

Представление данных

Основы информатики.

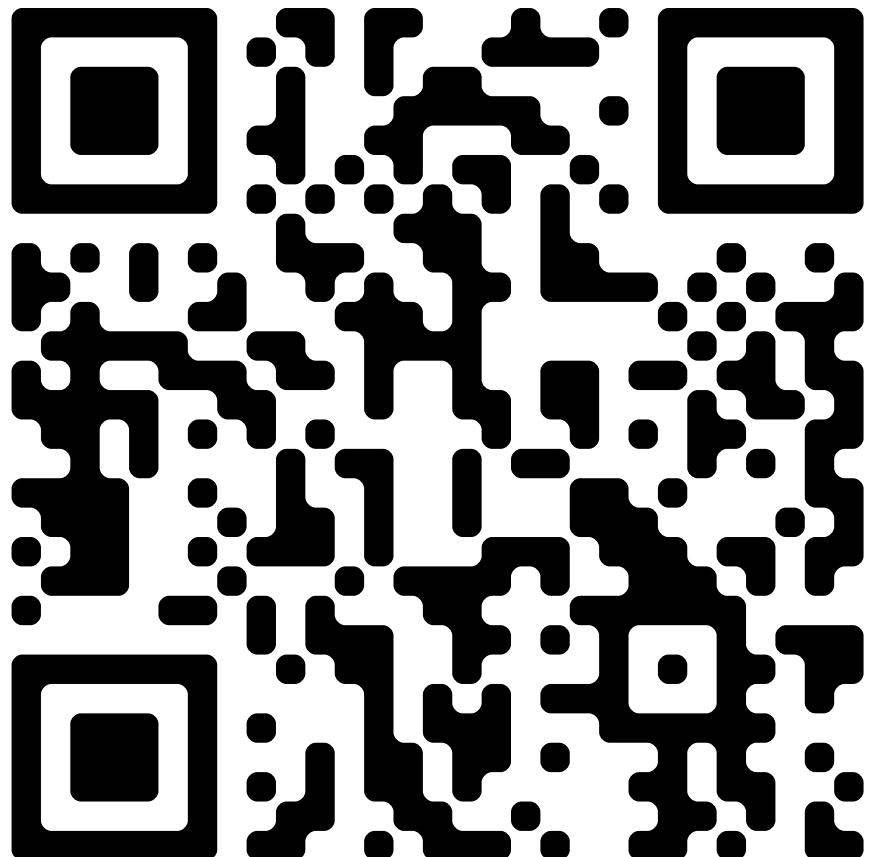
Компьютерные основы программирования

goo.gl/X7evF

На основе:

Introduction to Computer Systems

goo.gl/Q7vgWw



Лекции 2 и 3, 13 и 20 февраля 2017

Лектор:

Дмитрий Северов, кафедра информатики 608 КПМ

dseverov@mail.mipt.ru

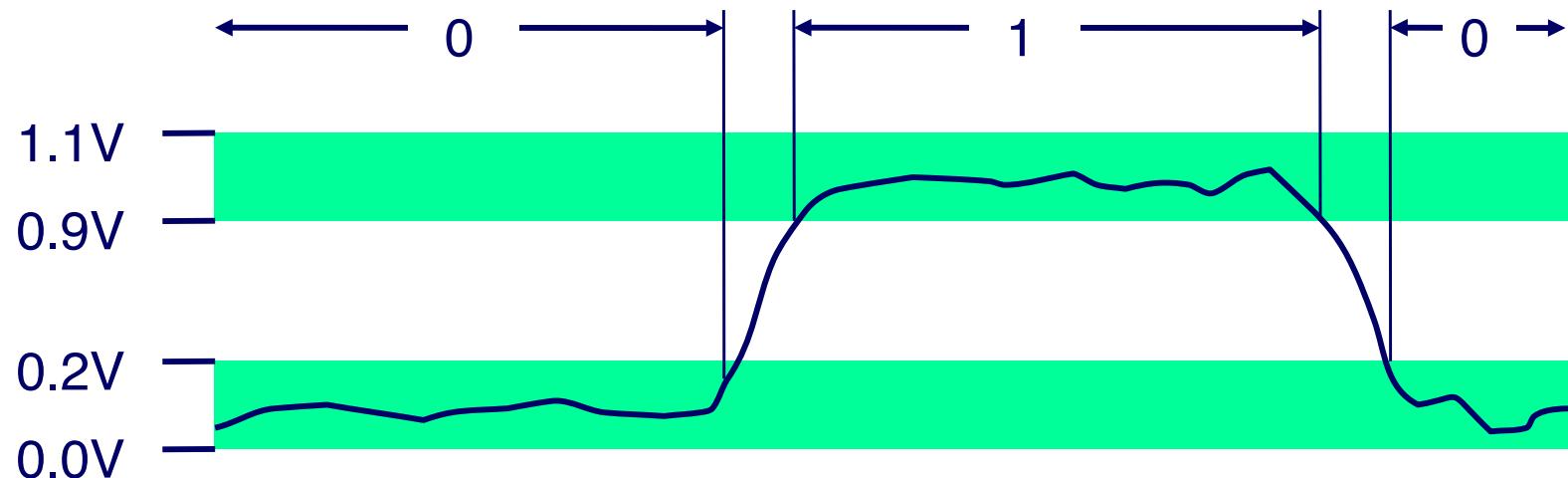
w27001.vdi.mipt.ru/wp/?page_id=346

Биты, байты и целые

- Представление информации в битах
- Манипуляции на уровне бит
- Целые
 - Представление: беззнаковое и знаковое
 - Преобразования
 - Расширение, сокращение
 - Сложение, отрицание, умножение, сдвиг
 - Сводка
- Представление в памяти, указатели, строки

(почти) Всё есть биты

- Каждый бит есть 0 или 1
- По разному кодируя/интерпретируя наборы бит
 - Компьютеры определяют действия (инструкции)
 - ... представляют, и обрабатывают числа, множества, строки...
- Почему биты? Из-за электронной реализации
 - Просто хранить с помощью бистабильных элементов
 - Надёжно передаются по плохим и зашумленным проводникам



Двоичный пример

■ Представление чисел в двоичной системе

- $15213_{10} = 11101101101101_2$
- $1.20_{10} = 1.0011001100110011[0011]\dots_2$
- $1.5213 \times 10^4 = 1.1101101101101_2 \times 2^{13}$

Кодирование значений байта

■ Байт = 8 бит

- Двоичное от 00000000_2 до 11111111_2
- Десятичное: от 0_{10} до 255_{10}
- Шестнадцатиричное от 00_{16} до FF_{16}
 - Представление 16 цифр
 - Используем символы от ‘0’ до ‘9’ и от ‘A’ до ‘F’
 - Запись FA1D37B₁₆ в Си как
 - 0xFA1D37B
 - 0xfa1d37b

		Шестнадцатиричное
		Десятичное
		Двоичное
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Пример представления данных

Тип данных языка С	32-бит типично	64-bit типично	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	8	8
float	4	4	4
double	8	8	8
long double	—	—	10/16
указатель	4	8	8

Биты, байты и целые

- Представление информации в битах
- Манипуляции на уровне бит
- Целые
 - Представление: беззнаковое и знаковое
 - Преобразования
 - Расширение, сокращение
 - Сложение, отрицание, умножение, сдвиг
 - Сводка
- Представление в памяти, указатели, строки

Булева алгебра

■ Предложена Джорджем Булем в XIX веке

- Алгебраическое представление одной из логик
 - Кодирует “Истина” как 1 и “Ложь” как 0

И (And)

- $A \& B = 1$ когда оба $A=1$ and $B=1$

&	0	1
0	0	0
1	0	1

ИЛИ (Or)

- $A | B = 1$ когда либо $A=1$, либо $B=1$

	0	1
0	0	1
1	1	1

НЕ(Not)

- $\sim A = 1$ when $A=0$

\sim	
0	1
1	0

Исключающее ИЛИ (Xor)

- $A ^ B = 1$ когда либо $A=1$, либо $B=1$, но не оба

$^$	0	1
0	0	1
1	1	0

Обобщение булевой алгебры

■ Операции на битовых наборах (векторах)

- Операции выполняются побитово

$$\begin{array}{r} 01101001 \\ \& 01010101 \\ \hline 01000001 \end{array} \quad \begin{array}{r} 01101001 \\ | 01010101 \\ \hline 01111101 \end{array} \quad \begin{array}{r} 01101001 \\ ^ 01010101 \\ \hline 00111100 \end{array} \quad \begin{array}{r} ~ 01010101 \\ \sim 10101010 \\ \hline 10101010 \end{array}$$

■ Применимы все выводы булевой алгебры

Представление и операции с множествами

■ Представление

- Вектор бит размером w представляет подмножество $\{0, \dots, w-1\}$
- $a_j = 1$ если $j \in A$

▪ 01101001 $\{0, 3, 5, 6\}$

▪ **76543210**

▪ 01010101 $\{0, 2, 4, 6\}$

▪ **76543210**

■ Операции

▪ & Пересечение	01000001	$\{0, 6\}$
▪ Объединение	01111101	$\{0, 2, 3, 4, 5, 6\}$
▪ ^ Разность	00111100	$\{2, 3, 4, 5\}$
▪ ~ Дополнение	10101010	$\{1, 3, 5, 7\}$

Побитовые операции в языке Си

■ Операции &, |, ~, ^ доступные в Си

- Применимы к любому “целостному” типу данных
 - long, int, short, char, unsigned
- Аргументы рассматриваются как вектора битов
- Каждый бит – независимый аргумент

■ Примеры (тип данных char)

- $\sim 0x41 \rightarrow 0xBE$
 - $\sim 01000001_2 \rightarrow 10111110_2$
- $\sim 0x00 \rightarrow 0xFF$
 - $\sim 00000000_2 \rightarrow 11111111_2$
- $0x69 \& 0x55 \rightarrow 0x41$
 - $01101001_2 \& 01010101_2 \rightarrow 01000001_2$
- $0x69 | 0x55 \rightarrow 0x7D$
 - $01101001_2 | 01010101_2 \rightarrow 01111101_2$

Сравните: логические операции в С

■ Логические операторы

- `&&`, `||`, `!`
 - 0 кодирует “False”
 - Всё, что не 0 кодирует “True”
 - Всегда выдаёт 0 или 1
 - Раннее завершение вычисления выражения

■ Примеры (тип данных `char`)

- `!0x41` → `0x00`
- `!0x00` → `0x01`
- `!!0x41` → `0x01`

- `0x69 && 0x55` → `0x01`
- `0x69 || 0x55` → `0x01`
- `p && *p` (способ избежать обращения по нулевому указателю)

Сравните: логические операции в С

■ Логические операторы

- `&&`, `||`,
 - 0 кодирует ложь
 - Всё, что не 0 — истина

Внимание!

- Проблема:
- `&&` вместо `&` (и `||` вместо `|`)...
 - одна из самых частых ошибок
 - !
 - !

- `0x69 && 0x55` → `0x01`
- `0x69 || 0x55` → `0x01`
- `p && *p` (способ избежать обращения по нулевому указателю)

Операции сдвига в Си

■ Сдвиг влево: $x \ll y$

- Сдвигает вектор битов X влево на Y позиций
 - Вытолкнутые слева биты теряются
 - Заполняет нулями справа

■ Сдвиг вправо: $x \gg y$

- Сдвигает вектор битов X вправо на Y позиций
 - Вытолкнутые справа биты теряются
- Логический сдвиг
 - Заполняет нулями справа
- Арифметический сдвиг
 - Повторяет вправо наиболее значимый бит

■ Неопределённый результат

- Сдвиг на величину меньше 0 или больше размера слова

Аргумент x	01100010
$\ll 3$	00010000
Логич. $\gg 2$	00011000
Ариф. $\gg 2$	00011000

Аргумент x	10100010
$\ll 3$	00010000
Логич. $\gg 2$	00101000
Ариф. $\gg 2$	11101000

Биты, байты и целые

- Представление информации в битах
- Манипуляции на уровне бит
- Целые
 - Представление: беззнаковое и знаковое
 - Преобразования
 - Расширение, сокращение
 - Сложение, отрицание, умножение, сдвиг
 - Сводка
- Представление в памяти, указатели, строки

Кодирование целочисленных значений

Беззнаковых

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

В дополнительном коде

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

```
short int x = 15213;  
short int y = -15213;
```

знаковый
бит

■ Си `short` длиной в 2 байта

	Десятичное	Шестнадцатиричное	Двоичное
<code>x</code>	15213	ЗВ 6D	00111011 01101101
<code>y</code>	-15213	C4 93	11000100 10010011

■ Знаковый бит

- В дополнительном коде, наиболее значимый бит обозначает знак
 - 0 для неотрицательных
 - 1 для отрицательных

Пример кодирования (продолжение)

x =	15213:	00111011	01101101
y =	-15213:	11000100	10010011

Вес	15213		-15213	
1	1	1	1	1
2	0	0	1	2
4	1	4	0	0
8	1	8	0	0
16	0	0	1	16
32	1	32	0	0
64	1	64	0	0
128	0	0	1	128
256	1	256	0	0
512	1	512	0	0
1024	0	0	1	1024
2048	1	2048	0	0
4096	1	4096	0	0
8192	1	8192	0	0
16384	0	0	1	16384
-32768	0	0	1	-32768
Итого:		15213	-15213	

Границы представления в W бит

■ Беззнаковые значения

- $UMin = 0$
000...0
- $UMax = 2^w - 1$
111...1

■ Значения в дополнительном коде

- $TMin = -2^{w-1}$
100...0
- $TMax = 2^{w-1} - 1$
011...1

■ Другие значения

- минус 1
111...1

Значения для $W = 16$

	Десятичные	Шестнадцатиричные	Двоичные
UMax	65535	FF FF	11111111 11111111
TMax	32767	7F FF	01111111 11111111
TMin	-32768	80 00	10000000 00000000
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000

Значения для разных размеров слова

	W			
	8	16	32	64
UMax	255	65,535	4,294,967,295	18,446,744,073,709,551,615
TMax	127	32,767	2,147,483,647	9,223,372,036,854,775,807
TMin	-128	-32,768	-2,147,483,648	-9,223,372,036,854,775,808

■ Важно!

