

# Лекция 2

## Представление целых чисел

# Стандарты Си и Си++

- ISO/IEC 9899:2018 – последний стандарт Си (C17 или C18) – bugfix C11
- ISO/IEC 14882:2017 – последний стандарт Си++ (C++17), но де-факто C++20
- Общая модель выполнения (“виртуальная машина”) - байт-адресуемый двоичный процессор

# БИТЫ

- Двоичная система счисления – натуральна для элементной базы (манипулирование напряжением для передачи сигнала, транзисторные ключи и т. д.)
- Попытки использовать симметричную троичную систему  $\{-1, 0, 1\}$  - ЭВМ “Сетунь”
- Некоторые теоретические преимущества не оправдываются усложнением схемотехники

# Byte vs Octet

- Почти всегда говоря “байт” мы подразумеваем “8 бит” - строго говоря, это не так
- С точки зрения стандартов C/C++ `byte == char` – минимальная адресуемая единица памяти
- `sizeof(char) == 1`
- Количество бит в `char` определяется константой `CHAR_BIT`
- Стандарт требует `CHAR_BIT >= 8`
- Если память можно адресовать только по 16-битным словам, то `CHAR_BIT == 16`
- Октет – всегда 8 бит

# Integral types

- `sizeof(signed T) == sizeof(unsigned T) == sizeof(T)`
- `short` как минимум 16 бит
- `int` как минимум 16 бит
- `long` как минимум 32 бита
- `long long` как минимум 64 бита
- `<stdint.h>`: `intN_t`, `uintN_t`
- `<inttypes.h>`: I/O макросы

# Byte (unsigned char)

- IEEE Std 1003.1-2017 – The Open Group Base Specifications – стандарт на API Unix-like
- Требуется `CHAR_BIT == 8`
- Далее будем предполагать что `CHAR_BIT==8`
- Беззнаковые целые числа представляют значения  $[0; 2^N - 1]$ ,  $N$  – число бит
- `Unsigned char` позволяет представлять значения  $[0; 255]$

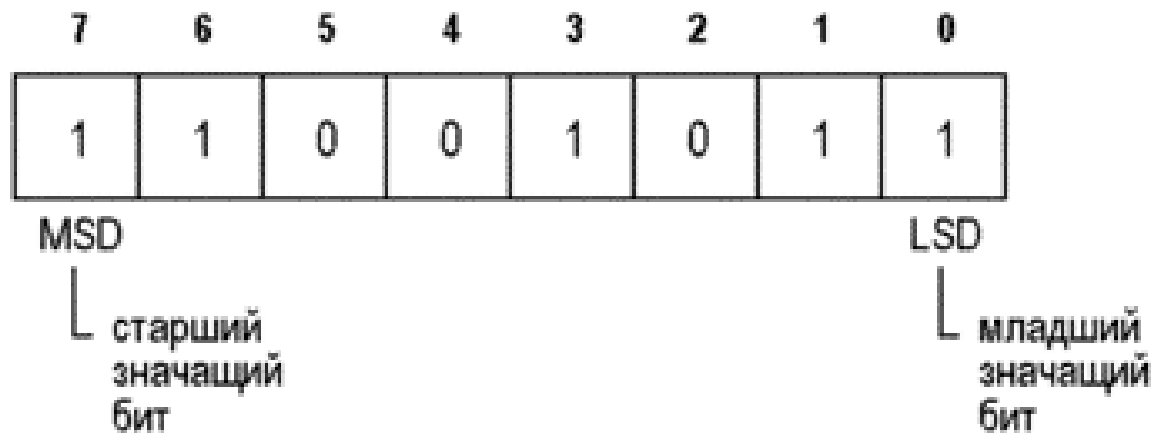
# Нумерация битов

- Индивидуальные биты в байте не адресуются, но для удобства мы их нумеруем
- 0 – младший (самый правый в двоичной записи) бит
- 7 – старший (самый левый в двоичной записи) бит

