

# Компьютерная арифметика

- § 24. Особенности представления чисел в компьютере
- § 25. Хранение в памяти целых чисел
- § 26. Операции с целыми числами
- § 27. Хранение в памяти вещественных чисел
- § 28. Операции с вещественными числами

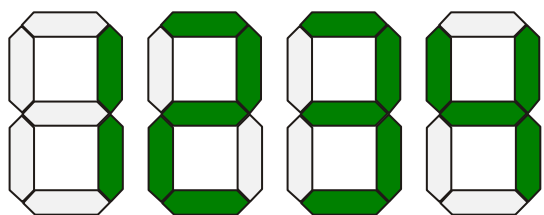
# Компьютерная арифметика

## § 24. Особенности представления чисел в компьютере

# Предельные значения чисел

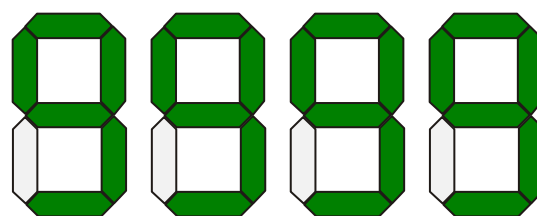
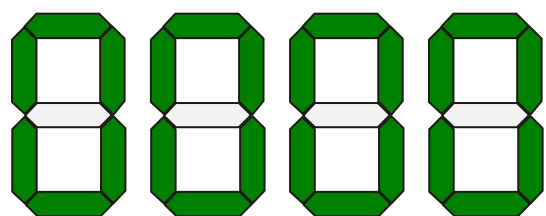
В математике *нет* предельных значений!

В компьютере – конечное число деталей, ограниченное количество разрядов.



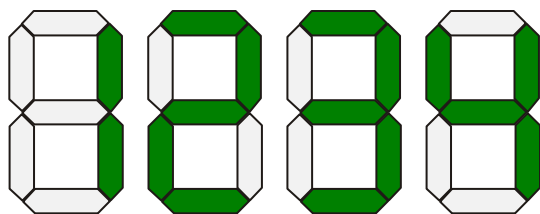
?

Какой диапазон?



~~10000?~~

# Предельные значения чисел



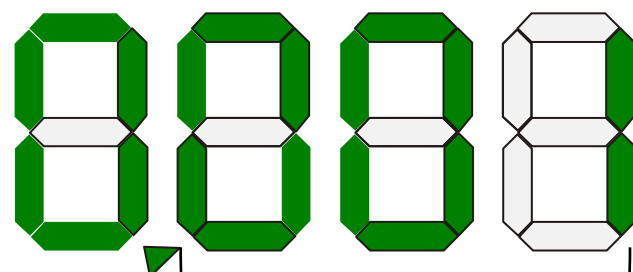
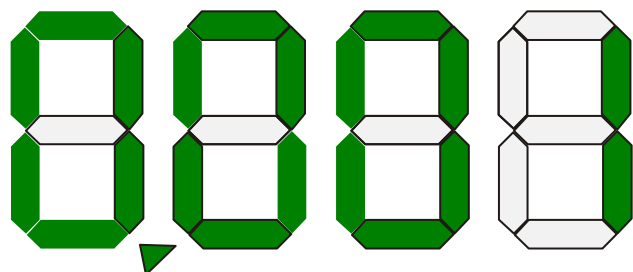
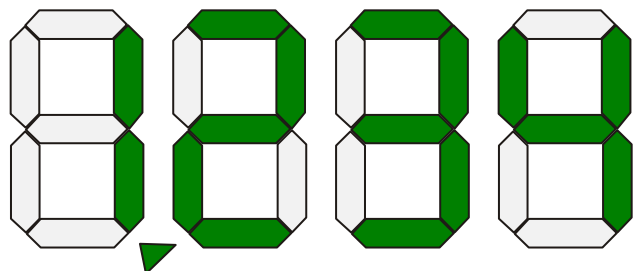
*K* разрядов

система счисления  
с основанием *B*

$$C_{\max} = B^K - 1$$

**Переполнение разрядной сетки** — это ситуация, когда число, которое требуется сохранить, не уместается в имеющемся количестве разрядов вычислительного устройства.

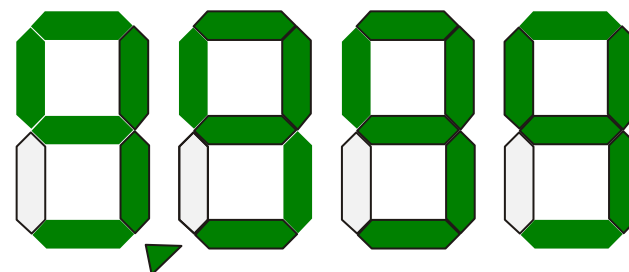
# Вещественные числа



$F$  разрядов в  
дробной части



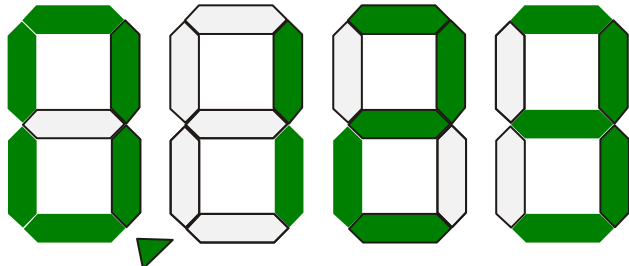
Какой диапазон?



система счисления с  
основанием  $B$

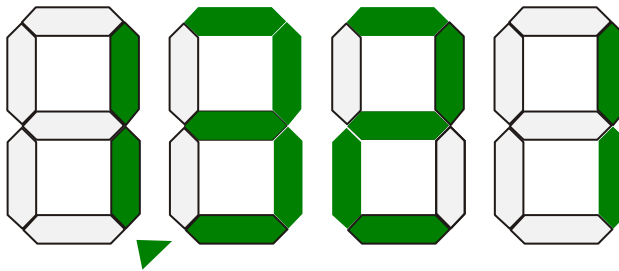
$$C_{\min} = B^{-F}$$

# Неточность представления

0,123**4567** → 



Не все вещественные числа могут быть представлены в компьютере точно!

