Компьютерные основы программирования Представление данных часть1

Лекция 2, 2 марта 2017

Лектор: Чуканова Ольга Владимировна

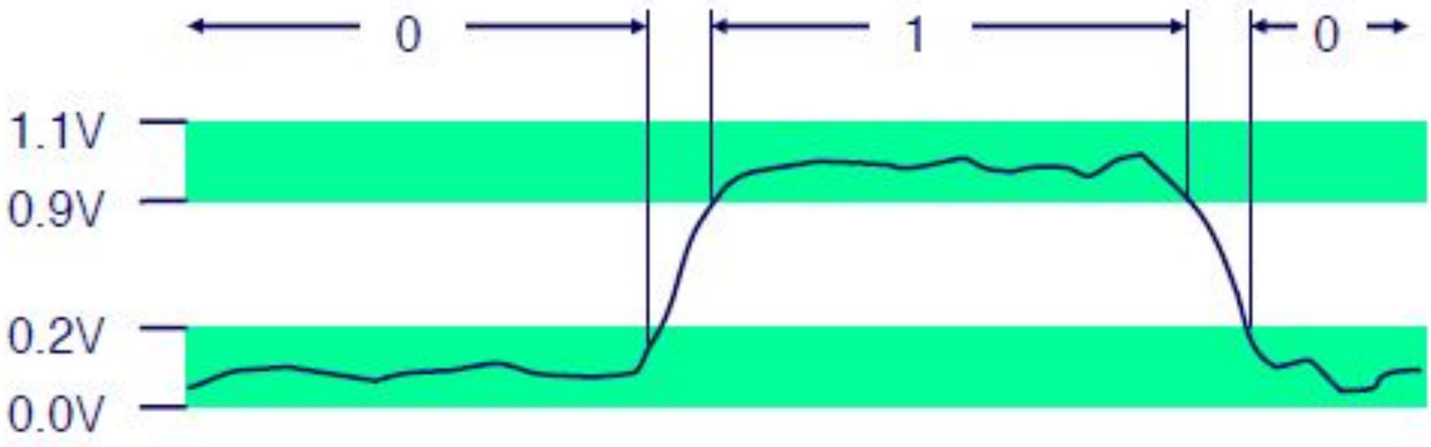
Кафедра информатики

602 АК

ovcha@mail.ru

(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

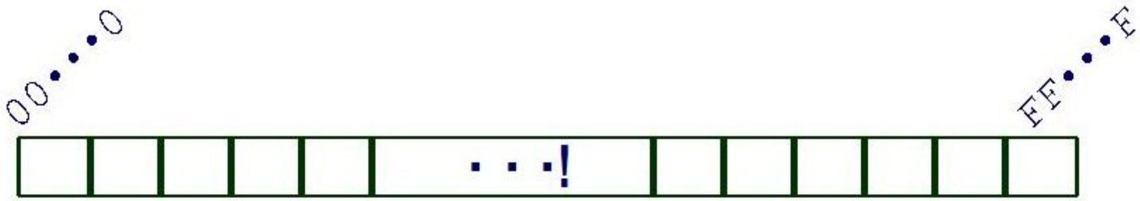
- **Представление чисел в двоичной системе**
  - $15213_{10} = 11101101101101_2$
  - $1.20_{10} = 1.0011001100110011[0011]..._2$
  - $1.5213 \text{ X } 10^4 = 1.1101101101101_2 \text{ X } 2^{13}$

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 $FA1D37B_{16}$ 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

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

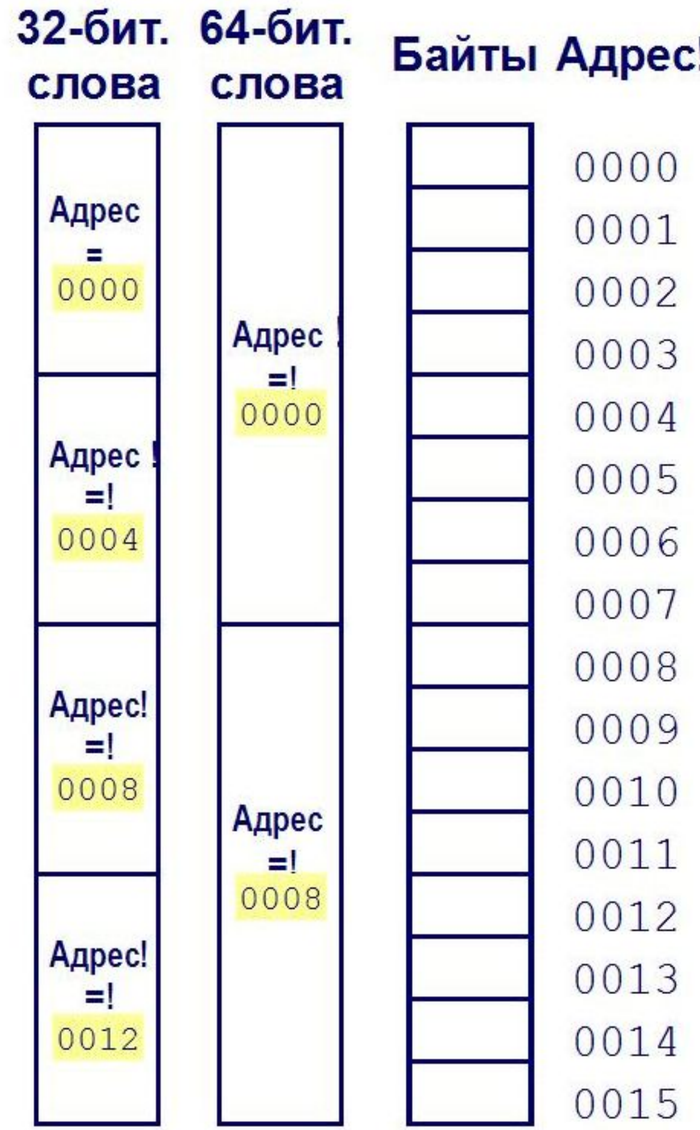
Машинные слова

- **С машиной связан “размер слова”**

- Обычный размер представления целых чисел
  - Включая адреса
- Большинство машин используют слова в 32 бита (4 байта)
  - Предел адресации 4ГБ
  - Недостаточно для интенсивной работы с памятью
- Мощные системы - слова используют в 64 бита (8 байт)
  - Потенциальное адресное пространство порядка 1.8 X 1019 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)



