微型计算机应用基础

WEIXING
JISUANJI
YINGYONG
JICHU

山东科学技术出版社

微型计算机应用基础

徐　萌

山东科学技术出版社
一九八六年·济南

责任编辑　孟爱平

微型计算机应用基础

徐　蕙

山东科学技术出版社出版

（济南市南郊宾馆西路中段）

山东省新华书店发行　　山东新华印刷厂潍坊厂印刷

787×1092 毫米 16 开本　18.25 印张　365 千字

1986 年 1 月第 1 版　　1986 年 1 月第 1 次印刷

印数：1—17,800

书号 13195·143　　定价 3.45 元

前　言

计算机是二十世纪最辉煌的科学技术成就。三十年来，计算机的发展和日益广泛的应用，对人类社会产生了深刻的影响。微型计算机具有功能强、体积小、寿命长、可靠性高、功耗小、价格便宜等特点，它一出现就奇迹般地登上了工业舞台，迅速形成以它为主导的信息工业，因而有希望取代传统工业而成为国民经济的支柱。这就迫使技术人员来了解它、使用它，大批原来不搞计算机的人员来学习计算机已成为时代的潮流。

本书从应用角度为微型计算机的初学者提供了通俗入门教材。它通过对Z80单板计算机的深入分析，来了解计算机的基本结构和工作原理，做到举一反三。在内容安排上由浅入深，循序渐近，遇到有关计算和概念都尽量讲清楚，便于初学者参考。

全书共分为八章。第一章介绍了计算机可以识别的数字和符号，主要是二进制及编码。第二章概述，讲述了微型计算机的发展、组成及应用，并阐述有关计算机软件及硬件的一些知识，使读者对什么是微型计算机系统，以及对它的过去、现在、将来有个轮廓了解。第三、四章介绍Z80 CPU及存贮器。这是组成微型计算机的核心部分，是硬件基础。这两章涉及到了许多计算机原理，具体分析计算机结构框图、各部件作用和相互关系，并从时间进程上讨论了计算机如何按拍节工作。在第五、六章中，全面介绍了Z80系统全部指令，分析程序的结构，并介绍一些典型程序。第七章讲述了接口电路及输入输出技术，对选配接口的原则以及通用接口芯片PIO、CTC、DAC和ADC的结构及使用也作了介绍。这些知识对组建一个微型计算机应用系统来说是必不可少的。第八章可以看作是全书的总结。通过对ZBUG分析，使读者体会软件和硬件的关系，接口电路的处理及编程的技巧。

本书曾作为教材在工程技术人员的微型计算机培训班中使用多次，现又做了系统、完整地修改和审订。在编写过程中，受到山东大学实验中心的领导及其他同志的诚挚帮助，在此表示衷心感谢。

徐　萌

1985.1.

目　录

第一章 准备知识

计算机的基本功能就是对信息进行变换处理。信息包括数字、文字和符号。所有电子计算机只能处理二进制数字，而八进制和十六进制数是二进制数的缩写方式。我们惯用的数制是十进制，因此首先要分析这些数制的特点及相互转换的关系。

计算机可以对数字进行计算，也可以对文字进行处理。所有字母、符号、汉字等都能用二进制的一组数表示并存放在计算机里。这些二进制数的集合就叫做编码，也叫代码。早在计算机出现之前，代码已经出现在电传、电报等邮政业务中，随着计算机在非数值计算领域里的应用不断扩展，编码的使用也就更加普及。

第一节 数 制

一、十进制数字系统

十进制数字系统 (Decimal Number System)是常用的一种数字系统，它起源于“屈指计数”，为人所习惯。大约公元 400 年印度数学家 Hidu 就发明了此系统。公元 800 年，阿拉伯人开始使用，后来习惯称作“阿拉伯”数字。