1,3211 → 

1,3212

1,3214

# Сравнение вещественных чисел

$$X = 10^6, Y = 10^{-6}$$

хранится неточно!

$$X \cdot Y \approx 1$$

неточный результат!

$$0,3 \approx 0,1 + 0,1 + 0,1$$



При сравнении вещественных чисел желательно избегать операции «равно»!

$$X \cdot Y \approx 1 \rightarrow |1 - X \cdot Y| \leq \varepsilon$$

допустимая погрешность ( $10^{-6}$ )

$$X \approx Y \rightarrow |X - Y| \leq \varepsilon$$

# Дискретность

---

1. Целые числа дискретны.
2. Вещественные числа **непрерывны**.
3. Компьютер работает только с **дискретными** данными.



Для хранения вещественных чисел  
нужна **дискретизация!**

4. При дискретизации может происходить **потеря информации** (искажение данных).
5. Большинство **трудностей** связано с кодированием вещественных чисел.



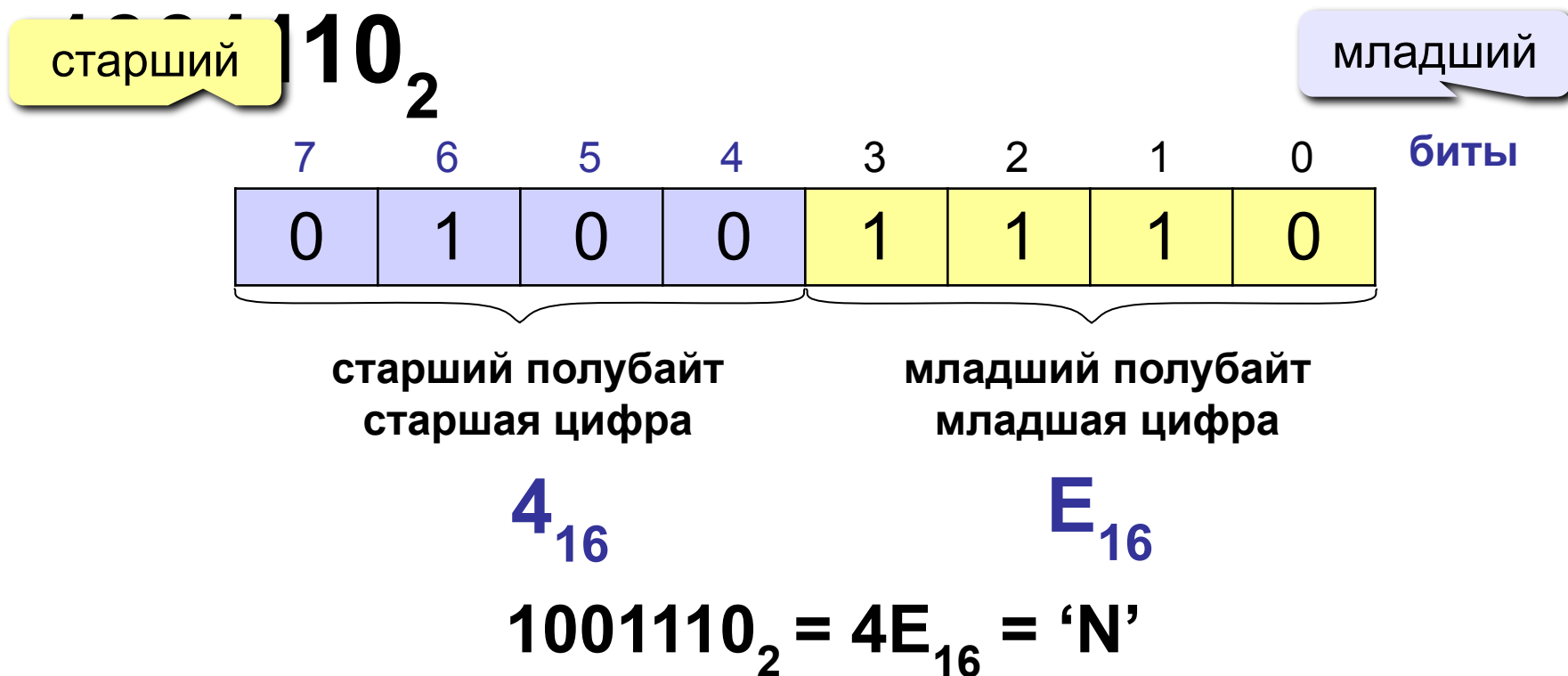
# Компьютерная арифметика

## § 25. Хранение в памяти целых чисел

# Целые числа без знака (*unsigned*)

**Беззнаковые данные** – не могут быть отрицательными.

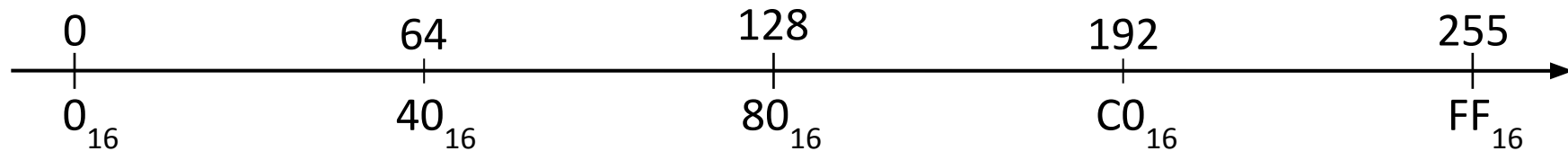
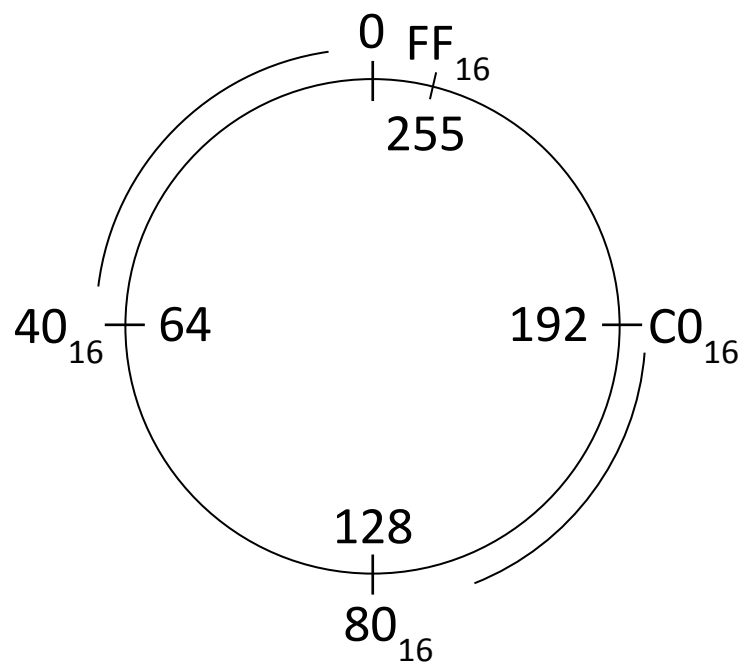
78 =



