

Представление целых

Основы информатики

Компьютерные основы программирования

`u.to/DbCmFA`

На основе CMU 15-213/18-243:

Introduction to Computer Systems

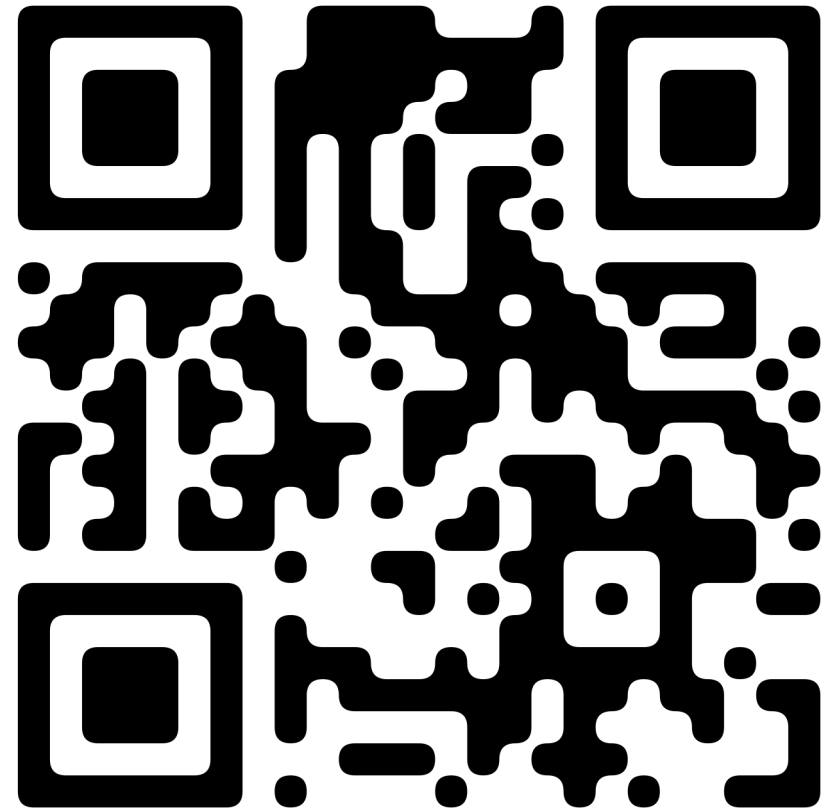
`u.to/XoKmFA`

Лекция 2, 12 февраля, 2024

Лектор:

Дмитрий Северов, кафедра информатики 608 КПМ

`cs.mipt.ru/wp/?page_id=346`

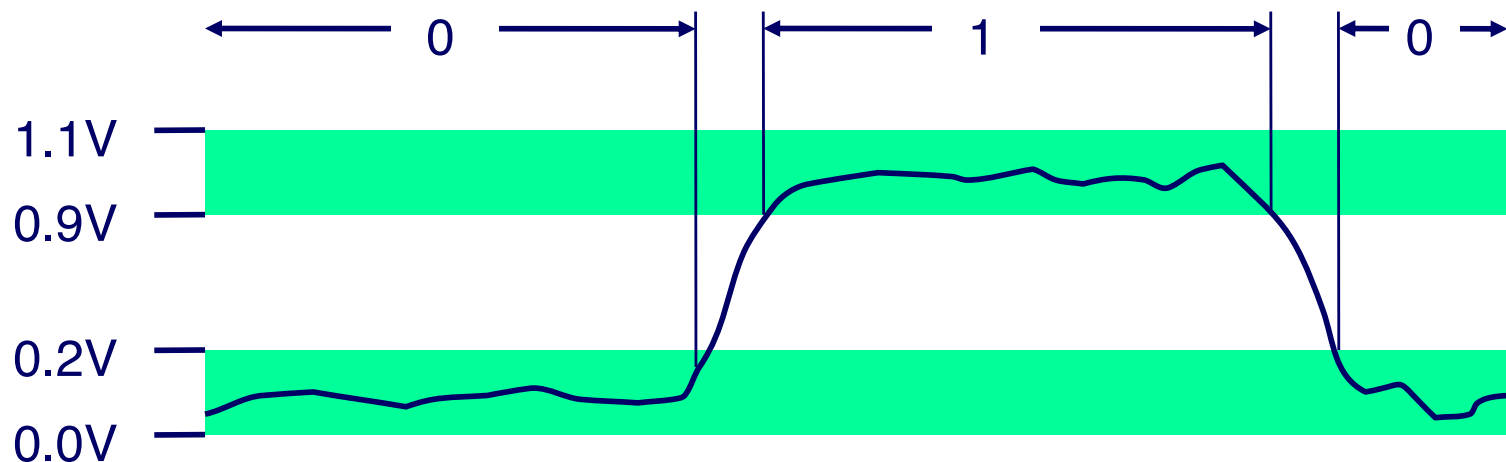


Биты, байты и целые

- **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 удерживать бистабильными элементами
 - Надёжно передаются по плохим и зашумленным проводникам



Двоичный пример

■ Представление чисел в двоичной системе

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

Кодирование значений байта

■ Байт = 8 бит

- Двоичное от 00000000_2 до 11111111_2
- Десятичное: от 0_{10} до 255_{10}
- Шестнадцатеричное от 00_{16} до FF_{16}
 - Представление 16 цифр
 - Используем символы от '0' до '9' и от 'A' до 'F'
 - Запись $FA1D37B_{16}$ в Си как
 - `0xFA1D37B`
 - `0xfa1d37b`

Шестнадцатеричное 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 даных языка C	32-бит типично	64-bit типично	x86-64
<code>char</code>	1	1	1
<code>short</code>	2	2	2
<code>int</code>	4	4	4
<code>long</code>	4	8	8
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
 - Кодировать “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

НЕ(Not)

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

\sim	
0	1
1	0

ИЛИ (Or)

- $A \mid B = 1$ когда либо $A=1$, либо $B=1$

\mid	0	1
0	0	1
1	1	1

Исключающее ИЛИ (Xor)

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

\wedge	0	1
0	0	1
1	1	0

Обобщение булевой алгебры

■ Операции на битовых наборах (векторах)

- Операции выполняются побитово

01101001	01101001	01101001	
& 01010101	01010101	^ 01010101	~ 01010101
<hr/>	<hr/>	<hr/>	<hr/>
01000001	01111101	00111100	10101010

■ 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 $\&$, $|$, \sim , \wedge 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` \Rightarrow `0x00`
- `!0x00` \Rightarrow `0x01`
- `!!0x41` \Rightarrow `0x01`

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

Сравните: логические операции в С

■ Логические операторы

- `&&`, `||`, `!`
 - 0 кодируется как “False”
 - Всё, что не 0, кодируется как “True”

Внимание!

