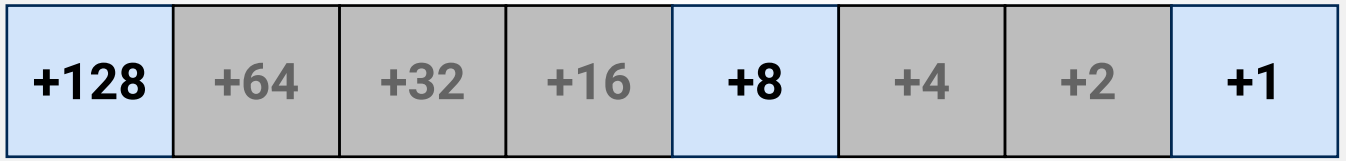
Представление данных в компьютере

АКОС, МФТИ

19 сентября, 2024

Как работают целые числа

$$137_{10} =$$

$$=$$ 

$$=$$

$$= 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1_2$$

Целочисленные типы в C

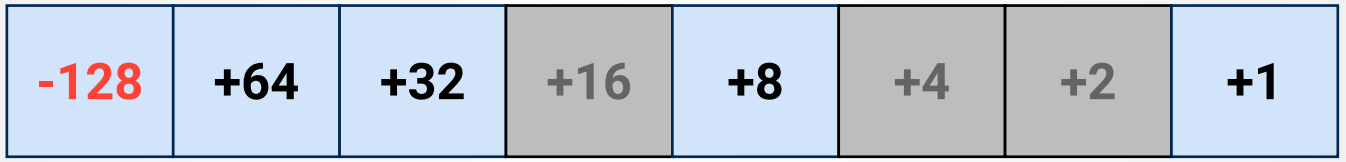
- `char` : **1 байт** (или `CHAR_BIT` бит) данных;
- `short` и `int` : не менее **16 бит** данных;
- `long` : не менее **32 бит** данных;
- `long long` : не менее **64 бит** данных.

Типы фиксированной длины (`#include <stdint.h>`):

- `int8_t` и `uint8_t` : строго **8 бит**;
- `int16_t` и `uint16_t` : строго **16 бит**;
- `int32_t` и `uint32_t` : строго **32 бита**;
- `int64_t` и `uint64_t` : строго **64 бита**.

Как работают знаковые числа

$$-23_{10} =$$

$$=$$ 

$$=$$

$$= 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1_2$$

⚠️ : Любое отрицательное число начинается с `1` и наоборот;

⚠️ : Конвертация знаковых типов друг к другу становится менее тривиальным.

Знаковые и беззнаковые типы в C

- `char` : **не определено стандартом**.
- `short` , `int` , `long` и `long long` : по умолчанию **знаковые**;
- Любой из типов выше можно сделать:
  - **Знаковым** (напр., `signed char`);
  - **Беззнаковым** (напр., `unsigned int`).

Знаковые и беззнаковые типы в C

- `char` : **не определено стандартом**.
- `short` , `int` , `long` и `long long` : по умолчанию **знаковые**;
- Любой из типов выше можно сделать:
  - **Знаковым** (напр., `signed char`);
  - **Беззнаковым** (напр., `unsigned int`).

Типы фиксированной длины (`#include <stdint.h>`):

- `int16_t` : **знаковое** 16-битное число;
- `uint16_t` : **беззнаковое** 16-битное число (`u` в начале от слова `unsigned`);

Знаковые и беззнаковые типы в C

- `char` : **не определено стандартом**.
- `short` , `int` , `long` и `long long` : по умолчанию **знаковые**;
- Любой из типов выше можно сделать:
  - **Знаковым** (напр., `signed char`);
  - **Беззнаковым** (напр., `unsigned int`).

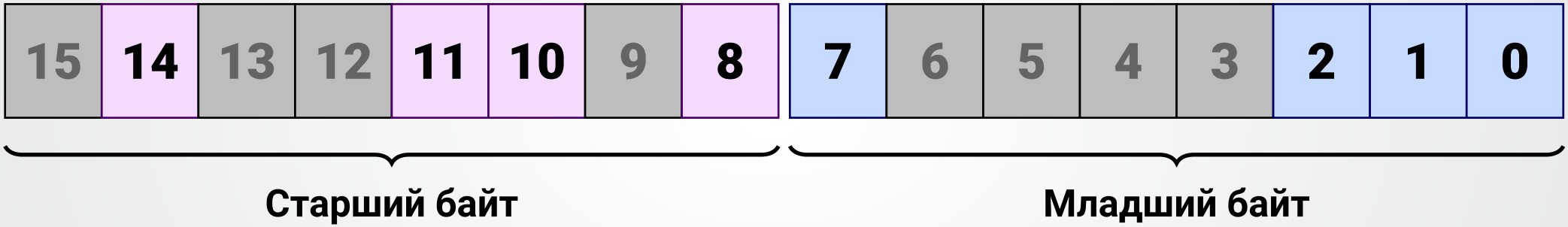
Типы фиксированной длины (`#include <stdint.h>`):

