

# Лекция 3

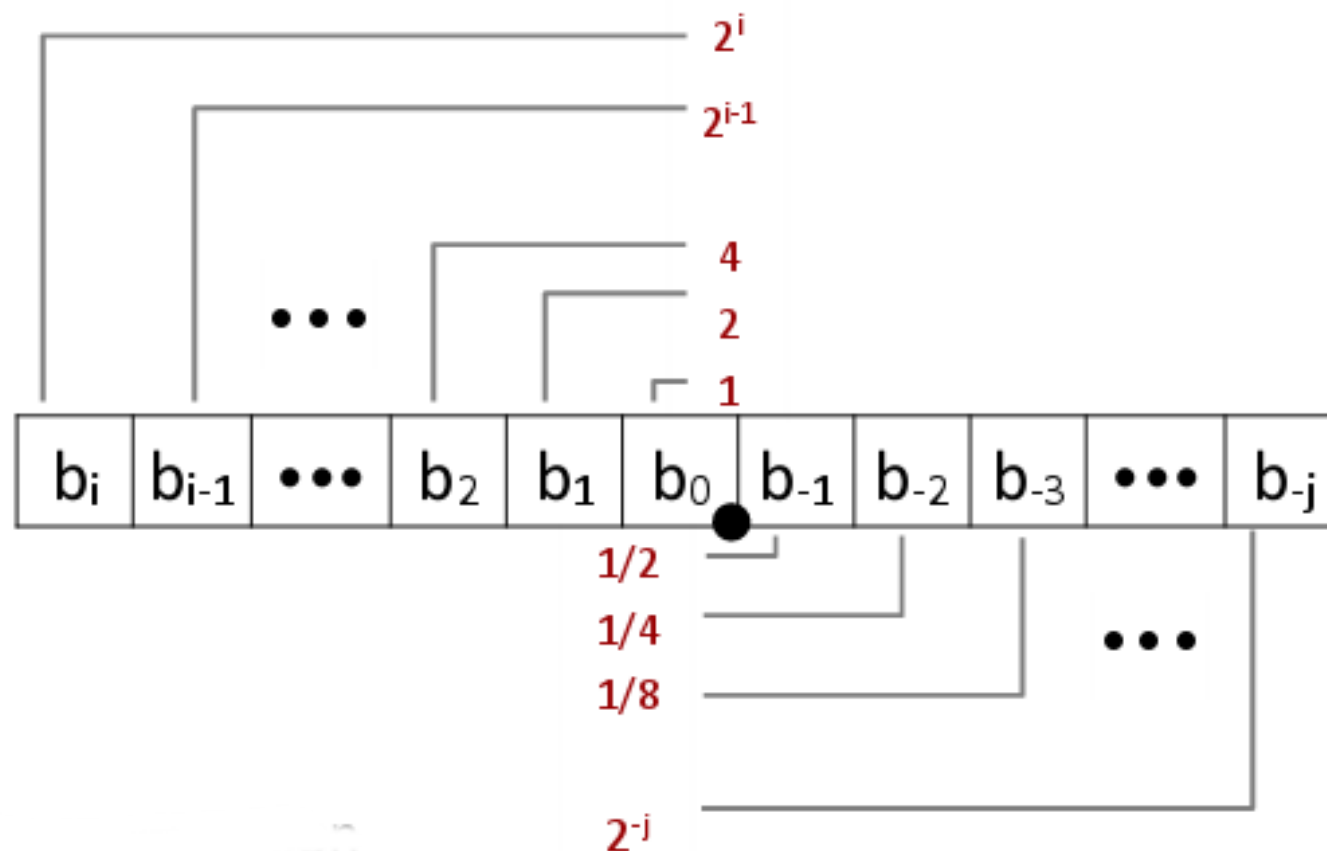
## Floating point

# Floating Point

- Fixed point
- IEEE 754
- Операции с плавающей точкой
- Decimal
- 
- <http://www.cs.cmu.edu/afs/cs/academic/class/15213-f15/www/lectures/04-float.pdf>
-

# Фиксированная точка (fixed point)

- Представление дробных чисел



# Представление fixed point

- Фиксированное число бит для целой и дробной части
- Рассмотрим пример: 4 бита – целая часть, 4 бита – дробная часть
  - $1 \rightarrow 00010000_2$  (первые 4 бита (старшие) – целая часть, младшие 4 бита - дробная)
  - $2.5 \rightarrow 00101000_2$
  - $5.6875 \rightarrow 01011011_2$
- $\text{ULP (unit at the last place)} = 0.0001_2 = 0.0625$
- $\text{ULP}/2 = .03125$

