

# Линейные сортировки. Представление чисел в памяти компьютера

Горденко Мария Константиновна

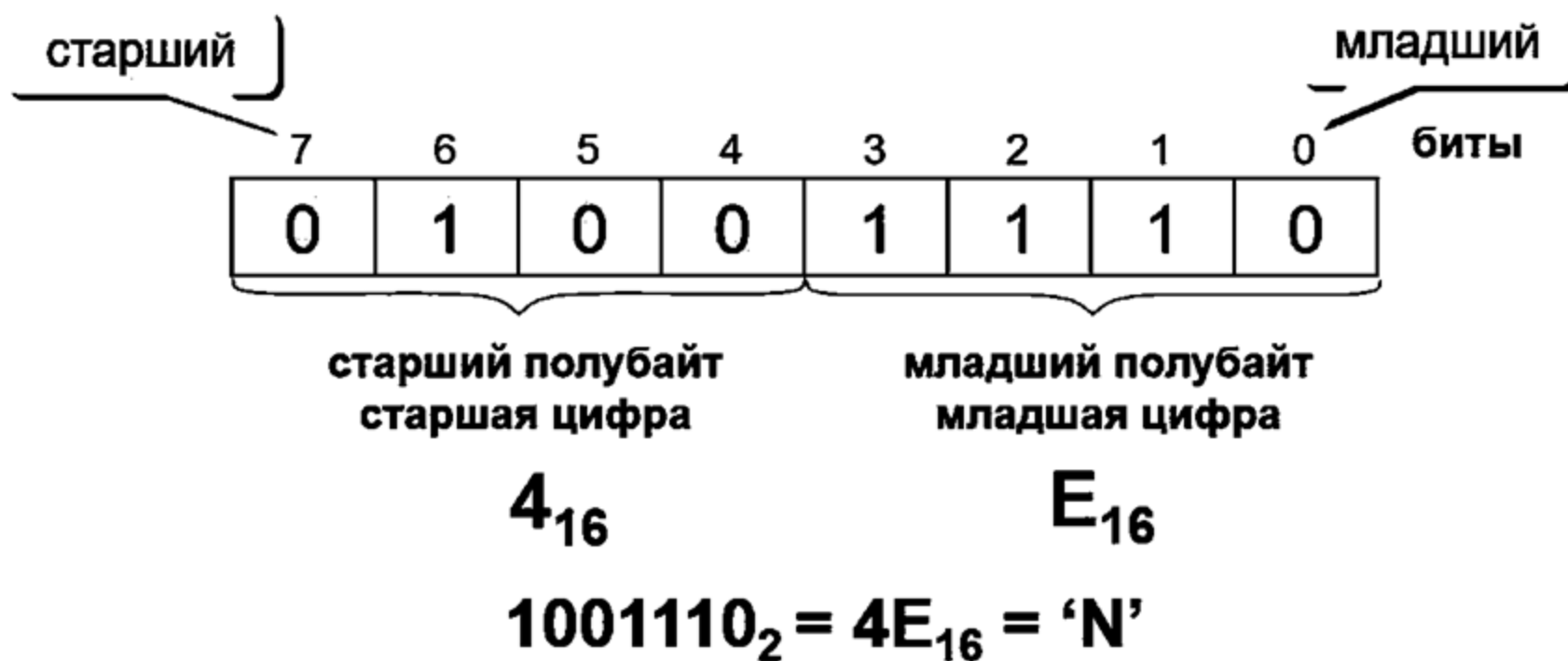
# Представление чисел в компьютере

- Целые знаковые
- Целые беззнаковые
- Вещественные числа

# Целые числа без знака

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

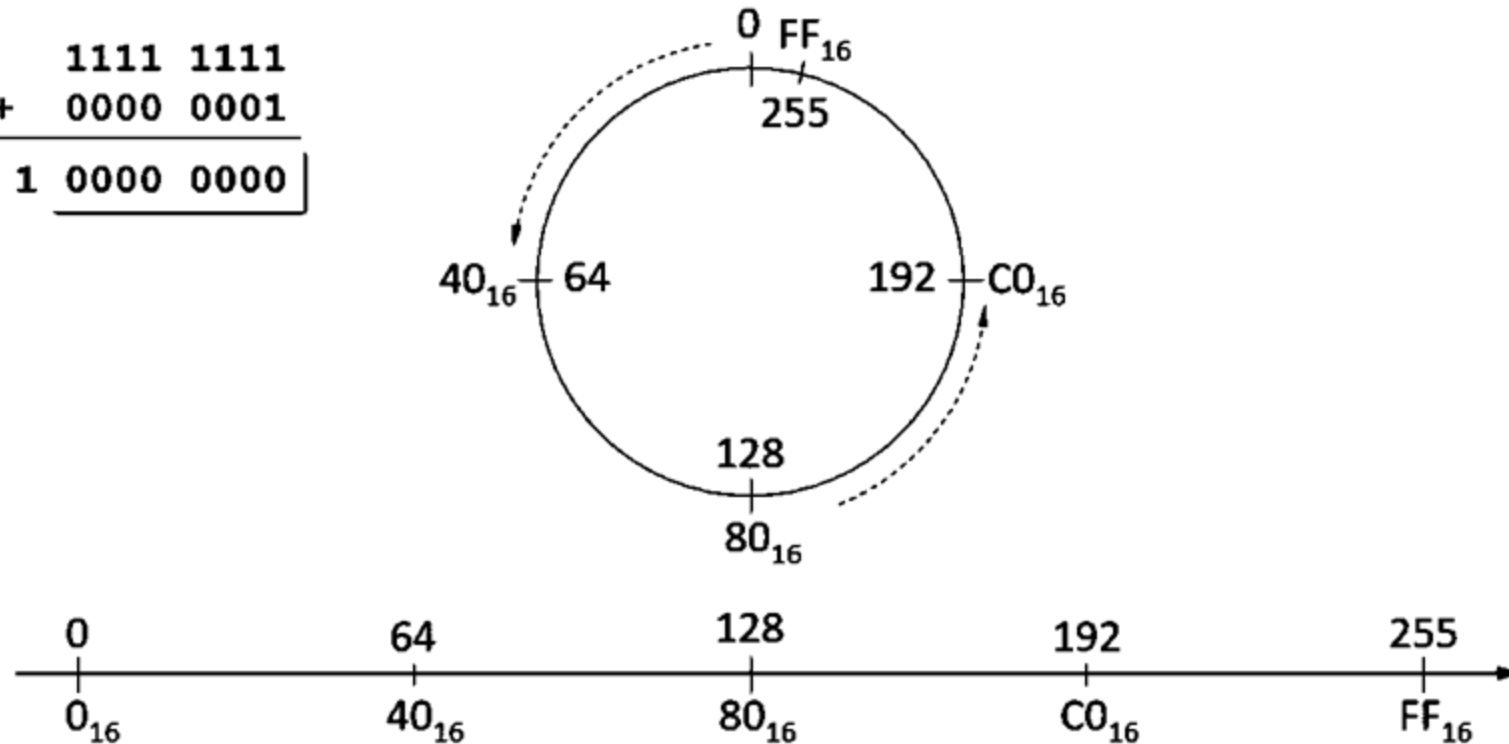
$$78 = 1001110_2$$



# Целые числа без знака

$X_{10}$	0	1	...	127	128	...	255
$X_{16}$	00 <sub>16</sub>	01 <sub>16</sub>	...	7F <sub>16</sub>	80 <sub>16</sub>	...	FF <sub>16</sub>
$X_2$	0000 0000 <sub>2</sub>	0000 0001 <sub>2</sub>	...	0111 1111 <sub>2</sub>	1000 0000 <sub>2</sub>	...	1111 1111 <sub>2</sub>

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



# Целые числа без знака

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

$K$	$X_{\min}$	$X_{\max}$	типы данных
8	0	255	<b>byte</b>
16	0	65 535	<b><u>ushort</u></b>
32	0	4 294 967 295	<b><u>uint</u></b>
64	0	18 446 744 073 709 551 615	<b>ulong</b>

# Представление целых чисел в памяти компьютера

Выбор способа хранения целых чисел в памяти компьютера — не такая тривиальная задача, как могло бы показаться на первый взгляд. Желательно, чтобы этот способ:


- не требовал усложнения архитектуры процессора для выполнения арифметических операций с отрицательными числами,
- не усложнял арифметические действия,
- хранил бы одинаковое количество положительных и отрицательных чисел.

Рассмотрим разные методы представления.

# Прямой код

- При записи числа в **прямом коде** старший разряд является знаковым разрядом. Если его значение равно нулю, то представлено положительное число или положительный ноль, если единице, то представлено отрицательное число или отрицательный ноль. В остальных разрядах (которые называются цифровыми) записывается двоичное представление модуля числа.
- Например, число  $-5$  в восьмибитном типе данных, использующем прямой код, будет выглядеть так: 10000101.

Таким способом в  $n$ -битовом типе данных можно представить диапазон чисел  $[-2^{n-1} + 1; 2^{n-1} + 1]$ .