Форматы данных

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
pointer	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

«Тупоконечники»: Sun, PPC Mac, Internet

Наименее значимый байт имеет <u>наибольший</u> адрес

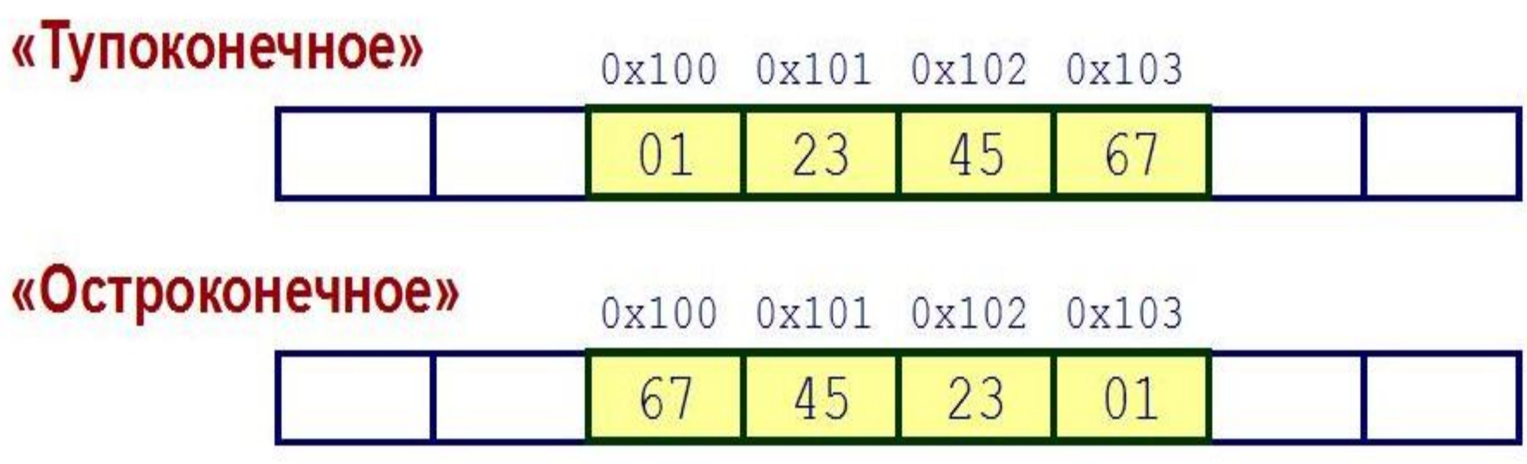
«Остроконечники»: x86

Наименее значимый байт имеет <u>наименьший</u> адрес

Пример

Переменная x имеет 4-байтовое представление 0x01234567

Расположена по адресу &x - 0x10



Чтение байт в обратном порядке

- **Результат дизассемблирования**

Текстовое представление машинного кода

Выдаётся программой читающей машинный код

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:	0x12ab
Размещение в слове (32 бита):	0x000012ab
Разделение на байты:	00 00 12 ab
Переупорядочение:	ab 12 00 00

Изучение представления данных

- **Вывод байтового представления данных** – Представление указателя как массива

```
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 для int

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

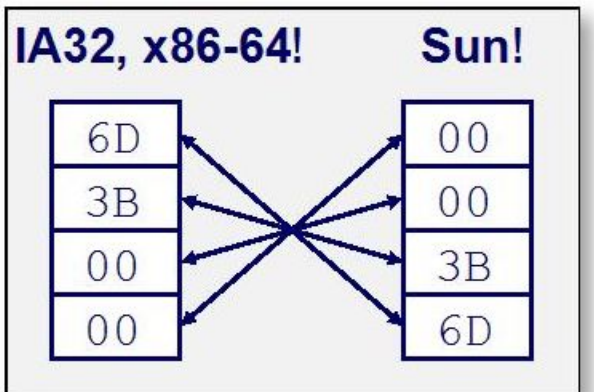
Результат (x86, Linux):!

```
int a = 15213;
0x 1ffffcb8 0x6d
0x 1ffffcb9 0x3b
0x 1ffffcba 0x00
0x 1ffffcbb 0x00
```

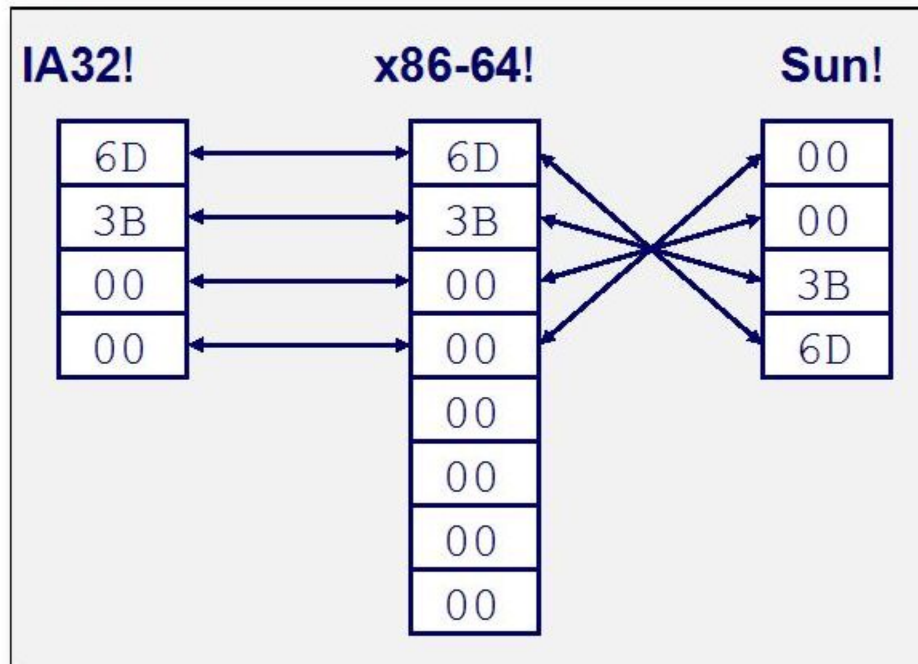
Целочисленное представление

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

`int A = 15213;`



`long int C = 15213;`



Представлениеуказателей

