Курс «Алгоритмы и алгоритмические языки»
1 семестр 2017/2018

Лекция 11

Вычисления с плавающей точкой

- Предпосылки: дробные двоичные числа
- Стандарт арифметики с плавающей точкой IEEE 754: Определение
- Пример и свойства
- Округление, сложение, умножение
- Плавающие типы языка Си
- Флаги компилятора gcc

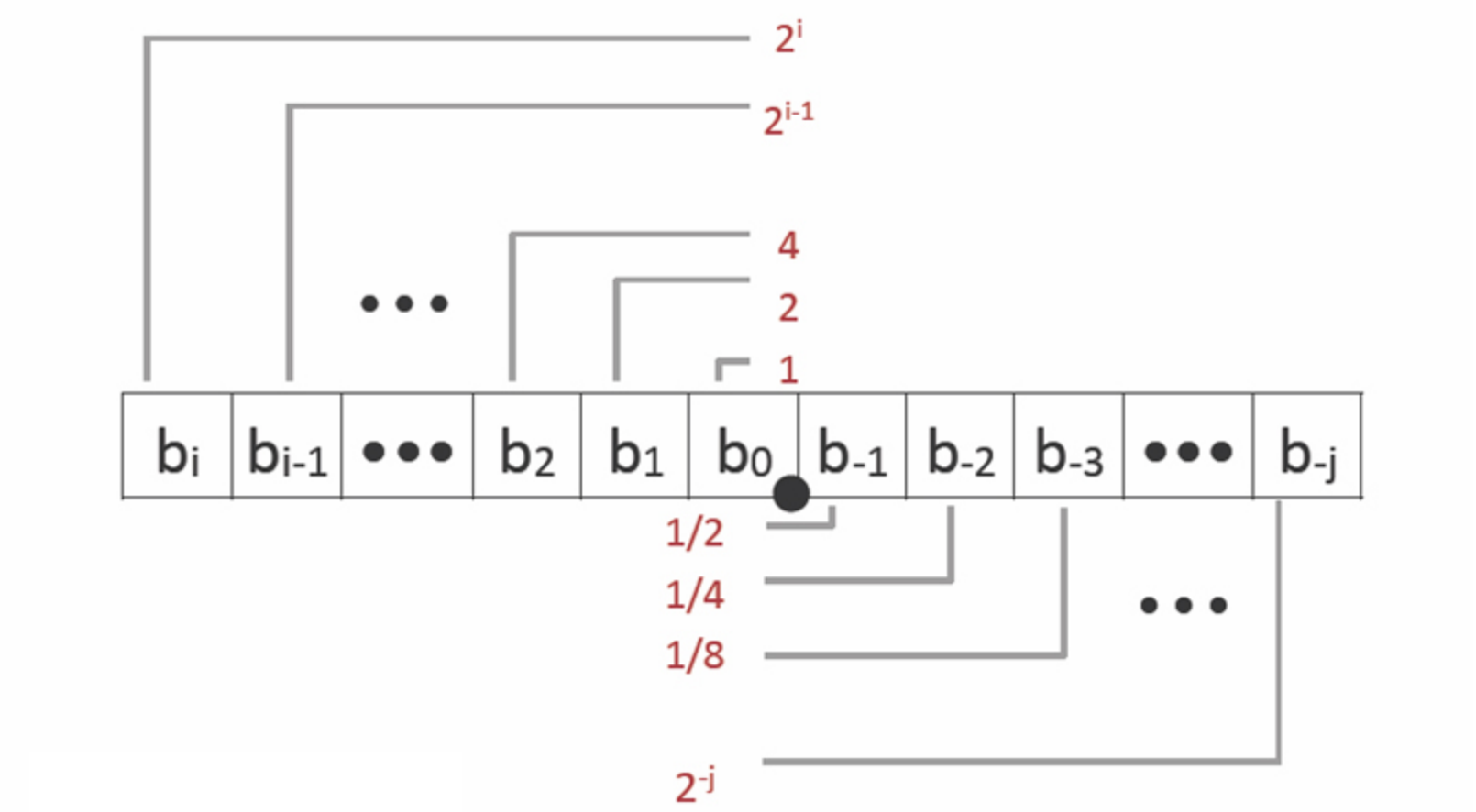
Дробные двоичные числа

- Что такое 1011.101_2 ?

$$1\times 2^3+0\times 2^2+1\times 2^1+1\times 2^0+1\times 2^{-1}+0\times 2^{-2}+1\times 2^{-3}=$$

$$=11\frac{5}{8}=11.625$$

Дробные двоичные числа



- Черное пятнышко – двоичная точка
- Биты слева от точки умножаются на положительные степени 2
- Биты справа от точки умножаются на отрицательные степени 2

