第一章　数制与编码

计算技术的进步促进了科学技术和生产的飞跃发展。现在，计算机已广泛地应用于科学与工程计算、数据和信息处理、过程和实时控制、计算机辅助设计和辅助制造以及人工智能等领域。计算机是数字系统中最常见、最有代表性的一种设备。

数字系统的特点是它所处理的信息都是离散元素，这些元素可以是各种数字、字母、算符及符号等。离散元素按不同方式排列可以表示各种信息，例如，字母 C,O,M,P,U,T,E 和 R 可形成单词 COMPUTER,数字 1999 表示一个确定的数,等等。

在数字系统中，信息的离散元素是以称为信号的物理量来表示的，电压和电流就是最常用的电信号。通常，数字系统的信号为“有”或“无”两个离散量，因此，称为二进制信号。由于只有导通和截止两种工作状态的电子器件能可靠地反映两个离散量，并且在工程上比较容易实现，加上人类的逻辑思维方式也倾向于二值，所以，数字系统常采用二进制信号。

信号的离散量有的是自然形成的，有的则是对连续过程有意加以量化后而得到的。例如，一份学生成绩单就是由离散量形成的，上面有学生的学号、姓名、课程名称、得分等。又如，科学技术工作者在进行科学研究工作时，常常要观察连续的过程，但仅将一些特定的数值记录下来并列成表，这样就将连续的信息离散并且量化了，表格中的每一个数字都已成为离散量。

数字系统处理的是离散元素，而这些离散元素通常以二进制形式出现，人们熟悉的十进制数不能被机器直接接受。因此，当人机通信时，需将十进制数转换成二进制数，以便机器接受。机器运算结束时，再将二进制数转换成十进制数。

本章主要讨论数字系统中数的表示方法。

1.1　进位计数制

1.1.1　十进制数的表示

用一组统一的符号和规则表示数的方法，称为进位计数制，简称数制。

原则上说，一个数可以用任何一种进位计数制来表示和运算。但不同数制

其运算方法及难易程度互不相同。选择什么样的进位计数制来表示数，对数字系统的性能影响很大。

在日常生活中，人们通常采用十进制数来计数，每位数可用十个数码之一来表示，即 0,1,2,3,4,5,6,7,8,9 。十进制的基数为 10，基数表示进位制所具有的数字符号个数，十进制具有的数字符号个数为 10。

十进制数的计算规律是由低位向高位进位时“逢十进一”，也就是说，每位累计不能超过 9，计满 10 就应向高位进 1。

当人们看到一个十进制数，如 632.45 时，就会立刻想到：这个数的最左位（即第一位）为百位（6 代表 600），第二位为十位（3 代表 30），第三位为个位（2 代表 2），小数点右面第一位为十分位（4 代表 4/10），第二位为百分位（5 代表 5/100）。这里百、十、个、十分之一和百分之一都是 10 的次幂。在一个进位制表示的数中，不同数位上的固定常数称之为“权”。十进制数 632.45 从左至右各位的权分别是 $10^2,10^1,10^0,10^{-1},10^{-2}$ 。这样，632.45 按权展开的形式如下：

$$632.45=6\times10^2+3\times10^1+2\times10^0+4\times10^{-1}+5\times10^{-2}$$

等式左边的表示方法称之为位置记数表示法，等式右边则是其按权展开表示法。

一般说来，对于任意一个十进制数 N，可用位置记数法表示如下：

$$(N)_{10}=(a_{n-1}a_{n-2}\cdots a_1a_0.a_{-1}a_{-2}\cdots a_{-m})_{10}$$

也可用按权展开表示法表示如下：

$$\begin{aligned}(N)_{10}&=a_{n-1}\times10^{n-1}+a_{n-2}\times10^{n-2}+\cdots+a_1\times10^1+a_0\times10^0\\&\quad+a_{-1}\times10^{-1}+a_{-2}\times10^{-2}+\cdots+a_{-m}\times10^{-m}\\&=\sum_{i=-m}^{n-1}a_i\times10^i\end{aligned}$$

式中：a_i 表示各个数字符号，为 0～9 这 10 个数码中的任意一个；n 为整数部分的位数；m 为小数部分的位数。

通常，对十进制数的表示，可以在数字的右下角标注 10 或 D。

1.1.2 二进制数的表示

数字系统中使用的进位制并不限于十进制，当进位基数为 2 时，称为二进制。在二进制中，只有 0 和 1 两个数码。二进制的计数规则是由低位向高位“逢二进一”，即每位计满 2 就向高位进 1，例如，$(1101)_2$ 就是一个二进制数。不同数位的数码表示的值不同，各位的权值是以 2 为底的连续整数幂，从右向左递增。

