# 数 控 基 础

北京工业学院二系231专业

1977.6

# 毛主席语录

人类总得不断地总结经验，有所发现，有所发明，有所创造，有所前进。

引自周总理在中华人民共和国第三届全国人民代表大会第一次会议上的政府工作报告

我们不能走世界各国技术发展的老路，跟在别人后面一步一步地爬行。我们必须打破常规，尽量采用先进技术，在一个不太长的历史时期内，把我国建设成为一个社会主义的现代化的强国。

引自周总理在中华人民共和国第三届全国人民代表大会第一次会议上的政府工作报告

中国应当对于人类有较大的贡献。

纪念孙中山先生

# 目　录

# 第一章　运算基础

## 第一节　进位计数制

### 一、十进制数的表示

在生产劳动和日常生活中，我们最常用最熟悉的就是十进制数，它的数值部分是用十个不同的数字符号0，1，2，3，4，5，6，7，8，9来表示的，这些数字符号叫做数码。数码处于不同的位置(或数位)代表的意义是不同的。例如在2313这个数中右边第一位3代表个位，表示它本身的数值；右边第二位1是十位表示$1\times10$；右边第三位3是百位表示$3\times100$；右边第四位是千位表示$2\times1000$。因此这个数可以写成：

$$2313=2\times10^3+3\times10^2+1\times10^1+3\times10^0$$

一般地，任意一个十进制数N（为方便起见假设是正整数）都可以表示为：

$$N=k_n10^n+k_{n-1}10^{n-1}+\cdots\cdots+k_110^1+k_010^0$$

$$=\sum_{i=0}^{n}k_i10^i$$

其中$k_i$可以是0，1，2，3，4，5，6，7，8，9十个数码中的任意一个。它由N决定，n是正整数。其中10称为计数制的基数。

我们所说的进位计数制简称进位制，就是说它是按进位的方式计数的。所谓进位制的基数，就是在该进位制中可能用到的数码的个数。当基数为十，每位计满十向高位进一，即逢十进一，这种进位制，就是十进制。除了十进制，在生产和生活中还会碰到非十进制的计数制。比如时间，六十秒为一分，六十分为一小时，它是六十进制的；再如通常说的一打铅笔，就是十二枝，它是十二进制的。特别是基数为二的进位制，即二进制，它是计算机中广泛采用的进位制。

### 二、二进制数的表示

二进制数每个数位只可能取两个不同的数码“0”和“1”，而且是“逢二进一”的。为了熟悉二进制的表示，现就几个简单的数字列出十进制与二进制的对照表如表1—1。

用下列记号表示十进制数与二进制数的对应：

$(9)_十=(1001)_二$

$(16)_十=(10000)_二$

十进制与二进制数的对照 表 1—1

| 十进制 | 二进制 |
|---|---|
| $0_{10}$ | 0 |
| 1 | 1 |
| 2 | 10 |
| 3 | 11 |
| 4 | 100 |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| 8 | 1000 |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |
| 16 | 10000 |

括号右下角的数字表示进位制的基数。上列两式的意思就是：十进制的 9 表示为二进制等于 1001，十进制的 16 表示为二进制等于 10000。

对于二进制，同样可以写成展开式的形式。比如 11011 可以写成

$$11011=1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0$$

一般地，任意一个二进制数 N 都可以表示成：

$$N=k_n2^n+k_{n-1}2^{n-1}+\cdots\cdots+k_12^1+k_02^0$$

$$=\sum_{i=0}^{n}k_i2^i$$

其中 $k_i$ 只能取 0 或 1，它由 N 决定；n 为正整数。

可以看出十进制与二进制的展开式完全类似，如果在二进制的展开式中用(10)二代替 2，则两个式子形式上完全一样。

一般地说，如果用正整数表示进位制基数，则一个 R 进位制数 N 都可以表示为：

$$N=\sum_{i=0}^{n}k_iR^i$$

其中 $k_i$ 可以是 0，1，2，…，(R−1)中的任一数码；n 为正整数。

R=10，就是十进制数的表示形式；R=2，就是二进制数的表示形式。

从以上分析可以看出，进位制有两个主要的共同规律：

1．每一个进位制数都有一个固定的基数 R，它的每一数位可能取 R 个不同的数码，而且是“逢 R 进一”的，即每一位计满 R 就向高位进一。

2．进位制数都能写成 $N=\sum_{i=0}^{n}k_iR^i$ 形式的展开式，它的每一个数码 $k_i$ 对应于一个固定的值 $R^i$，$R^i$ 称为 $k_i$ 的权。上展开式也称为进位制数按权的展开式。

