаться для обслуживания , не зависимо друг от другу (благодаря бункеру).

Как вы понимаете, такая система работает в бесконечном цикле.
Допустим в один из дней фасовочная машина простояла слишком долго и бункер заполнился до 300 кг.
Что произойдет после её включения?

Упрощенный пример работы цикла программы управления:

```
Private Sub Command1_Click()
        Dim a As Single 'вес таблетки в кг
        Dim c As Single 'продукции в бункере в кг
        Dim n As Long 'количество циклов

        c = 300 'исходный вес бункера
        a = 0.00001 'вес таблетки

        For n = 1 To 10000000
                c = c - a 'одна таблетка забирается фасовочной машиной
        Next n
        Text1.Text = c 'измененный вес бункера
End Sub
```

В данном примере фасовочная машина забрала из бункера 100 кг продукции, а вес продукции в бункере не изменился.
Почему не изменился?
Потому, что мантиссы чисел 300 и 0,00001 не пересекаются для формата single.

Далее формовочная машина доведет вес бункера до 500 кг и остановится.
Фасовочная машина заберет все таблетки из бункера и тоже остановится.
Программа будет показывать вес в бункере 500кг.

Прибегут специалисты, проверят датчики, провода, компьютер и скажут что программа зависла.
Но, программа не зависала, она продолжает работать без сбоев и любой контроль это подтвердит.
Просто число 0,0001 попало в циклическую дыру и выйти из него не может.

В результате нам крупно повезло, что это был фармацевтический цех, а не Саяно-Шушенская ГЭС.

На самом деле опытный программист никогда не будет делать циклическое вычитание(или суммирование) таким образом.
Этот пример вымышлен специально, и так считать нельзя, хотя с точки зрения математика здесь всё безупречно.
Эта ошибка свойственна математикам и начинающим программистам.
Я бы сказал, что основная работа программиста заключается в борьбе с погрешностями, а не в математических решениях поставленной задачи.

Привожу правильный пример решения этой задачи, любезно предоставленный Ситкаревым Григорием:

```
#include "stdlib.h"
#include "stdio.h"
#include "math.h"

struct acc_comp {
                        float value;
                        float compens;
                };

void
sub_compens(struct acc_comp *acc, float quantum)
        {
                float tmp, c;

                tmp = quantum - acc->compens;
                c = acc->value - tmp;
                acc->compens = acc->value - c - tmp;
                acc->value -= tmp;
        }

void
sum_compens(struct acc_comp *acc, float quantum)
        {
                float tmp, c;

                tmp = quantum - acc->compens;
                c = acc->value + tmp;
                acc->compens = c - acc->value - tmp;
                acc->value += tmp;
        }

void
sub_test()
        {
                struct acc_comp hopper;
                struct acc_comp bunker;
                float tablet;
                int n, i;

                n = 10000000;
                hopper.value = 300.0;
                hopper.compens = 0.0;
                bunker.value = 0.0;
                bunker.compens = 0.0;
                tablet = 0.00001;

                for (i = 0; i < n; i++)
                        {
                                sub_compens(&hopper, tablet);
                                sum_compens(&bunker, tablet);
                        }

                hopper.value -= hopper.compens;
                bunker.value += bunker.compens;

                printf("Left in hopper: %04.5f kg\n", hopper.value);
                printf("Held in bunker: %04.5f kg\n", bunker.value);
        }
```

```
int
main(int argc, char *argv[])
        {
                sub_test();

                return 0;
        }
```

Показанный выше пример взят из реального промышленного кода.
Для наглядности упростим выше приведенный пример.

```
#include "stdlib.h"
#include "stdio.h"
#include "math.h"

float bunker, bunker1, tablet, tablet1, compens;
long int n, i;

int
main(int argc, char *argv[])
        {
                tablet = 0.00001; /* вес таблетки */
                tablet1 = 0.0; /* вес таблетки с учетом ошибки в предыдущих итерациях */
                bunker = 300.0; /* исходный вес бункера */
                bunker1 = 0.0; /* вес бункера после очередной итерации */
                compens = 0.0; /* компенсация веса таблетки */

                n = 10000000; /* количество циклов */

                for (i = 0; i < n; i++)
                  {
                        /* вес таблетки с компенсацией ошибки */
                        tablet1 = tablet - compens;
                        /* вес бункера после вычета скомпенсированной таблетки */
                        bunker1 = bunker - tablet1;
                        /* вычисление компенсации для следующей итерации */
                        compens = (bunker - bunker1) - tablet1;
                        /* новый вес бункера */
                        bunker = bunker - tablet1;
                  }

                printf("Bunker: %04.5f kg\n", bunker);

                return 0;
        }
```

