Little-Endian and Big-Endian

Pilote Muhoza

Little-Endian and Big-Endian Explained

In computer architecture, little-endian and big-endian refer to the byte order used to store multibyte data types, such as integers, in memory.

Little-Endian

In little-endian systems:

- The least significant byte (LSB) is stored at the lowest memory address.
- The most significant byte (MSB) is stored at the highest memory address.

Big-Endian

In big-endian systems:

- The most significant byte (MSB) is stored at the lowest memory address.
- The least significant byte (LSB) is stored at the highest memory address.

Example

Given the following declaration: uint8_t a[] = {0x01, 0x10, 0x03};

```
uint8_t a[] = {0x01, 0x10, 0x03};
```

- The QUTy is a little-endian system (like x86), the bytes will be interpreted in reverse order:

In little-endian systems, the least significant byte (the byte representing the smallest value) is stored at the lowest memory address, and the most significant byte (the byte representing the largest value) is stored at the highest memory address.

Let's take the example array `a` in your code: `{0x01, 0x10, 0x03}`. In little-endian representation, this would be stored in memory as `0x01 0x10 0x03`. When interpreting this sequence of bytes as a 16-bit unsigned integer (`uint16_t`), the bytes are read in reverse order, resulting in **`0x1001`**.

So, in little-endian systems, the least significant byte is at the lower memory address, and when you interpret the array as a 16-bit integer, the bytes are read in reverse order. This is why the value of `*(uint16_t *)&a` is `0x1001` in a little-endian system.