微型计算机
原理与维护

本书编写组　编

中国商业出版社

微型计算机原理与维护

张　韶
檀晓红　编著
蒋翠清

中国商业出版社

图书在版编目(CIP)数据

微型计算机原理与维护/张韶等编著. —北京:中国商业出版社,1995.12
ISBN 7—5044—2911—2

Ⅰ.微… Ⅱ.张… Ⅲ.①微型计算机—理论②微型计算机—维修 Ⅳ.①TP360.1②TP360.7

中国版本图书馆 CIP 数据核字(95)第 22377 号

责任编辑:孙锦萍
特约编辑:张 辉
装帧设计:郭同桢

* * * *

中国商业出版社出版发行
(100053 北京广安门内报国寺 1 号)
新华书店总店北京发行所经销
蚌埠中发书刊发行有限责任公司激光照排
安徽省蚌埠市红旗印刷厂印刷
787×1092 毫米 16 开 印张:18 字数:409 千字
1995 年 12 月第 1 版 1996 年 2 月第 2 次印刷
印数:10 000—15 000 册 定价:19.50 元

* * * *

(如有印装质量问题可更换)

编 审 说 明

为适应各级各类大中专技校计算机专业的教学需要，我们组织编写了《微型计算机原理与维护》这本教材。

本书共分十二章。其原理部分详细介绍了微型计算机中数的表示方法、转换运行规律及基本组成电路和逻辑部件；初级模型机的工作原理及微型机的基本结构和指令系统；8086/8088汇编语言程序设计初步；微型机存储系统的存储原理及各种存储器的基本原理、特点和用途；输入输出设备与CPU的连接以及它们之间信息的传送方法；最后简要介绍了最具代表性的Inter主流系列——80186到Pentium的最新技术发展方向。其维护部分主要介绍了微机系统的日常维护和保养；提高微机系统平均无故障时间和延长使用寿命的方法；同时还介绍了计算机系统的一级维修技术。各章均有小结、习题、实例分析。

全书内容由浅入深，循序渐进，重点突出，难点分散，避免了复杂抽象概念的集中介绍，具有很强的系统性和实用性。

本书由张韶、檀晓红、蒋翠清编著。其中微机原理部分（第一章至第八章）由张韶、檀晓红编写，微机维护部分（第九章至第十二章）由蒋翠清编写。全书由张韶、檀晓红总纂。

本书编写工作得到全国部分大专院校专家教授的大力支持，书中引用和借鉴了许多国内外专家的著作和文献，由于种种原因，未能一一征求原著者意见，特此说明，并表示衷心的谢忱。

由于编者水平有限，书中难免存在缺点和错误之处，敬请广大读者不吝指评指正，以备不断修订完善。

《微型计算机原理与维护》编审组

1995年11月

目　录

五。十六进制的权是以 16 为底的幂，顺次分别称之为 0 权位、1 权位……等。

在微型计算机中，这些数制都是常用的，而二进制与十六进制更为常用。

二、二进制

(一)为什么要用二进制

电路中通常只有两种稳定状态：电压的高与低，电灯的亮与灭，电容器的充电与放电等，这两种不同的稳定状态可分别用 1 和 0 来表示，即二进制数码。所以，采用二进制，就可以利用电路来进行计数工作，而用电路来组成计算机，则有运算迅速、电路简便、成本低廉等优点。

(二)二进制数的表示

二进制数具有以下两个基本特点：

1. 具有两个不同的数字符号，即 0 和 1；

2. 逢二进位。

二进制数由数码 0 和 1 组成，每一位所表示的数值是该位的数码与该位权的乘积，如二进制数：

$$1\quad 0\quad 0\quad 1$$

该数的左数第一位的数码为 1，该位的权为 2^3，故这一位的值是：$1\times 2^3=8_{10}$

一个二进制数的值可以用它的按权展开式来表示，即：

$$1\,0\,0\,1_{(2)}=1\times 2^3+0\times 2^2+0\times 2^1+1\times 2^0=9_{(10)}$$

$$1\,1\,1\,1_{(2)}=1\times 2^3+1\times 2^2+1\times 2^1+1\times 2^0=15_{(10)}$$

所以，一个任意的二进制数可以表示为：

$$\begin{aligned}B_{(2)}&=B_{n-1}B_{n-2}\cdots\cdots B_2B_1B_{0(2)}\\&=B_{n-1}\times 2^{n-1}+B_{n-2}\times 2^{n-2}\cdots\cdots+B_2\times 2^2+B_1\times 2^1+B_0\times 2^0\\&=\sum_{i=0}^{n-1}B_i\times 2^i\end{aligned}$$

