

# 数据存储格式





上海交通大学  
SHANGHAI JIAO TONG UNIVERSITY



# 数据存储字节顺序

## 数据存储字节顺序

- 大数端(Big Endian)
  - 最低字节存储在最高地址
- 小数端(Little Endian)
  - 最低字节存储在最低地址
- 例:数据0x000F4240:

4	5	6	7
00	0F	42	40

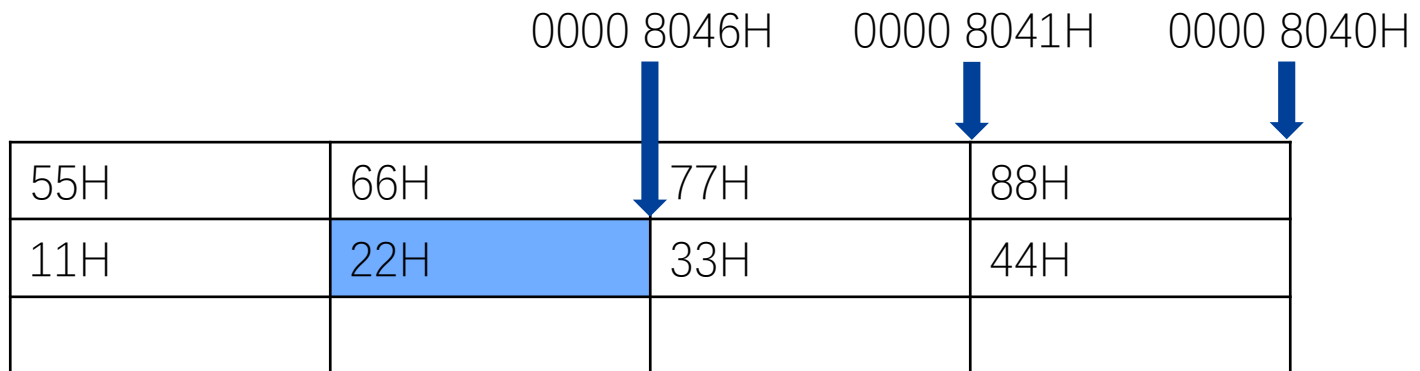
(a) 大数端存储方式

4	5	6	7
40	42	0F	00

(b) 小数端存储方式



- 举例：某计算机字长为32位，按字节编址，采用小端（little Endian）方式存放数据，假定有一个double型变量，其机器数表示为1122 3344 5566 7788H，存放在0000 8040H开始连续存储单元中，其存储单元 0000 8046H 中存放的是 22H 。





# 数据类型及其存储方式

## ■ 数据对齐方式(Alignment)

地址	3	2	1	0
0				
4				
8				
12				
16				
...	...	...	...	...

(a) 字不对齐

字节	3	2	1	0
0				
4				
8				
12				
16				
...	...	...	...	...

(b) 字对齐



# 不对齐方式下的数据存储

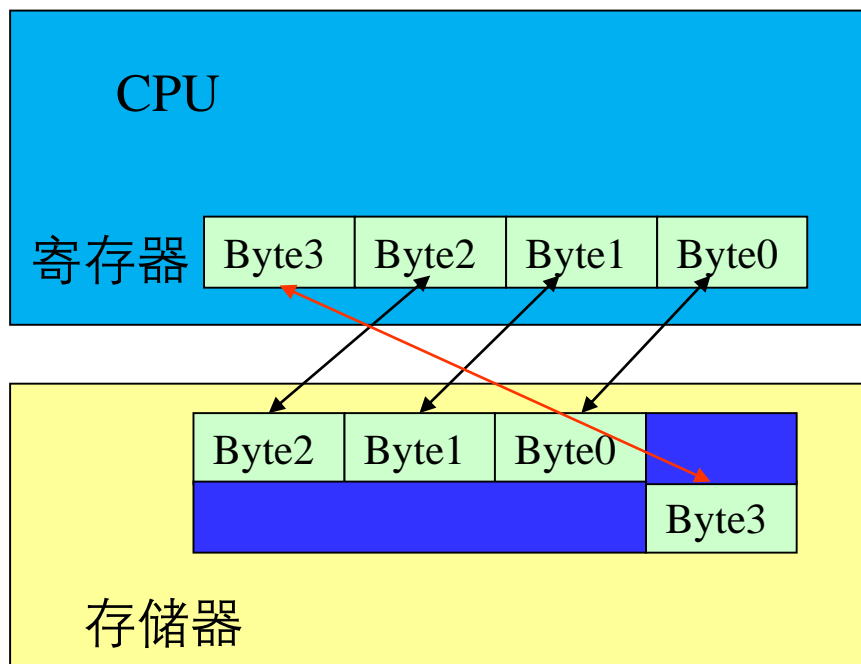
- 访存次数问题
  - char c;
  - short int i,j;
  - int k;
- 跨页问题