十进制计数的原则是“逢十进一”，表示数的大小不但和数本身取值有关，而且同它所处的位置有关，这也叫做“有权”(Weighing)计数。

十进制表示数的一般关系式为：

$$N = K_n(10)^n + K_{n-1}(10)^{n-1} + \cdots + K_0(10)^0 + K_{-1}(10)^{-1} + \cdots + K_{-m}(10)^{-m}$$

$$= \sum_{i=n}^{-m} K_i(10)^i$$

其中：10 为十进制的基数，K_i 为 i 位的选值，可以是：0；1；2；3；4；5；6；7；8；9 这十个数符中的任意一个。

对于整数，K_i 这个数在不同位置所表示的值不一样，它的大小由“权”来定，而“权”等于基数的 i 次幂。

例 1 十进制数 4603 可写成：

$$N = 4\times10^3 + 6\times10^2 + 0\times10^1 + 3\times10^0$$

$$= 4\times1000 + 6\times100 + 3\times1$$

$$= 4000 + 600 + 3$$

对于小数也完全一样，K_i 所在位置的“权”是基数的 i 次幂。

例 2 十进制小数 0.325 可写成：

$$N = 3\times10^{-1} + 2\times10^{-2} + 5\times10^{-3}$$
$$= 3\times0.1 + 2\times0.01 + 5\times0.001$$
$$= 0.3 + 0.02 + 0.005$$

任意十进制数包括整数和小数两部分，用小数点把它们分开。

例 3　278.94 可写成。

$$N = 2\times10^{2} + 7\times10^{1} + 8\times10^{0} + 9\times10^{-1} + 4\times10^{-2}$$
$$= 200 + 70 + 8 + 0.9 + 0.04$$

二、二进制数字系统

若用计算机直接处理十进制数字，则要求电子元件能表示出 10 个状态，但至今还未能找到一种器件可以反映 10 个不同状态。而反映 2 个状态，如开关的接通与断开；电位的高或低；电流的有或无；电极的正或负等电子器件是很容易做到的。因此，记数方式采用二进制的形式最简单，最经济，最可靠。二进制数值系统 (Binary Number System) 的应用，大大推动了计算技术的发展。

二进制表示数的一般关系式为：

$$N = K_n(2)^n + K_{n-1}(2)^{n-1} + \cdots + K_0(2)^0 + K_{-1}(2)^{-1} + \cdots + K_{-m}(2)^{-m}$$
$$= \sum_{i=n}^{-m} K_i(2)^i$$

其中：2 为二进制的基数，K_i 为 i 位的选值，只能是 0 或 1 这二个数符中任意一个。

二进制记数原则是“逢二进一”。二进制各位的“权”为基数的 i 次幂 (i 可为正或为负)。

各位权值见表 1—1 。

表 1—1　　权　值　表

$2^0=1$	$2^4=16$	$2^{-1}=\frac{1}{2}=0.5$	$2^{-5}=\frac{1}{32}=0.03125$
$2^1=2$	$2^5=32$	$2^{-2}=\frac{1}{4}=0.25$	$2^{-6}=\frac{1}{64}=0.015625$
$2^2=4$	$2^6=64$	$2^{-3}=\frac{1}{8}=0.125$	$2^{-7}=\frac{1}{128}=0.0078125$
$2^3=8$	$2^7=128$	$2^{-4}=\frac{1}{16}=0.0625$	$2^{-8}=\frac{1}{256}=0.00390625$