- `int16_t` : **знаковое** 16-битное число;
- `uint16_t` : **беззнаковое** 16-битное число (`u` в начале от слова `unsigned`);

⚠️ : Знаковые типы **нельзя переполнять** - в Си это UB. Беззнаковые – можно.

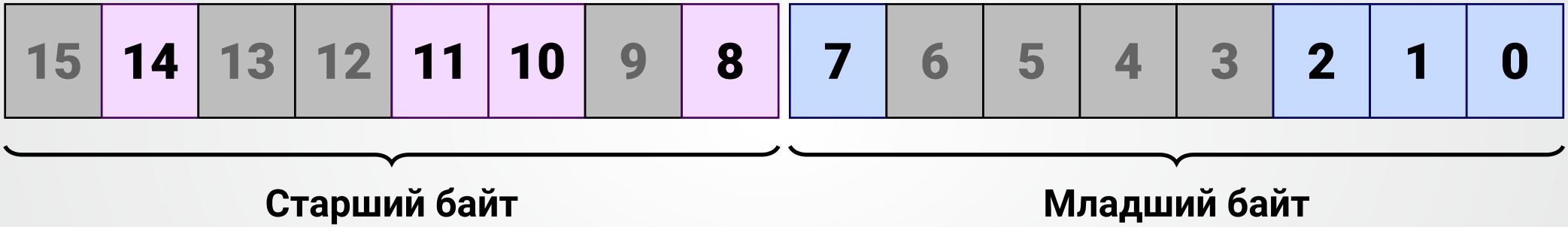
Как хранить длинные типы?

```
(uint16_t) 19847 =
```



Как хранить длинные типы?

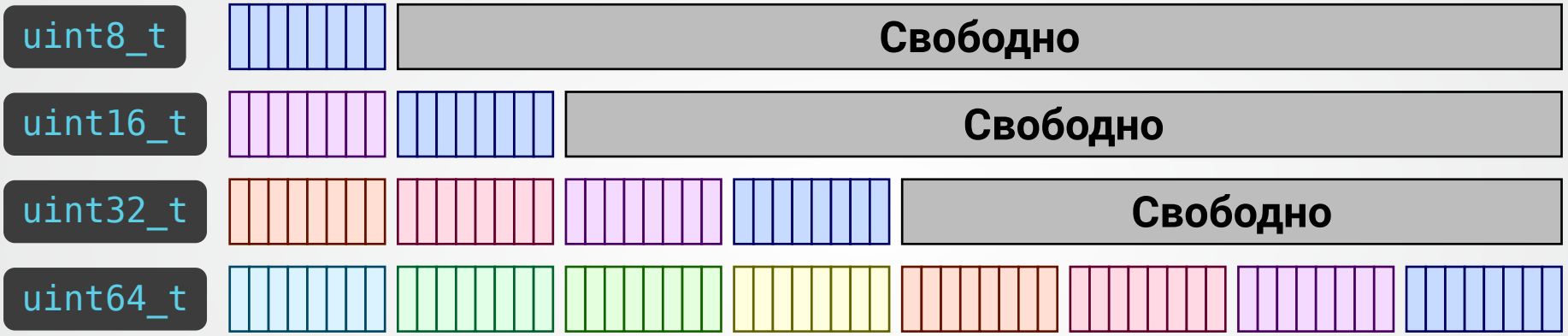
```
(uint16_t) 19847 =
```



Но не будет ли проблем?...

