Продолжаем говорить про арифметику

Андрей Акиньшин, JetBrains

DotNext 2016 Moscow

Знать математику — полезно

О чём будем разговаривать?

Сегодня в выпуске:

- Весёлые истории про арифметические баги
- Хитрые битовые хаки
- Удивительные рантаймы и JIT-компиляторы
- Коварные баги в .NET
- Неожиданные ASM-листинги
- Загадочный IEEE 754 и денормализованные числа

О чём будем разговаривать?

Сегодня в выпуске:

- Весёлые истории про арифметические баги
- Хитрые битовые хаки
- Удивительные рантаймы и JIT-компиляторы
- Коварные баги в .NET
- Неожиданные ASM-листинги
- Загадочный IEEE 754 и денормализованные числа
- **Проблемы с точностью:**
  кто виноват и что делать?
- **Скорость вычислений:**
  от чего зависит и как улучшать?

Отказ от ответственности

Нужно понимать:

- Все бенчмарки вымышлены, любые совпадения случайны
- Приведённые примеры нужны только для иллюстрации идей
- Погрешности малы, для целей доклада ими можно пренебречь
- На вашем железе цифры могут быть другие, это нормально

Конфигурация для всех примеров:

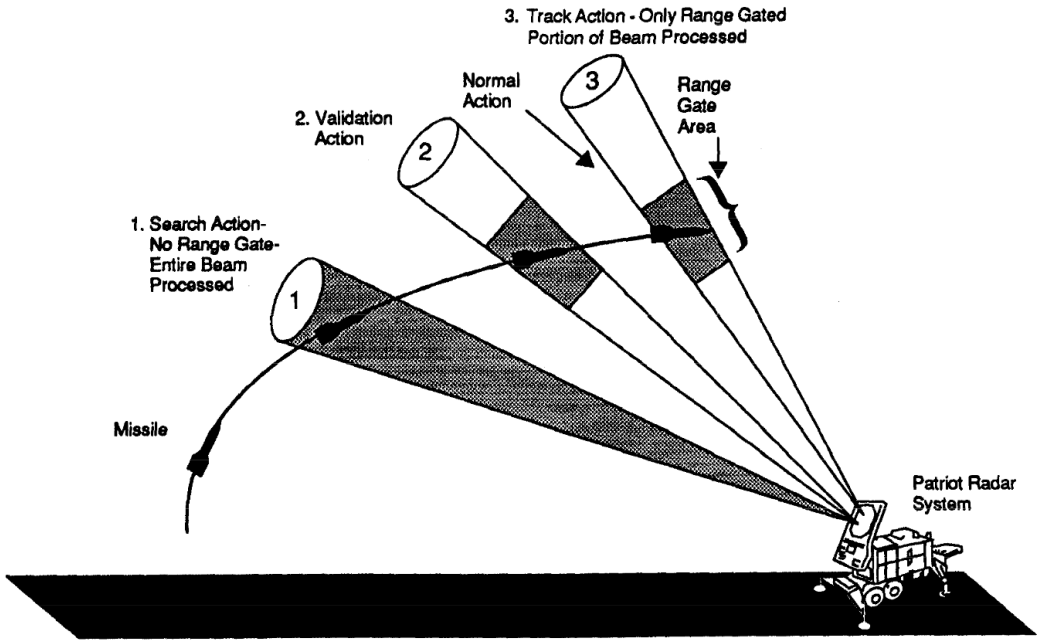
- BenchmarkDotNet v0.10.1 (supported by the .NET Foundation)
- Haswell Core i7-4702MQ CPU 2.20GHz, Windows 10
- .NET Framework 4.6.2 + clrjit/compatjit-v4.6.1586.0
- Mono 4.6.2 (x86/x64)
- Gist с исходниками: https://git.io/v1lUV

Quake 3 and Fast Inverse Square Root

Quake 3 and Fast Inverse Square Root

```
float Q_rsqrt(float number) { // 1/√number
  long i; float x2, y;
  const float threehalfs = 1.5F;
  x2 = number * 0.5F; y = number;
  i = *(long*)&y; // evil floating point bit level hacking
  i = 0x5f3759df - (i >> 1); // what the fuck?
  y = *(float*)&i;
  y = y * (threehalfs - (x2 * y * y)); // 1st iteration
  return y;
}
```

The Patriot Missile Failure



- Война в Персидском заливе, 25 февраля 1991
- Ракета Patriot не смогла сбить иракский снаряд Scud из-за бага
- Скорость Scud: $\approx$ 3750 миль в час
- Время хранилось в 24-битном регистре с точностью до 0.1 сек
- Ошибка на 1 тик составляет $\approx 9.5 \cdot 10^{-8}$ сек
- Суммарная ошибка за 100 часов: 0.3433 сек
- Итоговая погрешность $>$ **0.5 км**
- 28 человек убиты, 98 ранены, см. также GAO/IMTEC-92-26

Vancouver Stock Exchange

Vancouver Stock Exchange

- Январь 1982: введён индекс = 1000

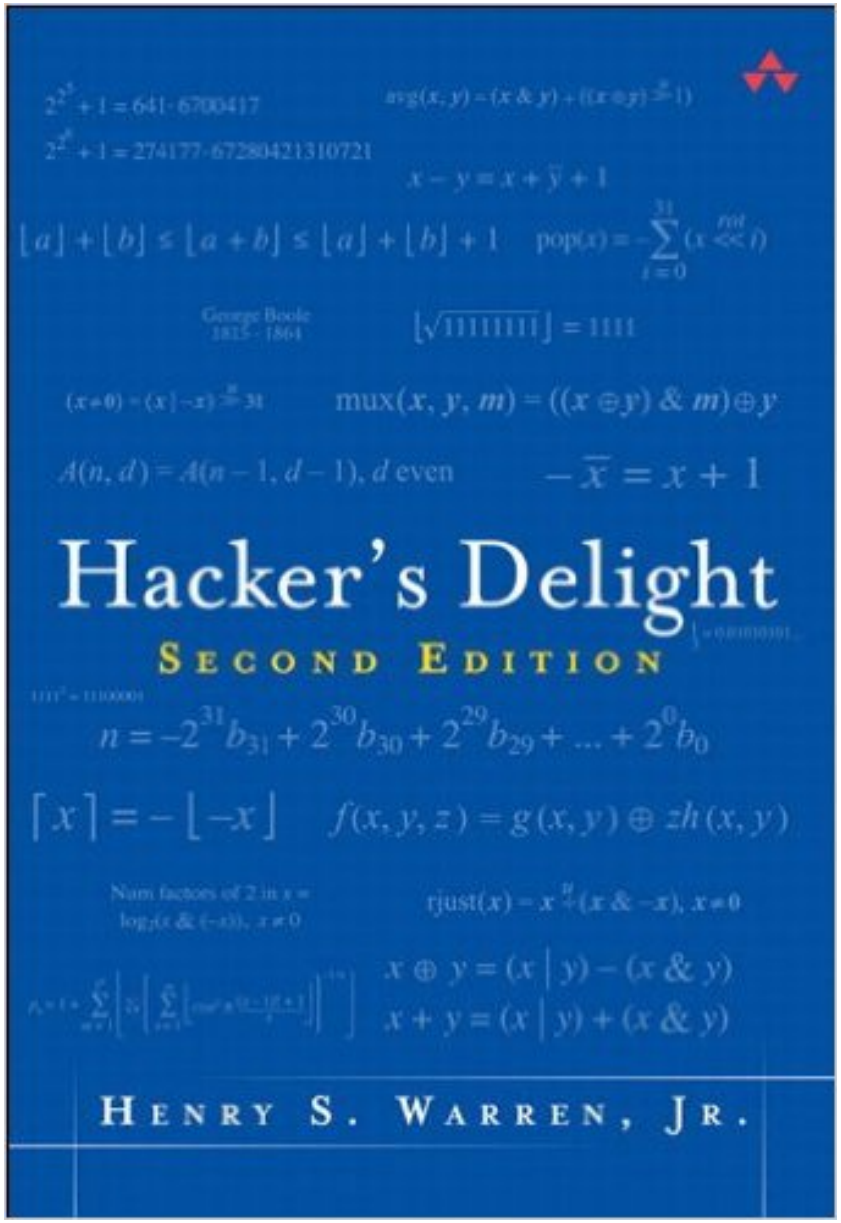
Vancouver Stock Exchange

- Январь 1982: введён индекс = 1000
- После каждой транзакции индекс обновлялся и округлялся до 3 знаков после точки (≈2800 раз в день)

