

# 第二单元学习笔记

yinxuhao [xuhao\_yin@163.com]

December 27, 2022

## Contents

<b>1</b>	<b>引言</b>	<b>2</b>
<b>2</b>	<b>信息存储</b>	<b>2</b>
2.1	十六进制表示法	2
2.2	字数据大小	3
2.3	寻址和字节顺序	3
2.4	布尔代数	5
2.5	移位运算	5
<b>3</b>	<b>整数表示</b>	<b>6</b>
3.1	无符号数的编码	6
3.2	补码编码	7
3.3	有符号和无符号数之间的转换	8
3.4	扩展一个数字的位表示	9
3.5	截断数字	10
3.6	关于有符号数和无符号数的建议	11
<b>4</b>	<b>整数运算</b>	<b>11</b>
4.1	无符号加法	11
4.2	补码加法	12

信息的表示和处理

1 引言

孤立地讲，单个的位不是非常有用，将位组合在一起，再加上某种解释 (interpretation)，即赋予不同的可能位模式以含意。我们就能表示任何有限集合的元素。

三种重要的数字表示：

- 1. 无符号unsigned编码给予传统的二进制表示法
- 2. 补码two's-complement编码是表示有符号整数的最常见的方式。
- 3. 浮点数floating-point编码是表示实数的科学计数法的以 2 为基数的版本。

数据溢出overflow是产生 bug 的一大原因。负数下溢产生极大的正数；正数上溢产生极小的负数。

浮点运算有完全不同的数学属性。

- 1. 由于表示的精度有限，浮点运算是不可结合的。例如

(3.14 + 1e20) - 1e20 = 0.0

but

(3.14 + 1e20 - 1e20) = 3.14

- 2. 该属性不同的原因，是处理数字表示有限性的方式不同——  
整数虽只能编码一个相对较小的数值范围，然该表示法是精确的；  
浮点数虽可以编码相对较大的数值范围，但这种表示只是近似的。

书中建议的本章学习方式：

深入学习数学语言  
学习编写公式和方程式  
以及重要属性的推导

2 信息存储

大多数计算机使用 8 位的块或者字节作为最小的可寻址内存单位，而不是内存中单独的比特。

机器级程序将内存视为一个非常大的字节数组，称为虚拟内存，所有可能的地址的集合称为虚拟地址空间virtual address space.

每个程序对象可以简单地视为一个字节块，而程序本身就是一个字节序列。

2.1 十六进制表示法

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111
Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

Figure 1: 十六进制表示法。每个十六进制数字都对 16 个值中的一个进行了编码

十六进制转二进制：将十六进制的每一位转换为二进制格式，然后拼接。例如：

十六进制	1	7	3	A	4	C
二进制	0001	0111	0011	1010	0100	1100

所以  $binary_{0x173a4c_{16}} = 000101110011101001001100_2$ 。

二进制转十六进制：将二进制从右到左做 4 个一组的划分，如最左侧不足 4 位则以 0 补之。然后将每个 4 位转换为对应的十六进制数字拼接即可。例如：

二进制	11	1100	1010	1101	1011	0011
十六进制	3	C	A	D	B	3

所以， $hex_{1111001010110110110011_2} = 3cadb3_{16}$

## 2.2 字数据大小

每台计算机都有一个字长，指明指针数据的标称大小。  
C 数据类型的典型大小见下图：

C declaration		Bytes	
Signed	Unsigned	32-bit	64-bit
[signed] char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned	4	4
long	unsigned long	4	8
int32_t	uint32_t	4	4
int64_t	uint64_t	8	8
char *		4	8
float		4	4
double		8	8

Figure 2: 基本 C 数据类型的典型大小 (以字节为单位)

## 2.3 寻址和字节顺序

**小端法**little endian: 最低有效字节在最前面放着。

**大端法**big endian: 最高有效字节在最前面放着。

具体示例见下图：

Big endian					
	0x100	0x101	0x102	0x103	
...	01	23	45	67	...

Little endian					
	0x100	0x101	0x102	0x103	
...	67	45	23	01	...

