普通高等教育"十一五"国家级规划教材

高等院校计算机专业及专业基础课系列教材

微型计算机基本原理与应用（第二版）

王克义 编著

北京大学出版社

普通高等教育“十一五”国家级规划教材

微型计算机基本原理与应用

（第二版）

王克义　编著

北京大学出版社
PEKING UNIVERSITY PRESS

内 容 简 介

本书全面、系统地介绍了微型计算机的基本组成结构、工作原理和典型接口技术。主要内容包括：数据在计算机中的表示与运算方法，计算机的基本结构与工作过程，微处理器结构，指令系统与汇编语言程序设计，存储器及其接口，输入/输出与 DMA 技术，中断及中断控制器，串并行通信及其接口电路，总线及总线标准，80x86/Pentium 保护模式原理与结构，高性能微处理器的先进技术及典型结构等。

本书内容精练，层次清楚，实用性强；在注重讲解基本概念的同时，也十分注意反映微型计算机发展中的新知识、新技术。本书可作普通高校理工科各专业计算机基础课程教材，也可作为成人教育以及各类职业学校的教材。

图书在版编目(CIP)数据

微型计算机基本原理与应用/王克义编著. —2 版. —北京：北京大学出版社，2010.7
（高等院校计算机专业及专业基础课系列教材）
ISBN 978-7-301-16554-6

Ⅰ. 微… Ⅱ. 王… Ⅲ. 微型计算机－高等学校－教材 Ⅳ. TP36

中国版本图书馆 CIP 数据核字(2010)第 002955 号

书　　名：微型计算机基本原理与应用(第二版)
著作责任者：王克义　编著
责 任 编 辑：沈承凤
标 准 书 号：ISBN 978-7-301-16554-6/TP · 1071
出 版 发 行：北京大学出版社
地　　址：北京市海淀区成府路 205 号　100871
网　　址：http://www.pup.cn　电子信箱：zpup@pup.pku.edu.cn
电　　话：邮购部 62752015　发行部 62750672　编辑部 62752038　出版部 62754962
印 刷 者：
经 销 者：新华书店
787 毫米 × 1092 毫米　16 开本　34.25 印张　855 千字
1997 年 10 月第 1 版
2010 年 7 月第 2 版　2010 年 7 月第 1 次印刷
定　　价：60.00 元

前　言

“微型计算机原理”是高等学校理工科各专业的一门重要计算机基础课程，也是理工科大学生学习和掌握计算机科学技术基础、汇编语言程序设计及常用接口技术的入门课程。通过本课程的学习，可以使学生从理论和实践上掌握微型计算机的基本组成和工作原理，建立计算机系统整体概念，具备利用微机技术进行软、硬件开发的初步能力。学习本课程对于掌握现代计算机的基本概念和技术，以及学习后续有关计算机课程（如计算机体系结构、操作系统、计算机网络、嵌入式系统等）均具有重要的意义。本书是该课程使用的基本教材。

众所周知，现代计算机技术发展极为迅速，新知识、新技术日新月异，层出不穷。本书命名为《微型计算机基本原理与应用》，旨在加强计算机基本概念、基本原理的阐述与介绍。同时，本书注重知识整合，在内容的选取上，尽力摒弃那些陈旧、重复的内容，精心选择课程内容体系中的核心知识和典型技术。在写法上，力求概念准确，表述清晰。

本书以 80x86/Pentium 系列微型计算机为背景机，全面、系统地介绍了微型计算机的基本结构、工作原理及典型接口技术。全书共分 16 章，从内容上可划分为四个知识单元：① 计算机的基本结构及工作原理（第 1、2、3、7、13、14 章）；② 指令系统及汇编语言程序设计（第 4、5、6 章）；③ I/O 接口技术（第 8、9、10、11、12 章）；④ 高性能微处理器的先进技术及典型结构（第 15、16 章）。学习本书的预备知识为数字电路及逻辑设计基础知识。

与其他同类教材不同的是，本书在第 1 章和第 2 章中有意识地融入了传统的“计算机组成原理”教材中的相关核心内容，如定点数与浮点数的表示与运算方法、典型的加减法硬件结构、硬布线控制器及微程序控制器的原理与实现技术等，并在第 3 章简要介绍了微处理器的编程结构之后，用第 4、5、6 三章的篇幅翔实地讲解了 80x86 的指令系统及汇编语言程序设计的基本概念与方法；后续章节则相继介绍了微处理器的内部结构及外部功能特性、存储器及其接口、输入/输出控制、可编程接口电路、总线及总线标准以及高性能微处理器的相关概念和技术等；其中，在第 15 章中，专门介绍了 80x86/Pentium 处理器的保护模式的原理和结构，如描述符及描述符表、保护模式的存储管理和地址转换以及多任务的实现及保护机制等；在第 16 章，重点介绍了现代高性能微处理器的多项先进技术和典型结构，如指令级并行及流水线中的“相关”及其处理技术、超标量流水线技术、超长指令字结构及 RISC 技术等。最后，还介绍了多核处理器的设计理念及现代 PC 机主板的结构和技术特点。

