总结一下，我们考虑无符号与补码表示之间互相转换的结果。对于在范围 $0 \leqslant x \leqslant TMax_w$ 之内的值 x 而言，我们得到 $T2U_w(x) = x$ 和 $U2T_w(x) = x$。也就是说，在这个范围内的数字有相同的无符号和补码表示。对于这个范围以外的数值，转换需要加上或者减去 2^w。例如，我们有 $T2U_w(-1) = -1 + 2^w = UMax_w$——最靠近 0 的负数映射为最大的无符号数。在另一个极端，我们可以看到 $T2U_w(TMin_w) = -2^{w-1} + 2^w = 2^{w-1} = TMax_w + 1$——最小的负数映射为一个刚好在补码的正数范围之外的无符号数。使用图 2-15 的示例，我们能看到 $T2U_{16}(-12\,345) = 65\,563 + -12\,345 = 53\,191$。

2.2.5　C 语言中的有符号数与无符号数

如图 2-9 和图 2-10 所示，C 语言支持所有整型数据类型的有符号和无符号运算。尽管 C 语言标准没有指定有符号数要采用某种表示，但是几乎所有的机器都使用补码。通常，大多数数字都默认为是有符号的。例如，当声明一个像 12345 或者 0x1A2B 这样的常量时，这个值就被认为是有符号的。要创建一个无符号常量，必须加上后缀字符 'U' 或者 'u'，例如，12345U 或者 0x1A2Bu。

C 语言允许无符号数和有符号数之间的转换。虽然 C 标准没有精确规定应如何进行这种转换，但大多数系统遵循的原则是底层的位表示保持不变。因此，在一台采用补码的机器上，当从无符号数转换为有符号数时，效果就是应用函数 $U2T_w$，而从有符号数转换为无符号数时，就是应用函数 $T2U_w$，其中 w 表示数据类型的位数。

显式的强制类型转换就会导致转换发生，就像下面的代码：

```
1       int tx, ty;
2       unsigned ux, uy;
3
4       tx = (int) ux;
5       uy = (unsigned) ty;
```

另外，当一种类型的表达式被赋值给另外一种类型的变量时，转换是隐式发生的，就像下面的代码：

```
1       int tx, ty;
2       unsigned ux, uy;
3
4       tx = ux; /* Cast to signed */
5       uy = ty; /* Cast to unsigned */
```

当用 printf 输出数值时，分别用指示符 %d、%u 和 %x 以有符号十进制、无符号十进制和十六进制格式输出一个数字。注意 printf 没有使用任何类型信息，所以它可以用指示符 %u 来输出类型为 int 的数值，也可以用指示符 %d 输出类型为 unsigned 的数值。例如，考虑下面的代码：

```
1       int x = -1;
2       unsigned u = 2147483648; /* 2 to the 31st */
3
4       printf("x = %u = %d\n", x, x);
5       printf("u = %u = %d\n", u, u);
```

当在一个 32 位机器上运行时，它的输出如下：

```
x = 4294967295 = -1
u = 2147483648 = -2147483648
```