比如，二进制数 1011 自左至右，每一位数字对应的权分别为 $2^3$，$2^2$，$2^1$，$2^0$

还要指出一点，对 R 进制，若小数点向左移一位（对整数如果最低为零即去掉最低位的零）则等于原数减小 R 倍，即乘了$\frac{1}{R}$；若小数点向右移一位（对整数即在最低位添一个零），则等于原数扩大了 R 倍，即乘了 R。对二进制，相应地为减少二倍，即乘了$\frac{1}{2}$，或增加了二倍，即乘了 2。

### 三、二进制的特点

在数控系统中究竟采用什么样的进位制，取决于该进位制在机器设计制造上是否容易实现，是否计算简便，是否节省器材。前面分析了不同进位制的共同规律，但由于各进位制的基数不同，因而也就产生了各自的特殊性。下面看看二进制有什么特点，从而进一步了解为什么二进制在机器中被广泛的使用。

首先，二进制只有两个数码 0 和 1，因此二进制的很大一个优点，就是它的每一数位都可以用任何具有两个不同稳定状态的元件来表示。一般说来，制造具有两个稳定状态的元件比制造多稳定状态的元件容易得多，可以找到很多这样的元件具有这样的特性。比如氖灯的亮和灭，继电器的闭合和断开，晶体管的导通和截止、只要规定其中的一种状态表示“1”，另一种状态表示“0”，就可以表示二进制了。

由于采用二进制，在机器中，数的存贮和传送，也就可以用简单而可靠的方式进行，如脉冲的有无，电位的高低。

其次，二进制的数运算简单。在进行简单的算术运算时，必须要记住两个数的和及乘积的表。如果是十进制的数，必须要记住$\frac{10(10+1)}{2}$个和与积，才能进行计算。一般地说，R 进制数就要记住$\frac{R(R+1)}{2}$个和与积，对十进制来说，需要记住 55 个和与积（九九表），因此，用这样的计算法设计机器的运算器就很庞大，控制线路也很复杂。

而用二进制，则 R＝2，所以

$$\frac{R(R+1)}{2}=3$$

这样的和数表与积数表就特别简单，因为只有两个数字进行相加或相乘，很容易得到它们。

加法表为：

0＋0＝0

0＋1＝1＋0＝1

1＋1＝10

乘积表为

0×0＝0

0×1＝1×0＝0

1×1＝1

下面举几个例子，看看二进制正整数的运算规则：

```
   1 1 1          1 1 0 1
 + 0 1 1        - 1 1 0
-------        -------
 1 0 1 0          1 1 1
```

加法的特点是“逢二进一”，减法的特点是“借一做二”。

```
       1 0 1 1 1
   ×       1 0 1 0
-----------------
         0 0 0 0 0
       1 0 1 1 1
     0 0 0 0 0
   1 0 1 1 1
-----------------
 1 1 1 0 0 1 1 0
```

```
                1 1 0 0 1 ……商
            ---------------
111 ) 1 0 1 1 0 0 0 1
        1 1 1
      -------
        1 0 0 0
          1 1 1
        -------
            1 0 0 1
              1 1 1
            -------
                1 0 ……余数
```

还要说明一点，在一些专用的机器中还经常采用十进制。但要制造有十种状态的元件却是一件困难的事，因此在这种机器中仍然是用两种状态的元件。为了表示十进制数中的一位数，需要用四位二进制数码，因而就要用四个双稳态元件。但四个双稳态元件本来相应地有

十六种不同状态，可以代表十六个不同的二进制数；现在表示十进制数的一位数，只利用了十种状态，表示了十个不同的数。在这个意义上讲，在设备使用上，利用二进制比十进制要节省。