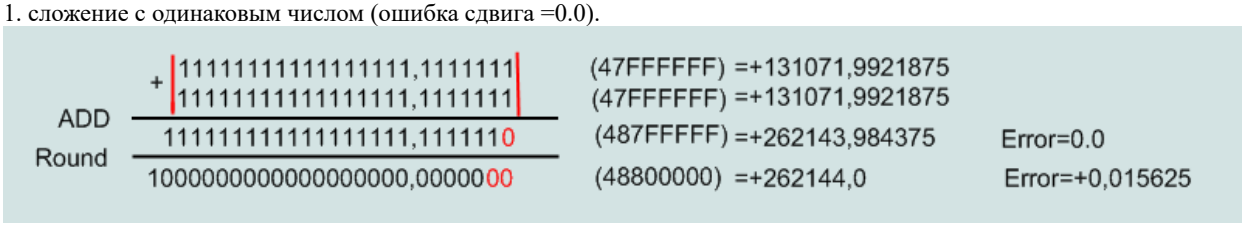
Как видно из этого примера, программисту приходится вычислять погрешность результата в каждом цикле, чтобы учесть её в следующем цикле.
Обратите внимание, что программист должен быть абсолютно готов к тому, что некоторые основные понятия математики могут не выполняться при вычислениях в формате IEEE754.
Например, правила алгебраической коммутативности (a + b) + c = (a + c) + b, обычно не выполняются при таких вычислениях.
К сожалению, в современном фундаментальном математическом образовании этому вопросу уделяют мало внимания, оставляя решения проблем инструментария на прикладного программиста.

9.4 Ошибки вызванные округлением. Грязный ноль.

При компьютерных вычислениях можно выделить два вида округления:
1. Результат выполненной арифметической операции всегда округляется.
2. Вывод и ввод вещественного числа в окно Windows происходит с округлением.

В первом случае переменная округляется по одному из 4-типов округления IEEE754, по умолчанию округление происходит к ближайшему целому.
При этом переменная принимает новое округлённое значение.
В п.9.2 мы рассматривали сложение двух одинаковых чисел:

1. сложение с одинаковым числом (ошибка сдвига =0.0).



Здесь результат операции сложения двух чисел абсолютно точен, но результат был округлен микропроцессором.
Таким образом к точному результату была добавлена ошибка округления. В общем случае ошибка округления находится в пределах точности числа.

Во втором случае переменная не меняет своего значения, просто в окно Windows выводится округлённое значение вещественного числа.
Получается, что исходная переменная и её отображение в окне Windows это разные числа.
Это не ошибка формата IEEE754, это ошибка Windows.
Для переменных типа Single в окне Windows отображается 7 значащих цифр округлённых до ближайшего целого.
3DFCD6EA =+0,12345679104328155517578125 в окне отображается как 0,1234568
Для переменных типа Double в окне Windows отображается 15 значащих цифр округлённых до ближайшего целого.
3FBF9ADD3746F67D=+0,1234567890123460937035204665335186291486024856567382812 5 отображается как 0,123456789012346

Вопрос какое значение имеет переменная когда мы вводим в окно Windows 0,123456789012346 ?
Это значение будет равно такому числу:
3FBF9ADD3746F676=+0,12345678901234599655900581183232134208083152770996093 75
То есть значение переменной 3FBF9ADD3746F67D мы вообще не можем вставить напрямую в код программы.
Но, мы можем схитрить и вставить в программу x=0.123456789012346 +1E-16.
Полученная переменная будет равна 3FBF9ADD3746F67D (это используется в примере с грязным нулём)
Отобразить или вести через окно ПК такое число невозможно.

В результате действий Windows возникает ряд неприятных ситуаций.
1. У вас нет технической возможности отображать или вводить точные значения переменных в окнах, что само по себе очень печально.
2. Возникновение серьёзных ошибок, таких как грязный ноль.
"Грязный ноль" это когда вы или программа считаете, что переменная не равная нулю - равна нулю

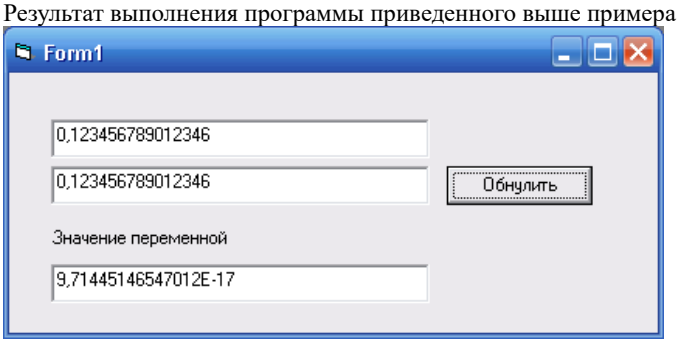
Очень часто эта ошибка возникает в интерфейсе "оператор-машина".
Например, при обнулении тары в весовых программах.

```
Dim a As Double
        'обнуление видимой величины
Private Sub Command1_Click()
                Dim b As Double
                Dim c As Double

                b = Val(Replace(Text2.Text, ",", "."))
                c = a - b
                 Text3.Text = c
End Sub

Private Sub Form_Load()
                'ввод числа 3FBF9ADD3746F67D
                a = 0.123456789012346 + 1E-16
                Text1.Text = a
End Sub
```

Результат выполнения программы приведенного выше примера



В результате переменную, которую оператор считает нулем - нулю не равна
Относительная погрешность результата равна бесконечности.
В логических операциях сравнения этот не ноль может увести исполнение программы в другую ветвь алгоритма.