$2^n - 1$	011...111	
.....	.....	
2	000...010	
1	000...001	
0	000...000	
-0	100...000	
-1	100...001	
-2	100...010	
.....	.....	
$-(2^n - 1)$	111...111	

# Прямой код

$7+9=16$	$7+(-9)=-2$	$-7+(-9)=-16$
$\begin{array}{r} 00000111 \\ + 00001001 \\ \hline 00010000 \end{array} = 16_{10}$	$\begin{array}{r} 00000111 \\ + 10001001 \\ \hline 10010000 \end{array} = -16_{10}$	$\begin{array}{r} 10000111 \\ + 10001001 \\ \hline 00010000 \end{array} = 16_{10}$

$$\begin{aligned} +0 &= \mathbf{00000000} && \text{при } N=8 \\ -0 &= \mathbf{10000000} && \text{при } N=8 \end{aligned}$$



+ VS -

### **Достоинства представления чисел с помощью прямого кода**

- Получить прямой код числа достаточно просто.
- Из-за того, что 0 обозначает +, коды положительных чисел относительно беззнакового кодирования остаются неизменными.
- Количество положительных чисел равно количеству отрицательных.

### **Недостатки представления чисел с помощью прямого кода**

- Выполнение арифметических операций с отрицательными числами требует усложнения архитектуры центрального процессора (например, для вычитания невозможно использовать сумматор, необходима отдельная схема для этого).
- Существуют два нуля:  $+0(100...000)$  и  $-0(000...000)$ , из-за чего усложняется арифметическое сравнение.

Из-за весьма существенных недостатков прямой код используется очень редко.

# Обратный код

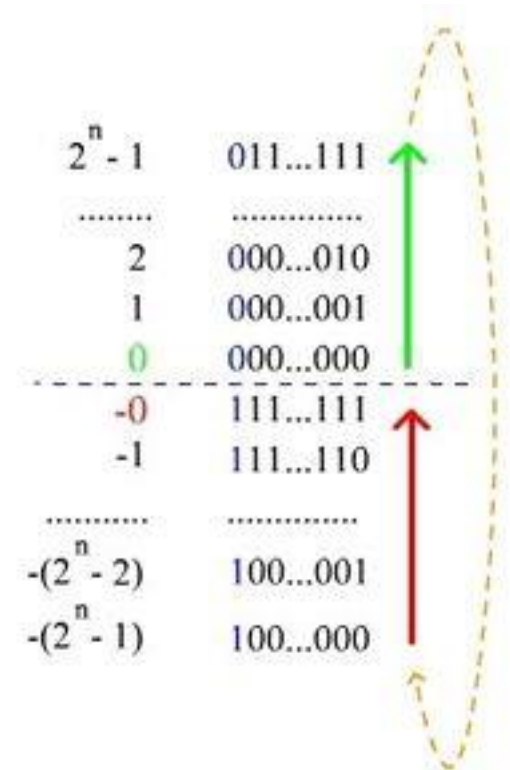
В качестве альтернативы представления целых чисел может использоваться обратный код.

Алгоритм получения кода числа:

- если число положительное, то в старший разряд (который является знаковым) записывается ноль, а далее записывается само число;
- если число отрицательное, то код получается инвертированием представления модуля числа;
- если число является нулем, то его можно представить двумя способами:  $+0(000...000)$  или  $-0(111...111)$ .

Пример: переведём число  $-13$  в двоичный восьмибитный код. Прямой код модуля  $-13$ :  $00001101$ , инвертируем и получаем  $11110010$ . Для получения из обратного кода самого числа достаточно инвертировать все разряды кода.

Таким способом в  $n$ -битовом типе данных можно представить диапазон чисел  $[-2^{n-1} + 1; 2^{n-1} + 1]$ .



# Обратный код

Положительное десятичное число	Двоичное число в обратном коде	Отрицательное десятичное число	Двоичное число в обратном коде
0	0000 0000	- 0	1111 1111
10	0000 1010	- 10	1111 0101
100	0110 0100	- 100	1001 1011
127	0111 1111	- 127	1000 0000

+ VS -

## **Достоинства представления чисел с помощью обратного кода**

- Простое получение кода отрицательных чисел.
- Из-за того, что 0 обозначает +, коды положительных чисел относительно беззнакового кодирования остаются неизменными.
- Количество положительных чисел равно количеству отрицательных.