```
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

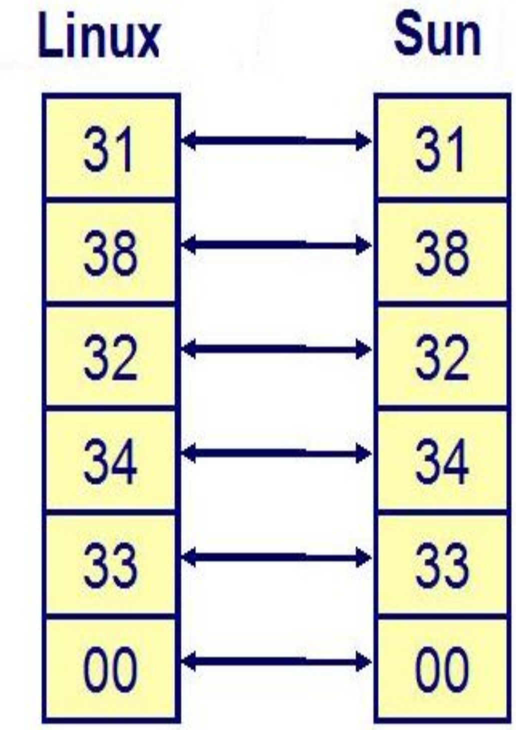
- Различные компиляторы, ОС и машины дают различное расположение в памяти

Представление строк

- **Строки в С**
- Представлены массивами символов
  - Каждый символ представлен ASCII-кодом
  - Стандартное кодирование набора символов буквы от A до Zимеют коды 0x41 до 0x5A
  - Символ "0" кодируется 0x30
  - Цифра i кодируется 0x30+i
  - Строки должны завершаться нулевым кодом
  - Символ окончания строки = 0

Пример

```
char *s="18243";
show_bytes(s, strlen(s));
```



ASCII - American National Standard Code for Information Interchange

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_a	_b	_c	_d	_e	_f
0_	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	nl	vt	np	cr	so	si
1_	dle	dcl	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
2_	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6_	`	a	b	c	d	e	f	g	h	I	j	k	l	m	n	o
7_	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del

Булева алгебра

- **Предложена Джорджем Булем в XIX веке**
  Алгебраическое представление логики
  Кодирует “Истина” как 1 и “Ложь” как 0

И (And)

1 когда оба A=1 and B=1

&	0	1
0	0	0
1	0	1

ИЛИ (Or)

A|B = 1 когда A&B = либо A=1, либо B=1

|	0	1
0	0	1
1	1	1

НЕ(Not)

~A = 1 когда A=0

~	
0	1
1	0

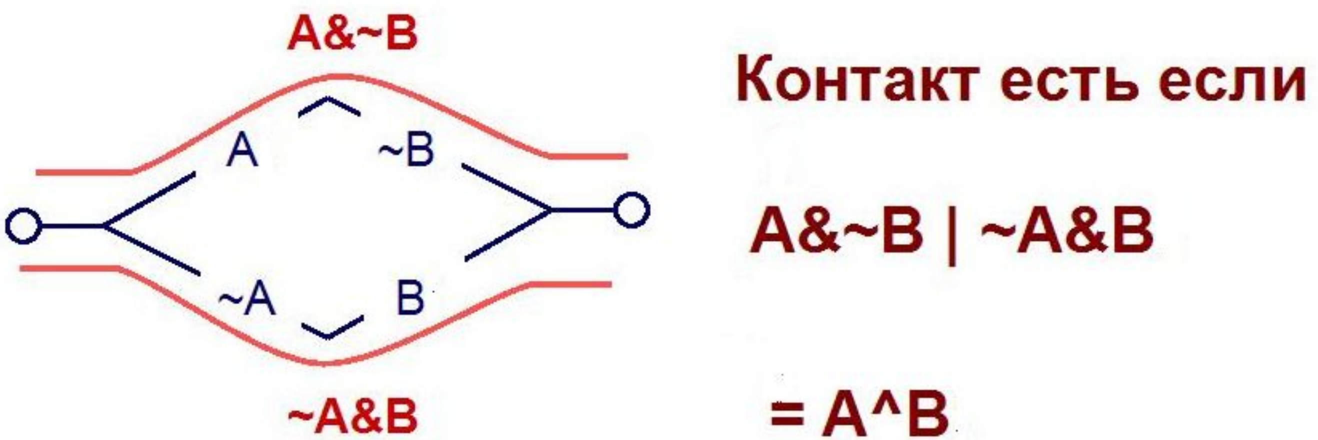
Исключающее ИЛИ (Xor)

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

^	0	1
0	0	1
1	1	0

Приложение булевой алгебры

- **Клодом Шенноном применена к цифровым системам**
- Диплом MIT 1937
- Рассмотрены схемы реле
- Замкнутый контакт кодируется как 1, разомкнутый как 0



Операции на уровне бита в С

- **Обобщение булевой алгебры**

Операции на битовых наборах (векторах)

- Операции выполняются побитово

```
   01101001     01101001     01101001
&  01010101   | 01010101   ^ 01010101   ~ 01010101
   --------     --------     --------     --------
   01000001     01111101     00111100     10101010
```

Применимы все выводы булевой алгебры

Представление и операции с множествами

- –

Ве

$a_j = 1$ if $j \in A$

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

Процедура перестановки

```
void inplace_swap(int *x, int *y)
{*x=(*x)^(*y); /*step 1*/
  *y =(*x)^(*y); /*step 2*/
   *x=(*x)^(*y); /*step 3*/
}
```

Данная процедура основана на том факте, что ИСКЛЮЧИТЕЛЬНОЕ ИЛИ — коммутативно и ассоциативно, и что $a \wedge a = 0$ для любого значения a. *В главе 5* описывается случай некорректной работы кода, когда два указателя x и y равны (указывают на одно и то же местоположение).