Vancouver Stock Exchange

- Январь 1982: введён индекс = 1000
- После каждой транзакции индекс обновлялся и округлялся до 3 знаков после точки (≈2800 раз в день)
- Ноябрь 1983: ребята скорректировали ошибки
  Изменение индекса: **524.811** → **1098.892**

Поговорим про битовые хаки



Ищем минимум

Задача. Дано два числа `a` и `b` типа `int`. Необходимо реализовать функцию `Min`, которая вернёт наименьшее из этих двух чисел.

Ищем минимум

Задача. Дано два числа a и b типа `int`. Необходимо реализовать функцию `Min`, которая вернёт наименьшее из этих двух чисел.

Решение A.

```
int MinA(int x, int y) => Math.Min(x, y);
```

Ищем минимум

Задача. Дано два числа a и b типа `int`. Необходимо реализовать функцию `Min`, которая вернёт наименьшее из этих двух чисел.

Решение A.

```
int MinA(int x, int y) => Math.Min(x, y);
```

Решение B.

```
int MinB(int x, int y) => x < y ? x : y;
```

Ищем минимум

Задача. Дано два числа a и b типа `int`. Необходимо реализовать функцию `Min`, которая вернёт наименьшее из этих двух чисел.

Решение A.

```
int MinA(int x, int y) => Math.Min(x, y);
```

Решение B.

```
int MinB(int x, int y) => x < y ? x : y;
```

Решение C.

```
int MinC(int x, int y) =>
  x & ( (x - y) >> 31) |
  y & (~(x - y) >> 31);
```

(Объяснение) Ищем минимум

```
int MinC(int x, int y) =>
  x & ( (x - y) >> 31) |
  y & (~(x - y) >> 31);
```

x = 8	0	0	...	1	0	0	0
y = 3	0	0	...	0	0	1	1
x-y	0	0	...	0	1	0	1
(x-y)>>31	0	0	...	0	0	0	0
~(x-y)>>31	1	1	...	1	1	1	1
x&((x-y)>>31)	0	0	...	0	0	0	0
y&(~(x-y)>>31)	0	0	...	0	0	1	1
Result	0	0	...	0	0	1	1

(Бенчмарки) Ищем минимум

```
const int N = 100001; int[] a, b, c; // int[N] arrays
[Benchmark] void MinB() {
    for (int i = 0; i < N; i++) {
        int x = a[i], y = b[i];
        c[i] = x < y ? x : y;
    }
}
[Benchmark] void MinC() {
    for (int i = 0; i < N; i++) {
        int x = a[i], y = b[i];
        c[i] = x & ((x - y) >> 31) | y & (~(x - y) >> 31);
    }
}
```

(Бенчмарки) Ищем минимум

```
const int N = 100001; int[] a, b, c; // int[N] arrays
[Benchmark] void MinB() {
    for (int i = 0; i < N; i++) {
        int x = a[i], y = b[i];
        c[i] = x < y ? x : y;
    }
}
[Benchmark] void MinC() {
    for (int i = 0; i < N; i++) {
        int x = a[i], y = b[i];
        c[i] = x & ((x - y) >> 31) | y & (~(x - y) >> 31);
    }
}
```

	Random		Const	
	MinB	MinC	MinB	MinC
LegacyJIT-x86	$\approx 782\mu s$	$\approx 292\mu s$	$\approx 160\mu s$	$\approx 224\mu s$
LegacyJIT-x64	$\approx 525\mu s$	$\approx 157\mu s$	$\approx 71\mu s$	$\approx 124\mu s$
RyuJIT-x64	$\approx 602\mu s$	$\approx 248\mu s$	$\approx 221\mu s$	$\approx 292\mu s$
Mono4.6.2-x64	$\approx 255\mu s$	$\approx 349\mu s$	$\approx 212\mu s$	$\approx 290\mu s$

(ASM) Ищем минимум

```
; RyuJIT-x64[1]
cmp         ecx,edx
jl          LESS
mov         eax,edx
ret
LESS:
mov         eax,ecx
ret
```

[1] Roslyn 1.3.1.60616, /o+

(ASM) Ищем минимум

```
; RyuJIT-x64[1]
cmp         ecx,edx
jl          LESS
mov         eax,edx
ret
LESS:
mov         eax,ecx
ret
```

```
; Mono4.6.2-x64
sub     $0x18,%rsp
mov     %rsi,(%rsp)
mov     %rdi,0x8(%rsp)
mov     %rcx,%rdi
mov     %rdx,%rsi
cmp     %esi,%edi
mov     %rsi,%rax
cmovl   %rdi,%rax
mov     (%rsp),%rsi
mov     0x8(%rsp),%rdi
add     $0x18,%rsp
retq
```

cmovl (0F 4C): Conditional move (if less)

[1] Roslyn 1.3.1.60616, /o+

Делим на константу

Задача. Дано число `n` типа `int`. Необходимо реализовать функцию `Div3`, которая делит заданное число на 3.

Делим на константу

Задача. Дано число `n` типа `int`. Необходимо реализовать функцию `Div3`, которая делит заданное число на 3.

Решение А.

```
int Div3A(int n) =>
    n / 3;
```

Делим на константу

Задача. Дано число `n` типа `int`. Необходимо реализовать функцию `Div3`, которая делит заданное число на 3.

Решение A.

```
int Div3A(int n) =>
    n / 3;
```

Решение B.

```
int Div3B(int n) =>
    (int)((n * 0xAAAAAAAB) >> 33);
```

Делим на константу

Задача. Дано число `n` типа `int`. Необходимо реализовать функцию `Div3`, которая делит заданное число на 3.