**`&&` вместо `&` (и `||` вместо `|`)...
одна из самых частых ошибок
программирования на С**

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

Операции сдвига в Си

■ Сдвиг влево: $X \ll u$

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

■ Сдвиг вправо: $X \gg u$

- Сдвигает вектор битов X вправо на u 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
$\ll 3$	00010 <u>000</u>
Логич. $\gg 2$	<u>00</u> 011000
Ариф. $\gg 2$	<u>00</u> 011000

Аргумент x	10100010
$\ll 3$	00010 <u>000</u>
Логич. $\gg 2$	<u>00</u> 101000
Ариф. $\gg 2$	<u>11</u> 101000

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
x	15213	3B 6D	00111011 01101101
y	-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

- Одинаковые кодировки неотрицательных величин

■ Взаимная однозначность

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

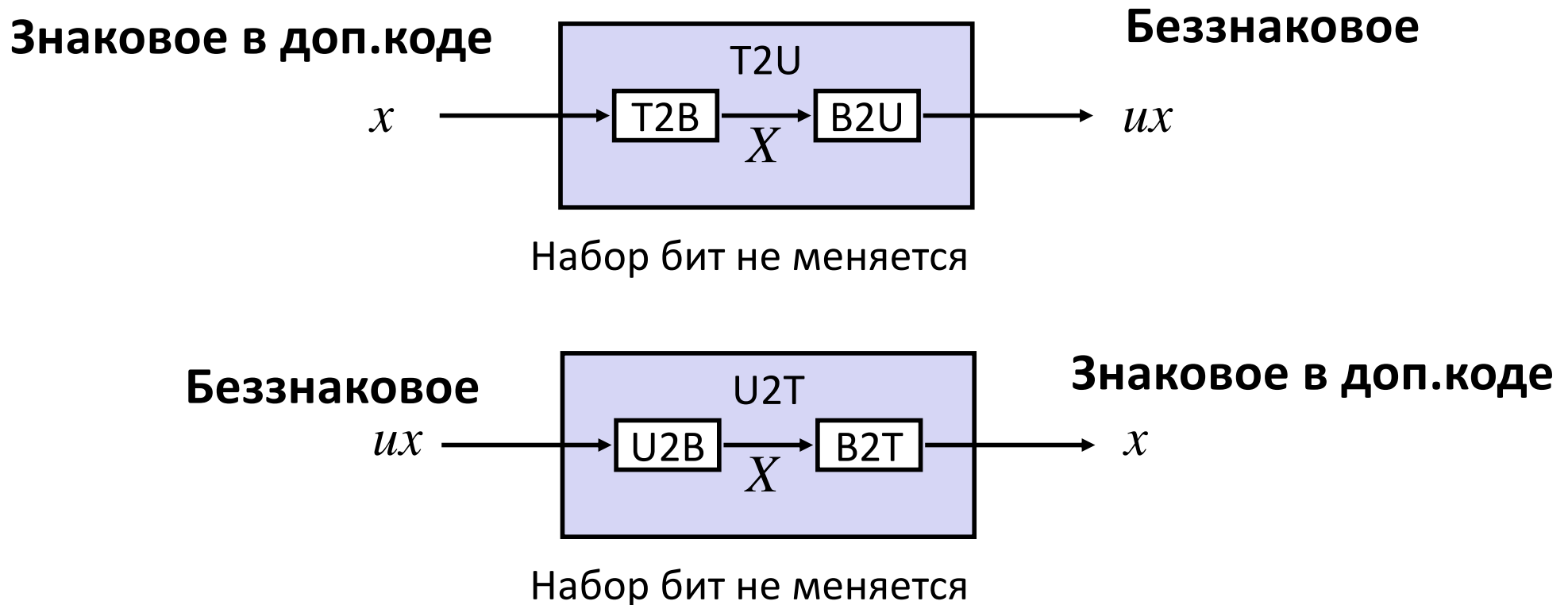
■ Отображения обратимы

- $\forall x, U2B(B2U^{-1}(x)) = x$
 - биты беззнаковых
- $\forall x, T2B(B2T^{-1}(x)) = x$
 - биты 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



- 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

→

T2U

→

←

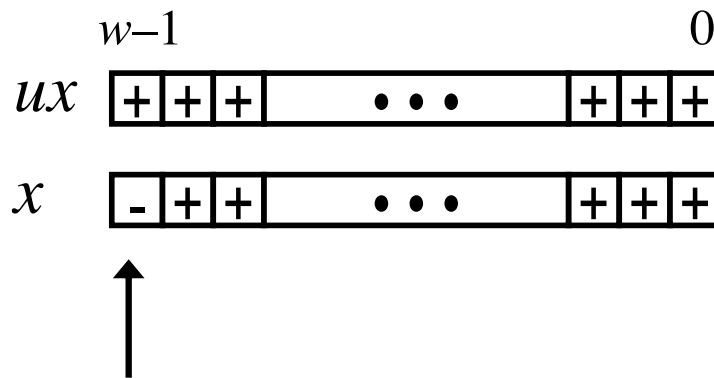
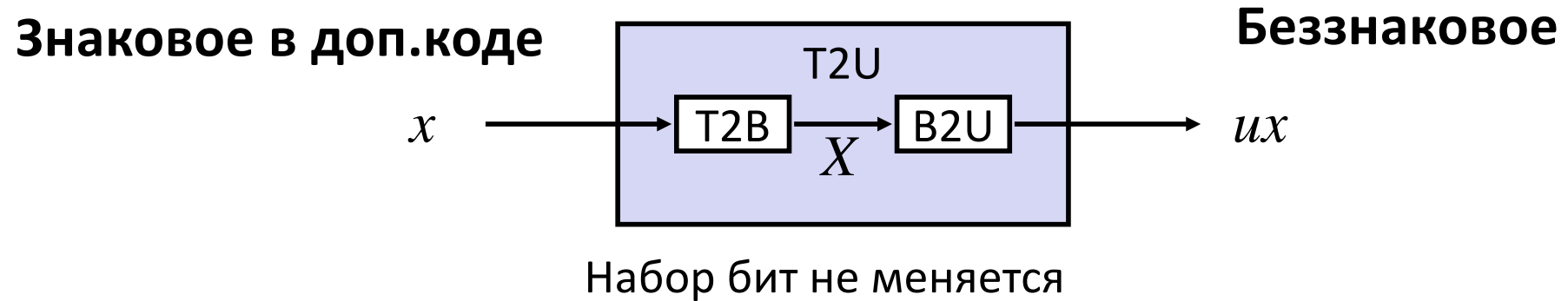
U2T