本书可供 48 ~ 60 学时的课堂教学使用，有些章节的内容可根据不同的教学要求进行适当取舍。每章后面列出的思考题与习题，主要供理解和复习本章基本内容而用，本书最后给出了部分习题的参考答案。

另外，鉴于“微型计算机原理”课程是技术性、实践性较强的课程，因此在教学中应安排相应的实验及编程上机环节。教师可根据具体实验设备及上机条件，安排适当的接口实验及汇编程序上机内容。对于尚不具备专门的微机接口实验设备的教学环境，教师可结合 PC 机上

已配备的键盘、鼠标及显示器等基本 I/O 设备，组织相应的接口实验内容，如键盘输入、显示器输出编程，鼠标器编程等，从而培养学生的 I/O 接口编程能力。关于这方面的内容，请参见第 6 章的介绍。

本书是在作者近年承担北京大学计算机系本科生、北京大学理科实验班教学实践的基础上编写而成的，并参考和吸收了国外较新的同类教科书及国内兄弟院校优秀教材的有关内容，在此，特向有关作者一并致谢。

在本书的编写和出版过程中，承蒙北京大学信息科学技术学院及北京大学出版社领导的热情支持和指导，北京大学出版社理科编辑室陈小红主任、责任编辑沈承凤高级工程师为本书的编辑、出版给予了大力支持和帮助。在此，谨向他/她们表示衷心的感谢。

由于编者的水平所限，书中一定存在不少差错和疏漏，诚请广大读者及专家批评指正。

编者

2010 年 2 月于北京大学

E-mail：wky@ pku. edu. cn

第 1 章　数据在计算机中的表示与运算方法

本章首先介绍计算机中几种常用进位计数制及不同进位制数之间的转换方法，然后介绍数据在计算机中的表示与运算方法，最后简要介绍几种常见的字符编码形式，为后续章节的学习打下必备的基础。

1.1　进位计数制

1.1.1　进位计数制及其基数和权

进位计数制（简称进位制）是指用一组固定的数字符号和特定的规则表示数的方法。在人们日常生活和工作中，最熟悉最常用的是十进制，此外还有十二进制、六十进制等。在数字系统和计算机领域，常用的进位计数制是二进制、八进制及十六进制。

研究和讨论进位计数制的问题涉及两个基本概念，即基数和权。在进位计数制中，一种进位制所允许选用的基本数字符号（也称数码）的个数称为这种进位制的基数。不同进位制的基数不同。例如在十进制中，是选用 0 ~ 9 这 10 个数字符号来表示的，它的基数是 10；在二进制中，是选用 0 和 1 这两个数字符号来表示的，它的基数是 2，等等。

同一个数字符号处在不同的数位时，它所代表的数值是不同的，每个数字符号所代表的数值等于它本身乘以一个与它所在数位对应的常数，这个常数叫做位权，简称权（weight）。例如十进制数个位的位权是 1，十位的位权是 10，百位的位权是 100，以此类推。一个数的数值大小就等于该数的各位数码乘以相应位权的总和。例如：

$$\text{十进制数 } 2918 = 2 \times 1000 + 9 \times 100 + 1 \times 10 + 8 \times 1$$

1.1.2　计算机中几种常用的进位计数制

1. 十进制

十进制数有 10 个不同的数字符号（0、1、2、3、4、5、6、7、8、9），即它的基数为 10；每个数位计满 10 就向高位进位，即它的进位规则是“逢十进一”。任何一个十进制数，都可以用一个多项式来表示，例如：

$$312.25 = 3 \times 10^2 + 1 \times 10^1 + 2 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$$

式中等号右边的表示形式，称为十进制数的多项式表示法，也叫按权展开式；等号左边的形式，称为十进制的位置记数法。位置记数法是一种与位置有关的表示方法，同一个数字符号处于不同的数位时，所代表的数值不同，即其权值不同。容易看出，上式各位的权值分别为 10^2、10^1、10^0、10^{-1}、10^{-2}。一般地说，任意一个十进制数 N 都可以用多项式表示法写成如下形式：

$$(N)_{10} = \pm(K_{n-1} \cdot 10^{n-1} + K_{n-2} \cdot 10^{n-2} + \cdots + K_1 \cdot 10^1 + K_0 \cdot 10^0 + K_{-1} \cdot 10^{-1} + \cdots + K_{-m} \cdot 10^{-m}) = \pm \sum_{i=-m}^{n-1} K_i \cdot 10^i$$