其中，$B_i(0\leqslant i\leqslant n-1)$为 0 或 1。

三、十六进制

(一)为什么要用十六进制

首先，我们来看一组等值的数：

$1\,0\,0\,0_{(2)}=8_{(16)}$　　(即八)

$1\,1\,1\,1_{(2)}=\mathrm{F}_{(16)}$　　(即十五)

$1\,1\,0\,0\,0\,0_{(2)}=30_{(16)}$　　(即四十八)

$1\,1\,1\,1\,1\,0\,0\,1_{(2)}=\mathrm{F}9_{(16)}$　　(即二百四十九)

暂且不管它们是如何转换的，只观察左右两边的数。左边是二进制数，表示起来既长又难于记忆，右边是十六进制数，它既可简化书写又便于记忆。当二进制数的位数越多时，越能体现出十六进制的优点。如：

$$1011\ 0110\ 0011\ 0101_{(2)}=\mathrm{B}635_{(16)}$$

注：以上书写格式中，在右下角注明的 2 或 16 是用来表示该数是二进制表示或十六进制表示。通常，可用字符来表示，即：*B*—二进制，*H*—十六进制，*D*—十进制，*O*—八进制。

另外，又由于目前大部分微机的字长是4的整数倍。基于以上原因，在很多情况下，数的表示广泛采用十六进制。

(二)十六进制数的表示

十六进制数具有以下两个基本特点：

1. 具有十六个不同的数字符号，即0—9和A～F十六个数码；

2. 逢16进位。

十六进制数由0～F十六个数码组成，且每一位所表示的数值是该位的数码与该位权的乘积。一个十六进制数的值可以用它的按权展开式来表示，如下列十六进制数：

$$3A_{(H)}=3\times16^1+10\times16^0=58_{(D)}$$

$$FB_{(H)}=15\times16^1+11\times16^0=251_{(D)}$$

故而，一个任意的十六进制数$D_{(H)}$可表示为：

$$\begin{aligned}D_{(H)}&=D_{n-1}D_{n-2}\cdots\cdots D_2D_1D_0{}_{(H)}\\&=D_{n-1}\times16^{n-1}+D_{n-2}\times16^{n-2}+\cdots+D_2\times16^2+D_1\times16^1+D_0\times16^0\\&=\sum_{i=0}^{n-1}D_i\times16^i\end{aligned}$$

其中，$D_i(0\leqslant i\leqslant n-1)$为1～9或A～H十六个数码中的任一个。

四、数制间的转换

(一)二进制与十进制之间的转换

1. 二进制转换成十进制

任意一个二进制数$A(a_{n-1}a_{n-2}\cdots a_2a_1a_0, a_i=0$或$1, 0\leqslant i\leqslant n-1)$所表示的实际值$F$是按权相加得到的，即$F_{(D)}=a_{n-1}\times2^{n-1}+a_{n-2}\times2^{n-2}+\cdots+a_0\times2^0$，所以任意一个二进制数转换成十进制数时，只须将该二进制数各位的权乘以各位的数再加起来即可。例如：

	1	0	1	0	1	1
权	2^5	2^4	2^3	2^2	2^1	2^0
乘积	32	0	8	0	2	1

累加和：43

故：$101011_{(B)}=43_{(D)}$

由此可以看出，任意的N进制转换成十进制的方法：将N进制数各位的权乘以各位的数码值，再加起来就可得到该数的十进制数值。

2. 十进制转换成二进制

一般可用下列方法求一个十进制数的二进制代码：

将这个十进制数一次次地被2除，直到除得的商为0，那么每一次得到的余数从后往前排列得到的代码就是该十进制数的二进制代码。

例如：

2 ⌊ 13	1	↑
2 ⌊ 6	0	｜
2 ⌊ 3	1	｜ (由下往上可写出二进制代码)
2 ⌊ 1	1	｜
0		↓

则：$13_{(D)}=1101_{(B)}$

由此，不难总结出，将十进制数转换成其它 N 进制的方法：将该十进制数一次次地被 N 除，直到所得的商为 0，然后将每次除得的余数从下向上排列所得的代码就是这个十进制数的 N 进制代码。

