微机原理

杨峰 卢洪武 柏静 王春静 翟临博 邵增珍 编著

济南出版社

微机原理

济南出版社

图书在版编目(CIP)数据

微机原理 / 杨峰,卢洪武,柏静编著. —济南:
济南出版社,2012.8
ISBN 978-7-5488-0525-0

Ⅰ.①微… Ⅱ.①杨… ②卢… ③柏… Ⅲ.①微型计算机—理论 Ⅳ.①TP36

中国版本图书馆 CIP 数据核字(2012)第 199000 号

责任编辑 张所建
封面设计 侯文英

出版发行 济南出版社
地　　址 济南市二环南路 1 号
邮　　编 250002
印　　刷 济南红河印业有限公司
开　　本 185 mm×260 mm
印　　张 16.25
字　　数 343 千
版　　次 2012 年 8 月第 1 版
印　　次 2012 年 8 月第 1 次印刷
定　　价 32.00 元
(济南版图书,如有印装错误,可随时调换)

前　言

随着微电子技术的日益进步，微型计算机向高性能的64位微机和适用于测控的单片微机两个方向迅速发展。单片微机是指在一块芯片上集成有CPU、ROM(或EPROM)、RAM、并行和串行I/O接口，以及定时器/计数器等多种功能部件的微型计算机，这种微型计算机也可称之为微控制器。它具有集成度高，可靠性高，性能价格比高，适应温度范围宽，抗干扰能力强，小巧、灵活，易于实现机电一体化等优点，现已广泛应用于检测、控制、智能化仪器仪表，以及生产设备自动化、家用电器等领域。

目前国内高校教授《微机原理》课程有两种情况：一种是以Intel 8088/8086CPU为例教授《微机原理》课程，另一种是以Intel的MCS－51单片机为例教授《微机原理》课程。它们的共同点是都能完成《微机原理》课程的理论教学任务；不同点是前种情况下教给学生的知识很难直接应用到实际中去，而后种情况下由于MCS－51单片机低廉的价格和齐全的功能，被更加广泛地应用到各行各业中去。为此，我们以MCS－51单片机为例编写《微机原理》教材，奉献给读者。

本书选材力求注重实用性、系统性、先进性，使读者能够轻松地学习微机的基本原理和具有应用单片机解决实际问题的能力。书中提供了大量实用电路，同时还提供了大量实用程序，便于读者学习和引用。鉴于本课程实践性很强，书中特给出实验指导，以辅导读者在学习过程中上机实习。每章末还附有小结和习题，有利于读者巩固和深化所学知识。

本书第一章由邵增珍编写，第二章和附录部分由柏静编写，第三章和第五章第一、二节由王春静编写，第四章由翟临博编写，第五章第三、四节由杨峰编写，第六章由卢洪武编写。全书由杨峰统一整理。

本书的出版得到了济南展雄电子有限公司的资助，在此一并致谢。

作　者

2012年6月于山东师范大学

目 录

第一章　数据基础及计算机概述

电子计算机是一种能对数据进行加工处理的信息处理机，具有存储、判断和运算能力，甚至可以模拟人类思维，做一些需要依靠智力才可以做的工作。1946 年，世界上第一台真正意义上的电子计算机 ENIAC（Electronic Numerical Integrator and Calculator）问世以来，计算机经历了电子管、晶体管、集成电路以及大规模和超大规模集成电路共 4 个时代。按照计算机的规模分，计算机可分为巨型机、大型机、中型机、小型机和微型机等，但从系统结构和基本工作原理上讲，微型计算机同其他类型的计算机没有本质区别，只是在体积、功能及性能等方面有所差异。

既然计算机具有强大的数据处理能力，那么，如何将人类习惯的数据以便捷易用的方式在计算机中表现，就成为非常基础且重要的工作。本章的主要内容就是介绍计算机中的数据表示、数据运算，以及计算机概述。