# Целые числа без знака

$X_{10}$	0	1	...	127	128	...	255
$X_{16}$	$00_{16}$	$01_{16}$	...	$7F_{16}$	$80_{16}$	...	$FF_{16}$
$X_2$	$0000\ 0000_2$	$0000\ 0001_2$	...	$0111\ 1111_2$	$1000\ 0000_2$	...	$1111\ 1111_2$

$$\begin{array}{r}
 1111\ 1111 \\
 + 0000\ 0001 \\
 \hline
 1\ 0000\ 0000
 \end{array}$$



# Целые числа без знака: диапазон

$$X_{\max} = 2^K - 1$$

$K$	$X_{\min}$	$X_{\max}$	типы данных
8	0	255	<b>byte</b> (Паскаль) <b>unsigned char</b> (C/C++)
16	0	65 535	<b>word</b> (Паскаль) <b>unsigned short</b> (C/C++)
32	0	4 294 967 295	<b>cardinal</b> (Delphi) <b>unsigned int</b> (C/C++)
64	0	18 446 744 073 709 551 615	<b>unsigned long</b> <b>long</b> (C/C++)

# Целые числа со знаком



Сколько места требуется для хранения знака?

**Старший (знаковый) бит** числа определяет его знак. Если он равен 0, число положительное, если 1, то отрицательное.

Прямой код:

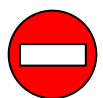
$\geq 0$

$$78 = 1001110_2$$

$$-78 = -1001110_2$$

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

$< 0$



операции с положительными и отрицательными числами выполняются по-разному!

## Целые числа со знаком

**Идея:** « $-1$ » должно быть представлено так, чтобы при сложении с числом « $1$ » получить  $0$ .



Как кодируется « $-1$ »?

$$\begin{array}{r}
 1111 \ 1111 \quad -1 \rightarrow 255 \\
 + \ 0000 \ 0001 \quad 1 \\
 \hline
 1 \ 0000 \ 0000 \quad 256
 \end{array}$$

**Для 8-битных чисел:** код числа « $-X$ » равен двоичному коду числа  $256 - X$  (дополнение до 256).



При  $K$ -битном кодировании *дополнительный* код числа « $-X$ » равен двоичному коду числа  $2^K - X$  (дополнение до  $2^K$ ).

## Как построить дополнительный код?

**Алгоритм А0:** перевести число  $2^K - X$  в двоичную систему счисления.

 для вычислений требуется  $K+1$  разряд

### Алгоритм А1:

- 1) перевести число  $X$  в двоичную систему счисления;
- 2) построить *обратный код*, выполнив *инверсию* всех битов (заменить 0 на 1 и наоборот);
- 3) к результату добавить 1.

$$\begin{array}{rcl}
 78 & = & 01001110_2 \\
 & & 10110001 \quad \leftarrow \text{инверсия} \\
 -78 & \rightarrow & 10110010 \quad +1
 \end{array}$$

## Как построить дополнительный код?

### Алгоритм А2:

- 1) перевести число  $X-1$  в двоичную систему счисления;
- 2) выполнить инверсию всех битов.

$$78 - 1 = 77 = 01001101_2$$

$$-78 \rightarrow 10110010_2 \leftarrow \text{инверсия}$$

### Алгоритм А3:

- 1) перевести число  $X$  в двоичную систему счисления;
- 2) выполнить инверсию всех старших битов числа, кроме младшей единицы и нулей после нее.