（二）二进制与十六进制间的转换

在机器中，数并不是用 16 进制表示的，由于以上提到的理由，机器中的数仍是二进制表示的。由于二进制与十六进制之间的特殊关系，即 $2^4=16$，于是，一位十六进制数可以用四位二进制代码来表示，而且一位十六进制数与四位二进制数的 0 和 1 代码的排列组合存在着一一对应的关系。如表 1—1 所示。

故而，二进制数和十六进制数之间的转换既简捷又方便。

表 1—1 二进制、十进制、十六进制代码对照表

十进制	二进制	十六进制	十进制	二进制	十六进制
0	0	0	8	1000	8
1	1	1	9	1001	9
2	10	2	10	1010	A
3	11	3	11	1011	B
4	100	4	12	1100	C
5	101	5	13	1101	D
6	110	6	14	1110	E
7	111	7	15	1111	F

1. 十六进制转换为二进制

方法：将这个十六进制数的每一位代码转换为四位二进制数，由高位向低位排列，即是它的二进制数。

如：

7	2	B	F	(H)
⇓	⇓	⇓	⇓	
0111	0010	1011	1111	(B)

即：$72BF_{(H)}=111\ 0010\ 1011\ 1111_{(B)}$

在转换中，每位十六进制代码一定要用四位二进制表示，如 $2_{(H)}=0010_{(B)}$ 不能写成 $10_{(B)}$。

2. 二进制转换为十六进制

方法：将二进制数的代码从右往左每四位分一组，最左边的代码不足四位时在前面补 0，然后将每组四位二进制代码转换成相对应的十六进制数码即可。

如：1101111100011 可转换为：

0001	1011	1110	0011
⇓	⇓	⇓	⇓
1	B	E	3

即：$110111110\ 0011_{(B)}=1BE3_{(H)}$

总之，数在机器中是用二进制表示的。但是，一个二进制数书写起来太长，易出错，并且目前大部分的微机字长均是四的整数倍，如十六位、三十二位、六十四位等。故在书写时可用十六进制表示。一个字节（八位）可用两位十六进制数表示，两个字节（十六位）可用四位十六进制数表示……等。书写方便且不易出错，又便于记忆。

第二节　计算机中数据的表示方法

在计算机中能直接表示和使用的数，可分为两大类，即数值数据和符号数据。符号数据又被称为非数值数据，用于表示一些符号标记，包括英文大、小写字母，数字符 0～9，专用字符十、一、*、/、(、)、[、]、,、;、:、'等等，汉字和图形，语音信息也属于符号数据。由于在计算机内表示任何数据都只能采用二进制的编码形式，因此，讨论各种数据的表示方法，实质上就是讨论它们在计算机内的组成格式和编码规则。

一、二进制编码

鉴于各种不同的符号数据在计算机内都是采用二进制的编码形式表示的，所以计算机能够识别和处理各种字符。那么，各种字符是怎样通过若干位二进制代码的组合来表示的呢？

(一)十进制数的二进制编码

虽然二进制数具有容易实现、可靠，且运算规律十分简单等优点，但二进制数并不直观，所以在计算机输入输出时仍然采用十进制数。不过，这样的十进制数要用二进制编码来表示。

一位十进制数可用四位二进制编码来表示，这样表示的方法很多，其中较常用的是 8421BCD 码。它选用 0000～1001 这十种编码值分别表示对应的 0～9 这十个数值。“8421”即四位二进制位的位权从高向低分别为 8、4、2、1。表 1—2 列出了一部分的编码关系。

表 1—2　BCD 码编码表

十进制数	8421BCD 码
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
10	0 0 0 1　0 0 0 0
11	0 0 0 1　0 0 0 1
12	0 0 0 1　0 0 1 0
13	0 0 0 1　0 0 1 1
14	0 0 0 1　0 1 0 0
15	0 0 0 1　0 1 0 1

8421BCD 码有十个不同的数字符号，且它是逢“十”进位，所以它是十进制数，但它的每一位数是用四位二进制码来表示的，因此称它为二进制码的十进制数(BCD——Birary Coded Decimal)。