Решение A.

```
int Div3A(int n) =>
    n / 3;
```

Решение B.

```
int Div3B(int n) =>
    (int)((n * 0xAAAAAAAB) >> 33);
```

$$\texttt{0xAAAAAAAB}_{16} = \frac{2^{33}}{3} + 1$$

(Бенчмарки) Делим на константу

```
int N = 100001; int[] a, b; // random arrays
[Benchmark] void Div3A() {
    for (int i = 0; i < N; i++)
        b[i] = a[i] / 3;
}
[Benchmark] void Div3B() {
    for (int i = 0; i < N; i++)
        b[i] = (int)((a[i] * 0xAAAAAAAB) >> 33);
}
```

(Бенчмарки) Делим на константу

```
int N = 100001; int[] a, b; // random arrays
[Benchmark] void Div3A() {
    for (int i = 0; i < N; i++)
        b[i] = a[i] / 3;
}
[Benchmark] void Div3B() {
    for (int i = 0; i < N; i++)
        b[i] = (int)((a[i] * 0xAAAAAAAB) >> 33);
}
```

	Div3A	Div3B
LegacyJIT-x86	$\approx 376\mu s$	$\approx 334\mu s$
LegacyJIT-x64	$\approx 122\mu s$	$\approx 67\mu s$
RyuJIT-x64	$\approx 173\mu s$	$\approx 140\mu s$
Mono4.6.2-x86	$\approx 366\mu s$	$\approx 1\,139\mu s$
Mono4.6.2-x64	$\approx 446\mu s$	$\approx 176\mu s$

(Объяснение) Делим на константу

```
long Mul(int x, uint y) => x * y;
```

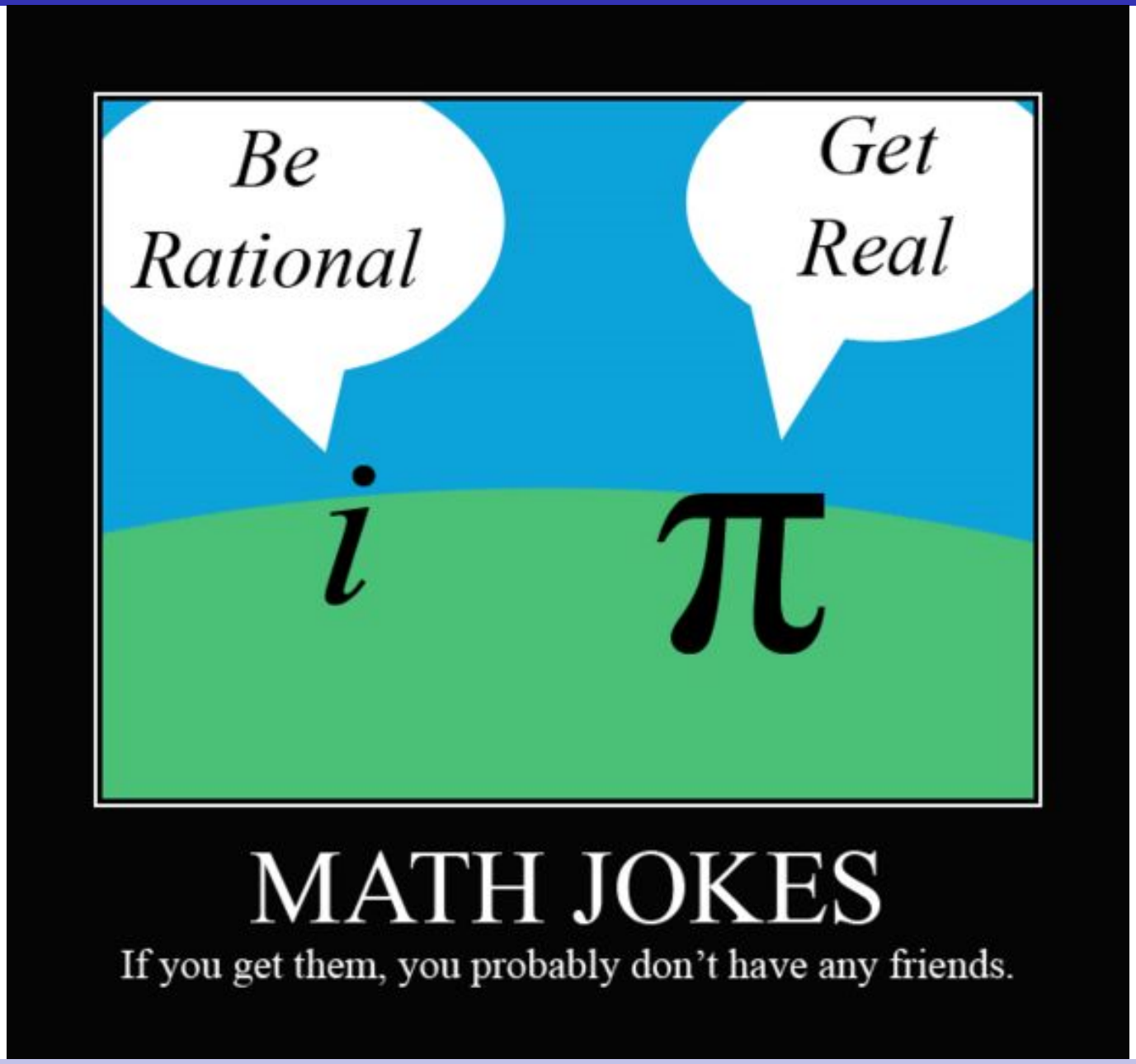
(Объяснение) Делим на константу

```
long Mul(int x, uint y) => x * y;
```

```
; Mono-x86
gram_Mul:
push   %rbp
mov    %esp,%ebp
push   %rdi
sub    $0x34,%esp
cmpl   $0xffffffff,0x8(%rbp)
setg   %al
movzbl %al,%eax
mov    0x8(%rbp),%rcx
mov    %ecx,-0x10(%rbp)
mov    %eax,-0xc(%rbp)
mov    0xc(%rbp),%eax
mov    %eax,-0x18(%rbp)
movl   $0x0,-0x14(%rbp)
mov    -0x14(%rbp),%eax
mov    %eax,0xc(%rsp)
mov    -0x18(%rbp),%eax
mov    %eax,0x8(%rsp)
mov    -0xc(%rbp),%eax
mov    %eax,0x4(%rsp)
```

```
mov    -0x10(%rbp),%eax
mov    %eax,(%rsp)
callq  c2d080c gram_Mul+0xc2d080c
mov    %edx,-0xc(%rbp)
mov    %eax,-0x10(%rbp)
mov    0x102ebc54(%rip),%eax
mov    -0x10(%rbp),%ecx
mov    %ecx,-0x18(%rbp)
mov    -0xc(%rbp),%ecx
mov    %ecx,-0x14(%rbp)
test   %eax,%eax
jne    gram_Mul+0xa8
jmp    gram_Mul+0x72
mov    -0x20(%rbp),%eax
mov    %eax,-0x18(%rbp)
mov    -0x1c(%rbp),%eax
mov    %eax,-0x14(%rbp)
mov    -0x18(%rbp),%eax
mov    %eax,-0x10(%rbp)
mov    -0x14(%rbp),%eax
; И ещё куча строк...
```

Нецелые числа — это сложно



Обычная арифметика не работает

$$(a + b) + c \neq a + (b + c)$$

$$(a \cdot b) \cdot c \neq a \cdot (b \cdot c)$$

$$(a + b) \cdot c \neq a \cdot c + b \cdot c$$

$$a^{x+y} \neq a^x \cdot a^y$$

$$\ldots$$

Обычная арифметика не работает

Жить сложно:

$$0.1d + 0.2d \neq 0.3d$$

$$\begin{array}{rcl}
0.1\text{d} & \approx & 0.10000000000000000555111512312578 3 \\
+0.2\text{d} & \approx & 0.20000000000000001110223024625156 5 \\
\hline
 & & 0.30000000000000004440892098500626 2 \\
0.3\text{d} & \approx & 0.29999999999999998889776975374843 5
\end{array}$$

Обычная арифметика не работает

Жить сложно:

$$0.1d + 0.2d \neq 0.3d$$

$$\begin{array}{rcl} 0.1\text{d} & \approx & 0.10000000000000000555111512312578 3 \\ +0.2\text{d} & \approx & 0.20000000000000001110223024625156 5 \\ \hline & & 0.30000000000000004440892098500626 2 \\ 0.3\text{d} & \approx & 0.29999999999999998889776975374843 5 \end{array}$$

Хорошая практика:

```
bool AreEqual(double a, double b,
              double eps = 1e-9)
{
    return Math.Abs(a - b) < eps;
}
```

Поговорим про конвертацию

Задача. Что выведет следующий код?

```
double d1  = 0.84551240822557006;
string str = d1.ToString("R");[1]
double d2  = double.Parse(str);
WriteLine(d1 == d2);
```

Поговорим про конвертацию

Задача. Что выведет следующий код?

```
double d1  = 0.84551240822557006;
string str = d1.ToString("R");[1]
double d2  = double.Parse(str);
WriteLine(d1 == d2);
```

MSDN: Standard Numeric Format Strings

[1] The round-trip ("R") format specifier is used to ensure that a numeric value that is converted to a string will be parsed back into the same numeric value.