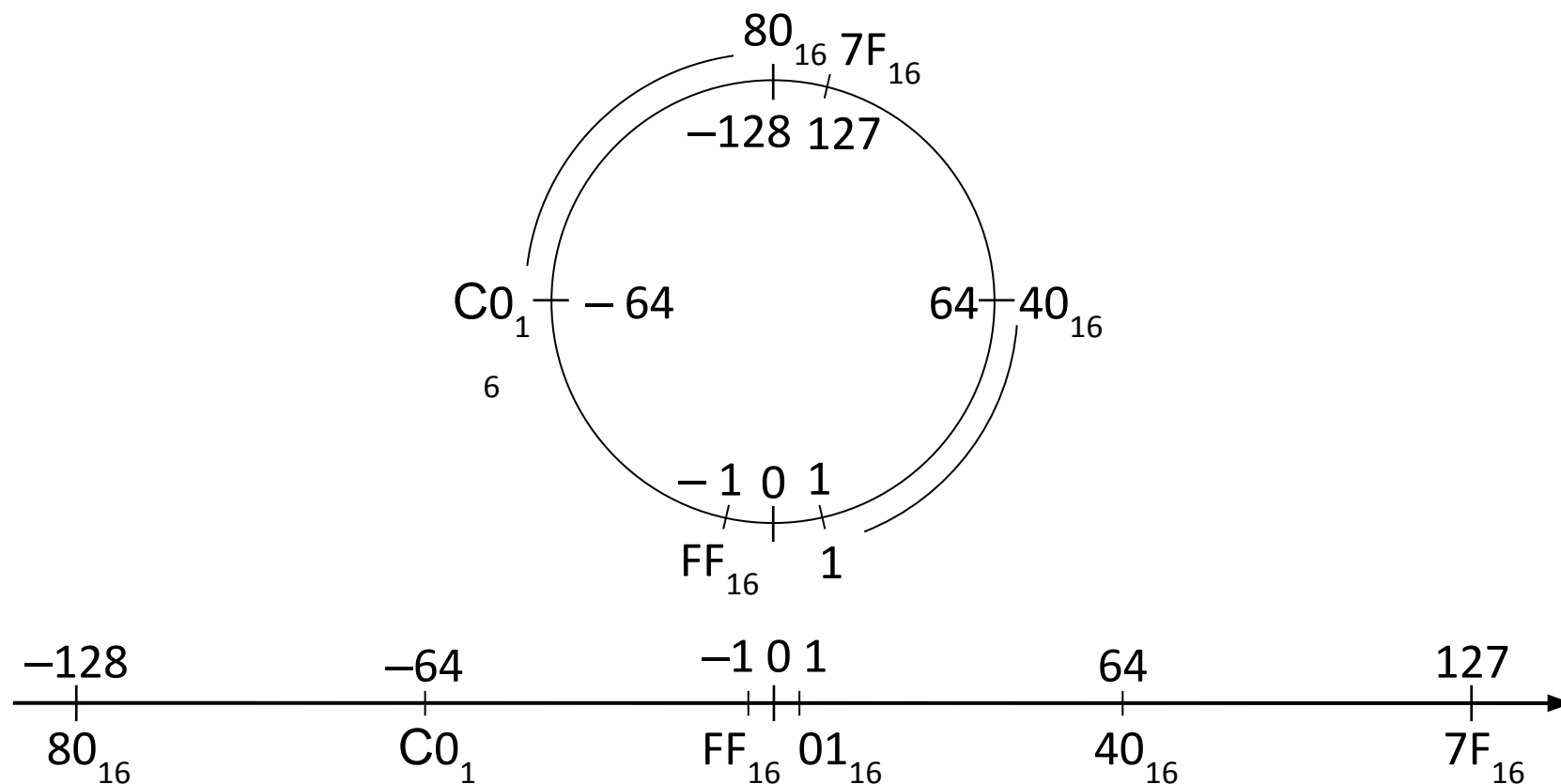
$$78 = 01001110_2$$

$$-78 \rightarrow 10110010_2 \leftarrow \text{инверсия}$$



# Целые числа со знаком

$X_{10}$	-128	-127	...	-1	0	...	127
$X_{16}$	$80_{16}$	$81_{16}$	...	$FF_{16}$	$00_{16}$	...	$7F_{16}$
$X_2$	$1000\ 0000_2$	$1000\ 0001_2$	...	$1111\ 1111_2$	$0000\ 0000_2$	...	$0111\ 1111_2$



## Целые числа со знаком: диапазон

$$X_{\min} = -2^{K-1}$$



$$X_{\max} = 2^{K-1} - 1$$

$K$	$X_{\min}$	$X_{\max}$	типы данных
8	-128	127	<code>shortInt</code> (Delphi) <code>char</code> (Cu)
16	-32 768	32 767	<code>smallInt</code> (Delphi) <code>short</code> (Cu)
32	-2 147 483 648	2 147 483 647	<code>integer</code> (Delphi) <code>int</code> (Cu)
64	$-2^{63}$	$2^{63} - 1$	<code>int64</code> (Delphi) <code>long long</code> (Cu)

# Компьютерная арифметика

## § 26. Операции с целыми числами

# Сложение и вычитание



Операции с положительными и отрицательными числами выполняются по одинаковым алгоритмам!

$$\begin{array}{r}
 + \quad 5 \\
 - \quad 9 \\
 \hline
 -4
 \end{array}
 \quad
 \begin{array}{r}
 + \quad 0000 \quad 0101 \\
 \quad 1111 \quad 0111 \\
 \hline
 \quad 1111 \quad 1100
 \end{array}$$

←



Вычитание = сложение с дополнительным кодом вычитаемого!

# Переполнение

дополнительный  
бит

$$\begin{array}{r|l}
 0 & 01100000 & 96 \\
 + & 000100001 & 33 \\
 \hline
 0 & 10000001 & -127 \\
 S' & S & \text{знаковый бит}
 \end{array}$$

$$\begin{array}{r|l}
 1 & 10100000 & -96 \\
 + & 11101111 & -33 \\
 \hline
 1 & 0111111 & 127 \\
 S' & S & 
 \end{array}$$



Если бит S не совпадает с битом S', произошло переполнение и результат неверный.

# Умножение

$$\begin{array}{r}
 \times 00001001 \quad 9 \\
 \quad 00000101 \quad 5 \\
 \hline
 \quad 00001001 \\
 + 00000000 \\
 00001001 \\
 \hline
 0000101101 \rightarrow 45
 \end{array}$$

$$\begin{array}{r}
 \times 11110111 \quad -9 \\
 \quad 00000101 \quad 5 \\
 \hline
 \quad 11110111 \\
 + 00000000 \\
 11110111 \\
 \hline
 10011010011 \rightarrow -45
 \end{array}$$



Умножение выполняется с помощью сложения и сдвига.

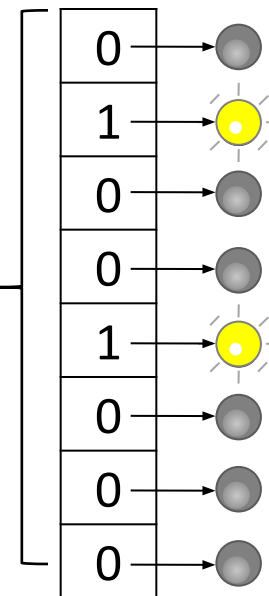
# Поразрядные логические операции

**Поразрядные операции** выполняются с отдельными битами числа и не влияют на остальные.



Сложение – это поразрядная операция?

регистр



**Операция «НЕ» (инверсия, not):**

R

1	0	0	1	1	0	1	0
---	---	---	---	---	---	---	---

not R

0	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---

# Логическая операция «И» (and, &)

данные

маска

D	M	D and M
0	0	0
1	0	0
0	1	0
1	1	1

**Маска** – константа, которая определяет область применения логической операции к битам многоразрядного числа.

$$AA_{16} \text{ and } 6C_{16} = ?$$

	7	6	5	4	3	2	1	0	
D	1	0	1	0	1	0	1	0	$AA_{16}$
M	0	1	1	0	1	1	0	0	$6C_{16}$
D and M	0	0	1	0	1	0	0	0	$28_{16}$



С помощью операции «И» можно сбросить (установить в ноль) биты, для которых маска равна 0!



