第5章

数据的机器级表示和运算

现代计算机存储和处理的信息在计算机内部的存储形式都是一致的，均为由0和1组成的字符串，即二进制编码。在计算机中，常用的数据表示格式有两种：一是定点格式，二是浮点格式。所谓定点格式，即在计算机中存储数据的小数点位置是确定的，通常定点数都表示为纯小数和纯整数。纯小数的小数点固定在数的最左边，纯整数的小数点固定在数的最右边，均可省略，因此计算机中可用“纯整数”来表示整数。所谓浮点格式，是指一个数的小数点的位置不是固定的，浮点数利用指数使得小数点的位置可以上下浮动。因此一般来说，浮点格式能表示的数的范围比定点格式大得多，但是其要求的处理硬件相对定点格式则更为复杂。

在本章中，将着重介绍各类数据的表示方法和算术运算方法。

5.1 数据类型及编码方式概述

从外部形式来看，虽然计算机也可以处理十进制数值、图、声音、文字、视频等一些常见的数据，但是这些形式的数据难以直接在计算机内部存储、传输、运算，仅仅是为了从键盘等输入设备输入或者从屏幕等输出设备输出。在计算机内部只有两类基本数据类型：数值数据和非数值数据。数值数据分为整数和实数，用来比较数据大小，整数用定点数表示，实数用浮点数表示；非数值数据是一个位串，没有大小之分，可以用来表示逻辑值、字符等信息。这些信息都需要经过二进制编码才能存储在计算机内部。

数据以二进制的形式存放在计算机内部有许多优点。

(1) 技术上容易实现：二进制只有两种状态0和1，使用双稳态电路可以轻松地表示出二进制的每一位。

(2) 运算规则简单：二进制运算规则简单，易于实现，并且可以用逻辑运算实现算术运算。

(3) 抗干扰能力强，可靠性高：用二进制表示数据时，因为每位数据只有高低两个状态，当受到一定程度的干扰时，仍能可靠地分辨出它是高还是低。

数值数据和非数值数据有不同的编码方式。数值数据有两大类表示方法：一类是直接用二进制数表示，例如，原码、反码、补码；一类是用二进制编码的十进制数表示，例如，ASCII码、BCD码。非数值数据由一个二进制位串表示，其中，西文字符使用最广泛的是ASCII编码方式，中文字符常见的有UTF-8编码方式等。

当确定数值数据的编码方式、进位记数制、定点还是浮点表示之后，其代表的值就可以转化成二进制 0/1 序列存储在计算机中。

5.1.1　数值数据及其编码方式


数据的表示

1. 数值数据类型

数值数据分为整数和实数。整数分为无符号整数和带符号整数。在计算机内部由定点数表示，实数在计算机内部由浮点数表示。

所谓定点数，即小数点的位置约定在固定位置的数。定点数可以表示定点整数和定点小数，定点整数的小数点总是固定在数的右边，定点小数的小数点总是固定在数的左边，所以用定点整数来表示整数，定点小数来表示浮点数的尾数部分。

定点数表达方式直观，但是表示范围较小，不适合表达非常大或者非常小的数。最终，绝大多数现代的计算机系统采纳了浮点数表达方式。所谓浮点数，即小数点位置约定可变的数。这种表达方式利用科学记数法来表达实数，即用一个尾数、一个基数、一个指数以及一个表示正负的符号来表达实数。

例如，对于十进制数 $X=111$，可表示为以下形式。

$$\begin{aligned} X &= 0.1101111 \times 2^7 \\ &= 0.01101111 \times 2^8 \\ &= 0.001101111 \times 2^9 \end{aligned}$$

2. 数值数据的编码方式

1）定点数编码表示

定点数编码有原码、反码、补码、移码四种表示方式。

2）浮点数编码表示

目前几乎所有的计算机都采用 IEEE 754 标准来表示浮点数。

3）十进制数的表示

计算机中的数都是由二进制表示和计算的，对于十进制，可以通过数值进制公式进行转换后计算和显示。但除了这种编码方式之外，计算机为了显示和计算方便，对十进制还定义了新的编码方式：ASCII 码字符表示、BCD 码表示。