Дробные двоичные числа

- $0.111111\ldots_2 = 1.0 - \varepsilon$ ($\varepsilon \to 0$), так как

$$1/2 + 1/4 + 1/8 + \ldots + 1/2^n + \ldots \to 1 \quad \text{при } n \to \infty$$

- Точно можно представить только числа вида $x/2^k$

- Остальные рациональные числа представляются периодическими двоичными дробями:

$$1/5 = 0.(0011)_2$$

- Иррациональные числа представляются апериодическими двоичными дробями и могут быть представлены только приближенно

Представление чисел с плавающей точкой (IEEE 754)

- Числа с плавающей точкой представляются в нормализованной форме: $(-1^s)\ M\ 2^e$
  - s – код знака числа (он же знак мантиссы)
  - M – мантисса ($1 \le M < 2$)
  - e – (двоичный) порядок

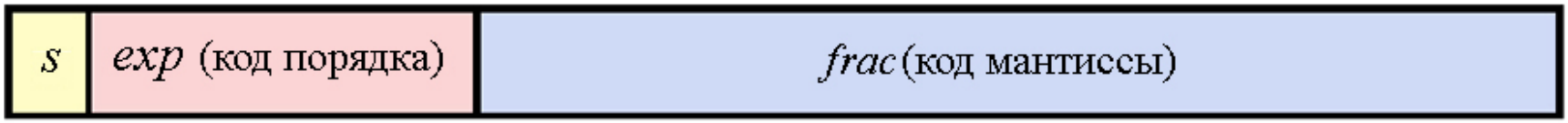
- Первая цифра мантиссы в нормализованном представлении всегда 1. В стандарте принято решение не записывать в представление числа эту единицу (тем самым мантисса как бы увеличивается на разряд).

  Экономия связана с тем, что в представление числа записывается не M, а $frac = M - 1$

Представление чисел с плавающей точкой

- Чтобы не записывать отрицательных чисел в поле порядка, вводится ***смещение*** $bias = 2^{k-1} - 1$, где k – количество бит в поле для записи порядка, и вместо порядка e записывается код порядка exp, связанный с e соотношением $e = exp - bias$.

- Нормализованное число $(-1^s)\ M\ 2^e$ упаковывается в машинное слово (структуру) с полями s, $frac$ и exp



Ширина поля s всегда равна 1.

Ширина полей exp и $frac$ зависит от точности числа

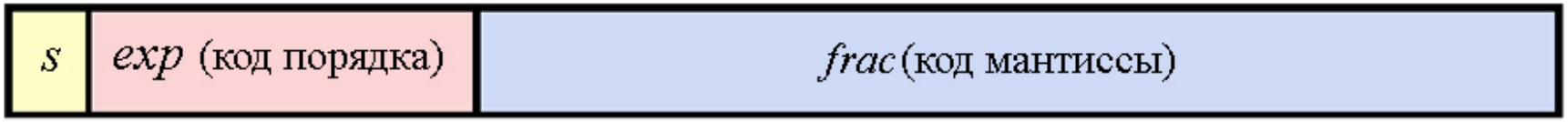
Представление чисел с плавающей точкой

- Одинарная точность (32 бита):

8 бит 23 бита

$bias = 127;$ $-126 \le e \le 127;$ $1 \le exp \le 254$

- Двойная точность (64 бита):



11 бит 52 бита

$bias = 1023;$ $-1022 \le e \le 1023;$ $1 \le exp \le 2046$

- Повышенная точность (80 бит):

15 бит 64 бита

Представление чисел с плавающей точкой

- Пример
  - **Значение** `float f = 15213.0`

    15213_{10} = 11101101101101_2 =

    $1.1101101101101_2 \times 2^{13}$
  - **Значащая часть**

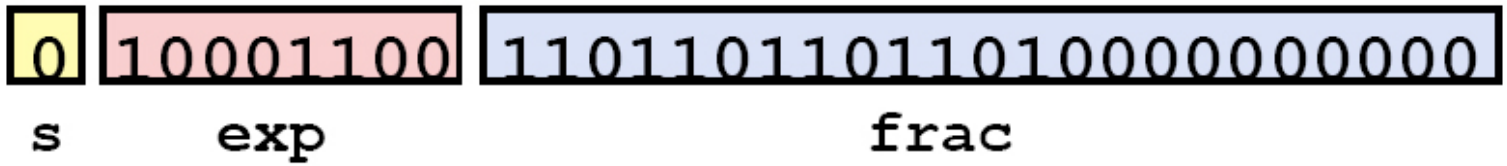
    $M = 1.\underline{1101101101101}_2$,

