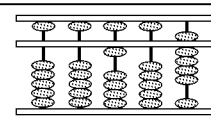
第 3 章

机器数与编码

3.1 机器数

数在计算机中的表示形式称为“机器数”(Machine Data),它所表示的实际数值称为“真值”(Truth Value),表 3.1 举例说明了机器数与真值之间的对应关系。

表 3.1 机器数与真值之间的对应关系

运算工具	机器数表示形式	对应的真值
算盘		109
计算机	位 7 6 5 4 3 2 1 0 0 1 1 0 1 1 0 1	109 或 1101101B

由于计算机是由二进制数字电路构成的,因此,它的机器数只能由二进制数码 0 和 1 构成。

3.1.1 字和小数点

1. 字

“字”(Word)是计算机中每次作为一个整体进行处理的二进制数码(参见图 3.1),字的位数称为“字长”(Word Length),字长标志着机器每次运算的最大精度。

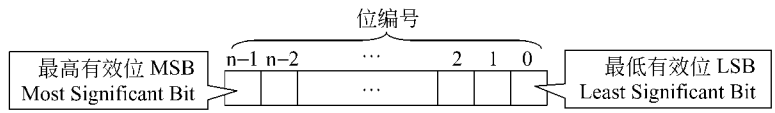


图 3.1 字长为 n 位的字

字节(Byte)是信息量的基本计量单位,一个字节可容纳 8 位二进制数码,因此,字长为 16、32 和 64 位的字,分别可容纳 2、4 和 8 个字节。

由排列组合原理可知,一个字长为n位的字,最多能够表示出2^n种不同的二进制数

码。其中，数值 2^n 被称为该字的“模”(Model)。

2. 小数点

在机器数中，小数点的位置是隐含指定的。小数点处理方式有以下两种：

(1) 定点(Fixed Point)数，在运算过程中小数点的位置始终固定不变，这与算盘中小数点的表示法类似。例如：

纯整数形式　XXXXXXXXXXXXXXXX.

纯小数形式　.XXXXXXXXXXXXXXXX

整数和小数形式　XXXXXXXXXX.XXXXXX

(2) 浮点(Floating Point)数，在运算过程中小数点的位置可以根据实际需要来改变，但不能改变数值的大小。例如：

$$124.5=1.245\times10^2=0.1245\times10^3=\cdots$$

可见，虽然浮点数中小数点的位置不同，但数值大小是不变的。

浮点数主要包括尾数(Mantissa)、指数(Exponent)和底数(Base)三个部分。

$$\underbrace{0.1245}_{\text{尾数}}\times\underbrace{10^{3}}_{\text{底数}}\quad\text{指数}$$

其中，尾数是数值的有效部分，尾数中的小数点只是形式上的，它在数值中的实际位置是由指数来确定的。底数值是固定不变的，一般在计算机中省略不写。

3.1.2　机器数表示法

在计算机中，通常把描述机器数正负的符号标记放在一个字的最高有效位 MSB 上。存放符号标记的位称为符号位。下面将以定点整数为例介绍有符号机器数的表示方法。

1. 原码

计算机是无法直接识别“+”(正)、“−”(负)符号的，如果在机器数中分别用“0”和“1”来代表“+”和“−”，那么这种有符号机器数的表示形式就称为“原码”(Primitive Encoding)，记为$[x]_{原}$。其中 x 为真值，[x]表示真值为 x 的机器数，下标“原”表示该机器数为原码表示形式。

【例 3.1】 设字长为 8，二进制数+10110 和−10110 的原码分别为

$$[+10110]_{原}=00010110$$

$$[-10110]_{原}=10010110$$

一般地，字长为 n 的原码可定义为

$$[x]_{原}=\begin{cases}x & 0\leqslant x<2^{n-1}\\ (10^{n-1}-x)_2 & -2^{n-1}<x<0\end{cases}$$