- $|TMin| = TMax + 1$
 - Границы несимметричны
- $UMax = 2 * TMax + 1$

■ Программирование на Си

- `#include <limits.h>`
- Объявленные константы, e.g.,
 - `ULONG_MAX`
 - `LONG_MAX`
 - `LONG_MIN`
- Значения констант зависят от платформы

Беззнаковые и знаковые величины

x	$B2U(x)$	$B2T(x)$
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Совпадение

- Однаковые кодировки неотрицательных величин

Взаимная однозначность

- Каждая комбинация бит представляет своё значение
- Каждое представимое целое имеет уникальную кодировку

Отображение обратимо

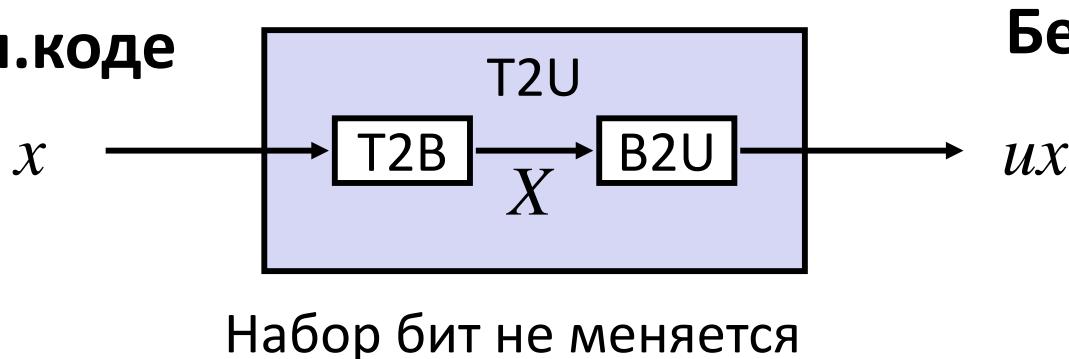
- $U2B(x) = B2U^{-1}(x)$
 - наборы бит беззнаковых
- $T2B(x) = B2T^{-1}(x)$
 - наборы бит знаковых в дополнительном коде

Биты, байты и целые

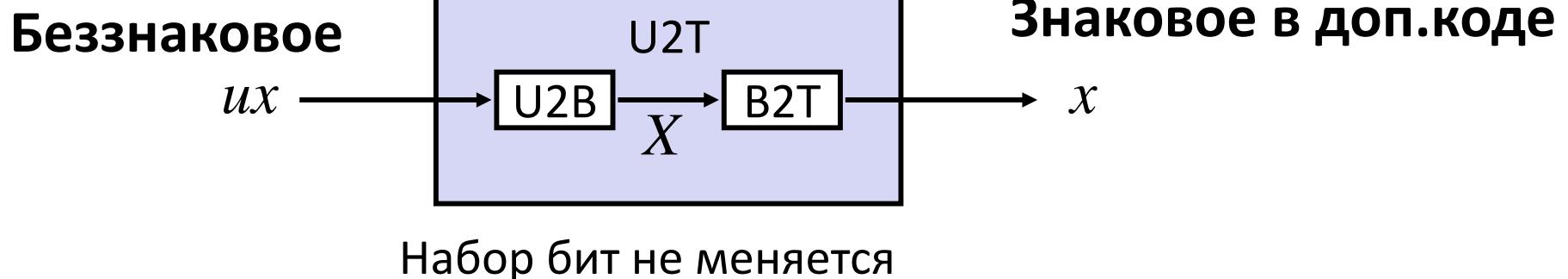
- Представление информации в битах
- Манипуляции на уровне бит
- Целые
 - Представление: беззнаковое и знаковое
 - Преобразования
 - Расширение, сокращение
 - Сложение, отрицание, умножение, сдвиг
 - Сводка
- Представление в памяти, указатели, строки

Соответствие знаковых и беззнаковых

Знаковое в доп.коде



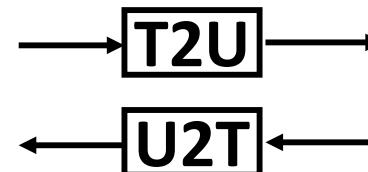
Беззнаковое



- Соответствие знаковых в доп.коде и беззнаковых:
Сохраняем битовое представление и переинтерпретируем

Соответствие знаковых и беззнаковых

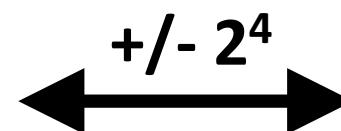
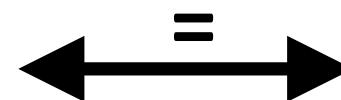
Биты	Знаковое	Беззнаковое
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15



Соответствие знаковых и беззнаковых

Биты
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

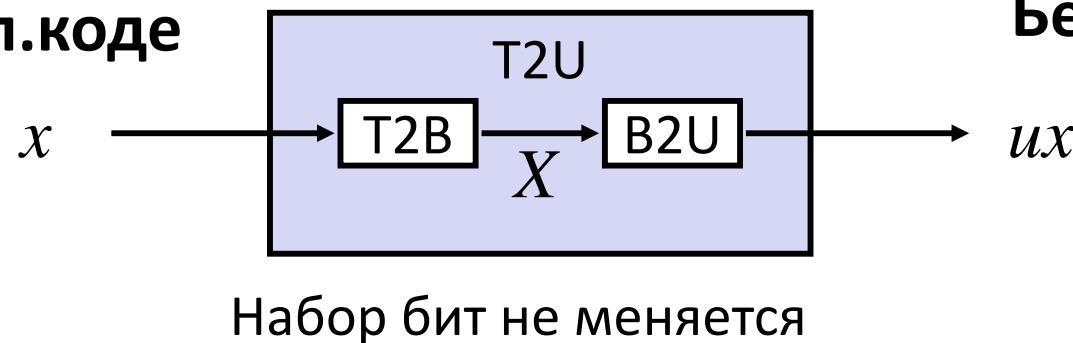
Знаковое
0
1
2
3
4
5
6
7
-8
-7
-6
-5
-4
-3
-2
-1



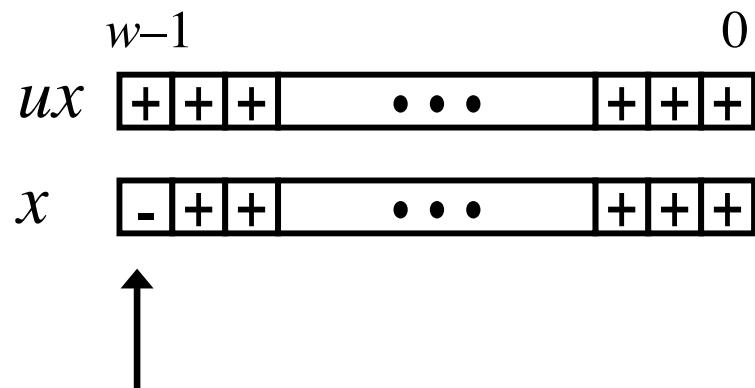
Беззнаковое
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Связь между знаковыми и беззнаковыми

Знаковое в доп.коде



Беззнаковое

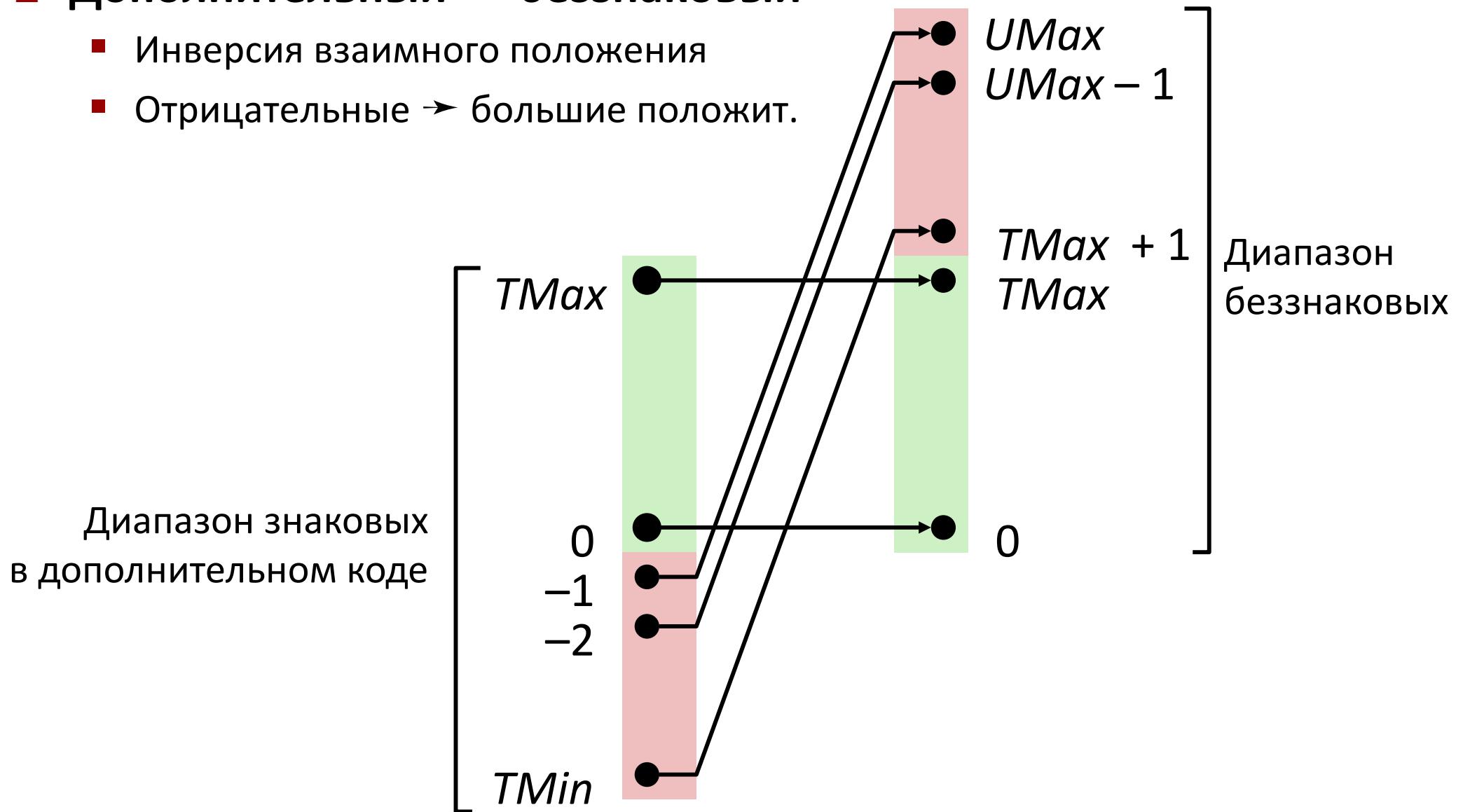


самый отрицательный бит
одиноко становится
самым положительным

Визуализация преобразования

■ Дополнительный \rightarrow беззнаковый

- Инверсия взаимного положения
- Отрицательные \rightarrow большие положит.



Знаковые и беззнаковые в Си

■ Константы

- По умолчанию целочисленные десятичные считаются знаковыми
- Беззнаковые обозначаются суффиксом “U”

`0U, 4294967259U`

■ Преобразование

- Явное

```
int tx, ty;  
unsigned ux, uy;  
tx = (int) ux;  
uy = (unsigned) ty;
```

- Неявное преобразование также происходит в присвоениях и вызовах процедур

```
tx = ux;  
uy = ty;
```

Неожиданные преобразования

■ Вычисления выражений

- При смешивании знаковых и беззнаковых,
знаковые неявно преобразуются в беззнаковые
- Включая операции сравнения $<$, $>$, $==$, $<=$, $>=$
- Пример для $W = 32$: **TMIN = -2,147,483,648**, **TMAX = 2,147,483,647**

■ Константа ₁	Константа ₂	Итог	Тип рез-та
0	0U	==	unsigned
-1	0	<	signed
-1	0U	>	unsigned
2147483647	-2147483647-1	>	signed
2147483647U	-2147483647-1	<	unsigned
-1	-2	>	signed
(unsigned)-1	-2	>	unsigned
2147483647	2147483648U	<	unsigned
2147483647	(int) 2147483648U	>	signed

Сводка: преобразование Знаковые \leftrightarrow Беззнаковые: правила

- Битовые последовательности сохраняются...
- ... но интерпретируются по разному
- Возможная неожиданность: добавление или вычитание 2^w

- В выражениях содержащих `signed int` и `unsigned int`
 - [signed] преобразуются в `unsigned !!`

Биты, байты и целые

- Представление информации в битах
- Манипуляции на уровне бит
- Целые
 - Представление: беззнаковое и знаковое
 - Преобразования
 - Расширение, сокращение
 - Сложение, отрицание, умножение, сдвиг
 - Сводка
- Представление в памяти, указатели, строки

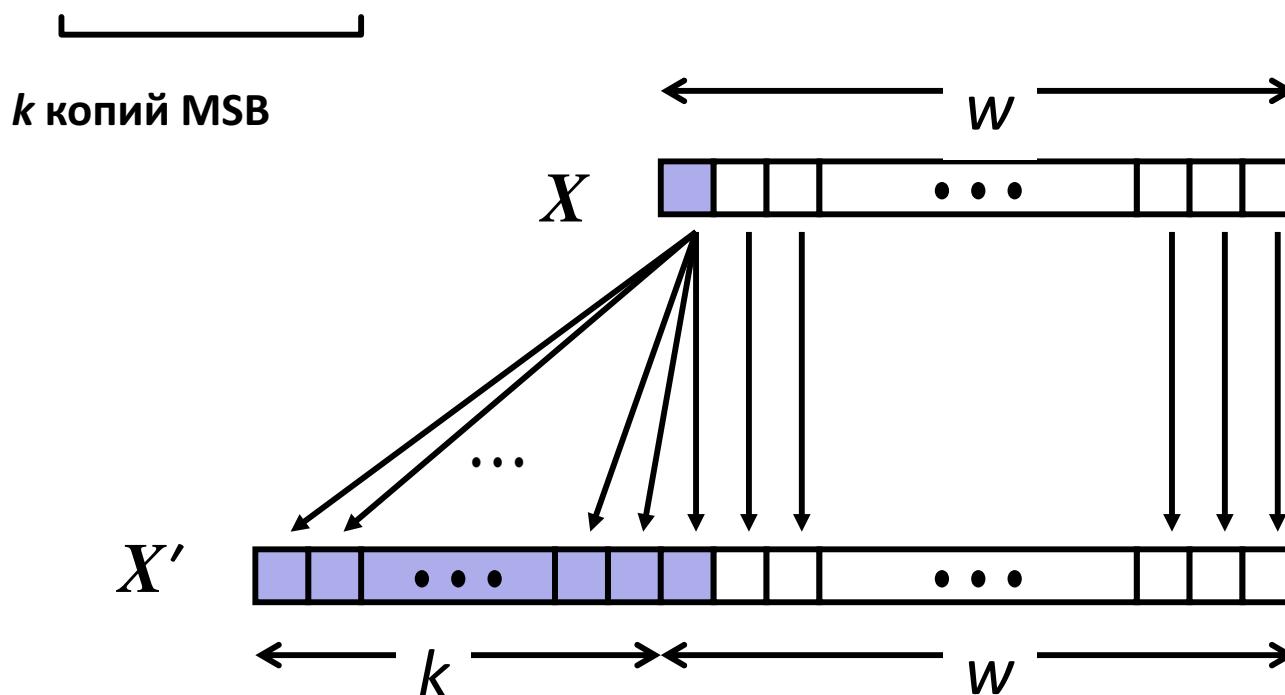
Расширение знака

■ Задача:

- Дано знаковое целое x размером w бит
- Преобразовать в целое того же значения размером $w+k$ бит

■ Правило:

- Сделать k копий знакового бита:
- $X' = x_{w-1}, \dots, x_{w-1}, x_{w-1}, x_{w-2}, \dots, x_0$



Пример расширения знака

```
short int x = 15213;  
int ix = (int) x;  
short int y = -15213;  
int iy = (int) y;
```

	Десятич.	Шестнадцатир.	Двоичное
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 3B 6D	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

- Преобразование из короткого в длинный целочисленный тип
- Си автоматически выполняет расширение знака

Сводка:

Расширение, усечение: правила

■ Расширение (например, short int в int)

- Беззнаковое: добавить нули
- Знаковое: расширить знак
- Оба действия дают ожидаемый результат

■ Усечение (например, unsigned в unsigned short)

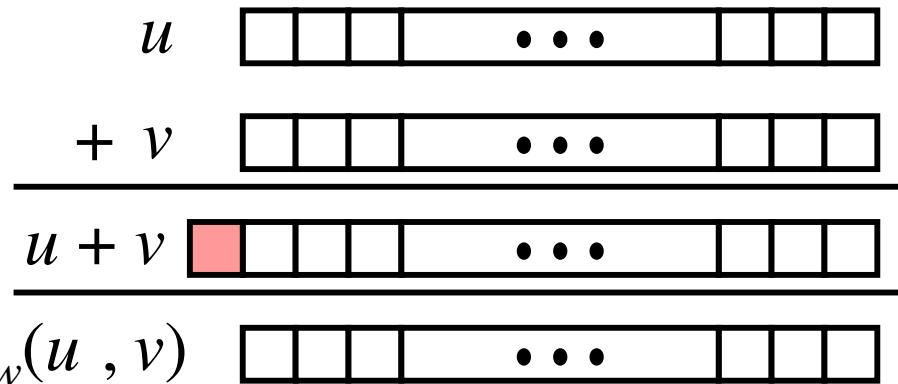
- Unsigned/signed: биты отбрасываются
- Результат переинтерпретируется для новой длины
- Unsigned: операция «остаток от деления»
- Signed: операция похожая на «остаток от деления»
- Ожидаемый результат – только для малых значений.

