

## C. [111011]

可以看到第二个和第三个位向量可以通过对第一个位向量做符号扩展得到。


值得一提的是，从一个数据大小到另一个数据大小的转换，以及无符号和有符号数字之间的转换的相对顺序能够影响一个程序的行为。考虑下面的代码：

```
1 short sx = -12345;      /* -12345 */
2 unsigned uy = sx;      /* Mystery! */
3
4 printf("uy = %u:\t", uy);
5 show_bytes((byte_pointer) &uy, sizeof(unsigned));
```

在一台大端法机器上，这部分代码产生如下输出：

```
uy = 4294954951: ff ff cf c7
```

这表明当把 short 转换成 unsigned 时，我们先要改变大小，之后再完成从有符号到无符号的转换。也就是说 (unsigned) sx 等价于 (unsigned) (int) sx，求值得到 4 294 954 951，而等价于 (unsigned) (unsigned short) sx，后者求值得到 53 191。事实上，这个规则是 C 语言标准要求的。

 **练习题 2.23** 考虑下面的 C 函数：

```
int fun1(unsigned word) {
    return (int) ((word << 24) >> 24);
}

int fun2(unsigned word) {
    return ((int) word << 24) >> 24;
}
```

假设在一个采用补码运算的机器上以 32 位程序来执行这些函数。还假设有符号数值的右移是算术右移，而无符号数值的右移是逻辑右移。

A. 填写下表，说明这些函数对几个示例参数的结果。你会发现用十六进制表示来会更方便，只要记住十六进制数字 8 到 F 的最高有效位等于 1。

w	fun1(w)	fun2(w)
0x00000076		
0x87654321		
0x000000C9		
0xEDCBA987		

B. 用语言来描述这些函数执行的有用的计算。

## 2.2.7 截断数字

假设我们不用额外的位来扩展一个数值，而是减少表示一个数字的位数。例如下面代码中这种情况：

```
1 int x = 53191;
2 short sx = (short) x; /* -12345 */
3 int y = sx;          /* -12345 */
```

当我们把 x 强制类型转换为 short 时，我们就将 32 位的 int 截断为了 16 位的 short int。