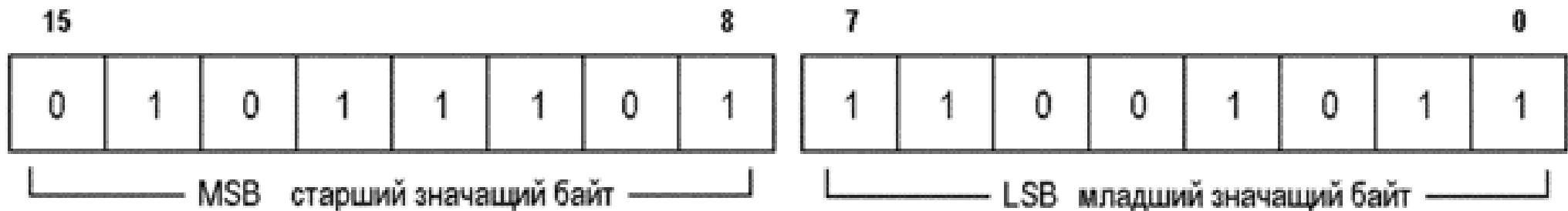
# Нумерация битов

- 

## Байт (8 бит)



## Слово (16 бит)





# Битовые операции

- $\sim$ ,  $\&$ ,  $|$ ,  $\wedge$  - применимы к целым числам
- Каждый бит операндов рассматривается независимо

And

■  $A \& B = 1$  when both  $A=1$  and  $B=1$

| $\&$ | 0 | 1 |
|------|---|---|
| 0    | 0 | 0 |
| 1    | 0 | 1 |

Not

■  $\sim A = 1$  when  $A=0$

| $\sim$ |   |
|--------|---|
| 0      | 1 |
| 1      | 0 |

Or

■  $A | B = 1$  when either  $A=1$  or  $B=1$

| $ $ | 0 | 1 |
|-----|---|---|
| 0   | 0 | 1 |
| 1   | 1 | 1 |

Exclusive-Or (Xor)

■  $A \wedge B = 1$  when either  $A=1$  or  $B=1$ , but not both

| $\wedge$ | 0 | 1 |
|----------|---|---|
| 0        | 0 | 1 |
| 1        | 1 | 0 |

# Битовые вектора для множеств

- В 8-битном числе храним множество не более чем из 8 эл-тов  $\{0,1,2,3,4,5,6,7\}$ 
  - 01101001     $\{0, 3, 5, 6\}$
  - 76543210
  - 01010101     $\{0, 2, 4, 6\}$
  - 76543210
- Операции
  - & - Пересечение – 01000001 -  $\{0, 6\}$
  - | - Объединение – 01111101 -  $\{0, 2, 3, 4, 5, 6\}$
  - ^ - Симметрическая разность – 00111100 -  $\{2, 3, 4, 5\}$
  - ~ - Дополнение (ко второму множеству) – 10101010 -  $\{1, 3, 5, 7\}$

# Сдвиги (беззнаковые)

- Сдвиг влево:  $x \ll n$ 
  - Пример (для 8 битных беззнаковых чисел)
  - $01101100 \ll 3 == \text{011}01100000$
  - Эквивалентно умножению на  $2^n$
- Сдвиг вправо:  $x \gg n$ 
  - $01101100 \gg 4 == 00000110\text{(1100)}$
  - Эквивалентно целой части от деления на  $2^n$

# Знаковые числа

- Необходимо кодировать знак числа в N бит представления числа
- Стандарты Си/Си++ допускают три типа представления знаковых чисел
  - Прямой код (sign-magnitude)
  - Обратный код (one's complement)
  - Дополнительный код (two's complement)

# Прямой код

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

# Обратный код

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

# Дополнительный код

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

# Знаковые числа

- Старший разряд – знаковый
  - 0 – положительные числа
  - 1 – отрицательные числа
- Практически универсально используется дополнительный код
- UNISYS 2200 – обратный код
- Прямой код – мантисса представления IEEE-754 вещественных чисел



