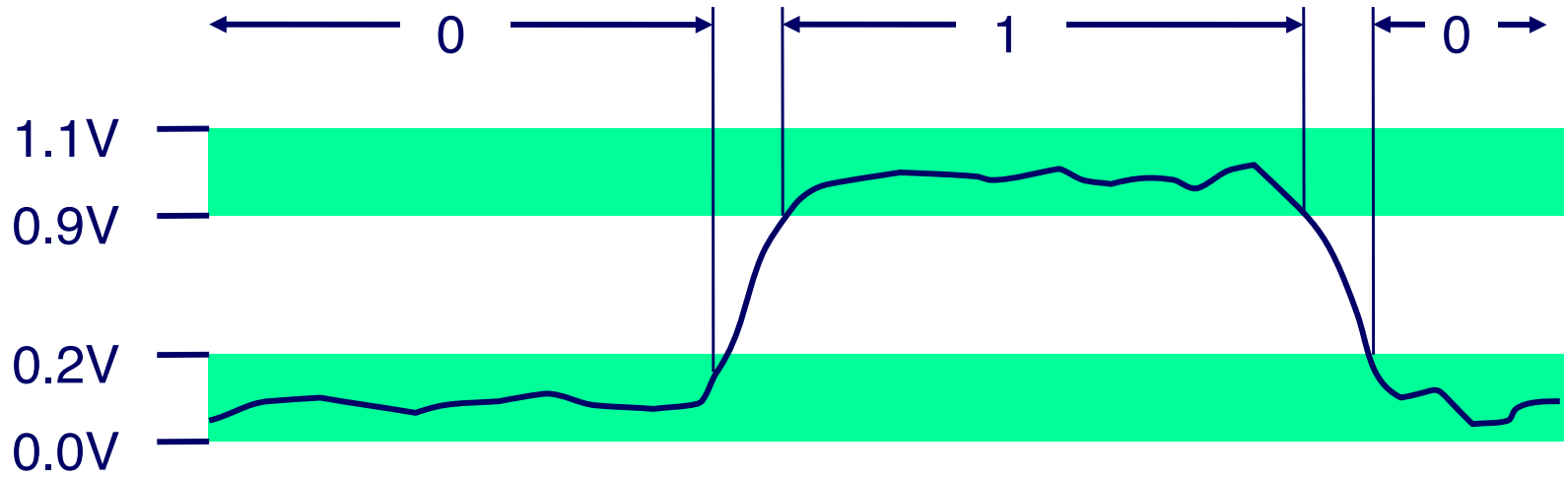
Биты, байты и целые

- **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} = $1.00110011001100110011[0011]\ldots_2$
  - 1.5213×10^4 = $1.11011011011011_2 \times 2^{13}$

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 $FA1D37B_{16}$ 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

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)

- **~A = 1 when 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

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**

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`
    - $01101001_2 \mid 01010101_2 \rightarrow 01111101_2$

Сравните: логические операции в C

- **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 C

- **Логические операторы**
  - &&, ||,
    - 0 кодируе
    - Всё, что не
  - Пр
    - !
    - !
    - !!
  - 0x69 && 0x55 → 0x01
  - 0x69 || 0x55 → 0x01
  - p && *p (способ избежать обращения по нулевому указателю)

Внимание!
&& вместо & (и || вместо |)…
одна из самых частых ошибок программирования на C

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	00010*000*
Логич. >> 2	*00*011000
Ариф. >> 2	*00*011000

Аргумент x	10100010
<< 3	00010*000*
Логич. >> 2	*00*101000
Ариф. >> 2	*11*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

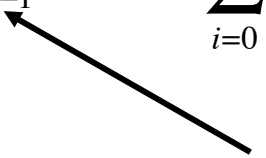
Кодирование целочисленных значений

Беззнаковых

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

В дополнительном коде

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



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
```

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**

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