←

Соответствие знаковых и беззнаковых

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

Связь между знаковыми и беззнаковыми

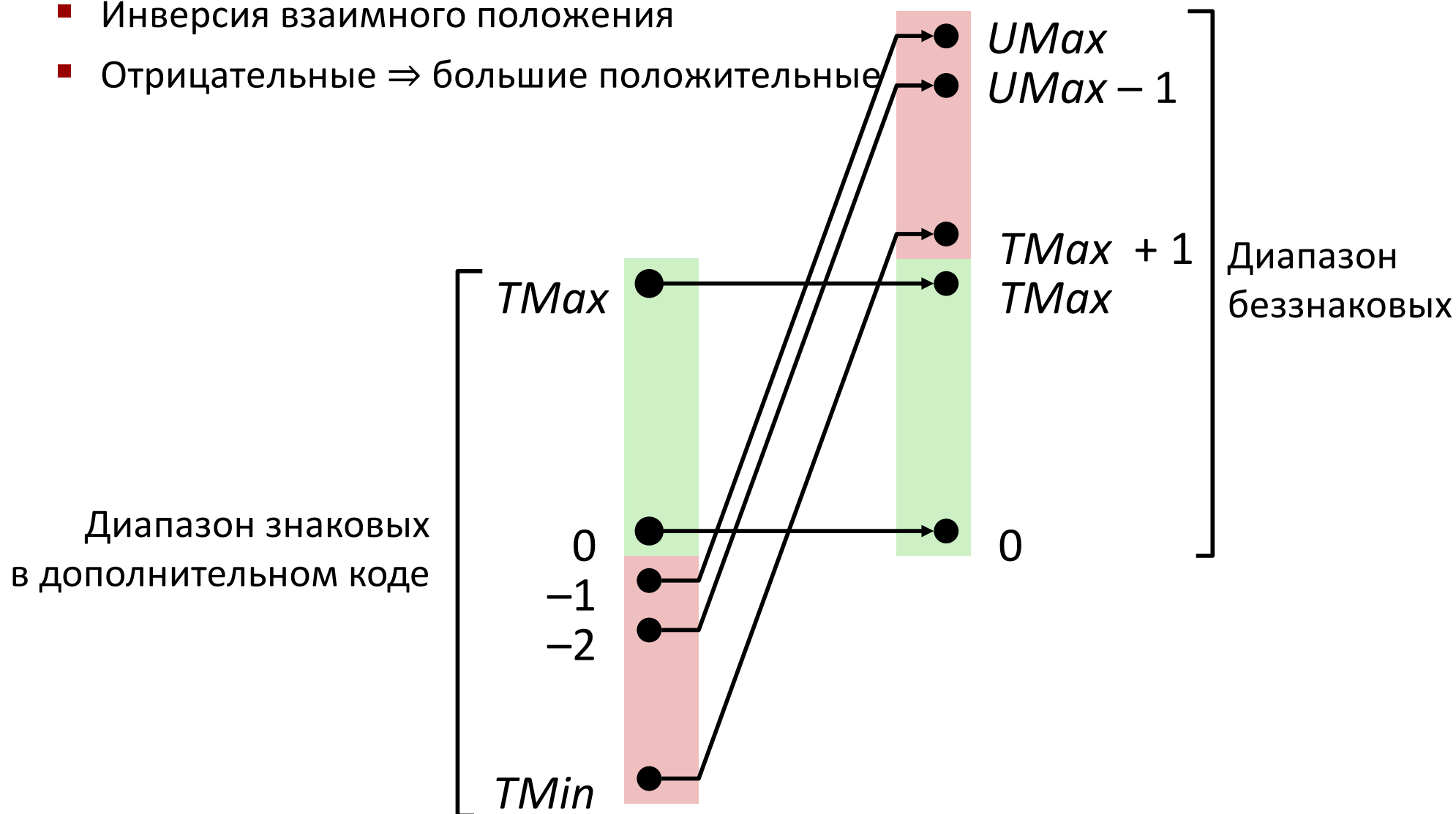


самый отрицательный бит
одиноким становится
самым положительным

Визуализация преобразования

■ Дополнительный \Rightarrow беззнаковый

- Инверсия взаимного положения
- Отрицательные \Rightarrow большие положительные



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 вызовах процедур, арифметике, сравнениях, присваиваниях

```
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

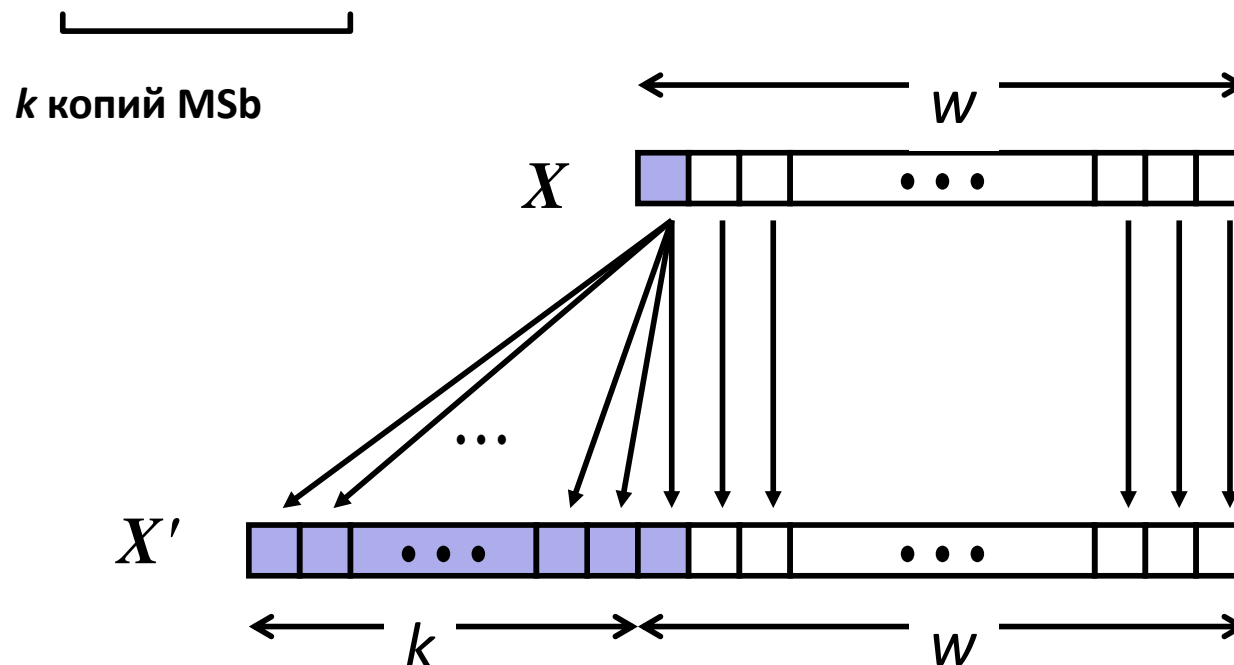
Расширение знака

■ Задача:

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

■ Правило:

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



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

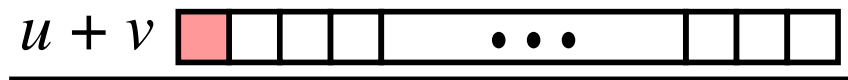
- Представление информации в битах