其中，真值 x(整数)的表示范围是$-2^{n-1}<x<2^{n-1}$。圆括号右侧的下标 2 表示其中的内容为二进制数。

原码表示简单易懂，它与真值之间的转换也非常方便。但计算机用原码进行加减运算的过程与手工计算类似，比较麻烦。例如，两个数相减，先要根据它们的绝对值大小来决定是否要交换位置相减，减的结果还要加上正确的符号。

2. 补码

为了简化加、减运算过程,提出了补码的概念。在介绍补码之前,先来做一个二进制计数的实验(参见表3.2)。

表3.2 计数实验

计数方向	计数器内容(机器数)	计数值(真值)		说明
		二进制	十进制	
加1 ⇧	0111	111	7	最大正数
	0110	110	6	
	0101	101	5	
	0100	100	4	
	0011	11	3	
	0010	10	2	
	0001	1	1	
起点	0000	0	0	初值
⇩ 减1	1111	−1	−1	0000−1≡10000−1
	1110	−10	−2	
	1101	−11	−3	
	1100	−100	−4	
	1011	−101	−5	
	1010	−110	−6	
	1001	−111	−7	
	1000	−1000	−8	最小负数

设有一个4位二进制加/减计数器,从计数器的初值0000开始,分别进行加1和减1计数。由表中可见,对于计数值为0～7的正数来说,计数器的最高有效位为0;而对于小于0但不小于−8的计数值来说,计数器的最高有效位为1,因此,在一定的计数范围内计数器最高有效位的状态可以用来表示数的正负。

注意计数器在0000状态下减1的情况,由于不够减,实际上计数器已向高位发出了借位信号,这时计数器的减1结果与10000−1结果是等价的。因此,当计数值x小于0时,计数器的内容(机器数)实际上已变成如下形式:

$$(10000+x)_2 \quad (-8\leqslant x<0)$$

其中,10000B=2^4就是4位计数器的"模"。把这种表示形式称为机器数的"补码"(Complement),它被定义为

$$[x]_{补}=\begin{cases} x & 0\leqslant x<2^{n-1} \\ (10^n+x)_2 & -2^{n-1}\leqslant x<0 \end{cases}$$

其中，n 为字长，真值 x(整数)的表示范围是：$-2^{n-1} \leqslant x < 2^{n-1}$，常数$(10^n)_2 = 2^n$ 是字的模。

【例 3.2】 设字长为 8，根据上面的补码定义，可分别写出如下二进制数＋10110 和－10110 的补码：

$$[+10110]_{补} = 00010110$$
$$[-10110]_{补} = 100000000 - 10110$$
$$= 11101010$$

由补码的定义可知，两个互为相反数的补码满足以下关系：

$$[x]_{补} + [-x]_{补} = 10^n B \quad (-2^{n-1} < x < 2^{n-1}, n \text{ 为字长})$$

这表明$[x]_{补}$与$[-x]_{补}$相对模 $10^n B$(即 2^n)是互补的，补码一词就是由此而来。利用补码表示的机器数可简化加减运算过程。

【例 3.3】 利用补码概念完成二进制数的减法运算：110－11。

由于 110－11＝110＋(－11)，将这两个数写成 4 位补码为

$$[110]_{补} = 0110, \quad [-11]_{补} = 1101$$

补码计算：		手工计算：
	0110	110
	＋ 1101	－ 11
进位舍弃	1 0011	11

【例 3.4】 利用补码概念完成二进制数的减法运算：100－111。

由于 100－111＝100＋(－111)，将这两个数写成 4 位补码为

$$[100]_{补} = 0100, \quad [-111]_{补} = 1001$$

补码计算：	手工计算：	
0100	111	(被减数小于减数，
＋ 1001	－ 100	颠倒位置相减)
1101	－11	(结果加上负号)