Биты, байты и целые

- Представление информации в битах
- Манипуляции на уровне бит
- Целые
 - Представление: беззнаковое и знаковое
 - Преобразования
 - Расширение, сокращение
 - Сложение, отрицание, умножение, сдвиг
 - Сводка
- Представление в памяти, указатели, строки

Беззнаковое сложение

Операнды: w бит



Полная сумма: $w+1$ бит

Без переноса: w бит

- Стандартная операция сложения
 - «Игнорирует» перенос
- Реализует сложение по модулю

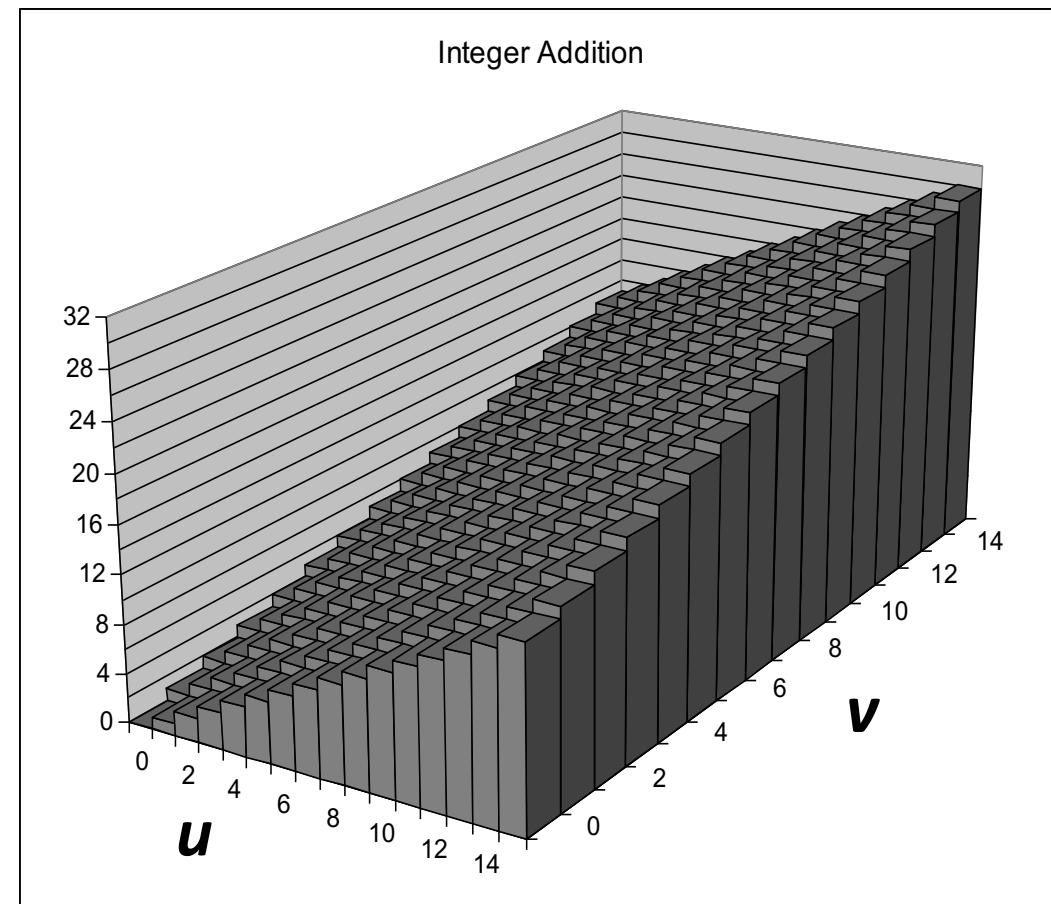
$$s = \text{UAdd}_w(u, v) = u + v \bmod 2^w$$

Математическое сложение визуально

■ Сложение целых

- 4-х битовые целые u , v
- Полная сумма $\text{Add}_4(u, v)$
- Значение растёт линейно по u и v
- Образует плоскость

$\text{Add}_4(u, v)$

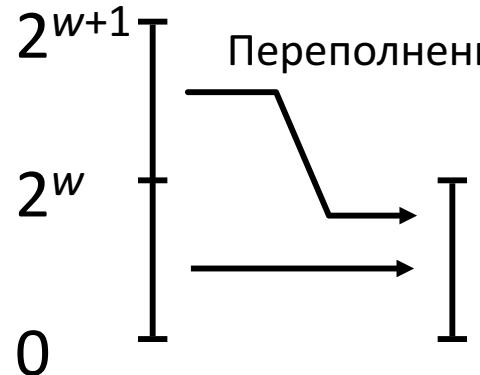


Беззнаковое сложение визуально

■ Усечение

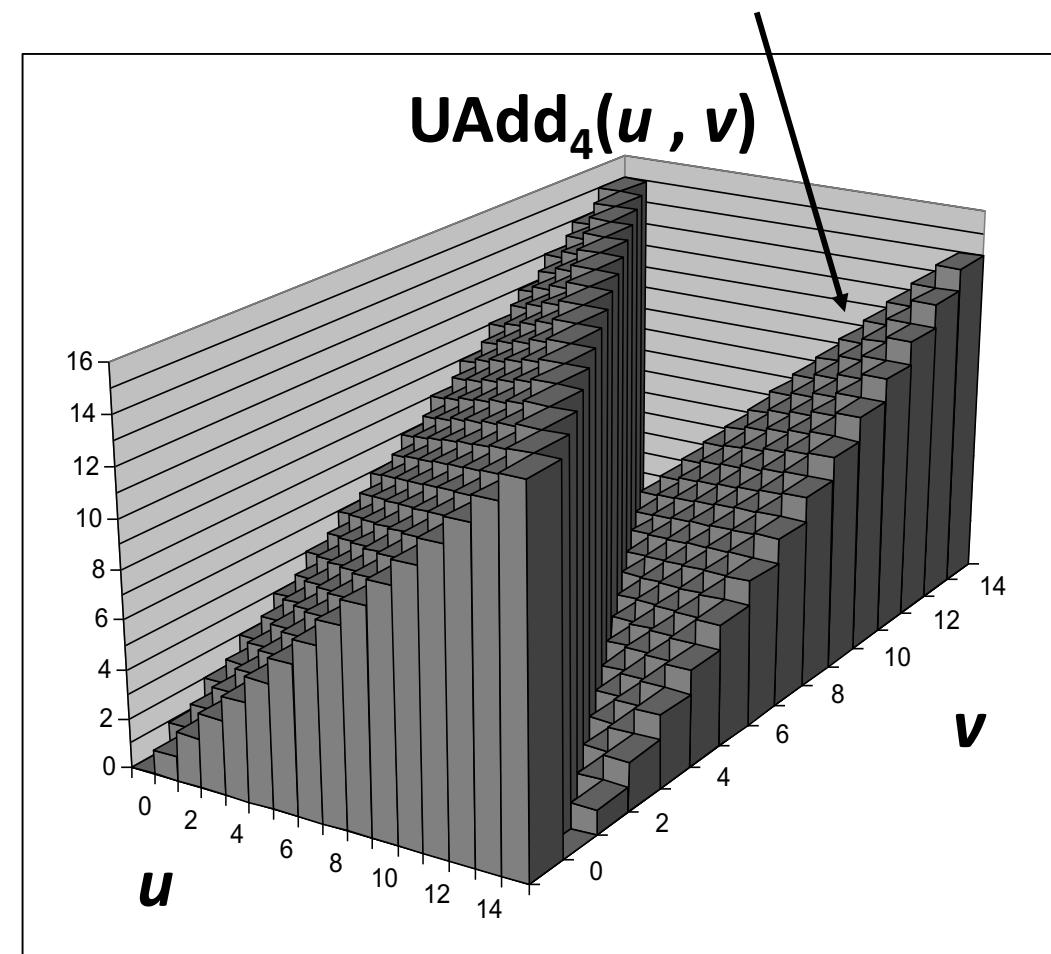
- Если полная сумма $\geq 2^w$
- Не более одного раза

Полная сумма



Сумма по модулю

Переполнение



Сложение в дополнительном коде

Операнды: w бит

u \dots

Полная сумма: $w+1$,бит

$+ v$ \dots

$u + v$ \dots

Без переноса: w бит

$TAdd_w(u, v)$ \dots

■ $TAdd$ и $UAdd$ преобразуют биты одинаково

- Знаковое и беззнаковое сложение в Си:

```
int s, t, u, v;  
s = (int) ((unsigned) u + (unsigned) v);  
t = u + v
```

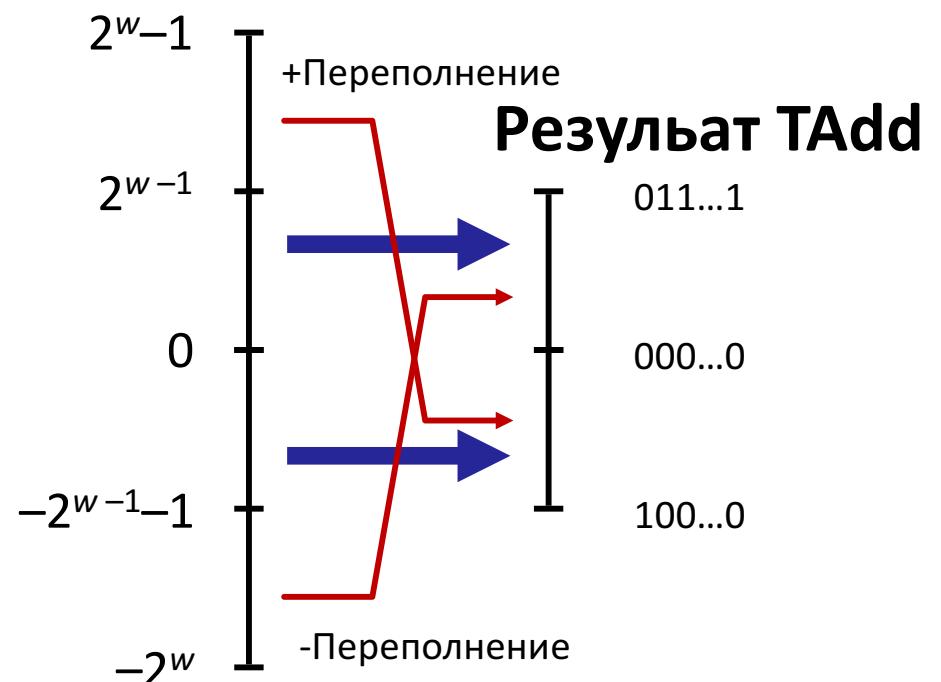
- Всегда даст $s == t$

Переполнение TAdd

■ Функциональность

- Полное суммирование требует $w+1$ бит
 - Отбрасывание MSB
 - Интерпретация оставшихся бит как целого числа в дополнительном коде
- | | | | | |
|-----------|-----------|-----------|-----------|-----------|
| 0 111...1 | 0 100...0 | 0 000...0 | 1 011...1 | 1 000...0 |
|-----------|-----------|-----------|-----------|-----------|

Полная сумма



Сложение в доп. коде визуально

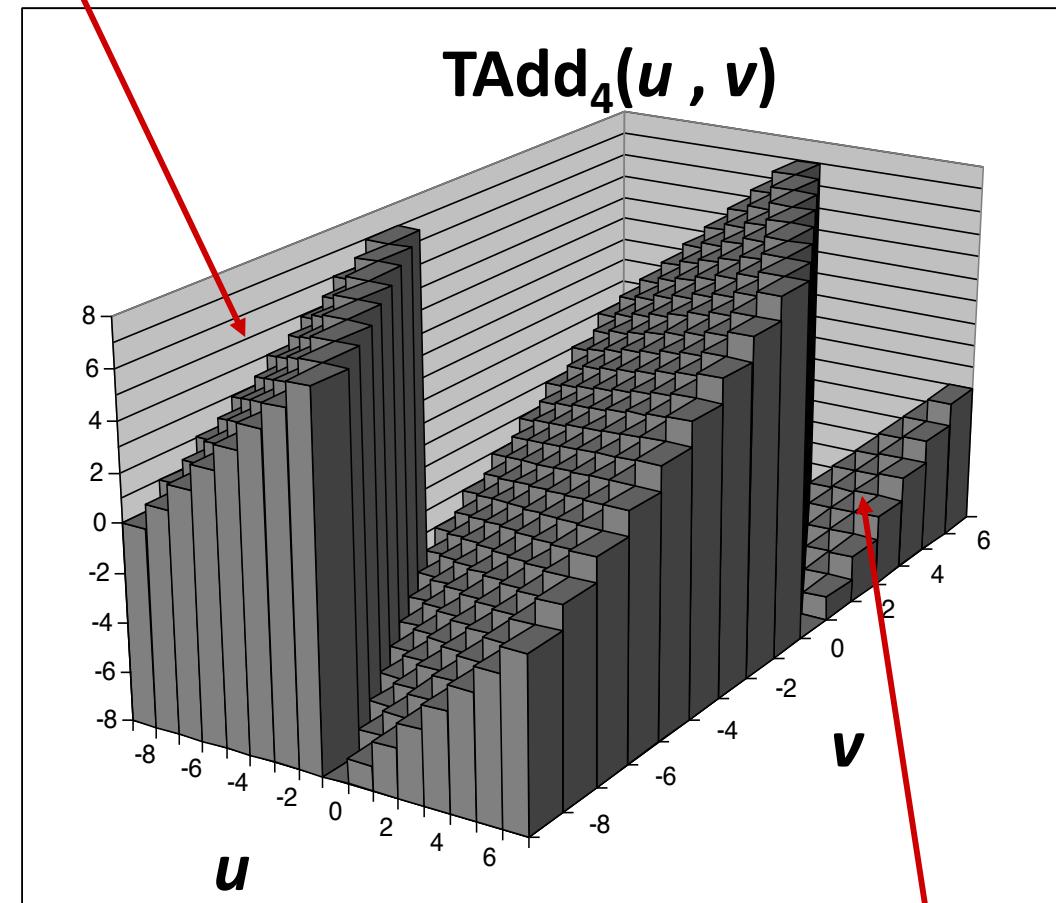
■ Значения

- 4-ре бита в доп. коде
- Диапазон от -8 до +7

■ Отсечение

- Если сумма $\geq 2^{w-1}$
 - Результат отрицательный
 - Не более 1 раза
- Если сумма $< -2^{w-1}$
 - Результат положительный
 - Не более 1 раза

Переполнение
отрицательными



Переполнение
положительными

Умножение

- Вычисление полного произведения w -битовых чисел x, y
 - Или знаковых или беззнаковых
- Границы
 - Беззнаковые: $0 \leq x * y \leq (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$
 - До $2w$ бит
 - Минимальное в доп.коде: $x * y \geq (-2^{w-1}) * (2^{w-1} - 1) = -2^{2w-2} + 2^{w-1}$
 - До $2w-1$ бит
 - Максимальное в доп коде : $x * y \leq (-2^{w-1})^2 = 2^{2w-2}$
 - До $2w$ бит, но только для $(TMin_w)^2$
- Обеспечение полноты результата
 - Необходимо расширение слова для каждого результата
 - Реализуется лишь в программных пакетах арифметики “произвольной точности”

Беззнаковое произведение в Си

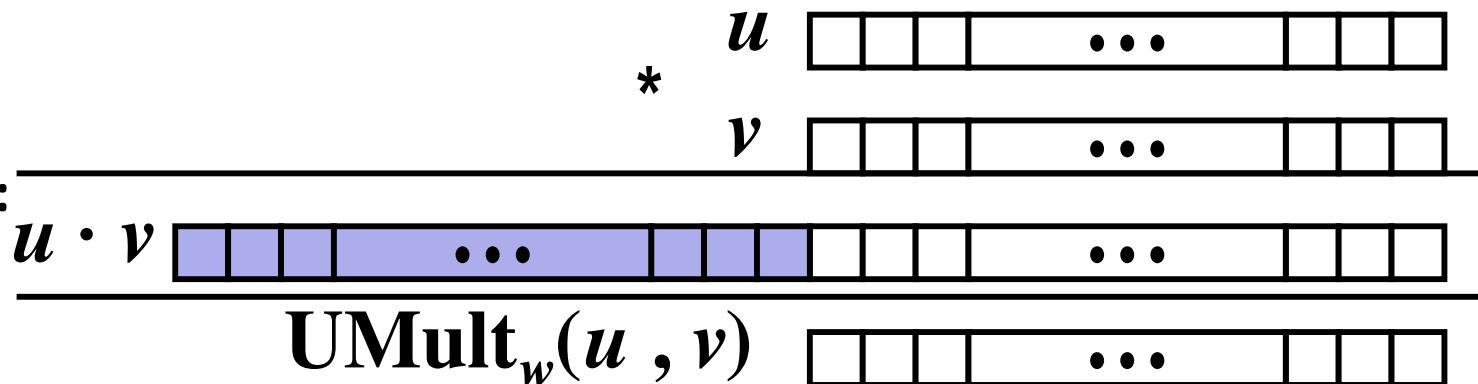
Операнды: w бит

Полное произведение:

$2^w w$ бит

Без w старших бит:

w младших бит



■ Стандартная функция умножения

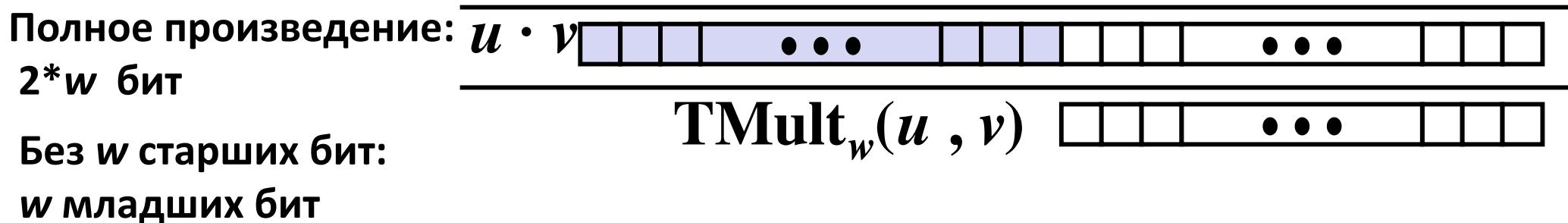
- Игнорирует старшие w бит

■ Реализует умножение по модулю

$$\text{UMult}_w(u, v) = u \cdot v \bmod 2^w$$

Знаковое умножение в Си

Операнды: w бит



■ Стандартная функция умножения

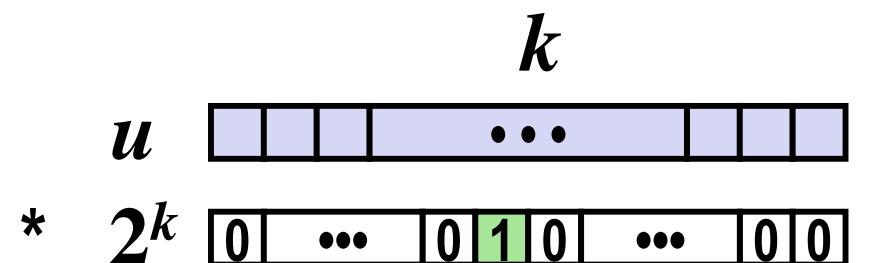
- Игнорирует старшие w бит
- Некоторые из них различаются для знакового и беззнакового умножений
- Младшие биты такие-же

Умножение сдвигом на степень двойки

■ Операция

- $u \ll k$ даёт $u * 2^k$
- Для знаковых и беззнаковых

Операнд: w бит



Полное произведение:
 $w+k$ бит

$$\begin{array}{r} u \cdot 2^k \\ \hline \end{array}$$

Без k старших бит:
 w младших бит

$$\begin{array}{l} \text{UMult}_w(u, 2^k) \\ \text{TMult}_w(u, 2^k) \end{array} \quad \dots \quad 0 \quad \dots \quad 0 \quad 0$$

■ Примеры

- $u \ll 3 == u * 8$
- $u \ll 5 - u \ll 3 == u * 24$
- Большинство машин сдвигает и складывает быстрее умножения
 - Компилятор создаёт соответствующий код автоматически

Биты, байты и целые

- Представление информации в битах
- Манипуляции на уровне бит
- Целые
 - Представление: беззнаковое и знаковое
 - Преобразования
 - Расширение, сокращение
 - Сложение, отрицание, умножение, сдвиг
 - Сводка
- Представление в памяти, указатели, строки

Арифметика: основные правила

■ Сложение:

- (Без)знаковые: Нормальное сложение с отсечением, идентичные действия с битами
- Беззнаковые: сложение по модулю 2^w
 - Математическое сложение и возможное уменьшение на 2^w
- Знаковые: изменённое сложение по модулю 2^w (результат в допустимом диапазоне)
 - Математическое сложение и возможное добавление или уменьшение на 2^w

■ Умножение:

- (Без)знаковые: Нормальное умножение с отсечением, разные действия с битами
- Беззнаковые: умножение по модулю 2^w
- Знаковые: изменённое умножение по модулю 2^w (результат в допустимом диапазоне)

Зачем использовать беззнаковые?

■ *Не используйте без понимания последствий*

- Легко сделать ошибку

```
unsigned i;  
for (i = cnt-2; i >= 0; i--)  
    a[i] += a[i+1];
```

- Может быть очень коварным

```
#define DELTA sizeof(int)  
int i;  
for (i = CNT; i-DELTA >= 0; i-= DELTA)  
    . . .
```

Обратный отсчёт с беззнаковыми

■ Правильно: счётчик цикла - беззнаковый

```
unsigned i;  
for (i = cnt-2; i < cnt; i--)  
    a[i] += a[i+1];
```

■ См. «Безопасное программирование на С и С++», Роберт Сикорд, ISBN 978-5-8459-1908-3, «ВИЛЬЯМС», 2015

- Стандарт гарантирует, что беззнаковое сложение работает как математическое сложение по модулю
 - $0 - 1 \rightarrow UMax$

■ Ещё лучше

```
size_t i;  
for (i = cnt-2; i < cnt; i--)  
    a[i] += a[i+1];
```

- Тип `size_t` определён как беззнаковый с длиной равной размеру слова
- Код работает даже если `cnt = Umax`
- А что если `cnt` – знаковый и < 0 ?

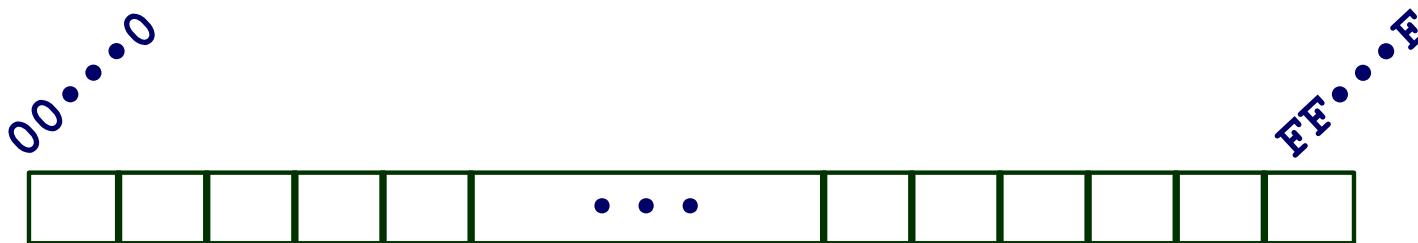
Зачем использовать беззнаковые? (ещё)

- *Используйте для модулярной арифметики*
 - Арифметика произвольной точности
- *Используйте для представления множеств*
 - Логический сдвиг вправо, без расширения знака

Биты, байты и целые

- Представление информации в битах
- Манипуляции на уровне бит
- Целые
 - Представление: беззнаковое и знаковое
 - Преобразования
 - Расширение, сокращение
 - Сложение, отрицание, умножение, сдвиг
 - Сводка
- Представление в памяти, указатели, строки

Байтовая организация памяти



- **Программы обращаются к виртуальным адресам**
 - Концептуально – очень большой массив байт
 - В действительности реализуется иерархией запоминающих устройств различного типа
 - ОС предоставляет своё адресное пространство каждому «процессу»
 - Программа исполняется в рамках своего «процесса»
 - Программа может повредить только свои, но не чужие данные
- **Компилятор, компоновщик, загрузчик и среда исполнения**
 - Определяют где хранятся различные программные объекты
 - Выделяют память внутри единого адресного пространства

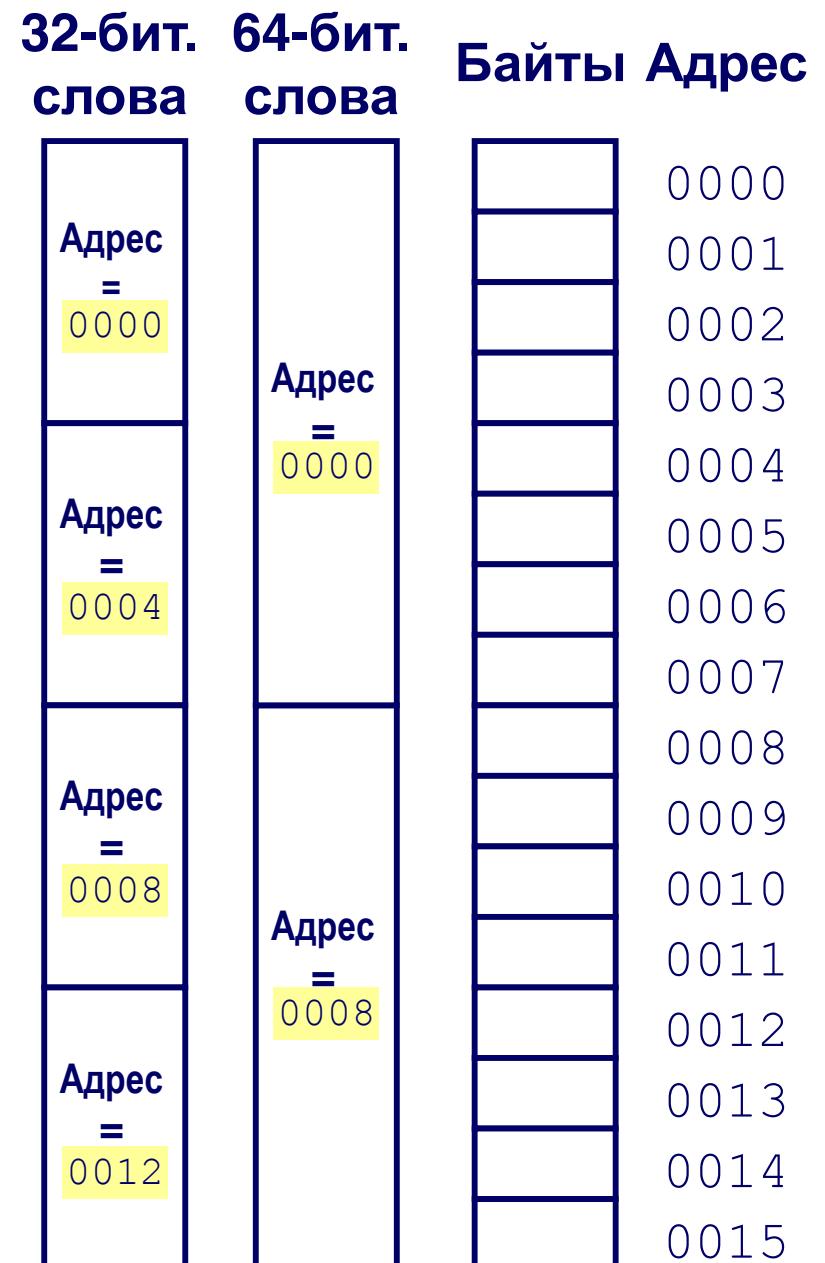
Машинные слова

■ С машиной связан “размер слова”

- Обычный размер представления целых чисел
 - Включая адреса
- Большинство мобильников используют слова в 32 бита (4 байта)
 - Предел адресации 4ГБ
 - Недостаточно для интенсивной работы с памятью
- ПК и более мощные системы - используют слова в 64 бита (8 байт)
 - Потенциальное адресное пространство $\approx 1.8 \times 10^{19}$ байт
 - Архитектура x86-64 использует 48-битовые адреса: 256 терабайт
- Машины поддерживают множество форматов данных
 - Доли размера слова или кратные ему
 - Всегда целое число байт

Словная организация памяти

- Адреса указывают расположение в байтах
 - Адрес первого байта в слове
 - Адреса последовательных слов различаются на 4 (32-битные) или 8 (64-битные)



Целочисленные форматы

Типы данных языка С	32-бит типично	Intel IA32	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	4	8
long long	8	8	8
float	4	4	4
double	8	8	8
long double	8	10/12	10/16
указатель	4	4	8

Порядок байт в слове

- В каком порядке располагаются в памяти байты многобайтового слова?
- Соглашения
 - «Тупоконечники»: Sun, PPC Mac, Internet
 - Наименее значимый байт имеет наибольший адрес
 - «Остроконечники»: x86
 - Наименее значимый байт имеет наименьший адрес

Примеры упорядочения байт

■ «Тупоконечное»

- Наименее значимый байт имеет наибольший адрес

■ «Остроконечное»

- Наименее значимый байт имеет наименьший адрес

■ Пример

- Переменная x

- имеет 4-байтовое представление 0x01234567
- Расположена по адресу &x - 0x100

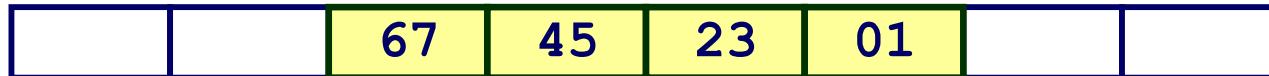
«Тупоконечное»

0x100 0x101 0x102 0x103



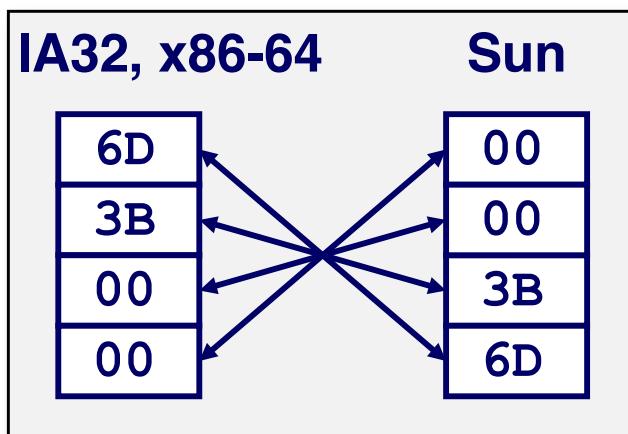
«Остроконечное»