另外，由于采用二进制，就可以使用逻辑代数（或称开关代数、布尔代数），这就为机器的逻辑设计提供了便利的工具。

正因为二进制有上述优点，故在计算机中被广泛地采用着。但二进制也有其不足之处，一方面书写起来很大，同一个数用二进制表示要比用十进制表示位数要多，粗略地讲，前者约为后者的三倍。念起来不易懂。另一方面，由于人们习惯上是使用十进制，因此，送入机器的原始数据大多数是表示为十进制的，这就必须将十进制的原始数据转换成二进制，机器才能进行计算，而机器运算的结果也必须转换成十进制数。这个工作是由机器自动进行的。二进制数与十进制数之间的互相换算的规则，我们下一节进行一些讨论。

## 第二节　进位制数之间的转换

### 一、二进制数转换成十进制数

要将一个二进制数转换十进制数，只要将二进制数按权相加。

例如要将二进制数 101011 转换成十进制数。

$$(101011)_{二}=1\times2^5+0\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0$$

$$=32+8+2+1=43$$

这就要求我们能熟练地记住二进制各位的权，现列表如下。

| 位序 | n | …… | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 权 | $2^{n-1}$ | …… | $2^{14}$ | $2^{13}$ | $2^{12}$ | $2^{11}$ | $2^{10}$ | $2^9$ | $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| 十进制的值 | | …… | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

### 二、十进制数转换成二进制数

1. 比较法：把十进制数与二进制数各位权所代表的十进制的值，从高位到低位逐位加以比较求得。

例如将 351 化为二进制数。

比较　　351 > 256 = $2^8$　　二进制数第 9 位为 1

−256

95 > 64 = $2^6$　　二进制数第 7 位为 1

−64

31 > 16 = $2^4$　　第 5 位为 1

−16

15 > 8 = $2^3$　　第 4 位为 1

−8

7 > 4 = $2^2$　　第 3 位为 1

−4

3 > 2 = $2^1$　　第 2 位为 1

−2

1 = 1 = $2^0$　　第 1 位为 1

−1

0

所以　　$351 = 256 + 64 + 16 + 8 + 4 + 2 + 1$

$= 2^8 + 2^6 + 2^4 + 2^3 + 2^2 + 2 + 1$

写成二进制数为 101011111

2. 除二取余法：

下面以具体例子说明这种转换

给出一个十进制整数 725，把它转换成二进制整数。

设　　$(725)_{十} = (k_n k_{n-1} \cdots\cdots k_1 k_0)_{二}$

那么问题就在于要决定 $k_n$，$k_{n-1}$，……$k_1$，$k_0$ 的数值，因为是二进制，它们只能是 0 或 1。我们逐个地确定 $k_n$，$k_{n-1}$，……$k_1$，$k_0$ 的数值。

由于任何一个二进制数都可以写成按权的展开式，因此将上式写成

$$(725)_{十} = (k_n k_{n-1} \cdots\cdots k_1 k_0)_{二}$$

$$= k_n 2^n + k_{n-1} 2^{n-1} + \cdots\cdots + k_1 2^1 + k_0 2^0$$

二等式两边同时除以 2，得到：

$$\frac{725}{2} = 362 + \frac{1}{2} = k_n 2^{n-1} + k_{n-1} 2^{n-2} + \cdots\cdots + k_1 + \frac{k_0}{2}$$

由于等式两边整数与小数必须相等，所以 $k_0=1$，它正好是 $\frac{725}{2}$ 的余数。

整数部分相等所以得

$$362=k_n2^{n-1}+k_{n-1}2^{n-2}+\cdots\cdots+k_22^1+k_1$$

同样将该式两边除以 2，得到

$$\frac{362}{2}=181=k_n2^{n-2}+k_{n-1}2^{n-3}+\cdots\cdots+k_2+\frac{k_1}{2}$$

由小数部分相等知 $k_1=0$，即是 $\frac{362}{2}$ 的余数。

用类似的方法继续下去，可以将 $k_n$，$k_{n-1}$，$\cdots k_1$，$k_0$ 都确定下来。

整个过程与计算结果如下：

| | |
|---|---|
| 2 \| 7 2 5 | 余数 $=1=k_0$ |
| 2 \| 3 6 2 | 余数 $=0=k_1$ |
| 2 \| 1 8 1 | 余数 $=1=k_2$ |
| 2 \| 9 0 | 余数 $=0=k_3$ |
| 2 \| 4 5 | 余数 $=1=k_4$ |
| 2 \| 2 2 | 余数 $=0=k_5$ |
| 2 \| 1 1 | 余数 $=1=k_6$ |
| 2 \| 5 | 余数 $=1=k_7$ |
| 2 \| 2 | 余数 $=0=k_8$ |
| 2 \| 1 | 余数 $=1=k_9$ |
| 0 | |

