$\vec{x}$		$B2U_4(\vec{x})$	$B2T_4(\vec{x})$
十六进制	二进制		
0xE	[1110]	$2^3+2^2+2^1=14$	$-2^3+2^2+2^1=-2$
0x0			
0x5			
0x8			
0xD			
0xF			

图 2-14 展示了针对不同字长，几个重要数字的位模式和数值。前三个给出的是可表示的整数的范围，用 $UMax_w$、$TMin_w$ 和 $TMax_w$ 来表示。在后面的讨论中，我们还会经常引用到这三个特殊的值。如果可以从上下文中推断出 w，或者 w 不是讨论的主要内容时，我们会省略下标 w，直接引用 $UMax$、$TMin$ 和 $TMax$。

数	字长w			
	8	16	32	64
$UMax_w$	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
	255	65 535	4 294 967 295	18 446 744 073 709 551 615
$TMin_w$	0x80	0x8000	0x80000000	0x8000000000000000
	−128	−32 768	−2 147 483 648	−9 223 372 036 854 775 808
$TMax_w$	0x7F	0x7FFF	0x7FFFFFFF	0x7FFFFFFFFFFFFFFF
	127	32 767	2 147 483 647	9 223 372 036 854 775 807
−1	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
0	0x00	0x0000	0x00000000	0x0000000000000000

图 2-14　重要的数字。图中给出了数值和十六进制表示

关于这些数字，有几点值得注意。第一，从图 2-9 和图 2-10 可以看到，补码的范围是不对称的：$|TMin| = |TMax| + 1$，也就是说，$TMin$ 没有与之对应的正数。正如我们将会看到的，这导致了补码运算的某些特殊的属性，并且容易造成程序中细微的错误。之所以会有这样的不对称性，是因为一半的位模式(符号位设置为 1 的数)表示负数，而另一半(符号位设置为 0 的数)表示非负数。因为 0 是非负数，也就意味着能表示的整数比负数少一个。第二，最大的无符号数值刚好比补码的最大值的两倍大一点：$UMax_w = 2TMax_w + 1$。补码表示中所有表示负数的位模式在无符号表示中都变成了正数。图 2-14 也给出了常数−1 和 0 的表示。注意−1 和 $UMax$ 有同样的位表示——一个全 1 的串。数值 0 在两种表示方式中都是全 0 的串。

C 语言标准并没有要求要用补码形式来表示有符号整数，但是几乎所有的机器都是这么做的。程序员如果希望代码具有最大可移植性，能够在所有可能的机器上运行，那么除了图 2-11 所示的那些范围之外，我们不应该假设任何可表示的数值范围，也不应该假设有符号数会使用何种特殊的表示方式。另一方面，许多程序的书写都假设用补码来表示有符号数，并且具有图 2-9 和图 2-10 所示的“典型的”取值范围，这些程序也能够在大量的机器和编译器上移植。C 库中的文件<limits.h>定义了一组常量，来限定编译器运行的这台机器的不同整型数据类型的取值范围。比如，它定义了常量 INT_MAX、INT_MIN 和 UINT_MAX，它们描述了有符号和无符号整数的范围。对于一个补码的机器，数据类型 int 有 w 位，这些常量就对应于 $TMax_w$、$TMin_w$ 和 $UMax_w$ 的值。