- Манипуляции на уровне бит
- **Целые**
 - Представление: беззнаковое и знаковое
 - Преобразования
 - Расширение, сокращение
 - Сложение, отрицание, умножение, сдвиг
 - Сводка
- Представление в памяти, указатели, строки

Беззнаковое сложение

Операнды: w бит



Полная сумма: $w+1$ бит



Без переноса: w бит



■ Стандартная операция сложения в Си

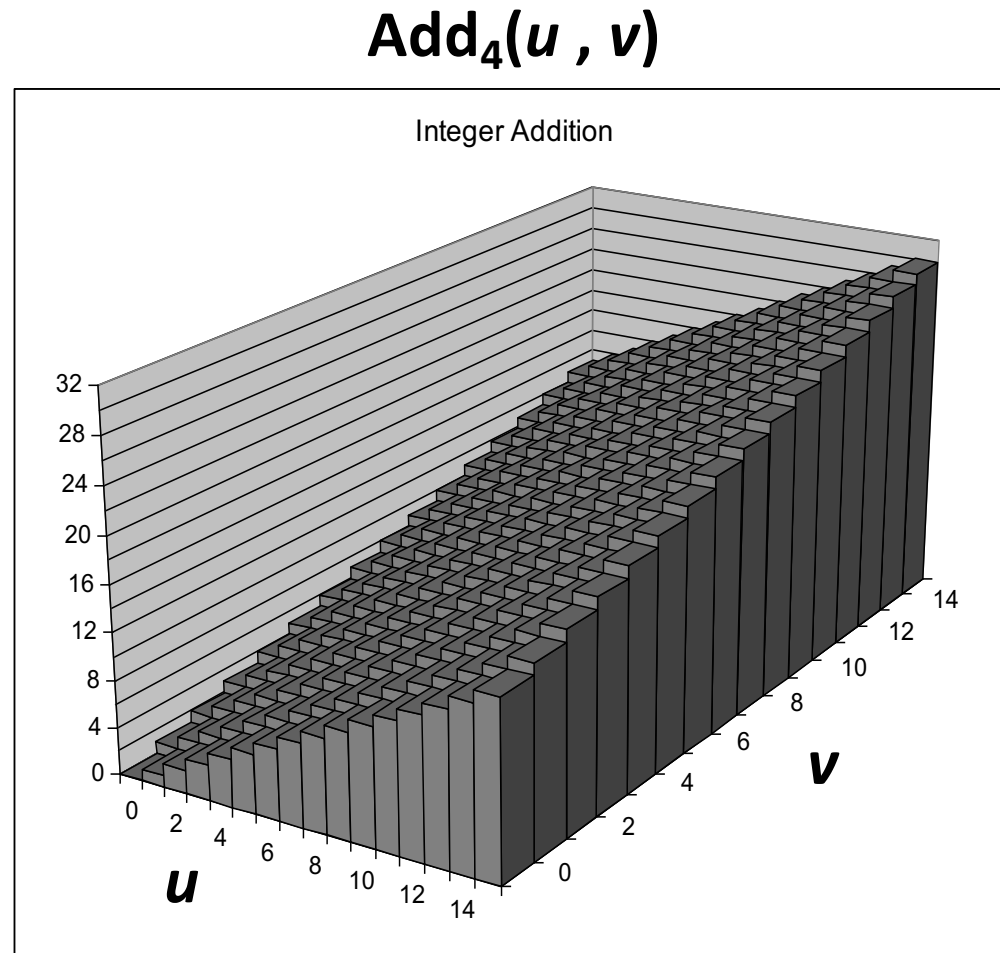
- «Игнорирует» перенос
- Реализует сложение по модулю

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

Математическое сложение визуально

■ Сложение целых

- 4-х битовые целые u, v
- Полная сумма $Add_4(u, v)$
- Значение растёт линейно по u и v
- Образует плоскость

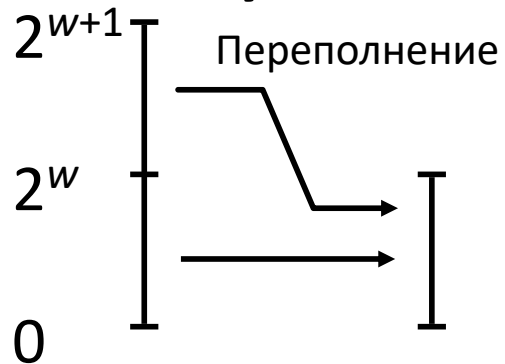


Беззнаковое сложение визуально

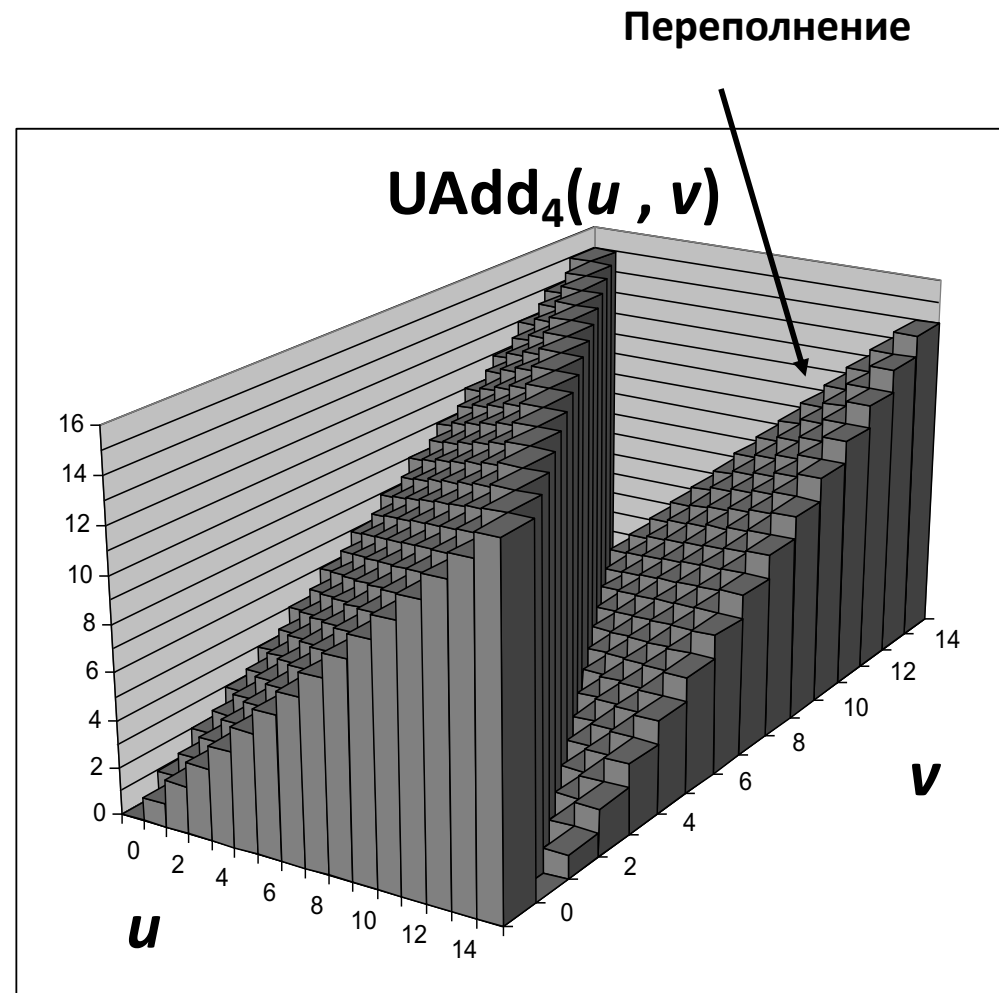
■ Усечение

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

Полная сумма



Сумма по модулю