# Преобразование из десятичной записи

- Не всегда преобразование может быть выполнено ТОЧНО
  - $10 = 2 * 5$
  - $1/5$  (0.2) – бесконечная периодичная двоичная дробь:  
 $0.001100110011[0011]..$
  - $00000011_2 \rightarrow 0.1875$ : ошибка = 0.0125
  - $0.1_{10} = 0.0001100110011[0011]..$
  - $00000010_2 \rightarrow 0.125$ : ошибка 0.025
  - $00000001_2 \rightarrow 0.0625$ : ошибка 0.0375 – хуже чем 0.025
  - $000000001_2 \rightarrow 0.0625$  – вес младшего разряда (ULP),  
ошибка преобразования не превышает половины веса  
младшего разряда (ULP/2) (0.03125)

# Операции с фиксированной точкой

- Сложение, вычитание – как целые числа
- Умножение:
  - Умножаем как целые числа, получаем 16-битный результат
  - $0010.0110 * 0100.0010 = 1001.11001100$   
округляем до  $1001.1101$
  - Точный результат: 9.796875, округленный: 9.8125, ошибка: .015625

# Деление с фиксированной точкой

- $0100.0010 / 0010.0110$ 
  - Деление как целые даст только целую часть
  - Дополняем справа 8 нулями, получаем 16-битное число  $0100001000000000_2$  (16896)
  - Делим как целое:  $16896 / 38 = 444 = 110111100_2$
  - Ставим точку на свое место и округляем  $110111100_2 \rightarrow 1.10111100_2 \rightarrow 1.1100_2 = 1.75$
  - Точные вычисления:  $4.125 / 2.375 = 1.73684210\dots$
  - Ошибка:  $0.01315789\dots$
- **Вопрос: какое минимальное количество нулей справа нужно добавить, чтобы получить ошибку не больше  $ULP/2$ ?**

# Округление (rounding)

Значение	1.40	1.60	1.50	2.50	-1.50
К нулю (towards 0)	1	1	1	2	-1
Вниз ( $-\infty$ )	1	1	1	2	-2
Вверх ( $+\infty$ )	2	2	2	3	-1
К ближайшем у целому вверх	1	2	2	3	-1
Ближайшее четное (умолчанию)	1	2	2	2	-2



# Округление к ближайшему четному

- Статистически несмещенное (прочие способы округления статистически смещены)
  - При суммировании накапливается систематическая ошибка
- При применении к другим десятичным/битовым позициям:
  - Когда ровно между двумя возможными значениями округляем к четной последней цифре:

7.8949999	7.89(Less than half way)
7.8950001	7.90(Greater than half way)
7.8950000	7.90(Half way—round up)
7.8850000	7.88(Half way—round down)

# Округление в fixed point

- Четное – младший значащий бит = 0
- “Half-Way” - биты справа от позиции округления равны 100000...
- Примеры (два знака после “.”):
  - $2.09375 = 10.00\textcolor{red}{011}_2 \rightarrow 10.00 = 2$  - down
  - $2.1875 = 10.00\textcolor{red}{110}_2 \rightarrow 10.01 = 2.25$  – up
  - $2.875 = 10.11\textcolor{red}{100}_2 \rightarrow 11.00 = 3$  – up
  - $2.625 = 10.10\textcolor{red}{100}_2 \rightarrow 10.10 = 2.5$  - down

# IEEE Floating Point

- Стандарт IEEE 754
  - 1985 год, до этого – форматы производителей оборудования
  - Поддерживается в основных ЦП
- Появление обусловлено требованиями числовых расчетов
  - Стандарт для переполнений, антипереполнений, округления
  - Трудно реализуется аппаратно
    - Gcc поддерживает -ffast-math и прочие флаги

# Представление чисел

## ■ Числовая форма:

$$(-1)^s M 2^E$$

- Sign bit **s** определяет положительное или отрицательное число
- Significand **M** мантисса определяет значение числа [1.0,2.0).
- Exponent **E** порядок умножает мантиссу на степень 2

## ■ Encoding

- MSB **s** бит знака **s**
- exp поле кодирует **E** (но не равно E)
- frac поле кодирует **M** (но не равно M)