Figure 3: 大端法与小端法

```
#include <stdio.h>

typedef unsigned char *byte_pointer;

void show_bytes(byte_pointer start, size_t len) {
    size_t i;
    for(i = 0; i < len; i++) {
        printf(" %.2x", start[i]);
    }
    printf("\n");
}

void show_int(int x) {
    show_bytes((byte_pointer) &x, sizeof(int));
}

void show_float(float x);

void show_pointer(void *x);

void test_show_bytes(int val) {
    int ival = val;
    float fval = (float) val;
    int *pval = &ival;
    show_int(ival);
    show_float(fval);
    show_pointer(pval);
}
```

通过以上代码，可以打印出数据的两位十六进制格式输出。对比结果可以发现，int和float的结果一样，只是排列的大小端不同，而指针值不同，与机器相关。

二进制代码是不兼容的。

## 2.4 布尔代数

$\sim$		$\&$	0	1	$ $	0	1	$\wedge$	0	1
0	1	0	0	0	0	0	1	0	0	1
1	0	1	0	1	1	1	1	1	1	0

Figure 4: 布尔代数的运算。二进制 0 和 1 代表逻辑值 TRUE 和 FALSE. 以上四张图依次是逻辑运算符 NOT AND OR EXCLUSIVE-OR

位向量一个很有用的应用就是**表示有限集合**。利用位向量  $[a_{w-1}, \dots, a_1, a_0]$  可以编码任何子集  $A \in 0, 1, \dots, w-1$ 。

例如，定义规则  $a_i = 1 \iff i \in A$ 。

位向量  $a \doteq [01101001]$  表示集合  $A = 0, 3, 5, 6$ ，而位向量  $b \doteq [01010101]$  表示集合  $B = 0, 2, 4, 6$ 。

编码集合的使用方法是使用布尔运算。

例如： $a \& b \rightarrow [01000001]$ ，对应于  $A \cap B = 0, 6$ 。

它的实际应用，还有使用位向量作为掩码有选择地使用或屏蔽一些信号，该掩码就是设置为有效信号的集合。

C 语言中的位级运算，其实是按照各个位对应的位运算来的。

而 C 语言中的逻辑运算 (`||`、`&&`、`!`) 则是把所有的非零参数都表示 TRUE，参数 0 表示为 FALSE。它们只返回 1 或 0。而位级运算只在参数特殊时才与之有相同的结果。

## 2.5 移位运算

$x \ll k$ : 左移  $k$  位，即丢弃最高  $k$  位，右端补充  $k$  个 0。

$x \gg k$ : 右移  $k$  位，支持逻辑右移和算术右移。逻辑右移在左端补充  $k$  个 0，算术右移则在左端补充  $k$  个最高有效位 (符号位)。

**对无符号数，右移必须是逻辑的。**

**移位运算符是从左至右可结合的。**

### 3 整数表示

Symbol	Type	Meaning
$B2T_w$	Function	Binary to two's complement
$B2U_w$	Function	Binary to unsigned
$U2B_w$	Function	Unsigned to binary
$U2T_w$	Function	Unsigned to two's complement
$T2B_w$	Function	Two's complement to binary
$T2U_w$	Function	Two's complement to unsigned
$TMin_w$	Constant	Minimum two's-complement value
$TMax_w$	Constant	Maximum two's-complement value
$UMax_w$	Constant	Maximum unsigned value
$+_w^t$	Operation	Two's-complement addition
$+_w^u$	Operation	Unsigned addition
$*_w^t$	Operation	Two's-complement multiplication
$*_w^u$	Operation	Unsigned multiplication
$-_w^t$	Operation	Two's-complement negation
$-_w^u$	Operation	Unsigned negation

Figure 5: 整数的数据与算术操作术语。下标  $w$  表示数据中表示中的位数

#### 3.1 无符号数的编码

**原理 1** 无符号数编码的定义

对向量  $\vec{x} = [x_{w-1}, x_{w-2}, \dots, x_0]$ :

$$B2U_w(\vec{x}) \doteq \sum_{i=0}^{w-1} x_i 2^i \quad (1)$$

形象的展示如下图:

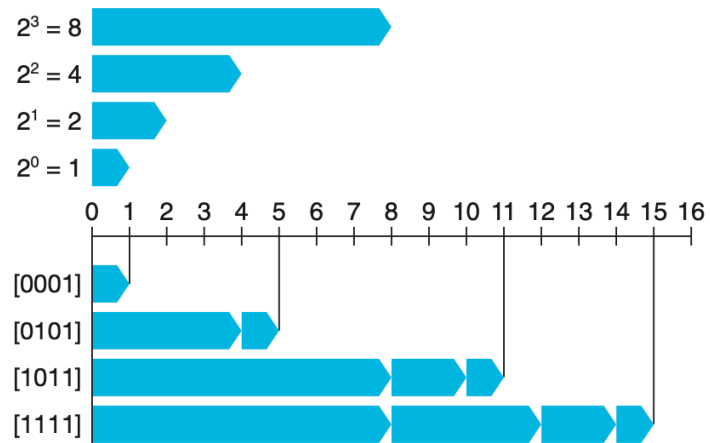


Figure 6:  $w=4$  的无符号数示例。当二进制表示中位  $i$  为 1，数值就会相应加上  $2^i$

**原理 2** 无符号数编码的唯一性  
函数  $B2U_w$  是一个双射

### 3.2 补码编码

**原理 3** 补码编码的定义  
对向量  $\vec{x} = [x_{w-1}, x_{w-2}, \dots, x_0]$ :

$$B2T_w(\vec{x}) \doteq -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i \quad (2)$$

形象地展示如下图:

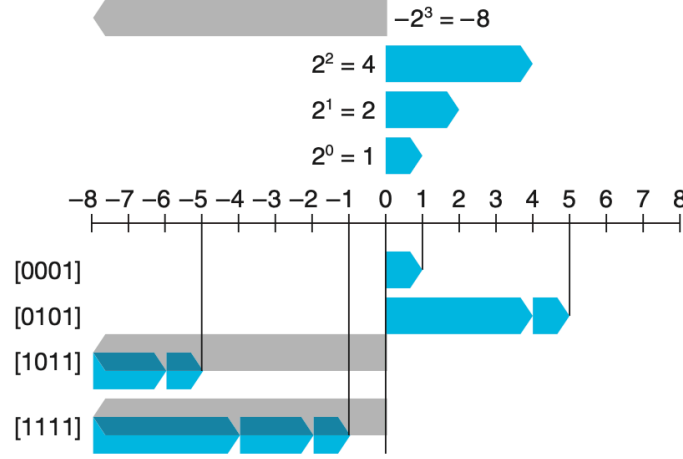


Figure 7:  $w=4$  的补码示例。把位 3 作为符号位，因此当它为 1 时，对数值的影响是  $-2^3 = -8$ 。这个权重在图中用带向左箭头的条表示

#### 原理 4 补码编码的唯一性

函数  $B2T_w$  是一个双射。

1. 补码的范围是不对称的： $|TMin| = |TMax| + 1$ ，即 TMin 没有与之对应的正数。这是因为 0 是非负数。
2. 最大的无符号数值刚好比补码的最大值的两倍大一点： $UMax_w = 2TMax_w + 1$

### 3.3 有符号和无符号数之间的转换

#### 原理 5 补码转换为无符号数

对满足  $TMin_w \leq x \leq TMax_w$  的  $x$  有：

$$T2U_w(x) = \begin{cases} x + 2^w, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (3)$$

#### 推导 1 补码转换为无符号数

比较式 1 和 2，我们发现对于位模式  $\vec{x}$ ，如果我们计算  $B2U_w(\vec{x}) - B2T_w(\vec{x})$  之差，得到：

$$B2U_w(\vec{x}) - B2T_w(\vec{x}) = x_{w-1}2^w$$

由此得到一个关系：

$$B2U_w(\vec{x}) = x_{w-1}2^w + B2T_w(\vec{x}) \quad (4)$$

由此可得：

$$B2U_w(T2B_w(x)) = T2U_w(x) = x + x_{w-1}2^w \quad (5)$$

式 5 的计算：将  $T2B_w(x)$  当作  $x$  代入 4 后得到。由于运算  $T2B_w$  与  $B2T_w$  是对  $\vec{x}$  的逆运算，故

$$\therefore B2U_w(T2B_w(x)) = x_{w-1}2^w + B2T_w(T2B_w(x)) \therefore T2U_w(x) = x + x_{w-1}2^w$$