因此，转换的结果为：

$$(725)_{十}=(k_9k_8\cdots k_1k_0)_{二}=(1011010101)_{二}$$

十进制整数换算成二进制数的规则：

对十进制整数可不断地用 2 去除，将所得的余数（0 或 1）依次记为 $k_0$，$k_1$，$k_2\cdots$，一直到商等于零时为止，将最后一次除得的余数记为 $k_n$，那么 $k_nk_{n-1}\cdots k_1k_0$ 即为该数的二进制表示。

## 第三节　码　　制

### 一、机器数与真值

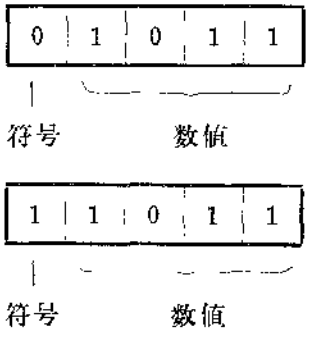
前面讨论了机器中普遍采用的二进制数，我们很自然会想：一个数的正负号在机器中是

如何表示的呢？

在机器中，数是存放在由寄存单元组成的寄存器中。二进制数码1和0是用寄存器单元的两种不同的状态（比如高低电平）来表示的，对于数的正号“+”或负号“−”也只能用这两种不同的状态来区别。因此，在机器中符号也被“数码化”了。数在机器中的一种简单的表示法就是，正数符号位用“0”表示，负数符号位用“1”表示。例如：

$$N_1=0.1011 \qquad N_2=-0.1011$$

在机器中就表示为：




为了区别原来的数与它在机器中的表示形式，将一个数（连同符号）在机器中的表示加以数值化称为机器数。如0.1011，−0.1011对应的机器数的一种表示就是0.1011，1.1011。这种表示法称为原码，记作$[0.1011]_{原}=0.1011$，$[-0.1011]_{原}=1.1011$。而把原来的数值称为机器数的真值。

那么，机器数有哪几种表示法呢？它们有哪些特点呢？机器数与真值之间存在什么关系？机器数之间的运算规律还和真值之间的运算规律一样吗？等等，这些都是需要分析和讨论的问题。

## 二、原码表示法

在前一段中提到了原码，它的定义如下：

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

式中$[x]_{原}$就是机器数，x即为真值。

原码有以下简单性质：

1. 若$x=0.x_1x_2\cdots\cdots x_n$，则$[x]_{原}=x=0.x_1x_2\cdots\cdots x_n$，且$0\leqslant[x]_{原}<1$

若$x=-0.x_1x_2\cdots x_n$，则$[x]_{原}=1-x=1+|x|=1-(-0.x_1x_2\cdots x_n)=1+0.x_1x_2\cdots x_n=1.x_1x_2\cdots x_n$，且$1\leqslant[x]_{原}<2$

所以，当x为正数时，$[x]_{原}$就是它自己，当x是负数时，$[x]_{原}$只要将小数点左边用“1”表示，小数点右边部分不变即可得到。

例如 $x=0.1011$

$[x]_{补}=0.1011$

$x=-0.1011$

$[x]_{补}=1.1011$

2. 设$[x]_{原}=x_0.x_1x_2x_3\cdots x_n$ 则：

$x_0=0$ 时 $0\leqslant[x]_{原}<1$ $\therefore\ 0\leqslant x=[x]_{原}<1$

$x_0=1$ 时 $1\leqslant[x]_{原}<2$ $\therefore\ -1<x=1-[x]_{原}\leqslant 0$

因此，在原码表示中，$x_0$ 为符号位。当 $x_0=0$ 时，表示$[x]_{原}$的真值 $x$ 是正数，当 $x_0=1$ 时，表示 $x$ 为负数，它所对应的原码机器数$[x]_{原}$是大于或等于 1 的。