把一个十进制数变成它的 8421 码串，是对十进制数的每一位单独进行的。如：

$$1592_{(D)}=0001\ 0101\ 1001\ 0010_{(BCD)}$$

反过来，BCD 码表示的十进制数转换成相应的十进制表示的数也类似，即将每一个 BCD 码(四位二进制数码)转换成相应的十进制数码。如：

$$0110\ 1000\ 0100\ 0000_{(BCD)}=6840_{(D)}$$

所以，只要掌握了 BCD 的十位编码，就容易实现十进制数与 BCD 码之间的转换。

但是要注意的是，BCD 码与二进制数之间的转换是不直接的，要先经过十进制数。如：

$$0001\ 0111_{(BCD)}=17_{(D)}=1\ 0001_{(B)}$$

$$1100_{(B)}=12_{(D)}=0001\ 0010_{(BCD)}$$

(二)字符的编码

1. ASCII 码

计算机中用得最多的符号数据是字符，字符数据主要是指数字、字母、通用符号、控制符号等，在微机内它们都被转换成计算机能识别的二进制编码进行工作。编码的方式可以有多种，但国际上普遍采用的一种编码是美国国家信息交换标准代码(American Standard Code for Information Interchenge)，简称为 ASCII 码。编码表见附录 I 。

ASCII 码是一个 7 位二进制的编码，它包括了 128 个不同的字符。其中 95 个编码所对应的字符为可显字符，另外 33 个编码所对应的字符为控制符。

在计算机中，二进制数的位又叫“比特”(Binary Digit 的缩写)，所以说一个 ASCII 码长为 7bit，而计算机中一个字节长为 8bit，故一个 ASCII 码存放在一个字节里，其字节的最高位为 0 或被计算机用作检测编码出错的奇偶校验位。

2. 奇偶校验码

在计算机中采用二进制编码，解决了计算机中对数或字符的存储、运算和传递的问题。在输入输出处理操作中，特别是在计算机网络环境中的数据传输过程可能会由于器件质量不可靠、线路工艺不过关、远距离传送带来的干扰或受来自电源、空间磁场影响等因素，使得信息在存取、传送和计算过程中难免会发生诸如“1”误变为“0”的错误。为了保证其准确性，计算机从硬件、软件上采取了很多措施提高机器抗干扰能力，一旦出错，要能及时检测并纠正错误。人们在传送字符之前要使所有被传送的字符具有同一特征，在信息传送到目的地之后，要对每个字符进行特征的检验。如果被接收的信息不具备原有的特征就表明字符在传送过程中发生了错误，这时需要重新发送直至正确为止。

“奇偶校验”是一种简单的最常用的检验方法，它包括奇校验和偶校验。

将 7 位 ASCII 码的高位再附加一位作为校验位，这个校验位取 0 还是取 1 的原则是：若是奇校验，附加一位后的八位编码中含“1”的个数是奇数；若是偶校验，则为偶数。

例如，假设约定是奇校验，信息编码是 $001\ 1010_{(B)}$，求校验位代码？

信息编码中有 4 个“1”，故校验位取“1”才能使扩展后的编码中含“1”的个数为奇数。所以，它的校验码为：$1011\ 1010$ 。

↑ 校验位

在计算机处理数据的各个地方都设有奇或偶校验器，以便对全部通过的代码进行检查。当检测信息编码中“1”的个数是偶数，与奇校验设定不符时，意味着该代码在传送过程中发生了错误，需要重新传送。

一个字符的 ASCII 码占用一个字节的第 0 位到第 6 位，空缺的第 7 位可以用来存放奇偶校验位，使 8 位代码成为一个整体用于信息的存储和传送。

(三)汉字的编码表示

汉字也是一种字符，但汉字的计算机处理技术远比拼音文字复杂。例如英语是一种拼音文

字，计算机键盘只需要配备 26 个字母键，并规定 26 个字母的编码（比如通用的 ASCII 码）就能方便地输入任意英文信息了。然而，汉字是象形字，字的数目很多，形状和笔划多少差异也很大，故不能用少数确定的符号把它们表示出来。那么就需要采取一些特殊的方式描述汉字，同时对汉字信息处理系统的硬件设备和软件也提出了特殊的要求。

1. 汉字输入编码方案

与其它字符一样，汉字也需要使用二进制编码之后才能被计算机接受。目前汉字编码的实质就是用字母、数字和一些符号代码的组合来描述，然后将代表汉字的一组代码存到计算机中去。现已推出的汉字编码方案有几百种，各有千秋，这些方案可归结为以下几种类型：