注意：补码运算结果仍为补码，由于补码运算结果的符号位是“1”，表明它是一个负数，它所对应的真值从前面的表 3.2 中可查得为－11，这与手工计算结果是一致的。

从上面的例子可见，用补码表示的机器数不但可以省去减法运算，而且在加法运算过程中不必考虑两个数的大小和正负，大大简化了加减运算过程。

3. 反码

虽然补码可以省去减法运算，但根据补码定义求补码仍需要做减运算，如例 3.2。为了避免求补码过程中的减法运算，提出了反码(Invert Encoding)的概念。

反码的定义与补码的定义类似：

$$[x]_{反} = \begin{cases} x & 0 \leqslant x < 2^{n-1} \\ (10^n - 1 + x)_2 & -2^{n-1} < x < 0 \end{cases}$$

其中，n 为字长，真值 x(整数)的表示范围是$-2^{n-1} < x < 2^{n-1}$。

【例 3.5】 设字长为8,根据上面的反码定义,可分别写出如下二进制数+10110和-10110的反码:

$$[+10110]_{反}=00010110$$
$$[-10110]_{反}=100000000-1-10110$$
$$=11111111-10110$$
$$=11101001$$

注意:反码定义式中

$$10^n-1=11\cdots1B$$

的情况。这样一个n位全为"1"的数与负数x相加的结果就是对负数x逐位取反。这个特点也是反码一词的由来。

【例 3.6】 设字长为8,对二进制数-10110逐位取反(按字长指定的位数操作,不足位用0补足)的结果是:

真值: -00010110
反码: 11101001

可见,它与通过反码定义得到的结果是一致的(参见例3.5)。

对照补码和反码定义可知,负数的反码是由补码减1得到的。因此,可以得到以下重要的关系:

$$[x]_{补}=[x]_{反}+1 \quad (-2^{n-1}<x<0)$$

由于在计算机中对二进制数的取反操作是十分简单的,所以,若要得到一个数的补码可先取其反码,然后再对反码执行加1运算就可以了。

【例 3.7】 通过二进制数-10110的反码求它的补码。

$$[-10110]_{补}=[-10110]_{反}+1$$
$$=11101001+1$$
$$=11101010$$

根据上面补码与反码之间的关系,还可以得到一种直接通过二进制数取负数补码的方法,即从二进制数的最低有效位(LSB)开始起向左,直到遇见第一个"1"为止的所有数字都不变(包括这个"1"),第一个"1"以后各位数字全部取反,直到满足相应字长要求为止。例如:

二进制数 -0001011 0
补码 11101010

3.1.3 负数补码的真值

负数补码与其真值之间的对应关系不像原码那样直观、简单。那么,如何才能知道负数补码所对应的真值呢?

由前面介绍的两个互为相反数的补码关系 $[x]_{补}+[-x]_{补}=2^n$ 和补码的定义,可以得到:

$$[x]_{补}=2^n-[-x]_{补}=[-[-x]_{补}]_{补}$$
$$[-x]_{补}=2^n-[x]_{补}=[-[x]_{补}]_{补}$$

它们反映了真值取负与对应机器数取补之间的内在关系。例如：

真值		补码
$[+11010B]_{补}$	=	00011010B
取负 ↕		取补 ↕（取反加 1）
$[-11010B]_{补}$	=	11100110B
取负 ↕		取补 ↕（取反加 1）
$[+11010B]_{补}$	=	00011010B

从中可见，真值取负与对应的机器数取补是等价的。因此，只要对负数 x 的补码再取一次补，就能得到该负数相反数－x(正数)的补码$[-x]_{补}$，由此补码可以直接写出－x 的取值，进而得到负数 x 的真值。