Поговорим про конвертацию

Задача. Что выведет следующий код?

```
double d1  = 0.84551240822557006;
string str = d1.ToString("R");[1]
double d2  = double.Parse(str);
WriteLine(d1 == d2);
```

MSDN: Standard Numeric Format Strings

[1] The round-trip ("R") format specifier is used to ensure that a numeric value that is converted to a string will be parsed back into the same numeric value.

.NET-x64	.NET-x86	Mono
false (баг)	true	true

Загадка на производительность

Вопрос. Какая программа работает намного медленнее остальных под LegacyJIT-x86?

```
const int N = 100001;
public double Sum()
{
  double x = 1, y = 1;

  for (int i = 0; i < N; i++)
    x = x + y;

  return x;
}
// A
Sum();
// B
Write(Stopwatch.Frequency);
Sum();
```

Загадка на производительность

Вопрос. Какая программа работает намного медленнее остальных под LegacyJIT-x86?

```
const int N = 100001;
public double Sum()
{
  double x = 1, y = 1;

  for (int i = 0; i < N; i++)
    x = x + y;

  return x;
}
// A
Sum();
// B
Write(Stopwatch.Frequency);
Sum();
```

```
const int N = 100001;
public double Sum2()
{
  double x = 1, y = 1;
  var sw = Stopwatch.StartNew();
  for (int i = 0; i < N; i++)
    x = x + y;
  sw.Stop();
  return x;
}
// C
Sum2();
// D
Write(Stopwatch.Frequency);
Sum2();
```

Загадка на производительность

Вопрос. Какая программа работает намного медленнее остальных под LegacyJIT-x86?

```
const int N = 100001;
public double Sum()
{
  double x = 1, y = 1;

  for (int i = 0; i < N; i++)
    x = x + y;

  return x;
}
// A
Sum();
// B
Write(Stopwatch.Frequency);
Sum();
```

```
const int N = 100001;
public double Sum2()
{
  double x = 1, y = 1;
  var sw = Stopwatch.StartNew();
  for (int i = 0; i < N; i++)
    x = x + y;
  sw.Stop();
  return x;
}
// C
Sum2();
// D
Write(Stopwatch.Frequency);
Sum2();
```

Ответ. Программа C:

	A	B	C	D
LegacyJIT-x86	$\approx 100\mu s$	$\approx 100\mu s$	$\approx 335\mu s$	$\approx 100\mu s$

Смотрим ASM

```
// D
// Статический конструктор класса
// Stopwatch вызывается при
// обращении к Stopwatch.Frequency
Write(Stopwatch.Frequency);

// Поэтому нам не нужно
// вызывать его из метода.
Sum2();
```

Смотрим ASM

```
// D
// Статический конструктор класса
// Stopwatch вызывается при
// обращении к Stopwatch.Frequency
Write(Stopwatch.Frequency);

// Поэтому нам не нужно
// вызывать его из метода.
Sum2();
```

```
// C
// Статический конструктор класса
// Stopwatch ещё не вызывался.



// Поэтому нам придётся
// вызвать его из метода.
Sum2();
```

Смотрим ASM

```
// D
// Статический конструктор класса
// Stopwatch вызывается при
// обращении к Stopwatch.Frequency
Write(Stopwatch.Frequency);

// Поэтому нам не нужно
// вызывать его из метода.
Sum2();
```

```
// C
// Статический конструктор класса
// Stopwatch ещё не вызывался.



// Поэтому нам придётся
// вызвать его из метода.
Sum2();
```

```
; x + y (A, B, D)
fld1
faddp   st(1),st
```

```
; x + y (C)
fld1
fadd    qword ptr [ebp-0Ch]
fstp    qword ptr [ebp-0Ch]
```

Browse Standards > IEEE Std 754-2008 ...

754-2008 - IEEE Standard for Floating-Point Arithmetic

Full Text
Sign-In or Purchase

Revision of ANSI/IEEE Std 754-1985

Status: **Active** - **Approved**

Abstract | References | Versions | Definitions | Cited By | Keywords

Download Citations
Email
Print
Request Permissions
Export

This standard specifies interchange and arithmetic formats and methods for binary and decimal floating-point arithmetic in computer programming environments. This standard specifies exception conditions and their default handling. An implementation of a floating-point system conforming to this standard may be realized entirely in software, entirely in hardware, or in any combination of software and hardware. For operations specified in the normative part of this standard, numerical results and exceptions are uniquely determined by the values of the input data, sequence of operations, and destination formats, all under user control.

Date of Publication :
Aug. 29 2008

Status :
Active

INSPEC Accession Number:
10369888

DOI:
10.1109/IEEESTD.2008.4610935

Нормализованные числа

S	Exponent	Mantissa

$$V = (-1)^S \cdot \mathbf{1}.M \cdot 2^{E-E_{bias}}, \quad E_{min} \leq E \leq E_{max}$$

Нормализованные числа

S	Exponent	Mantissa

$$V = (-1)^S \cdot \mathbf{1}.M \cdot 2^{E-E_{bias}}, \quad E_{min} \leq E \leq E_{max}$$

	Sign	Exponent	Mantissa	E_{bias}
`float`	1	8	23	127
`double`	1	11	52	1023
80`bit`	1	15	64	16383

Нормализованные числа

S	Exponent	Mantissa

$$V = (-1)^S \cdot \mathbf{1}.M \cdot 2^{E-E_{bias}}, \quad E_{min} \le E \le E_{max}$$

	Sign	Exponent	Mantissa	E_{bias}
`float`	1	8	23	127
`double`	1	11	52	1023
`80bit`	1	15	64	16383

	Digits	Lower	Upper
`float`	≈ 7.2	$1.2 \cdot 10^{-38}$	$3.4 \cdot 10^{+38}$
`double`	≈ 15.9	$2.3 \cdot 10^{-308}$	$1.7 \cdot 10^{+308}$
`80bit`	≈ 19.2	$3.4 \cdot 10^{-4932}$	$1.1 \cdot 10^{+4932}$

Пример

0	1	0	0	1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	1	0	1	1	0	0	1	0	1	0	0	1

$$S = 0$$

$$E = 10011100_2 = 156_{10}$$

$$M = 1.11011100110101100101001_2 = 15\,625\,001 \cdot 2^{-23}$$

$$V = (-1)^0 \cdot \left(15\,625\,001 \cdot 2^{-23}\right) \cdot 2^{156-127}$$

$$= 15\,625\,001 \cdot 2^6 = 1\,000\,000\,064$$

Пример

0	1	0	0	1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	1	0	1	1	0	0	1	0	1	0	0	1

$$S = 0$$

$$E = 10011100_2 = 156_{10}$$

$$M = 1.11011100110101100101001_2 = 15\,625\,001 \cdot 2^{-23}$$

$$V = (-1)^0 \cdot \left(15\,625\,001 \cdot 2^{-23}\right) \cdot 2^{156-127}$$

$$= 15\,625\,001 \cdot 2^6 = 1\,000\,000\,064$$

Задача. Чему равно значение выражения?

```
((float)1000000064).ToString("G10")
```