对于 8 位字长的微处理机 (如 Z80)，只能完成简单的算术运算，即加法和减法运算。而比较复杂的算术运算，象乘法和除法，则要用一段程序来实现。

(一) 二进制加法运算

加法规则见表 1—2 。

表 1—2　　加 法 规 则

被加数		加数		进位	和
0	+	0	=	0	0
0	+	1	=	0	1
1	+	0	=	0	1
1	+	1	=	1	0

例　10110101 + 00110111 = 11101100

被加数	1 0 1 1 0 1 0 1	1 8 1
加　数	+ 0 0 1 1 0 1 1 1	+ 5 5
和	1 1 1 0 1 1 0 0	2 3 6

（二）二进制减法运算

减法规则见表 1—3 。

表 1—3　　减 法 规 则

被减数		减数		借位	差
0	−	0	=	0	0
0	−	1	=	1	1
1	−	0	=	0	1
1	−	1	=	0	0

例　10110101 − 00110111 = 01111110

被减数	1 0 1 1 0 1 0 1	1 8 1
减　数	− 0 0 1 1 0 1 1 1	− 5 5
差	0 1 1 1 1 1 1 0	1 2 6

（三）二进制乘法运算

乘法规则见表 1—4 。

表 1—4　　乘 法 规 则

被乘数		乘数		乘积
0	×	0	=	0
0	×	1	=	0
1	×	0	=	0
1	×	1	=	1

例　1011 × 1001 = 1100011

被乘数	1 0 1 1	1 1
乘　数	× 1 0 0 1	× 9
	1 0 1 1	9 9
	0 0 0 0	
	0 0 0 0	
	1 0 1 1	
	1 1 0 0 0 1 1	

（四）二进制除法运算

除法规则见表 1—5 。

表 1—5　　除 法 规 则

被除数		除 数		余 数	商 数
0	÷	0	=	不定	不定
0	÷	1	=	0	0
1	÷	0	=	不定	不定
1	÷	1	=	0	1

例　1100011 ÷ 1001 = 1011

```
                1 0 1 1   商
除数  1 0 0 1 ) 1 1 0 0 0 1 1   被除数
              - 1 0 0 1
                1 1 0 1
              - 1 0 0 1
                  1 0 0 1
                - 1 0 0 1
                        0
```

从上述运算关系看出，二进制的运算同十进制运算是十分相似的。

三、二进制与十进制的关系

(一) 二进制转换十进制 (Binary To Decimal Conversion)

二进制转换成十进制的方法很简单，只要把二进制各位的数值与它本位的权相乘，然后各项相加就可完成。

例　把二进制数 101011.1001 转换成十进制数。

解　$(101011.1001)_B = 1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 + 1\times2^{-1}$
$+ 0\times2^{-2} + 0\times2^{-3} + 1\times2^{-4}$

$= 32 + 3 + 2 + 1 + 0.5 + 0.0625$

$= 43.5625$

为了区分不同进制，将数值括起来并加个下标，十进制一般不另加说明，二进制下标用 B 表示。

(二) 十进制转换二进制 (Decimal To Binary Conversion)

十进制转换成二进制通常使用连乘连除法。这种方法是把十进制的整数和小数分别处理，先把十进制整数部分转换成二进制整数，再把十进制小数部分转换成二进制小数。最后再把两部分合起来。

1．十进制整数转换成二进制

十进制整数转换成二进制用连除法。将十进制数除以 2，所得的余数就是对应的二进制最右一位的值，商数不为零就连续除以 2，依次得到一串余数，这就是二进制由高位到低位的值，具体描述如下：

(1) 将十进制数除以 2，保存余数。

(2) 若商不为零，则商取代原数，重复除以2，若商为零，转换结束，二进制数由余数组成，第一位余数是二进制最低位。

例　把十进制数725转换成二进制数。

```
2 |  7 2 5
2 |  3 6 2     余数等于1
2 |  1 8 1     余数等于0
2 |  9 0       余数等于1
2 |  4 5       余数等于0
2 |  2 2       余数等于1
2 |  1 1       余数等于0
2 |  5         余数等于1
2 |  2         余数等于1
2 |  1         余数等于0
   0           余数等于1
```

结果：$725=(1011010101)_B$

2．十进制小数转换成二进制

十进制小数转换成二进制用连乘法。十进制小数部分乘2，产生进位，这进位数就是二进制小数部分的第一位，剩下小数部分继续乘2，直到小数部分为零，或达到足够的二进制小数位数为止，具体描述如下：

(1) 将十进制小数部分乘2，乘积的整数部分保存，这数是1或零，是二进制小数部分的第一位。

(2) 把乘积的小数部分再乘2，若乘积的小数部分不为零，则连续乘2，保存整数部分。若乘积为零，则转换结束，二进制小数由进位组成，第一次进位值为小数点后的第一位。有的十进制数转换成二进制小数时，二进制小数位数可能很多，则根据需要取到足够的位数为止，也就是说，如果用一定长的二进制位来表示十进制数，可能是近似的，存在误差。

