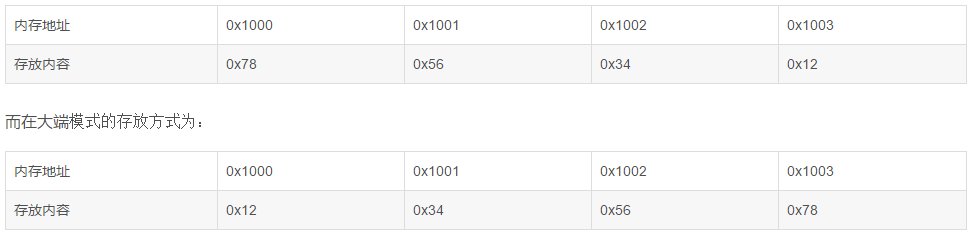
大端模式与小端模式

# 概述：小端与大端模式

**小端模式**：是指数据的高字节保存在内存的高地址中，而数据的低字节保存在内存的**低地址**；

**大端模式**：是指数据的高字节保存在内存的低地址中，而数据的低字节保存在内存的**高地址**；

不同的系统在存储数据时是分**大端（bit-endian）小端（little-endian）**存储的，比如，Inter x86、ARM核采用的是**小端模式**，Power PC、MIPS UNIX和HP-PA UNIX采用**大端模式**。都是以字节为单位的。

大小端CPU字节序：32位宽的十六进制数0x12345678在小端模式的存放方式（假设从地址0x1000开始存放）为：  


一般操作系统都是小端，而**通信协议**是大端。

属于小端的机器: x86 、 DEC

属于大端的机器：PowerPC、IBM、Sun

ARM既可以工作在大端模式，也可以工作在小端模式。

<https://blog.csdn.net/jinnian_123/article/details/51335155>

# 小端与大端模式的判断

两种方法：指针类型转换与共用体。

## 指针类型转换

将int型地址转换成char型地址，再获取值判断是**高字节或低字节**即可。

/\*

\* 1: little-endian

\* 0: big-endian

\*/

int checkEndian(){

int a = 1;

char \*p = (char \*)&a;

return **(\*p == 1);**

}

## 共用体判断

/\*

\* 1: little-endian

\* 0: big-endian

\*/

int checkEndian(){

**union w**

**{**

**int a;**

**char b;**

**} c;**

**c.a = 1;**

return **(c.b == 1)**;

}

注：联合体union的存放顺序是所有成员都从低地址开始存放；

printf("%s\n", checkEndian() ? "little-endian" : "big-endian");

# 小端模式与大端模式的转换

int big\_litle\_endian(int x) {

int tmp;

tmp = (**((x)&0xff)**<<24) + (**((x>>8)&0xff)**<<16) + (**((x>>16)&0xff)**<<8) + (**((x>>24)&0xff)**);

return tmp;

}