Пример

0	1	0	0	1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	1	0	1	1	0	0	1	0	1	0	0	1

$$S = 0$$

$$E = 10011100_2 = 156_{10}$$

$$M = 1.11011100110101100101001_2 = 15\,625\,001 \cdot 2^{-23}$$

$$V = (-1)^0 \cdot \left(15\,625\,001 \cdot 2^{-23}\right) \cdot 2^{156-127}$$

$$= 15\,625\,001 \cdot 2^6 = 1\,000\,000\,064$$

Задача. Чему равно значение выражения?

```
((float)1000000064).ToString("G10")
```

Решение.

```
1000000060 // 9 значащих цифр
```

Денормализованные числа

S	0	0	0	0	0	0	0	0	Mantissa

$$V = (-1)^S \cdot \mathbf{0}.M \cdot 2^{E_{denorm}}, \quad E_{denorm} = E_{bias} - 1$$

Денормализованные числа

S	0	0	0	0	0	0	0	0	Mantissa

$$V = (-1)^S \cdot \mathbf{0}.M \cdot 2^{E_{denorm}}, \quad E_{denorm} = E_{bias} - 1$$

	E_{denorm}	Lower	Upper
`float`	-126	$1.4 \cdot 10^{-45}$	$1.2 \cdot 10^{-38}$
`double`	-1022	$5.0 \cdot 10^{-324}$	$2.3 \cdot 10^{-308}$
80`bit`	-16382	$1.9 \cdot 10^{-4951}$	$3.4 \cdot 10^{-4932}$

Денормализованные числа

S	0	0	0	0	0	0	0	0	Mantissa

$$V = (-1)^S \cdot \mathbf{0}.M \cdot 2^{E_{denorm}}, \quad E_{denorm} = E_{bias} - 1$$

	E_{denorm}	Lower	Upper
`float`	-126	$1.4 \cdot 10^{-45}$	$1.2 \cdot 10^{-38}$
`double`	-1022	$5.0 \cdot 10^{-324}$	$2.3 \cdot 10^{-308}$
`80bit`	-16382	$1.9 \cdot 10^{-4951}$	$3.4 \cdot 10^{-4932}$

- Хорошие новости: $a \neq b \Rightarrow a - b \neq 0$

Денормализованные числа

S	0	0	0	0	0	0	0	0	Mantissa

$$V = (-1)^S \cdot \mathbf{0}.M \cdot 2^{E_{denorm}}, \quad E_{denorm} = E_{bias} - 1$$

	E_{denorm}	Lower	Upper
`float`	-126	$1.4 \cdot 10^{-45}$	$1.2 \cdot 10^{-38}$
`double`	-1022	$5.0 \cdot 10^{-324}$	$2.3 \cdot 10^{-308}$
80`bit`	-16382	$1.9 \cdot 10^{-4951}$	$3.4 \cdot 10^{-4932}$

- Хорошие новости: $a \neq b \Rightarrow a - b \neq 0$
- Плохие новости: денормализованные числа работают медленно ☹

(Задачка) Денормализованные числа

Задача. Подсчитать $0.96^{100\,000}$, $0.96^{1\,000\,000}$, используя `double`.

(Задачка) Денормализованные числа

Задача. Подсчитать $0.96^{100\,000}$, $0.96^{1\,000\,000}$, используя `double`.

Решение A.

```
public double PowerA() {
 double res = 1.0;
 for (int i = 0; i < N; i++)
   res = res * 0.96;
 return res;
}
```

(Задачка) Денормализованные числа

Задача. Подсчитать $0.96^{100\,000}$, $0.96^{1\,000\,000}$, используя `double`.

Решение A.

```
public double PowerA() {
 double res = 1.0;
 for (int i = 0; i < N; i++)
   res = res * 0.96;
 return res;
}
```

Решение B.

```
private double resB;
public double PowerB() {
 resB = 1.0;
 for (int i = 0; i < N; i++)
   resB = resB * 0.96;
 return resB;
}
```

(Задачка) Денормализованные числа

Задача. Подсчитать $0.96^{100\,000}$, $0.96^{1\,000\,000}$, используя `double`.

Решение A.

```
public double PowerA() {
 double res = 1.0;
 for (int i = 0; i < N; i++)
   res = res * 0.96;
 return res;
}
```

Решение B.

```
private double resB;
public double PowerB() {
 resB = 1.0;
 for (int i = 0; i < N; i++)
   resB = resB * 0.96;
 return resB;
}
```

Решение C.

```
public double PowerC() {
 double res = 1.0;
 for (int i = 0; i < N; i++)
   res = res * 0.96
             + 0.1 - 0.1;
 return res;
}
```

```
[Params(100000, 1000000)]
public int N;
```

(Бенчмарки) Денормализованные числа

	N	PowerA	PowerB	PowerC
LegacyJIT-x86	10^5	$\approx 168\mu s$	$\approx 19\,264\mu s$	$\approx 370\mu s$
RyuJIT-x64	10^5	$\approx 3\,961\mu s$	$\approx 4\,142\mu s$	$\approx 370\mu s$
LegacyJIT-x86	10^6	$\approx 152\,770\mu s$	$\approx 227\,781\mu s$	$\approx 3\,851\mu s$
RyuJIT-x64	10^6	$\approx 47\,956\mu s$	$\approx 49\,670\mu s$	$\approx 3\,782\mu s$

(Бенчмарки) Денормализованные числа

	N	PowerA	PowerB	PowerC
LegacyJIT-x86	10^5	$\approx 168\mu s$	$\approx 19\,264\mu s$	$\approx 370\mu s$
RyuJIT-x64	10^5	$\approx 3\,961\mu s$	$\approx 4\,142\mu s$	$\approx 370\mu s$
LegacyJIT-x86	10^6	$\approx 152\,770\mu s$	$\approx 227\,781\mu s$	$\approx 3\,851\mu s$
RyuJIT-x64	10^6	$\approx 47\,956\mu s$	$\approx 49\,670\mu s$	$\approx 3\,782\mu s$

Логичные вопросы:

- Почему `PowerC` работает так быстро?
- Почему `PowerA` работает очень быстро, но только на LegacyJIT-x86 при $N = 10^5$?

(Бенчмарки) Денормализованные числа

	N	PowerA	PowerB	PowerC
LegacyJIT-x86	10^5	$\approx 168\mu s$	$\approx 19\,264\mu s$	$\approx 370\mu s$
RyuJIT-x64	10^5	$\approx 3\,961\mu s$	$\approx 4\,142\mu s$	$\approx 370\mu s$
LegacyJIT-x86	10^6	$\approx 152\,770\mu s$	$\approx 227\,781\mu s$	$\approx 3\,851\mu s$
RyuJIT-x64	10^6	$\approx 47\,956\mu s$	$\approx 49\,670\mu s$	$\approx 3\,782\mu s$

Логичные вопросы:

- Почему `PowerC` работает так быстро?
- Почему `PowerA` работает очень быстро, но только на LegacyJIT-x86 при $N = 10^5$?