short j	short int i	char c
int k		short j
		int k



# 不对齐方式下的存储器访问

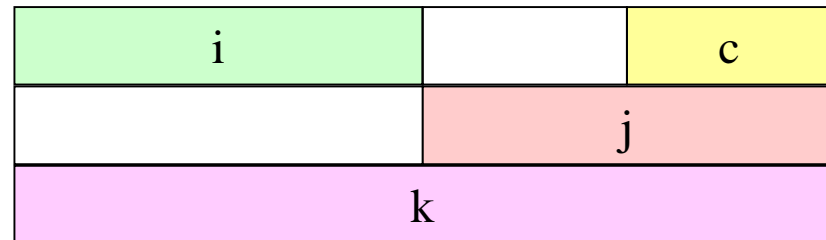
- 节省空间
- 访存速度慢
- 接口复杂





# 对齐方式下的数据存储

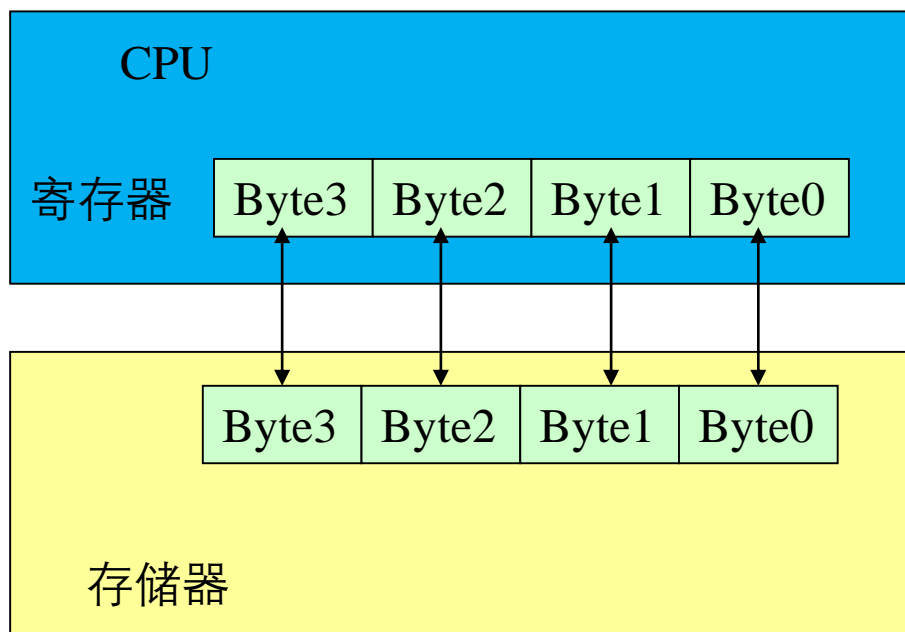
- 空间浪费问题
  - char c;
  - short int i,j;
  - int k;





# 对齐方式下的存储器访问

- 速度较高
- 接口较简单





# 对齐存储



- 数据的存储：
  - char: 字符型数据: 字节对齐存储
  - short 短整型: 双字节对齐存储
  - int, float 整型/浮点型: 4字节对齐存储
  - double : 8字节对齐存储
- 数据所存放在的起始内存地址（按字节），是该数据类型长度的长度（按字节）的倍数

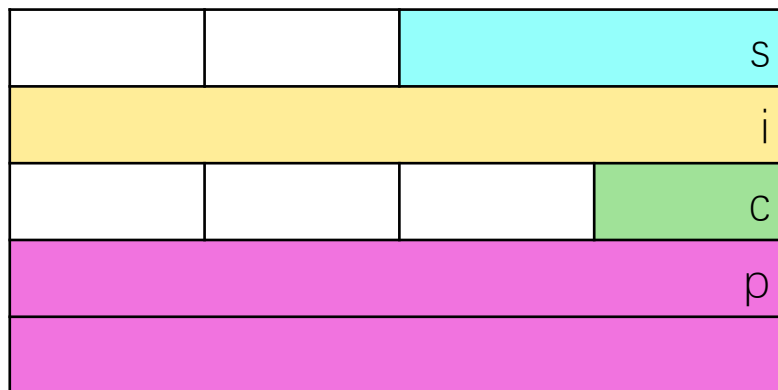


# 数据对齐的例子

- 试改变以下数据结构定义以减少对齐的开销

```
struct Loose {  
    short s;           // 16 bit  
    int i;             // 32bit  
    char c;            // 8 bit  
    Foo* p;            // 64 bit  
};
```

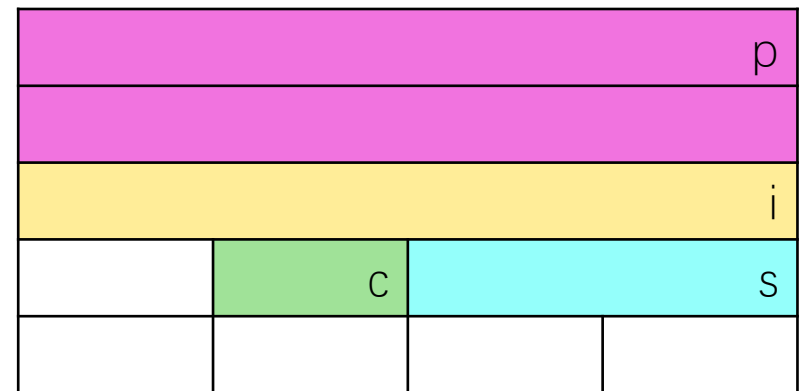
改变前:





```
struct Tight {  
    Foo* p; // 64 bit  
    int i;           // 32bit  
    short s; // 16 bit  
    char c; // 8 bit  
};
```

改变后:





# 小结

- 数据存储字节顺序
  - 大数端
  - 小数端
- 数据存储方式
  - 对齐
  - 非对齐