## **Недостатки представления чисел с помощью обратного кода**

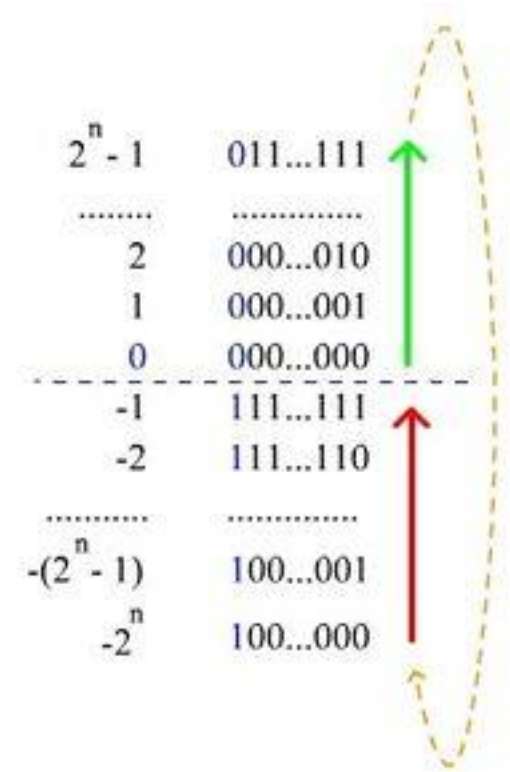
Выполнение арифметических операций с отрицательными числами требует усложнения архитектуры центрального процессора.

- Существуют два нуля: +0 и -0.

# Дополнительный код

- Чаще всего для представления отрицательных чисел используется код с **дополнением**.
- Алгоритм получения дополнительного кода числа:
- если число неотрицательное, то в старший разряд записывается ноль, далее записывается само число;
- если число отрицательное, то все биты модуля числа инвертируются, то есть все единицы меняются на нули, а нули — на единицы, к инвертированному числу прибавляется единица, далее к результату дописывается знаковый разряд, равный единице.

В качестве примера переведем число  $-5$  в дополнительный восьмибитный код. Прямой код модуля  $-5$ :  $0000101$ , обратный —  $1111010$ , прибавляем  $1$ , получаем  $1111011$ , приписываем  $1$  в качестве знакового разряда, в результате получаем  $11111011$ .



- Также дополнительный код отрицательного числа  $A$ , хранящегося в  $n$  битах, равен  $2^n - |A|$ . По сути, дополнительный код представляет собой дополнение  $|A|$  до 0: так как в  $n$ -разрядной арифметике  $2^n = 0$  (двоичная запись этого числа состоит из единицы и  $n$  нулей, а в  $n$ -разрядную ячейку помещаются только  $n$  младших разрядов, то есть  $n$  нулей), то верно равенство  $2^n - |A| + |A| = 0$ .
- Для получения из дополнительного кода самого числа нужно инвертировать все разряды кода и прибавить к нему единицу. Можно проверить правильность, сложив дополнительный код с самим числом: результат должен быть равен  $2^n$ . Переведём 11111011 обратно. Инвертируем — 00000100, прибавляем 1, получаем 00000101 — модуль исходного числа -5. Проверим: 11111011+00000101=100000000.
- Можно получить диапазон значений  $[-2^{n-1}; 2^{n-1} - 1]$ .

# Дополнительный код

Десятичное число	Прямой код	Дополнительный код
12	00001100	00001100
-12	10001100	11110100
121	01111001	01111001
-121	11111001	10000111
1	00000001	00000001
-1	10000001	11111111

$$\begin{array}{r}
 00001100 \\
 + 11110100 \\
 \hline
 100000000
 \end{array}$$

$$100\ 000\ 000_2 - 10\ 000\ 000_2 = 10\ 000\ 000_2 = 128_{10}.$$

$$100000000_2 - 00000000_2 = 100000000_2.$$

+ VS -

- **Достоинства представления чисел с помощью кода с дополнением до двух**
- Возможность заменить арифметическую операцию вычитания операцией сложения и сделать операции сложения одинаковыми для знаковых и беззнаковых типов данных, что существенно упрощает архитектуру процессора и увеличивает его быстродействие.
- Нет проблемы двух нулей.
- **Недостатки представления чисел с помощью кода с дополнением до двух**
- Ряд положительных и отрицательных чисел несимметричен, но это не так важно: с помощью дополнительного кода выполнены гораздо более важные вещи, желаемые от способа представления целых чисел.
- В отличие от сложения, числа в дополнительном коде нельзя сравнивать как беззнаковые, или вычитать без расширения разрядности.