# Логическая операция «ИЛИ» (or, |)

D	M	D or M
0	0	0
1	0	1
0	1	1
1	1	1

$$AA_{16} \text{ or } 6C_{16} = ?$$

	7	6	5	4	3	2	1	0	
D	1	0	1	0	1	0	1	0	$AA_{16}$
M	0	1	1	0	1	1	0	0	$6C_{16}$
D or M	1	1	1	0	1	1	1	0	$EE_{16}$



С помощью операции «ИЛИ» можно записать единицу в биты, для которых маска равна 1!

# Операция «исключающее ИЛИ» (**xor**, $\wedge$ )

D	M	D <b>xor</b> M
0	0	0
1	0	1
0	1	1
1	1	0

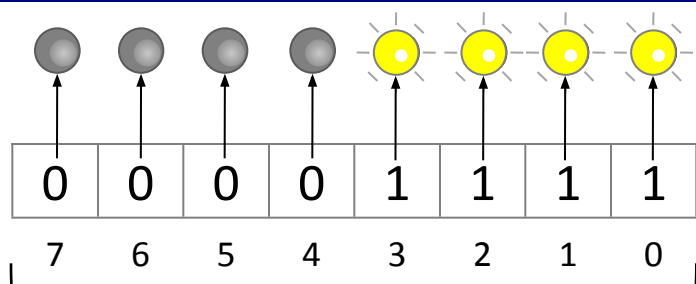
$$AA_{16} \mathbf{xor} 6C_{16} = ?$$

	7	6	5	4	3	2	1	0	
D	1	0	1	0	1	0	1	0	$AA_{16}$
M	0	1	1	0	1	1	0	0	$6C_{16}$
D <b>xor</b> M	1	1	0	0	0	1	1	0	$C6_{16}$

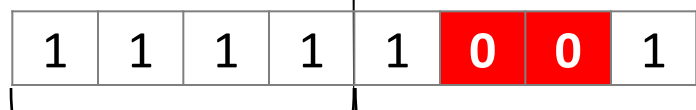
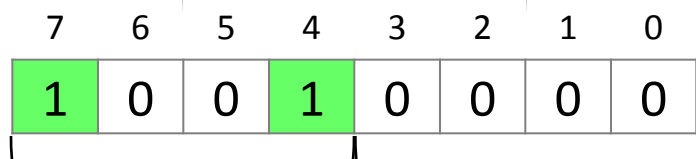
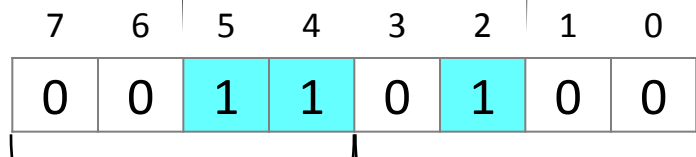


С помощью операции «исключающее ИЛИ» можно инвертировать биты, для которых маска равна 1!

# Битовые логические операции (итог)



R

 $F_{16}$  $9_{16}$  $9_{16}$  $0_{16}$  $3_{16}$  $4_{16}$ 

1) отключить лампочки 2 и 1, не трогая остальные

$$R = R \text{ and } F9_{16}$$

2) включить лампочки 7 и 4

$$R = R \text{ or } 90_{16}$$

3) изменить состояние лампочек 5, 4 и 2

$$R = R \text{ xor } 34_{16}$$

# Шифрование с помощью xor

**Идея:**  $(A \text{ xor } B) \text{ xor } B = A$



Операция «исключающее ИЛИ» *обратима*, то есть ее повторное применение восстанавливает исходное значение!

**Текст:**  $2 * 2 = 4$

**Коды символов:**

$$'2' = 32_{16} = 00110010_2$$

$$'*' = 2A_{16} = 00101010_2$$

$$'=' = 3D_{16} = 00111101_2$$

$$'4' = 34_{16} = 00110100_2$$

## Шифрование с помощью **xor**

**Маска:**  $23 = 17_{16} = 00010111_2$

$$'2' \rightarrow 32_{16} \text{ xor } 17_{16} = 25_{16} \rightarrow '\%'$$

$$'*' \rightarrow 2A_{16} \text{ xor } 17_{16} = 3D_{16} \rightarrow '='$$

$$'=' \rightarrow 3D_{16} \text{ xor } 17_{16} = 2A_{16} \rightarrow '*'$$

$$'4' \rightarrow 34_{16} \text{ xor } 17_{16} = 23_{16} \rightarrow '\#'$$

**Исходный текст:**  $2*2=4$

**Зашифрованный текст:**  $\%=\%*\#$

**Расшифровка:**

$$'%' \rightarrow 25_{16} \text{ xor } 17_{16} = 32_{16} \rightarrow '2'$$

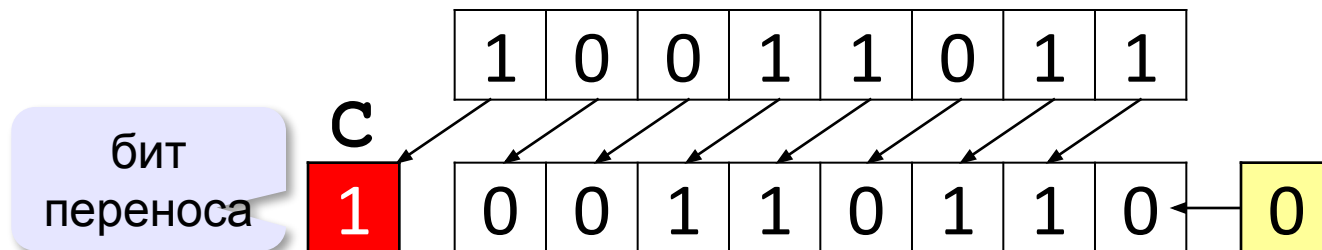
$$'=' \rightarrow 3D_{16} \text{ xor } 17_{16} = 2A_{16} \rightarrow '*'$$

$$'*' \rightarrow 2A_{16} \text{ xor } 17_{16} = 3D_{16} \rightarrow '='$$

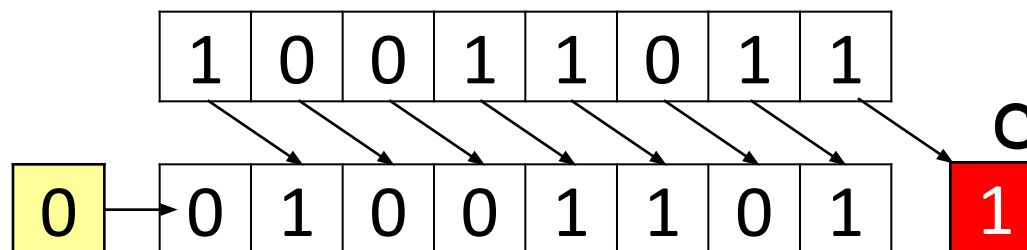
$$'#' \rightarrow 23_{16} \text{ xor } 17_{16} = 34_{16} \rightarrow '4'$$