    $frac = \underline{1101101101101}0000000000_2$
  - **Порядок**

    $e = 13$

    $bias = 127$

    $exp = 140 = 10001100_2$
  - **Результат**



Представление нуля

- Для типа **`float`** код порядка **`exp`** изменяется от **`00000001`** до **`11111110`**
  (значению **`00000001`** соответствует порядок **`e = - 126`**,
  значению **`11111110`** – порядок **`e = 127`**)

- Код **`exp = 00000000, frac = 000…0`**
  представляет нулевое значение; в зависимости от значения знакового разряда **`s`** это либо **`+0`** либо **`-0`**

- А какое значение представляют коды
  `exp = 00000000, frac ≠ 000…0`?
  `exp = 11111111?`

Большие числа

Пусть **exp = 111…1**

- если при этом **frac = 000…0**, то коду будет соответствовать значение ∞ (со знаком **s**)
- если же **frac ≠ 000…0**, то код не будет представлять никакое число («значение», представляемое таким кодом, так и называется: «не число» – **NaN** – Not a number)

Денормализованные числа

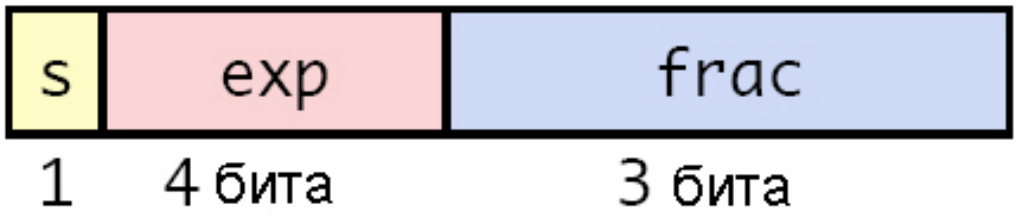
- Это числа, представляемые кодами

  exp = 00000000, frac ≠ 000…0

- **exp** вносит в значение такого числа постоянный вклад 2^{-k-2},

  frac меняется от **000…01** до **111…1 и** рассматривается уже не как мантисса, а как значение, умножаемое на **exp**

- Рассмотрим это на модельном примере:

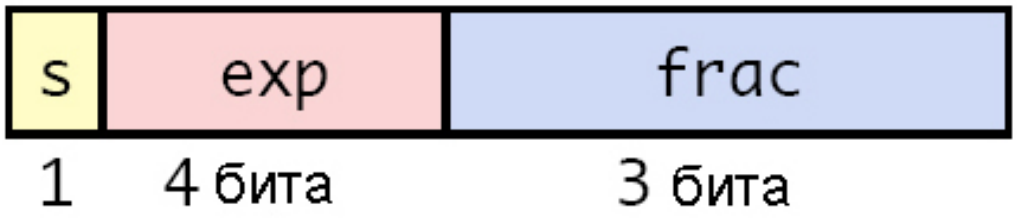


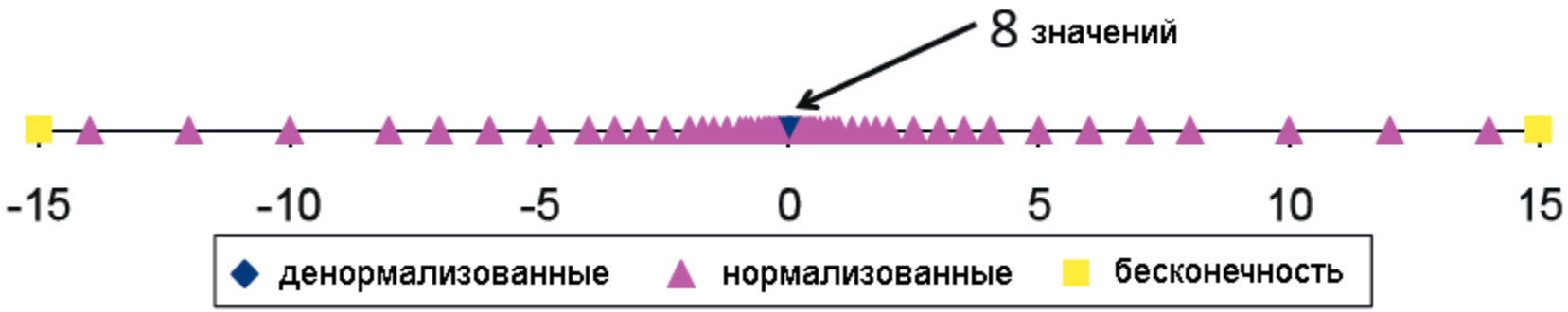
8-разрядные числа с плавающей точкой (положительные)