其中,K_i($n-1 \leqslant i \leqslant -m$)表示第 i 位的数字符号,可以是10个数字(0~9)符号中的任何一个,由具体的十进制数 N 来确定。m、n 为正整数,m 为小数位数,n 为整数位数。

实际的数字系统以及人们日常使用的进位计数制并不仅仅是十进制,其他进位制的计数规律可以看成是十进制计数规律的推广。对于任意的 R 进制来说,它有 R 个不同的数字符号,即基数为 R,计数进位规则为"逢 R 进一"。数 N 可以用类似上面十进制数的多项式表示法书写如下:

$$(N)_R = \pm(K_{n-1} \cdot R^{n-1} + K_{n-2} \cdot R^{n-2} + \cdots + K_1 \cdot R^1 + K_0 \cdot R^0 + K_{-1} \cdot R^{-1} + \cdots + K_{-m} \cdot R^{-m}) = \pm \sum_{i=-m}^{n-1} K_i \cdot R^i$$

其中 K_i($n-1 \leqslant i \leqslant -m$)表示第 i 位的数字符号,可以是 R 个数字符号中的任何一个,由具体的 R 进制数 N 来确定。m、n 为正整数,m 为小数位数,n 为整数位数。若 $R=2$,即为二进制计数制。它是数字系统特别是电子计算机中普遍采用的进位计数制。

2. 二进制

二进制数的基数为2,即它所用的数字符号个数只有两个("0"和"1")。它的计数进位规则为"逢二进一"。

在二进制中,由于每个数位只能有两种不同的取值(要么为0,要么为1),这就特别适合于使用仅有两种状态(如导通、截止;高电平、低电平等)的开关元件来表示,一般是采用电子开关元件,目前绝大多数是采用半导体集成电路的开关器件来实现。

对于一个二进制数,也可以用类似十进制数的按权展开式予以展开,例如二进制数11011.101可以写成:

$$(11011.101)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

一般地说,任意一个二进制数 N,都可以表示为:

$$(N)_2 = \pm(K_{n-1} \cdot 2^{n-1} + K_{n-2} \cdot 2^{n-2} + \cdots + K_1 \cdot 2^1 + K_0 \cdot 2^0 + K_{-1} \cdot 2^{-1} + \cdots + K_{-m} \cdot 2^{-m}) = \pm \sum_{i=-m}^{n-1} K_i \cdot 2^i$$

其中,K_i 为0或1,由具体的数 N 来确定。m、n 为正整数,m 为小数位数,n 为整数位数。

二进制数的优点不仅仅是由于它只有两种数字符号,因而便于数字系统与电子计算机内部的表示与存储。它的另一个优点就是运算规则的简便性,而运算规则的简单,必然导致运算电路的简单以及相关控制的简化。后面我们将具体讨论二进制算术运算及逻辑运算的规则。

3. 八进制

八进制数的基数 $R=8$,每位可能取8个不同的数字符号0、1、2、3、4、5、6、7中的任何一个,进位规则是"逢八进一"。

由于3位二进制数刚好有8种不同的数位组合(如下所示),所以一位八进制数容易改写成相应的3位二进制数来表示。

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

这样,把1个八进制数每位变换为相等的3位二进制数,组合在一起就成了相等的二进制数。

例 1.1　将八进制数 53 转换成二进制数。

八进制	5	3
	↓	↓
二进制	101	011

所以，$(53)_8 = (101011)_2$。

例 1.2　将八进制数 67.721 转换成二进制数。

八进制	6	7.	7	2	1
	↓	↓	↓	↓	↓
二进制	110	111.	111	010	001

所以，$(67.721)_8 = (110111.111010001)_2$。

例 1.3　将二进制数转换成八进制数。

二进制	101	111	011.	011	111
	↓	↓	↓	↓	↓
八进制	5	7	3.	3	7

所以，$(101111011.011111)_2 = (573.37)_8$。

显然，用八进制比二进制书写要简短、易读，而且与二进制间的转换也较方便。

4. 十六进制

十六进制数的基数 $R = 16$，每位用 16 个数字符号 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F 中的一个表示，进位规则是“逢十六进一”。

由于 4 位二进制数刚好有 16 种不同的数位组合（如下所示），所以一位十六进制数可以改写成相应的 4 位二进制数来表示：

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

这样，把一个十六进制数的每位变换为相等的 4 位二进制数，组合在一起就变成了相等的二进制数。

例 1.4　十六进制数转换为二进制数。

十六进制	D	3	F
	↓	↓	↓
二进制	1101	0011	1111

所以，$(D3F)_{16} = (110100111111)_2$。

例 1.5 二进制转换为十六进制。

二进制	1110	0010
	↓	↓
十六进制	E	2

所以,$(11100010)_2 = (E2)_{16}$。

由上面的介绍可以看出,使用十六进制或八进制表示具有如下的优点:

(1)容易书写、阅读,也便于人们记忆。

(2)容易转换成可用电子开关元件存储、记忆的二进制数。所以,它们是电子计算机工作者所普遍采用的数据表示形式。

1.2 不同进位制数之间的转换

一个数从一种进位制表示变成另外一种进位制表示,称为数的进位制转换。实现这种转换的方法是多项式替代法和基数乘除法。下面结合具体例子来讨论这两种方法的应用。

1.2.1 二进制数转换为十进制数

例 1.6 将二进制数 101011.101 转换为十进制数。

这里,只要将二进制数用多项式表示法写出,并在十进制中运算,即按十进制的运算规则算出相应的十进制数值即可。

$$\begin{aligned}(101011.101)_2 &= (2^5 + 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-3})_{10} \\ &= (32 + 8 + 2 + 1 + 0.5 + 0.125)_{10} \\ &= (43.625)_{10}\end{aligned}$$

这个例子说明,为了求得某二进制数的十进制表示形式,只要把该二进制数的按权展开式写出,并在十进制系统中计算,所得结果就是该二进制数的十进制形式,即实现了由二进制到十进制的数制转换。

顺便指出,用类似的方法可将八进制数转换为十进制数。

例 1.7 将八进制数 155 转换为十进制数。

$$\begin{aligned}(155)_8 &= (1 \times 8^2 + 5 \times 8^1 + 5 \times 8^0)_{10} \\ &= (109)_{10}\end{aligned}$$

上述这种用以实现数制转换的方法,称为多项式替代法。

1.2.2 十进制数转换为二进制数

1. 十进制整数转换为二进制整数

例 1.8 将十进制数 935 转换为二进制数。

设$(935)_{10} = (B_nB_{n-1}\cdots B_1B_0)_2$,其中 $B_nB_{n-1}\cdots B_1B_0$ 为所求二进制数的各位数值,当然,它们非 0 即 1。因此,只要准确地定出所求二进制数的各位数值究竟是 0 还是 1,就可确定与$(935)_{10}$相等的二进制数。

将上式右边写成按权展开式,则有

$$(935)_{10} = B_n \cdot 2^n + B_{n-1} \cdot 2^{n-1} + \cdots + B_1 \cdot 2^1 + B_0 \cdot 2^0 \qquad (1\text{-}1)$$

将式(1-1)两边同除以2，并且为了简便，省略十进制的下标注，则可得：

$$\frac{935}{2} = B_n \cdot 2^{n-1} + B_{n-1} \cdot 2^{n-2} + \cdots + B_1 \cdot 2^0 + \frac{B_0}{2} \qquad (1\text{-}2)$$

这里，首先确定 B_0 是1还是0，由(1-2)式容易得出如下判断：

① 如果935能够被2整除，则说明 B_0 必为0；

② 如果935不能被2整除，则说明 B_0 必为1。

在本例中，因为935不能被2整除，所以 $B_0 = 1$，它正好是$\frac{935}{2}$的余数。

根据 $B_0 = 1$，由式(1-2)又可得：

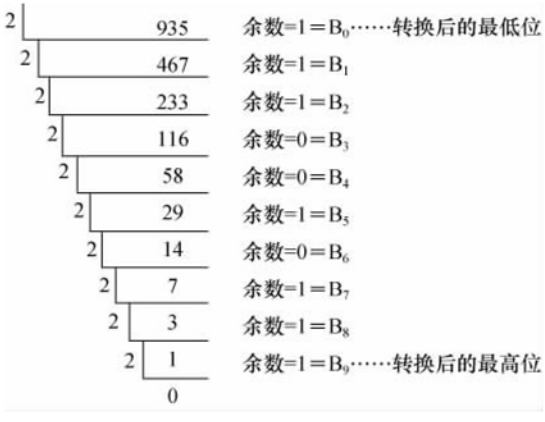
$$\frac{935}{2} - \frac{1}{2} = B_n \cdot 2^{n-1} + B_{n-1} \cdot 2^{n-2} + \cdots + B_1$$

$$467 = B_n \cdot 2^{n-1} + B_{n-1} \cdot 2^{n-2} + \cdots + B_1 \qquad (1\text{-}3)$$