0x100 0x101 0x102 0x103

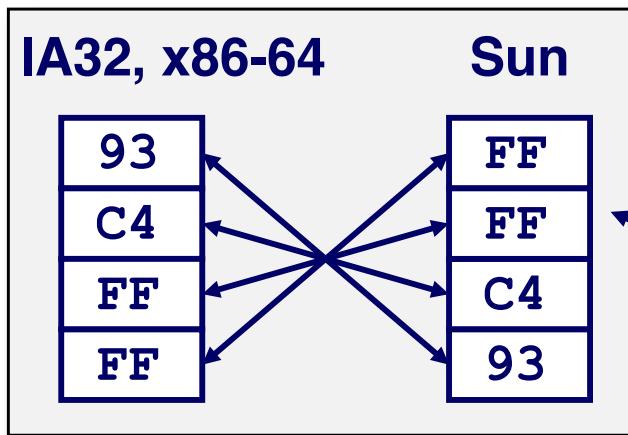


Представление Целочисленное

```
int A = 15213;
```

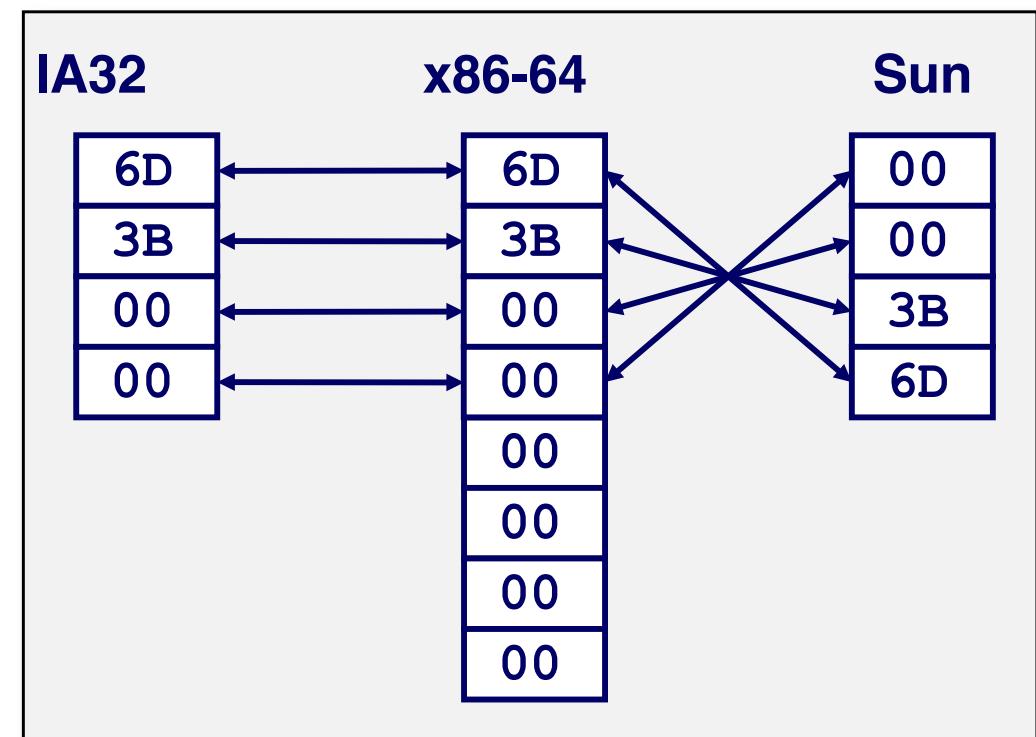


```
int B = -15213;
```



Десятичное:	15213
Двоичное:	0011 1011 0110 1101
Шестнадцатиричное:	3 B 6 D

```
long int C = 15213;
```



Два представления в дополнительном коде
(пояснения последуют)

Изучение представления данных

■ Вывод байтового представления данных

- Представление указателя как массива `unsigned char *`

```
typedef unsigned char *pointer;

void show_bytes(pointer start, int len) {
    int i;
    for (i = 0; i < len; i++)
        printf("%p\t0x%.2x\n", start+i, start[i]);
    printf("\n");
}
```

Спецификации преобразования:
%p: Вывод указателя
%x: Вывод шестнадцатиричного

Пример исполнения show_bytes

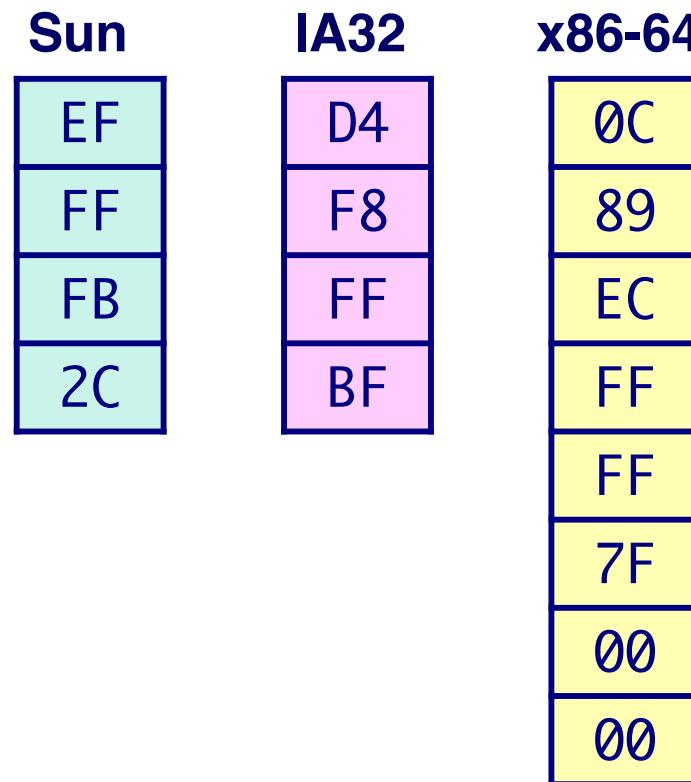
```
int a = 15213;  
printf("int a = 15213;\\n");  
show_bytes((pointer) &a, sizeof(int));
```

Результат (x86, Linux):

```
int a = 15213;  
0x11ffffcb8 0x6d  
0x11ffffcb9 0x3b  
0x11ffffcba 0x00  
0x11ffffcbb 0x00
```

Представление указателей

```
int B = -15213;  
int *P = &B;
```



Различные компиляторы, ОС и машины дают различное расположение в памяти

Представление строк

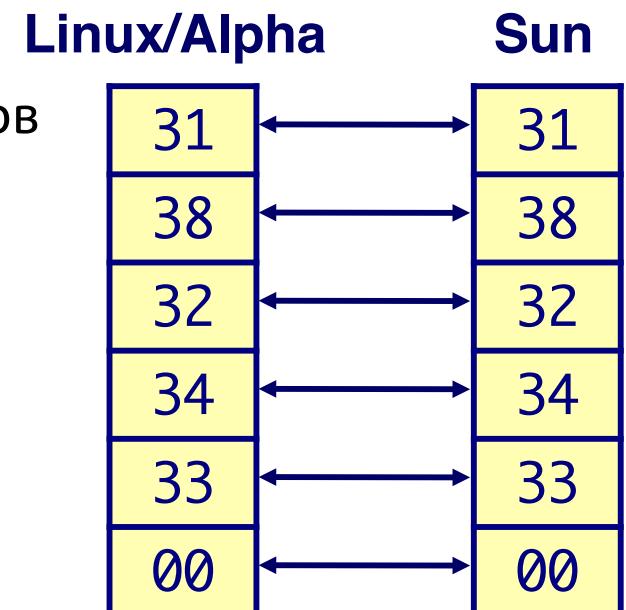
```
char S[6] = "18243";
```

■ Строки в С

- Представлены массивами символов
- Каждый символ представлен ASCII-кодом
 - Стандартное 7-кодирование набора символов
 - Символ “0” кодируется 0x30
 - Цифра i кодируется $0x30+i$
- Строки должны завершаться нулевым кодом
 - Символ окончания строки = 0

■ Совместимость

- ASCII-код не единственный
- Однобайтные коды не переупорядочиваются



Целочисленные головоломки Си

■ Для каждого следующего выражения Си :

- Либо объясните почему оно верно для любых значений аргументов
- Либо - почему неверно

- $x < 0 \Rightarrow ((x^2) < 0)$
- $ux \geq 0$
- $x \& 7 == 7 \Rightarrow (x << 30) < 0$
- $ux > -1$
- $x > y \Rightarrow -x < -y$
- $x * x \geq 0$
- $x > 0 \&\& y > 0 \Rightarrow x + y > 0$
- $x \geq 0 \Rightarrow -x \leq 0$
- $x \leq 0 \Rightarrow -x \geq 0$
- $(x|-x)>>31 == -1$
- $ux >> 3 == ux/8$
- $x >> 3 == x/8$
- $x \& (x-1) != 0$

Инициализация

```
int x = function1();
int y = funciton2();
unsigned ux = x;
unsigned uy = y;
```

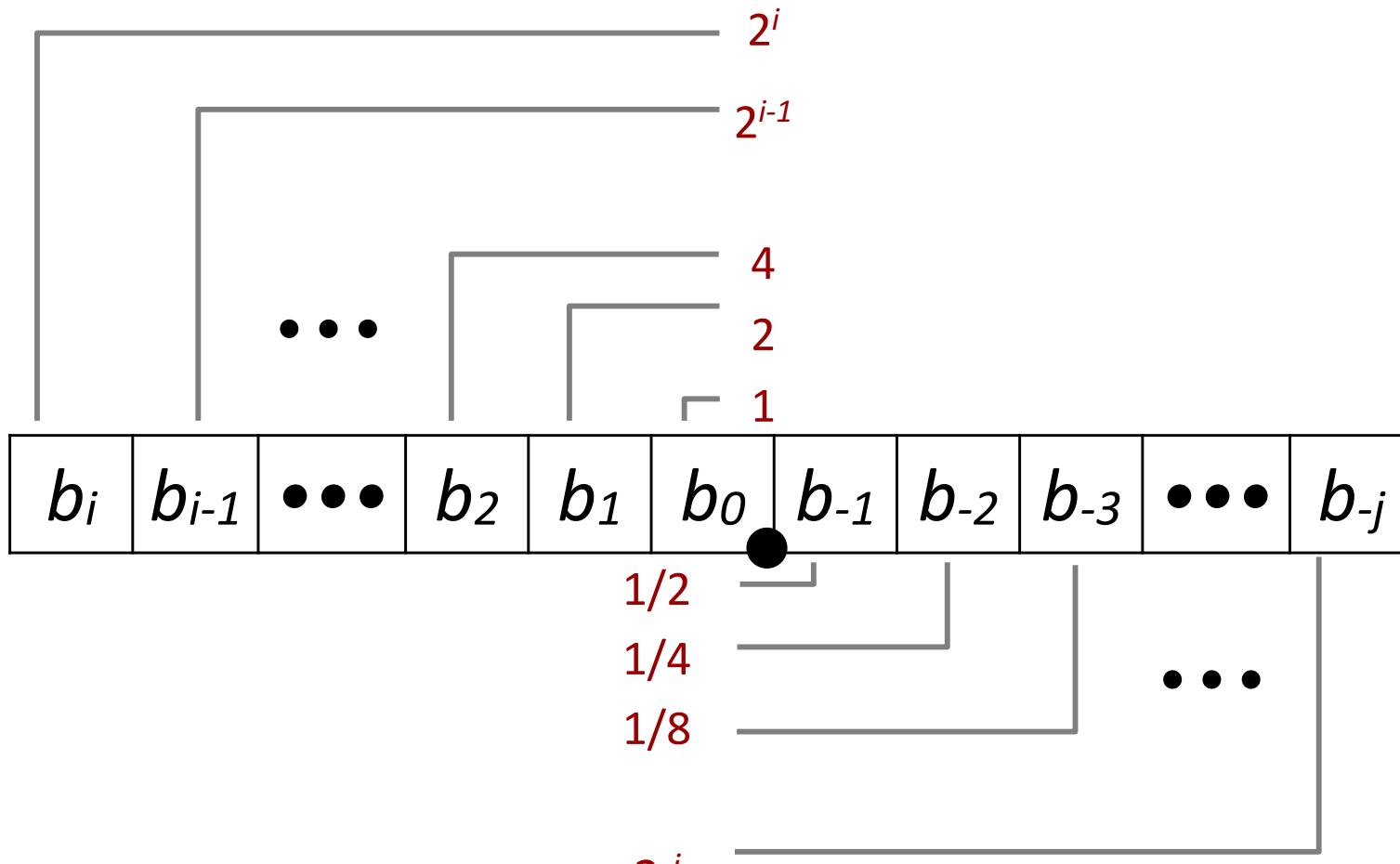
Плавающая точка

- **Основы: Двоичные дроби**
- Стандарт «плавающей точки» IEEE : Определение
- Примеры и свойства
- Округление, сложение, умножение
- Плавающая точка в Си
- Сводка

Двоичные дроби

- Что такое 1011.101_2 ?

Двоичные дроби



■ Представление

$$2^{-j}$$

- Биты справа от «двоичной точки» представляют дробные степени 2
- Представление рациональных чисел:

$$\sum_{k=-j}^i b_k \times 2^k$$

Двоичные дроби: Примеры

■ Значение	Представление
5 и $\frac{3}{4}$	101.11_2
2 и $\frac{7}{8}$	10.111_2
1 и $\frac{7}{16}$	1.0111_2
$\frac{23}{32}$	0.10111_2

■ Важно

- Деление на 2 сдвигом вправо
 - Умножение на 2 сдвигом влево
 - Числа вида $0.111111\dots_2$ меньше 1.0
 - $1/2 + 1/4 + 1/8 + \dots + 1/2^i + \dots \rightarrow 1.0$
 - Обозначим $1.0 - \varepsilon$

(не)Представимые числа (1)

■ Ограничения

- Двоичными дробями точно представимы только числа вида $n/2^k$
- Другие дроби представимы бесконечными периодическими «дробями»

■ Значение Представление

- $1/3$ $0.[01]..._2$
- $1/5$ $0.[0011]..._2$
- $1/10$ $0.0[0011]..._2$

Плавающая точка

- Основы: Двоичные дроби
- Стандарт «плавающей точки» IEEE : Определение
- Примеры и свойства
- Округление, сложение, умножение
- Плавающая точка в Си
- Сводка

Плавающая точка IEEE

■ Стандарт IEEE 754

- Принят в 1985 как единый стандарт арифметики с плавающей точкой
 - До этого множество уникальных стандартов
- Поддерживается всеми основными CPU/FPU

■ В основе - вопросы вычислений

- Удачно стандартизованы
 - округления,
 - переполнения,
 - потеря значимости
 - Сложно сделать быстрым в аппаратуре
 - При создании стандарта численные аналитики доминировали над разработчиками аппаратуры
 - Есть реализации «с отклонениями»
- ## ■ Есть версии и альтернативы

Представление с плавающей точкой

■ Численная форма:

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

- Знаковый бит **s** определяет положительность/отрицательность
- Мантисса **M** - обычно дробь в интервале [1.0,2.0).
- Порядок **E** изменяет значение на степень двойки

■ Кодирование

- Наиболее значимый бит **S** – знаковый
- Поле **exp** кодирует **E**, но не совпадает с двоичным значением **E**
- Поле **frac** кодирует **M**, но не совпадает с двоичным значением **M**



Применяемые точности

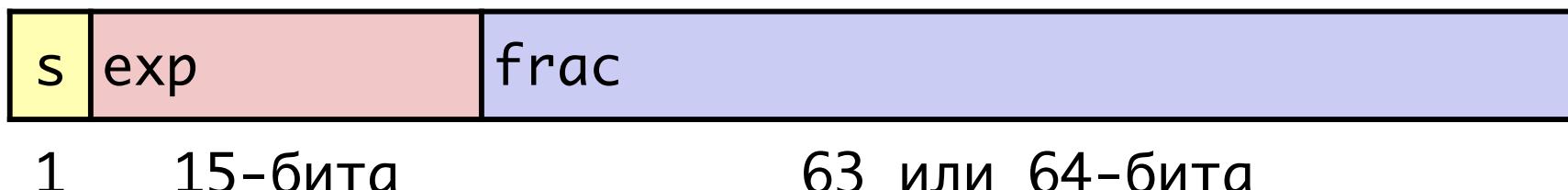
■ Одинарная: 32 бита



■ Двойная: 64 бита



■ Расширенная: 80 бит (только для Intel)



Нормализованные значения

- Признак: $\text{exp} \neq 000\dots0$ и $\text{exp} \neq 111\dots1$
- Порядок кодируется со смещением : $E = Exp - Bias$
 - Exp : беззнаковое значение поля **exp**
 - $Bias = 2^{k-1} - 1$, где k - количество бит порядка – смещение
 - Одинарная точность: 127 ($Exp: 1\dots254$, $E: -126\dots127$)
 - Двойная точность: 1023 ($Exp: 1\dots2046$, $E: -1022\dots1023$)
- Код мантиссы подразумевает старшую 1: $M = 1.\text{xxx...x}_2$
 - xxx...x : биты поля **frac**
 - Минимальное значение $M = 1.0$, когда **frac** = 000...0
 - Минимальное значение $M = 2.0 - \varepsilon$, когда **frac** = 111...1
 - Старший бит мантиссы не расходует ресурсы оборудования
 - Не подразумевается для расширенной точности 80-бит Intel !