ASCII 码将十进制数看成由 0～9 字符组成的字符串，每一个字符占一个字节，也就是 8 位二进制数，0 和 9 分别对应 ASCII 码中的 30H 和 39H。

例如，十进制数 90 经过 ASCII 编码可表示为 3930H。

如果只是对十进制数进行打印或者显示，用 ACSII 码非常方便，但是这种形式的十进制数计算起来非常不便，因为高 4 位编码含有非数值信息，必须要转换为二进制数或者 BCD 码才能计算。

BCD 码用 4 位二进制数来表示十进制数。十进制 0～9 共 10 个数字，而 4 位二进制位可以组合出 16 种状态，所以从 16 种状态中选取 10 种状态就可以表示十进制数，并且可以产生多种 BCD 码。BCD 码分为有权 BCD 码和无权 BCD 码。

有权 BCD 码指表示十进制数的 4 位二进制数的每一位都有一个确定的权。其中最常

用的编码是 8421 码,它选取 4 位二进制数并按顺序取前 10 个代码与十进制数字对应,每位权从左到右为 8、4、2、1,因此称为 8421 码。

例如,十进制数 321 的 8421 码为 0011 0010 0001B。

与有权码相对应,无权码的每个十进制位没有确定的权。无权码方案中用的较多的是余 3 码和格雷码。余 3 码是在 8421 码的基础上,把每个代码都加 0011 形成的,优点是执行十进制数加法时能正确进位,且为减法运算提供便利。格雷码规则是任何两个相邻的代码只有一个二进制位的状态不同,其余 3 个二进制必须相同,这样设计的好处在于从一个编码变到下一个编码时,只有一位发生变化,编码速度最快,利于电路的设计和运行。余 3 码和格雷码的编码方案如表 5-1 所示。

表 5-1 余 3 码与格雷码编码方案

十进制数	余 3 码	格雷码 1	格雷码 2
0	0011	0000	0000
1	0100	0001	0100
2	0101	0011	0110
3	0110	0010	0010
4	0111	0110	1010
5	1000	1110	1011
6	1001	1010	0011
7	1010	1000	0001
8	1011	1100	1001
9	1100	0100	1000

5.1.2 非数值数据及其编码方式

非数值数据的表示

非数值数据类型包括逻辑值、字符等数据,在计算机内部由一个二进制位串表示。逻辑值是一串 0/1 序列,在形式上和二进制数并无差别,只是代表含义不同。例如,逻辑值可以表示逻辑表达式中逻辑值的真假,真为 1,假为 0。字符如拉丁字母、数字、标点符号、汉字等,不能直接在计算机内部进行处理,也需要对其进行二进制编码。所有字符的集合称为字符集,字符集中每一个字符都有一串二进制代码与其对应,所有字符对应的二进制代码集合则称为码表。当确定字符集和编码之后,外部字符和内部的二进制串就有了一一对应的关系。

1. 西文字符编码方式

计算机是由美国人发明的,美国人为了让计算机能够表示字母、标点符号等字符,设计了字符编码。英文字符数量少,不超过 127 种,用 7 位二进制数就足以表示所有的西文字

符。目前，计算机中使用最广泛的字符集及其编码是 ASCII 码。

如表 5-2 所示，每一个字符都由一个 8 位的二进制串表示，最高位为 b_7，通常为 0，也可以用来奇偶校验，最低位为 b_0。ASCII 编码有以下两个规律。

(1) 便于与十进制数转换。字符 0～9 在 ASCII 码中，$b_6b_5b_4$ 高三位编码都是 011，而低四位 $b_3b_2b_1b_0$ 分别对应 0～9 的 8421 码，转换为十进制数非常方便。