根据 3 的两种情况，在  $x$  的补码中，位  $x_{w-1}$  决定了  $x$  是否为负。 ■



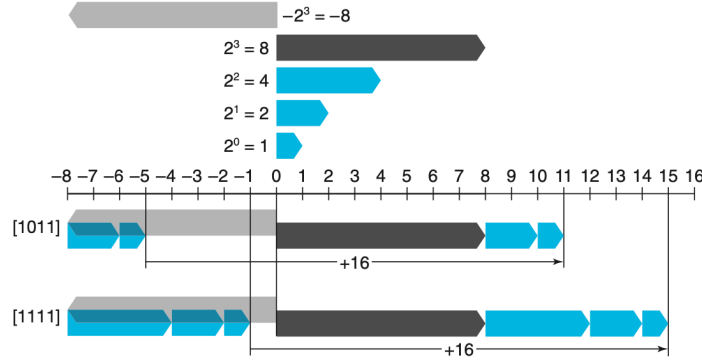


Figure 8: 比较当  $w=4$  时无符号数表示和补码表示 (对补码和无符号数来说, 最高有效位的权重分别是  $-8$  和  $+8$ , 因此产生一个差为  $16$ )

**原理 6** 无符号数转换为补码

对满足  $0 \leq x \leq UMax_w$  的  $u$  有:

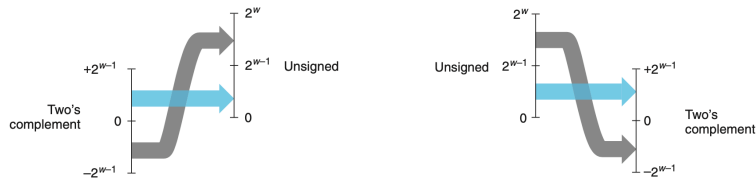
$$U2T_w(u) = \begin{cases} u, & u \leq TMax_w \\ u - 2^w, & u > TMax_w \end{cases} \quad (6)$$

**推导 2** 设  $\vec{x} = U2B_w(u)$ , 则这个位向量也是  $U2T_w(u)$  的补码表示。式1和式2结合起来有

$$U2T_w(u) = -u_{w-1}2^w + u \quad (7)$$

在  $u$  的无符号表示中, 对式6的两种情况来说, 位  $u_{w-1}$  决定了  $u$  是否大于  $TMax_w = 2^{w-1} - 1$ 。 ■

以下图说明了函数  $U2T$  的行为。对于小的数, 从无符号到有符号保留原值; 一旦大于  $TMax_w$ , 数字将被转换为一个负数值。



(a) 从补码到无符号数的转换。函数  $T2U$  将负数转换为大的正数 (b) 从无符号数到补码的转换。函数  $U2T$  把大于  $2^{w-1} - 1$  的数字转换为负值

Figure 9: 无符号数和补码的转换

### 3.4 扩展一个数字的位表示

用于将数据类型转换为一个更大的数据类型, 例如  $32$  位  $\rightarrow 64$  位。

**原理 7** 无符号数的零扩展

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

**原理 8** 补码数的符号扩展

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

**推导 3** 补码数值的符号扩展

令  $w' = w + k$ , 证明

$$B2T_{w+k}(\underbrace{[x_{w-1}, \dots, x_{w-1}]_{k \text{ times}}, x_{w-2}, \dots, x_0}) = B2T_w([x_{w-1}, x_{w-2}, \dots, x_0])$$

下面的证明是对  $k$  进行归纳。即：如果我们能够证明符号扩展一位保持了数值不变，那么符号扩展任意位都能保持这种属性。即：

$$B2T_{w+1}([x_{w-1}, x_{w-1}, x_{w-2}, \dots, x_0]) = B2T_w([x_{w-1}, x_{w-2}, \dots, x_0])$$

用式2展开左边的表达式，得：

$$\begin{aligned} B2T_{w+1}([x_{w-1}, x_{w-1}, x_{w-2}, \dots, x_0]) &= -x_{w-1}2^w + \sum_{i=0}^{w-1} x_i 2^i \\ &= -x_{w-1}2^w + x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i \\ &= -x_{w-1}(2^w - 2^{w-1}) + \sum_{i=0}^{w-2} x_i 2^i \\ &= -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i \\ &= B2T_w([x_{w-1}, x_{w-2}, \dots, x_0]). \end{aligned}$$