Пример нормализованного кода

- Значение: `Float F = 15213.0;`

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

- Мантисса

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

- $\text{frac} = \underline{11011011011010000000000}_2$

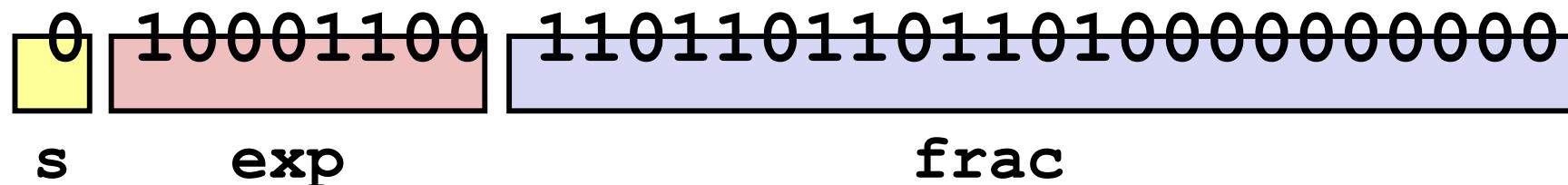
- Порядок

- $E = 13$

- $Bias = 127$

- $Exp = 140 = 10001100_2$

- Результат:



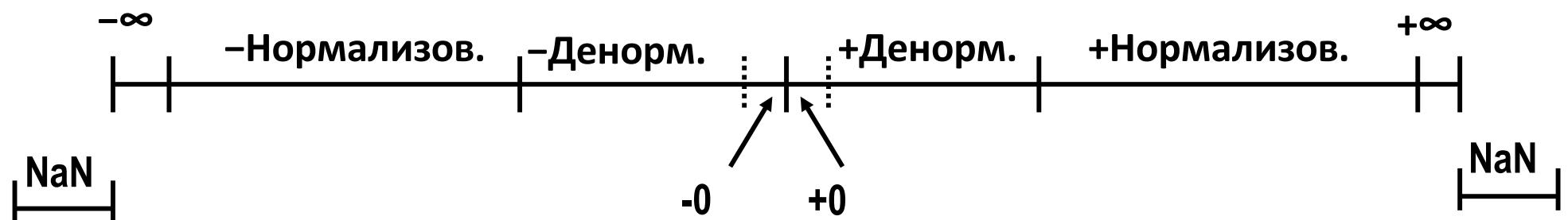
Денормализованные значения

- Признак: $\text{exp} = 000\dots0$
- Значение порядка: $E = 1 - \text{Bias}$ (вместо $E = 0 - \text{Bias}$)
- Код мантиссы подразумевает старший 0: $M = 0.\text{xxx}\dots\text{x}_2$
 - $\text{xxx}\dots\text{x}$: биты `frac`
- Варианты
 - $\text{exp} = 000\dots0, \text{frac} = 000\dots0$
 - Обозначает нулевое значение
 - Представление неоднозначно: +0 and –0 (почему?)
 - $\text{exp} = 000\dots0, \text{frac} \neq 000\dots0$
 - числа очень близкие к 0.0
 - чем меньше, тем хуже относительная погрешность
 - равноотстоящие

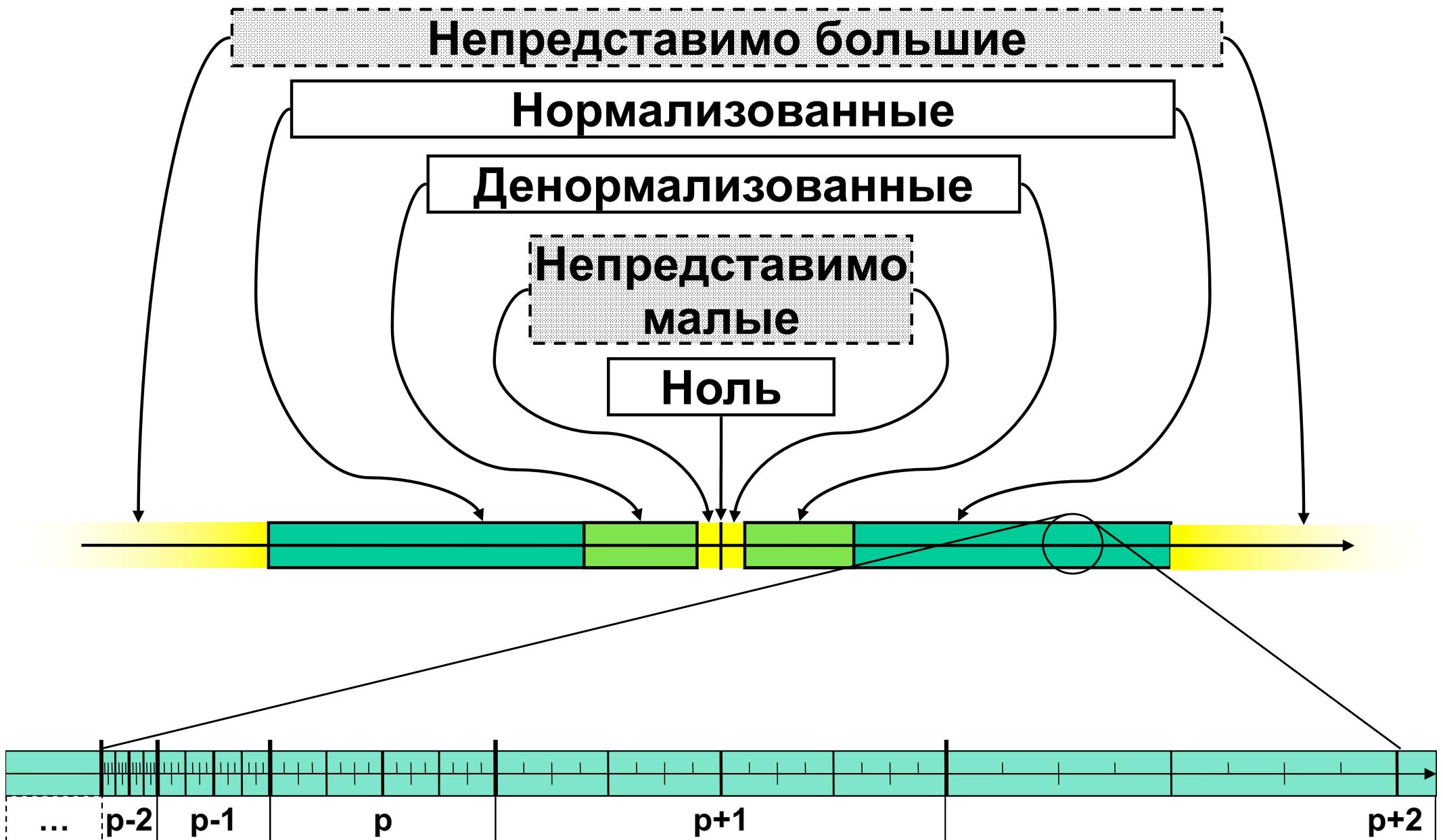
Специальные коды

- Признак: **exp = 111...1**
- Вариант 1: **exp = 111...1, frac = 000..0**
 - Обозначает значение ∞ (бесконечность)
 - Результат операции при переполнении
 - Есть оба варианта: отрицательная и положительная
 - E.g., $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$
- Вариант 2: **exp = 111...1, frac \neq 000..0**
 - Не число, Not-a-Number (NaN)
 - Обозначает случаи когда невозможно определить численное значение
 - Примеры: $\sqrt{-1}$, $\infty - \infty$, $\infty \times 0$

Коды с плавающей точкой визуально



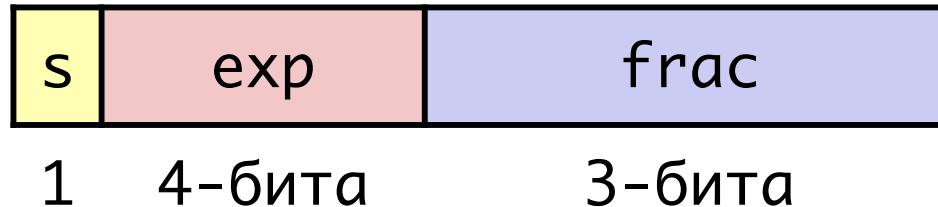
(не)Представимые числа (2)



Плавающая точка

- Основы: Двоичные дроби
- Стандарт «плавающей точки» IEEE : Определение
- Примеры и свойства
- Округление, сложение, умножение
- Плавающая точка в Си
- Сводка

Укороченный пример с плавающей точкой



■ 8-битное представление с плавающей точкой

- Знаковый бит – самый значимый
- следующие четыре бита - показатель со смещением $2^{4-1}-1 = 7$
- последние три бита - **код мантиссы**

■ Форма также что в стандарте IEEE

- нормализованные и денормализованные
- обозначаются 0, NaN, бесконечность

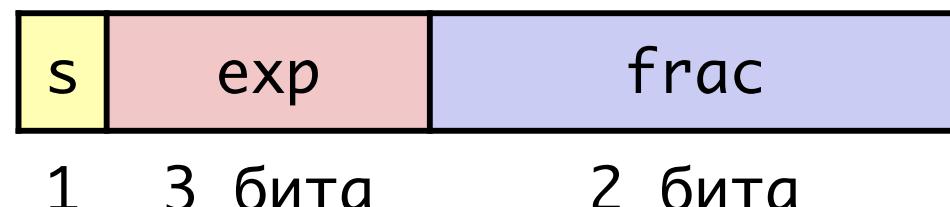
Динамический диапазон положительных

	s	exp	frac	E	Значение	
	0	0000	000	-6	0	
	0	0000	001	-6	$1/8 * 1/64 = 1/512$	Ближайшее к нулю
Деномализ.	0	0000	010	-6	$2/8 * 1/64 = 2/512$	
числа	...					
	0	0000	110	-6	$6/8 * 1/64 = 6/512$	
	0	0000	111	-6	$7/8 * 1/64 = 7/512$	Наибольшее денорм.
	0	0001	000	-6	$8/8 * 1/64 = 8/512$	Наименьшее нормализ.
	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
Нормализов.	0	0111	000	0	$8/8 * 1 = 1$	
числа	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 нормализ.
	0	1111	000	n/a	inf	

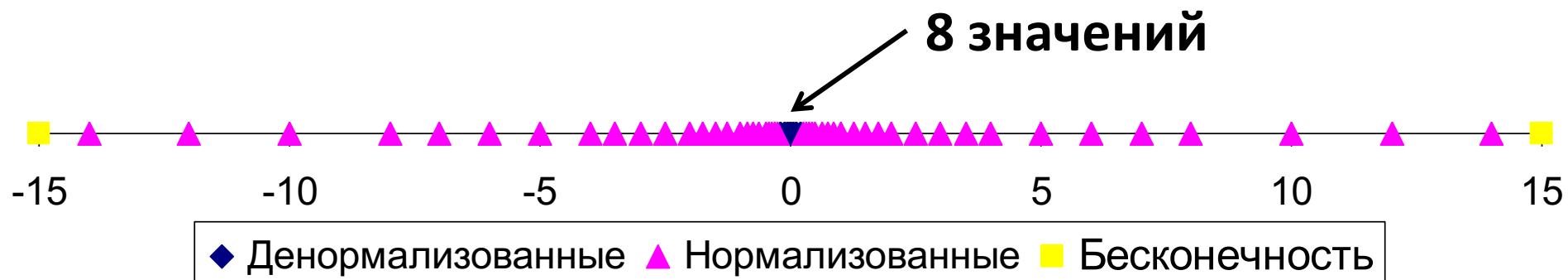
Распределение значений

■ Шестибитный формат подобный IEEE

- 3 бита порядка
- 2 дробных бита
- смещение: $2^{3-1}-1 = 3$



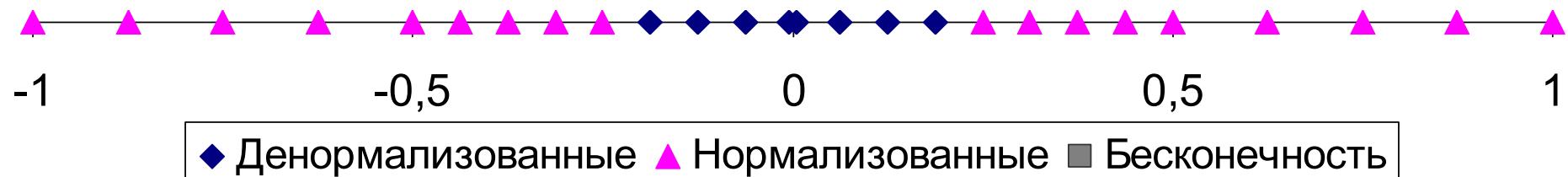
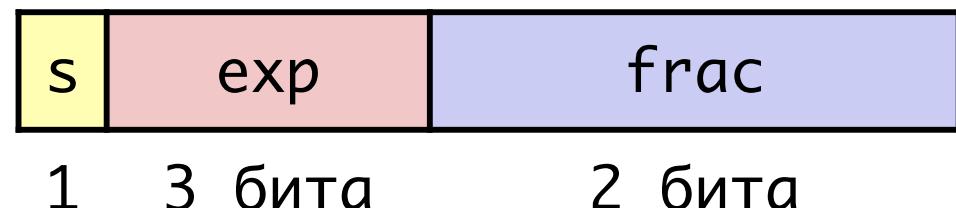
■ Сгущение вокруг 0



Распределение значений (крупным планом)

■ Шестибитный формат подобный IEEE

- 3 бита порядка
- 2 дробных бита
- смещение: $2^{3-1}-1 = 3$



Особые свойства кодирования IEEE

- Совпадают коды целочисленного нуля и «положительного» нуля с плавающей точкой
 - Все биты - нулевые
- Можно ли сравнить как беззнаковые целые?
 - Сравнить знаковые биты
 - Учесть $-0 = 0$
 - Решить проблему с NaN
 - NaN код больше любого численного кода
 - Что должно выдать сравнение?
 - В остальном - ОК
 - Денормализованные с нормализованными
 - Нормализованные с бесконечностью

Плавающая точка

- Основы: Двоичные дроби
- Стандарт «плавающей точки» IEEE : Определение
- Примеры и свойства
- **Округление, сложение, умножение**
- Плавающая точка в Си
- Сводка

Операции с плав. точкой: Основная идея

- $x +_f y = \text{Round}(x + y)$
- $x \times_f y = \text{Round}(x \times y)$
- **Основная идея**
 - Сначала вычислить полный результат
 - Затем втиснуть его в желаемую точность
 - Возможно с переполнением, если порядок слишком велик
 - Возможно с округлением, чтобы уместиться во `frac`

Округление

■ Режим округления	1.40	1.60	1.50	2.50	-
■ 1.50					
■ К нулю	1	1	1	2	-1
■ Вниз ($-\infty$)	1	1	1	2	-2
■ Вверх ($+\infty$)	2	2	2	3	-1
■ К чётному (по умолч.)	1	2	2	2	-2
■ В чём преимущества режимов?					