Шаг	*x	*y
Первоначально	a	b
Шаг 1	$a \wedge b$	b
Шаг 2	$a \wedge b$	$(a \wedge b) \wedge b = (b \wedge b) \wedge a = a$
Шаг 3	$(a \wedge b) \wedge a = (a \wedge a) \wedge b = b$	a

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)

Операторы сдвига

Операции сдвига в Си

- **Сдвиг влево:** **x << y**
  - Сдвигает вектор битов X влево на Y позиций
    - Вытолкнутые слева биты теряются
    - Заполняет нулями справа
- **Сдвиг вправо:** **x >> 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
<< 3	00010000
Логич. >> 2	00011000
Ариф. >> 2	00011000

Аргумент x	10100010
<< 3	00010000
Логич. >> 2	00101000
Ариф. >> 2	11101000

Целочисленное представление **натуральные числа без знака**

0 1 … 254 255

$11111111_2 =$
$FF_{16} =$
$15*16+15 =$
255_{10}

255 0 1 - для 8 двоичных символов

Типы целого в C

Описание C	Обеспечено		Типичные 32-битовые	
	Минимум	Максимум	Минимум	Максимум
`char`	–127	127	–128	127
`unsigned char`	0	255	0	255
`short [int]`	–32 767	32 767	–32 768	32 767
`unsigned short [int]`	0	63 535	0	63 535
`int`	–32 767	32 767	–2 147 483 648	2 147 483 647
`unsigned [int]`	0	63 535	0	4 294 967 295
`long [int]`	–2 147 483 647	2 147 483 647	–2 147 483 648	2 147 483 647
`unsigned long [int]`	0	4 294 967 295	0	4 294 967 295

Представление отрицательных чисел

- путем простого выделения бита под знак числа

0******* - положительное число

1******* - отрицательное число

Результат:

00000000 = +0   10000000 = -0

- путем записи в дополнительном коде: x + (-x) = 0

Кодирование целочисленных значений

Беззнаковых

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

В дополнительном коде

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

знаковый бит

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

- **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
```

Weight	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

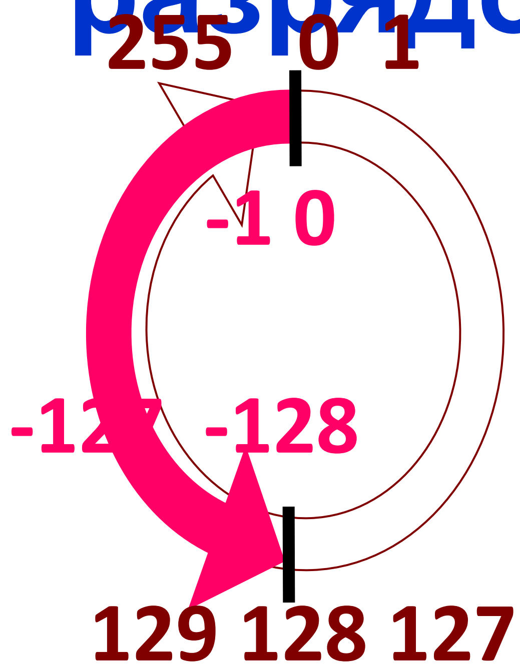
- **Важно!**
  - $|TMin| = TMax + 1$
    - Границы несимметричны
  - $UMax = 2 * TMax + 1$
- **Программирование на Си**
  - #include <limits.h>
  - Объявленные константы, e.g.,
    - ULONG_MAX
    - LONG_MAX
    - LONG_MIN
  - Значения констант зависят от платформы

Представление отрицательных чисел ограниченным числом разрядов

$255_{10} =$

$11111111_2 =$

-1_{10}



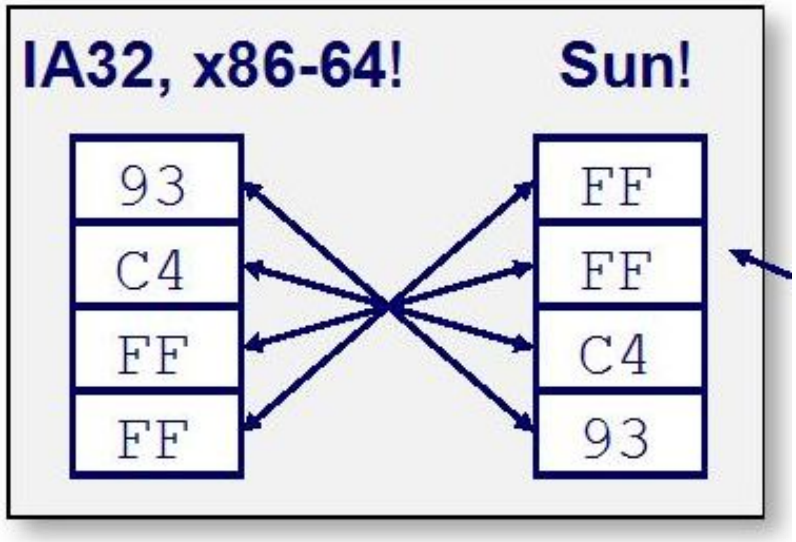
$127_{10} =$

01111111_2

$10000000_2 = +128_{10} = -?_{10}$

Хранение в памяти числа в дополнительном коде

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



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

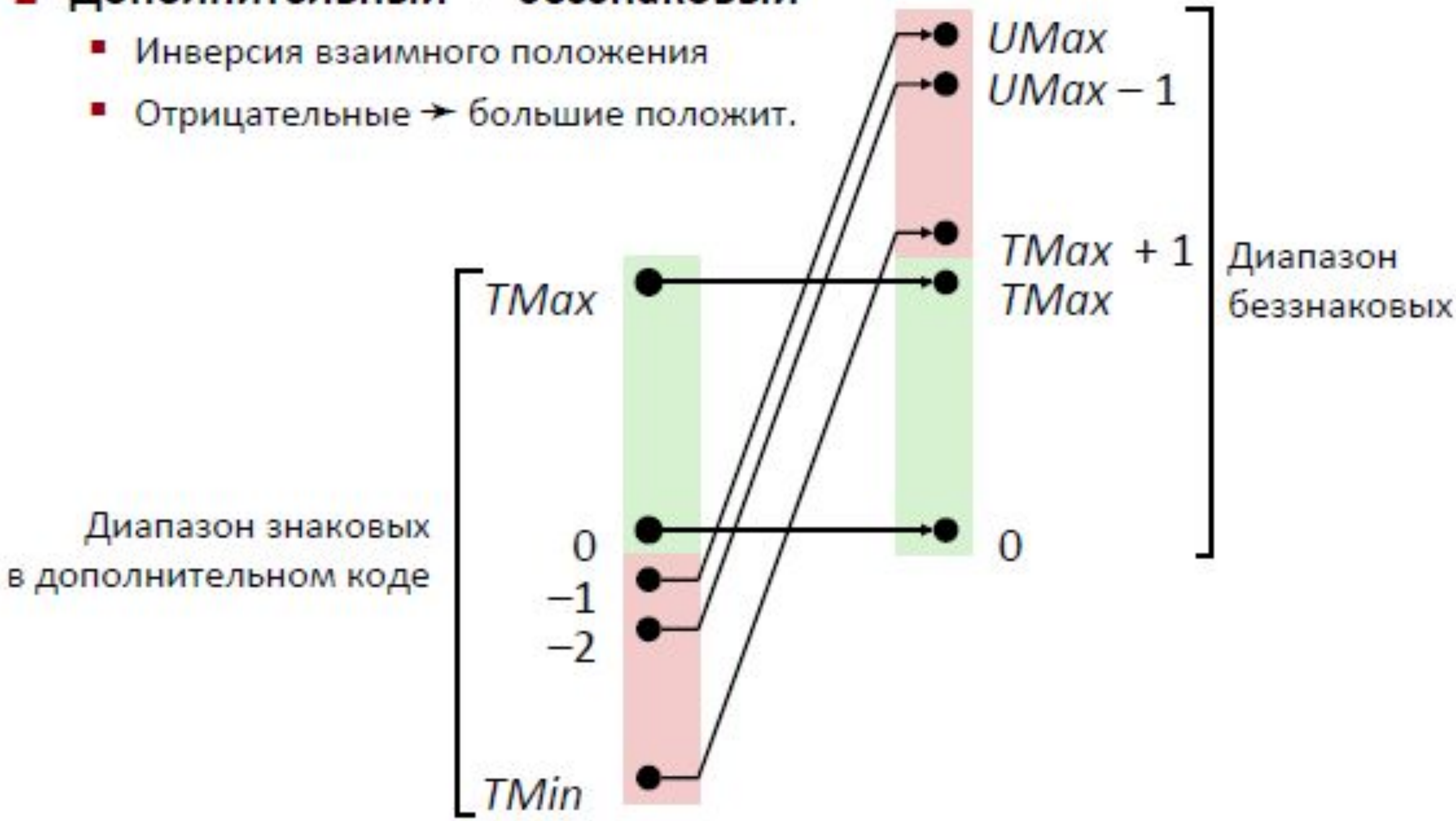
- **Совпадение**
  - Одинаковые кодировки неотрицательных величин
- **Взаимная однозначность**
  - Каждая комбинация бит представляет своё значение
  - Каждое представимое целое имеет уникальную кодировку

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	+/- 16	11
1100	–4		12
1101	–3		13
1110	–2		14
1111	–1		15

Визуализация преобразования

- **Дополнительный ➛ беззнаковый**
  - Инверсия взаимного положения
  - Отрицательные ➛ 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;
    ```

Пример