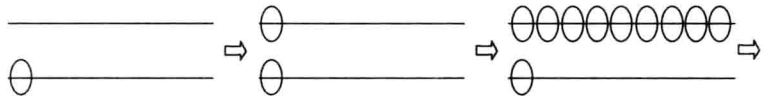
1.1　数值型数据的表示

计算机中处理的数据包括数值型数据和非数值型数据。数值型数据是指可进行大小比较的有数值含义的数据；非数值数据是指难以进行比较，只是用于表示某些含义的数据，例如字符、图像等数据。本节介绍数值型数据。

1.1.1　进位计数制

生活在原始社会的人类先祖，狩猎时需要记住狩猎数量，以便完成狩猎活动后的猎物分配工作。先祖们的狩猎活动一般是群体活动，狩猎数量较多，计数工作往往推举聪明且有威望的人来执行，我们暂且称之为计数员。最初，计数员通过在一条长长的野草藤上系草环来表示打到猎物，打到一只猎物就系上一个草环，打到两只猎物就系上两个草环，……，以此类推。这种计数方式当然没有什么大的问题，但很不方便。聪明的计数员后来想：能不能用多条野草藤来计数？考虑到人类主要用双手劳动，而两只手上共有十个手指，干脆采用 10 为单位。于是，计数员采用了图 1－1－1 的方法。

(a) 在一条草藤上系草环



(b) 在两条草藤上系草环

图 1-1-1　十进制计数方法图示

从图 1-1-1(a)可以看出,当前仅有一条草藤,计数员在草藤上系草环时,最多系 9 个,表示狩到 9 只猎物。当再有新的一只猎物被狩获时,计数员并没有在第一条草藤上继续增加草环,而是增加了一条新的草藤,同时做了如下工作:将第一条草藤上的所有草环全部去掉,仅在第二条草藤上新系一个草环,表示 10 只猎物。以此类推,当狩到 99 只猎物时,两条草藤就都系了 9 个草环了。如果再有新的一只猎物,则只能再增加一条草藤,同时做如下工作:将第一条、第二条草藤上的所有草环全部去掉,仅在第三条草藤上新系一个草环,表示 100 只猎物。如此循环。

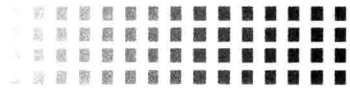
显然,不同草藤上的草环所代表的含义是不同的:第一条草藤上的一个草环代表 1 只猎物,第二条草藤上的一个草环代表 10 只猎物,第三条草藤上的一个草环就代表 100 只猎物了,这实际上就是"权"的概念。从加法的角度看,当第一条草藤已经计数满(9 个草环)时,如果再增加 1,则将该草藤的所有草环去掉,而在第二条草藤上系一个草环,这就是"逢十进一"的概念。

以上故事当然杜撰的成分居多,但从中可得出明显的"进位计数"、"进制"、"权"的概念。如果将每条草藤上最多系的草环的数目改成 1,就得到二进制的概念;改成 7,就得到八进制的概念,改成 15,就得到 16 进制的概念。

用若干个数字的有序组合来表示一个数值,从而形成一段有序的代码,如 $X_{n-1}\cdots X_0$ 就是一个由 n 位数字组成的数值。考虑整数,如果从 0 开始计数以得到各种数值,就存在一个由低位向高位进位的过程(考虑图 1-1-1 所表示的含义)。按照一定的进位方式进行计数的数值,称为进位计数制,简称进制。还是以我们常用的十进制为例,可用展开的方式来表示进位计数的思想,如下:

$$(N)_{10}=32\ 598=3\times10^4+2\times10^3+5\times10^2+9\times10^1+8\times10^0 \quad (1-1)$$