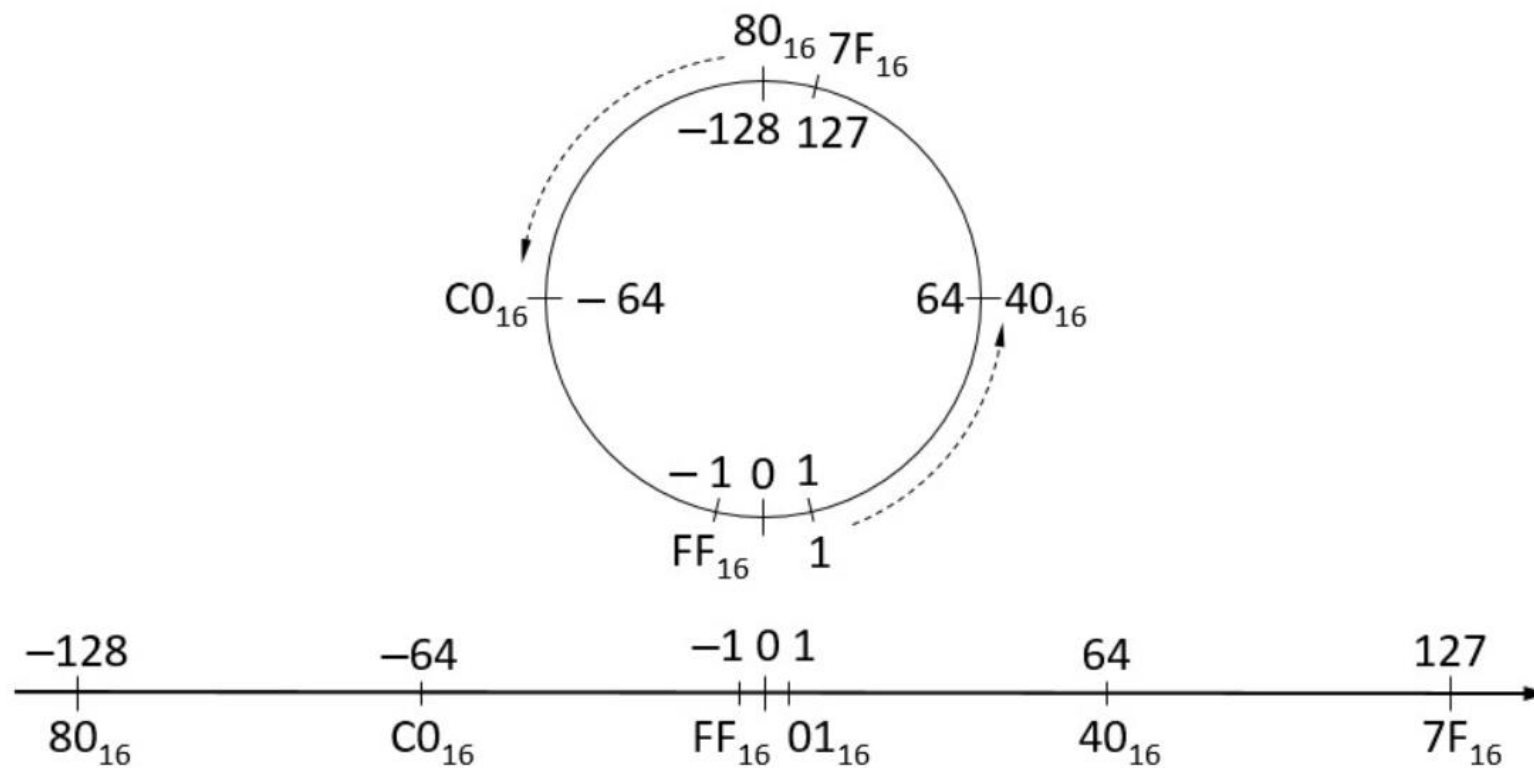
Несмотря на недостатки, дополнение до двух в современных вычислительных системах используется чаще всего.



Десятичное представление	Двоичное представление (8 бит)		
	прямой	обратный	дополнительный
127	0111 1111	0111 1111	0111 1111
1	0000 0001	0000 0001	0000 0001
0	0000 0000	0000 0000	0000 0000
-0	1000 0000	1111 1111	---
-1	1000 0001	1111 1110	1111 1111
-2	1000 0010	1111 1101	1111 1110
-3	1000 0011	1111 1100	1111 1101
-4	1000 0100	1111 1011	1111 1100
-5	1000 0101	1111 1010	1111 1011
-6	1000 0110	1111 1001	1111 1010
-7	1000 0111	1111 1000	1111 1001
-8	1000 1000	1111 0111	1111 1000
-9	1000 1001	1111 0110	1111 0111
-10	1000 1010	1111 0101	1111 0110
-11	1000 1011	1111 0100	1111 0101
-127	1111 1111	1000 0000	1000 0001
-128	---	---	1000 0000

# 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

$$\boxed{X_{\min} = -2^{K-1}} \quad \rightarrow \quad \boxed{X_{\max} = 2^{K-1} - 1}$$

$K$	$X_{\min}$	$X_{\max}$	типы данных
8	-128	127	<u>sbyte</u>
16	-32 768	32 767	<b>short</b>
32	-2 147 483 648	2 147 483 647	<u>int</u>
64	-9 223 372 036 854 775 808	9 223 372 036 854 775 807	<b>long</b>

# Алгоритм перевода отрицательного числа в дополнительный код

- Записать модуль числа в двоичных разрядах прямым кодом
- Инвертировать значения всех битов в полученной записи
- К полученному коду, рассматриваемому как двоичное натуральное число, прибавить 1.