# Логический сдвиг

Влево:



Вправо:



C, C++, Python:

```
N = N << 1;
N = N >> 1;
```

Паскаль:

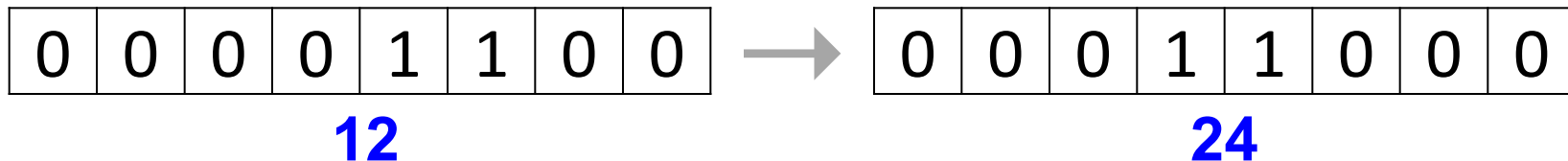
shift left

```
N := N shl 1;
N := N shr 1;
```

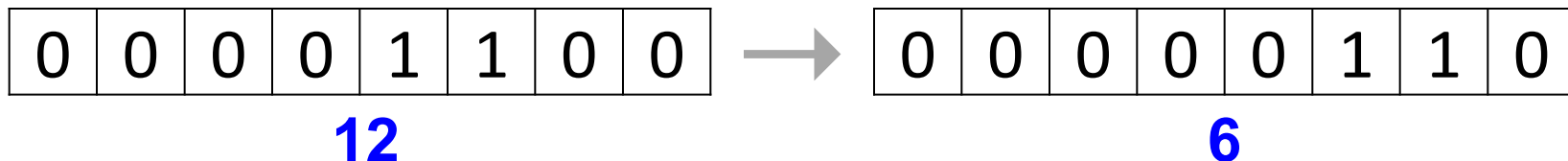
shift right

# Логический сдвиг

**Влево:**



**Вправо:**



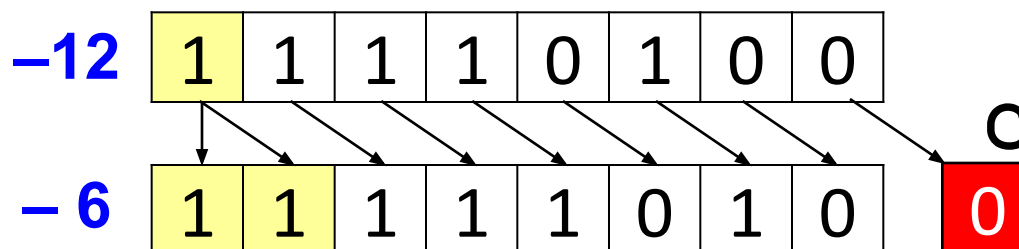
Если число нечётное?

Логический сдвиг влево (вправо) – это быстрый способ **умножения** (деления без остатка) положительного числа **на 2**.



Если число отрицательное?

# Арифметический сдвиг (вправо)



Если число нечётное?

## Примеры:

20 → 10

15 → 7

11 → 5

3 → 1

1 → 0

-20 → -10

-15 → -8

-11 → -6

-3 → -2

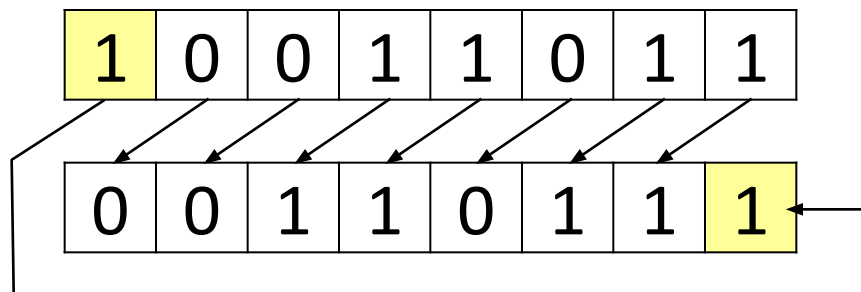
-1 → -1

Арифметический сдвиг вправо – деление на 2 нацело с округлением «вниз» (к ближайшему меньшему целому).

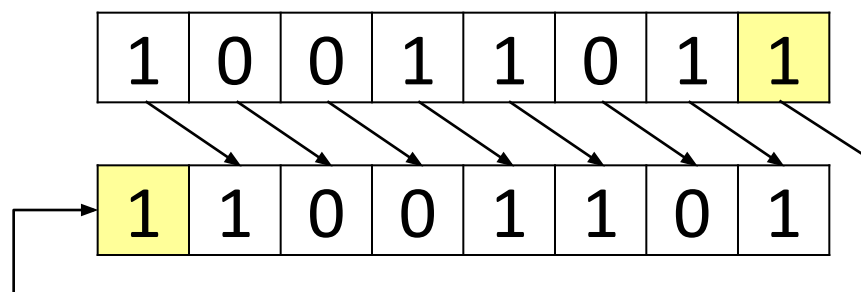


# Циклический сдвиг

Влево:



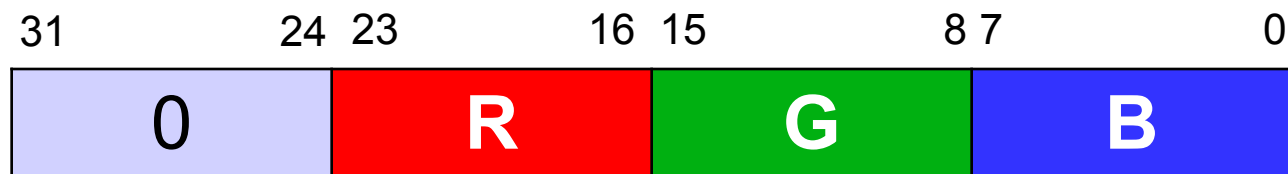
Вправо:



Циклический сдвиг предназначен для последовательного просмотра битов без потери данных.