9.5 Ошибки на границе норма/денорма числа. Числа убийцы.

Эти ошибки возникают при работе с числами находящимися на границе нормализованного/денормализованного представления чисел.
Они связанны с различием в представление чисел в формате IEEE754 и в различии формул перевода формата IEEE754 в вещественные числа.
То есть устройства (или программы) должны применять различные алгоритмы в зависимости от положения вещественного числа на числовой оси формата.
Кроме того, что это приводит к усложнению устройств и алгоритмов, ещё возникает неопределённость переходной зоны.
Неопределённость переходной зоны заключается в том, что стандарт не определяет конкретного значения границы перехода.
По сути дела граница перехода находится между двумя вещественными числами:
Последним денормализованным числом 000FFFFFFFFFFFF:
Точное десятичное значение этого числа:
+2,225073858507200889024586876085859887650423112240959465493524802562440009228235695178775888803759155264230978095043431208587738715835729182199302029437922422355981982750124204178896957131179108226104397197960400045489739193807919893608152561311337614984204327175103362739154978273159414382813627511383860409424946494228631669542910508020181592664213499660651780309507591305871984642390606863710200510872328278467884363194451586613504122347901479236958520832159762106637540161373658304419360371477835530668283453563400507407304013560296804637591858316312422452159926254649430083685186171942241764645513713542013221703137049658321015465406803539741790602258950302350193751977303094576317321085250729930508976158251915972075723245543477091246131749358028173446655273437 5e-308

и первым нормализованным числом 0010000000000000:
Точное десятичное значение этого числа:
+2,225073858507201383090232717332404064219215980462331830553327416887204434813918195854283159012511020564067339731035811005152434161553460108856012385377718821130777993532002330479610147442583636071921565046942503734208375250806650616658158948720491179968591639648500635908770118304874799780887753749949451580451605050915399856582470818645113537935804992115981085766051992433352114352390148795699609591288891602992641511063466313393663477586513029371762047325631781485664350872122828637642044846811407613911477062801689853244110024161447421618567166150540154285084716752901903161322778896729707373123334086988983175067838846926092773977972858659654941091369095406136467568702398678315290680984617210924625396728515625e-308
Так как граница является вещественным числом, то её точность можно задавать до бесконечности и цифровому устройству или программе может не хватить разрядности для принятия решения к какому диапазону отнести данное число.

Для примера можно привести баг №53632 для PHP, который вызвал панику в начале 2011 года

```
< html>
	<body>
	<?php $d = 2.2250738585072011e-308; ?>
	end
	</body>
</html>
```

Ввод числа 2.2250738585072011e-308 вызывал зависание процесса почти с 100% загрузкой CPU.
Другие числа из этого диапазона проблем не вызвали (2.2250738585072009e-308, 2.2250738585072010e-308, 2.2250738585072012e-308)
Сообщение о баге поступило 30.12.2010, исправлено разработчиком 10.01.2011.
Так как PHP препроцессор используют большинство серверов, то у любого пользователя сети в течение 10 дней была возможность "вырубить" любой сервер.
Как пишут разработчики, что этот баг работает только в 32-х разрядных системах, но если повысить точность граничного числа, то я думаю, что и 64-х разрядные системы тоже повиснут (не проверено!).
Причина паники понятна: любой пользователь сети, при определённом уровне усердия и знания, имел возможность "вырубить" большую часть информационных ресурсов планеты в течении десяти дней.
Не хотелось - бы приводить ещё примеры таких чисел и таких ошибок.

§10 Заключение.

Из сказанного выше видно, что мнение о том, что при вычислениях с плавающей точкой результат не выходит за пределы относительной погрешности представления наибольшего числа является ложным.
Ошибки перечисленные в п.9 складываются друг с другом. Такие ошибки как грязный ноль и опасная редукция могут сделать погрешность вычислений неприемлемой.
Особое внимание при программировании компьютерных вычислений программист должен обратить на результаты близкие к нулю.

Некоторые специалисты считают, что этот формат чисел представляет угрозу человечеству.
Вы можете прочитать об этом в статье IEEE754-тика угрожает человечеству
Хотя многие факты из этой статьи излишне драматизированы, и возможно неправильно интерпретированы, но проблема компьютерных вычислений философски отражена правильно.