Дано десятичное число -10

Переводим в прямой код:

$10 = 0000\ 1010 \rightarrow -10 = 1000\ 1010$

Инвертируем значение (получаем обратный код):

$1000\ 1010 \rightarrow 1111\ 0101$

К полученной инверсии прибавляем 1:

$1111\ 0101 + 1 = 1111\ 0110$  - десятичное число -10 в дополнительном коде

# Алгоритм перевода дополнительного года отрицательного числа в десятичное число

- Инвертировать дополнительный код;
- Прибавить к полученному коду 1;
- Перевести полученное двоичное число в десятичное и приписать знак “–”.

В результате выполнения следующего фрагмента программы на экран будет выведено

```
sbyte a = 6;
```

```
Console.WriteLine((sbyte)((a << 5)));
```

# Ответ: -64

В результате выполнения следующего фрагмента программы на экран будет выведено

```
sbyte a = 6;
```

```
Console.WriteLine((sbyte)((a << 5)));
```

00000110 = 6

11000000 = 6 << 5 (прямой код)

10111111 = инверсия

11000000 = +1 к числу (доп. код)

Итого получаем -64

# Представление вещественных чисел

$$A = \pm M \cdot P^{\pm N}$$

где  $M$  – мантисса,  $P$  – основание системы счисления,  $N$  – порядок.

Пример:  $A_{10} = 0.0225 \cdot 10^3 = 0.225 \cdot 10^2 = 2.25 \cdot 10^1 = 22.5 \cdot 10^0 = 225 \cdot 10^{-1} = 22.5$

## **Вещественные числа одинарной длины.**

- Длина разрядной сетки – 32 разряда
- Число разрядов мантиссы  $N_M = 23$
- Число разрядов характеристики  $N_Z = 8$
- Знак – 1 разряд
- Правило формирования характеристики:  $Z_2 = 2^7 - 1 + N_2$  или  $Z_{10} = 127_{10} + N_{10}$



# Пример

Выполним кодирование числа  $A_{10} = 0.75$ .

Двоичное представление числа соответствует  $A_2 = 0.11 \cdot 2^0$ .

После нормализации  $A_2 = 1.1 \cdot 2^{-1}$ . Отбрасываем старшую единицу и получаем код мантиссы: 100 0000 0000 0000 0000 0000.

Характеристика  $Z_{10} = 127 + (-1) = 126$  или  $Z_2 = 0111\ 1110$

Знак числа = 0 (+)

После записи компонентов кода в разрядную сетку получим:

0	011	1111	0	100	0000	0000	0000	0000	0000
	<b>3</b> <sub>16</sub>	<b>F</b> <sub>16</sub>		<b>4</b> <sub>16</sub>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>

## Вещественные числа двойной длины.

- Длина разрядной сетки – 64 разряда
- Число разрядов мантиисы  $N_M=52$
- Число разрядов характеристики  $N_Z=11$
- Знак – 1 разряд
- Правило формирования характеристики:  $Z_2 = 2^{10} - 1 + N_2$  или  $Z_{10} = 1023_{10} + N_{10}$

Выполним кодирование числа  $A_{10} = -0.75$ .

Двоичное представление числа соответствует  $A_2 = -0.11 \cdot 2^0$ .

После нормализации  $A_2 = -1.1 \cdot 2^{-1}$ . Отбрасываем старшую единицу и получаем код мантиисы: 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Характеристика  $Z_{10} = 1023 + (-1) = 1022$  или  $Z_2 = 011\ 1111\ 1110$

Знак числа = 1 (–)

После записи компонентов кода в разрядную сетку получим:

1	011 1111 1110	1000 0000 0000 0000 0000 0000 ... 0000 0000 0000
---	---------------	--

или в виде шестнадцатеричного кода **BFE800000000000000**<sub>16</sub>.

# Пример

Код **C0880000**<sub>16</sub> соответствует представлению вещественного числа одинарной длины.

Определить десятичное число, представленное этим кодом.