## Пример

**Задача:** в целой переменной **N** (32 бита) закодирована информация о цвете пикселя в **RGB**:



Записать в переменные R, G, B составляющие цвета.

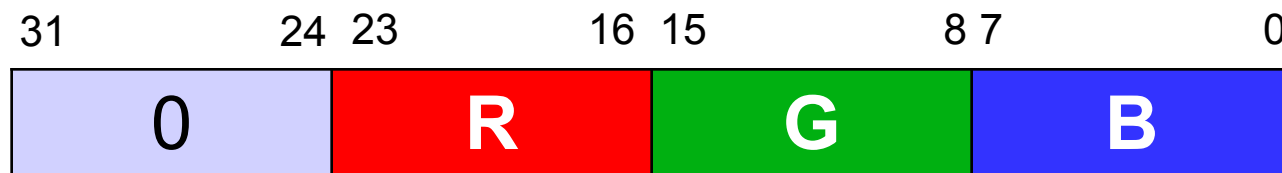
### Вариант 1:

1. Обнулить все биты, кроме **G**.  
Маска для выделения **G**: **0000FF00**<sub>16</sub>
2. Сдвинуть вправо так, чтобы число **G** передвинулось в младший байт.

**C, C++, Python:** `G = (N & 0xFF00) >> 8;`

**Паскаль:** `G := (N and $FF00) shr 8;`

# Пример



## Вариант 2:

1. Сдвинуть вправо так, чтобы число **G** передвинулось в младший байт.
2. Обнулить все биты, кроме **G**.

Маска для выделения **G**:  $000000FF_{16}$

**C, C++, Python :**  $G = (N \gg 8) \& 0xFF;$

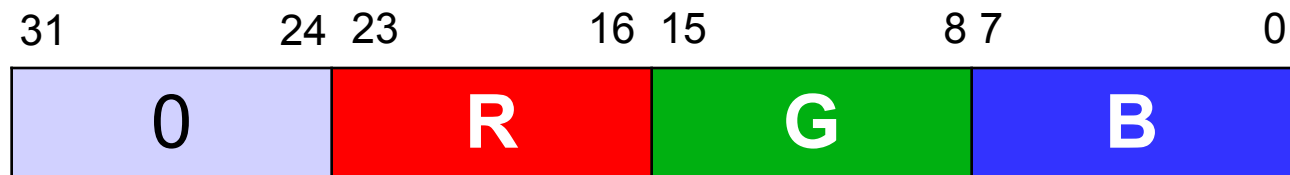
**Паскаль:**  $G := (N \text{ shr } 8) \text{ and } \$FF;$



Как решить, используя только сдвиги?

# Пример

---



**C, C++, Python:** `R =`

`B =`

**Паскаль:** `R :=`

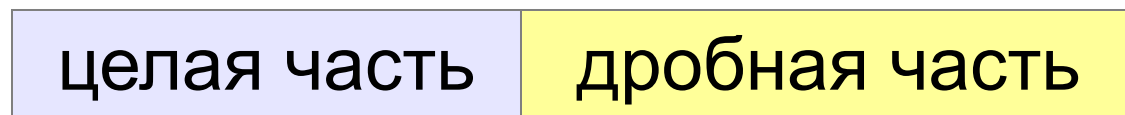
`B :=`

# Компьютерная арифметика

## § 27. Хранение в памяти вещественных чисел

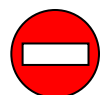
# Хранение вещественных чисел

**С фиксированной запятой** (в первых ЭВМ):



0,000000000000000012345

1234500000000000000,0



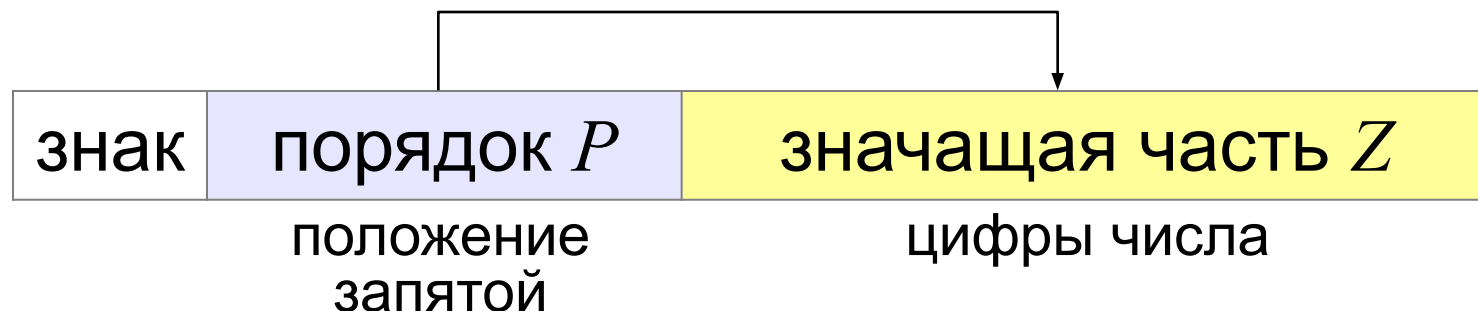
для больших и маленьких чисел нужно масштабирование

**С плавающей запятой** (автоматическое масштабирование):

$$A = \pm Z \cdot B^P$$

$$1,2345 \cdot 10^{-14}$$

$$1,2345 \cdot 10^{17}$$



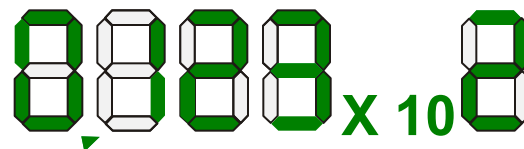
# Хранение вещественных чисел

**Теоретически оптимальный вариант** (целая часть = 0):

$$0,0012345 = 0,12345 \cdot 10^{-2}$$

$$12,345 = 0,12345 \cdot 10^2$$

всегда 0



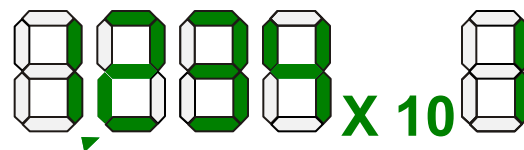
один разряд расходуется впустую!