# Точность

- Single precision (одиночная): 32 bits – тип float

s	exp	frac
1	8-bits	23-bits

- Double precision (двойная): 64 bits – тип double

s	exp	frac
1	11-bits	52-bits

- Extended precision (расширенная): 80 bits (Intel only)

s	exp	frac
1	15-bits	63 or 64-bits

# Нормализованные значения

- $\text{exp} \neq 0\dots 0 \ \&\& \ \text{exp} \neq 1\dots 1$  (не все нулевые и не все единичные биты)
- Порядок кодируется со смещением:  $E = \text{exp} - \text{bias}$ 
  - $\text{Exp}$  – беззнаковое значение
  - $\text{Bias} = 2^{k-1}$ ,  $k$  – число бит порядка:
    - Float: 127 ( $\text{exp}$ : 1..254,  $E$ : -126..127)
    - Double: 1023 ( $\text{exp}$ : 1..2046,  $E$ : -1022..1023)
- Мантисса кодируется со “скрытой” ведущей 1:  $M = 1.\text{xxxxxx}_2$ 
  - $\text{xxxxxx}$ : биты мантиссы
  - Минимальное значение:  $\text{frac} = 0..0$  ( $M = 1.0$ )
  - Максимальное значение:  $\text{frac} = 1..1$  ( $M = 2 - \text{eps}$ )

# Пример

$$v = (-1)^s M 2^E$$
$$E = \text{Exp} - \text{Bias}$$

- Значение: float  $F = 15213.0$

$$- 15213_{10} = 11101101101101_2 = 1.1101101101101_2 \times 2^{13}$$

- Мантисса:

$$M = 1.1101101101101_2$$

$$\text{Frac} = 110110110110100000000000_2$$

- Порядок:

$$E = 13$$

$$\text{Bias} = 127$$

$$\text{Exp} = 140 = 10001100_2$$

- Результат

$$0 \ 10001100 \ 110110110110100000000000$$

# Денормализованные значения

- Условие:  $\text{exp} = 0..0$  (все нулевые биты)
- Порядок:  $E = 1 - \text{Bias}$
- Мантисса кодируется со “скрытым” 0:  $M = 0.\text{xxxxxx}_2$ 
  - $\text{xxxxxx}$  – биты мантиссы
- Случаи:
  - $\text{Exp} = 0..0, \text{frac} = 0.0$ 
    - Представление 0
    - Поддерживается знак нуля (0.0 и -0.0)
    - Значение 0.0 – все нулевые биты
  - $\text{Exp} = 0..0, \text{frac} \neq 0..0$  (ненулевые биты мантиссы)
    - Самые близкие к 0 числа
    - На равном расстоянии друг от друга

$$v = (-1)^s M 2^E$$
$$E = \text{Exp} - \text{Bias}$$



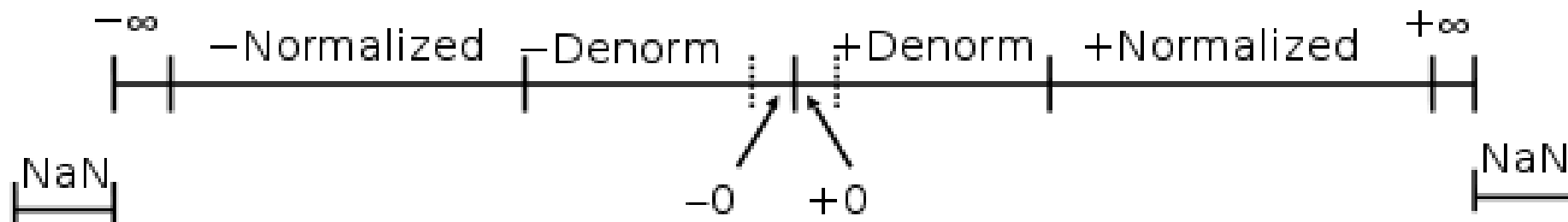
# Специальные значения

$$v = (-1)^s M 2^E$$
$$E = \text{Exp} - \text{Bias}$$

- Exp = 1..1 (все единичные биты)
- Exp = 1..1, frac = 0..0 (все нулевые биты)
  - Представляет бесконечное значение
  - Для операций, результат которых переполняется
  - Сохраняет знак
  - Примеры:  $1.0/0.0 = -1.0/-0.0 = +\infty$ ,  $1.0/-0.0 = -\infty$
- Exp = 1..1, frac != 0..0 (ненулевые биты)
  - Нечисло (Not-a-number – NaN)
  - Для случаев, когда числовой результат не существует
  - Примеры:  $\text{sqrt}(-1)$ ,  $\infty - \infty$ ,  $\infty \times 0$