例　把十进制小数0.6875转换成二进制小数。

```
  0.6875
×      2
--------
  1.3750      进位=1
×      2
--------
  0.750       进位=0
×      2
--------
  1.50        进位=1
×      2
--------
  1.0         进位=1
```

结果：$0.6875=(0.1011)_B$

四、定点与浮点

送到计算机里处理的数，有时是很大的，既有整数又有小数，而且还对处理精度有要求，如何使得在一定的存贮区间内存放更多的数，如何保证数的精度和处理方便，这就要求根据具体情况选择不同的表示方法。

任何二进制数，都可以用指数形式表示：

$$10011.101 = 0.10011101 \times 25$$

用一般形式表示：

$$N = \pm d \times 2^{\pm p}$$

d 称为数的尾数，它的位数多少决定了数的精度。p 称为数的阶码（二进制数表示），它的位数决定了数的范围。

（一）定点表示 (Fixed point Notation)

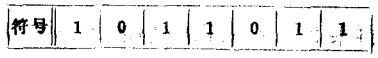
参加运算数的阶码是恒定的。取 p 为常数，这样小数点的位数固定不变，因此叫"定点"表示。通常取 p = 0，所以 $2^{\pm p}$ 项为 1。

$$N = \pm d$$

为了方便，使尾数 d 表示纯小数或纯整数。这样，数在送入计算机之前先要经过处理，除以固定数或乘以固定数，使计算机处理数的过程中其数或是纯小数或是纯整数这个固定的数叫做比例因子，比例因子要选择适当。

为了形象的表示定点数的形式，我们用"位网格"来表示。

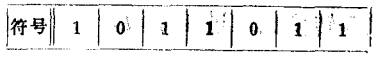
例 1　　N = 0.1011011　　（纯小数表示）



符号部分　　数值部分（尾数）

位网格的第一位是符号位，取 0 或取 1 表示数为正或为负。后面是尾数，小数点看作放在尾数的前面。

例 2　　N = 1011011　　（纯整数表示）



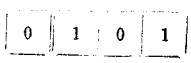
这同例 1 表示形式一样，只是我们认为小数点是在后一位的后边。

（二）浮点表示 (Floating Point Notation)

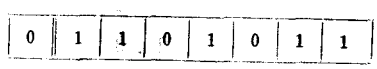
参加运算数的阶码 p 是变的，随着阶码值不同，小数点的位置是变化的。因此在运算时必须考虑小数点的位置，也就是要对阶。对阶指的是使两数的阶相等，这样运算要比定点复杂的多。但是浮点表示可以大大扩大数的表示范围。

在用浮点表示时，既要表示尾数及其符号，又要表示阶码的大小和它的符号。

例　0.1101011×2^{101}



阶码符　阶码



尾数符　尾数

为了提高精度，对浮点表示数要进行规格化，规格化就是要求尾数上的数最高位不能

是“0”，必须是“1”。即：

$$\frac{1}{2} \leqslant |d| < 1$$

五、带符号数的表示

上面已介绍了整数和小数的处理，但未涉及符号的问题，实际上送入计算机的数可能是正数，也可能是负数。然而计算机对正负号是不能识别的，因此带符号的数在送入计算机之前要做变换，把二进制数的最高位空出来，用做符号位，存放二进制数0或1，用它表示+或－。做了这种变换的数，叫做原码，而对应未变换的数叫真值。

(一) 原码 (True form)

原码是用符号位来表示数的正负极性，对于正数的符号位记“0”，对于负数的符号位记“1”，后面其余的各位表示尾数。

例1

X = +0.0101011	$[X]_{原} = 0.0101011$
X = −0.0101011	$[X]_{原} = 1.0101011$
X = +1101101	$[X]_{原} = 01101101$
X = −1101101	$[X]_{原} = 11101101$
X = +105	$[X]_{原} = 01101001$
X = −105	$[X]_{原} = 11101001$