3. 对于 0 可认为它是+0，也可认为是−0，这样

$[+0]_{原}=0.00\cdots\cdots0$

$[-0]_{原}=1.00\cdots\cdots0$

所以在原码表示中，0 有两种表示形式。

4. 若$[x]_{原}=x_0.x_1x_2\cdots\cdots x_n$，$x_0$ 表示符号位，满足 $x_0=\begin{cases}0 & x\geqslant 0\\ 1 & x\leqslant 0\end{cases}$

则有 $[x]_{原}=x_0+|x|$

原码表示简单易懂，而且与真值的转换方便，但是在做加法运算中就遇到了麻烦。当两个数进行相加时，如果是同号，则数值相加，符号不变；如果是异号，数值部分实际上是相减，而且还必须比较两个数那个绝对值大，才能确定谁减谁，最后再加上绝对值大的数的符号。这件事在手算时是容易解决的，但是，要在机器中进行，为了判断同号还是异号，比较哪个数值大，就要增加机器的设备或运算时间，为了更好地解决这些矛盾，人们找到了更适合于计算机进行运算的新的机器数表示法。

### 三、补码表示法

毛主席教导我们：“**人类总得不断地总结经验，有所发现，有所发明，有所创造，有所前进。**”（转引自周总理在三届人大第一次会议上的政府工作报告）为了解决原码做减法运算遇到的麻烦，人们又前进了一步，用一种新的码代替原码，将负数转化成正数，将减法转化成加法。

让我们先看一个例子。齿轮计算机的数字轮如图 1−1 示，轮子分成十等份，每个等份区域对应一个数，按顺序为 0，1，2，3，4，5，6，7，8，9。如果要做加法，如 5+3？则先把字轮“5”对准观察孔，然后按箭头方向（顺时针）转动三格，观察孔上看到的“8”就是答案。如果要计算 8+(−6)=？

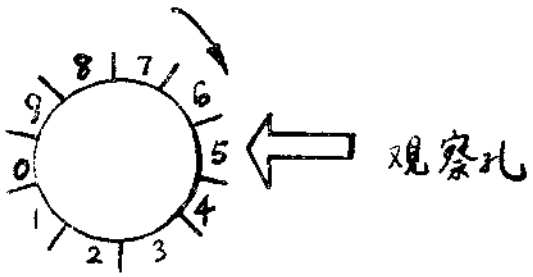



图 1—1 数字轮

这时实际上是要做减法 8－6＝?

可以将 8 转到观察孔位置，然后按箭头的相反方向（逆时针）转动 6 格，观察孔上便可看到结果为 2。但是如果数字轮不能向逆时针方向转动，而向正方向转动 4 格，在观察孔上看到的也是 2。事实上，现在是变成了

$$8+4=12$$

而“1”是进位，用一个数字轮表示不出来，而自动地丢失了。这样，就得到了所需的 8－6 的结果。再分析一下，可以看出－6 与 4 存在着一定的关系，10＋(－6)＝4，即 4 是 －6 加 10 后得到的。在加上的数（称为模数）满足一定的关系，而且将进位丢掉的情况下，正数加负数可以转化为正数加正数。在这里把 4 叫做－6 对 10 的补数，用下面的数学式表示。

$$-6=4 \quad (\mathrm{mod}10)$$

在机器中同样可以这样做，设 a＝＋0.1011，b＝－0.1001，求 a＋b，实际上是要做减法

$$\begin{array}{r} 0.1011 \\ -0.1001 \\ \hline 0.0010 \end{array}$$

在机器中数的表示是有限位的，类似于数字轮，在这里一旦出现 2，即 10.0…0，在机器中自然丢掉。因此将 b 加 2，变成－0.1001＋10＝1.0111，然后作加法：

$$\begin{array}{r} 0.1011 \\ +1.0111 \\ \hline \boxed{1}0.0010 \end{array}$$

比较一下计算结果，与上面相差 2，只要将小数点左边第 2 位丢掉不要（在机器中自然丢掉），则结果就完全一样。也就是说，将负数加 2 之后，就能把正数加负数变成正数加正数。这种负数的表示法，就是补码表示。