对于任意一个二进制数 N，用位置计数法表示为

$$(N)_2=(a_{n-1}a_{n-2}\cdots a_1a_0.a_{-1}a_{-2}\cdots a_{-m})_2$$

用按权展开法表示为

$$(N)_2=a_{n-1}\times2^{n-1}+a_{n-2}\times2^{n-2}+\cdots+a_1\times2^1+a_0\times2^0+a_{-1}\times2^{-1}$$

$$+a_{-2}\times 2^{-2}+\cdots+a_{-m}\times 2^{-m}$$

$$=\sum_{i=-m}^{n-1}a_i\times 2^i$$

式中：a_i 表示各个数字符号，为 0 或 1；n 为整数部分的位数；m 为小数部分的位数。

通常，对二进制数的表示，可以在数字右下角标注 2 或 B。

在数字系统中，常用二进制来表示数字和进行运算。因为它具有如下特点：

(1) 二进制数只有 0 和 1 两个数码，任何具有两个不同稳定状态的元件都可用来表示 1 位二进制数，例如，晶体管的导通和截止，脉冲信号的“有”和“无”等。

(2) 二进制运算规则简单。其运算规则如下：

加法规则

$0+0=0$　　$0+1=1$

$1+0=1$　　$1+1=0$(同时向相邻高位进 1)

减法规则

$0-0=0$　　$0-1=1$(同时向相邻高位借 1)

$1-0=1$　　$1-1=0$

乘法规则

$0\times 0=0$　　$0\times 1=0$

$1\times 0=0$　　$1\times 1=1$

除法规则

$0\div 1=0$　　$1\div 1=1$

下面举例说明。

例 1.1　进行 1101+1011 运算。

解

$$\begin{array}{r} 1\ 1\ 0\ 1 \\ +)\ 1\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 0\ 0 \end{array}$$

两个二进制数的加法运算和十进制数的加法运算相似，但采用“逢二进一”的法则，每位数累计到 2 时，本位就记为 0，同时向相邻高位进 1。

例 1.2　进行 11101－10011 运算。

解

$$\begin{array}{r} 1\ 1\ 1\ 0\ 1 \\ -)\ 1\ 0\ 0\ 1\ 1 \\ \hline 1\ 0\ 1\ 0 \end{array}$$

二进制减法运算从低位起按位进行，在遇到 0 减 1 时，就要采用“借一当二”法则向相邻高位借 1，也就是从那一位减去 1。

例 1.3　进行 1101×1001 运算。

解

$$
\begin{array}{r}
1\ 1\ 0\ 1 \\
\times)\ 1\ 0\ 0\ 1 \\
\hline
1\ 1\ 0\ 1 \\
0\ 0\ 0\ 0 \\
0\ 0\ 0\ 0 \\
1\ 1\ 0\ 1 \\
\hline
1\ 1\ 1\ 0\ 1\ 0\ 1
\end{array}
$$

二进制数的乘法运算和十进制数的乘法运算相似，所不同的是对部分积进行累加时要按“逢二进一”的原则。

例 1.4　进行 $10010001\div1011$ 运算。

解

$$
\begin{array}{r}
1\ 1\ 0\ 1 \cdots\cdots \text{商} \\
1\ 0\ 1\ 1\ \sqrt{1\ 0\ 0\ 1\ 0\ 0\ 0\ 1} \\
1\ 0\ 1\ 1 \\
\hline
1\ 1\ 1\ 0 \\
1\ 0\ 1\ 1 \\
\hline
1\ 1\ 0\ 1 \\
1\ 0\ 1\ 1 \\
\hline
1\ 0 \cdots\cdots \text{余数}
\end{array}
$$

二进制数的除法运算同十进制数的除法运算类似，但采用二进制数的运算规则。

(3) 二进制数只有两个状态，数字的传输和处理不容易出错，可靠性高。

(4) 二进制数的数码 0和 1,可与逻辑代数中逻辑变量的值“假”和“真”对应起来。也就是说，可用一个逻辑变量来表示一个二进制数码。这样，在逻辑运算中就可以使用逻辑代数这一数学工具。

1.1.3　其他进制数的表示

二进制数运算规则简单，便于电路实现，它是数字系统中广泛采用的一种数制。但用二进制表示一个数时，所用的位数比用十进制数表示的位数多，人们读写很不方便,容易出错,不便记忆。因此,人们常采用八进制数和十六进制数,这两种数制不但容易书写和阅读，便于记忆，而且具有二进制数的特点，十分容易将它们转换成二进制数。

八进制数的基数是 8,采用的数码是 0,1,2,3,4,5,6,7。计数规则是从低位向高位“逢八进一”,对于相邻两位来说,高位的权值是低位权值的 8 倍。例如,数$(47.6)_8$就表示一个八进制数。由于八进制的数码和十进制前 8 个数码相

