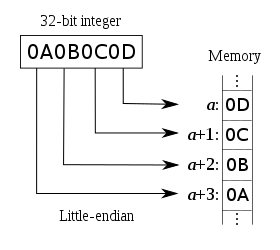
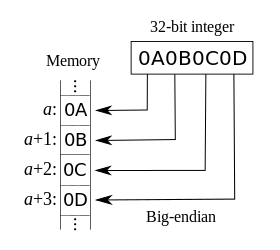
在计算机内存中，通常是以字节（Byte），也就是 8 个位（Bit）为基本存储单元（也有以 16 位为基本存储单元的）。对于跨越多个字节的数据类型（比如 int 长 4 个字节），如何在内存中对这些字节进行排序有两种常见的方法：大端法（Big-endian）和小端法（Little-endian）。

       【注】不管是大端法还是小端法存储，计算机在内存中存放数据的顺序都是从低地址到高地址，所不同的是首先取低字节的数据存放在低地址还是取高字节数据存放在低地址。

* 若首先取高字节的数据存放在低地址，则是大端法；
* 若首先取低字节的数据存放在低地址，则是小端法。

字节序还是位序  
大端法和小端法指的是字节在内存中存储时的排列规则，而不是数据中的位的排列规则。也有以位序排列的机器，但很少见。另外，再次明确一下，大端法或小端法是数据在存储时的表现，而不是在寄存器中参与运算时的表现。  
  
浮点数的字节序  
在所有机器上，浮点数在存储时的字节顺序是和整数的字节顺序一样的，所以在进行网络传输时，可以把浮点数当作整数进行字节序转换。但在历史上，曾经有段时间因为 IEEE 并没有规定浮点数在网络上传送的标准，所以浮点数都是以大端法进行存储的。

例如0x0A0B0C0CD的在大端和小端中的存放方式分别为：

小端法在变换数据指针类型的时候，尤其是从长字节指针向短字节指针强制转换的时候，不容易改变变量的值。

    #include<stdio.h>//查看计算机是什么储存方式

    int mem\_cheak()

    {

     int i = 1;

     char \*ptr = (char \*)(&i);

     if (\*ptr == 1)

     {

     return 1;

     }

     else if (\*ptr == 0)

     {

     return 0;

     }

    }

    int main()

    {

     int ret = mem\_cheak();

     if (ret == 1)

     {

     printf("little\n");

     }

     else

     {

     printf("big\n");

     }

     system("pause");

     return 0;

    }