```
int x=-1;
unsigned u=2147483648;
printf(“x = %u = %d\n”,x,x);
printf (“u = %u = %d\n”,u,u);
```

Результат

x= 4294967295 = -1

u=2147483648 = -2147483648

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**

Выражение	**Тип**	**Оценка**
`0 == 0U`	Без знака	1
`- 1 < 0`	Со знаком	1
`- 1 < 0U`	Без знака	0*
`2147483648 > - 2147483647 - 1`	Со знаком	1
`2147483648U > - 2147483647 - 1`	Без знака	0*
`2147483647 > (int) 2147483648U`	Со знаком	1*
`- 1 > -2`	Со знаком	1
`(unsigned) - 1 > -2`	Без знака	1

Сводка: преобразование Знаковые ↔ Беззнаковые: правила

- Битовые последовательности неразличимы
- Интерпретируются по разному
- Возможная неожиданность: добавление или вычитание 2^w

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

Расщирение битового представления числа

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 переинтепретируется для новой длины
  - Unsigned: операция «остаток от деления»
  - Signed: операция похожая на «остаток от деления»
  - Ожидаемый результат – только для малых значений.

Пример

```
short sx = val;/* -12345 */
unsigned short usx = sx; /* 53191  */
int x = sx;/* -12345 */
unsigned ux = usx;/* 53191 */

printf ("sx = %d:\t", sx);
show_bytes((byte_pointer) &sx, sizeof(short));
printf ("usx = %u:\t", usx);
show_bytes((byte_pointer) &usx, sizeof(unsigned short));
printf ("x = %d:\t", x);
show_bytes((byte_pointer) &x, sizeof(int));
printf ("ux = %u:\t", ux);
show_bytes((byte_pointer) &ux, sizeof(unsigned));
```

```
sx = -12345:cf c7
usx = 53191:cf c7
x = -12345:ff ff cf c7
ux = 53191:00 00 cf c7
```

Добавление к примеру

```
unsigned uy = x;/* Мистика! */

printf ("uy = %u:\t", uy);
show_bytes ((byte_pointer) &uy, sizeof (unsigned));
```

Эта часть кода вызывает распечатку следующего:

```
uy = 4294954951:ff ff cf c7
```

Этим показывается, что выражения

```
(unsigned) (int) sx/* 4294954951 */
```

и

```
(unsigned) (unsigned short) sx/* 53191 */
```

Изменение знака: Инверсия и увеличение

- **Утверждение: для дополнительного кода верно**

  `~x + 1 == -x`

- **Инверсия**
  - Важно: `~x + x == 1111…111 == -1`

$$
\begin{array}{rr}
x & \boxed{1|0|0|1|1|1|0|1} \\
+\quad \sim x & \boxed{0|1|1|0|0|0|1|0} \\
\hline
-1 & \boxed{1|1|1|1|1|1|1|1}
\end{array}
$$

Примеры инверсии и увеличения

x = 15213

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

x = 0

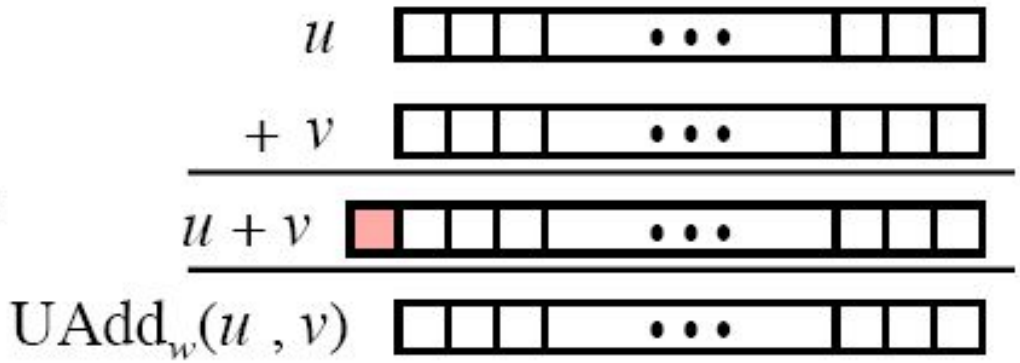
	Десятичное	Шестнадцатиричное	Двоичное
0	0	00 00	00000000 00000000
~0	-1	FF FF	11111111 11111111
~0+1	0	00 00	00000000 00000000

Беззнаковое сложение

Операнды: *w* бит

Полная сумма: *w*+1 бит

Без переноса: *w* бит



- **Стандартная операция сложения**
  - «Игнорирует» перенос
- **Реалилизует сложение по модулю**

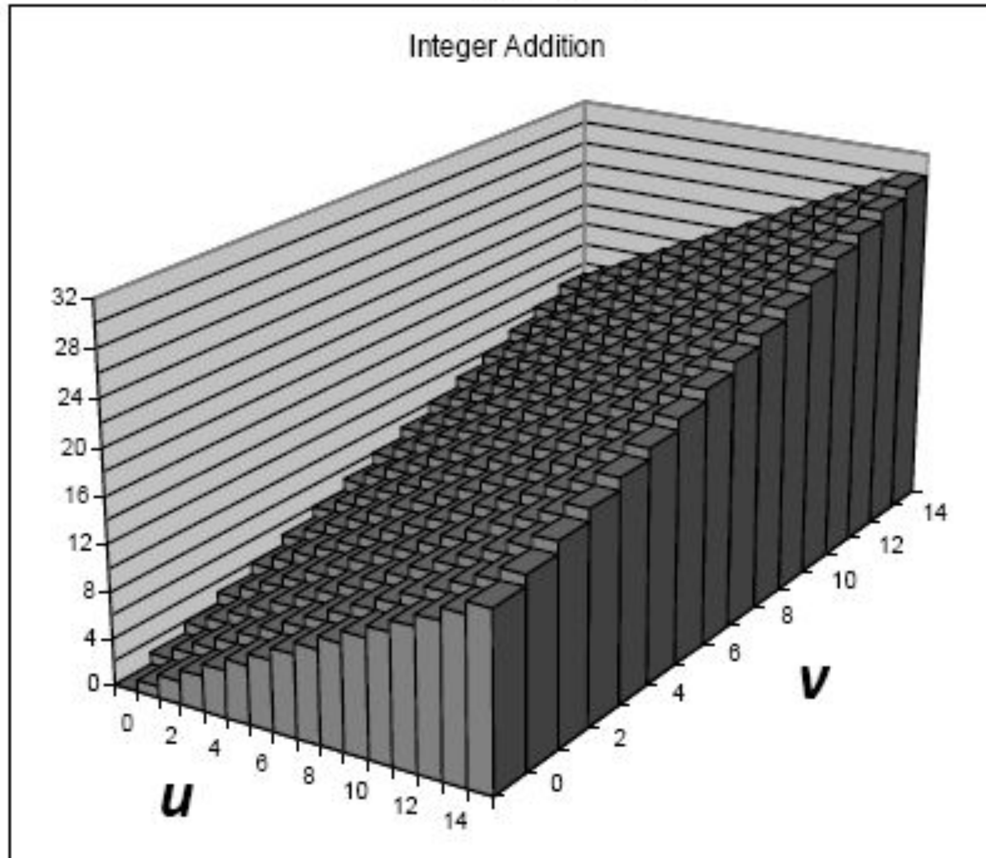
  $s \quad = \quad \mathrm{UAdd}_w(u\ ,v) \quad = \quad u+v \mod 2^w$

$$UAdd_w(u,v) \quad = \quad \begin{cases} u+v & u+v<2^w \\ u+v-2^w & u+v \geq 2^w \end{cases}$$

Математическое сложение визуально