采用原码可以明显地区分正数和负数，但是用这种方法做加减法运算时就变得很复杂。因为计算机若要对两个数的原码进行加法运算，首先要判别它们的符号是否相同，如果相同就进行加法运算，否则要做减法运算。而在进行减法运算时，还要先比较两个数的绝对值的大小，用大数减去小数，最后给运算结果选择正确的符号。因此采用原码进行加减运算是不方便的，需要寻找一种更理想的形式。

(二) 补码 (Two's Complemet)

采用补码运算的实质，是将减法运算变为加法运算。为了说明这个道理，下面例举钟表对时拨针的例子（图1—1）。

表的时针指在7点，要把时针拨到2点，有两种拨动方法：第一种方法，是把时针往回拨（逆时针方向），拨五个格可写成减法处理式：

$$7 - 5 = 2$$

第二种方法，是把时针向前拨（顺时针方向），拨七个格可写成加法处理式：

$$7 + 7 = 14 = 12 + 2$$

因为从表上显示不出14，实际时针指的是2。

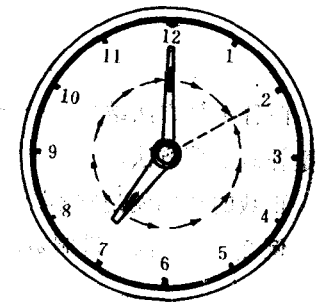


图1—1

分别采用减法与加法两种完全不同的处理方法，但得出了同样的结果。从这个例子

可以得到启示：减法问题可以变成加法问题来处理。

进行这种处理是有一定条件的。例子中，条件是12格为一圈，而表的时针转一圈又回到了原来的位置，从数的进位关系上看不出来。用减法处理式7 − 5 = 2，可以加上12。于是得出了加法处理方式的表达式：

$$7 - 5 + 12 = 7 + (12 - 5) = 7 + 7$$

式中12这个数，在数学中称为“模”，模表示了此记数系统所能表示的最大数。如钟表最大指示为12。例中，+5和−7虽然是不同的两个数，但它们同除以12，其余数相同。因此，它们对模12是同余的，严格的说法应是：

对两个整数a、b，若用某一个整数M去除，所得的余数是相等的，于是a,b对模M同余。记作：

$$a = b \quad (\text{Mod } M)$$

由于计算机表示的位数是有限的。对定点表示情况来说，如果用纯小数表示，8位字长表示数的最大值为$2-2^{-7}$。如果一旦出现2，则就会从最高位溢出，在计算机中自然被丢掉，表示不出来。这就相当模数为2。同样，如果用纯整数表示也是一样，8位字长表示数的最大值为127如果出现128，则就会从高位溢出，这时模数为128。如果4位字长，模数取16。

参照钟表处理，把减法运算问题转化为加法。

$X = 7 = (0111)_B$　　$Y = 5 = (0101)_B$

① 直接用二进制减法

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

结果为2

② 先求出−Y的同余Z，(四位二进制数模16)

$Z = M - Y = 10000 - 0101 = 1011$

做加法处理

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

最高位进位丢失，结果为2。

两种运算方法结果是一样的，就是说负数加模处理后就可用加法处理了。这种方法就是补码运算方法。

正数的补码就是它本身，负数的补码就是将这个负值与此数确定的模相加。

例2 对8位二进制数求补码。

$X = +0.0110110$　　$[X]_{补} = 0.0110110$

$X = -0.0110110$　　$[X]_{补} = 1.1001010$

$X = +01010101$　　$[X]_{补} = 01010101$

$X=-01010101 \quad [X]_{补}=10101011$

$X=+150 \quad [X]_{补}=01101001$

$X=-150 \quad [X]_{补}=10010111$

采用补码表示法进行加减法运算比用原码计算简单，但是求负数X的补码时，要将负数与模做求和运算，这显然不方便。因此需要寻找一种求补码的简便方法，这就是借助反码求补码。