(2) 便于大小写字母的转换。大写字母与小写字母的差别仅在 b_5 这一位上，若为 1，则为小写字母，反之为大写字母。仅需要更改一位就可以完成大小写字母的转换。

表 5-2　ASCII 码表

	$b_6b_5b_4=000$	$b_6b_5b_4=001$	$b_6b_5b_4=010$	$b_6b_5b_4=011$	$b_6b_5b_4=100$	$b_6b_5b_4=101$	$b_6b_5b_4=110$	$b_6b_5b_4=111$
$b_3b_2b_1b_0=0000$	NUL	DLE	SP	0	@	P	`	p
$b_3b_2b_1b_0=0001$	SOH	DC1	!	1	A	Q	a	q
$b_3b_2b_1b_0=0010$	STX	DC2	"	2	B	R	b	r
$b_3b_2b_1b_0=0011$	ETX	DC3	#	3	C	S	c	s
$b_3b_2b_1b_0=0100$	EOT	DC4	$	4	D	T	d	t
$b_3b_2b_1b_0=0101$	ENQ	NAK	%	5	E	U	e	u
$b_3b_2b_1b_0=0110$	ACK	SYN	&	6	F	V	f	v
$b_3b_2b_1b_0=0111$	BEL	ETB	'	7	G	W	g	w
$b_3b_2b_1b_0=1000$	BS	CAN	(	8	H	X	h	x
$b_3b_2b_1b_0=1001$	HT	EM	)	9	I	Y	i	y
$b_3b_2b_1b_0=1010$	LF	SUB	*	:	J	Z	j	z
$b_3b_2b_1b_0=1011$	VT	ESC	+	;	K	[	k	{
$b_3b_2b_1b_0=1100$	FF	FS	,	<	L	\	l	|
$b_3b_2b_1b_0=1101$	CR	GS	-	=	M	]	m	
$b_3b_2b_1b_0=1110$	SO	RS	.	>	N	^	n	~
$b_3b_2b_1b_0=1111$	SI	US	/	?	O	_	o	DEL

2. 汉字的编码方式

汉字总共有 6 万多个,数量巨大,只有 8 位的 ASCII 码明显不能满足编码汉字的需求。要让汉字能够在计算机内部表示、传输、交换,有以下三个难点。

(1) 输入困难。用键盘输入西文字符非常方便,一个或者两个西文字符对应于键盘上的一个按键。如果也让一个汉字对应一个按键,那是不可能的事情。必须通过按键来对汉字进行编码。

(2) 编码复杂。第一,汉字数量巨大,ASCII 码不能满足存储汉字的需要,必须设计一种全新的编码方式;第二,新的编码方式不能与 ASCII 码混淆。

(3) 输出困难。要使汉字在屏幕上显示或者打印,必须把汉字用人们可以阅读的方块字的形式表现出来,也就是存放汉字的字形。

所以,要实现计算机对汉字信息的处理,汉字系统必须处理以下几种汉字代码:输入码、内码、字模点阵码。

5.1.3 进位记数制

1. 进制数的表示

在日常生活中,人们常使用十进制表示数据,而在计算机系统中则采用二进制来表示数据。事实上,任意一个数都可用下式表示:

$$\begin{aligned} N &= (d_{n-1}d_{n-2}\cdots d_1 d_0 . d_{-1}\cdots d_{-m})_r \\ &= d_{n-1}\,r^{n-1} + d_{n-2}\,r^{n-2} + \cdots + d_1\,r^1 + d_0\,r^0 + d_{-1}\,r^{-1} + \cdots + d_{-m}\,r^{-m} \\ &= \sum_{i=-m}^{n-1} d_i\,r^i \end{aligned} \tag{5.1}$$

其中,r 被称为基值;n、m 是代表整数位和小数位位数的正整数;d_i 为系数,可以是 0～$r-1$ 数码中的任意一个,是用来代表第 i 位的数码;r^i 则为第 i 位的权重。

