位运算基础

- `&`
  - 按位与
  - 如果两个相应的二进制位都为1，则该位的结果值为1，否则为0
- `|`
  - 按位或
  - 两个相应的二进制位中只要有一个为1，该位的结果值为1
- `^`
  - 按位异或
  - 若参加运算的两个二进制位值相同则为0，否则为1
- `~`
  - 取反
  - ~是一元运算符，用来对一个二进制数按位取反，即将0变1，将1
- `<<`
  - 左移
  - 用来将一个数的各二进制位全部左移N位，右补0
- `>>`
  - 右移
  - 将一个数的各二进制位右移N位，移到右端的低位被舍弃，对于无符号数， 高位补0

1.二进制反码或按位取反：～

一元运算符～把1变为0，把0变为1。如下例子所示：

～(10011010)　// 表达式

(01100101)　　// 结果值

假设val的类型是unsigned　char，已被赋值为2。在二进制中，00000010表示2。那么，～val的值是11111101，即253。注意，该运算符不会改变val的值，就像3 * val不会改变val的值一样， val仍然是2。但是，该运算符确实创建了一个可以使用或赋值的新值：

```
newval = ～val;

printf("%d", ～val);
```

如果要把val的值改为～val，使用下面这条语句：

val = ～val;

2.按位与：**&**

二元运算符&通过逐位比较两个运算对象，生成一个新值。对于每个位，只有两个运算对象中相应的位都为1时，结果才为1（从真/假方面看，只有当两个位都为真时，结果才为真）。因此，对下面的表达式求值：

(10010011) & (00111101)　　// 表达式

由于两个运算对象中编号为4和0的位都为1，得：

(00010001)　　// 结果值

C有一个按位与和赋值结合的运算符：&=。下面两条语句产生的最终结果相同：

val &= 0377;

val = val & 0377;

3.按位或：|

二元运算符|，通过逐位比较两个运算对象，生成一个新值。对于每个位，如果两个运算对象中相应的位为1，结果就为1（从真/假方面看，如果两个运算对象中相应的一个位为真或两个位都为真，那么结果为真）。因此，对下面的表达式求值：

(10010011) | (00111101) // 表达式

除了编号为6的位，这两个运算对象的其他位至少有一个位为1，得：

(10111111) // 结果值

C有一个按位或和赋值结合的运算符：|=。下面两条语句产生的最终作用相同：

val |= 0377;

val = val | 0377;

4.按位异或：^

二元运算符^逐位比较两个运算对象。对于每个位，如果两个运算对象中相应的位一个为1（但不是两个为1），结果为1（从真/假方面看，如果两个运算对象中相应的一个位为真且不是两个为同为1，那么结果为真）。因此，对下面表达式求值：

(10010011) ^ (00111101) // 表达式

编号为0的位都是1，所以结果为0，得：

(10101110)　　// 结果值

C有一个按位异或和赋值结合的运算符：^=。下面两条语句产生的最终作用相同：

val ^= 0377;

val = val ^ 0377;

以下内容非常重要!!!! 在操作寄存器时，常用

15.3.2 用法：掩码

按位与运算符常用于掩码（mask）。所谓掩码指的是一些设置为开（1）或关（0）的位组合。要明白称其为掩码的原因，先来看通过&把一个量与掩码结合后发生什么情况。例如，假设定义符号常量MASK为2　（即，二进制形式为00000010），只有1号位是1，其他位都是0。下面的语句：

flags = flags & MASK;

把flags中除1号位以外的所有位都设置为0，因为使用按位与运算符（&）任何位与0组合都得0。1号位的值不变（如果1号位是1，那么　1&1得

1；如果 1号位是0，那么 0&1也得0）。这个过程叫作“使用掩码”，因为掩码中的0隐藏了flags中相应的位。

可以这样类比：把掩码中的0看作不透明，1看作透明。表达式flags & MASK相当于用掩码覆盖在flags的位组合上，只有MASK为1的位才可见（见图15.2）。