其中使用的关键属性是  $2^w - 2^{w-1} = 2^{w-1}$ 。 ■

### 3.5 截断数字

**原理 9** 截断无符号数

令  $\vec{x}$  等于位向量  $[x_{w-1}, x_{w-2}, \dots, x_0]$ , 而  $\vec{x}'$  是将其截断为  $k$  位的结果： $\vec{x}' = [x_{k-1}, x_{k-2}, \dots, x_0]$ 。令  $x = B2U_w(\vec{x})$ 。则  $\vec{x}' = x \bmod 2^k$ 。

**推导 4** 截断补码数值

使用无符号数截断相同参数，则有

$$B2U_w([x_{w-1}, x_{w-2}, \dots, x_0]) \bmod 2^k = B2U_k[x_{k-1}, x_{k-2}, \dots, x_0]$$

即， $x \bmod 2^k$  能够被一个位级表示为  $[x_{k-1}, x_{k-2}, \dots, x_0]$  的无符号数表示。将其转换为补码数则有  $x' = U2T_k(x \bmod 2^k)$ 。 ■

总结：

无符号数的截断结果：

$$B2U_k[x_{k-1}, x_{k-2}, \dots, x_0] = B2U_w([x_{w-1}, x_{w-2}, \dots, x_0]) \bmod 2^k$$

补码数字的截断结果：

$$B2T_l[x_{k-1}, x_{k-2}, \dots, x_0] = U2T_k(B2U_w([x_{w-1}, x_{w-2}, \dots, x_0]) \bmod 2^k)$$

### 3.6 关于有符号数和无符号数的建议

有符号数到无符号数的隐式转换，会导致错误或者漏洞。避免这类错误的一种方法是绝不使用无符号数。（例如除 C 语言外，少有语言支持无符号整数。）

但是当我们想要把字仅仅看做是位的集合而没有任何数字意义时，无符号数值是非常有用的。

所以，见机行事。

## 4 整数运算

### 4.1 无符号加法

**原理 10** 无符号数加法

对满足  $0 \leq x, y \leq 2^w$  的  $x$  和  $y$  有：

$$x +_w^u y = \begin{cases} x + y, & x + y < 2^w \\ x + y - 2^w, & 2^w \leq x + y < 2^{w+1} \end{cases} \quad \begin{matrix} \text{Normal} \\ \text{Overflow} \end{matrix}$$

**推导 5** 无符号数加法

一般而言，我们可以看到，如果  $x + y < 2^w$ ，和的  $w + 1$  位表示中最高位会等于 0，因此丢弃它不会改变这个数值。

另一方面，如果  $2^w \leq x + y < 2^{w+1}$ ，和的  $w + 1$  位表示中的最高位会等于 1，因此丢弃它就相当于从和中减去了  $2^w$ 。 ■

形象表示见下图：

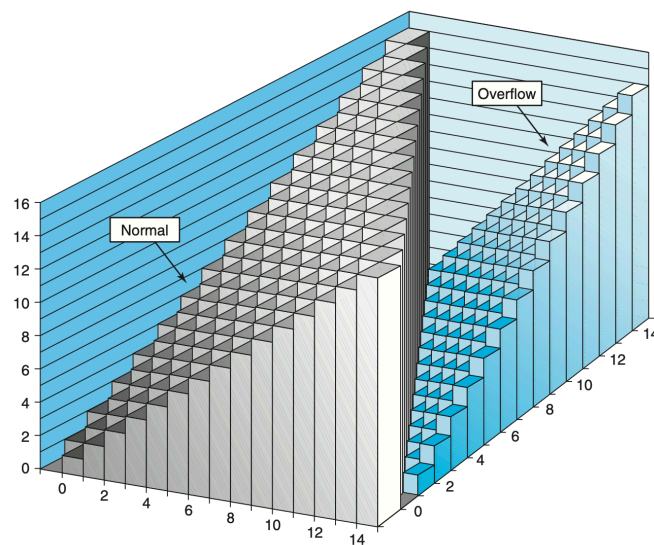


Figure 10: 无符号加法 (4 位字长，加法是模 16 的)

整数加法和无符号加法着急拿的关系见下图：

**原理 11** 检测无符号数加法中的溢出

对在范围  $0 \leq x, y \leq UMax_w$  中的  $x$  和  $y$ ，令  $s \doteq x +_w^u y$ 。则对计算  $s$ ，当且仅当  $s < x$  (或者等价的  $s < y$ ) 时，发生了溢出。