式(1-1)中,3、2、5、9、8 称为十进制中的数码;10^4、10^3、10^2、10^1、10^0 称为各数数位



的权值；“10”称为十进制的“基数”，也是十进制中所有数码的个数。

利用多项式可清晰表现进制中数位之间的关系。假设有某 r 进制数 $(S)_r$，其多项式可表现为：

$$(S)_r = (X_{n-1}X_{n-2}\cdots X_1X_0X_{-1}\cdots X_{-m})_r$$
$$= X_{n-1}r^{n-1} + X_{n-2}r^{n-2} + \cdots + X_1r^1 + X_0r^0 + X_{-1}r^{-1} + \cdots + X_{-m}r^{-m} \quad (1-2)$$

式(1-2)中，基数为 r；S 为 r 进制数；r^i 是各数位的权值；共有 $m+n$ 个数位，包括 n 位整数及 m 位小数。

在日常生活中人们一般采用十进制表示数值信息，但是计算机系统内部采用的却是二进制。之所以计算机内部采用二进制，是因为二态器件从物理上容易实现，而且运算规则简单。但是，人们习惯的十进制和计算机采用的二进制虽然表示的数值含义相同，但形式差别非常大，这就需要进行进制之间的相互转换。另外，在计算机内部，为了书写方便，人们在编程时常常用八进制、十六进制等来表示数值信息，这也需要进行数制转换。

1. 二进制

在二进制中，每个数位仅允许选择 0 或 1 两个数码，加法时“逢二进一”，减法时“借一当二”，基数 $r=2$。用多项式表示为：

$$(S)_2 = (X_{n-1}X_{n-2}\cdots X_1X_0X_{-1}\cdots X_{-m})_2$$
$$= X_{n-1}2^{n-1} + X_{n-2}2^{n-2} + \cdots + X_12^1 + X_02^0 + X_{-1}2^{-1} + \cdots + X_{-m}2^{-m} \quad (1-3)$$

其中 X_i 取值 0 或者 1。举例说明：

$$(1\,011.11)_2 = 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} = (11.75)_{10}$$

需要说明，为了区别数码属于哪种进制，往往利用小括号把数码括起来，并用下标注明。一般情况下，如果没有说明，可默认为十进制。

2. 八进制

在八进制中，每个数位可选择 0～7 中的数码，共 8 种选择，加法时“逢八进一”，减法时“借一当八”，基数 $r=8$。用多项式表示为：

$$(S)_8 = (X_{n-1}X_{n-2}\cdots X_1X_0X_{-1}\cdots X_{-m})_8$$
$$= X_{n-1}8^{n-1} + X_{n-2}8^{n-2} + \cdots + X_18^1 + X_08^0 + X_{-1}8^{-1} + \cdots + X_{-m}8^{-m} \quad (1-4)$$

其中 X_i 取值 0～7。举例说明：

$$(107.1)_2 = 1\times8^2 + 0\times8^1 + 7\times8^0 + 1\times8^{-1} = (71.125)_{10}$$

实际上，八进制的每个数码可以用三位二进制表示，它们之间的对应关系见表 1-1-1。

表 1-1-1　八进制同三位二进制的对应关系

八进制	二进制	八进制	二进制
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

[例 1-1-1] $(176)_8 = (001\ 111\ 110)_2$

以上例子还可以写为 176 O = 001 111 110 B。其中“O”是八进制的缩写,“B”是二进制的缩写。

3. 十六进制

在十六进制中,每个数位选择的数码个数为 16 个,即 0 ~ 15,书写时为 0 ~ 9 和 A、B、C、D、E、F,加法时“逢十六进一”,减法时“借一当十六”,基数 $r = 16$。用多项式表示为:

$$(S)_{16} = (X_{n-1}X_{n-2}\cdots X_1X_0X_{-1}\cdots X_{-m})_{16}$$

$$= X_{n-1}16^{n-1} + X_{n-2}16^{n-2} + \cdots + X_1 16^1 + X_0 16^0 + X_{-1}16^{-1} + \cdots + X_{-m}16^{-m} \quad (1-5)$$

其中 X_i 取值 0 ~ 15。举例说明:

$(6B.C)_{16} = 6 \times 16^1 + 11 \times 16^0 + 12 \times 16^{-1} = (107.75)_{10}$