因此,任意一个十进制数 $D=d_{n-1}d_{n-2}\cdots d_1 d_0 . d_{-1}\cdots d_{-m}$ 都可以表示成如下形式。

$$\begin{aligned} N(D) = {} & d_{n-1}\,10^{n-1} + d_{n-2}\,10^{n-2} + \cdots + d_1\,10^1 + d_0\,10^0 + \\ & d_{-1}\,10^{-1} + \cdots + d_{-m}\,10^{-m} \end{aligned} \tag{5.2}$$

其中,系数 $d_i(i=n-1,n-2,\cdots 1,0,-1,\cdots,-m)$ 可以是 0,1,2,3,4,5,6,7,8,9,10 这 10 个数字符号中的任意一个,基值 $r=10$。以十进制数 1563.21 为例,其代表的值为:

$$(1563.21)_{10} = 1\times 10^3 + 5\times 10^2 + 6\times 10^1 + 3\times 10^0 + 2\times 10^{-1} + 1\times 10^{-2}$$

在上式中,10^i 为第 i 位上的权。在进行十进制数运算时,采用“逢十进一”的计算规则,即每位计满十之后就要向高位进一。

类似地,任意一个二进制数 $B=d_{n-1}d_{n-2}\cdots d_1 d_0 . d_{-1}\cdots d_{-m}$ 均可表示为如下形式。

$$N(B) = d_{n-1}2^{n-1} + d_{n-2}2^{n-2} + \cdots d_1 2^1 + d_0 2^0 + d_{-1}2^{-1} + \cdots + d_{-m}2^{-m} \tag{5.3}$$

与十进制不同的是,二进制的基值是 2,系数 d_i 只可取 0 和 1,运算时采用“逢二进一”的运算法则。例如,二进制数 $(1101.01)_2$ 代表的值是:

$$(1101.01)_2 = 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} = (13.25)_{10}$$

扩展到一般情况,在 R 进制的数字系统中,系数 d_i 的取值范围应该为 0～$R-1$,采用

“逢 R 进一”的进位规则。如表 5-3 所示为常用的二进制、八进制、十进制、十六进制使用的进位规则、系数取值与表示符号。

表 5-3　不同进制的进位规则、基本符号与表示

进　制	二　进　制	八　进　制	十　进　制	十 六 进 制
进位规则	逢二进一	逢十进一	逢八进一	逢十六进一
系数取值	0,1	0,1,2,…,7	0,1,2,…,9	0,1,2,…,A,B,C,D,E,F
表示符号	B(binary)	O(octal)	D(decimal)	H(hexadecimal)

表 5-4 则给出了二进制、八进制、十进制与十六进制数的对照。

表 5-4　四种进制数的对照

二进制数	八进制数	十进制数	十六进制数	二进制数	八进制数	十进制数	十六进制数
0000	0	0	0	1000	10	8	8
0001	1	1	1	1001	11	9	9
0010	2	2	2	1010	12	10	A
0011	3	3	3	1011	13	11	B
0100	4	4	4	1100	14	12	C
0101	5	5	5	1101	15	13	D
0110	6	6	6	1110	16	14	E
0111	7	7	7	1111	17	15	F

虽然在计算机内部，所有的信息都是采用二进制编码表示，但是在日常生活中，人们通常使用八进制、十进制或十六进制等数据表示方式。因此，计算机在数据输入前和输入后都必须进行进制间的转换。以下将介绍各进位记数制之间的转换。

2. 进制数的转换

1）R 进制数转换成十进制数

在进行 R 进制数转换成十进制数时采用“按权展开”即可。例如：

$(1110.01)_2 = 1\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0 + 0\times2^{-1} + 1\times2^{-2} = (14.25)_{10}$

$(215.4)_8 = 2\times8^2 + 1\times8^1 + 5\times8^0 + 4\times8^{-1} = (141.5)_{10}$

$(C5F.C)_{16} = 12\times16^2 + 5\times16^1 + 15\times16^0 + 12\times16^{-1} = (3167.75)_{10}$