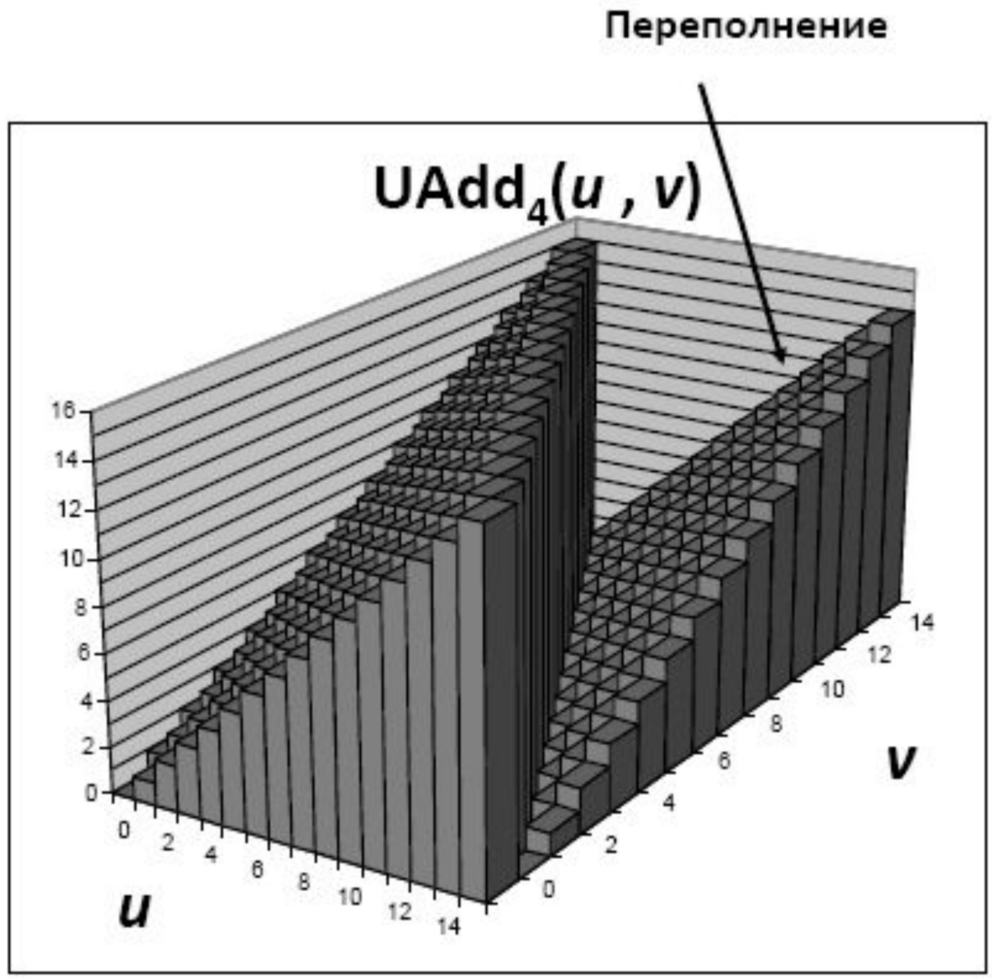
- **Сложение целых**
  - 4-х битовые целые *u*, *v*
  - Полная сумма $\text{Add}_4(u , v)$
  - Значение растёт линейно по *u* и *v*
  - Образует плоскость

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



Беззнаковое сложение визуально

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



Математические свойства

- **Сложение по модулю образует абелеву группу**
  - **Замкнута** относительно сложения

    $0 \le \mathrm{UAdd}_w(u, v) \le 2^w - 1$
  - **Коммутативна**

    $\mathrm{UAdd}_w(u, v) = \mathrm{UAdd}_w(v, u)$
  - **Ассоциативна**

    $\mathrm{UAdd}_w(t, \mathrm{UAdd}_w(u, v)) = \mathrm{UAdd}_w(\mathrm{UAdd}_w(t, u), v)$
  - **0** является нейтральным элементом

    $\mathrm{UAdd}_w(u, 0) = u$
  - Любому элементу есть **обратный**
    - Let $\mathrm{UComp}_w(u) = 2^w - u$

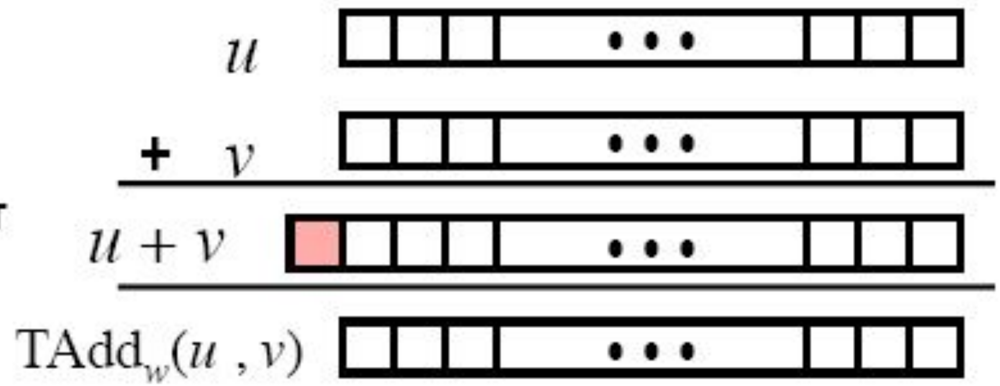
      $\mathrm{UAdd}_w(u, \mathrm{UComp}_w(u)) = 0$

Сложение в дополнительном коде

Операнды: *w* бит

Полная сумма: *w*+1 ,бит

Без преноса: *w* бит



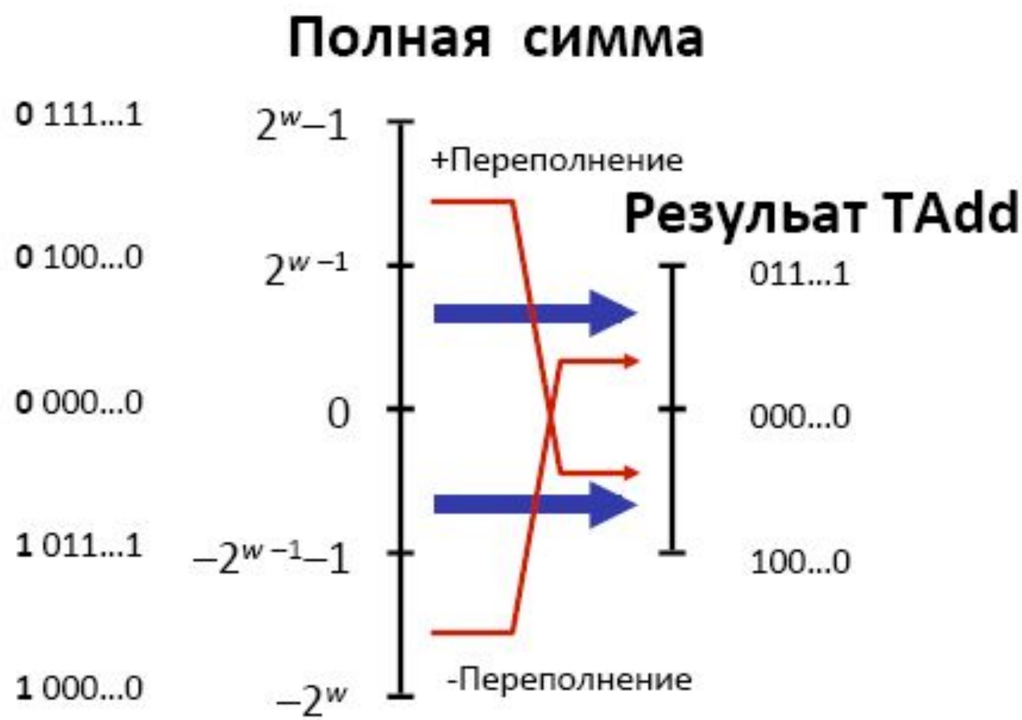
- **TAdd и UAdd преобразуют биты одинаково**
  - Знаковое и беззнаковое сложение в Си:

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

  - Всегда даст `s == t`

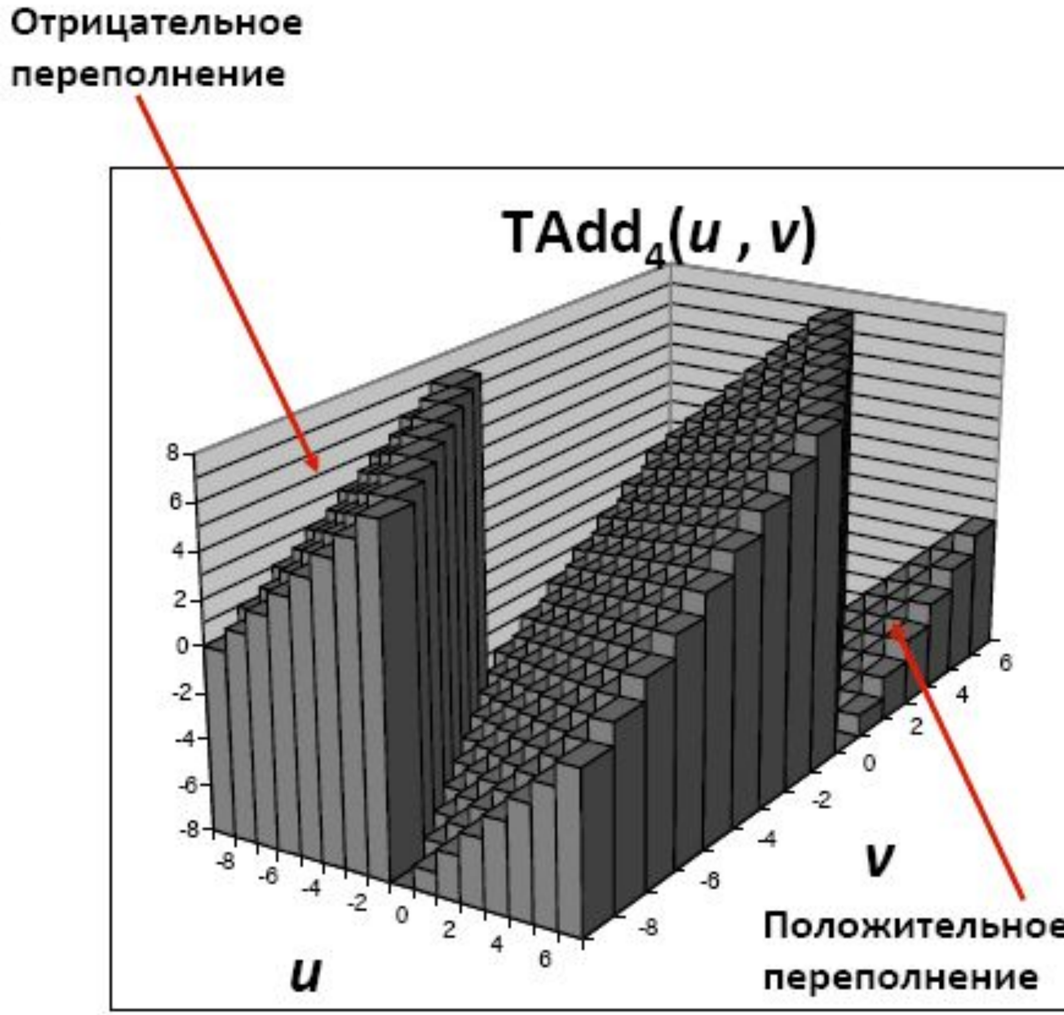
Переполнение TAdd

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



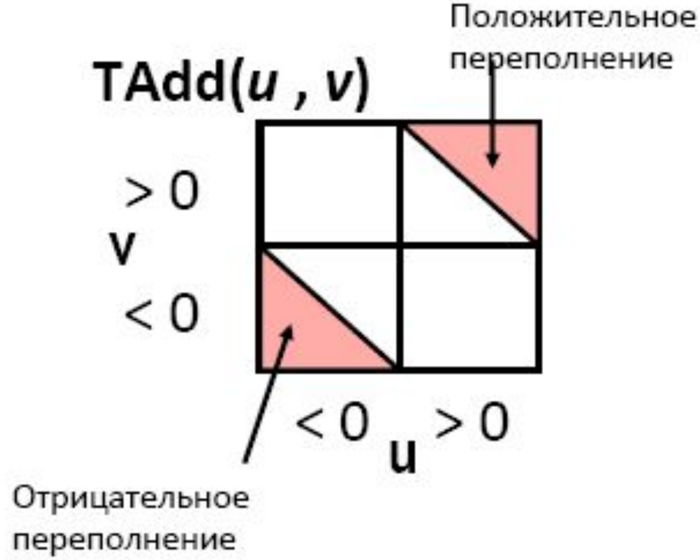
Сложение в доп. коде визуально

- **Значения**
  - 4-ре бита в доп. коде
  - Диапазон от -8 до +7
- **Отсечение**
  - Если сумма $\geq 2^{w-1}$
    - Результат отрицательный
    - Не более раза
  - Если сумма $< -2^{w-1}$
    - Результат положительный
    - Не более раза



Определение TAdd

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



$$TAdd_w(u,v) = \begin{cases} u+v+2^{w-1} & u+v < TMin_w \\ u+v & TMin_w \le u+v \le TMax_w \\ u+v-2^{w-1} & TMax_w < u+v \end{cases}$$

Отрицательное переполнение

Положительное переполнение

Математичские свойства TAdd

- **Группа изоморфная to беззнаковым с UAdd**
  - $TAdd_w(u, v) = U2T(UAdd_w(T2U(u), T2U(v)))$
    - т.к. битовые преобразования идентичны

- **Дополнительны коды с TAdd образуют группу**
  - Замкнута, коммутативна, ассоциатична, 0 – нейтральный элемент
  - Для каждого элемента есть обратный>

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