(1)汉字字音编码：这是一类以汉语拼音为基础的汉字输入编码。在汉语拼音键盘或经过处理的西文键盘上，根据汉字读音直接键入拼音即可。当遇到同音异形字时，屏幕显示重码汉字，再指定或输入附加信息，最后选定一个汉字。

(2)汉字字形编码：把汉字逐一分解归纳成一组基本构字部件，每个部件都赋予一个编码并规定选取字形构架的顺序，不同的汉字由于组成的构字部件和字形构架顺序不同，就能获得一组组不同的编码，以表达一个个特定的汉字。汉字字形与编码是一一对应的。

一般字形编码表示法，其字形的基本部件与输入键的对应关系比较复杂。

(3)汉字音形编码：根据汉语拼音和字形结构双重因素规定汉字编码。比如有的方案把一个汉字按声母、韵母、部首、起笔四个部分组成，各部分再由编码对照表确定。

(4)电报码：计算机沿用传统的电报码直接作汉字输入。每个汉字的电报码用四个字符描述，每组代码与汉字单一对应。输入时直接使用西文键敲入一组组电报码。

(5)整字编码：专门设计直观的汉字整字大键盘。一般设置常用汉字和符号 3000 个左右，直接在键盘上呈矩阵排列，每个汉字占一个键，类似中文打字机。操作人员见字打键或用光笔触盘面选取汉字或符号，机器自动将所选汉字在盘面上位置信息进行编码，产生作为这个汉字的编码送入计算机。

上述每一种类型都有许多不同的实现方案、改进方法和编码规则以及相应的软、硬件设备。目前，中文信息处理系统已经研究成功具有支持多种输入输出编码的通用输入输出模块，用户可以方便地使用自己所需的输入输出方法。

2. 国标码和汉字内部码

为了能在不同的汉字系统之间交换信息，并能高效率高质量地共享运行，近年来国家推出了一系列有关中文信息处理的标准。1981 年我国制定了 GB2312－80 国家标准信息交换用汉字编码字符集（基本集）——简称国标码，以及若干辅助集。国标码收集、制定的基本图形字符有 7 千余个，每个图形字符都规定了二进制表示的编码，每个编码字长为两字节，每字节内占用 7bit 信息，最高位补 0。如汉字“啊”的国标码的两个字节分别为 0011 0000 和 0010 0001，编码为 $3021_{(H)}$。

汉字内部码是汉字在计算机内部存储、运算的信息代码，内部码的设计要求与西文信息处理有较好的兼容性。前面讲到，普通的西文信息编码是采用 0～127 这 128 个编码值，当用一个字节存放一个字符时，字节最高一个二进制位为 0。可以规定，这一最高位的值为 1 时，该字节中的内容被理解为汉字编码，这时最多也只能有 128 个编码，根本满足不了具有上万字符的汉字编码的需要，必须将单字节扩充为多字节。如果给国标码的每字节最高位加“1”，作为汉字标识符，就成为一种机器内部表示汉字的代码——汉字内部码。如上述的“啊”字的内部码则是

1011 0000 1010 0001，即 $B0A1_{(H)}$。

汉字内部码的特点可概述为：一是结构简短。一个汉字内部码只占两个字节，且能表达数千汉字和各种符号图形，同时又能节省计算机存储空间；二是汉字内部码便于和西文字符兼容。西文字符的 ASCII 码占一个字节，两字节的汉字内部码可以看成是它扩展的字符代码。在同一个计算机系统中，只要从最高位就能区分这两种代码。最高位是“0”则为 ASCII 码，最高位是“1”则是汉字内部码。

在某些应用领域，如实现计算机通信，每个字节最高一个二进制位常用作校验码，就与汉字表示方法产生矛盾，为此，有时也用三个字节表示一个汉字，即三个字长的汉字内部码，如王安系统的汉字内部码就是三字节的。此外，还有的汉字内部码设计成四字节的，可以描述更多的汉字字符。

二、数的机器码表示

（一）机器数与真值

数值数据在计算机中的表示形式广义上统称为机器数。上面提到的二进制数，都没有涉及到符号问题。但是，在机器中，数值数据是有正、负之分的。那么其符号是怎样表示的呢？在二进制中，当然可以用“+”和“−”来表示正、负符号，但计算机硬件能辨认的仅有 0 和 1 两个符号。为了处理简便，我们用最高位放置符号，并以“0”表示正，以“1”表示负，其余的低位放置二进制数值，这样连同符号位在一起的数就是机器数，它所对应的数值为机器数的真值。