（三）反码（One's Complement）

反码最高位是符号位，正数的反码与原码相同，负数的反码符号位为“1”其余各位对应原码求反。

例3　对8位二进制数求反码。

$X=+0.0110110$

$[X]_{原}=0.0110110 \quad [X]_{反}=0.0110110$

$X=-0.0110110$

$[X]_{原}=1.0110110 \quad [X]_{反}=1.1001001$

$X=+01010101$

$[X]_{原}=01010101 \quad [X]_{反}=01010101$

$X=-01010101$

$[X]_{原}=11010101 \quad [X]_{反}=10101010$

$X=+150$

$[X]_{原}=01101001 \quad [X]_{反}=01101001$

$X=-150$

$[X]_{原}=11101001 \quad [X]_{反}=10010110$

（四）补码与反码的关系

正数的补码就等于反码，负数的补码等于它的反码在最低位加1。

例4　对8位二进制数求补码。

$X=+0.0110110$

$[X]_{原}=0.0110110 \quad [X]_{反}=0.0110110 \quad [X]_{补}=0.0110110$

$X=-0.0110110$

$[X]_{原}=1.0110110 \quad [X]_{反}=1.1001001 \quad [X]_{补}=1.1001010$

$X=+01010101$

$[X]_{原}=01010101 \quad [X]_{反}=01010101 \quad [X]_{补}=01010101$

$X=-01010101$

$[X]_{原}=11010101 \quad [X]_{反}=10101010 \quad [X]_{补}=10101011$

$X=+150$

$[X]_{原}=01101001$　　$[X]_{反}=01101001$　　$[X]_{补}=01101001$

$X=-150$

$[X]_{原}=11101001$　　$[X]_{反}=10010110$　　$[X]_{补}=10010111$

例4 中求得的补码值和例2 中求得的补码值完全相同。

通过例题比较原码运算和补码运算，可看出补码处理的优越性。

例5　求 $Z=X-Y$　　其中：$X=00001010$

$Y=00000011$

① 用原码运算，则要分析X，Y。因X绝对值大于Y的绝对值，X减Y后差值为正。

$$\begin{array}{r} X=00001010 \\ -Y=00000011 \\ \hline Z=00000111 \end{array}$$

结果$[Z]_{原}=00000111$　　真值 $Z=+00000111$

② 用补码运算，不必分析X，Y大小，要将X和$-Y$变补，X正数变补为本身。$-Y$变补要做求反加1，$[Y]_{补}=11111101$

$$\begin{array}{rl} [X]_{补}= & 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0 \\ +[-Y]_{补}= & 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1 \\ \hline [Z]_{补}= & 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1 \end{array}$$

丢掉进位

补码运算时，符号位参加了运算。符号位的进位数（模128）被丢掉了，被丢掉的128正是$-Y$求补码时加模运算加进来的，因去丢掉后结果是正确的。

例6　求 $Z=Y-X$，其中 $X=00001010$

$Y=00000011$

① 用原码运算，则要分析X，Y。因Y绝对值小于X绝对值，所以要X减Y，结果为负。

$$\begin{array}{r} X=0\ 0\ 0\ 0\ 1\ 0\ 1\ 0 \\ -Y=0\ 0\ 0\ 0\ 0\ 0\ 1\ 1 \\ \hline -Z=0\ 0\ 0\ 0\ 0\ 1\ 1\ 1 \end{array}$$

结果应加上负号　$[Z]_{原}=10000111$

真值　　$Z=-00000111$

② 用补码运算，不必分析X，Y。将Y和$-X$变补，Y是正数变补仍为原码，$-X$变补要做求反加1，$[-X]_{补}=11110110$

$$\begin{array}{rl} [Y]_{补}= & 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1 \\ +[-X]_{补}= & 1\ 1\ 1\ 1\ 0\ 1\ 1\ 0 \\ \hline [Z]_{补}= & 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1 \end{array}$$

结果　$[Z]_{补}=11111001$，　　$[Z]_{原}=10000111$