同,为了便于区分,通常在八进制数字的右下角标注 8 或数字后边接 O。

十六进制数的基数为 16,采用的数码是 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F。其中,A,B,C,D,E,F 分别代表十进制数字 10,11,12,13,14,15。十六进制的计数规则是从低位向高位"逢十六进一",对于相邻两位来说,高位的权值是低位权值的 16 倍。例如,$(54AF.8B)_{16}$就是一个十六进制数。通常,在十六进制数字的右下角标注 16 或数字后边接 H。

与二进制数一样,八进制数和十六进制数均可用位置计数法的形式和按权展开法的形式表示。

一般说来,对于任意的数 N,都能表示成以 r 为基数的 r 进制数。数 N 的位置记数法为

$$(N)_r=(a_{n-1}a_{n-2}\cdots a_1a_0.a_{-1}a_{-2}\cdots a_{-m})_r$$

而相应的按权展开表示法为

$$\begin{aligned}(N)_r &= a_{n-1}\times r^{n-1}+a_{n-2}\times r^{n-2}+\cdots+a_1\times r^1+a_0\times r^0\\ &\quad +a_{-1}\times r^{-1}+a_{-2}\times r^{-2}+\cdots+a_{-m}\times r^{-m}\\ &=\sum_{i=-m}^{n-1}a_ir^i\end{aligned}$$

式中:a_i 表示各个数字符号,为 $0\sim r-1$ 数码中任意一个;r 为进位制的基数;n 为整数部分的位数;m 为小数部分的位数。

r 进制的计数规则是从低位向高位"逢 r 进一"。

常用的不同数制的各种数码见表 1.1,该表列出了当 r 为 10,2,8,16 时各种进位计数制中开头的 16 个自然数。

表 1.1 不同进位计数制的各种数码

十进制数 ($r=10$)	二进制数 ($r=2$)	八进制数 ($r=8$)	十六进制数 ($r=16$)
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1.2 数 制 转 换

1.2.1 二进制数与十进制数的转换

在计算机和其他数字系统中，普遍使用二进制数，采用二进制数的数字系统只能处理二进制数或用二进制代码形式表示的其他进位制数。而人们习惯于使用十进制数，所以，在信息处理中首先必须把十进制数转换成计算机能加工和处理的二进制数进行运算，然后再将二进制数的计算结果转换成人们习惯的十进制数。

二进制数转换成十进制数是很方便的，只要将二进制数写成按权展开式，并将式中各乘积项的积算出来，然后各项相加，即可得到与该二进制数相对应的十进制数。例如

$$\begin{aligned}(11010.101)_2 &= 1\times2^4+1\times2^3+0\times2^2+1\times2^1+0\times2^0\\&\quad+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}\\&=16+8+2+0.5+0.125\\&=(26.625)_{10}\end{aligned}$$

十进制数转换成二进制数时，需将待转换的数分成整数部分和小数部分，并分别加以转换。一个十进制数可写成

$$(N)_{10}=\langle\text{整数部分}\rangle_{10}.\langle\text{小数部分}\rangle_{10}$$

转换时,首先将〈整数部分〉$_{10}$转换成〈整数部分〉$_2$,然后再将〈小数部分〉$_{10}$转换成〈小数部分〉$_2$。待整数部分和小数部分确定后，就可写成

$$(N)_2=\langle\text{整数部分}\rangle_2.\langle\text{小数部分}\rangle_2$$

1. 整数转换

十进制数的整数部分采用“除 2 取余”法进行转换,即把十进制数除以 2,取出余数 1 或 0 作为相应二进制数的最低位，把得到的商再除以 2,取余数 1 或 0 作为二进制数的次低位，依次类推，继续上述过程，直至商为 0,所得余数为最高位。

例如，为将十进制整数 58 转换为二进制整数，就要把它写成如下形式：

$$\begin{aligned}(58)_{10}&=(a_{n-1}a_{n-2}\cdots a_1a_0)_2\\&=a_{n-1}\times2^{n-1}+a_{n-2}\times2^{n-2}+\cdots+a_1\times2^1+a_0\times2^0\\&=2(a_{n-1}\times2^{n-2}+a_{n-2}\times2^{n-3}+\cdots+a_1)+a_0\end{aligned}$$

只要求出等式中的各个系数 $a_{n-1},a_{n-2},\cdots,a_1,a_0$ 便得到二进制整数。将上式两边除以 2,得

$$(29)_{10}=a_{n-1}\times 2^{n-2}+a_{n-2}\times 2^{n-3}+\cdots+a_1+\frac{a_0}{2}$$