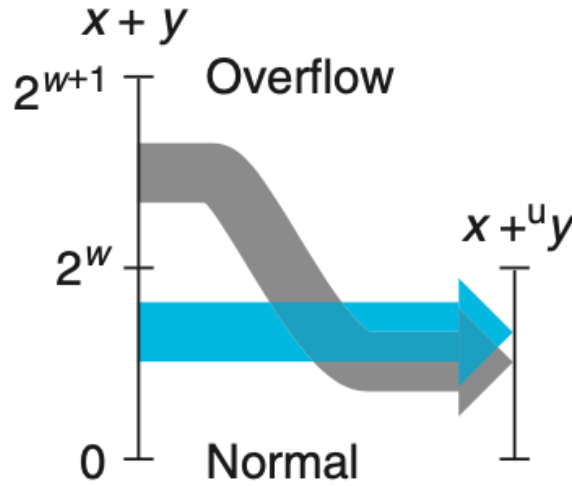


Figure 11: 整数加法和无符号加法间的关系。当  $x+y$  大于  $2^w - 1$  时, 其和溢出

#### 推导 6 检测无符号数加法中的溢出

通过观察发现  $x + y \geq x$ , 因此如果  $s$  没有溢出, 我们能够肯定  $s \geq x$ 。

另一方面, 如果  $s$  确实溢出了, 我们就有  $s = x + y - 2^w$ 。假设  $y < 2^w$ , 我们就有  $y - 2^w < 0$ , 因此  $s = x + (y - 2^w) < x$ 。 ■

模数加法形成了一种数学结构, 称为阿贝尔群 (Abelian group), 它是可交换的和可结合的。它有一个单位元  $0$ , 并且每个元素有一个加法逆元。

#### 原理 12 无符号数求反

对满足  $0 \leq x < 2^w$  的任意  $x$ , 其  $w$  位的无符号逆元  $-^u_w x$  由下式给出:

$$-^u_w x = \begin{cases} x, & x = 0 \\ 2^w - x, & x > 0 \end{cases}$$

#### 推导 7 无符号数求反

当  $x = 0$  时, 加法逆元显然是  $0$ 。对于  $x > 0$ , 考虑值  $2^w - x$ 。我们观察到这个数字在  $0 < 2^w - x < 2^w$  范围之内, 并且  $(x + 2^w - x) \bmod 2^w = 2^w \bmod 2^w = 0$ 。因此, 它就是  $x$  在  $+^u_w$  下的逆元。 ■

## 4.2 补码加法

定义  $x +^t_w y$  为整数和  $x + y$  被截断为  $w$  位的结果, 并将这个结果看做是补码数。

#### 原理 13 补码加法

对满足  $-2^{w-1} \leq x, y \leq 2^{w-1} - 1$  的整数  $x$  和  $y$ , 有:

$$x +^t_w y = \begin{cases} x + y - 2^w, & 2^{w-1} \leq x + y & \text{Positive overflow} \\ x + y, & -2^{w-1} \leq x + y < 2^{w-1} & \text{Normal} \\ x + y + 2^w, & x + y < -2^{w-1} & \text{Negative overflow} \end{cases}$$

### 推导 8 补码加法

由于补码加法和无符号数加法有相同的位级表示，故可以按照如下步骤表示运算  $+_w^t$ ：

1. 将参数转换为无符号数
2. 执行无符号数加法
3. 将结果转换为补码

$$x +_w^t y \doteq U2T_w(T2U_w(x) +_w^u T2U_w(y))$$

由式 5,  $T2U_w(x) \iff x_{w-1}2^w + x$ ,  $T2U_w(y) \iff y_{w-1}2^w + y$ 。使用属性  $\mathbf{[+}_w^u$  是模  $2^w$  的加法，以及模数加法的属性，我们得到：

$$\begin{aligned} x +_w^t y &= U2T_w(T2U_w(x) +_w^u T2U_w(y)) \\ &= U2T_w[(x_{w-1}2^w + x + y_{w-1}2^w + y) \bmod 2^w] \\ &= U2T_w[(x + y) \bmod 2^w] \end{aligned}$$

定义  $z \doteq x + y$ ,  $z' \doteq z \bmod 2^w$ ,  $z'' \doteq U2T_w(z')$ ,  $z'' = x +_w^t y$ 。下面分 4 种情况讨论：