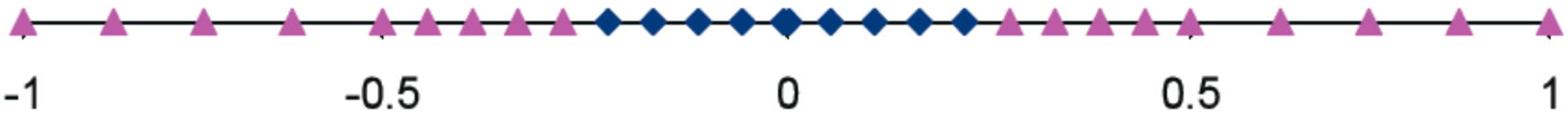
1 4 бита 3 бита

	s	exp	frac	E	Value	
Ненормализованные числа	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	Близкие к 0
	0	0000	010	-6	2/8*1/64 = 2/512	
	…					
	0	0000	110	-6	6/8*1/64 = 6/512	
	0	0000	111	-6	7/8*1/64 = 7/512	Наибольшее ненормализованное
Нормализованные числа	0	0001	000	-6	8/8*1/64 = 8/512	Наименьшее нормализованное
	0	0001	001	-6	9/8*1/64 = 9/512	
	…					
	0	0110	110	-1	14/8*1/2 = 14/16	
	0	0110	111	-1	15/8*1/2 = 15/16	Ближайшее к 1 снизу
	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	Ближайшее к 1 сверху
	0	0111	010	0	10/8*1 = 10/8	
	…					
	0	1110	110	7	14/8*128 = 224	
	0	1110	111	7	15/8*128 = 240	Наибольшее нормализованное
	0	1111	000	n/a	inf	

8-разрядные числа с плавающей точкой



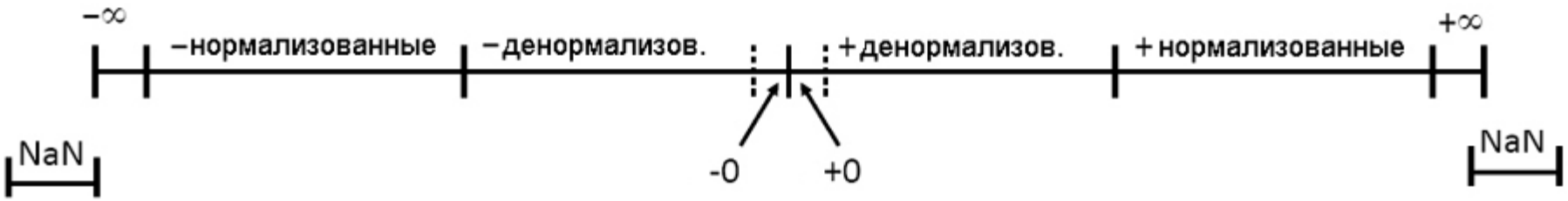




Центральная область более крупно

Важные частные случаи

	exp	frac	Численное значение
◈ Нуль	00…00	00…00	0.0
◈ Наим. положит. денорм.	00…00	00…01	
♦ float ≈ 1.4×10^{-45}			$2^{-23}\times2^{-126}$
♦ double ≈ 4.9×10^{-324}			$2^{-52}\times2^{-1022}$
◈ Наиб. положит. денорм.	00…00	11…11	
♦ float ≈ 1.18×10^{-38}			$(1.0 - \varepsilon)\times2^{-126}$
♦ double ≈ 2.2×10^{-308}			$(1.0 - \varepsilon)\times2^{-1022}$
◈ Наим. положит. норм.	00…01	00…00	
♦ float			1.0×2^{-126}
♦ double			1.0×2^{-1022}
◈ Единица	01…11	00…00	1.0
◈ Наиб. положит. норм.			
♦ float ≈ 3.4×10^{38}			$(2.0 - \varepsilon)\times2^{127}$
♦ double ≈ 1.8×10^{308}			$(2.0 - \varepsilon)\times2^{1023}$



Операции над числами с плавающей точкой

- $x +_{FP} y = Round(x + y)$

  $x \times_{FP} y = Round(x \times y)$

  где $Round()$ означает округление

- Выполнение операции
  - Сначала вычисляется точный результат (получается более длинная мантисса, чем запоминаемая, иногда в два раза)
  - Потом фиксируется исключение (например, переполнение)
  - Потом результат округляется, чтобы поместиться в поле $frac$