【例 3.8】 已知机器数$[x]_{补}=11100111B$，求 x。

由于这个补码的符号位(MSB)为 1，它的真值 x 小于 0。对这个补码再取补，即可得到真值相反数－x 的补码：

$[x]_{补}$	=	11100111B
取负 ↓		取补 ↓（取反加 1）
$[-x]_{补}$	=	00011001B

由$[-x]_{补}$可以直接写出－x 的取值 11001B，进而得到 x 的取值－11001B。

3.1.4　补码加减运算

1. 补码加法

补码加法公式为

$$[x+y]_{补}=[x]_{补}+[y]_{补}\quad (\mathrm{mod}\ 2^n)$$

其中，$-2^{n-1}\leqslant x、y、x+y<2^{n-1}$，x 和 y 为整数，n 为字长，mod 为取余运算，$\mathrm{mod}\ 2^n$ 表示加法运算结果只取低 n 位数，超过部分舍去。

下面分 4 种情况来举例说明上面的补码加法公式。

(1) 当 $x\geqslant 0, y\geqslant 0$ 时

因为

$$[x]_{补}=x,\quad [y]_{补}=y,\quad x+y\geqslant 0$$

所以

$$[x+y]_{补}=x+y=[x]_{补}+[y]_{补}$$

【例 3.9】 设字长为 8，二进制数 x＝10111，y＝10010，求 x＋y。

$$[x]_{补}=00010111,\quad [y]_{补}=00010010$$

$$[x+y]_{补}=00010111+00010010$$

$$=00101001$$

$$\begin{array}{r} 00010111 \\ +\ 00010010 \\ \hline 00101001 \end{array}$$

由于符号位为0,结果为正,所以 x+y=101001。

(2) 当 x≥0,y<0 且 x+y<0 时

因为

$$[x]_{补}=x,\quad [y]_{补}=(10^n+y)_2,\quad x+y<0$$

所以

$$\begin{aligned}[x+y]_{补}&=(10^n+x+y)_2\\&=x+(10^n+y)_2\\&=[x]_{补}+[y]_{补}\end{aligned}$$

【例 3.10】 设字长为8,二进制数 x=10010,y=−10111,求 x+y。

$$[x]_{补}=00010010,\quad [y]_{补}=11101001$$

$$[x+y]_{补}=00010010+11101001$$

$$=11111011$$

$$\begin{array}{r}00010010\\+\ 11101001\\\hline 11111011\end{array}$$

由于符号位为1,结果为负,其相反数的补码为 $[-(x+y)]_{补}=00000101$,所以 −(x+y)=101,x+y=−101。

(3) 当 x≥0,y<0 且 x+y≥0 时

因为

$$[x]_{补}=x,\quad [y]_{补}=(10^n+y)_2,\quad x+y\geqslant 0$$

所以

$$\begin{aligned}[x+y]_{补}&=x+y\\&=(10^n)_2+(x+y)\qquad(\bmod 2^n)\\&=x+(10^n+y)_2\\&=[x]_{补}+[y]_{补}\end{aligned}$$

【例 3.11】 设字长为8,二进制数 x=10111,y=−10010,求 x+y。

$$[x]_{补}=00010111,\quad [y]_{补}=11101110$$

$$[x+y]_{补}=00010111+11101110\qquad(\bmod 2^8)$$

$$=00000101$$

$$\begin{array}{rr}&00010111\\&+\ 11101110\\\hline \text{进位舍弃}\ 1&00000101\end{array}$$

由于 x+y>0,所以 x+y=101。

(4) 当 x<0,y<0 时

因为

$$[x]_{补}=(10^n+x)_2,\quad [y]_{补}=(10^n+y)_2,\quad -2^{n-1}\leqslant x+y<0$$

所以

$$
\begin{aligned}
[x+y]_{补} &= (10^n + x + y)_2 \qquad (>0)\\
&= (10^n)_2 + (10^n + x + y)_2 \qquad (\bmod 2^n)\\
&= (10^n + x)_2 + (10^n + y)_2\\
&= [x]_{补} + [y]_{补}
\end{aligned}
$$