两数相等，整数部分和小数部分必须对应相等，等式左边余数为 0，则 a_0 为 0。因而得

$$(29)_{10}=2(a_{n-1}\times 2^{n-3}+a_{n-2}\times 2^{n-4}+\cdots+a_2)+a_1$$

将等式两边再除以 2，得

$$(14+\frac{1}{2})_{10}=a_{n-1}\times 2^{n-3}+a_{n-2}\times 2^{n-4}+\cdots+a_2+\frac{a_1}{2}$$

比较等式两边，等式左边余数为 1，则取 a_1 为 1。

依次类推，可得系数 $a_2,a_3,\cdots,a_{n-2},a_{n-1}$。

根据上面讨论的方法，可用下列形式很方便地将十进制整数转换成二进制整数：

2|58

2|29　　余数 $0(a_0)$ 最低位

2|14　　余数 $1(a_1)$

2|7　　余数 $0(a_2)$

2|3　　余数 $1(a_3)$

2|1　　余数 $1(a_4)$

0　　余数 $1(a_5)$ 最高位

因此，$(58)_{10}=(111010)_2$。

2. 纯小数转换

十进制数的小数部分采用“乘 2 取整”法进行转换，即先将十进制小数乘以 2，取其整数 1 或 0 作为二进制小数的最高位；然后将乘积的小数部分再乘以 2，并再取整数作为次高位。重复上述过程，直到小数部分为 0 或达到所要求的精度。

例如，将十进制小数 0.625 转换为二进制小数，需把它写成如下形式：

$$\begin{aligned}(0.625)_{10}&=(0.a_{-1}a_{-2}\cdots a_{-m})_2\\&=a_{-1}\times 2^{-1}+a_{-2}\times 2^{-2}+\cdots+a_{-m}\times 2^{-m}\\&=\frac{a_{-1}}{2}+\frac{1}{2}(a_{-2}\times 2^{-1}+\cdots+a_{-m}\times 2^{-m+1})\end{aligned}$$

只要求出各系数 $a_{-1},a_{-2},\cdots,a_{-m}$，便得到二进制小数。

将上式两边乘 2，得

$$(1.25)_{10}=a_{-1}+(a_{-2}\times 2^{-1}+\cdots+a_{-m}\times 2^{-m+1})$$

根据两个数相等，其整数部分和小数部分必须分别相等的道理，a_{-1} 等于左边的

整数，则 a_{-1} 为 1。

等式右边括号内的数仍为小数，因而

$$(0.25)_{10}=\frac{a_{-2}}{2}+\frac{1}{2}(a_{-3}\times 2^{-1}+\cdots+a_{-m}\times 2^{-m+2})$$

再将等式两边乘 2，得

$$(0.5)_{10}=a_{-2}+(a_{-3}\times 2^{-1}+\cdots+a_{-m}\times 2^{-m+2})$$

比较等式两边的整数，又取 a_{-2} 为 0。如此连续乘 2，直到小数部分等于 0，即可求得系数 $a_{-1},a_{-2},\cdots,a_{-m}$。

根据上面讨论的方法，可用下列形式很方便地将十进制小数转换成二进制小数：

```
    0.625
×)      2
-------
  [1].250      整数 1(a_-1)最高小数位
×)      2
-------
  [0].500      整数 0(a_-2)
×)      2
-------
  [1].000      整数 1(a_-3)最低小数位
```

因此，$(0.625)_{10}=(0.101)_2$。

必须指出：运算时，式中的整数不参加连乘。

在十进制的小数部分转换中，有时连续乘 2 不一定能使小数部分等于 0，这说明该十进制小数不能用有限位二进制小数表示。这时，只要取足够多的位数，使其误差达到所要求的精度就可以了。下面举例说明。

例 1.5　将十进制数 0.18 转换成二进制数，精确到小数点后 4 位。

解

```
    0.18
×)     2
-------
  [0].36       整数 0(a_-1)
×)     2
-------
  [0].72       整数 0(a_-2)
×)     2
-------
  [1].44       整数 1(a_-3)
×)     2
-------
  [0].88       整数 0(a_-4)
×)     2
-------
  [1].76       整数 1(a_-5)
```

十进制数 0.18 连续四次乘 2 后，其小数部分等于 0.88，仍不为 0。由于要

求精确到小数点后 4位,因此将 0.88再乘一次 2,小数点后第 5位为 1,得

$$(0.18)_{10}\approx(0.00101)_2$$

如果一个十进制数既有整数部分又有小数部分，转换时，整数部分采用“除 2取余”法,小数部分采用“乘 2取整”法，然后再把转换的结果合并起来。下面举例说明。

例 **1.6** 将$(58.625)_{10}$转换成二进制数。

解 $(58.625)_{10}=(58)_{10}+(0.625)_{10}$

$=(111010)_2+(0.101)_2$

$=(111010.101)_2$

1.2.2 八进制数、十六进制数与二进制数的转换

八进制数的基数是 $8(8=2^3)$，十六进制数的基数为 $16(16=2^4)$。二进制数、八进制数和十六进制数之间具有 2 的整指数倍关系，因而可直接进行转换。