1. Запишем двоичное представление кода в разрядной сетке

1	100 0000 1	000 1000 0000 0000 0000 0000
<b>C</b> <sub>16</sub>	<b>0</b>	<b>8 8 0 0 0 0</b>

2. Из анализа старшего разряда следует, что это код отрицательного числа

3.  $Z_2 = 10000001$  или  $Z_{10} = 129$ . Отсюда  $N_{10} = Z_{10} - 127$  или  $N_{10} = 2$

4. Восстановим отброшенную при записи кода единицу в записи мантиссы:

$M_2 = 1.000100000000000000000000$  или без учета незначащих нулей

$M_2 = 1.0001$

5. Модуль двоичного числа  $= 1.0001 \cdot 2^2$  или 100.01

6. Модуль десятичного числа  $= 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 4.25$

**Ответ:** код **C0880000**<sub>16</sub> соответствует десятичному числу (−4.25)

# Алгоритм для получения представления действительного числа в памяти ЭВМ

- перевести модуль данного числа в двоичную систему счисления;
- нормализовать двоичное число, т.е. записать в виде  $M \cdot 2^p$ , где  $M$  - мантисса (ее целая часть равна  $1_2$ ) и  $p$  - порядок, записанный в десятичной системе счисления;
- прибавить к порядку смещение и перевести смещенный порядок в двоичную систему счисления;
- учитывая знак заданного числа (0 = положительное; 1 = отрицательное), выписать его представление в памяти ЭВМ.

<https://www.youtube.com/watch?v=1SKsKHOYpJQ>

[https://neerc.ifmo.ru/wiki/index.php?title=Представление вещественных чисел](https://neerc.ifmo.ru/wiki/index.php?title=Представление_вещественных_чисел)

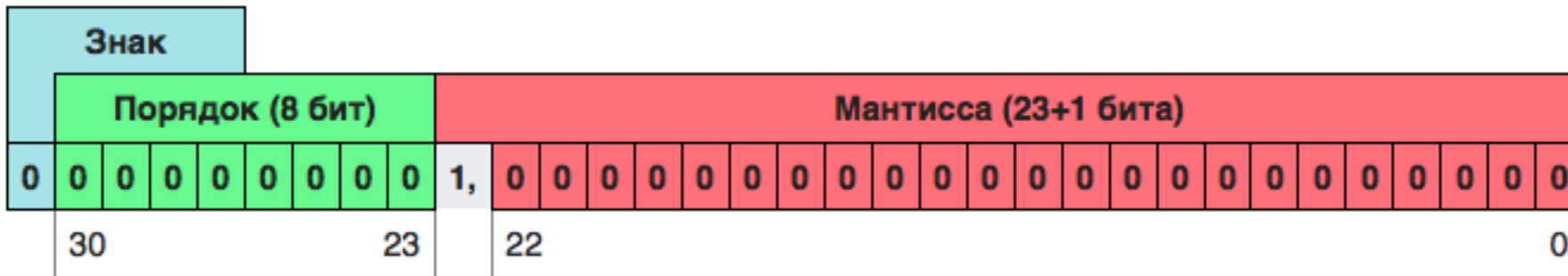
$$-121_{10} = -79_{16} = -111\ 1001_2$$

$$0.1_{10} = 0.0(0011)_2$$

$$A = -111\ 1001.0(0011) = -1.1110010(0011) \cdot 2^6$$

$$E = 6 + 127 = 133 = 10000101_2$$

$$M = 1110010(0011)$$



Порядок записан со сдвигом  $-127$ .

0	0.1
	2
0	0.2
	2
0	0.4
	2
0	0.8
	2
1	1.6
	2
1	1.2

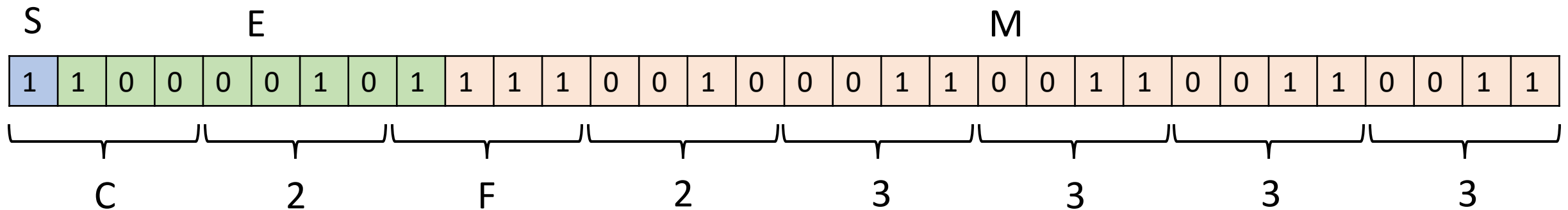
$$-121_{10} = -79_{16} = -111\ 1001_2$$

$$0.1_{10} = 0.0(0011)_2$$

$$A = -111\ 1001.0(0011) = -1.1110010(0011) \cdot 2^6$$

$$E = 6 + 127 = 133 = 10000101_2$$

$$M = 1110010(0011)$$



**Ответ: C2F23333**

# Ноль (со знаком)

Как уже было оговорено выше, в нормализованной форме числа с плавающей точкой невозможно представить ноль. Поэтому для его представления зарезервированы специальные значения мантиссы и порядка — число считается нулём, если все его биты, кроме знакового, равны нулю. При этом в зависимости от значения бита знака ноль может быть как положительным, так и отрицательным.



