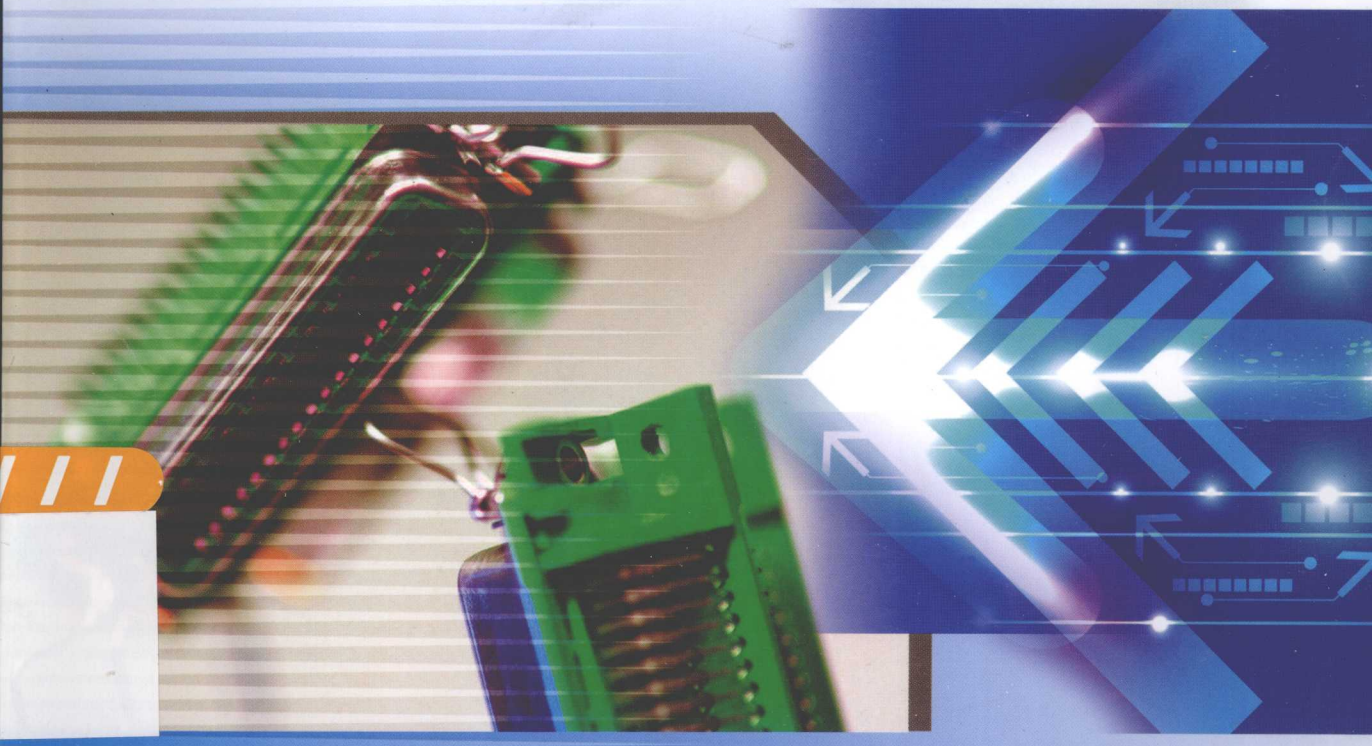
微型计算机原理与接口技术

吕淑平 于立君 刘 心 曾薄文 编著



HEUP 哈尔滨工程大学出版社
Harbin Engineering University Press

微型计算机原理与接口技术

吕淑平　于立君　刘　心　曾薄文　编著

哈尔滨工程大学出版社

内容简介

本书以流行的 Intel 系列微处理器为背景，以 16 位微处理器 8086/8088 为核心，全面系统地讲述微型计算机系统的基本组成及工作原理、汇编语言程序设计、微机接口技术、微机应用技术，同时扼要地介绍了高档 80x86 和 Pentium 系列微处理器结构特点和技术精髓以及总线相关知识。使学生能系统掌握汇编语言程序设计的基本方法和微机接口技术，建立微型计算机系统的整体概念，并具有微型计算机软件及硬件初步开发、设计能力。为后续课程的学习和跟踪计算机技术的新发展，进一步学习和应用相关方面的新知识、新技术打下必要基础。

本书内容共分 8 章。主要包括：计算机基础知识、微处理器与总线、存储器、指令系统、汇编语言程序设计基础、输入/输出及中断系统、可编程接口芯片及应用、8088 微机系统应用举例。

本书内容先进、重点突出、通俗易懂、详略得当。以实际应用为主，理论够用为度，便于教学与自学。既可作为高等学校工科非计算机专业“微型计算机原理与接口技术”课程的教材，也可供从事微型计算机硬件和软件设计的工程技术人员参考。