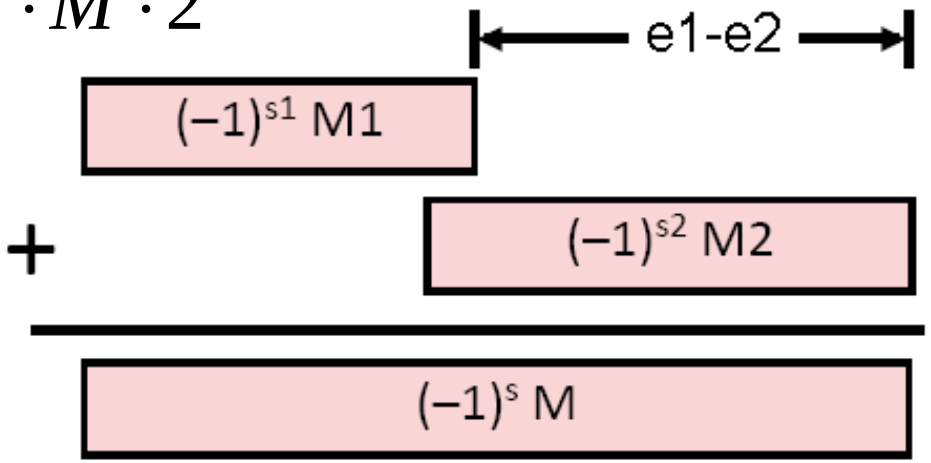
Умножение чисел с плавающей точкой

- $(-1)^{s1} \cdot M_1 \cdot 2^{e1} \times (-1)^{s2} \cdot M_2 \cdot 2^{e2}$
- Точный результат $(-1)^{s} \cdot M \cdot 2^{e}$
  - Знак s $s1 \wedge s2$
  - Значащие цифры M $M_1 \times M_2$
  - Порядок e $e1 + e2$
- Преобразование
  - Если $M \geq 2$, сдвиг M вправо с одновременным увеличением e
  - Если e не помещается в поле exp, переполнение
  - Округление M, чтобы оно поместилось в поле $frac$
- Основные затраты на перемножение мантисс

Сложение чисел с плавающей точкой

- $(-1)^{s1} \cdot M_1 \cdot 2^{e1} + (-1)^{s2} \cdot M_2 \cdot 2^{e2}$
  Пусть $e1 > e2$

- Точный результат $(-1)^{s} \cdot M \cdot 2^{e}$
  - Знак s и значащие цифры M вычисляются как показано на рисунке
  - Порядок суммы – $e1$



- Преобразование
  - Если $M \geq 2$, сдвиг M вправо с одновременным увеличением e
  - Если $M < 1$, сдвиг M влево на k позиций с одновременным вычитанием k из e
  - Если e не помещается в поле exp, переполнение
  - Округление M, чтобы оно поместилось в поле $frac$

Пример 1. Вычисление суммы 5 чисел типа **`float`**
(мантисса – 6 десятичных цифр, порядок – 2 десятичных цифры):

$0.231876*10^{02} + 0.645391*10^{-03} + 0.231834*10^{-01} + 0.245383*10^{-02} + 0.945722*10^{-03} =$

a) $\mathbf{0.231876*10^{02} + 0.645391*10^{-03} + 0.231834*10^{-01} + 0.245383*10^{-02} + 0.945722*10^{-03} = 0.2321\underline{4}7*10^{02}}$;

23.1876 + 0.000645391 = 23.188245391 = 23.1882 = 0. $231882*10^{02}$;

23.1882 + 0.0231834 = 23.2113834 = 23.2114 = $0.232114*10^{02}$;

23.2114 + 0.00245383 = 23.21385383 = $23.2138*10^{02}$;

23. 2138 + 0.000945722 = 23.214745722 = 23.2147 = $0.232147*10^{02}$;

b) $\mathbf{0.645391*10^{-03} + 0.9457*10^{-03} + 0.245383*10^{-02} + 0.231834*10^{-01} + 0.231876*10^{02} = 0.2321\underline{5}7*10^{02}}$;

0.000645391 + 0.000945722 = 0.001591113 = 0.00159111 = 0. $159111*10^{-02}$;

0.00159111 + 0.00245383 = 0.00494493 = 0. $494493*10^{-02}$;

0.00494493 + 0.0231834 = 0.02812833 = 0.0281283 = 0. $281283*10^{-01}$;

0.0281283 + 23.1876 = 23.2157283 = 23.2157 = $0.232157*10^{02}$;

Пример 2. Вычисление разности плавающих чисел

(мантисса – 6 десятичных цифр, порядок – 2 десятичных цифры):

