


回到  $y=2^k$  的情况，C 表达式  $x+(1<<k)-1$  得到数值  $x+2^k-1$ 。将这个值算术右移  $k$  位即产生  $\lfloor x/2^k \rfloor$ 。


这个分析表明对于使用算术右移的补码机器，C 表达式

```
(x<0 ? x+(1<<k)-1 : x) >> k
```

将会计算数值  $x/2^k$ 。

 **练习题 2.42** 写一个函数 `div16`，对于整数参数  $x$  返回  $x/16$  的值。你的函数不能使用除法、模运算、乘法、任何条件语句（`if` 或者 `?:`）、任何比较运算符（例如 `<`、`>` 或 `==`）或任何循环。你可以假设数据类型 `int` 是 32 位长，使用补码表示，而右移是算术右移。

现在我们可以看到，除以 2 的幂可以通过逻辑或者算术右移来实现。这也正是为什么大多数机器上提供这两种类型的右移。不幸的是，这种方法不能推广到除以任意常数。同乘法不同，我们不能用除以 2 的幂的除法来表示除以任意常数  $K$  的除法。

 **练习题 2.43** 在下面的代码中，我们省略了常数  $M$  和  $N$  的定义：

```
#define M      /* Mystery number 1 */
#define N      /* Mystery number 2 */
int arith(int x, int y) {
    int result = 0;
    result = x*M + y/N; /* M and N are mystery numbers. */
    return result;
}
```

我们以某个  $M$  和  $N$  的值编译这段代码。编译器用我们讨论过的方法优化乘法和除法。下面是将产生出的机器代码翻译回 C 语言的结果：

```
/* Translation of assembly code for arith */
int optarith(int x, int y) {
    int t = x;
    x <= 5;
    x -= t;
    if (y < 0) y += 7;
    y >>= 3; /* Arithmetic shift */
    return x+y;
}
```

$M$  和  $N$  的值为多少？

### 2.3.8 关于整数运算的最后思考

正如我们看到的，计算机执行的“整数”运算实际上是一种模运算形式。表示数字的有限字长限制了可能的值的取值范围，结果运算可能溢出。我们还看到，补码表示提供了一种既能表示负数也能表示正数的灵活方法，同时使用了与执行无符号算术相同的位级实现，这些运算包括像加法、减法、乘法，甚至除法，无论运算数是以无符号形式还是以补码形式表示的，都有完全一样或者非常类似的位级行为。

我们看到了 C 语言中的某些规定可能会产生令人意想不到的结果，而这些结果可能是难以察觉或理解的缺陷的源头。我们特别看到了 `unsigned` 数据类型，虽然它概念上很简单，但可能导致即使是资深程序员都意想不到的行为。我们还看到这种数据类型会以出乎意料的方式出现，比如，当书写整数常数和当调用库函数时。