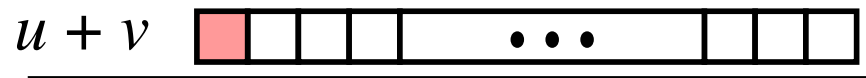


Сложение в дополнительном коде

Операнды: w бит



Полная сумма: $w+1$ бит



Без преноса: w бит



■ 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

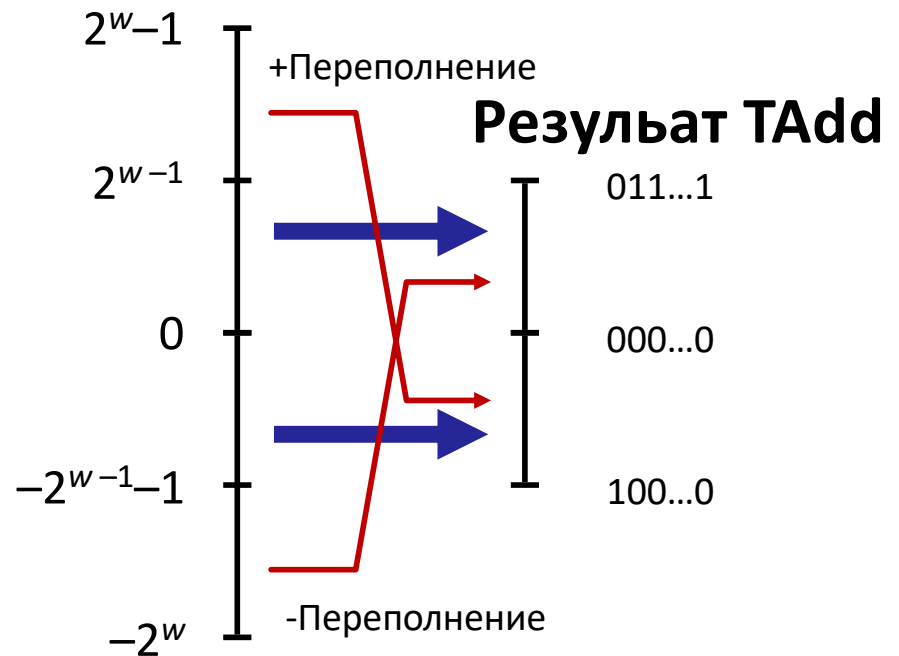
0 100...0

0 000...0

1 011...1

1 000...0

Полная сумма



Сложение в доп. коде визуально

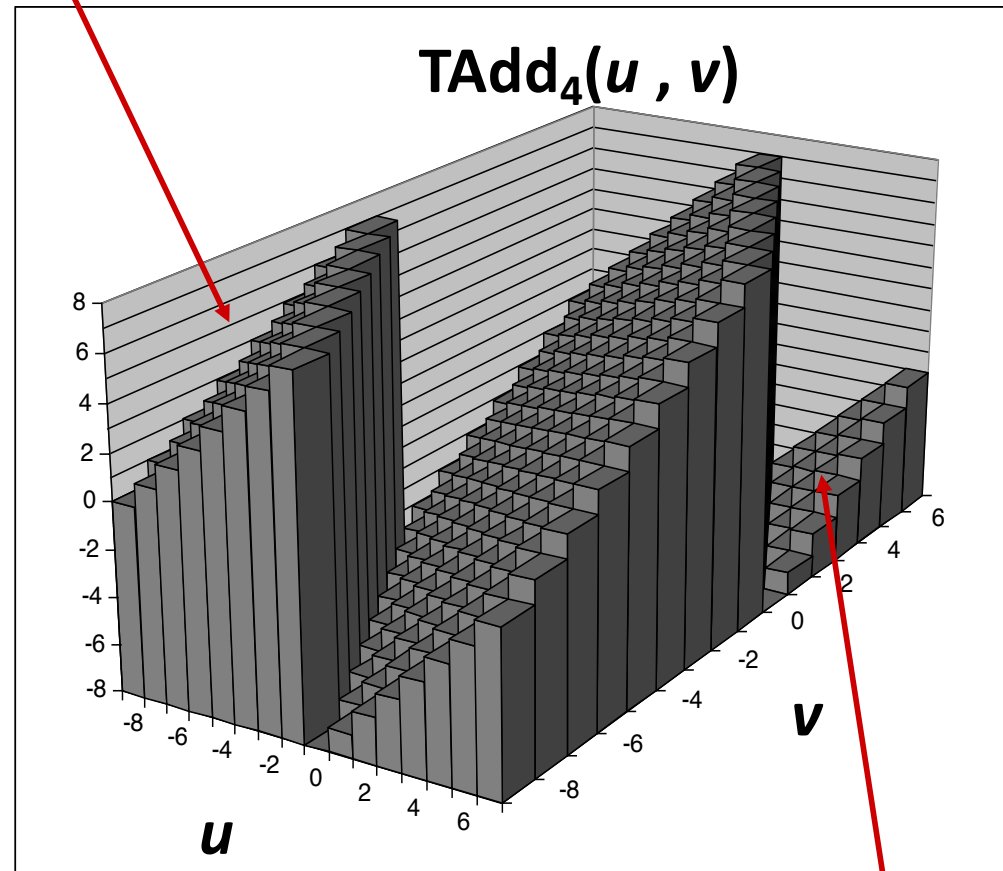
■ Значения

- 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

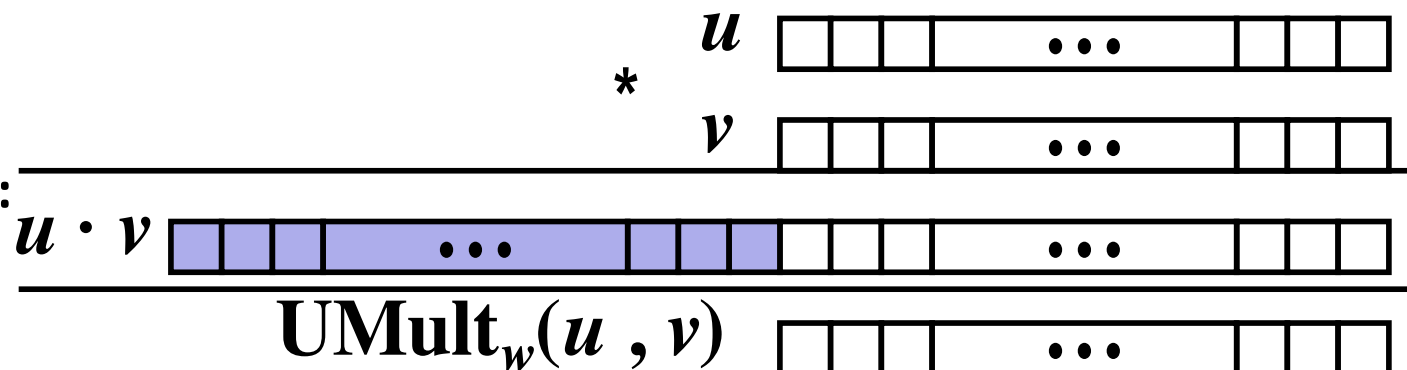
- Необходимо расширение слова для каждого результата
- Реализуется лишь в программных пакетах арифметики “произвольной точности”

Умножение беззнаковых

Операнды: w бит

Полное произведение:
 2^*w бит

Без w старших бит:
 w младших бит



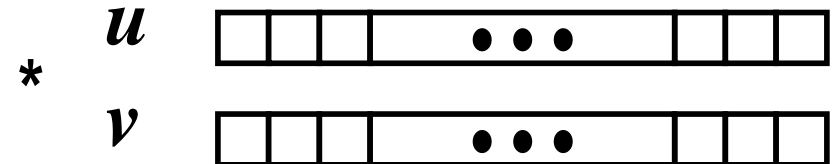
■ Стандартное умножение беззнаковых в Си

- Игнорирует старшие w бит
- Реализует умножение по модулю

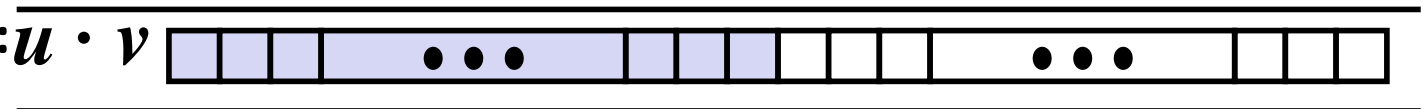
