这种不寻常的方式来写 $TMin_{32}$。虽然理解这个问题需要我们钻研 C 语言标准的一些比较隐晦的角落，但是它能够帮助我们充分领会整数数据类型和表示的一些细微之处。

2.2.6　扩展一个数字的位表示

一个常见的运算是在不同字长的整数之间转换，同时又保持数值不变。当然，当目标数据类型太小以至于不能表示想要的值时，这根本就是不可能的。然而，从一个较小的数据类型转换到一个较大的类型，应该总是可能的。

要将一个无符号数转换为一个更大的数据类型，我们只要简单地在表示的开头添加 0。这种运算被称为零扩展(zero extension)，表示原理如下：

原理：无符号数的零扩展

定义宽度为 w 的位向量 $\vec{u}=[u_{w-1}, u_{w-2}, \cdots, u_0]$ 和宽度为 w' 的位向量 $\vec{u}'=[0, \cdots, 0, u_{w-1}, u_{w-2}, \cdots, u_0]$，其中 $w'>w$。则 $B2U_w(\vec{u})=B2U_{w'}(\vec{u}')$。

按照公式(2.1)，该原理可以看作是直接遵循了无符号数编码的定义。

要将一个补码数字转换为一个更大的数据类型，可以执行一个符号扩展(sign extension)，在表示中添加最高有效位的值，表示为如下原理。我们用蓝色标出符号位 x_{w-1} 来突出它在符号扩展中的角色。

原理：补码数的符号扩展

定义宽度为 w 的位向量 $\vec{x}=[x_{w-1}, x_{w-2}, \cdots, x_0]$ 和宽度为 w 的位向量 $\vec{x}'=[x_{w-1}, \cdots, x_{w-1}, x_{w-1}, x_{w-2}, \cdots, x_0]$，其中 $w'>w$。则 $B2T_w(\vec{x})=B2T_{w'}(\vec{x}')$。

例如，考虑下面的代码：

```
short sx = -12345;          /* -12345 */
unsigned short usx = sx;    /*  53191 */
int x = sx;                 /* -12345 */
unsigned ux = usx;          /*  53191 */

printf("sx  = %d:\t", sx);
show_bytes((byte_pointer) &sx, sizeof(short));
printf("usx = %u:\t", usx);
show_bytes((byte_pointer) &usx, sizeof(unsigned short));
printf("x   = %d:\t", x);
show_bytes((byte_pointer) &x, sizeof(int));
printf("ux  = %u:\t", ux);
show_bytes((byte_pointer) &ux, sizeof(unsigned));
```

在采用补码表示的 32 位大端法机器上运行这段代码时，打印出如下输出：

```
sx  = -12345:  cf c7
usx = 53191:   cf c7
x   = -12345:  ff ff cf c7
ux  = 53191:   00 00 cf c7
```

我们看到，尽管−12 345 的补码表示和 53 191 的无符号表示在 16 位字长时是相同的，但是在 32 位字长时却是不同的。特别地，−12 345 的十六进制表示为 0xFFFFCFC7，而 53 191 的十六进制表示为 0x0000CFC7。前者使用的是符号扩展——最开头加了 16 位，都是最高有效位 1，表示为十六进制就是 0xFFFF。后者开头使用 16 个 0 来扩展，表示为十六进制就是 0x0000。

图 2-20 给出了从字长 $w=3$ 到 $w=4$ 的符号扩展的结果。位向量[101]表示值−4+1=