$0.238617*10^{02} – 0.238616*10^{02} + 0.645391*10^{04} – 0.645392*10^{04} + 0.845791*10^{00} – 0.835790*10^{00} =$

a) $0.238617*10^{02} – 0.238616*10^{02} + 0.645391*10^{04} – 0.645392*10^{04} + 0.845791*10^{00} – 0.835790*10^{00} =$ **$0.100000*10^{-05}$**

$0.238617*10^{02} – 0.238616*10^{02} = 23.8617 – 23.8616 = 0.0001 =$ **$0.100000*10^{03}$**

$0.100000*10^{-03} + 0.645391*10^{04} = 0.0001 + 6453.91 = 6453.9101 =$ **$0.645391*10^{04}$**

$0.645391*10^{04} – 0.645392*10^{04} = – 0.000001*10^{04} =$ **$– 0.100000*10^{-01}$**

$– 0.100000*10^{-01} + 0.845791*10^{00} = – 0.01 + 0.845791 = 0.835791 *10^{00}$

$0.835791 *10^{00} – 0.835790*10^{00} =$ **$0.000001*10^{00} = 0.100000*\underline{10^{-05}}$**

b) $0.238617*10^{02} + 0.645391*10^{04} + 0.845791*10^{00} – (0.238616*10^{02} + 0.645392*10^{04} + 0.835790*10^{00}) =$ **$0.100000*10^{00}$**

$0.238617*10^{02} + 0.645391*10^{04} = 23.8617 – 6453.91 = 6478.6 =$ **$0.647777*10^{04}$**

$0.647777*10^{04} + 0.845791*10^{00} = 6477.77 + 0.845791 = 6478.615791 =$ ***$0.647862*10^{04}$***

$0.238616*10^{02} + 0.645392*10^{04} = 23.8616 + 6453.92 = 6477.7816 =$ **$6477.78*10^{04}$**

$6477.78*10^{04} + 0.835790*10^{00} = 6477.78 + 0.835790 = 6478.61579 =$ ***$0.647852*10^{04}$***

$0.647862*10^{04} – 0.647852*10^{04}=$ **$0.000010*10^{04} = 0.100000*\underline{10^{-00}}$**

Выводы по операциям

(1) ***При вычислении суммы чисел с одинаковыми знаками*** необходимо упорядочить слагаемые по возрастанию и складывать, начиная с наименьших слагаемых.

(2) ***При вычислении суммы чисел с разными знаками*** необходимо сначала сложить все положительные числа, потом – все отрицательные числа и в конце выполнить одно вычитание.

(3) ***Вычитание*** (сложение чисел с противоположными знаками) ***часто приводит к потере точности***, которая у чисел с плавающей точкой определяется количеством значащих цифр в мантиссе (при вычитании двух близких чисел мантисса «исчезает», что ведет к резкой потере точности).
Итак, ***чем меньше вычитаний, тем точнее результат*.**

Значащими цифрами числа с плавающей точкой называются все цифры его мантиссы за исключением нулей, стоящих в ее конце. Например, у числа 0.**6700089**0000 * 10^3 все цифры, выделенные жирным шрифтом, значащие. При вычитании двух близких чисел почти все значащие цифры пропадают. Например, 0.67000890 * 10^3 - 0.67000880 * 10^3 = 0.00000010 * 10^3 = 0.10 * 10^{-4}. Таким образом, у результата всего одна значащая цифра, хотя у операндов было по 7 значащих цифр.