$$\text{UMult}_w(u, v) = (u \cdot v) \bmod 2^w$$

Умножение знаковых

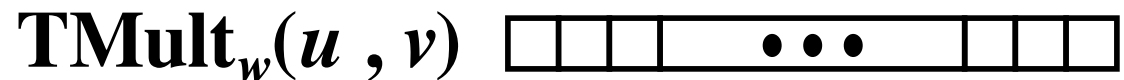
Операнды: w бит



Полное произведение: $u \cdot v$
 $2 \cdot w$ бит



Без w старших бит:
 w младших бит



■ Стандартное умножение знаковых в Си

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

Умножение сдвигом на степень двойки

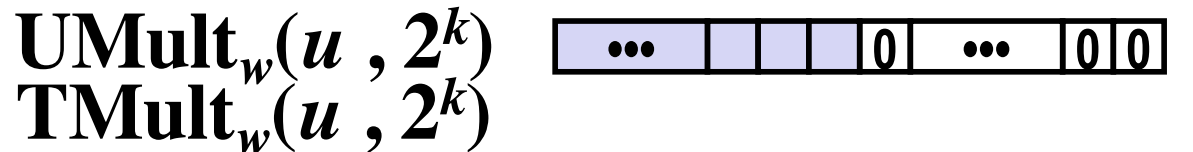
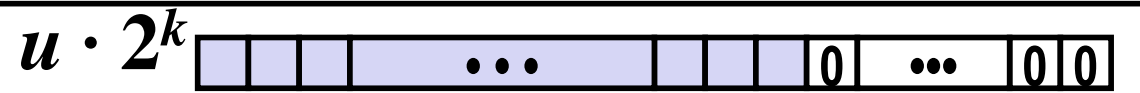
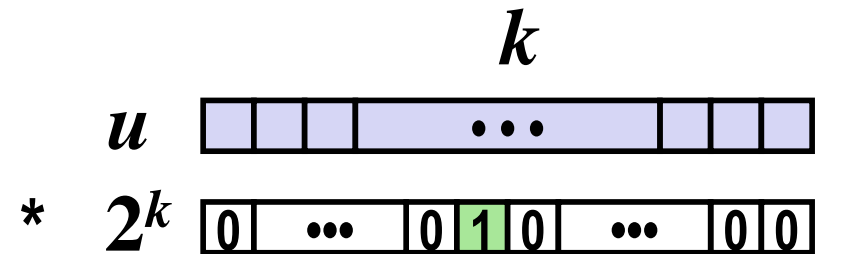
■ Операция

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

Операнд: w бит

Полное произведение:
 $w+k$ бит

Без k старших бит:
 w младших бит



■ Примеры

- $u \ll 3 \quad \quad \quad == \quad u * 8$
- $u \ll 5 - u \ll 3 \quad \quad == \quad u * 24$
- 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`?

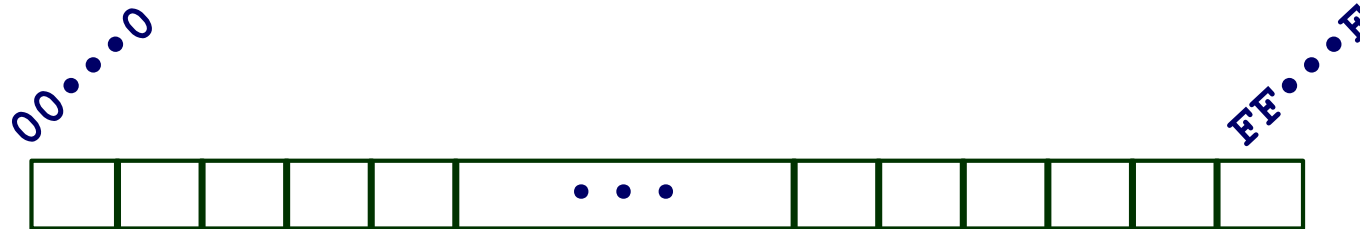
Зачем использовать беззнаковые? (ещё)