Что может пойти не так?

```
1    int main() {
2      uint64_t my_long = 42;
3
4      printf("%d\n", &my_long); // Что выведет?
5    }
```



⚠️ : **Младший синий байт оказывается в разных местах.**

Сложности приведения типов

```
1    uint8_t a_byte = 42;
2
3    uint16_t a_16b = a_byte; // Перенесёт 42 во второй байт
4    uint32_t a_32b = a_byte; // Перенесёт 42 в четвёртый байт
5    uint64_t a_64b = a_byte; // Перенесет 42 в восьмой байт
```

Сложности приведения типов

```
1    uint8_t a_byte = 42;
2
3    uint16_t a_16b = a_byte; // Перенесёт 42 во второй байт
4    uint32_t a_32b = a_byte; // Перенесёт 42 в четвёртый байт
5    uint64_t a_64b = a_byte; // Перенесет 42 в восьмой байт
```

: В схеме big-endian **каждый целочисленный каст** требует **перемещение байт**:

- **Либо на стороне компилятора** (усложнением генерируемого кода)
- **Либо на стороне процессора** (тратой лишних транзисторов)

Сложности приведения типов

```
1    uint8_t a_byte = 42;
2
3    uint16_t a_16b = a_byte; // Перенесёт 42 во второй байт
4    uint32_t a_32b = a_byte; // Перенесёт 42 в четвёртый байт
5    uint64_t a_64b = a_byte; // Перенесет 42 в восьмой байт
```

: В схеме big-endian **каждый целочисленный каст** требует **перемещение байт**:

- **Либо на стороне компилятора** (усложнением генерируемого кода)
- **Либо на стороне процессора** (тратой лишних транзисторов)
- А если число еще и знаковое…

Часто ли приходится приводить типы?

```
char c = fgetc(file);
```

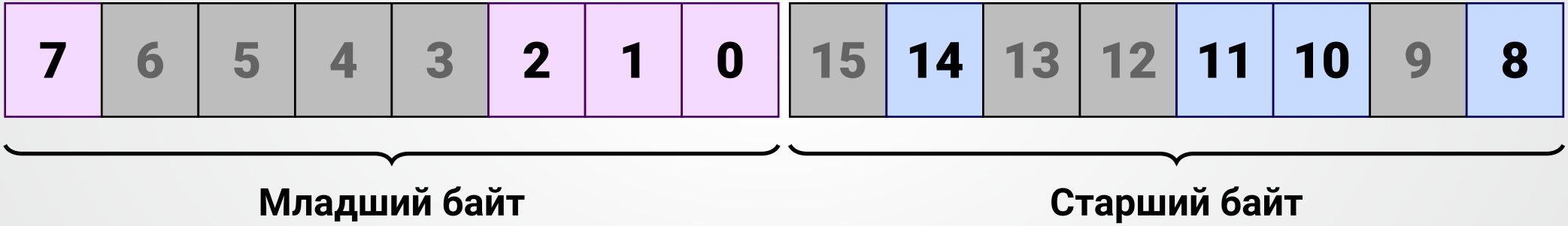
Часто ли приходится приводить типы?

```
char c = (char)fgetc(file);
```

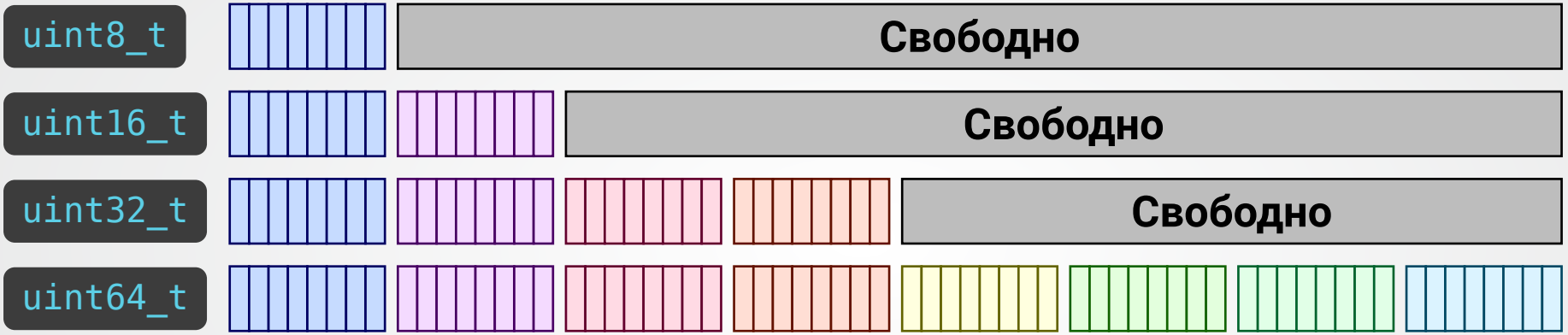
Да, достаточно часто

Что, если хранить байты наоборот?

```
(uint16_t) 19847 =
```



Что, если хранить байты наоборот?



✓ В такой схеме (little-endian) приведение целочисленных типов не требует перемещения байт.