Плавающие типы языка Си

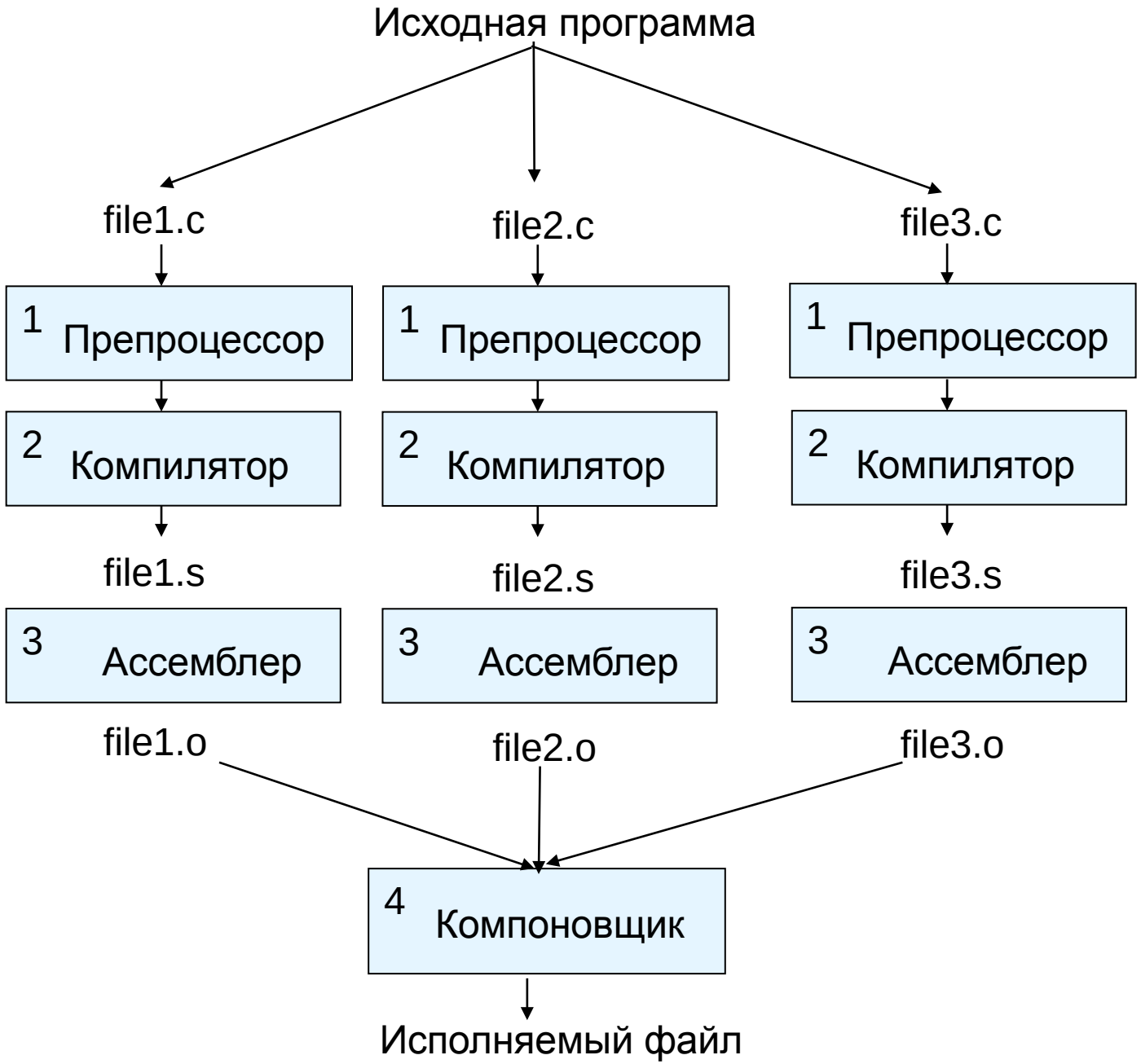
`float`, `double`, `long double`

- ***Операции над данными с плавающей точкой***.
  - *Одноместные*: изменение знака («одноместный минус»: **–**), одноместный плюс (**+**).
  - *Двухместные*: сложение (**+**), вычитание (**–**), умножение (*****), деление (**/**).
- ***Порядок выполнения арифметических операций в выражениях (приоритет).***
  - самый низкий приоритет у двуместных **+** и **–**,
  - более высокий приоритет у двуместных ***** и **/**,
  - еще более высокий приоритет у одноместных **+** и **–**.

Режимы gcc для работы с плавающей точкой*

- **`https://gcc.gnu.org/wiki/FloatingPointMath`**
  Детальное резюме того, что бывает в **`gcc`**, и таблица преобразований, влияющих на результат вычислений
- **`-ffast-math`**: считать максимально быстро, но, возможно, нарушать стандарт IEEE-754
  - Полезно для тестирования, но не распространения финальной версии программы
- **`-fno-math-errno`**: не устанавливать переменную **`errno`** как результат ошибочного выполнения математических функций
  - Можно обойтись и без этого, но зависит от библиотеки Си
  - Компилятор может заменять вызовы функций инструкциями процессора (например, sqrt)
- **`-fno-trapping-math`**: считать, что вычисления с плавающей точкой не могут вызывать исключений процессора (traps)
  - Т.е. вы гарантируете отсутствие в своем коде ситуаций, вызывающих деления на ноль, переполнения, некорректные операции
  - Компилятор может более свободно комбинировать, переставлять, удалять операции с плавающей точкой
- David Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.* 23, 1 (March 1991), 5-48
  https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