同样道理,十六进制的每个数码可以用四位二进制表示,它们之间的对应关系见表 1-1-2。

表 1-1-2 十六进制同四位二进制的对应关系

十六进制	二进制
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

[例 1-1-2] $(36AB.C)_{16} = (0011\ 0110\ 1010\ 1011.1100)_2$

以上例子还可以写为 36AB.CH = 0011 0110 1010 1011.1100 B。其中“H”是十六进制的缩写。

4. BCD 码

计算机内部数的表示和运算以二进制为基础，而人类在生活中习惯利用十进制，这就需要采取某些措施进行转化。当前有两种方法可供选择。一种是实现二进制和十进制的相互转换：二进制转换为十进制利用“按权展开法”，利用式(1-2)的方式进行转换；十进制转换为二进制时，整数采用“除 2 取余”法，小数采用“乘 2 取整”法，后文将有介绍。另外一种是采用“二-十进制表示法”，也就是 BCD(Binary Coded Decimal)码。

所谓 BCD 码，其含义是用四位二进制数来表示一位十进制数，从左起高位的权值依次是 2^3、2^2、2^1、2^0，即 8、4、2、1，故这种编码又称为“8421 码”，但所能表示的数仅限于十进制的 10 个数码 0~9。

[**例** 1-1-3] $(189)_{10} = (0001\ 1000\ 1001)_{BCD}$

这里一定要注意，四位二进制数实际上有 16 种编码形式($2^4 = 16$)，但十进制数码只有 0~9，也就是说 BCD 码中只用了 16 种编码形式中的 10 个(从 0000~1001)，剩下的 6 个编码(1010~1111)没有使用。可以发现，在 0~9 的范围内，BCD 码形式同十六进制的二进制形式完全相同，但 BCD 码是“逢十进一”，而十六进制的四位二进制形式是“逢十六进一”，两者相差 6。因此，在进行加法运算时，对 BCD 码可先进行传统的二进制运算，然后再进行调整：如果每位的和小于等于 9，则不必修正；如果和大于 9，则做“加 6 调整”。具体步骤在后文有详细讲解。

表 1-1-3 是几种进制之间的对应关系。

表 1-1-3　几种进制之间的对应关系

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

1.1.2 进位计数制之间的转换

进制之间的转换可分为三种情况。

第一种是二进制、八进制、十六进制之间的转换，它们之间可以分段对应转换。以二进制转化为八进制为例，转换方法是：以小数点为左右起点，每三位为一组，左右不足三位用0补充，即可根据表1-1-1将二进制转换为八进制；如果要转换为十六进制，则是以小数点为左右起点，每四位为一组，左右不足四位用0补充，即可根据表1-1-2将二进制转换为十六进制。八进制、十六进制转换为二进制执行“一位八进制数对应三位二进制数，一位十六进制数对应四位二进制数”的原则即可完成。

如果需要在八进制和十六进制之间转换，我们采取的措施是以二进制为桥梁，即先转换为二进制，然后再由二进制转换为其他进制。

[例1-1-4]将101010111.01101 B转换为八进制和十六进制。

101010111.01101 B=101 010 111.011 010 B=527.32 O

101010111.01101 B=0001 0101 0111.0110 1000 B=157.68 H

[例1-1-5]将2763 O转换为十六进制。

2763 O=010 111 110 011 B=0101 1111 0011 B=5F3 H

第二种是十进制同BCD码(二-十进制)之间的转换，也可以分段进行。

[例1-1-6]将十进制数286转换为BCD码。

$286=(0010\ 1000\ 0110)_{BCD}$

第三种是二进制和十进制直接转换。因为这两种进制之间不存在直接的分段对应关系，因此需要某种转换算法，且整数转换和小数转换也不相同。

1. 二进制数转换为十进制数(包括整数和小数部分)