Подробнее об округлении к чётному

■ Режим округления по умолчанию

- Любой другой режим требует обращения к ассемблеру
- Все остальные режимы дают статистический сдвиг
 - Сумма округлённых устойчиво меньше или устойчиво больше округлённой суммы

■ Применяя к десятичным разрядам

- Если точно посередине между двумя возможными значениями
 - Округляется так, что младшая цифра чётная
- Например, округление к ближайшей сотой

1.2349999	1.23	(Меньше половины - вниз)
-----------	------	--------------------------

1.2350001	1.24	(Больше половины - вверх)
-----------	------	---------------------------

1.2350000	1.24	(Половина - вверх)
-----------	------	--------------------

1.2450000	1.24	(Половина - вниз)
-----------	------	-------------------

Округление двоичных чисел

■ Двоичные дроби

- “Чётные” когда наименьший бит = 0
- “Половина” когда биты справа от позиции округления = $100\dots_2$

■ Примеры

- Округление к $1/4$ (2 бита справа от двоичной точки)

Значение	Двоичное Округлённое	Двоичное Округление	Действие	Значение
$2\frac{3}{32}$	$10.00\textcolor{red}{011}_2$	10.00_2	($<1/2$ —вниз)	2
$2\frac{3}{16}$	$10.00\textcolor{red}{110}_2$	10.01_2	($>1/2$ —вверх)	$2\frac{1}{4}$
$2\frac{7}{8}$	$10.11\textcolor{red}{100}_2$	11.00_2	($1/2$ —вверх)	3
$2\frac{5}{8}$	$10.10\textcolor{red}{100}_2$	10.10_2	($1/2$ —вниз)	$2\frac{1}{2}$

Умножение с плавающей точкой

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

- Полный результат: $(-1)^s M 2^E$

- Знак s : $s_1 \wedge s_2$
- Мантисса M : $M_1 \times M_2$
- Порядок E : $E_1 + E_2$

- Нормализация

- Если $M \geq 2$, сдвинуть M вправо, увеличив E
- Если E за рамками, то переполнение
- Округлить M чтобы уложиться в точность `frac`

- Реализация

- Трудоёмкая часть - перемножение мантисс

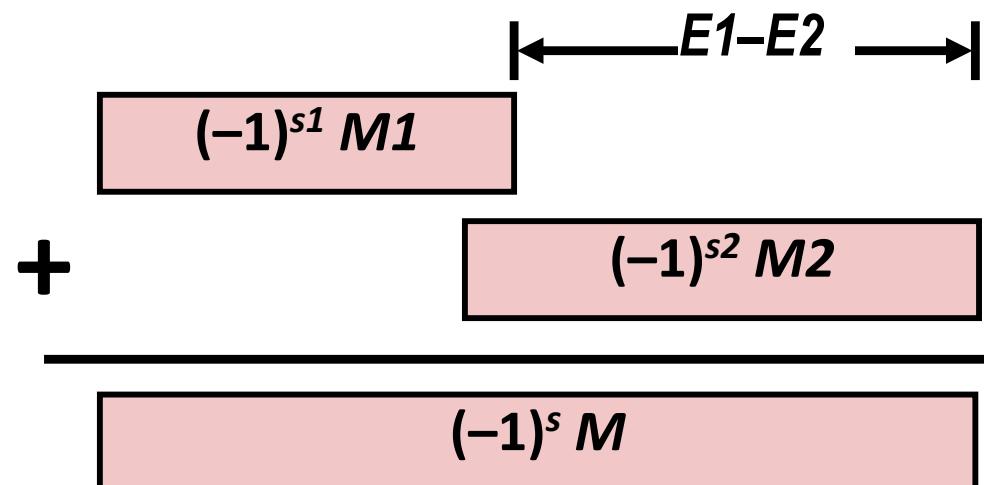
Сложение с плавающей точкой

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

- Предположим $E1 > E2$

- Полный результат: $(-1)^s M 2^E$

- Знак s , мантисса M :
 - Результат знакового выравнивания и сложения
 - Порядок E : $E1$



■ Нормализация

- Если $M \geq 2$, сдвинуть M вправо, увеличив E
 - Если $M < 1$, сдвинуть M влево на k разрядов, уменьшив E на k
 - Если E за рамками, то переполнение
 - Округлить M чтобы уложиться в точность `frac`

Математические свойства сложения с ПТ

■ Сравним со свойствами абелевой группы

- Замкнута относительно сложения ? **Да, но...**
 - Может выдавать бесконечность или нечисло
- Коммутативно ? **Да**
- Ассоциативно ? **Нет**
 - Переполнение и округление!
- Ноль – нейтральный элемент ? **Да**
- Каждый элемент имеет обратный ? **Почти**
 - Кроме бесконечности и нечисла

■ Монотонность

- $a \geq b \Rightarrow a+c \geq b+c$? **Почти**
 - Кроме бесконечности и нечисла

Математические свойства умножения с ПТ

■ Сравним со свойствами коммутативного кольца

- Замкнут относительно умножения ?
 - Может выдавать бесконечность или нечислоДа
- Умножение коммутативно ?Да
- Умножение ассоциативно ?Нет
 - Переполнение и округление!
- Единица нейтральный элемент умножения ?Да
- Умножение дистрибутивно относительно сложения ?Нет
 - Переполнение и округление!

■ Монотонность

- $a \geq b \ \& \ c \geq 0 \Rightarrow a * c \geq b * c$?Почти
 - Кроме бесконечностей и нечисел

Плавающая точка

- Основы: Двоичные дроби
- Стандарт «плавающей точки» IEEE : Определение
- Примеры и свойства
- Округление, сложение, умножение
- Плавающая точка в Си
- Сводка

Плавающая точка в Си

- Си гарантирует как минимум два уровня точности
 - **float** одиночная
 - **double** двойная
- Преобразования
 - Преобразование между **int**, **float**, и **double** меняет набор битов
 - **double/float → int**
 - Отсекает дробную часть
 - Подобно округлению к нулю
 - Не определено для запредельных или NaN: обычно TMin
 - **int → double**
 - Точное преобразование, т.к. **int** имеет разрядность ≤ 53 бит
 - **int → float**
 - Округляется в соответствии в режимом
- x86: вычисления выражения в 80-битном представлении

Головоломки плавающей точки

■ Для каждого следующего выражения Си объясните:

- либо почему оно верно для любых значений аргументов
- либо почему неверно

```
int x = ...;
float f = ...;
double d = ...;
```

Принять, что
ни d ни f не есть NaN

- $x == (int)(float) x$
- $x == (int)(double) x$
- $f == (float)(double) f$
- $d == (float) d$
- $f == -(-f);$
- $2/3 == 2/3.0$
- $d < 0.0 \Rightarrow ((d*2) < 0.0)$
- $d > f \Rightarrow -f > -d$
- $d * d >= 0.0$
- $(d+f)-d == f$

Плавающая точка

- Основы: Двоичные дроби
- Стандарт «плавающей точки» IEEE : Определение
- Примеры и свойства
- Округление, сложение, умножение
- Плавающая точка в Си
- Сводка

Сводка

- Плавающая точка IEEE имеет ясные математические свойства. Почти!
- Представимы числа вида $(-1)^s M \times 2^E$
- Операции независимы от реализации
 - Если все выполнены с двойной точностью и затем округлены
- Не совпадают с вещественной арифметикой
 - Нарушены свойства ассоциативности/дистрибутивности
 - Создаёт сложности для разработчиков компиляторов и ответственных численных приложений

Дополнения

Создание числа с плав. точкой

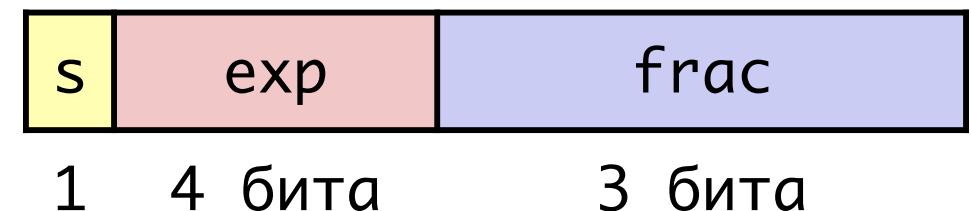
■ Шаги

- Нормализовать до получения старшей 1
- Округлить до укладывания в мантиссу
- Перенормализовать для учёта округления

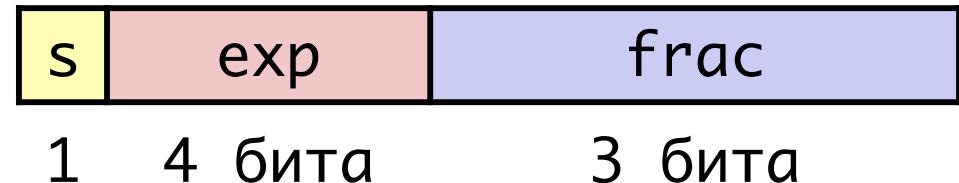
■ Пример

- Преобразовать 8 битовые беззнаковые числа в формат с плавающей точкой

128	10000000
13	00001101
15	00001111
33	00010001
35	00010011
138	10001010
63	00111111



Нормализация



<i>Значение</i>	<i>Двоичное</i>	<i>Дробь</i>	<i>Порядок</i>
128	10000000	1.0000000	7
15	00001111	1.1110000	3
17	00010001	1.0001000	4
19	00010011	1.0011000	4
138	10001010	1.0001010	7
63	00111111	1.1111100	5

Округление

1. BBGRXXXX

Guard bit

Sticky bit: ИЛИ остальных бит

Round bit: 1-й удалённый

■ Правила округления

- Round = 1, Sticky = 1 $\rightarrow > 0.5$
- Guard = 1, Round = 1, Sticky = 0 \rightarrow округление к четному

Значение	Дробь	GRS	Добав.	Округлённое
128	1.0000000	000	N	1.000
15	1.1110000	100	N	1.111
17	1.0001000	010	N	1.000
19	1.0011000	110	Y	1.010
138	1.0001010	011	Y	1.001
63	1.1111100	111	Y	10.000

Перенормализация

■ Проблема

- Округление может вызвать переполнение
- Решается сдвигом одиночным сдвигом вправо и увеличением порядка

<i>Значение</i>	<i>Округленное</i>	<i>Пор.</i>	<i>Уточнённое</i>	<i>Результат</i>
128	1.000	7		128
15	1.111	3		15
17	1.000	4		16
19	1.010	4		20
138	1.001	7		134
63	10.000	5	1.000	64 (пор.
6)				

Интересные числа

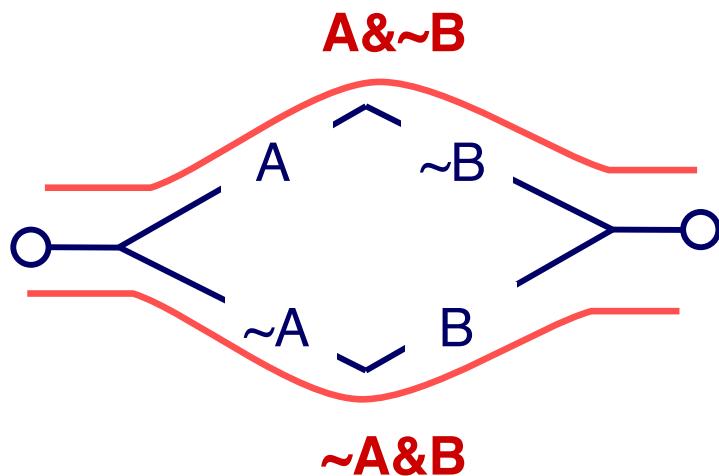
{одинарные , двойные}

<i>Описание</i>	<i>exp</i>	<i>frac</i>	<i>Численное значение</i>
■ Ноль	00...00	00...00	0.0
■ Наименьшее денормализ.	00...00	00...01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
■ ■ Одинарное $\approx 1.4 \times 10^{-45}$			
■ ■ Двойное $\approx 4.9 \times 10^{-324}$			
■ Наибольшее денормализ.	00...00	11...11	$(1.0 - \varepsilon) \times 2^{-\{126,1022\}}$
■ ■ Одинарное $\approx 1.18 \times 10^{-38}$			
■ ■ Двойное $\approx 2.2 \times 10^{-308}$			
■ Наименьшее нормализ.	00...01	00...00	$1.0 \times 2^{-\{126,1022\}}$
■ ■ Ближайшее большее наибольшего денормализованного			
■ Единица	01...11	00...00	1.0
■ Наибольшее нормализ.	11...10	11...11	$(2.0 - \varepsilon) \times 2^{\{127,1023\}}$
■ ■ Одинарное $\approx 3.4 \times 10^{38}$			
■ ■ Двойное $\approx 1.8 \times 10^{308}$			

Приложение булевой алгебры

■ Клодом Шенноном применена к цифровым системам

- Диплом МИТ 1937 год
- Рассмотрены схемы реле
 - Замкнутый контакт кодируется как 1, разомкнутый как 0



Контакт есть если

$$A \& \sim B \mid \sim A \& B$$

$$= A \wedge B$$

Пример небезопасного кода

```
/* Участок памяти ядра содержит данные доступные пользователю */
#define KSIZE 1024
char kbuf[FSIZE];

/* Копируем не более maxlen байт из области ядра в буфер пользователя */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Счётчик байт len - минимальное из размера буфера и maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

- Аналог кода найденного в реализации `getpeername` FreeBSD's
- Легионы способных людей пытаются находить уязвимости в программах

Типичное применение

```
/* Участок памяти ядра содержит данные доступные пользователю */
#define KSIZE 1024
char kbuf[ysize];

/* Копируем не более maxlen байт из области ядра в буфер пользователя */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Счётчик байт len - минимальное из размера буфера и maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, MSIZE);
    printf("%s\n", mybuf);
}
```

Вредоносное применение

```
/* Определение библиотечной функции memcpy */
void *memcpy(void *dest, void *src, size_t n);
```

```
/* Участок памяти ядра содержит данные доступные пользователю */
#define KSIZE 1024
char kbuf[ysize];

/* Копируем не более maxlen байт из области ядра в буфер пользователя */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Счётчик байт len - минимальное из размера буфера и maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, -MSIZE);
    . . .
}
```

Математические свойства UAdd

■ Сложение по модулю образует абелеву группу

- **Замкнута** относительно сложения

$$0 \leq \text{UAdd}_w(u, v) \leq 2^w - 1$$

- **Коммутативна**

$$\text{UAdd}_w(u, v) = \text{UAdd}_w(v, u)$$

- **Ассоциативна**

$$\text{UAdd}_w(t, \text{UAdd}_w(u, v)) = \text{UAdd}_w(\text{UAdd}_w(t, u), v)$$

- **0** является нейтральным элементом

$$\text{UAdd}_w(u, 0) = u$$

- Любому элементу есть **обратный**

- Let $\text{UComp}_w(u) = 2^w - u$

$$\text{UAdd}_w(u, \text{UComp}_w(u)) = 0$$

Математические свойства TAdd

■ Группа изоморфная беззнаковым с UAdd

- $TAdd_w(u, v) = U2T(UAdd_w(T2U(u), T2U(v)))$
 - т.к. битовые преобразования идентичны

■ Дополнительные коды с TAdd образуют группу

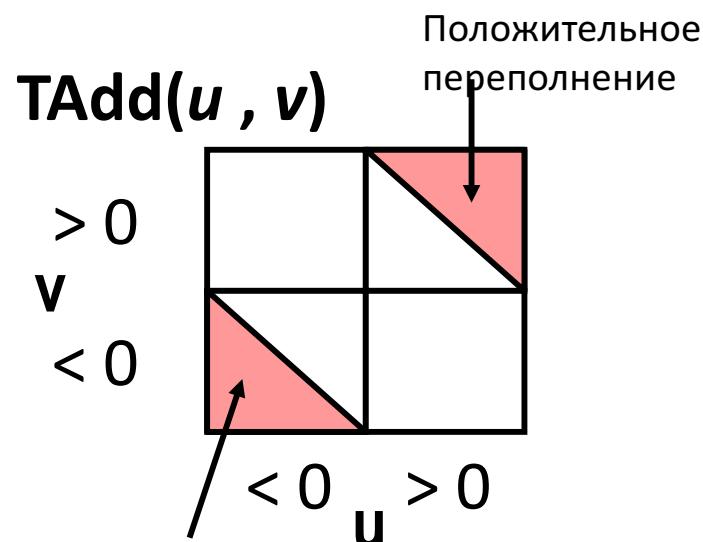
- Замкнута, коммутативна, ассоциативна, 0 – нейтральный элемент
- Для каждого элемента есть обратный

$$TComp_w(u) = \begin{cases} -u & u \neq TMin_w \\ TMin_w & u = TMin_w \end{cases}$$

Определение TAdd

■ Функциональность

- Полное суммирование требует $w+1$ бит
- Отбрасывание MSB
- Интерпретация оставшихся бит как целого числа в дополнительном коде



$$TAdd_w(u, v) = \begin{cases} u + v + 2^w & u + v < TMin_w \\ u + v & TMin_w \leq u + v \leq TMax_w \\ u + v - 2^w & TMax_w < u + v \end{cases}$$

Переполнение
отрицательными

Переполнение
положительными

Изменение знака: Инверсия и увеличение

- Утверждение: для дополнительного кода верно

$$\sim x + 1 == -x$$

- Инверсия

- Важно: $\sim x + x == 1111\dots111 == -1$

$$\begin{array}{r} x \quad \boxed{10011101} \\ + \sim x \quad \boxed{01100010} \\ \hline -1 \quad \boxed{11111111} \end{array}$$

Примеры инверсии и увеличения

$x = 15213$

	Десятичное	Шестнадцатиричное	Двоичное
x	15213	3B 6D	00111011 01101101
$\sim x$	-15214	C4 92	11000100 10010010
$\sim x + 1$	-15213	C4 93	11000100 10010011
y	-15213	C4 93	11000100 10010011

$x = 0$

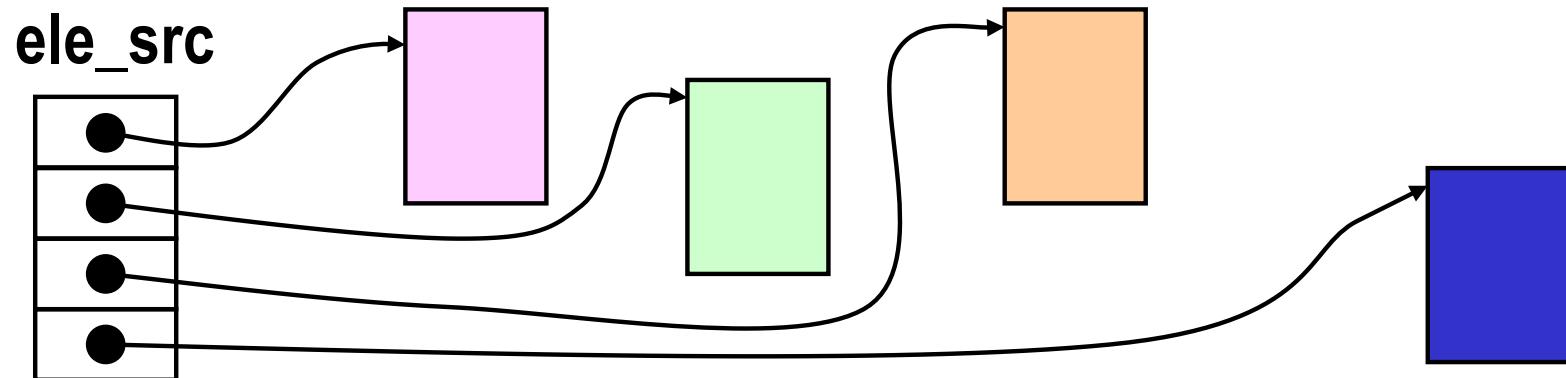
	Десятичное	Шестнадцатиричное	Двоичное
0	0	00 00	00000000 00000000
~ 0	-1	FF FF	11111111 11111111
$\sim 0 + 1$	0	00 00	00000000 00000000

Пример небезопасного кода №2

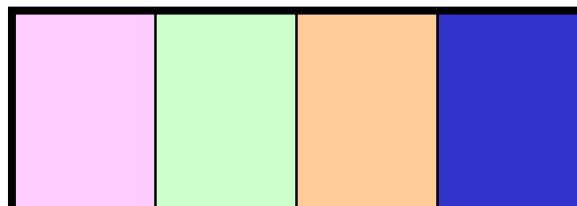
■ Библиотека SUN XDR

- Широко применяется для межмашинного обмена данными