再将式(1-3)两边同除以2，用同样的判断方法又可以确定出 B_1 是0还是1。因为467不能被2整除，所以 $B_1 = 1$。

如此连续地做下去，就可以将 $B_0B_1\cdots B_{n-1}B_n$，逐一确定下来，也即得到所求的二进制数。

现将整个推算过程表示如下：



所以，转换结果为：

$$(935)_{10} = (B_9B_8\cdots B_1B_0)_2 = (1110100111)_2$$

另外，类似地，采用“除8取余”或“除16取余”的方法，即可将一个十进制整数转换为八进制整数或十六进制整数。

一般地，可以将给定的一个十进制整数转换为任意进制的整数，只要用所要转换的数制的基数去连续除给定的十进制整数，最后将每次得到的余数依次按正确的高、低位顺序列出，即可得到所要转换成的数制的数。

例 1.9　转换十进制数2803为十六进制数，即$(2803)_{10} = (?)_{16}$。

解

16	2803	余数=3，H_0=3……转换后的最低位
16	175	余数=15，H_1=F
16	10	余数=10，H_2=A ……转换后的最高位
	0	

因此，$(2803)_{10}=(H_2H_1H_0)_{16}=(AF3)_{16}$

上述这种数制转换的方法称为基数除法或“除基取余”法。可概括为：“除基取余，直至商为0，注意确定高、低位”。

2. 十进制小数转换为二进制小数

例1.10 将十进制小数0.5625转换成二进制小数。

解 设$(0.5625)_{10}=(0.B_{-1}B_{-2}\cdots B_{-m})_2$，即：

$$(0.5625)_{10}=B_{-1}\cdot 2^{-1}+B_{-2}\cdot 2^{-2}+\cdots+B_{-m}\cdot 2^{-m} \tag{1-4}$$

其中，$B_{-1},B_{-2},\cdots,B_{-m}$为所求二进制小数的各位数值，它们非0即1。将式(1-4)两边同乘以2，得：

$$1.1250=B_{-1}+B_{-2}\cdot 2^{-1}+\cdots+B_{-m}\cdot 2^{-m+1} \tag{1-5}$$

由于等式两边的整数与小数必须对应相等，得到$B_{-1}=1$。同时由式(1-5)又可得：

$$0.1250=B_{-2}\cdot 2^{-1}+\cdots+B_{-m}\cdot 2^{-m+1} \tag{1-6}$$

式(1-6)两边再乘以2，得：

$$0.2500=B_{-2}+\cdots+B_{-m}\cdot 2^{-m+2}$$

因此又可得到$B_{-2}=0$。

如此继续下去，可逐一确定出$B_{-1},B_{-2},\cdots,B_{-m}$的值，我们可以写出整个转换过程如下：

0.5625	
× 2	
1.1250	整数部分=1，B_{-1}=1 ……转换后的最高位
0.1250	
× 2	
0.2500	整数部分=0，B_{-2}=0
× 2	
0.5000	整数部分=0，B_{-3}=0
× 2	
1.0000	整数部分=1，B_{-4}=1 ……转换后的最低位

因此，$(0.5625)_{10}=(0.B_{-1}B_{-2}B_{-3}B_{-4})_2=(0.1001)_2$。

值得注意的是，在十进制小数转换成二进制小数时，整个计算过程可能无限地进行下去，这时，一般考虑到计算机实际字长的限制，只取有限位数的近似值就可以了。

同样，这个方法也可推广到十进制小数转换为任意进制的小数，只需用所要转换成的数制的基数去连续乘给定的十进制小数，每次得到的整数部分即依次为所求数制小数的各位数。不过应注意，最先得到的整数部分应是所求数制小数的最高有效位。

上述这种数制转换方法称为基数乘法或“乘基取整”法。可概括如下：“乘基取整，注意确定高、低位及有效位数。”

另外，如果一个数既有整数部分又有小数部分，则可用前述的“除基取余”及“乘基取整”

的方法分别将整数部分与小数部分进行转换，然后合并起来就可得到所求结果。例如：

$$(17.25)_{10} \Rightarrow (17)_{10} + (0.25)_{10}$$
$$\downarrow \qquad \downarrow$$
$$(10001)_2 + (0.01)_2 \Rightarrow (10001.01)_2$$

所以，$(17.25)_{10} = (10001.01)_2$。

1.2.3　任意两种进位制数之间的转换

由前面的介绍可知，为实现任意两种进位制数之间的转换（例如从 α 进制转换成 β 进制），可以用“基数乘除法”或“多项式替代法”直接从 α 进制转换成 β 进制，此时如果熟悉 α 进制的运算规则就可以采用“基数乘除法”；如果熟悉 β 进制的运算规则就采用“多项式替代法”。但有时可能对 α 进制与 β 进制的运算规则都不熟悉，那么一种方便的方法就是利用十进制作桥梁。

首先将 $(N)_\alpha$ 转换为 $(N)_{10}$，这里采用“多项式替代法”，在十进制系统中进行计算；然后将 $(N)_{10}$ 转换为 $(N)_\beta$，这里采用“基数乘除法”，也是在十进制系统中进行计算。