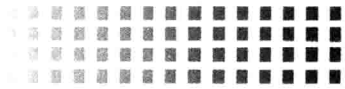
利用式(1-3)，即可实现二进制数到十进制数的转换。

2. 十进制整数转换为二进制整数

利用“除2取余”法，即可将十进制整数转换为二进制整数。其规律为：将十进制整数除以2，所得余数为对应二进制数的最低位；继续对得到的商除以2，所得的各个余数就是所求的二进制数的各位值；如此进行，直到商等于0为止，最后一项得到的余数作为二进制数的最高位(最左位)。利用竖式可方便实现以上算法，如下例。

[例1-1-7]将十进制整数98转换为二进制数表示。

2	98		
2	49	---	0
2	24	---	1
2	12	---	0
2	6	---	0
2	3	---	0
2	1	---	1
	0	---	1



则有 $98=(1100010)_2$

用横式表示以上转换过程，如下：

$$
\begin{aligned}
98 &= 2\times 49+0\\
&= 2\times(2\times 24+1)+0\\
&= 2\times(2\times(2\times 12+0)+0)+1)+0\\
&= 2\times(2\times(2\times(2\times 6+0)+0)+0)+1)+0\\
&= 2\times(2\times(2\times(2\times(2\times 3+0)+0)+0)+0)+1)+0\\
&= 2\times(2\times(2\times(2\times(2\times(2\times 1+1)+0)+0)+0)+0)+1)+0\\
&= 2\times(2\times(2\times(2\times(2\times(2\times \underbrace{0}+\underbrace{1)+1)+0)+0)+0)+0)+1)+0}
\end{aligned}
$$

3. 十进制小数转换为二进制小数

利用“乘 2 取整”法，即可将十进制小数转换为二进制小数。其规律为：将十进制小数乘以 2，所得到的整数即为二进制小数的最高位值（小数点右第一位）；将整数部分去掉，继续对乘积的所余小数部分乘以 2，所得整数就是二进制小数的次高位值；如此继续，直到去掉整数部分的乘积变为 0，或者满足精度要求时结束。

[**例** 1－1－8]将十进制小数 0.812 5 转换为二进制小数。

$$
\begin{aligned}
0.812\,5\times 2 &= 1.625 \quad \ldots\ldots 1\\
0.625\times 2 &= 1.25 \quad \ldots\ldots 1\\
0.25\times 2 &= 0.5 \quad \ldots\ldots 0\\
0.5\times 2 &= 1.0 \quad \ldots\ldots 1
\end{aligned}
$$

则 0.812 5 = 0.1101 B

另需说明，前文所述“满足精度”的原因是在转换过程中，有些十进制数并不能精确转化为二进制数，这种情况下，算法最后的整数部分永远不可能是 0，此时只能以满足精度为算法的结束条件，如下例。

[**例** 1－1－9]将十进制小数 0.681 25 转换为二进制小数。

$$
\begin{aligned}
0.681\,25\times 2 &= 1.362\,5 \quad \ldots\ldots 1\\
0.362\,5\times 2 &= 0.725 \quad \ldots\ldots 0\\
0.725\times 2 &= 1.45 \quad \ldots\ldots 1\\
0.45\times 2 &= 0.9 \quad \ldots\ldots 0\\
0.9\times 2 &= 1.8 \quad \ldots\ldots 1\\
0.8\times 2 &= 1.6 \quad \ldots\ldots 1\\
0.6\times 2 &= 1.2 \quad \ldots\ldots 1\\
0.2\times 2 &= 0.4 \quad \ldots\ldots 0\\
0.4\times 2 &= 0.8 \quad \ldots\ldots 0\\
0.8\times 2 &= 1.6 \quad \ldots\ldots 1\\
0.6\times 2 &= 1.2 \quad \ldots\ldots 1\\
0.2\times 2 &= 0.4 \quad \ldots\ldots 0\\
0.4\times 2 &= 0.8 \quad \ldots\ldots 0
\end{aligned}
$$