# Визуализация значений

- Визуализация на числовой прямой

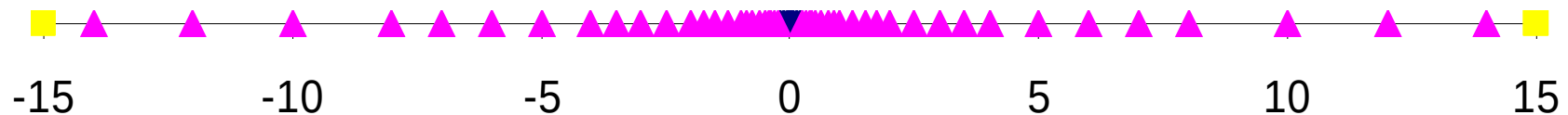


# Распределение значений

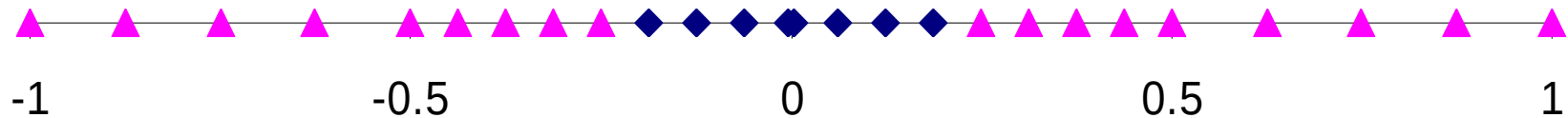
- 6-битовый формат типа IEEE

s	exp	frac
1	3-bits	2-bits

- Значения “уплотняются” к нулю



◆ Denormalized    ▲ Normalized    ■ Infinity



◆ Denormalized    ▲ Normalized    ■ Infinity

# Операции с плавающей точкой

- Сначала вычисляется точный результат
- Потом помещается в требуемую точность
  - Возможное переполнение ( $+\text{INF}$  или  $-\text{INF}$ ) если порядок слишком велик
  - Округление чтобы поместить в мантиссу

# FP Multiplication

■  $(-1)^{s_1} M_1 2^{E_1} \times (-1)^{s_2} M_2 2^{E_2}$

■ Exact Result:  $(-1)^s M 2^E$

- Sign  $s$ :  $s_1 \wedge s_2$
- Significand  $M$ :  $M_1 \times M_2$
- Exponent  $E$ :  $E_1 + E_2$

## ■ Fixing

- If  $M \geq 2$ , shift  $M$  right, increment  $E$
- If  $E$  out of range, overflow
- Round  $M$  to fit frac precision

## ■ Implementation

- Biggest chore is multiplying significands

# Floating Point Addition

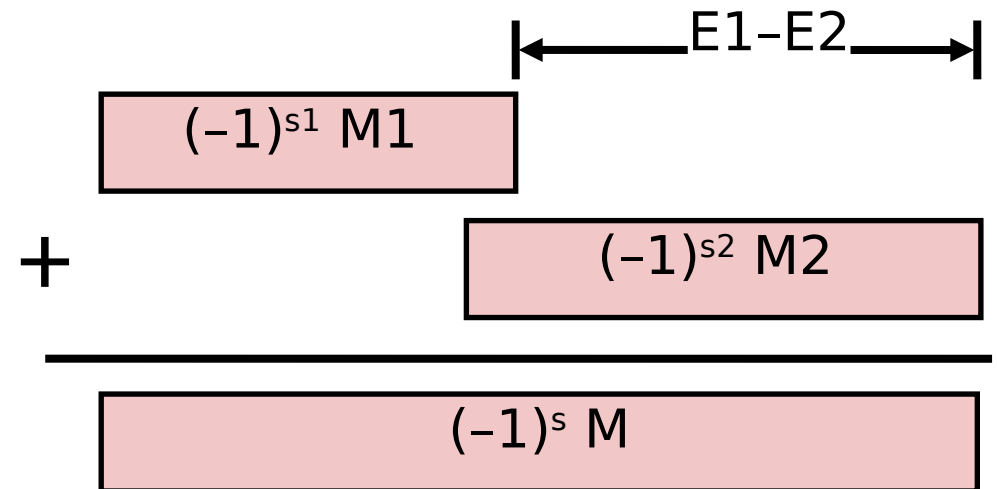
$$\blacksquare (-1)^{s1} M1 2^{E1} + (-1)^{s2} M2 2^{E2}$$