例 1.11　把 $(301.23)_4$ 转换成五进制数。

第一步，采用多项式替代法把该数转换成十进制数。

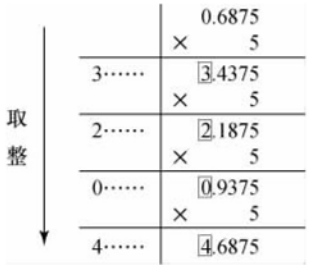
$$\begin{aligned} N &= 3 \times 4^2 + 0 \times 4^1 + 1 \times 4^0 + 2 \times 4^{-1} + 3 \times 4^{-2} \\ &= 48 + 0 + 1 + 0.5 + 0.1875 \\ &= 49.6875 \end{aligned}$$

即 $(301.23)_4 = (49.6875)_{10}$。

第二步，采用基数乘除法把该数从十进制转换为五进制（整数部分与小数部分分开进行）。

整数部分：

小数部分：



即 $(49.6875)_{10} = (144.3204)_5$（取 4 位小数）。

所以 $(301.23)_4 = (144.3024)_5$（取 4 位小数）。

1.3 二进制数的算术运算和逻辑运算

1.3.1 二进制数的算术运算

二进制数的算术运算规则非常简单,其具体运算规则如下。

1. 加法运算规则

二进制加法规则是:0+0=0,0+1=1,1+0=1,1+1=10("逢二进一")。

例1.12 1010+111=10001　　1011.101+10.01=1101.111

	1010	被加数		1011.101	被加数
+	111	加数	+	10.01	加数
	10001	和		1101.111	和

2. 减法运算规则

二进制减法规则是:0-0=0,1-0=1,1-1=0,0-1=1("借一当二")。

例1.13 1011-101=110　　1101.111-10.01=1011.101

	1011	被减数		1101.111	被减数
-	101	减数	-	10.01	减数
	110	差		1011.101	差

3. 乘法运算规则

二进制乘法规则是:0×0=0,0×1=0,1×0=0,1×1=1。

例1.14 1011×1010=1101110。

	1011	被乘数
×	1010	乘数
	0000	部分积
	1011	
	0000	
+	1011	
	1101110	乘积

从这个例子中可以看出,在二进制乘法运算时,若相应的乘数位为1,则把被乘数照写一遍,只是它的最后一位应与相应的乘数位对齐(这实际上是一种移位操作);若相应的乘数位为0,则部分积各位均为0;当所有的乘数位都乘过之后,再把各部分积相加,便得到最后乘积。所以,实质上二进制数的乘法运算可以归结为"加"(加被乘数)和"移位"两种操作。

4. 除法运算规则

二进制数的除法是乘法的逆运算,这与十进制数的除法是乘法的逆运算一样。因此利用二进制数的乘法及减法规则可以容易地实现二进制数的除法运算。

例1.15 110110÷1010=101…100。

		101	商
除数 1010	)	110110	被除数
		-1010	
		1110	
		1010	
		100	余数

1.3.2　二进制数的逻辑运算

计算机能够实现的另一种基本运算是逻辑运算。逻辑运算是按位进行的，其运算的对象及运算结果只能是 0 和 1 这样的逻辑量。这里的 0 和 1 并不具有数值大小的意义，而仅仅具有如“真”和“假”、“是”和“非”这样的逻辑意义。二进制数的逻辑运算实际上是将二进制数的每一位都看成逻辑量时进行的运算。

基本的逻辑运算有逻辑“或”、逻辑“与”和逻辑“非”三种，常用的还有逻辑“异或”运算。下面分别予以说明。

1. 逻辑“或”

逻辑“或”也称逻辑加，其运算规则是：两个逻辑量中只要有一个为 1，其运算结果就为 1；只有当两个逻辑量全为 0 时，其运算结果才为 0，可简述为“有 1 得 1，全 0 得 0”。逻辑“或”的运算符号为“∨”或“+”。表示如下：

$$0 \vee 0 = 0 \quad 0 \vee 1 = 1 \quad 1 \vee 0 = 1 \quad 1 \vee 1 = 1$$

也可表示为：

$$0 + 0 = 0 \quad 0 + 1 = 1 \quad 1 + 0 = 1 \quad 1 + 1 = 1$$

例 1.16　$0110 \vee 0001 = 0111$。

逻辑“或”运算常用于将一个已知二进制数的某一位或某几位置 1，而其余各位保持不变。例如，欲使二进制数 10101100 的最低一位置 1，而其余各位不变，就可用 00000001 与之相“或”来实现。即：

$$\begin{array}{cccccccc} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ \vee\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{array}$$

2. 逻辑“与”