则有 $0.681\ 25 = 0.10101\ \dot{1}\dot{1}\dot{0}\dot{0}\ \dot{1}\dot{1}\dot{0}\dot{0}\cdots$

1.1.3 带符号数表示

计算机内部只能存储0、1信号，不能直接存储正负号，而在现实应用中，正负数值是经常使用的。具有正负号的数值称为带符号数，在日常应用中，我们也称之为真值。真值包含正负号(+、-)，以及用十进制或二进制表示的数值部分，其中正号可以省略。由于计算机无法直接存储正负号，需对正负号数字化(0、1化，用0表示正号，1表示负号)后，才可以存储带符号数。将带符号数的形式进行转化，变成可以在计算机中存储及应用的形式，得到的数称为机器数。机器数的形式有多种，最常用的包括原码、反码和补码。

1. 原码

原码是一种最为简单的机器数，其表现形式为：约定数码序列的最高一位为符号位，用0表示正数，用1表示负数；剩余的有效数值部分用二进制的绝对值表示。假设用8位二进制表示机器数，下面举几个例子。

[例1-1-10]

真值	$X_{原}$
+1011	0 000 1011
-1011	1 000 1011
+0.1011	0.1011 000
-0.1011	1.1011 000

可以看出，不管是整数还是小数，其原码的变换规则是相同的。需要注意的是，小数原码表示中的小数点在计算机存储时是不存在的，实际应用中加上小数点是为了方便使用。在计算机中，表示小数的机器数时，约定小数点放在符号位和最高数值位之间；表示整数的机器数时，约定小数点放在最低有效位之后。后面章节中讨论定点整数机和定点小数机时，我们还将详细介绍。

以整数为例，我们给出定点整数的真值和原码的转换公式。假设定点整数的原码序列为 $X_nX_{n-1}\cdots X_1X_0$，令 X 表示真值，则有

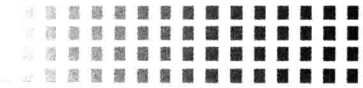
$$[X]_{原}=\begin{cases}X & 0\leqslant X<2^n\\ 2^n-X=2^n+|X| & -2^n<X\leqslant 0\end{cases}\qquad(1-6)$$

从式(1-6)可以看出，当 X 为0或正数时，$[X]_{原}$ 同 X 相同(将 X 的“+”变为0)；当 O 为0或负数时，$[X]_{原}=2^n+|X|$，其中 2^n 是符号位的权值(但是不具有数值含义，仅是一种表示形式)，在 $|X|$ 上加上 2^n 相当于将 X 的“-”变为1。需要注意，连同符号位，$[X]_{原}$ 总共有 $n+1$ 位，其中包括1位符号位，n 为有效数值位。

分析式(1-6)，我们得出如下结论：

(1)整数0有两种原码形式。根据符号不同，$[+0]_{真}=0\ 000\ 0000$ B，$[-0]_{真}=1\ 000\ 0000$。

(2)符号位不是有效数值位，不能参与算术运算，它们仅是人为约定的符号“0、1化”



的产物，在运算中需单独处理。

(3) $|X| < 2^n$。

原码表示中因为采用绝对值表示数值大小，非常直观。但是由于其最高位的符号位是硬性规定的，不能参与运算，这个二进制的加减运算带来不便。因此，原码目前很少用于算术运算。

2. 补码

算术运算是计算机基本且重要的功能，必须克服原码加减的缺点，让符号位也能直接参与运算。在计算机中广泛采用的方法是补码。补码是以模运算为基础的一种码制，具有良好的特点。以整数为例，补码的表示方法是：如果数是正数，则其补码形式同原码形式相同；如果数是负数，则首先得到负数的原码，然后除原码符号位外，其余所有位取反，最后在末位加 1。