将二进制数转换成八进制或十六进制数的方法为：从小数点开始，分别向左、右按 3位(转换成八进制)或 4 位(转换成十六进制)分组,最后不满 3 位或 4 位的则需加 0。将每组以对应的八进制数或十六进制数代替，即为等值的八进制数和十六进制数。例如：

八进制	2	5	7	.	0	5	5	4
二进制	010	101	111	.	000	101	101	100
十六进制	A	F		.	1	6	C	

则 $(257.0544)_8=(10101111.0001011011)_2=(AF.16C)_{16}$

将八进制数或十六进制数转换成二进制数时，可按上述方法的逆过程进行。

1.3 带符号数的代码表示

1.3.1 真值与机器数

前面讨论的数都没有考虑符号，一般认为是正数，但在算术运算中总会出现负数。不带符号的数是数的绝对值，在绝对值前加上表示正负的符号就成了带符号数。一个带符号的数由两部分组成：一部分表示数的符号，另一部分表示数的数值。对于一个 n位二进制数,如果数的第一位为符号位,则剩下的 $n-1$ 位就表示数的数值部分。一般，直接用正号“+”和负号“-”来表示带符号的二进制数，叫做符号数的真值。数的真值形式是一种原始形式，不能直接用于计算机

中。但是，当使符号数值化以后，就可在计算机中使用它。数的符号是一个具有正、负两种值的离散信息，它可以用一位二进制数来表示。习惯上以 0 表示正数，而以 1 表示负数。计算机中使用的符号数叫做机器数。

例如，二进制正数 $+0.1011$ 在机器中的表示如图 1.1(a)所示，二进制负数 -0.1011 在机器中的表示如图 1.1(b)所示。

由二进制数的加、减、乘、除 4 种运算可知，乘法运算实际上是做移位加法运算，而除法运算则是做移位减法运算。这就是说，在机器中只需要做加、减两种运算。但做减法运算时，必须先比较两个数绝对值的大小，将绝对值大的数减绝对值小的数，最后在相减结果的前面加上正确的符号。虽然逻辑电路可以实现减法运算，但所需的电路复杂，运算时间较长。为了能使减法运算变成加法运算，人们提出了三种机器数的表示形式，即原码、反码和补码。

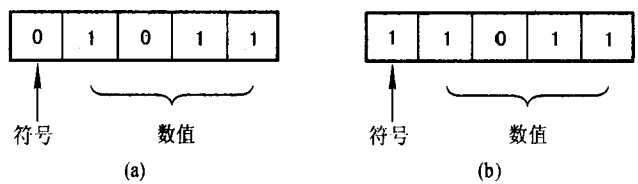

图 1.1　二进制数在机器中的表示

1.3.2　原码

原码又称为“符号一数值表示”。在以原码形式表示的正数和负数中，第 1 位表示符号位，对于正数，符号位记作 0，对于负数，符号位记作 1，其余各位表示数值部分。

假如两个带符号的二进制数分别为 N_1 和 N_2，其真值形式为

$$N_1 = +10011 \qquad N_2 = -01010$$

则 N_1 和 N_2 的原码表示形式为

$$[N_1]_{原} = 010011 \qquad [N_2]_{原} = 101010$$

根据上述原码形成规则，一个 n 位的整数 N（包括一位符号位）的原码一般表示式为

$$[N]_{原} = \begin{cases} N & 0 \leqslant N < 2^{n-1} \\ 2^{n-1} - N & -2^{n-1} < N \leqslant 0 \end{cases}$$

对于定点小数，通常小数点定在最高位的左边，这时，数值小于 1。定点小数原码一般表示式为

$$[N]_{原} = \begin{cases} N & 0 \leqslant N < 1 \\ 1 - N & -1 < N \leqslant 0 \end{cases}$$

从原码的一般表示式可以看出：

(1) 当 N 为正数时，$[N]_{原}$和 N 的区别只是增加一位用 0 表示的符号位。由于在数的左边增加一位 0 对该数的数值并无影响，所以 $[N]_{原}$ 就是 N 本身。

(2) 当 N 为负数时，$[N]_{原}$和 N 的区别是增加一位用 1 表示的符号位。

(3) 在原码表示中，有两种不同形式的 0，即

$$[+0]_{原}=0.00\cdots0$$

$$[-0]_{原}=1.00\cdots0$$

1.3.3 反码

反码又称为“对 1 的补数”。用反码表示时，左边第 1 位也为符号位，符号位为 0 代表正数，符号位为 1 代表负数。对于负数，反码的数值是将原码数值按位求反，即原码的某位为 1，反码的相应位就为 0，或者原码的某位为 0，反码的相应位就为 1。而对于正数，反码和原码相同。所以，反码数值的形成与它的符号位有关。