2）十进制数转换成 R 进制数

在进行十进制数转换成 R 进制数时，要将整数部分和小数部分分别转换。本书主要采用“重复相除(乘)”的方法，这种方法的规则是在进行整数部分的转换时，整数部分除以 R 取余数，直到商为 0 为止；在进行小数部分的转换时，小数部分乘以 R 取整数，直到小数部分为 0(或按照精度要求确定位数)。

(1) 整数部分的转换。

从上文中可知，针对整数部分的转换采取除以 R 取余数，直到商为 0 为止的转换方式。

先得到的余数作为右边低位上的数位,后得到的余数作为左边高位上的数位。

例 5.1 以十进制转二进制为例,将十进制数 215 转换成二进制数。

解: 将 215 处以 2,将每次的余数按照从低位到高位的顺序排列。

重复除以 2	得商	取余数	
215÷2	107	1	最低位
107÷2	53	1	
53÷2	26	1	
26÷2	13	0	
13÷2	6	1	
6÷2	3	0	
3÷2	1	1	
1÷2	0	1	最高位

所以,$(215)_{10}=(11010111)_2$。

(2) 小数部分的转换。

从上文中可知,针对小数部分的转换采取乘以 R 取整数,直到小数部分为 0 为止的转换方式。最后,将上面的整数部分作为左边高位上的数位,下面的整数部分作为右边低位上的位数。

例 5.2 以十进制转二进制为例,将十进制小数 0.8125 转换成二进制数。

解: 将 0.8125 乘以 2,将每次结果的整数部分按照从高位到低位排序。

重复乘以 2	得小数部分	取整数	
0.8125×2	0.6250	1	最高位
0.6250×2	0.2500	1	
0.2500×2	0.5000	0	
0.5000×2	0.0000	1	最低位

所以,$(0.8125)_{10}=(0.1101)_2$。

当然,在转换过程中,可能乘积的小数部分总得不到 0,这种情况下得到的是近似值。

例 5.3 以十进制转二进制为例,将十进制小数 0.624 转换成二进制数。

解: 将 0.624 乘以 2,将每次结果的整数部分按照从高位到低位排序。

重复乘以 2	得小数部分	取整数	
0.624×2	0.248	1	最高位
0.248×2	0.496	0	
0.496×2	0.992	0	
0.992×2	0.984	1	最低位

所以,$(0.624)_{10}=(0.1001\cdots)_2$。

(3) 含整数和小数部分的十进制数转换成 R 进制数。

只要将十进制数的整数部分与小数部分分别进行转换,再将转换后的 R 进制整数与小数组合起来,便可以得到一个完整的 R 进制数。

例 5.4 以十进制转二进制为例,将十进制数 215.8125 转换为二进制数。

解: 由例 5.1 和例 5.2 可知:

$$(215)_{10}=(11010111)_2$$

$$(0.8125)_{10}=(0.1101)_2$$

因此，只需将二者结合起来就可，即：

$$(215.8125)_{10}=(11010111.1101)_2$$

3）二进制数、八进制数与十六进制数的相互转换

（1）八进制数转换为二进制数。

在进行八进制转换成二进制时，需要按从高位到低位的顺序，将每一个八进制数字转换为对应的 3 位二进制数字。八进制数字与二进制数字的对应关系如下。

$$(0)_8=(000)_2 \qquad (1)_8=(001)_2$$

$$(2)_8=(010)_2 \qquad (3)_8=(011)_2$$

$$(4)_8=(100)_2 \qquad (5)_8=(101)_2$$

$$(6)_8=(110)_2 \qquad (7)_8=(111)_2$$

（2）二进制数转换为八进制数。

在进行二进制转换为八进制数时，需要将二进制的整数部分按照从低位到高位的顺序，每 3 位二进制数字用对应的 1 位八进制数字来替换。如果最后高位不足 3 位时，则需要采取高位补 0 的方式来凑满 3 位。而针对二进制小数部分，则按照从高位到低位的顺序，每 3 位二进制数字用对应的 1 位八进制数字来替换。如果最后低位不足 3 位时，则需要采取低位补 0 的方式来凑满三位。例如：