1.  $-2^w \leq z < -2^{w-1}$ ，则  $z' = z + 2^w$ 。于是得出  $0 \leq z' < -2^{w-1} + 2^w = 2^{w-1}$ 。检查式 6 可以看到  $z'$  在满足  $z'' = z'$  的范围之内。这种情况称作**负溢出 (negative overflow)**。将两个负数  $x$  和  $y$  相加 (这是得到  $z < -2^{w-1}$  的唯一方式)，得到一个非负的结果  $z'' = x + y + 2^w$ 。
2.  $-2^{w-1} \leq z < 0$ ，则  $z' = z + 2^w$ 。于是得出  $-2^{w-1} + 2^w = 2^{w-1} \leq z' < 2^w$ 。检查式 6 可以看到  $z'$  在满足  $z'' = z' - 2^w$  的范围之内。因此  $z'' = z' - 2^w = z + 2^w - 2^w = z$ 。即，补码和  $z''$  等于整数和  $x + y$ 。
3.  $0 \leq z < 2^{w-1}$ ，则  $z' = z$ 。于是得出  $0 \leq z' < 2^{w-1}$ ，因此  $z'' = z' = z$ 。于是补码和  $z''$  又等于整数和  $x + y$ 。
4.  $2^{w-1} \leq z < 2^w$ ，则  $z' = z$ 。于是得出  $2^{w-1} \leq z' < 2^w$ 。在这个范围内， $z'' = z' - 2^w$ 。因此得到  $z'' = x + y - 2^w$ 。这种情况称作**正溢出 (positive overflow)**。将整数  $x$  和  $y$  相加 (这是得到  $z \geq 2^{w-1}$  的唯一方式)，得出一个负数结果  $z'' = x + y - 2^w$ 。 ■

补码加法的形象表示见下图：

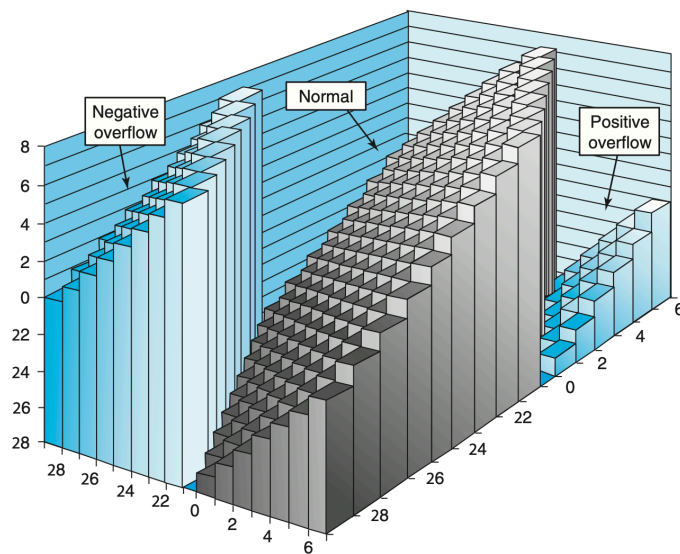


Figure 12: 补码加法 (字长为 4 位的情况下, 当  $x + y < -8$  时, 产生负溢出;  $x + y \geq 8$  时, 产生正溢出)

**原理 14** 检测补码加法中的溢出

对满足  $TMin_w \leq x, y \leq TMax_w$  的  $x$  和  $y$ , 令  $s \doteq x +_w^t y$ 。当且仅当  $x > 0, y > 0$ , 但  $s \leq 0$  时, 计算  $s$  发生了正溢出。当且仅当  $x < 0, y < 0$ , 但  $s \geq 0$  时, 计算  $s$  发生了负溢出。

**推导 9** 检测补码加法中的溢出

1. 分析正溢出: 若  $x > 0, y > 0$ , 而  $s \leq 0$ , 那么显然发生了正溢出。反过来, 正溢出的条件为 1)  $x > 0, y > 0$  (或者  $x + y < TMax_w$ ), 2)  $s \leq 0$ 。
2. 分析负溢出: 若  $x < 0, y < 0$ , 而  $s \geq 0$ , 那么显然发生了负溢出。反过来, 负溢出的条件为 1)  $x < 0, y < 0$  (或者  $x + y > TMin_w$ ), 2)  $s \geq 0$ 。 ■