- ***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 выполняется 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

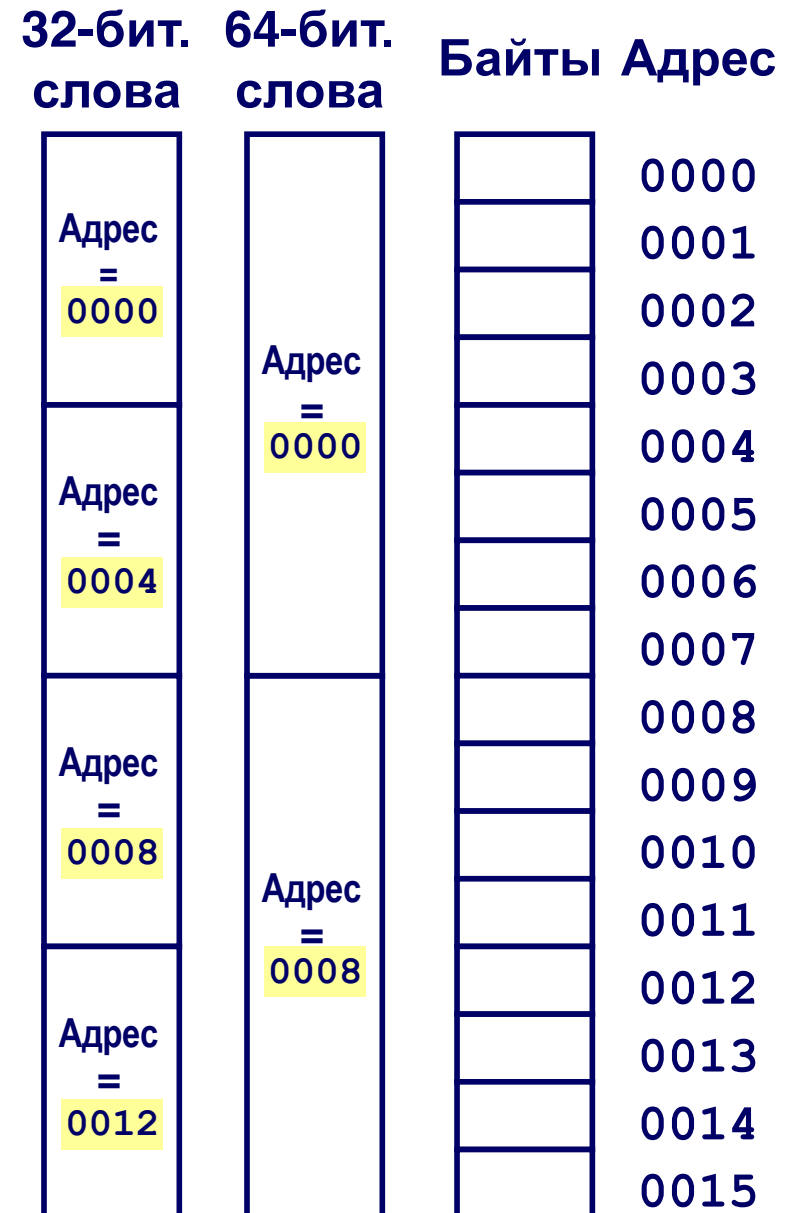
Машинные слова

■ С машиной связан “размер слова”

- Обычный размер представления целых чисел
 - Включая адреса
- Большинство мобильных ещё 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 C	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
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		
		01	23	45	67		

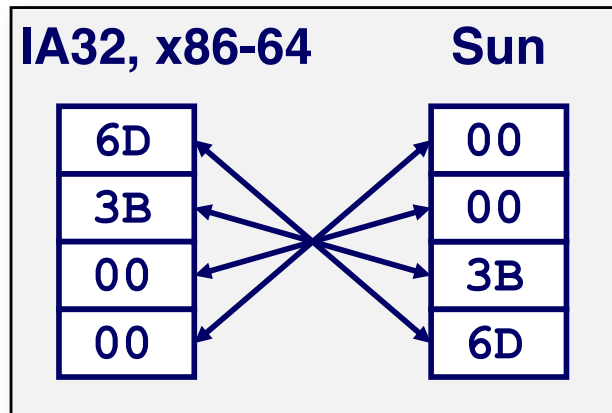
«Остроконечное»

		0x100	0x101	0x102	0x103		
		67	45	23	01		

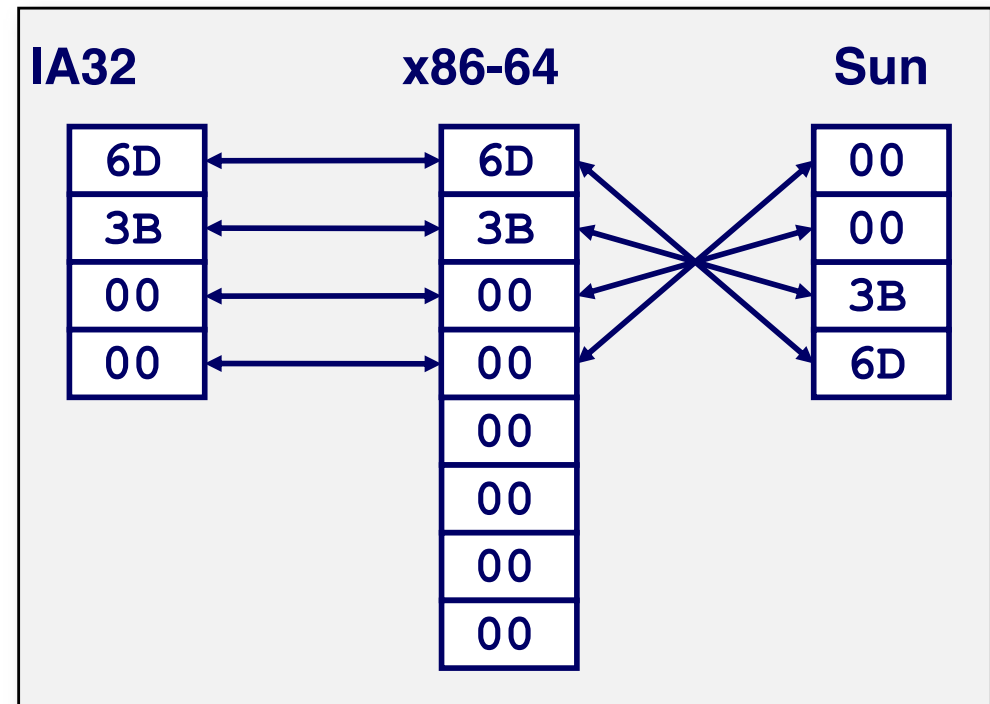
Представление Целочисленное

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

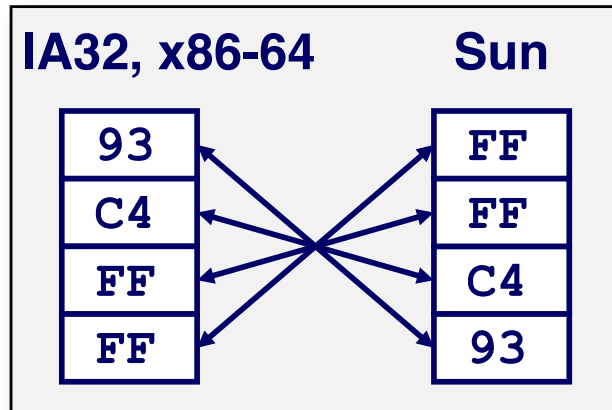
`int A = 15213;`



`long int C = 15213;`



`int B = -15213;`



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;
```