```
void* copy_elements(void *ele_src[], int ele_cnt, size_t ele_size);
```



```
malloc(ele_cnt * ele_size)
```



Код XDR

```
void* copy_elements(void *ele_src[], int ele_cnt, size_t ele_size) {
/*
 * Занимаем буфер для ele_cnt объектов, каждый ele_size байт
 * и копируем с места указанного ele_src
 */
void *result = malloc(ele_cnt * ele_size);
if (result == NULL)
    /* Ошибка malloc */
    return NULL;
void *next = result;
int i;
for (i = 0; i < ele_cnt; i++) {
    /* Копирование объектов по назначению */
    memcpuy(next, ele_src[i], ele_size);
    /* Перемещение указателя на следующую позицию в памяти */
    next += ele_size;
}
return result;
}
```

Уязвимость XDR

`malloc(ele_cnt * ele_size)`

- Что если:
 - `ele_cnt` = $2^{20} + 1$
 - `ele_size` = $4096 = 2^{12}$
 - Выделится = ??
- Как сделать функцию безопасной?

Компилированный код умножения

Функция Си

```
int mul12(int x)
{
    return x*12;
}
```

Арифметические операции в результате компиляции

```
leal (%eax,%eax,2), %eax
sall $2, %eax
```

Пояснение

```
t <- x+x*2
return t << 2;
```

- Компилятор Си автоматически создаёт код сдвига и сложения при умножении на константу

Компилированный код деления беззнакового

Функция Си

```
unsigned udiv8(unsigned x)
{
    return x/8;
}
```

Арифметические операции в результате компиляции

```
shrl $3, %eax
```

Пояснение

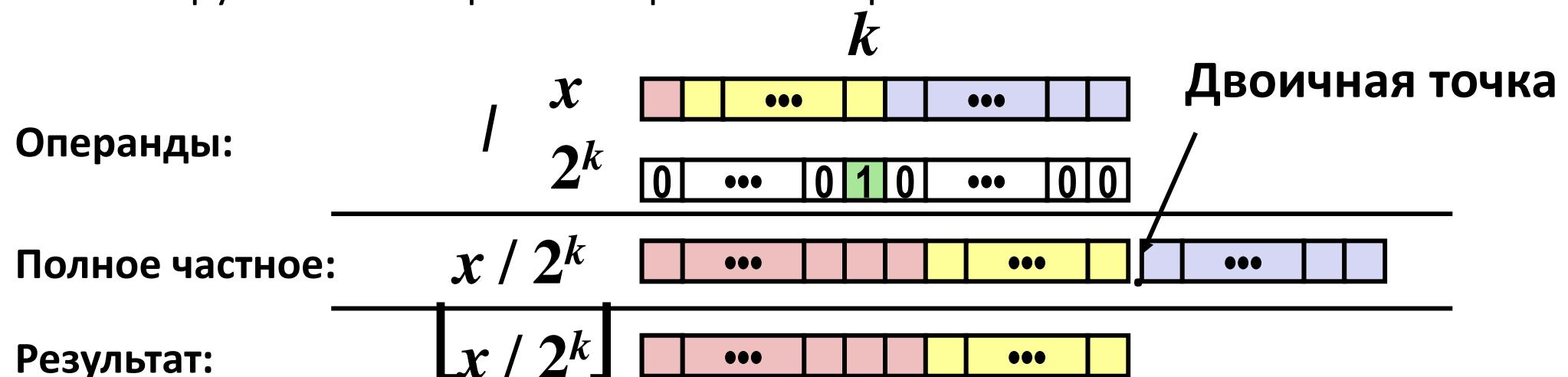
```
# Логический сдвиг
return x >> 3;
```

- Для беззнаковых используется логический сдвиг
- Для пользователей Явы
 - Логический сдвиг пишется как >>>

Деление сдвигом знакового на степень двойки

■ Частное от деления знакового на степень двойки

- $x \gg k$ даёт $\lfloor x / 2^k \rfloor$
- Использует арифметический сдвиг
- Округляет в неверном направлении при $u < 0$



	Полное частное	Результат	Шестн.	Двоичный
у	-15213	-15213	C4 93	11000100 10010011
у \gg 1	-7606.5	-7607	E2 49	11100010 01001001
у \gg 4	-950.8125	-951	FC 49	11111100 01001001
у \gg 8	-59.4257813	-60	FF C4	11111111 11000100

Правильное деление на степень двойки(1)

■ Частное от деления знакового на степень двойки

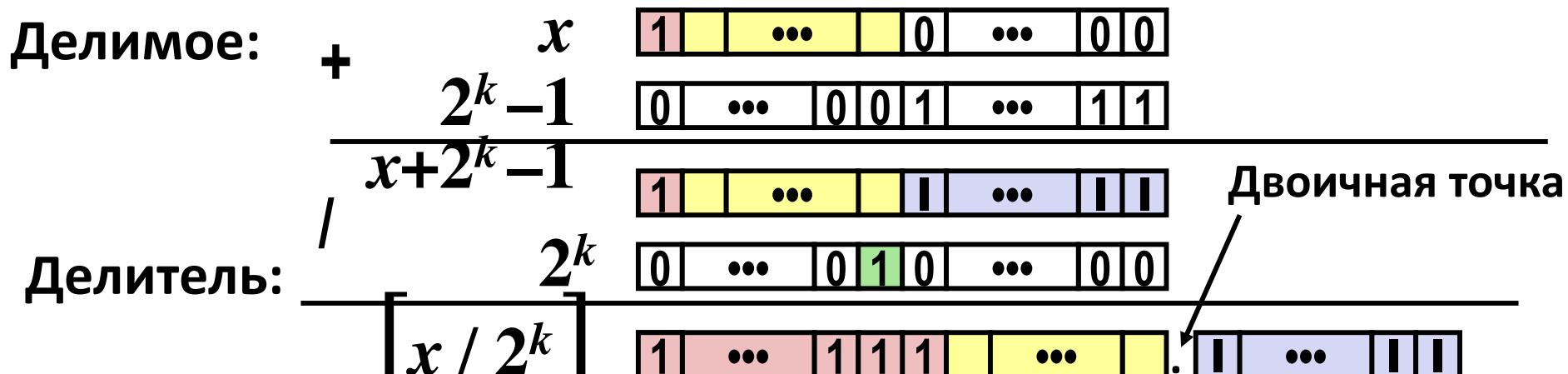
- Необходимо $\lfloor x/2^k \rfloor$ (Округление к 0)

- Вычисляется как $\lfloor (x+2^k-1)/2^k \rfloor$

- В Си: $(x + (1<<k) - 1) >> k$

- Смещение делимого к 0

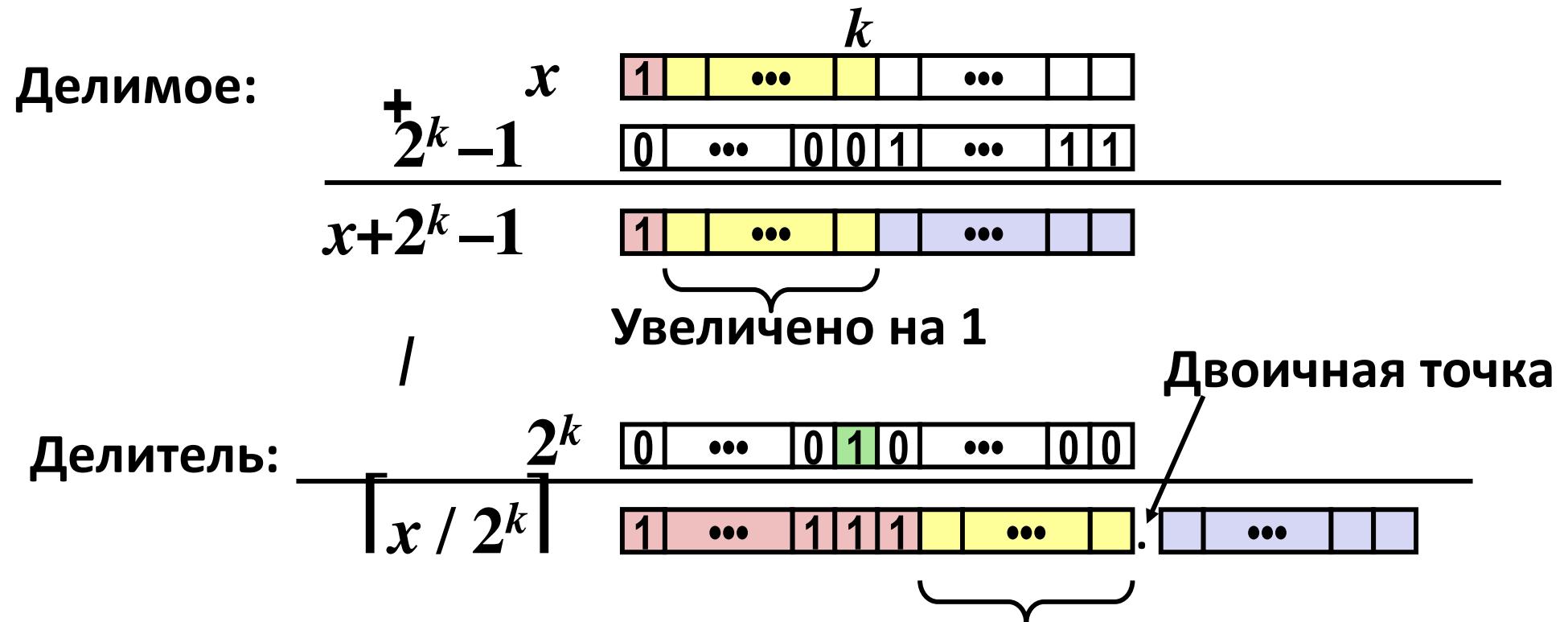
Вариант 1: Без округления



Смещение не имеет эффекта

Правильное деление на степень двойки(2)

Вариант 2: округление



Смещение добавляет 1 к результату

Компилированный код деления знакового

Функция Си

```
int idiv8(int x)
{
    return x/8;
}
```

Арифметические операции в результате компиляции

```
testl %eax, %eax
js L4
L3:
    sarl $3, %eax
    ret
L4:
    addl $7, %eax
    jmp L3
```

Пояснения

```
if x < 0
    x += 7;
# Арифметический сдвиг
return x >> 3;
```

- Арифметический сдвиг int
- Для пользователей Ява
 - Арифм. сдвиг пишется как >>

Арифметика: основные правила(2)

- Беззнаковые и знаковые в доп.коде – изоморфные кольца:
изоморфизм = преобразование типов
- Сдвиг влево
 - Беззнаковые: умножение на 2^k
 - Всегда логический сдвиг
- Сдвиг вправо
 - Беззнаковые: логический сдвиг, деление на 2^k с округлением к 0
 - Знаковые: арифметический сдвиг
 - Положительные: деление на 2^k с округлением к 0
 - Отрицательные: деление на 2^k с округлением от 0
используется смещение для исправления

Свойства арифметики беззнаковых

■ Умножение и сложение беззнаковых образуют коммутативное кольцо

- Сложение – коммутативная группа
- Замкнутая относительно умножения

$$0 \leq \text{UMult}_w(u, v) \leq 2^w - 1$$

- Умножение – коммутативно

$$\text{UMult}_w(u, v) = \text{UMult}_w(v, u)$$

- Умножение – ассоциативно

$$\text{UMult}_w(t, \text{UMult}_w(u, v)) = \text{UMult}_w(\text{UMult}_w(t, u), v)$$

- 1 – нейтральный элемент умножения

$$\text{UMult}_w(u, 1) = u$$

- Умножение дистрибутивно относительно сложения

$$\text{UMult}_w(t, \text{UAdd}_w(u, v)) = \text{UAdd}_w(\text{UMult}_w(t, u), \text{UMult}_w(t, v))$$

Свойства арифметики знаковых в доп.коде

■ Изморфные алгебры

- Беззнаковые умножение и сложения
 - Окращаются до w
- Two's complement multiplication and addition
 - Truncating to w bits

■ Обе образуют кольца

- Изоморфные кольцу вычетов по модулю 2^w

■ Сравнение с арифметикой (математических) целых

- И там, и там – кольца
- Целые обладают свойствами упорядоченности, т.е.,

$$u > 0 \quad \Rightarrow \quad u + v > v$$

$$u > 0, v > 0 \quad \Rightarrow \quad u \cdot v > 0$$

- Этими свойствами не обладают беззнаковые в доп.коде

$$TMax + 1 == Tmin$$

$$15213 * 30426 == -10030 \quad (\text{для 16-битных слов})$$

Чтение байт в обратном порядке

■ Результат дизассемблирования

- Текстовое представление машинного кода
- Выдаётся программой читающей машинный код

■ Пример фрагмента

Адрес	Машинный код	Представление на Ассемблере
8048365:	5b	pop %ebx
8048366:	81 c3 ab 12 00 00	add \$0x12ab,%ebx
804836c:	83 bb 28 00 00 00 00	cmpl \$0x0,0x28(%ebx)

■ «Расшифровка» значений

- Значение:
- Размещение в слове (32 бита):
- Разделение на байты:
- Переупорядочение:

0x12ab
0x000012ab
00 00 12 ab
ab 12 00 00