例如：

机器数（二进制）		真值（十进制）
0010 1011	⟶	$+43_{(D)}$
1010 1011	⟶	$-43_{(D)}$
0111 1111	⟶	$+127_{(D)}$
1111 1111	⟶	$-127_{(D)}$
0000 0000	⟶	$+0_{(D)}$
1000 0000	⟶	$-0_{(D)}$

机器数有两个特点，其一是数据的符号位数值化；其二是二进制的位数受机器设备的限制。机器内部设备一次能表示的二进制位数叫做计算机的字长，一台机器的字长是固定的，字长 8 位叫一个字节（byte），现有机器的字长一般都是字节的整数倍，如字长为 8 位、16 位、32 位、64 位等等。

符号位经数值化后，就能更方便地对机器数进行运算，并能提高其运算速度。为了达到上述目的，计算机设计了多种符号位与数值一起编码的方案。最常用的机器数表示方法有三种：原码、反码和补码。

机器数的三种表示方法主要是针对负数而言。正数的表示都相同，即用“0”表示最高符号位，以下各位给出该正数的数值。如：

$$X=+105$$

则：

$$[X]_{原}=[X]_{反}=[X]_{补}=0110\ 1001$$

下面对这三种表示方法分别介绍。

(二)原码表示法

用机器的最高一位表示符号,余下各位给出数值绝对值的表示方法,为原码表示法。

设符号位为 X_0,X 的真值绝对值 $|X|=X_1X_2\cdots X_n$,则 X 的机器数原码为:

$$[X]_{原}=X_0X_1X_2\cdots X_n;\text{当 } X\geqslant 0 \text{ 时},X_0=0$$

当 $X<0$ 时,$X_0=1$

【例 1】 写出 $X=-1011_{(B)}$和 $X'=+1001_{(B)}$的原码

$$[X]_{原}=1\ 1011_{(B)}\qquad [X']_{原}=0\ 1001_{(B)}$$

【例 2】 写出 8 位原码表示的最大、最小整数机器数,指出 8 位原码能表示的数的范围。

8 位原码表示的最大整数是绝对值最大的正数:

$$\text{Max}[X]_{原}=[+111\ 1111]_{原}=0\ 111\ 1111_{(B)}$$

在这里最高位"0"是符号位,后面七位是有效数值部分。8 位原码表示的最小整数则是绝对值最大的负数。

$$\text{Min}[X]_{原}=[-111\ 1111]_{原}=1\ 111\ 1111_{(B)}$$

所以,8 位原码表示的整数范围是:

$$1\ 111\ 1111_{(B)}\sim 0\ 111\ 1111_{(B)}$$

即:

$$-127_{(D)}\sim +127_{(D)}$$

原码表示直接明了,而且与真值的转换方法简便。原码相加遵循基本的代数运算规则。如果符号相同,则绝对值相加,结果带上原来的符号位,如果符号位不同,则将绝对值较大的数值减去绝对值较小的数值,结果带上绝对值数大的那个数的符号位。显然,在符号相异时,用原码来运算就比较麻烦,即要选定符号,又要作相减运算。由于计算机的计算单元往往是一个加法器,而将其它的运算都转换成加法运算来实现。为了能在加法器上实现减法运算,我们引进了新的表示负数的方法——补码表示法。

(三)反码表示法及补码表示法

在讲述补码表示方法之前,先来看看机器数的另一种表示形式,即反码表示法。

一个负数的原码符号位不动,其余位取相反码就构成这个负数的反码表示形式。正数的反码与原码相同。

设 $[X]_{原}=X_0X_1X_2\cdots X_n$,则:当 $X_0=0$ 时,$[X]_{反}=[X]_{原}=X_0X_1X_2\cdots X_n$;当 $X_0=1$ 时,$[X]_{反}=X_0\bar{X}_1\bar{X}_2\cdots\bar{X}_n$