Я не сторонник драматизации вычислений по стандарту IEEE754. Стандарт работает с 1985 года и полностью вошел в стандарт IEEE754-2008, который расширил точность вычислений.
Но, проблема достоверности компьютерных вычислений сегодня очень актуальна, и стандарт IEEE754-2008 и рекомендации ISO не решили эту проблему.
Я думаю, в этой области нужна инновационная идея, которой разработчики стандарта IEEE754-2008 к сожалению не обладают.

Инновационные идеи, как правило, приходят со стороны.
Основные инновационные идеи в нашем мире были выдвинуты любителями (людьми мыслящими не за деньги).
Ярким примером такой ситуации было изобретение телефона.
Когда школьный учитель Александр Белл (Alexander Graham Bell) пришел с патентом на изобретение телефона к президенту

телекоммуникационной компании Western Union Company, которая владела трансатлантическим кабелем связи, с предложением купить у него патент на изобретение телефона, он его не выгнал - нет.
Президент этой компании предложил рассмотреть этот вопрос совету экспертов в области телеграфии, состоящему из специалистов и ученых в области телекоммуникаций.
Эксперты дали заключение, что это изобретение бесполезно в области телекоммуникаций и оно бесперспективно.
Некоторые специалисты даже написали в отчете, что это циркачество и шарлатанство !
Александр Белл, вместе со своим тестем, решил самостоятельно продвигать своё изобретение.
Примерно через 10 лет, гигант телекоммуникаций Western Union Co., был фактически вытеснен телефонным бизнесом из сферы телекоммуникационных технологий .
Сегодня вы можете увидеть во многих российских городах окошки с надписью Western Union, эта компания которая занимается переводом денег по всему миру, а когда-то она была международным гигантом телекоммуникаций.
Отсюда можно сделать вывод: **мнения экспертов в инновационных технологиях бесполезны!**
Если Вы думает что с момента изобретения телефона (1877 г.) в головах людей что-то изменилось, то вы ошибаетесь.

Если ученые(которые придумывают новое) и специалисты (которые знают как пользоваться известным) не могут решить проблему, то нужна инновация.

Ссылки на новые идеи в области представления вещественных чисел в аппаратных средствах:
1. Аппроксиметика
2. ....?
Если вы знаете другие инновационные идей в области представления вещественных чисел, то буду рад получить ссылки на эти источники.

Можно представлять вещественные числа, как числа с фиксированной запятой.
Для представления всего диапазона чисел Double достаточно иметь переменную состоящую из 1075 битов целой части и 1075 битов дробной части, т.е. около 270 байт на одну переменную.
При этом все числа будут представлены с одинаковой абсолютной точностью.
Можно работать с числами во всем диапазоне числовой оси, то есть становится возможным суммировать большие числа с малыми числами.
Шаг чисел по числовой оси будет равномерен, то есть числовая ось будет линейна.
Тип данных будет только один, т.е. не нужны целые, вещественные и др. типы.
Здесь возникает проблема реализации регистров микропроцессоров размерностью в 270 байт, но это не проблема для современной технологии.

Для написания п.9 мне пришлось создать программу которая представляет числа в виде переменной с фиксированной точкой, длиною 1075,1075 байтов.
Где число можно представить в виде строки символов ASCII, то есть один символ равен одному числовому разряду.
Так же пришлось написать все арифметические операции при работе со строками ASCII.
Это программа является аналогом бумажного расчета. Так как математические способности микропроцессора в ней не используются, то считает она довольно медленно. Зачем я это сделал?
Я не смог найти программы, которая смогла бы точно представить число формата IEEE754 в десятичном виде.
Так же я не нашел программы (хотя они безусловно есть, в чём нет сомнений) где в окно можно ввести 1075 значащих десятичных цифр.

Вот к примеру точно десятичное значение числа double 7FEFFFFFFFFFFFFF:
+179769313486231570814527423731704356798070567525844996598917476803157260780028538760589558632766878171540458953514382464234321326889464182768467546703537516986049910576551282076245490090389328944075868508455133942304583236903222948165808559332123348274797826204144723168738177180919299881250404026184124858368,0

Вы можете воспользоваться программой IEEE754 v.1.0
для изучения и оценки погрешностей при работе с вещественными числами представленными в формате IEEE754.

Список литературы:
1. *IEEE Standard for Binary Floating-Point Arithmetic. Copyright 1985 by The Institute of Electrical and Electronics Engineers, Inc 345 East 47th Street, New York, NY 10017, USA*

Благодарности:
Ситкареву Григорию Александровичу (sitkarev@komitex.ru, sinclair80@gmail.com). За помощь в создании статьи.

Назад Главная