Рассмотрим два случая:

- LegacyJIT-x64, Mono-x64 **(SSE)**; RyuJIT-x64 **(AVX)**
- LegacyJIT-x86 **(x87 FPU)**

(SSE/AVX) Денормализованные числа

i	PowerA	PowerC
0	1.0 3FF0000000000000	1.0 3FF0000000000000

(SSE/AVX) Денормализованные числа

i	PowerA	PowerC
0	1.0 3FF0000000000000	1.0 3FF0000000000000
1	0.96 3FEEB851EB851EB8	0.96000000000000008 3FEEB851EB851EB9

(SSE/AVX) Денормализованные числа

i	PowerA	PowerC
0	1.0 3FF0000000000000	1.0 3FF0000000000000
1	0.96 3FEEB851EB851EB8	0.96000000000000008 3FEEB851EB851EB9
885	2.0419318345555615E-16 3CAD6D6617566397	1.8041124150158794E-16 3CAA000000000000
886	1.9602545611733389E-16 3CAC4010166769D8	1.6653345369377348E-16 3CA8000000000000
887	1.8818443787264053E-16 3CAB1EC7C3967A17	1.6653345369377348E-16 3CA8000000000000

(SSE/AVX) Денормализованные числа

i	PowerA	PowerC
0	1.0 3FF0000000000000	1.0 3FF0000000000000
1	0.96 3FEEB851EB851EB8	0.96000000000000008 3FEEB851EB851EB9
885	2.0419318345555615E-16 3CAD6D6617566397	1.8041124150158794E-16 3CAA000000000000
886	1.9602545611733389E-16 3CAC4010166769D8	1.6653345369377348E-16 3CA8000000000000
887	1.8818443787264053E-16 3CAB1EC7C3967A17	1.6653345369377348E-16 3CA8000000000000
18171	6.42285339593621E-323 000000000000000D	1.6653345369377348E-16 3CA8000000000000
18172	5.92878775009496E-323 000000000000000C	1.6653345369377348E-16 3CA8000000000000
18173	5.92878775009496E-323 000000000000000C	1.6653345369377348E-16 3CA8000000000000

(x87 FPU) Денормализованные числа

```
; PowerA (10^5 : ≈ 167μs          10^6 : ≈ 151 947μs)
fld     qword ptr ds:[14D2E28h] ; 0.96
fmulp   st(1),st                ; Значение в регистре
; PowerB (10^5 : ≈ 19 079μs      10^6 : ≈ 226 219μs)
fld     qword ptr ds:[892E20h]  ; 0.96
fmul    qword ptr [ecx+4]
fstp    qword ptr [ecx+4]       ; Значение в памяти
```

(x87 FPU) Денормализованные числа

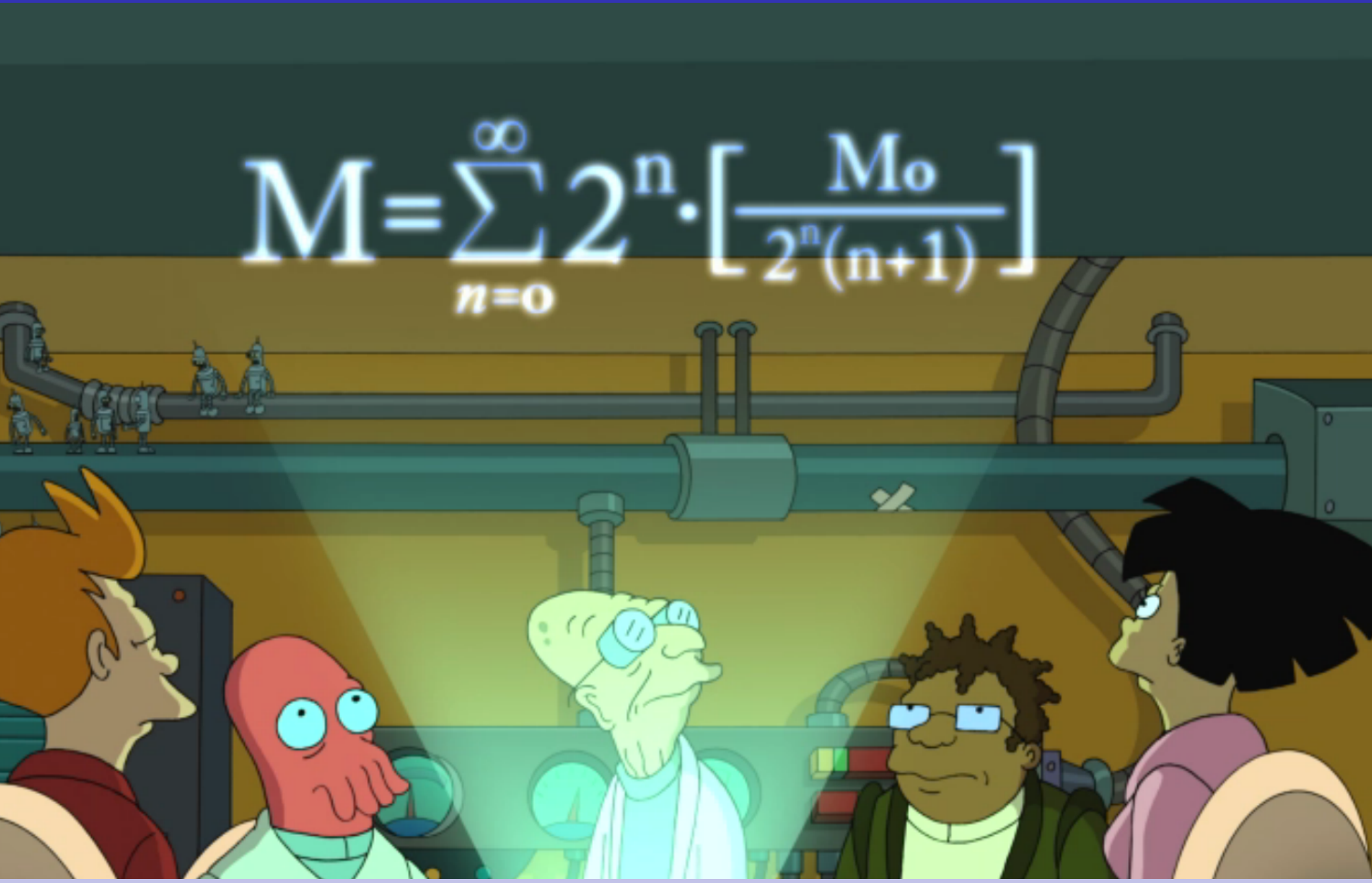
```
; PowerA (10^5 : ≈ 167μs          10^6 : ≈ 151 947μs)
fld     qword ptr ds:[14D2E28h]  ; 0.96
fmulp   st(1),st                 ; Значение в регистре
; PowerB (10^5 : ≈ 19 079μs      10^6 : ≈ 226 219μs)
fld     qword ptr ds:[892E20h]   ; 0.96
fmul    qword ptr [ecx+4]
fstp    qword ptr [ecx+4]        ; Значение в памяти
```

Intel® 64 and IA-32 Architectures Software Developer's Manual §8.2 X87 FPU Data Types

With the exception of the 80-bit double extended-precision format, all of data types exist in memory only. When they are loaded into x87 FPU data registers, they are converted into double extended-precision format and operated on in that format.

When a denormal number is used as a source operand, the x87 FPU automatically normalizes the number when it is converted to double extended-precision format.

Поговорим про суммирование



Сумма чисел

Задача. Дан `List<double>`. Необходимо реализовать функцию `Sum`, которая вернёт сумму элементов этого списка.