ОСНОВАНИЕ СИСТЕМЫ  
СЧИСЛЕНИЯ

**Экономный вариант** (целая часть от 1 до B):

$$0,0012345 = 1,2345 \cdot 10^{-3}$$

$$12,345 = 1,2345 \cdot 10^1$$



повышение точности при конечном числе разрядов

# Нормализация

**Нормализованная форма:** значащая часть  $Z$  удовлетворяет условию  $1 \leq Z < B$ , где  $B$  – основание системы счисления (стандарт *IEEE 754*).

**Пример:**

$$17,25 = 10001,01_2 = 1,000101_2 \cdot 2^4$$

$$5,375 =$$

$$7,625 =$$

$$27,875 =$$

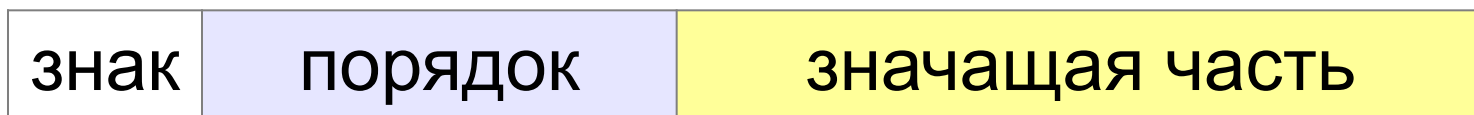
$$13,5 =$$

$$0,125 =$$

всегда 1, её можно  
не хранить в памяти!

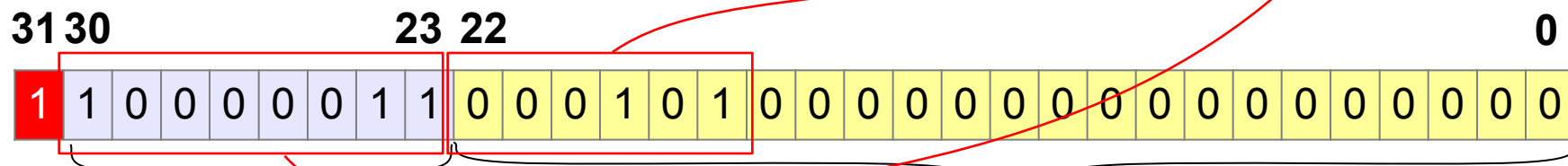


# Число обычной точности (*single*)



$$-17,25 = -10001,01_2 = -1,000101_2 \cdot 2^4$$

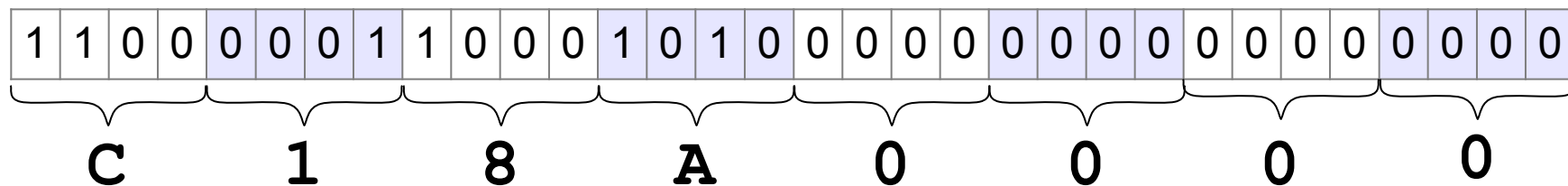
**single:** 4 байта = 32 бита



знак      порядок со  
              смещением      мантисса = дробная часть  $Z$

$$p = 4 + 127 = 131 = 10000011_2$$

для *single*



# Диапазон вещественных чисел

тип	диапазон	число десятичных значащих цифр	размер (байт)
<i>single</i>	$1,4 \cdot 10^{-45} - 3,4 \cdot 10^{38}$	7-8	4
<i>double</i>	$4,9 \cdot 10^{-324} - 1,8 \cdot 10^{308}$	15-16	8
<i>extended</i>	$3,6 \cdot 10^{-4951} - 1,2 \cdot 10^{4932}$	19-20	10

*Extended* – тип для вычислений в сопроцессоре, единица в значащей части не скрывается.

*Single, double* – только для хранения.

# Компьютерная арифметика

## § 28. Операции с вещественными числами

# Сложение и вычитание



Как сложить два числа с плавающей запятой?

$$1,2345 \cdot 10^{-5} + 1,2345 \cdot 10^5 = ?$$

Пример:

$$7,25 = 111,01_2 = 1,1101_2 \cdot 2^2$$

$$1,75 = 1,11_2 = 1,11_2 \cdot 2^0$$

1) порядки выравниваются до большего

$$1,75 = 0,0111_2 \cdot 2^2$$

2) значащие части складываются (или вычитаются)

$$\begin{array}{r} 1,1101_2 \\ + 0,0111_2 \\ \hline 10,0100_2 \end{array}$$

3) результат нормализуется

$$10,01_2 \cdot 2^2 = 1,001_2 \cdot 2^3$$



Почему порядки выравнивают до большего?

# Умножение и деление



Как умножить два числа с плавающей запятой?

$$1,2345 \cdot 10^{-5} \cdot 1,2345 \cdot 10^5 = ?$$

Пример:

$$1,75 = 1,11_2 = 1,11_2 \cdot 2^0$$

$$6 = 110_2 = 1,1_2 \cdot 2^2$$

1) значащие части умножаются (или делятся)

$$1,11_2 \cdot 1,1_2 = 10,101_2$$

2) порядки складываются (или вычитаются)

$$0 + 2 = 2$$

3) результат нормализуется

$$10,101_2 \cdot 2^2 = 1,0101_2 \cdot 2^3$$



Надо ли выравнивать порядки?

# Конец фильма

---

**ПОЛЯКОВ Константин Юрьевич**

д.т.н., учитель информатики

ГБОУ СОШ № 163, г. Санкт-Петербург

[kpolyakov@mail.ru](mailto:kpolyakov@mail.ru)

**ЕРЕМИН Евгений Александрович**

к.ф.-м.н., доцент кафедры мультимедийной

дидактики и ИТО ПГГПУ, г. Пермь

[eremin@pspu.ac.ru](mailto:eremin@pspu.ac.ru)

# Источники иллюстраций

---

1. авторские материалы