## Арифметика нуля со знаком

Арифметика отрицательного нуля аналогична таковой для любого отрицательного числа и понятна интуитивно. Вот несколько примеров:

- $\frac{-0}{|x|} = -0$  (если  $x \neq 0$ )
- $(-0) \cdot (-0) = +0$
- $|x| \cdot (-0) = -0$
- $x + (\pm 0) = x$
- $(-0) + (-0) = -0$
- $(+0) + (+0) = +0$
- $\frac{-0}{-\infty} = +0$
- $\frac{|x|}{-0} = -\infty$  (если  $x \neq 0$ )

# Неопределенность (NaN)

**NaN** — это аббревиатура от фразы "*not a number*". NaN является результатом арифметических операций, если во время их выполнения произошла ошибка (примеры см. ниже). В IEEE 754 NaN представлен как число, в котором все двоичные разряды порядка — единицы, а мантисса не нулевая.

Знак																	
		Порядок					Мантисса										
0/1	1	1	1	1	1	1,	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	= NaN
	14				10		9									0	

Любая операция с NaN возвращает NaN. При желании в мантиссу можно записывать информацию, которую программа сможет интерпретировать. Стандартом это не оговорено и мантисса чаще всего игнорируется.

## Как можно получить NaN?

- $\infty + (-\infty) = NaN$
- $0 \times \infty = NaN$
- $\frac{\pm 0}{\pm 0} = \frac{\pm \infty}{\pm \infty} = NaN$
- $\sqrt{x} = NaN$ , где  $x < 0$



## Бесконечности

В число с плавающей запятой можно записать значение  $+\infty$  или  $-\infty$ . Как и нули со знаком, бесконечности позволяют получить хотя бы близкий к правильному результат вычисления в случае переполнения. Согласно стандарту IEEE 754 число с плавающей запятой считается равным бесконечности, если все двоичные разряды его порядка — единицы, а мантисса равна нулю. Знак бесконечности определяется знаковым битом числа.

Знак																	
		Порядок					Мантисса										
0/1	1	1	1	1	1	1,	0	0	0	0	0	0	0	0	0	0	= ±∞
	14				10		9									0	

# Поразрядная сортировка для данных с плавающей запятой

Старший бит типов с плавающей точкой кодирует знак числа, поэтому сортировка по старшему биту должна выполняться в обратном порядке (1 “меньше” 0).

Для положительных чисел большее значение порядка соответствует большему числу, т.к:

- в случае нормализованного представления неявный старший бит мантиссы равен 1;

$$x = \pm 0.\overset{\text{Мантисса}}{\underset{\text{Мантисса}}{1}bbbb..bbb \times 2^{\underset{\text{Порядок}}{E}}$$

Знак числа

- в случае денормализованного представления значение поля соответствует минимальному, а неявный старший бит мантиссы равен 0.

# Поразрядная сортировка для данных с плавающей запятой

Для положительных чисел в случае равенства порядка, большее значение мантиссы соответствует большему числу. Таким образом, побайтовая сортировка положительных чисел допустима и по всем байтам должна выполняться в обычном порядке (0 “меньше” 1).

Для отрицательных чисел большее значение порядка соответствует меньшему числу (рассуждения аналогичны рассуждениям о положительных числах). Таким образом, сортировка отрицательных чисел по всем байтам должна выполняться в обратном порядке (1 “меньше” 0).

Общий алгоритм сортировки типов с плавающей точкой:

- сортировка чисел по старшему биту, разделяя числа на две группы: отрицательные и положительные;
- для положительных чисел сортировка по всем байтам выполняется в обычном порядке (0 “меньше” 1);
- для отрицательных чисел сортировка по всем байтам выполняется в обратном порядке (1 “меньше” 0).

# КОНТАКТЫ



**LinkedIn**

<https://ru.linkedin.com/in/mariia-gordenko-78617618b>



**Telegram, Instagram**

oduvan\_ja



**Адрес электронной почты**

[mgordenko@hse.ru](mailto:mgordenko@hse.ru) [mkgordenko@gmail.com](mailto:mkgordenko@gmail.com)



**Личная страница**

<https://www.hse.ru/staff/gordenko>