# Дополнительный код

- $-x = \sim x + 1$
- Несимметричный диапазон, например, для signed char - [-128; 127]
- Рассмотрим далее 4-битные числа: [-8; 7]
- Ограниченная разрядность – сохраняются только младшие биты результата соотв. разрядности (например, 4), старшие биты – теряются
- Примеры в двоичной системе:
  - $-0000 = \sim 0000 + 1 = 1111 + 1 = \pm 0000 \rightarrow 0$
  - $-1000 = \sim 1000 + 1 = 0111 + 1 = 1000$
  - $0110 + (-0110) = 0110 + (\sim 0110 + 1) = 0110 + (1001 + 1) = (0110 + 1001 + 1) = 1111 + 1 = \pm 0000 \rightarrow 0$

# Удобство дополнительного кода

- Операции сложения и вычитания n-битных беззнаковых чисел дают правильный результат для знаковых чисел
- В процессоре не нужно различать знаковые и беззнаковые числа при сложении и вычитании – меньше инструкций, проще аппаратура
- Для 4-битных чисел (0b – запись двоичного числа в Си):
  - $5 + 9 = 0b0101 + 0b1001 = 0b1110 = 14$
  - $5 + (-7) = 0b0101 + 0b1001 = 0b1110 = -2$
  - $12 + 13 = 0b1100 + 0b1101 = 0b\pm1001 = 9$  – беззнаковое переполнение
  - $-4 + (-3) = 0b1100 + 0b1101 = 0b\pm1001 = -7$  – со знаковыми ОК

# Типы char в Си/Си++

- `sizeof(char) == sizeof(signed char) == sizeof(unsigned char) == 1`
- Char может быть либо знаковым, либо беззнаковым (implementation defined behavior)
- На x86/x64 char – знаковый, на ARM char – беззнаковый
- Но формально char и signed char (или char и unsigned char) – разные типы
- GCC позволяет переключаться между режимами: -fsigned-char -funsigned-char
- Программа должна быть корректной в любом случае

# Byte order

- Память адресуется побайтно
- Целые числа большей длины могут размещаться в памяти по-разному
- Преобразование прозначно для программиста
- Little-endian: x86
- Big-endian: SPARC
- Переключаемые: ARM, PPC (Android — LE, iOS — LE)
- Подавляющее большинство компьютерных систем в настоящее время работает в режиме LE

# Byte order

|                               | Low address |        |        |                            | High address |        |        |        |
|-------------------------------|-------------|--------|--------|----------------------------|--------------|--------|--------|--------|
| Address                       | 0           | 1      | 2      | 3                          | 4            | 5      | 6      | 7      |
| Little-endian                 | Byte 0      | Byte 1 | Byte 2 | Byte 3                     | Byte 4       | Byte 5 | Byte 6 | Byte 7 |
| Big-endian                    | Byte 7      | Byte 6 | Byte 5 | Byte 4                     | Byte 3       | Byte 2 | Byte 1 | Byte 0 |
| Memory content                | 0x11        | 0x22   | 0x33   | 0x44                       | 0x55         | 0x66   | 0x77   | 0x88   |
| 64 bit value on Little-endian |             |        |        | 64 bit value on Big-endian |              |        |        |        |
| 0x8877665544332211            |             |        |        | 0x1122334455667788         |              |        |        |        |

# Типы данных в Си/Си++

- `sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)`
- `sizeof(float) <= sizeof(double) <= sizeof(long double)`
- `sizeof(signed T) == sizeof(unsigned T)`
- Преобразование знаковое ↔ беззнаковое сохраняет битовое представление
- Преобразование более “широкого” значения в более “узкое” отсекает старшие биты

# Расширение типа

- Беззнаковое расширение: unsigned char → unsigned short – дополнение нулями
  - 0b11100011 → 0b00000000 11100011
- Знаковое расширение: signed char → signed short
  - 0b01110011 → 0b00000000 01110011
  - 0b11100011 → 0b11111111 11100011