假如两个带符号的二进制数分别为 N_1 和 N_2，其真值形式为

$$N_1=+10011 \qquad N_2=-01010$$

则 N_1 和 N_2 的反码表示形式为

$$[N_1]_{反}=010011 \qquad [N_2]_{反}=110101$$

根据上述的反码形成规则，一个 n 位的整数 N(包括一位符号位)的反码一般表示式为

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

同样，对于定点小数，若小数部分的位数为 m，则它的反码一般表示为

$$[N]_{反}=\begin{cases}N & 0\leqslant N<1\\ (2-2^{-m})+N & -1<N\leqslant 0\end{cases}$$

从反码的一般表示式可以看出：

(1) 正数 N 的反码 $[N]_{反}$ 与原码 $[N]_{原}$ 相同

(2) 对于负数 N，其反码 $[N]_{反}$ 的符号位为 1，数值部分是将原码数值按位变反。

(3) 在反码中有两种不同的 0 表示形式，即

$$[+0]_{反}=0.00\cdots0$$

$$[-0]_{反}=1.11\cdots1$$

1.3.4 补码

补码又称为“对 2 的补数”。在补码表示法中，正数的表示同原码和反码的表示是一样的，而负数的表示却不同。对于负数，其符号位为 1,而数值位是将原码按位变反加 1,即按位变反,再在最低位加 1。

假如两个带符号的二进制数分别为 N_1 和 N_2,其真值形式为

$$N_1=+10011 \qquad N_2=-01010$$

则 N_1 和 N_2 的补码表示形式为

$$[N_1]_{补}=010011 \qquad [N_2]_{补}=110110$$

根据上述补码形成规则，一个 n 位的整数 N(包括一位符号位)的补码一般表示式为

$$[N]_{补}=\begin{cases}N & 0\leqslant N<2^{n-1}\\ 2^n+N & -2^{n-1}\leqslant N<0\end{cases}$$

同样,对于定点小数,补码一般表示式可写成

$$[N]_{补}=\begin{cases}N & 0\leqslant N<1\\ 2+N & -1\leqslant N<0\end{cases}$$

由补码的一般表示式可以看出：

(1) 正数 N 的补码 $[N]_{补}$ 与原码 $[N]_{原}$ 和反码 $[N]_{反}$ 相同。

(2) 对于负数,补码 $[N]_{补}$ 的符号位为 1,其数值部分为反码数值加 1。

(3) 在补码表示法中,0 的表示形式是唯一的，即

$$[+0]_{补}=0.00\cdots0$$

$$[-0]_{补}=0.00\cdots0$$

1.3.5 机器数的加、减运算

带符号数的三种表示法的形成规则不同，其加、减运算的规律也不相同。

1. 原码运算

原码中的符号位仅用于表示数的正、负，不参加运算，进行运算的只是数值部分。原码运算时，应首先比较两个数的符号，若两数的符号相同，则两数相加就是将两个数的数值相加，结果的符号不变；若两数的符号不同，就得进一步比较两数的数值相对大小，两数相加是将数值较大的数减去数值较小的数，结果的符号与数值较大的数的符号相同。下面举例说明。

例 1.7 已知 $N_1=-0.0011, N_2=0.1011$,求 $[N_1+N_2]_{原}$ 和 $[N_1-N_2]_{原}$.

解 $[N_1+N_2]_{原}=[(-0.0011)+0.1011]_{原}$

由于 N_1 和 N_2 的符号不同,并且 N_2 的绝对值大于 N_1 的绝对值,因此,要进行 N_2 减 N_1 的运算,其结果为正。

$$\begin{array}{r} 0.1011 \\ -)\ 0.0011 \\ \hline 0.1000 \end{array}$$

运算结果为原码,即

$$[N_1+N_2]_{原}=0.1000$$

故其真值为

$$N_1+N_2=0.1000$$

而

$$[N_1-N_2]_{原}=[(-0.0011)-0.1011]_{原}$$

由于 N_1 和 N_2 的符号相同,因此,实际上要进行 N_1 加 N_2 的运算,其结果为负。

$$\begin{array}{r} 0.0011 \\ +)\ 0.1011 \\ \hline 0.1110 \end{array}$$

运算结果为原码,即

$$[N_1-N_2]_{原}=1.1110$$

故其真值为

$$N_1-N_2=-0.1110$$

2. 补码运算

由补码的定义可以证明如下补码加、减运算规则:

$$[N_1+N_2]_{补}=[N_1]_{补}+[N_2]_{补}$$

$$[N_1-N_2]_{补}=[N_1]_{补}+[-N_2]_{补}$$

补码的加、减运算规则表明:两数和的补码等于两数的补码之和,而两数差的补码也可以用加法来实现。运算时,符号位和数值位一样参加运算,如果符号位产生进位,则需将此进位“丢掉”。运算结果的符号位为 0 时,说明是正数的补码;运算结果的符号为 1 时,说明是负数的补码。下面举例说明。