(Проблема) Сумма чисел

Имеется следующая проблема[1]:

$$10^{16} + 1 = 10^{16}$$

[1] Научное название: absorption

(Проблема) Сумма чисел

Имеется следующая проблема[1]:

$$10^{16} + 1 = 10^{16}$$

Вспоминаем мат. часть:

	Digits	Lower	Upper
`double`	$\approx \mathbf{15.9}$	$2.3 \cdot 10^{-308}$	$1.7 \cdot 10^{+308}$

[1] Научное название: absorption

(Алгоритм Кэхэна) Сумма чисел

```
double KahanSum(this List<double> list)
{
    double sum = 0, error = 0;
    foreach (var x in list)
    {
        var y = x - error;
        var newSum = sum + y;
        error = (newSum - sum) - y;
        sum = newSum;
    }
    return sum;
}
```

(Пример) Сумма чисел

```
var a = new List<double>()
```

$$\{\ 10^{16}, \underbrace{1, 1, \ldots, 1}_{100\ \text{раз}}\ \};$$

```
WriteLine("{0:N}", a.Sum());
WriteLine("{0:N}", a.KahanSum());

// 10,000,000,000,000,000.00
// 10,000,000,000,000,100.00
```

(Пример) Сумма чисел

```
var a = new List<double>()
  { 10^16, 1, 1, ..., 1 };
```

$$\{\ 10^{16}, \underbrace{1, 1, \ldots, 1}_{100\ \text{раз}}\ \};$$

```
WriteLine("{0:N}", a.Sum());
WriteLine("{0:N}", a.KahanSum());

// 10,000,000,000,000,000.00
// 10,000,000,000,000,100.00
```

Проблема возникает в самых разных местах:

- Банковская сфера
- Страховая сфера
- Численные алгоритмы
- Ситуации с операциями над числами разных порядков

(А теперь про скорость) Сумма чисел

Задача. Дан `double[]` длины `4000`. Необходимо реализовать функцию `Sum`, которая *быстро* вернёт сумму элементов массива.

(А теперь про скорость) Сумма чисел

Задача. Дан `double[]` длины 4000. Необходимо реализовать функцию `Sum`, которая *быстро* вернёт сумму элементов массива.

Решение А.

```
public double Bcl() => a.Sum();
```

(А теперь про скорость) Сумма чисел

Задача. Дан `double[]` длины 4000. Необходимо реализовать функцию `Sum`, которая *быстро* вернёт сумму элементов массива.

Решение A.

```
public double Bcl() => a.Sum();
```

Решение B.

```
public double Loop() {
    double sum = 0;
    for (int i = 0; i < a.Length; i++)
        sum += a[i];
    return sum;
}
```

(А теперь про скорость) Сумма чисел

Задача. Дан `double[]` длины 4000. Необходимо реализовать функцию Sum, которая *быстро* вернёт сумму элементов массива.

Решение A.

```
public double Bcl() => a.Sum();
```

Решение B.

```
public double Loop() {
    double sum = 0;
    for (int i = 0; i < a.Length; i++)
        sum += a[i];
    return sum;
}
```

Решение C.

```
public double Unroll() {
    double sum = 0;
    for (int i = 0; i < a.Length; i += 4)
        sum += a[i] + a[i + 1] + a[i + 2] + a[i + 3];
    return sum;
}
```

(Бенчмарки) Сумма чисел

```
const int N = 4000; double[] a; // random double[N] array
[Benchmark] double Bcl() => a.Sum();
[Benchmark] double Loop() {
    double sum = 0;
    for (int i = 0; i < a.Length; i++)
        sum += a[i];
    return sum;
}
[Benchmark] double Unroll() {
    double sum = 0;
    for (int i = 0; i < a.Length; i += 4)
        sum += a[i] + a[i + 1] + a[i + 2] + a[i + 3];
    return sum;
}
```

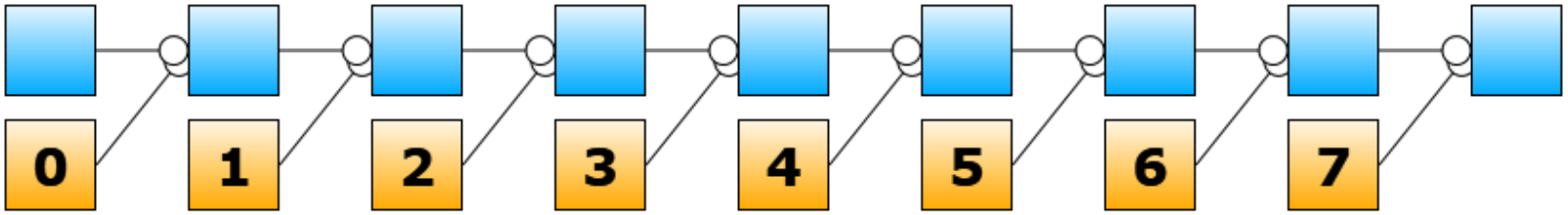
(Бенчмарки) Сумма чисел

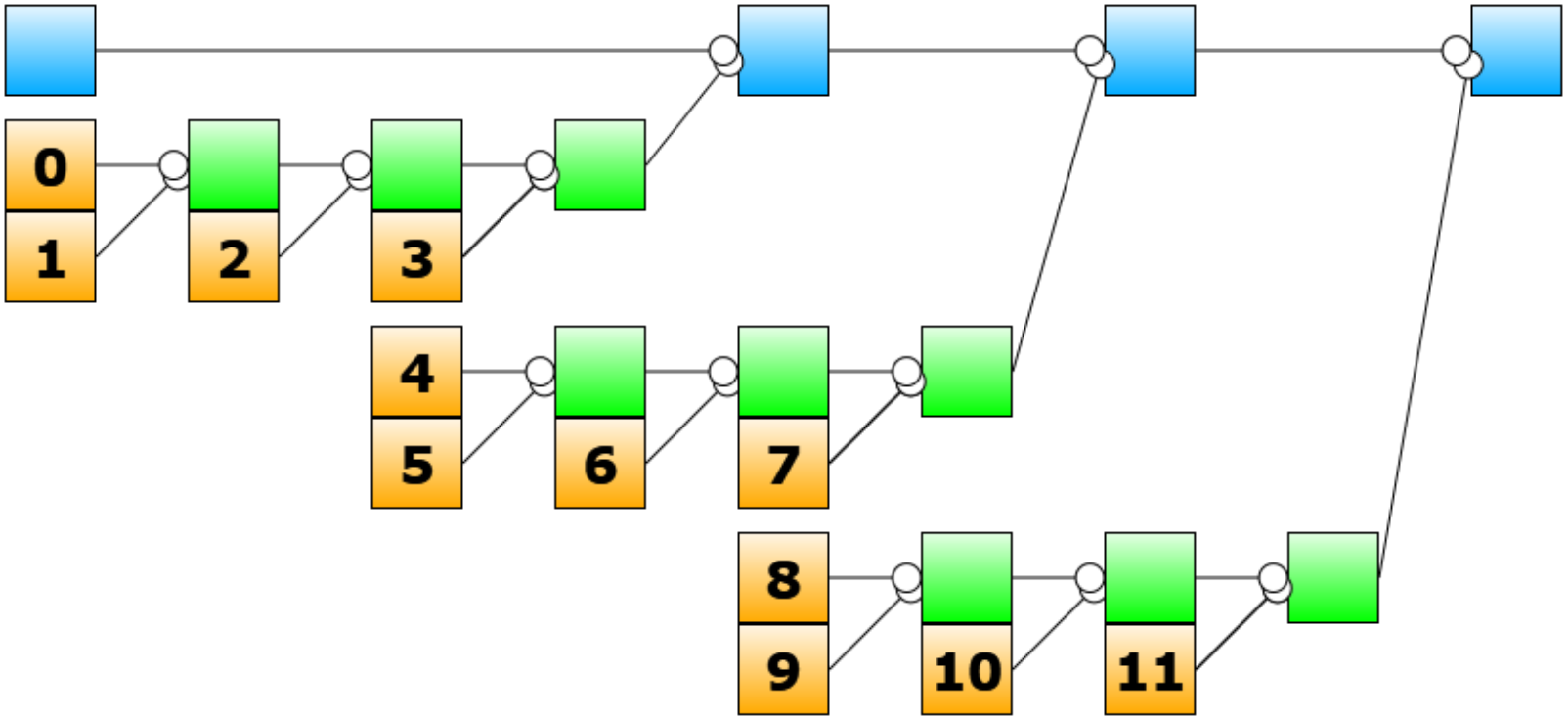
```
const int N = 4000; double[] a; // random double[N] array
[Benchmark] double Bcl() => a.Sum();
[Benchmark] double Loop() {
    double sum = 0;
    for (int i = 0; i < a.Length; i++)
        sum += a[i];
    return sum;
}
[Benchmark] double Unroll() {
    double sum = 0;
    for (int i = 0; i < a.Length; i += 4)
        sum += a[i] + a[i + 1] + a[i + 2] + a[i + 3];
    return sum;
}
```