图书在版编目(CIP)数据

微型计算机原理与接口技术/吕淑平等编著. —哈尔滨：哈尔滨工程大学出版社，2013. 1

ISBN 978-7-5661-0541-7

Ⅰ. ①微… Ⅱ. ①吕… Ⅲ. ①微型计算机-理论②微型计算机-接口技术 Ⅳ. ①TP36

中国版本图书馆 CIP 数据核字(2013)第 025484 号

出版发行 哈尔滨工程大学出版社
社　　址 哈尔滨市南岗区东大直街 124 号
邮政编码 150001
发行电话 0451-82519328
传　　真 0451-82519699
经　　销 新华书店
印　　刷 黑龙江省地质测绘印制中心
开　　本 787mm×1 092mm　1/16
印　　张 23
字　　数 561 千字
版　　次 2013 年 2 月第 1 版
印　　次 2013 年 2 月第 1 次印刷
定　　价 45.00 元

http://www.hrbeupress.com
E-mail:heupress@hrbeu.edu.cn

前　言

本书以 Intel 系列微处理器为背景，以 16 位微处理器 8086/8088 为核心，全面系统地讲述了微型计算机系统的基本组成及工作原理、汇编语言程序设计、微机接口技术、微机应用技术。30 多年来，Intel 系列 CPU 一直占据主导地位，虽然 Intel 公司生产的微处理器芯片种类很多，但是它们都有相似性和继承性。尽管 8086/8088 后续的 80x86 和 Pentium 系列 CPU 功能与结构已经发生很大的变化，但从基本概念、基本结构和指令格式上，仍然是经典的 8086/8088 CPU 的延续与提升。并且，其他系列的 CPU 也可以与 80x86 CPU 兼容。因此，读者一旦掌握了基本的 8086/8088 微处理器后，就很容易通过进一步学习掌握高档微处理器技术。

本书是作者根据微型计算机技术的发展以及教学过程中的经验体会进行撰稿，并对微处理器、总线技术、指令系统、汇编语言程序设计、存储器系统、接口芯片及应用等章节加入使其更能反映当前微机领域的新进展、新技术内容。同时本书在强调基本概念的基础上，注重实际应用，部分章节加入了工程应用背景。

本书编写依据“四模块”体系：即“微机组成及原理模块、汇编语言程序设计模块、微机接口技术模块、微机系统应用设计模块”。学生通过前三个模块的学习，已掌握微机系统基础知识、中断技术、I/O 接口设计、常用应用程序设计，具备了设计微机系统和智能化仪表的硬件及软件基础。所以在第四个“微机系统应用设计模块”中，我们结合控制专业特点扩展了采样监控系统设计和直流电机闭环调速系统设计等内容，并由学生最终动手在实验台上实现。以往学生反映学完本课程后，还是不知道怎么设计、开发一个实际系统，第四个模块设置的目的就是从整体上让学生掌握一个完整系统的设计和开发过程，训练和提高学生的工程实践和应用能力。“四模块”体系相互衔接，逐步提升学生微型计算机软件及硬件开发、设计能力。

本书编写力求做到：通俗易懂、简明实用、重点突出、深浅适度、体例合理。以实际应用为主，理论够用为度。

本书主要由吕淑平组织和统稿，并负责编写第 2 章、第 6 章（不含 6.5 节）；于立君负责编写第 3 章、第 7 章；刘心负责编写第 1 章、第 4 章、第 5 章；曾薄文负责编写 6.5 节和第 8 章。

本书在编写过程中引用和参考了一些专家学者的著作和研究成果，在此表示衷心的感谢。由于编者水平有限，书中不妥之处，敬请读者批评指正。

编　者

2013 年 1 月

目 录

第1章 计算机基础知识

按进位的方法进行计数，称为**进位计数制**，简称计数制。本章将从十进制数入手，再将数的基本概念引申到二进制、八进制、十六进制等进位制。要求了解各种计数制的基本概念、书写规则，尤其要熟练掌握这几种进位制数的相互转换，了解计算机中符号数的表示方法及运行原理。最后介绍数的浮点、定点表示方法，以及字符的表示法。尽管这些内容比较简单，但对于学习微型计算机原理来说，它们是必不可少的基础知识，应当熟练掌握。

1.1 数制及其转换

1.1.1 进位计数制

1. 十进位计数制

十进位计数制是使用最广，人们最熟悉的一种计数方式。

在十进制中，用0,1,2,…,9十个数字表示数的大小。在进位计数制中，常用**基数**来区别不同的计数制，而某种计数制的基数就是表示该进位制所用的数码或字符的个数。因为十进制中有十个数字，所以十进位计数制的基数为10。

在各种计数制中采用位值法则，即对每个数位赋予一定的位值，又称**权**。也就是说，对于同一个数码在不同的位，它代表的数值就不同，或者说不同位的权是不一样的。每个数位的权由基数的 I 次幂确定（I 为正、负整数）。

例如，4 539.31 这个数可以写成：

$$4\,539.31 = 4\times10^{3} + 5\times10^{2} + 3\times10^{1} + 9\times10^{0} + 3\times10^{-1} + 1\times10^{-2}$$