# Sizeof для типов

| Тип                | Atmel AVR | 32-bit   | Win64 | 64-bit |
|--------------------|-----------|----------|-------|--------|
| <b>char</b>        | 1         | 1        | 1     | 1      |
| <b>short</b>       | 2         | 2        | 2     | 2      |
| <b>int</b>         | 2         | 4        | 4     | 4      |
| <b>long</b>        | 4         | 4        | 4     | 8      |
| <b>long long</b>   | -         | 8        | 8     | 8      |
| <b>__int128 !</b>  | -         | -        | -     | 16     |
| <b>float</b>       | 4         | 4        | 4     | 4      |
| <b>double</b>      | 4         | 8        | 8     | 8      |
| <b>long double</b> | 4         | 8 или 12 | 8     | 16     |
| <b>void *</b>      | 2         | 4        | 8     | 8      |



# Максимальные/минимальные значения типов

- C++
  - `#include <limits>`
  - `std::numeric_limits<T>::max()`
  - `std::numeric_limits<T>::min()`
- C
  - `#include <limits.h>`
  - `CHAR_MIN, CHAR_MAX, SCHAR_MIN, SCHAR_MAX, UCHAR_MIN, ..., INT_MIN, INT_MAX, UINT_MAX, LONG_MIN, LONG_MAX, ULONG_MAX, LLONG_MIN, LLONG_MAX, ULLONG_MAX`

# Типы фиксированной битности

- Заголовочный файл `<stdint.h>`
  - Знаковые типы: `int8_t`, `int16_t`, `int32_t`, `int64_t`
  - Беззнаковые типы: `uint8_t`, `uint16_t`, `uint32_t`, `uint64_t`
  - Типы размера, достаточного для хранения указателя: `intptr_t`, `uintptr_t`
- Дополнительно в `<inttypes.h>`:
  - Макросы для `printf` и `scanf` для использования этих типов: `PRId32`, ...
  - `printf("%" PRId64 "\n", t);`

# Переполнение беззнаковых чисел

- Все операции над беззнаковыми числами выполняются по модулю  $2^N$  ( $N$  – битность)
- Поведение строго определено стандартом
- $UINT\_MAX + 1 == 0$
- $0 - 1U == UINT\_MAX$

# Implementation-defined behavior

- Каждая реализация компилятора Си/Си++ должна реализовывать implementation-defined behavior разумным образом (одним из нескольких predetermined вариантов) и документировать это поведение
- Пример: представление отрицательных чисел

# Unspecified behavior

- Каждая реализация компилятора Си/Си++ может реализовывать unspecified behavior по-разному, даже в пределах одной программы, не обязана документировать. Unspecified behavior – это корректное (но недетерминированное) поведение корректной программы
- Пример: порядок вычисления аргументов при вызове функции

# Undefined behavior

- Если программа выполнила операцию, описанную как undefined behavior, дальнейшее поведение программы **не определено** – это ошибочная программа
- Варианты поведения: ignore, crash, burn computer
- Примеры: разыменовывание нулевого указателя, выход за пределы массива

# Undefined behavior

- Компилятор Си/Си++ вправе предполагать, что при выполнении программы **никогда не произойдет** undefined behavior и использовать это при оптимизации

```
*p = 'a';  
if (!p) {  
— fprintf(stderr, “NULL pointer\n”); return;  
— }
```

- if можно удалить, исходя из предположения выше

# Операции со знаковыми значениями

- Преобразование широкого типа к знаковому узкому, при условии, что значение непредставимо узким знаковым типом – *implementation defined* (обычно *truncation*)
  - Пример: `(char) 384 == -128`
- Сдвиг отрицательного числа вправо – *implementation defined* (обычно знаковый бит остается на месте – арифметический сдвиг)



# Операции со знаковыми значениями

- Переполнение при выполнении знаковых операций  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$  - undefined behavior
  - Для компилятора  $N < N + 1$  выполняется всегда!
- Сдвиг на число бит, большее размера типа – undefined behavior
- Сдвиг отрицательного числа влево – undefined behavior
- Сдвиг на отрицательную величину – undefined behavior

# Опции gcc

- `-ftrapv` – `abort()` при знаковом целочисленном переполнении
- `-fwrapv` – выполнять знаковые операции по модулю  $2^N$ , но компилятор не делает никаких оптимизационных предположений
- `-fsanitize=undefined` – проверка на undefined behavior при работе программы – обсудим в следующий раз