【例 3】 $X=-1011_{(B)}\qquad [X]_{反}=1\ 0100_{(B)}$

$X'=+1001_{(B)}\qquad [X']_{反}=0\ 1001_{(B)}$

"补"的概念在日常生活中屡见不鲜。以钟表校时为例,若现在的实际时间是八点整,而表上的时间为 11 点,要想得到正确的实际时间,调整时针的方法有两种:一是按逆时针方向回拨三小时,即 11−3=8;二是按顺时针方法再拨 9 小时,即 11+9=12+8,也是八点整。前一种方法为减法运算,后一种为加法运算,其效果是一样的。其中,第二种方法的 12 称为模,即一个数字系统的最大量程或此系统所能表示的最大数。运算中它可以自然丢失。于是,我们把+9 和−3 的关系说成是对模 12 互为补数。

我们再看两个十进制数运算:

$$79-38=41$$

$$79+62=141=100+41$$

如果使用两位数的运算器，做 79+62 时多余的 100 因为超出了运算器的两位数的范围而自动丢弃，即 79+62=141(mod 100)，这样做 79−38 的减法时，用 79+62 的加法同样得到正确结果。这里的 100 也称之为模。两位十进制数测量范围是 0～99，溢出量是 100，模就是 10^2=100，上述运算也称之为模运算，可以写作：

$$79-38=79+62\ (\text{mod}\ 100)$$

进一步写为 $-38=62\ (\text{mod}\ 100)$

此时便可说 −38 与 62 是对模 100 互为补数，即 −38 的补码(对模 100)是 62，一个负数用其补码代替同样可以得出正确运算结果。

计算机是一种有字长限制的数字系统，因此都是有模运算，超过模的运算结果都将溢出。n 位二进制整数的模是 2^n。

补码的定义：

设模是 M，X 是真值，则

$$[X]_{补}=\begin{cases}[X]_{原} & X\geqslant 0\\ M+X & X<0\end{cases}$$

【例 4】 设计算机的字长为 8，机器数真值 $X=-1011\ 011_{(B)}$ 求 $[X]_{补}$

$\because n=8$，模 $M=2^8=1000\ 0000_{(B)}$，$X<0$

$\therefore [X]_{补}=M+X$

$=1000\ 0000_{(B)}-1011011_{(B)}=10100101_{(B)}$

在这里 X 的补码最高位为 1，表示符号为负。对于二进制数还有一个更简单的方法由原码求得补码：

(1)正数的补码表示与原码一样，$[X]_{补}=[X]_{原}$

(2)负数的补码是将原码符号位保持“1”之后其余各位取其反码，末位加 1 便得到补码，即取其反码再加 1：$[X]_{补}=[X]_{反}+1$。

如上例中 $X=-1011011_B$，则

$$[X]_{补}=[X]_{反}+1=1010\ 0100_{(B)}+1_{(B)}=1010\ 0101_{(B)}$$

【例 5】 列表求出 ±0，±39，±127 及 −128 的 8 位二进制原码、反码和补码表示。

真值	原码(B)	反码(B)	补码(B)
+127	0 111 1111	0 111 1111	0 111 1111
−127	1 111 1111	1 000 0000	1 000 0001
+39	0 010 0111	0 010 0111	0 010 0111
−39	1 010 0111	1 101 1000	1 101 1001
+0	0 000 0000	0 000 0000	0 000 0000
−0	1 000 0000	1 111 1111	0 000 0000
−128	/	/	1 000 0000

上例举了两个特殊的数。真值 +0 和 −0 的原码及反码表示形式不同，但在补码中的表示是一致的。原码和反码表示范围是 −127～+127，而八位补码机器数可以表示 −128，但不存在 +128 的补码与之对应，由此可知 8 位二进制补码的表示范围是 −128～+127。

采用了二进制的补码，当我们遇到两个二进制数 A、$B(A>0, B>0)$ 相减时 $(A-B)$，可把

它们看成 $A+(-B)$，然后分别求得 A，$(-B)$的补码，就可由加法器组成的运算器进行补码相加的运算，但是要注意，这种运算之后所得的结果仍然是一种补码表示而不是真值，还需将补码转换成实际结果，即补码求真。实现这一个过程也不必做“求非加 1”的逆处理“减 1 求非”，只需对补码再来一次“求非加 1”就可以了，因为一个数 D 的补码 D'的补码仍然是 D，即 D 和 D'互补。这样我们在采用补码来完成减法运算的过程中就彻底摒弃了相减运算。