Sun	IA32	x86-64
EF	D4	0C
FF	F8	89
FB	FF	EC
2C	BF	FF
		FF
		7F
		00
		00

Различные компиляторы, ОС и машины дают различное расположение в памяти

Представление строк

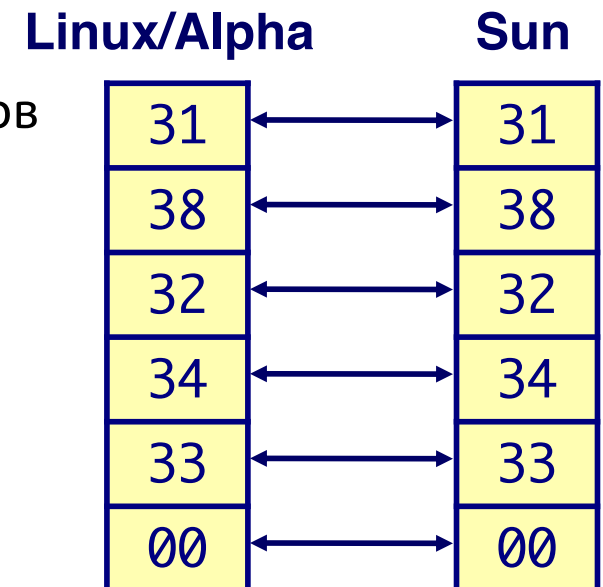
```
char S[6] = "18243";
```

■ 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 объясните почему оно...

- либо верно для любых значений аргументов,
- либо неверно.

Инициализация

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

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