例 1.8　已知 $N_1=-0.1100, N_2=-0.0010$,求 $[N_1+N_2]_{补}$ 和 $[N_1-N_2]_{补}$。

解　$[N_1+N_2]_{补}=[N_1]_{补}+[N_2]_{补}=1.0100+1.1110$

$$\begin{array}{rr} & 1.0100 \\ +) & 1.1110 \\ \hline \text{丢掉}\leftarrow 1 & 1.0010 \end{array}$$

由于符号位产生了进位,因此,要将此进位丢掉,即

$$[N_1+N_2]_{补}=1.0010$$

运算结果的符号位为 1，说明是负数的补码，应对运算结果再求补码才得到原码,即

$$[N_1+N_2]_{原}=1.1110$$

故其真值为

$$N_1+N_2=-0.1110$$

而

$$[N_1-N_2]_{补}=[N_1]_{补}+[-N_2]_{补}=1.0100+0.0010$$

$$\begin{array}{r} 1.0100 \\ +)\ 0.0010 \\ \hline 1.0110 \end{array}$$

即

$$[N_1-N_2]_{补}=1.0110$$

由于运算结果的符号位为 1，是负数的补码，应再求结果的补码才得原码，即

$$[N_1-N_2]_{原}=1.1010$$

故其真值为

$$N_1-N_2=-0.1010$$

3. 反码运算

反码运算同补码运算一样，两数和的反码等于两数的反码之和，两数差的反码可以用两数反码的加法来实现。反码加、减运算规则是

$$[N_1+N_2]_{反}=[N_1]_{反}+[N_2]_{反}$$

$$[N_1-N_2]_{反}=[N_1]_{反}+[-N_2]_{反}$$

运算时，符号位和数值位一样参加运算，如果符号位产生了进位，则此进位应加到和数的最低位，称之为“循环进位”。运算结果符号位为 0,说明是正数的反码，与原码相同。运算结果符号位为 1，说明是负数的反码，应对结果再求反码才得原码。下面举例说明。

例 1.9　已知 $N_1=0.1001, N_2=0.0011$,求 $[N_1+N_2]_{反}$ 和 $[N_1-N_2]_{反}$.

解　$[N_1+N_2]_{反}=[N_1]_{反}+[N_2]_{反}=0.1001+0.0011$

$$\begin{array}{r} 0.1001 \\ +)\ 0.0011 \\ \hline 0.1100 \end{array}$$

即

$$[N_1+N_2]_{反}=0.1100$$

故其真值为

$$N_1 + N_2 = 0.1100$$

而

$$[N_1 - N_2]_{反} = [N_1]_{反} + [-N_2]_{反}$$
$$= 0.1001 + 1.1100$$

$$\begin{array}{r} 0.\ 1\ 0\ 0\ 1 \\ +)\ 1.\ 1\ 1\ 0\ 0 \\ \hline 1\ 0.\ 0\ 1\ 0\ 1 \\ +)\ \longrightarrow 1 \\ \hline 0.\ 0\ 1\ 1\ 0 \end{array}$$

由于符号位产生了进位,因此要进行“循环进位”,故

$$[N_1 - N_2]_{反} = 0.0110$$

其真值为

$$N_1 - N_2 = 0.0110$$

通过以上讨论可以看出，原码表示法简单直观，容易变换，但原码减法必须做真正的减法，不能用加法代替，这样，进行加、减运算很麻烦，增加了机器的运算时间，所需的逻辑电路也较复杂。反码和补码表示法可以简化减法运算，将减法变成加法且容易用逻辑电路实现。用补码进行减法运算较方便，因为它只需进行一次算术相加。用反码进行减法运算时，若符号位产生进位，则需进行两次算术相加。因此，在近代计算机中，加、减法几乎都采用补码运算。

1.3.6 十进制数的补数

二进制数的补码和反码的引入，使二进制数的减法运算转换成加法运算。在某些情况下，计算机或其他数字系统常用二进制代码直接表示十进制数，并进行运算。为了使十进制数的减法运算也能转换成加法运算，带符号十进制数也引入了同二进制数相似的三种表示方法，它们分别是符号－数值表示；“对 9 的补数”及“对 10 的补数”。十进制数的符号用 4 位二进制数表示，习惯上用 0000(等效于十进制数 0)表示正，而用 1001(相当于十进制数 9)表示负。