逻辑“与”也称逻辑乘，其运算规则是：两个逻辑量中只要有一个为 0，其运算结果就为 0；只有当两个逻辑量全为 1 时，其运算结果才为 1。可简述为“有 0 得 0，全 1 得 1”。逻辑“与”的运算符号为“∧”或“×”。如下所示：

$$0 \wedge 0 = 0 \quad 0 \wedge 1 = 0 \quad 1 \wedge 0 = 0 \quad 1 \wedge 1 = 1$$

也可表示为：

$$0 \times 0 = 0 \quad 0 \times 1 = 0 \quad 1 \times 0 = 0 \quad 1 \times 1 = 1$$

例 1.17　$1001 \times 1110 = 1000$。

逻辑“与”运算常用于将一个已知二进制数的某一位或某几位置 0，而其余各位保持不变。例如，欲使二进制数 01010011 的最低两位置 0，而其余各位保持不变，就可用 11111100 与之相“与”来实现。即：

$$\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ \wedge\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}$$

3. 逻辑“非”

逻辑“非”也称逻辑反，其运算规则是：1“非”为 0，0“非”为 1。其运算符号为“—”或“

¬”。表示如下:

$$\overline{1}=0 \qquad \overline{0}=1$$

例 1.18 ¬1001 =0110。

4. 逻辑“异或”

逻辑“异或”又称模 2 加,其运算规则是:0 和任何数相“异或”该数不变,1 和任何数相“异或”该数变反。可简述为“相同得 0,不同得 1”。其运算符号为“∀”或“⊕”。表示如下:

$$0 \forall 0=0 \quad 0 \forall 1=1 \quad 1 \forall 0=1 \quad 1 \forall 1=0$$

也可表示为:

$$0 \oplus 0=0 \quad 0 \oplus 1=1 \quad 1 \oplus 0=1 \quad 1 \oplus 1=0$$

例 1.19 0110∀1001 =1111。

逻辑“异或”运算常用于将一个已知二进制数的某些位变反而其余各位不变。例如,欲使 10101100 的最低两位变反而其余各位不变,就可以用 00000011 与之进行“异或”运算来完现。即:

$$\begin{array}{cccccccc} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ \forall\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{array}$$

1.3.3 移位运算

移位是二进制数的算术、逻辑运算的又一种基本运算。计算机指令系统中都设置有各种移位指令。移位分为逻辑移位和算术移位两大类。

1. 逻辑移位

所谓逻辑移位,通常是把操作数当成纯逻辑代码,没有数值含义,因此没有符号与数值变化的概念;操作数也可能是一组无符号的数值代码(即无符号数),通过逻辑移位对其进行数值的变化、判别或某种加工。逻辑移位可分为逻辑左移、逻辑右移、循环左移和循环右移。

逻辑左移是将操作数的所有位同时左移,最高位移出原操作数之外,最低位补 0。逻辑左移一位相当于将无符号数乘以 2。例如,将 01100101 逻辑左移一位后变成 11001010,相当于 $(101)_{10} \times 2 = 202$。

逻辑右移是将操作数的所有位同时右移,最低位移出原操作数之外,最高位补 0。逻辑右移一位相当于将无符号数除以 2。例如,将 10010100 逻辑右移一位后变成 01001010,相当于 $148 \div 2 = 74$。

循环左移就是将操作数的所有位同时左移,并将移出的最高位送到最低位。循环左移的结果不会丢失被移动的数据位。例如,将 10010100 循环左移一位后变成 00101001。

循环右移就是将操作数的所有位同时右移,并将移出的最低位送到最高位。它也不会丢失被移动的数据位。例如,将 10010100 循环右移一位后变成 01001010。

2. 算术移位

算术移位是把操作数当作带符号数进行移位,所以在算术移位中,必须保持符号位不变,例如一个正数在移位后还应该是正数。如果由于移位操作使符号位发生了改变(由 1 变 0,或由 0 变 1),则应通过专门的方法指示出错信息(如将“溢出”标志位置 1)。

与逻辑移位类似，算术移位可分为算术左移、算术右移、循环左移和循环右移。

算术左移的移位方法与逻辑左移相同，就是将操作数的所有位同时左移，最高位移出原操作数之外，最低位补 0。算术左移一位相当于将带符号数（补码）乘以 2。例如，将 11000101 算术左移一位后变成 10001010，相当于（－59）×2 = －118，未溢出，结果正确。

算术右移是将操作数的所有位同时右移，最低位移出原操作数之外，最高位不变。算术右移一位相当于将带符号数（补码）除以 2。例如，将 10111010 算术右移一位变成 11011101，相当于（－70）÷2 = －35，未溢出，结果正确。