即 $1.0111=-0.1001 \quad (\mathrm{mod}2)$

也就是说在模 2 的意义下 1.0111 与 −0.1001 是相等的。

显然，对于一正数，如 0.1010，加上 2，再丢掉小数点左边第 2 位，仍是其本身。

即　　$0.1010=0.1010 \pmod 2$

因此补码的定义是

$$(x)_{补}=\begin{cases} x & 0\leqslant x<1 \\ 2+x & -1<x<0 \end{cases}$$

例如　　$x=-0.1011$　　$(x)_{补}=0.1011$

$x=-0.1011$　　$(x)_{补}=2+x=10-0.1011=1.0101$

下面对补码进行一些分析和讨论。

1. 补码的简单性质

① 设 $(x)_{补}=x_0.x_1x_2\cdots\cdots x_n$，那么

当 $0\leqslant x<1$　　则 $0\leqslant(x)_{补}=x<1$　　$\therefore\ x_0=0$

当 $-1\leqslant x<0$　　则 $1<(x)_{补}=2+x<2$　　$\therefore\ x_0=1$

反之也对。因此，如果 $x_0=0$，表示真值 x 为正数，$x_0=1$，表示真值为负数。$x_0$ 是补码的符号位，标志着 x 的符号。同时 x 与 $(x)_{补}$ 是一一对应的。即如果 $x_1=x_2$ 则 $(x_1)_{补}=(x_2)_{补}$。反之也对，即若 $(x_1)_{补}=(x_2)_{补}$，则 $x_1=x_2$。

② $(+0)_{补}=0.00\cdots0$

$(-0)_{补}=10-0.00\cdots0=10.00\cdots0=0.00\cdots0 \pmod 2$

2 在机器中自动丢失。所以在补码表示中，“0”的表示是唯一的，即 $(+0)_{补}=(-0)_{补}=0.00\cdots0$。

③ 补码的补码即是原码。

$$\{(x)_{补}\}_{补}=(x)_{原}$$

2. 负数补码的求法：

正数的补码就是其本身，而要求一个负数 x 的补码，要计算 10+x，实际上要做一次减法，不太方便。但可以看出，如 $x=-0.1011$ 时，它的原码 $(x)_{原}=1.1011$，如果将原码除符号位外，每位变反（即 1 变 0，0 变 1），1.1011 变为 1.0100 则

$$\begin{array}{r} 1.1011 \\ +0.0100 \\ \hline 1.1111 \end{array}$$

再在末位加 1，则得和为 2，即

$$1.1011+0.0100+0.0001=2$$

所以　　$1.0100+0.0001=2-0.1011=2+x=(x)_{补}$

$$[x]_{补}=1.0101$$

计算表明，求负数的补码即将其原码除符号位外，每位变反然后在所得到的数的末位加1（简称除符号位外变反加1）。

一般说来，设$-1<x<0$，$[x]_{原}=1.x_1x_2\cdots x_n$则

$$[x]_{补}=2+x=2-0.x_1x_2\cdots x_n$$

$$=1+(1-0.x_1x_2\cdots x_n)$$

注意到 $1=\sum_{i=1}^{n}2^{-i}+2^{-n}$ 上式可写成

$$[x]_{补}=1+\sum_{i=1}^{n}2^{-i}+2^{-n}-\sum_{i=1}^{n}x_i2^{-i}$$

$$=1+\sum_{i=1}^{n}(1-x_i)2^{-i}+2^{-n}$$

$$=1.\bar{x}_1\bar{x}_2\cdots\bar{x}_n+2^{-n}$$

其中 $$\overline{x_i}=\begin{cases}0 & x_i=1\\ 1 & x_i=0\end{cases}$$

即对于满足$-1<x<0$的任何负数，$[x]_{补}$等于$[x]_{原}$除符号位外变反加1。

反之，当$-1<x<0$时，已知$[x]_{补}$，同样可以通过对$[x]_{补}$除符号位外变反加1得$[x]_{原}$。

3. 变补器的工作原理

当$-1\leqslant x<1$时，若$[x]_{补}=x_0.x_1x_2\cdots x_n$则

$$[-x]_{补}=\bar{x}_0.\bar{x}_1\bar{x}_2\cdots\bar{x}_n+2^{-n}$$

分两种情况证明，首先若$0<x<1$，那么

$$[x]_{补}=0.x_1x_2\cdots n_n$$

$$[x]_{原}=0.x_1x_2\cdots x_n$$

明显地 $$[-x]_{原}=1.x_1x_2\cdots x_n$$

根据负数补码的求法，得到

$$[-x]_{补}=1.\bar{x}_1\bar{x}_2\cdots\bar{x}_n+2^{-n}$$

其次，若$-1<x<0$，那么

$$[x]_{补}=1.x_1\cdots x_n$$

因此有 $$[x]_{原}=1.\overline{x_1}\cdots\overline{x_n}+2^{-n}$$

由$-1<x<0$，可知若$x_1x_2\cdots x_n$不全为0，则$\overline{x_1}$，$\overline{x_2}\cdots\overline{x_n}$不全为1，所以