- Assume  $E1 > E2$

Get binary points lined up

$$\blacksquare \text{Exact Result: } (-1)^s M 2^E$$

- Sign S, significand M:
  - Result of signed align & add
- Exponent E:  $E1$



## Fixing

- If  $M \geq 2$ , shift M right, increment E
- if  $M < 1$ , shift M left k positions, decrement E by k
- Overflow if E out of range
- Round M to fit frac precision

# Свойства операций

- Сложение неассоциативно:  $(a+b)+c \neq a+(b+c)$ :

$$(3.14+1e10)-1e10 = 0, \quad 3.14+(1e10-1e10) = 3.14$$

- Умножение неассоциативно:

$$(1e20*1e20)*1e-20 = \text{inf}, \quad 1e20*(1e20*1e-20) = 1e20$$

- Умножение недистрибутивно:  $a*(b+c) \neq a*b+a*c$

$$1e20*(1e20-1e20) = 0.0, \quad 1e20*1e20 - 1e20*1e20 = \text{NaN}$$

# ULP

- ULP (unit in the last place) – единица на младшей позиции, если  $a, b$  – ближайшие друг к другу представимые числа,  $a < b$ , то  $ULP = b - a$
- Если  $a == 1.0$ , то ULP – machine epsilon
- `#include <float.h>`  
`FLT_EPSILON / DBL_EPSILON /`  
`LDBL_EPSILON`



# IEEE 754 Precision

- Все операции (включая алгебраические и трансцендентные: sqrt, sin/cos, atan2, exp, ln) должны давать ошибку не более ULP/2
- Известно, что x86 sin/cos дает намного большую ошибку, пользоваться процессорными инструкциями нельзя
- Glibc реализует вычисления, не используя инструкции процессора

# Накопление ошибки

- Ошибка одной операции –  $ULP/2$
- Предположим, что ошибка  $E$  – равномерно распределенная случайная величина  $[-ULP/2; ULP/2]$
- Мат. Ожидание  $M[E] = 0.0$
- Дисперсия:  $D[E] = ?$

# Накопление ошибки

- Дисперсия:  $D[E] = 1/12$ ,  $\sigma = 1/(2\sqrt{3}) \approx 0.289$  ULP
- Какова будет ошибка при сложении 100 чисел?

# Центральная предельная теорема

- $N$  – норм. Распр.  $(0, 1)$
  - То есть СКО растёт пропорционально корню из числа чисел
  - В случае 100 чисел,  $\sigma \approx 2.89 \text{ ULP}$
  - “правило 3-х сигм”: с вероятностью 99.8% результат лежит на отрезке  $[-8.66\text{ULP}; 8.66\text{ULP}]$
- $$\frac{S_n - \mu n}{\sigma \sqrt{n}} \rightarrow N(0, 1)$$

# Cancellation (потеря точности)

- При сложении двух близких чисел разных знаков в результате сложения останется мало значащих цифр:
- $123457.1467 - 123456.659$   
   $e=5; s=1.234571$   
–  $e=5; s=1.234567$   
-----  
   $e=5; s=0.000004$   
   $e=-1; s=4.000000$
- $e = -1; s = 4.877000$  – ошибка 20%

# Сложение чисел

- Чтобы минимизировать эффекты от потери точности последовательность вещественных чисел следует складывать следующим образом:
  - Положительные числа в порядке возрастания
  - Отрицательные числа в порядке убывания
  - Сложить два результата сложения, полученных выше

# Математические библиотеки

- GMP (libgmp: <https://gmplib.org/>) – целые и рациональные числа произвольной точности
- MPFR (libmpfr: <http://www.mpfr.org/>) – floating-point числа произвольной точности

# Финансовые вычисления

- Важно точное представление десятичных знаков после точки:  
EUR/USD: 1.10685
- Для каждой операции четко определены правила выполнения промежуточных вычислений и округлений:  
ОФЗ 26207: цена 111,4650, 8.15%, НКД 19.20
- **Обычный floating-point нельзя использовать из-за ошибки преобразования и накопления ошибки при сложении/вычитании**



# Финансовые вычисления

- Использование decimal fixed point
  - Хранение в целых числах (например, `int64_t`) с десятичным масштабным коэф. (например, 10000)
  - Использование “длинной арифметики” (`BigDecimal` в Java)
- Типы decimal floating point – порядок – десятичный!
  - IEEE 754-2008
  - Тип `decimal` (C#)
  - Ожидается поддержка в C/C++ (`<decimal/decimal>`)