以时钟为例引出补码的定义。我们常见的圆盘时钟是以 12 为计数循环的，表盘上的数从 1 到 12。12 点也可以看成是 0 点，所以也可认为表盘上的数是从 0 到 11 的。假设当前为 0 点，以时针为研究对象，分别进行如下操作：

a. 顺时针旋转 9 格，时针指向 9 点；

b. 逆时针旋转 3 格，时针也指向 9 点。

可见，上面两个操作的效果是一样的，都使时针指向了相同的位置。操作不同，而结论相同的原因是：圆盘时钟上的时针做圆周运动，其指向始终在 0 ~ 11 之间变化，即时针的读书是以 12 为模的。

将正数、负数的概念引入时钟转动，将顺时针旋转定义为正向旋转，相当于执行加法运算；将逆时针旋转定义为反向旋转，相当于执行减法运算。由此可知，上述 a 操作相当于在原来的基础上执行了 +9 操作，b 操作相当于在原来的基础上执行了 -3 操作。换句话说，在模 12 的前提下，+9 和 -3 具有相同的作用。更进一步说，在模 12 的前提下，-3 可以映射为 +9，它们互为模数。

另一个很有代表性且容易理解的例子是三角函数中用到的圆周。圆周从 1°到 360°，或者认为是从 0°到 359°。显然圆周的模是 360。按照上面时钟的思路，从 0°顺时针旋转 330°，和逆时针旋转 30°，效果是一样的。可以说，在模 360 的前提下，-30 可以映射为 330，它们互为模数。

从数学上看，以上两个例子都是有模运算。计算机运算时用到的寄存器也是有一定的字长限制的，因此它的运算也是有模运算。

补码定义：在有模运算中，假设模为 M，则某数 X 对该模的补数称为其补码，定义为：

$$[X]_{补} = M + X \quad (mod \quad M) \quad (1-7)$$

如果 X 是正数，则 M 可以作为正常的溢出量被舍去，则补码形式同原码相同。

以整数为例，我们从另一方面给出补码的计算方式。假定 $X_nX_{n-1}\ldots X_1X_0$ 为数 X 的补码序列，则有

$$[X]_{补} = \begin{cases} X & 0 \leqslant X < 2^n \\ 2^{n+1} + X = 2^{n+1} - |X| & -2^n \leqslant X < 0 \end{cases} \quad (mod \quad 2^{n+1}) \quad (1-8)$$

可以按照式(1-7)、(1-8)得到补码形式,但考虑原码和补码在形式上的差异,我们可以找到更为便捷的转换方法。

(1)由原码得到补码:正数的补码形式同原码形式相同;负数的补码是在负数原码的基础上,将所有的有效数值位(也称为尾数)取反,然后在末位加1(可简称为"求反加一")。

[例1-1-11] $[X]_{原}=0.0010\ 101$

$[X]_{补}=0.0010\ 101$

[例1-1-12] $[X]_{原}=1.0010\ 101$

变反 $1.1101\ 010$

末位加一 1

$[X]_{补}=1.1101\ 011$

(2)由原码得到补码:正数的补码形式同原码形式相同;负数的补码是在负数原码的基础上,符号位保持不变,有效数值位自右向左,找到第一个1,将这个1以及右边的所有0保持不变,这个1前面的所有位全部按位取反。

[例1-1-13] $[X]_{原}=1.0101\ 100$

$[X]_{补}=1.1010\ 100$

该方法是方法(1)的变形方法,适合手算负数的补码。请读者自行考虑方法的原理。

(3)由补码得到原码及真值:利用方法(1)和方法(2)对补码形式进行相同的运算,就可以得到对应的原码形式。再由原码变成真值,只需要根据符号位进行相应转换就可以了。

[例1-1-14] $[X]_{补}=1.1010\ 011$

$[X]_{原}=1.0101\ 101$

$X_{真}=-0.0101\ 101$

分析有关补码的特点,我们得到补码的一些性质。以整数为例,有

(1)补码的最高位 X_n 表示数的正负,1表示负数,0表示正数。这从形式上同原码是一致的,但补码的符号位是数值的一部分,是通过补码定义计算出来的,因此可以参与运算。