$$(111001.11)_2=(71.6)_8。$$

（3）十六进制转换为二进制。

类似于八进制转换为二进制的方法，只需将每一个十六进制数字按照高低位次序改写成等值的 4 位二进制即可，十六进制数字与二进制数字的对应关系如下。

$$(0)_{16}=(0000)_2 \qquad (1)_{16}=(0001)_2$$

$$(2)_{16}=(0010)_2 \qquad (3)_{16}=(0011)_2$$

$$(4)_{16}=(0100)_2 \qquad (5)_{16}=(0101)_2$$

$$(6)_{16}=(0110)_2 \qquad (7)_{16}=(0111)_2$$

$$(8)_{16}=(1000)_2 \qquad (9)_{16}=(1001)_2$$

$$(A)_{16}=(1010)_2 \qquad (B)_{16}=(1011)_2$$

$$(C)_{16}=(1100)_2 \qquad (D)_{16}=(1101)_2$$

$$(E)_{16}=(1110)_2 \qquad (F)_{16}=(1111)_2$$

（4）二进制转换为十六进制。

二进制数转换为十六进制数与二进制数转换为八进制数的规则类似。将二进制的整数部分按照从低位到高位的顺序，每 4 位二进制数字用对应的 1 位十六进制数字来替换。如果最后高位不足 4 位时，则需要采取高位补 0 的方式来凑满 4 位。而针对二进制小数部分，则按照从高位到低位的顺序，每 4 位二进制数字用对应的 1 位十六进制数字来替换。如果最后低位不足 4 位时，则需要采取低位补 0 的方式来凑满 4 位。例如：

$$(11101001.11)_2=(E9.C)_{16}$$

5.2 整数的表示


定点整数的表示

如上所述,纯整数的小数点固定在数的最右侧,因此用“定点纯整数”来表示整数,计算机中的整数可分为无符号整数和有符号整数。本节将重点讨论整数的表示方法,本节所提及的无符号数和有符号数均为整数而不涉及小数。

在数据计算阶段,计算机中的立即数均存放在寄存器中,通常称寄存器存储数字的位数为机器字长。有符号数和无符号数的区别,就在于寄存器中是否有位置存放符号。无符号数,即没有符号的数,机器字长的全部二进制位均用来表示数值位。相对的,有符号数,需要寄存器留出一位存放数值的符号。因此,无符号数只能表示非负数,而有符号数能够表示负数、零和正数。

5.2.1 无符号数编码

假设有一个整数数据有 n 位,可用 x 表示或者写成 $x_{n-1}x_{n-2}\cdots x_1x_0$ 形式。将 x 看作一个二进制表示的数,就获得了 x 的无符号编码方法,在这种编码方式中,每个 x_i 都取值为 0 或 1。

对于二进制数 $x=x_{n-1}x_{n-2}\cdots x_1x_0$,令 $[x]_D$ 为 x 的十进制整数表示,则有

$$[x]_D=\sum_{i=0}^{n-1}x_i2^i \tag{5.4}$$

例如:

$$\begin{cases}[0001]_D=0\times2^3+0\times2^2+0\times2^1+1\times2^0=0+0+0+1=1\\ [0101]_D=0\times2^3+1\times2^2+0\times2^1+1\times2^0=0+4+0+1=5\\ [1001]_D=1\times2^3+0\times2^2+0\times2^1+1\times2^0=8+0+0+1=9\\ [1111]_D=1\times2^3+1\times2^2+1\times2^1+1\times2^0=8+4+2+1=15\end{cases} \tag{5.5}$$

无符号数编码方式简单易懂,对于 n 位的无符号数,其能代表的数值范围很好计算,最小值为 0,最大值为 2^n-1。例如,对于 16 位机器字长的无符号数,其能表示的最大数据范围是 0～65 535。