【例 6】 $[X]_{补}=10010100$

则 X 的值不等于$(-20)_D$，它的值为$[X]_{补}$再求补而得，即 $X=-1101100_{(B)}=(-108)_D$

【例 7】 $64_{(D)}-10_{(D)}=54_{(D)}$，现用二进制的补码运算。

【解】(1)$64_{(D)}$的二进制补码表示 $01000000_{(B)}$；

$-10_{(D)}$的二进制补码表示由 0001010 求非为 111 0101，再加 1 得 1110110，最后添上符号位得 $1\ 111\ 0110_{(B)}$

(2)进行补码相加运算

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

超过八位自然丢失，得到 0011 0110，这是一个补码表示。

(3)补码求真，因为符号位为 0，所以是一个正数，真值也就是补码本身，因此结果为正 $0110110_{(B)}=54_{(D)}$

【例 8】 $34_{(D)}-68_{(D)}=-34_{(D)}$，用二进制补码运算。

【解】(1)$34_{(D)}$的补码表示 $00100010_{(B)}$；

$-68_{(D)}$的补码表示 $1011\ 1100_{(B)}$；

(2)进行补码相加运算

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

(3)补码求真，结果的补码表示为 11011110，符号位为 1，所以是一个负数，去掉符号位再去求补，可得真值的绝对值。

101 1110 求非为 010 0001

010 0001 +1 =010 0010

所以结果为负，即 $1\ 010\ 0010_{(B)}=-34_{(D)}$

从上述两个例子的运算中可看出由于采取二进制的补码后，我们不仅摒弃了减法运算而代之以求非相加的过程，简化了计算机的设计，而且在这种二进制的补码表示中，符号位可连同绝对值部分一起参加相加运算，得到的结果自然包括符号位和绝对值的两部分表示，不必再去另做符号的判断选取工作，给运算带来了极大的便利。

第三节 定点数和浮点数

一、定点数

在原码或补码表示中，符号位的“0”和“1”是用来表示正负，使它们可同数值位一样遵循二进制运算规则并参加运算，运算结果的正负仍由相应符号位是“0”还是“1”来决定，因而原码及补码表示，不仅使计算机硬件能识别被处理数据的量值，而且能直接辨认和处理它的符号。上节讨论中，我们只考虑了整数情况，但实际运算中必然会涉及到小数。例如科学计算中用到正弦值时，它就是一个小于或等于 1 的小数，那么如何来表示一个小数点的位置呢？这完全是机器中的事先约定，而不包含在二进制数据表示之中。

定点数是计算机内最基本的一种数据表示。数的定点表示就是把数据中小数点的位置固定不变。由于定点位置不同，一般分为两类：定点纯整数（小数点固定在最低位右边的数称为整数）和定点纯小数（小数点固定在最高位左边的数称为小数）。一旦假定了小数点位置，处理中对数据都同样看待，小数点位置不再变化。

定点数可以表示为带符号的或不带符号的数。不带符号的数一般表示逻辑量或某些特征值，逻辑运算也是按位进行的。表示算术操作时，应用带符号的数，一般以左边最高位表示符号位。可以有原码定点表示，补码定点表示，还可以有反码定点表示。以八位一字节为例，常用定点表示如图 1—1。

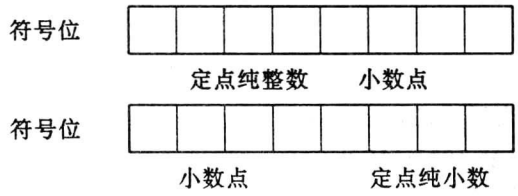

图 1—1

【例】 0.1001101 表示为：

0	1	0	0	1	1	0	1

−0.1001101 表示为补码形式：

1	0	1	1	0	0	1	1

定点表示的数据存放在计算机存储单元时并不带有小数点，如上例的八个二进制位中只有一位符号位，它们分别表示正或负和七位数值位而无小数点对应的信息位。小数点体现在机器处理数据时的规定之中。定点数的表示由于受到字长表示范围的限制，它所能表示的数值范围非常有限。在计算机进行定点运算时很容易产生溢出。因此，在科学研究和数据处理方面，目前大多采用的是浮点表示法，并进行浮点数运算。

二、浮点数

在计算机用作科学计算时，面临着数值范围及有效位数这两个基本问题。例如研究太阳中的某一物理现象，就涉及到电子和太阳的质量，写成定点小数的形式为：