$$[-x]_{补}=0.\overline{x}_1\overline{x}_2\cdots\overline{x}_n+2^{-n}$$

$$[-x]_{补}=-0.\overline{x}_1\overline{x}_2\cdots\overline{x}_n\cdots2^{-n}$$

总之，当$-1<x<1$时，通过对$[x]_{补}$连同符号位变反加1，即可得到$[-x]_{补}$，将$[-x]_{补}$称为$[x]_{补}$的机器负数。由$[x]_{补}$求$[-x]_{补}$的计算过程称为对$[x]_{补}$变补。

### 四、反码表示法

在补码表示法中已说明了补码可以通过对原码变反加1得到。若只变反不加1，则可得到机器数的另一种表示法，即反码表示法。

反码定义如下：

$$[x]_{反}=\begin{cases}x & 0\leqslant x<1\\ (2-2^{-n})+x & -1<x\leqslant0\end{cases}$$

其中$n$代表数的位数。

例　$x=0.1101$

$[x]_{反}=x=0.1101$

$x=-0.1101$

$[x]_{反}=(2-2^{-4})+x=1.1111-0.1101=1.0010$

它正好等于$[x]_{原}=1.1101$除符号位外每位求反，因此要得到反码是容易的。

反码有下列简单性质：

1. 设$[x]_{反}=x_0.x_1x_2\cdots x_n$

当$0\leqslant x<1$时，$0\leqslant[x]_{反}<1$，且$x_0=0$，

$-1<x\leqslant0$时，$1\leqslant[x]_{反}<2$，且$x_0=1$。

反之也对。所以当$x_0=0$时表示真值$x$为正数，当$x_0=1$时，表示真值$x$为负数，$x_0$是符号位。

2. $[+0]_{反}=0.00\cdots0$

$[-0]_{反}=1.11\cdots1$

所以反码表示中“0”的表示法也不是唯一的。

综上所述，可以看出原码、补码、反码有两个共同点：

1. 它们本身主要是解决负数在机器中的表示法。对于正数它们都等于真值本身，对于负数有不同的表示。

2. 三种表示中小数点前的$x_0$都表示符号位，$x_0=0$表示真值为正数，$x_0=1$表示真值为负数。

三种表示法在机器中都被采用，但目前采用补码者较多，在这种机器中数均为补码表示，运算按补码规则进行。下面介绍一下补码的加减法运算。

## 第四节 运 算 方 法

由于机器数与真值的区别，产生了机器数本身的特殊的运算规则，下面讨论一下补码的加减法运算。

### 一、补码加法

原码的加法比较复杂，为了使加法单纯进行两数相加，以便机器只要一个加法器就能实现，引进了补码。补码表示可以将“正数加负数”转化为“正数加正数”。现在来证明一下这个重要的结论。

对模 2 补码，若 $-1<x<1$，$-1<y<1$，则

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

分为四种情况来证明：

1. $0\leqslant x<1$，$0\leqslant y<1$

所以 $0\leqslant x+y<1$

由补码定义有 $[x]_{补}=x$，$[y]_{补}=y$

所以 $[x]_{补}+[y]_{补}=x+y=[x+y]_{补}$

2. $0\leqslant x<1$，$-1<y<0$

$[x]_{补}=x$ $[y]_{补}=2+y$

所以 $[x]_{补}+[y]_{补}=x+2+y$

若 $x+y\geqslant 0$，则

$$x+y+2=x+y \quad (\mathrm{mod}2)$$

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

若 $-1<x+y<0$，则

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

3. $-1<x<0$，$0\leqslant y<1$

这种情况，只需将 2 中的 x，y 位置对调即可得。

4. $-1<x<0$，$-1<y<0$