5.2.2 有符号数编码

1. 机器数和真值

正如上面所提,无符号数只能表示非负数,但是计算机的运算过程中,不可避免地会遇到负数运算的问题,由于机器只能识别 0 和 1,不能直接识别“正”和“负”,因此可以用 0 表示“正”,用 1 表示“负”,这样“正”“负”符号就被表示成计算机可以识别的数字,规定在计算机中用一个数的最高位存放符号,有效数字存放在符号位之后,这样就组成了有符号数。

例如,对于有符号数+10011,在机器中可以表示为 010011,而有符号数-10011,在机器中可以表示为 110011。一个有符号数在计算机中的二进制表示形式叫作这个数的机器数,将机器数所对应的真正的数值叫作真值。例如,上述 010011 和 110011 为机器数,而其真正表示的带“+”“-”符号的数为真值。

一旦将真值转换为机器数，即将“+”“−”号数字化，符号和数值就会形成新的编码，下面将介绍机器数的 4 种编码方式，分别是原码、补码、反码和移码。

2. 原码表示法

原码表示是机器数最简单的一种表示形式，一个数的原码表示直接由符号位和数值位构成，符号位 0 代表正数，1 代表负数，数值位即为真值的绝对值。

对于整数 x，其原码表示的定义为

$$[x]_{原}=\begin{cases}0,x, & 2^n>x\geqslant 0\\ 2^n-x, & 0\geqslant x>-2^n\end{cases} \tag{5.6}$$

式中，x 为真值，n 为整数的位数。

例如，$x=+10010$，则$[x]_{原}=010010$。$x=-10010$，则$[x]_{原}=110010$。

一般情况下，对于正数 $x=+x_nx_{n-1}\cdots,x_1x_0$，则有

$$[x]_{原}=0x_{n-1}x_{n-2}\cdots x_1x_0 \tag{5.7}$$

对于负数 $x=-x_nx_{n-1}\cdots,x_1x_0$，则有

$$[x]_{原}=1x_{n-1}x_{n-2}\cdots x_1x_0 \tag{5.8}$$

当 $x=0$ 时，则有

$$\begin{cases}[+0]_{原}=0000000\\ [-0]_{原}=1000000\end{cases} \tag{5.9}$$

原码表示法十分简单易懂，即符号位加上真值绝对值的二进制数，与真值的对应关系直观。但是原码的缺陷在于 0 的表示不唯一，给使用带来了不便，更重要的是加减法的操作十分复杂，当进行加法运算时，首先要判断两数是否同号，同号则相加，异号则相减。进行减法运算时，先要比较两个数的绝对值大小，然后使用绝对值大的数减去绝对值小的数，最后还要赋予结果正确的符号，这种运算过程十分复杂费时。为了解决这些问题，人们提出了补码表示法。

3. 补码表示法

1）模运算

在学习补码之前，首先以时钟为例，了解一下模运算和补数的概念。

在模运算系统中，若 A、B、M 满足 $A=B+K\times M$（K 为整数），则记为 $A=B(\mathrm{mod}\ M)$。即 A、B 各除以 M 后的余数相同，称为 A 和 B 模 M 同余。假如现在时钟指向 9 点，要将它拨向 4 点，最简单的则有两种拨法：一是逆时针拨 5 格；二是顺时针拨 7 格。这两种方法最终的结果是一样的，时钟的指针都会指向 6 点。令顺时针为正，逆时针为负，则容易得到：

$$9-5=4$$

$$9+7=16\equiv 4(\mathrm{mod}\ 12)$$

由于时钟转一圈为 12h，一旦时间超过了 12 点，“12”就会自动丢失而不被显示，即 16−12=4，因此，16 点和 4 点在时钟中均显示为 4 点。也就是说，在时钟系统中，−5 和+7 是等价的。在数学上称 12 为模，写作 mod 12，称+7 是−5 以 12 为模的补数，即

$$-5\equiv 7(\mathrm{mod}\ 12)$$