【例 3.12】 设字长为 8，二进制数 x=－10111，y=－10010，求 x+y。

$$
\begin{aligned}
&[x]_{补} = 11101001, \quad [y]_{补} = 11101110\\
&[x+y]_{补} = 11101001 + 11101110 \qquad (\bmod 2^8)\\
&\qquad = 11010111
\end{aligned}
$$

$$
\begin{array}{ll}
 & 11101001\\
+ & 11101110\\
\hline
1 & 11010111
\end{array}
$$

进位舍弃

由于符号位为 1，结果为负，其相反数的补码为$[-(x+y)]_{补} = 00101001$，所以$-(x+y)=101001$，$x+y=-101001$。

2. 补码减法

补码减法公式为

$$[x-y]_{补} = (10^n + [x]_{补} - [y]_{补})_2 \quad (\bmod 2^n)$$

其中，$-2^{n-1} \leqslant x, y, x-y < 2^{n-1}$，x 和 y 为整数，n 为字长。

减法公式中的模 10^nB(即 2^n)可以理解为：当$[x]_{补} < [y]_{补}$时，它被作为减法运算的借位值；当$[x]_{补} \geqslant [y]_{补}$时，它被舍弃。

通过两个互为相反数的补码关系$[y]_{补} + [-y]_{补} = (10^n)_2$和补码加法公式，可以推出上面的补码减法公式：

$$
\begin{aligned}
[x-y]_{补} &= [x+(-y)]_{补}\\
&= [x]_{补} + [-y]_{补} \qquad (\bmod 2^n)\\
&= [x]_{补} + (10^n - [y]_{补})_2\\
&= (10^n + [x]_{补} - [y]_{补})_2
\end{aligned}
$$

【例 3.13】 设字长为 8，二进制数 x=10010，y=10111，求 x－y。

$$
\begin{aligned}
&[x]_{补} = 00010010, \quad [y]_{补} = 00010111\\
&[x-y]_{补} = 100000000 + 00010010 - 00010111 \qquad (\bmod 2^8)\\
&\qquad = 11111011
\end{aligned}
$$

借位

$$
\begin{array}{ll}
 & 1\ \ 00010010\\
- & 00010111\\
\hline
 & 11111011
\end{array}
$$

由于符号位为 1，结果为负，其相反数的补码为$[-(x-y)]_{补} = 00000101$，所以$-(x-y)=101$，$x-y=-101$。

【例 3.14】 设字长为 8，二进制数 x=－10111，y=10010，求 x－y。

$$
\begin{aligned}
&[x]_{补} = 11101001, \quad [y]_{补} = 00010010\\
&[x-y]_{补} = 100000000 + 11101001 - 00010010 \qquad (\bmod 2^8)\\
&\qquad = 11010111
\end{aligned}
$$

无借位舍弃

$$\begin{array}{rl} & 1\ \ 1101001 \\ - & 0\ \ 0010010 \\ \hline & 1\ \ 1010111 \end{array} \quad (\mathrm{mod}\ 2^8)$$

由于符号位为1,结果为负,其相反数的补码为$[-(x-y)]_{补}=00101001$,所以$-(x-y)=101001$,$x-y=-101001$。

在进行补码加减运算时,应注意以下几点:

(1) 补码加减的运算结果仍为补码;

(2) 补码的符号位是数的一部分,它同其他数字位一样参与运算;

(3) 补码表示范围是有限的,在字长为n位的情况下,它的表示范围是$-2^{n-1}\leqslant x<2^{n-1}$。

补码加减的运算结果也不得超过此范围,否则会因无法表示而出现溢出(Overflow)。

【例3.15】 字长为8,二进制数$x=-1000110$,$y=-1000001$,求$x+y$。

因为 $[x]_{补}=10111010$,$[y]_{补}=10111111$

所以 $[x+y]_{补}=10111010+10111111 \quad (\mathrm{mod}\ 2^8)$

$=01111001$

由此得到$x+y>0$的错误结果,出错的原因是由于真值运算结果:

$$x+y=-10000111$$

已超出了8位补码的下限-10000000。

【例3.16】 字长为8,二进制数$x=1100100$,$y=-110010$,求$x-y$。

因为 $[x]_{补}=01100100$,$[y]_{补}=11001110$

所以 $[x-y]_{补}=100000000+01100100-11001110 \quad (\mathrm{mod}\ 2^8)$

$=10010110$

由此得到$x-y<0$的错误结果,出错的原因是由于真值运算结果:

$$x-y=+10010110$$

已超出了8位补码的上限$+1111111$。

值得一提的是:只有当两个同号数相加(或两个异号数相减)时,才有可能出现溢出,而两个异号数相加(或两个同号数相减)只能使结果的绝对值变得更小,是不可能出现溢出的。

3.1.5 机器数的大小比较与移码

在计算机中,通常利用减法运算结果来判断两个数的大小。例如,通过表3.3中给出的方法可以判断两个数之间的大小关系。

表3.3 数的比较方法

无符号数a、b的比较方法			有符号数x、y的比较方法		
比较运算结果	判断依据	判断结果	比较运算结果	判断依据	判断结果
$a-b>0$	无借位且结果非0	$a>b$	$x-y>0$	结果为正且非0	$x>y$
$a-b=0$	结果为0	$a=b$	$x-y=0$	结果为0	$x=y$
$a-b<0$	有借位	$a<b$	$x-y<0$	结果为负	$x<y$

在机器数中，通过借位状态来判断两个无符号数的大小是比较方便的。但通过结果的符号位来判断两个有符号数(补码)的大小则较为麻烦。首先，不能通过借位状态来判断它们的大小，因为如果是两个异号数相减，由于执行的是相加运算，借位状态已无意义。其次，相减结果的正负有可能因溢出而无法正确表示。为了便于有符号数的大小比较，在计算机中常用“移码”(Biased Encoding)来表示有符号数。所谓移码是指将有符号数 x 加上正偏移值 B 后所形成的无符号数，即

$$[x]_{移}=B+x$$

其中，偏移值 B 为一个正数，它等于真值 x 最小数的绝对值。真值 x 的取值不能超出有符号机器数的表示范围。

字长为 n 的移码与补码之间的关系是：

$$[x]_{移}=B+[x]_{补} \pmod{2^n}$$

其中，x 为整数，其范围是 $-2^{n-1}\leqslant x<2^{n-1}$；B 为一个正数，在这里 $B=|x_{min}|=2^{n-1}$。下面列出了 8 位补码、移码与真值之间的对应关系：

真值 x	补码$[x]_{补}$	移码$[x]_{移}$
127	01111111	11111111
126	01111110	11111110
⋮	⋮	⋮
1	00000001	10000001
0	00000000	10000000
−1	11111111	01111111
⋮	⋮	⋮
−127	10000001	00000001
−128	10000000	00000000

可见，移码已将有符号数转化为无符号数，因此可以像判断无符号数那样来判断移码的大小。由于移码的大小关系与对应真值的大小关系是一样的，它与补码的转换又十分方便，所以可以通过移码间接地判断补码(有符号数)的大小。

【例 3.17】 设字长为 8，二进制数 x=1010，y=−10100，分别用补码和移码的相减结果来比较 x 和 y 的大小。

已知：$[x]_{补}=00001010$，$[y]_{补}=11101100$

$[x]_{移}=10000000+[x]_{补} \pmod{2^8}=10001010$

$[y]_{移}=10000000+[y]_{补} \pmod{2^8}=01101100$

(1) 补码比较结果($[x]_{补}-[y]_{补}$)

借位 1　00001010

−　11101100

00011110

两个异号数相减实质上是相加运算，这里借位信号已没有意义，不能通过有借位来判定 x<y。由于结果没有溢出，因此可以通过符号位是“0”来判定 x>y。