x		$-\frac{1}{4}x$	
十六进制	十进制	十进制	十六进制
0	0	0	0
5	5	−5	B
8	−8	−8	8
D	−3	3	3
F	−1	1	1

对于无符号数的非，位的模式是相同的。

2.34　本题目是确保你理解了补码乘法。

模式	x		y		$x \cdot y$		截断了的$x \cdot y$	
无符号数	4	[100]	5	[101]	20	[010100]	4	[100]
补码	−4	[100]	−3	[101]	12	[001100]	−4	[100]
无符号数	2	[010]	7	[111]	14	[001110]	6	[110]
补码	2	[010]	−1	[111]	−2	[111110]	−2	[110]
无符号数	6	[110]	6	[110]	36	[100100]	4	[100]
补码	−2	[110]	−2	[110]	4	[000100]	−4	[100]

2.35　对所有可能的 x 和 y 测试一遍这个函数是不现实的。当数据类型 int 为 32 位时，即使你每秒运行一百亿个测试，也需要 58 年才能测试完所有的组合。另一方面，把函数中的数据类型改成 short 或者 char，然后再穷尽测试，倒是测试代码的一种可行的方法。

我们提出以下论据，这是一个更理论的方法：

1) 我们知道 $x \cdot y$ 可以写成一个 $2w$ 位的补码数字。用 u 来表示低 w 位表示的无符号数，v 表示高 w 位的补码数字。那么，根据公式(2.3)，我们可以得到 $x \cdot y = v2^w + u$。

我们还知道 $u = T2U_w(p)$，因为它们是从同一个位模式得出来的无符号和补码数字，因此根据等式(2.6)，我们有 $u = p + p_{w-1}2^w$，这里 p_{w-1} 是 p 的最高有效位。设 $t = v + p_{w-1}$，我们有 $x \cdot y = p + t2^w$。

当 $t=0$ 时，有 $x \cdot y = p$；乘法不会溢出。当 $t \neq 0$ 时，有 $x \cdot y \neq p$；乘法不会溢出。

2) 根据整数除法的定义，用非零数 x 除以 p 会得到商 q 和余数 r，即 $p = x \cdot q + r$，且 $|r| < |x|$。(这里用的是绝对值，因为 x 和 r 的符号可能不一致。例如，−7 除以 2 得到商 −3 和余数 −1。)

3) 假设 $q = y$。那么有 $x \cdot y = x \cdot y + r + t2^w$。在此，我们可以得到 $r + t2^w = 0$。但是 $|r| < |x| \leqslant 2^w$，所以只有当 $t=0$ 时，这个等式才会成立，此时 $r=0$。

假设 $r = t = 0$。那么我们有 $x \cdot y = x \cdot q$，隐含有 $y = q$。

当 $x=0$ 时，乘法不溢出，所以我们的代码提供了一种可靠的方法来测试补码乘法是否会导致溢出。

2.36　如果用 64 位表示，乘法就不会有溢出。然后我们来验证将乘积强制类型转换为 32 位是否会改变它的值：

```
1   /* Determine whether the arguments can be multiplied
2      without overflow */
3   int tmult_ok(int x, int y) {
4       /* Compute product without overflow */
5       int64_t pll = (int64_t) x*y;
6       /* See if casting to int preserves value */
7       return pll == (int) pll;
8   }
```

注意，第 5 行右边的强制类型转换至关重要。如果我们将这一行写成

```
int64_t pll = x*y;
```

就会用 32 位值来计算乘积(可能会溢出)，然后再符号扩展到 64 位。