	Bcl	Loop	Unroll
LegacyJIT-x86	$\approx 23.4\mu s$	$\approx 4.0\mu s$	$\approx 1.6\mu s$
LegacyJIT-x64	$\approx 28.3\mu s$	$\approx 4.0\mu s$	$\approx 1.8\mu s$
RyuJIT-x64	$\approx 31.1\mu s$	$\approx 4.2\mu s$	$\approx 2.4\mu s$
Mono4.6.2-x86	$\approx 72.7\mu s$	$\approx 14.3\mu s$	$\approx 2.2\mu s$
Mono4.6.2-x64	$\approx 54.7\mu s$	$\approx 12.3\mu s$	$\approx 3.6\mu s$

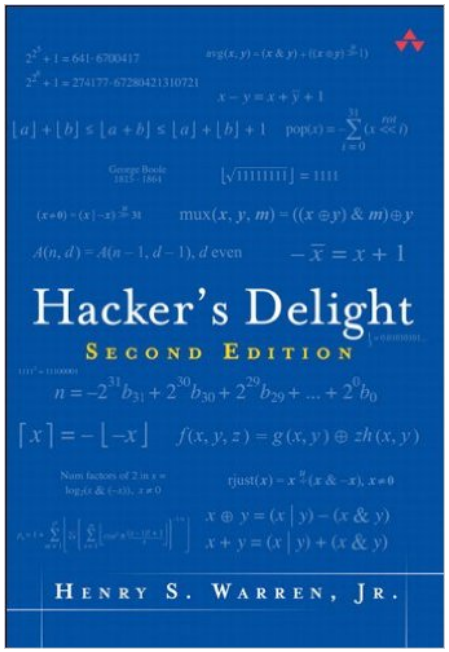
(Примерная схема) Сумма чисел

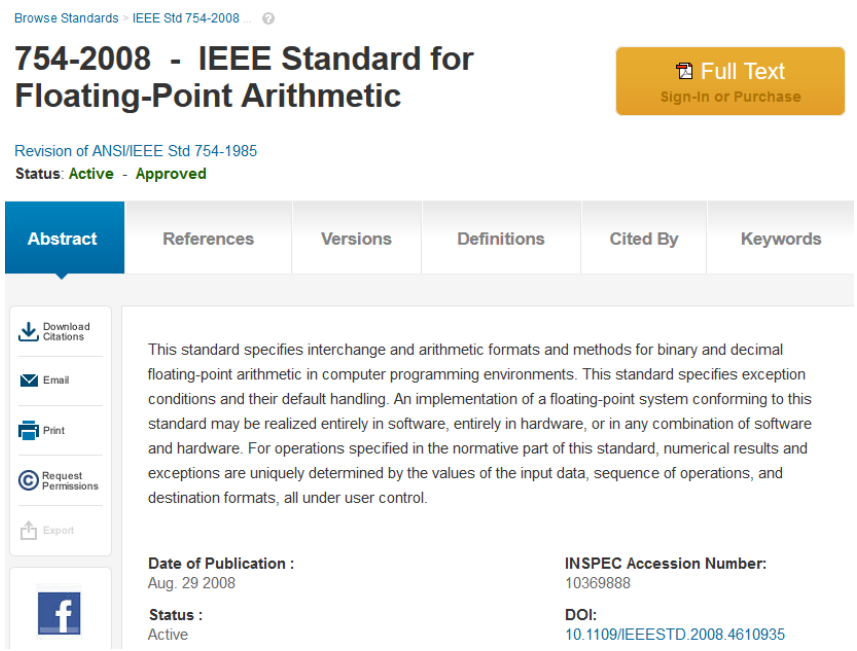
Instruction Level Parallelism спешит на помощь!





Методическая литература





Browse Standards > IEEE Std 754-2008 ...

754-2008 - IEEE Standard for Floating-Point Arithmetic

Full Text
Sign-In or Purchase

Revision of ANSI/IEEE Std 754-1985
Status: **Active** - **Approved**

Abstract | References | Versions | Definitions | Cited By | Keywords

Download Citations
Email
Print
Request Permissions
Export

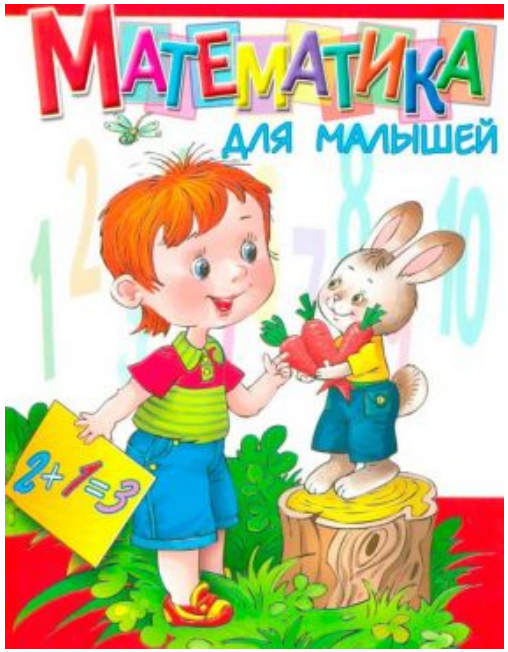
This standard specifies interchange and arithmetic formats and methods for binary and decimal floating-point arithmetic in computer programming environments. This standard specifies exception conditions and their default handling. An implementation of a floating-point system conforming to this standard may be realized entirely in software, entirely in hardware, or in any combination of software and hardware. For operations specified in the normative part of this standard, numerical results and exceptions are uniquely determined by the values of the input data, sequence of operations, and destination formats, all under user control.

Date of Publication :
Aug. 29 2008

Status :
Active

INSPEC Accession Number:
10369888

DOI:
10.1109/IEEESTD.2008.4610935



Вопросы?

Андрей Акиньшин, JetBrains
http://aakinshin.net
https://github.com/AndreyAkinshin
https://twitter.com/andrey_akinshin
andrey.akinshin@gmail.com