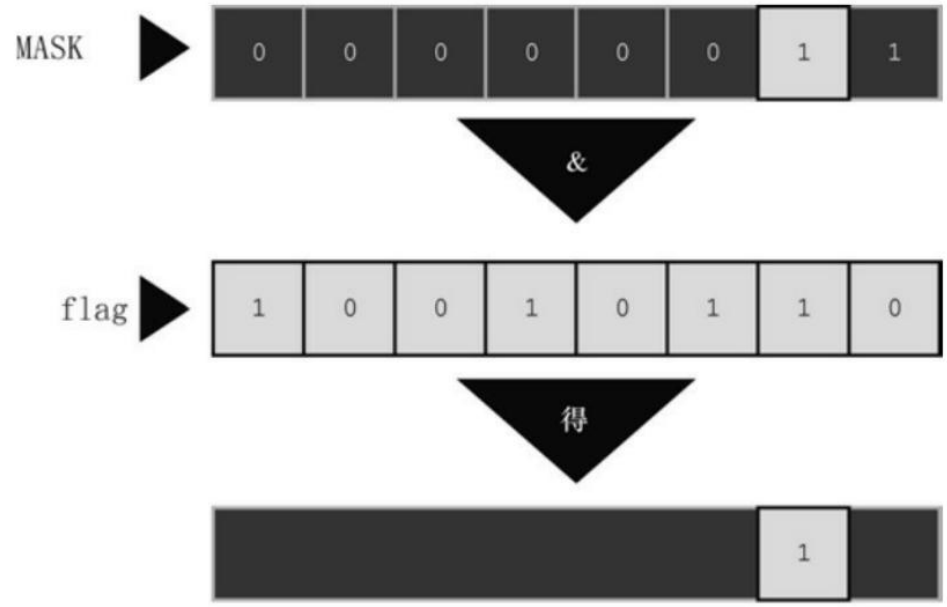

图15.2 掩码示例

用&=运算符可以简化前面的代码，如下所示：

```
flags &= MASK;
```

下面这条语句是按位与的一种常见用法：

```
ch &= 0xff; /* 或者 ch &= 0377; */
```

前面介绍过oxff的二进制形式是11111111，八进制形式是0377。这个掩码保持ch中最后8位不变，其他位都设置为0。无论ch原来是8位、16位或是其他更多位，最终的值都被修改为1个8位字节。在该例中，掩码的宽度为8

位。

15.3.3 用法：打开位（设置位）

有时，需要打开一个值中的特定位，同时保持其他位不变。例如，一台IBM PC 通过向端口发送值来控制硬件。例如，为了打开内置扬声器，必须打开 1 号位，同时保持其他位不变。这种情况可以使用按位或运算符（|）。

以上一节的flags和MASK（只有1号位为1）为例。下面的语句：

```
flags = flags | MASK;
```

把flags的1号位设置为1，且其他位不变。因为使用|运算符，任何位与0组合，结果都为本身；任何位与1组合，结果都为1。

例如，假设flags是00001111，MASK是10110110。下面的表达式：

```
flags | MASK
```

即是：

```
(00001111) | (10110110)      // 表达式
```

其结果为：

```
(10111111)                    // 结果值
```

MASK中为1的位，flags与其对应的位也为1。MASK中为0的位，flags与其对应的位不变。

用|=运算符可以简化上面的代码，如下所示：

```
flags |= MASK;
```

同样，这种方法根据MASK中为1的位，把flags中对应的位设置为1，其

他位不变。

15.3.4 用法：关闭位（清空位）

和打开特定的位类似，有时也需要在不影响其他位的情况下关闭指定的位。假设要关闭变量flags中的1号位。同样，MASK只有1号位为1（即，打开）。可以这样做：

```
flags = flags & ~MASK;
```

由于MASK除1号位为1以外，其他位全为0，所以~MASK除1号位为0以外，其他位全为1。使用&，任何位与1组合都得本身，所以这条语句保持1号位不变，改变其他各位。另外，使用&，任何位与0组合都的0。所以无论1号位的初始值是什么，都将其设置为0。

例如，假设flags是00001111，MASK是10110110。下面的表达式：

```
flags & ~MASK
```

即是：

```
(00001111) & ~(10110110)   // 表达式
```

其结果为：

```
(00001001)                 // 结果值
```

MASK中为1的位在结果中都被设置（清空）为0。flags中与MASK为0的位相应的位在结果中都未改变。

可以使用下面的简化形式：

```
flags &= ~MASK;
```

15.3.5 用法：切换位

切换位指的是打开已关闭的位，或关闭已打开的位。可以使用按位异或运算符（^）切换位。也就是说，假设b是一个位（1或0），如果b为1，则1^b为0；如果b为0，则1^b为1。另外，无论b为1还是0，0^b均为b。因此，如果使用^组合一个值和一个掩码，将切换该值与MASK为1的位相对应的位，该

值与MASK为0的位相对应的位不变。要切换flags中的1号位，可以使用下面两种方法：

```
flags = flags ^ MASK;

flags ^= MASK;
```

例如，假设flags是00001111，MASK是10110110。表达式：

```
flags ^ MASK
```

即是：

```
(00001111) ^ (10110110)      // 表达式
```

其结果为：

```
(10111001)                   // 结果值
```

flags中与MASK为1的位相对应的位都被切换了，MASK为0的位相对应的位不变。

15.3.6 用法：检查位的值

前面介绍了如何改变位的值。有时，需要检查某位的值。例如，flags中1号位是否被设置为1？不能这样直接比较flags和MASK：

```
if (flags == MASK)
puts("Wow!"); /* 不能正常工作 */
```

这样做即使flags的1号位为1，其他位的值会导致比较结果为假。因此，必须覆盖flags中的其他位，只用1号位和MASK比较：

```
if ((flags & MASK) == MASK)
puts("Wow!");
```

由于按位运算符的优先级比==低，所以必须在flags & MASK周围加上圆括号。

为了避免信息漏过边界，掩码至少要与其覆盖的值宽度相同。

3.用法：移位运算符

移位运算符针对2的幂提供快速有效的乘法和除法：

number << n　　number乘以2的n次幂

number >> n　　如果number为非负，则用number除以2的n次幂

这些移位运算符类似于在十进制中移动小数点来乘以或除以10。

移位运算符还可用于从较大单元中提取一些位。例如，假设用一个unsigned long类型的值表示颜色值，低阶位字节储存红色的强度，下一个字节储存绿色的强度，第 3 个字节储存蓝色的强度。随后你希望把每种颜色的强度分别储存在3个不同的unsigned char类型的变量中。那么，可以使用下面的语句：

```
#define BYTE_MASK 0xff

unsigned long color = 0x002a162f;

unsigned char blue, green, red;

red = color & BYTE_MASK;

green = (color >> 8) & BYTE_MASK;

blue = (color >> 16) & BYTE_MASK;
```

以上代码中，使用右移运算符将 8 位颜色值移动至低阶字节，然后使用掩码技术把低阶字节赋给指定的变量。