Схема раздельной компиляции



Препроцессор

- Перед компиляцией выполняется этап препроцессирования. Это обработка программного модуля для получения его окончательного текста, который отдается компилятору.
- Управление препроцессированием выполняется с помощью *директив* препроцессора:

```
#include <...> - системные библиотеки
#include "..." – пользовательские файлы

#define name(parameters) text
#undef name

#define MAX 128
#define ABS(x) ((x) >= 0 ? (x) : -(x))

x -> y – 7
ABS(x) -> ((y – 7) >= 0 ? (y – 7) : -(y – 7))

x -> a-- ?
```

Препроцессор и условная компиляция

- Препроцессор позволяет организовать условное включение фрагментов кода в программу

#ifdef name / #endif – проверка определения имени

```
#ifndef _STDIO_H
#define _STDIO_H
<... текст файла ...>
#endif
```

Препроцессор и условная компиляция

- Препроцессор позволяет организовать условное включение фрагментов кода в программу

#if/#if defined/#elif/#else/#endif – общие проверки условий

```
#if HOST_BITS_PER_INT >= 32
typedef unsigned int gfc_char_t;
#elif HOST_BITS_PER_LONG >= 32
typedef unsigned long gfc_char_t;
#elif defined(HAVE_LONG_LONG)
      && (HOST_BITS_PER_LONGLONG >= 32)
typedef unsigned long long gfc_char_t;
#else
#error "Cannot find an integer type with at least 32 bits"
#endif
```

Препроцессор: операции # и

- Операция # позволяет получить строковое представление аргумента

```
#define FAIL(op)                                                \
          do {                                                  \
            fprintf (stderr, "Operation " #op "failed: "        \
                     "at file %s, line %d\n", __FILE__,         \
                      __LINE__);                                \
            abort ();                                           \
          } while (0)

int foo (int x, int y) {
  if (y == 0)
    FAIL (division);
  return x / y;
}

do { fprintf (stderr, "Operation " "division" "failed: " "at file
%s, line %d\n", "fail.c", 13); abort (); } while (0);
```

Препроцессор: операции # и

- Операция ## позволяет объединить фактические аргументы макроса в одну строку

java-opcodes.h:

```
enum java_opcode {
#define JAVAOP(NAME, CODE, KIND, TYPE, VALUE) \
        OPCODE_##NAME = CODE,
#include "javaop.def"
#undef JAVAOP
LAST_AND_UNUSED_JAVA_OPCODE
};
```

javaop.def:

```
JAVAOP (nop,              0, STACK,   POP,    0)
JAVAOP (aconst_null,      1, PUSHC,   PTR,    0)
JAVAOP (iconst_m1,        2, PUSHC,   INT,    -1)
<...>
JAVAOP (ret_w,          209, RET,     RETURN, VAR_INDEX_2)
JAVAOP (impdep1,        254, IMPL,    ANY,    1)
JAVAOP (impdep2,        255, IMPL,    ANY,    2)
```

Препроцессор: операции # и

- Операция ## позволяет объединить фактические аргументы макроса в одну строку

```
gcc –E java-opcodes.h:
enum java_opcode {
OPCODE_nop = 0,
OPCODE_aconst_null = 1,
OPCODE_iconst_m1 = 2,
OPCODE_iconst_0 = 3,
<...>
OPCODE_impdep2 = 255,
LAST_AND_UNUSED_JAVA_OPCODE
};
```

Компоновка и классы памяти

Класс памяти	Время жизни	Видимость	Компоновка	Определена
автоматический	автоматическое	блок	нет	В блоке
регистровый	автоматическое	блок	нет	В блоке как **`register`**
статический	статическое	файл	внешняя	Вне функций
статический	статическое	файл	внутренняя	Вне функций как **`static`**
статический	статическое	блок	нет	В блоке как **`static`**

- Квалификатор **`extern`**: переменная определена и память под нее выделена в другом файле
- Классы памяти функций:
  - статическая (объявлена с квалифатором **`static`**)
  - внешняя (**`extern`**), по умолчанию
  - встраиваемая (**`inline`**, C99)
- Объявление внешних функций в заголовочных файлах:

```
extern void *realloc (void *ptr, size_t size);
```

Компоновщик

- Организовывает слияние нескольких объектных файлов в одну программу
- Разрешает неизвестные символы (внешние переменные и функции)
  - Глобальные переменные с одним именем получают одну область памяти
  - Ошибки, если необходимых имен нет или есть несколько объектов с одним именем
  - Опции для указания места поиска
- Хорошим стилем программирования является экспорт лишь тех объектов, которые используются в других файлах (интерфейс модуля)
  - Используйте квалификатор **`static`**
- Сборка исполняемого файла или библиотеки (*статической* или *динамической*)