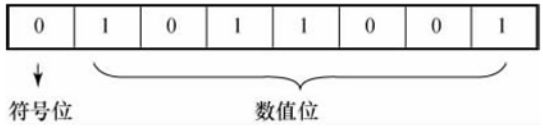
循环左移和循环右移的操作与前述逻辑移位时的情况相同，都是不丢失移出原操作数的位，而将其返回到操作数的另一端。

1.4　数据在计算机中的表示形式

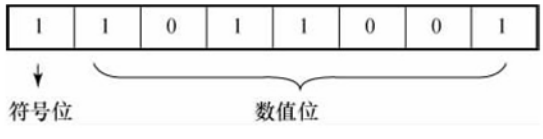
1.4.1　机器数与真值

电子计算机实质上是一个二进制的数字系统，在机器内部，二进制数总是存放在由具有两种相反状态的存储元件构成的寄存器或存储单元中，即二进制数码 0 和 1 是由存储元件的两种相反状态来表示的。另外，对于数的符号（正号“＋”和负号“－”）也只能用这两种相反的状态来区别。也就是说，只能用 0 或 1 来表示。

数的符号在机器中的一种简单表示方法为：规定在数的前面设置一位符号位，正数符号位用 0 表示，负数符号位用 1 表示。这样，数的符号标识也就“数码化”了。即带符号数的数值和符号统一由数码形式（仅用 0 和 1 两种数字符号）来表示。例如，正二进制数 N_1 = ＋1011001，在计算机中表示为：



负二进制数 N_2 = －1011001，在计算机表示为：



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）在机器中加以数码化后的表示形式，称为机器数，而把原来的数称为机器数的真值。例如，上面例子中的 N_1 = ＋1011001、N_2 = －1011001 为真值，它们在计算机中的表示 01011001 和 11011001 为机器数。

在将数的符号用数码（0 或 1）表示后，数值部分究竟是保留原来的形式，还是按一定规则做某些变化，这要取决于运算方法的需要。从而有机器数的三种常见形式，即原码、补码和反码。下面首先介绍机器数的这三种常见表示形式，然后介绍移码的特点及用途。

1.4.2 四种常见的机器数形式

1. 原码

原码是一种比较直观的机器数表示形式。约定数码序列中的最高位为符号位,符号位为0表示该数为正数,为1表示该数为负数;其余有效数值部分则用二进制的绝对值表示。

例如:

真值 x	$[x]_{原}$
+0.1001	0.1001
-0.1001	1.1001
+1001	01001
-1001	11001

在后面讨论定点数表示时,将会看到,定点数又有定点小数和定点整数之分,所以下面分别给出定点小数和定点整数的原码定义。

① 若定点小数原码序列为 $x_0x_1x_2\cdots x_n$,则

$$[x]_{原}=\begin{cases}x & 0\leqslant x<1\\ 1-x & -1<x\leqslant 0\end{cases} \tag{1-7}$$

式中 x 代表真值,$[x]_{原}$为原码表示的机器数。

例如:

$x=+0.1011$,则$[x]_{原}=0.1011$;

$x=-0.1011$,则$[x]_{原}=1-(-0.1011)=1+0.1011=1.1011$。

② 若定点整数原码序列为 $x_0x_1\cdots x_n$,则

$$[x]_{原}=\begin{cases}x & 0\leqslant x<2^n\\ 2^n-x & -2^n<x\leqslant 0\end{cases} \tag{1-8}$$

例如:

$x=+1011$,则$[x]_{原}=01011$;

$x=-1011$,则$[x]_{原}=2^4-(-1011)=10000+1011=11011$。

需要注意的是,在(1-7)式和(1-8)式中,有效数位是 n 位(即 $x_1\sim x_n$),连同符号位是 $n+1$ 位。

对于原码表示,具有如下特点:

① 原码表示中,真值0有2种表示形式。

以定点小数的原码表示为例:

$$[+0]_{原}=0.00\cdots0;\quad [-0]_{原}=1-(-0.00\cdots0)=1+0.00\cdots0=1.00\cdots0$$

② 在原码表示中,符号位不是数值的一部分,它们仅是人为约定("0为正,1为负"),所以符号位在运算过程中需要单独处理,不能当作数值的一部分直接参与运算。

原码表示简单直观,而且容易由其真值求得,相互转换也较方便。但计算机在用原码做加减运算时比较麻烦。比如当两个数相加时,如果是同号,则数值相加,符号不变;如果是异号,则数值部分实际上是相减,此时必须比较两个数绝对值的大小,才能确定谁减谁,并要确定结果的符号。这件事在手工计算时是容易解决的,但在计算机中,为了判断同号还是异号,比较绝对值的大小,就要增加机器的硬件设备,并增加机器的运行时间。为此,人们找到了更适合于计算机进行运算的其他机器数表示法。