1. 对 10 的补数

十进制数“对 10 的补数”与二进制数的补码类似。

对于十进制正数 N,其“对 10 的补数”的表示形式是符号位为 0,数值部分则为十进制数 N 本身。

例如，十进制正数 $N = 5493$,其“对 10 的补数”为

$$[N]_{10补} = 05493$$

对于十进制负数 N，其“对 10 的补数”的一般形式为

$$[N]_{10补}=10^n+N \qquad -10^{n-1}<N<0$$

其中，n 是十进制负数 N 的整数部分的位数(包括 1 位符号位)。

例如，十进制数 $N=-3250$，其“对 10 的补数”为

$$[-3250]_{10补}=10^5-3250=96750$$

又如，十进制数 $N=-0.3267$，其“对 10 的补数”为

$$[-0.3267]_{10补}=10-0.3267=9.6733$$

由此可见，只要用 10 减最低位非 0 的数，然后用 9 减所有较高位的数，就可以形成十进制数的“对 10 的补数”。

十进制数“对 10 的补数”的减法运算和二进制数补码的减法运算相似，可将减法转换成加法来实现。下面举例说明。

例 1.10　给定 $N_1=72532$，$N_2=33256$，求 $N=N_1-N_2$。

解 用“对 10 的补数”进行运算，即

$$\begin{aligned}[N_1-N_2]_{10补}&=[72532-33256]_{10补}\\&=[72532]_{10补}+[-33256]_{10补}\\&=072532+966744\end{aligned}$$

$$\begin{array}{r}0\ 7\ 2\ 5\ 3\ 2\\ +)\ 9\ 6\ 6\ 7\ 4\ 4\\ \hline 1\ 0\ 3\ 9\ 2\ 7\ 6\end{array}$$

丢掉 ←

符号位产生的进位必须丢掉。运算结果是“对 10 的补数”，即

$$[N_1-N_2]_{10补}=039276$$

其真值为

$$N_1-N_2=39276$$

2. 对 9 的补数

十进制数的“对 9 的补数”与二进制数的反码类似。

对于十进制正数 N，其“对 9 的补数”的表示形式与 N 的“对 10 的补数”的表示形式相同。

例如，十进制正数 $N=8954$，其“对 9 的补数”为

$$[8954]_{9补}=08954$$

对于十进制负数 N，其“对 9 的补数”的一般表示形式为

$$[N]_{9补}=10^n-10^{-m}+N \qquad -10^{n-1}<N<0$$

其中：n 是十进制负数 N 的整数部分的位数（包括 1 位符号位）；m 是十进制负数 N 的小数部分的位数。

例如，十进制数 $N=-3250$，其“对 9 的补数”为

$$[-3250]_{9补}=10^5-1-3250=96749$$

又如，十进制数 $N=-25.639$ ，其“对 9 的补数”为

$$[-25.639]_{9补}=10^3-10^{-3}-25.639=974.360$$

由此可见，只要分别用 9 减十进制数的各位就可得到这个数的“对 9 的补数”。

十进制数的“对 9 的补数”的减法运算同二进制数的反码运算相类似，可将减法转换成加法来实现。下面举例说明。

例 1.11　若给定 $N_1=5489$ 和 $N_2=3250$，试求 $N=N_1-N_2$。

解 用“对 9 的补数”进行运算，即

$$\begin{aligned}[N_1-N_2]_{9补}&=[5489-3250]_{9补}\\&=[5489]_{9补}+[-3250]_{9补}\\&=05489+96749\end{aligned}$$

$$\begin{array}{r} 0\,5\,4\,8\,9 \\ +)\ 9\,6\,7\,4\,9 \\ \hline 1\ 0\,2\,2\,3\,8 \\ +)\quad \longrightarrow 1 \\ \hline 0\,2\,2\,3\,9 \end{array}$$

符号位产生的进位需加在和数的最低位上。运算的结果是“对 9 的补数”，即

$$[N_1-N_2]_{9补}=02239$$

其真值为

$$N_1-N_2=2239$$

1.4　数的定点表示和浮点表示

1.4.1　数的定点表示

在计算机中，处理小数点的方法有两种。一种是数的定点表示，另一种是数的浮点表示。

定点数是计算机内最基本的一种数据表示，在这种表示方法中，小数点的位置固定不变。由于定点位置不同，定点数一般分为两类：小数点固定在最低位右边的数称为整数；小数点固定在数的最左端称为分数或小数。在机器中，小数点实际上是不表示出来的，而是一个约定的位置。

定点数可表示为带符号的或不带符号的数，算术运算使用带符号的数，一般以左边最高位表示符号位。不带符号的数一般表示逻辑量或某些特征值，逻辑运算是按位进行的。

例如，定点整数 +1010110 在机器中的表示如图 1.2(a) 所示，定点小数