(2)0的补码形式只有一种,即0000 0000,这同原码中0有两种原码形式是不同的。

(3)负数的补码形式比原码的表示范围多一个数码组合。对定点整数 $X_nX_{n-1}\cdots X_1X_0$,负数补码的最小值为 -2^n,而负数原码的最小值是 $-(2^n-1)$。

(4)补码运算可以变减为加,有利于简化运算器的设计。

3. 反码

对原码的符号位保持不变,仅将所有的位数部分按位取反,即得到另一种机器数的表示形式,即反码。反码同补码类似,当数值为负数时,其形式比补码形式少加一个1。

假定 $X_nX_{n-1}\cdots X_1X_0$ 为数 X 的反码序列,则有

$$[X]_{反}=\begin{cases}X & 0\leqslant X<2^n\\ (2^{n+1}-1)+X=2^{n+1}-|X| & -2^n\leqslant X<0\end{cases}\qquad(1-9)$$

正数的反码同原码相同，负数的反码的转换规则是：符号位是1，尾数由原码按位取反。

[例1-1-15] $[X]_{原}=0.1010\ 001$

$[X]_{反}=0.1010\ 001$

[例1-1-16] $[X]_{原}=1.1010\ 001$

$[X]_{反}=1.0101\ 110$

反码一般是作为中间机器码形式使用，当前使用领域已经较少。

1.1.4　定点表示与浮点表示

实际应用的数值信息中，可能既有整数部分，又有小数部分。而且，在进行加减运算时，小数点的位置还需要对齐，这就提出了一个如何在计算机中表示小数点位置的问题。根据小数点位置是否固定，数的格式分为两种形式：定点数和浮点数。

1. 定点数的表示方法

在定点数中，小数点的位置是固定的。因为计算机只能存储0、1信息，所以小数点的位置一旦固定，就以约定的方式固定下来，并没有给小数点预留存储空间。有三种形式的定点数。

(1)无符号整数：这是将正号略去的整数形式，其所有数位全部用于表示数值大小，小数点默认在最右位(最低位)之后。假设无符号整数 $X_nX_{n-1}\cdots X_1X_0$，则其表示范围为 $0\sim(2^{n+1}-1)$。例如当 $n=7$ 时，其表示范围为0～255。

(2)带符号定点整数：假设 $X_nX_{n-1}\cdots X_1X_0$ 为整数序列，则 X_n 是符号位，小数点默认在最右位(最低位)之后，其所能表示的数值范围根据其机器数形式不同而不同。例如，原码定点整数的表示范围是 $-(2^n-1)\sim(2^n-1)$，补码定点整数的表示范围是 $-2^n\sim(2^n-1)$。

(3)带符号定点小数：假设 $X_nX_{n-1}\cdots X_1X_0$ 为整数序列，则 X_n 是符号位，小数点默认在最高位 X_n 之后，其所能表示的数值范围根据其机器数形式不同而不同。可以看出，带符号定点小数是纯小数。原码定点整数的表示范围是 $-(1-2^{-n})\sim(1-2^{-n})$，补码定点整数的表示范围是 $-1\sim(1-2^{-n})$。

从计算机的角度出发，在设计时仅考虑定点运算的计算机称为定点机，根据其支持的定点数形式又分为定点整数机和定点小数机。小数点的位置是隐含约定的，也就是说，小数点本身并不存在，没有固定的存储空间。

从实际考虑，数值信息中往往既包括整数部分，又包括小数部分。为了将数值信息规约为约定的数值形式，可考虑在编程过程中设定比例因子，将数值按照比例因子缩放。

2. 浮点数的表示方法

定点数表示方法简单，硬件实现成本也比较低，在低档机型中得到了广泛的应用。但是，由于小数点的位置是固定的，这使得定点数的数值表示范围和表示精度相互矛盾，在使用过程中受到较大限制。而在实际应用中，数值范围很大的数其精度要求往往不