我们称上式为十进制数的“按权展开式”。按照“权”的定义，对于十进制来讲，上式中的 10^{0},10^{1},10^{2},10^{3} 等就是整数的权；而 10^{-1},10^{-2},10^{-3} 等就是小数的权。我们经常说的个位、十位、百位、千位、万位以及十分位、百分位等说的就是权。从上式可以看出，某位数的值是用该数乘以该位的权得到的。

十进制的主要特点：

（1）有十个不同的数码（或称数字符号）：0,1,2,3,4,5,6,7,8,9。因为其基数为10，所以称为十进制。

（2）按“逢10进1”的原则计数。即在计数时，每计到10就向左进一位，或者说，上一位的权是下一位的权的10倍。

（3）十进制数的三种表示方法：①在数的右下脚注明数制，如 $(349.602)_{10}$。②在数的后面加规定的符号“D”，如1257D。③允许十进制数后面什么也不加，如407.65。本书采用第③种方法。

2. 二进位计数制

计算机的最基本功能是对数据进行计算、处理。但计算机只能识别二进制数，可以说，

二进制及其编码是所有计算机最基本的语言。

二进制是计数制中最简单的一种，表示二进制数的数码只有两个：0 和 1。

与十进位计数制相似，二进制也采用权值法则，即每个数位的权由基数 2 的 I 次幂确定。

例如，依照十进制按权展开式的写法，二进制数 1011.11 的按权展开式可表示为

$$(1011.11)_2 = 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2}$$

从上式中可知，二进制数各位的权依次为…，2^3，2^2，2^1，2^0，2^{-1}，2^{-2}，…。

二进制的主要特点：

(1)有两个不同的数码：0，1。因为其基数为 2，所以称为二进制。

(2)按“逢 2 进 1”的原则计数。即在计数时，每计到 2 就向左进一位，或者说，上一位的权是下一位的权的 2 倍。

(3)二进制数的两种表示方法：①在数的右下脚注明数制，如$(1101001.011)_2$；②在数的后面加规定的符号“B”，如 11101010B。

3. 八进位计数制

八进位计数制采用 0，1，2，…，7 这样八个数字来表示。每个八进制数由 3 位二进制数构成。同前面介绍的十进制、二进制一样，八进位计数制中的不同位也具有不同的权，它的权是用 8 的 I 次幂来表示的。

例如，八进制数 6372.14 的按权展开式为

$$(6372.14)_8 = 6\times8^3 + 3\times8^2 + 7\times8^1 + 2\times8^0 + 1\times8^{-1} + 4\times8^{-2}$$

从上式中可知，八进制数各位的权依次为…，8^3，8^2，8^1，8^0，8^{-1}，8^{-2}，…。

八进制的主要特点：

(1)有八个不同的数码：0，1，2，3，4，5，6，7。因为其基数为 8，所以称为八进制。

(2)按“逢 8 进 1”的原则计数。即在计数时，每计到 8 就向左进一位，或者说，上一位的权是下一位的权的 8 倍。

(3)八进制数的两种表示方法：①在数的右下脚注明数制，如$(4510.01)_8$；②在数的后面加规定的符号“Q”，如 315Q。

4. 十六进位计数制

十六进位计数制是计算机中最常用的一种进位制。它在数的结构上类似于八进制，易于与二进制转换，在数据的表示、存储、输入、输出和显示中更优于八进制。本书中主要使用十六进位计数制。

十六进位计数制的基数为 16，即由 16 个不同的符号数码组成。除了 0～9 十个数字外，还用字母 A，B，C，D，E，F 分别表示 10，11，12，13，14，15。每个十六进制数由 4 位二进制数构成。十六进位计数制的权是用 16 的 I 次幂来表示的。

例如，十六进制数 23CA.59 的按权展开式为

$$(23CA.59)_{16} = 2\times16^3 + 3\times16^2 + 12\times16^1 + 10\times16^0 + 5\times16^{-1} + 9\times16^{-2}$$

十六进制的主要特点：

(1)有十六个不同的数码和符号：0，1，2，3，4，5，6，7，A，B，C，D，E，F。因为其基数为 16，所以称为十六进制。

(2)按“逢 16 进 1”的原则计数。即在计数时，每计到 16 就向左进一位，或者说，上一位的权是下一位的权的 16 倍。

(3)十六进制数的两种表示方法:①在数的右下脚注明数制,如$(4FE.05A)_{16}$;②在数的后面加规定的符号“H”,如21FH。

在这里,还要特别强调一点,按照汇编语言的规定:凡是计算机所表示的数,不管是哪一种进制的数,都必须以数字0~9作为开头。例如,十六进制数F6H,因为它的第一个符号不是数字,所以必须在开头加一个“0”,写成0F6H。

1.1.2 进位数制之间的转换

在微型计算机原理的学习和应用中,经常要遇到各种进制的相互转换问题。尤其在本书中经常用到十进制、二进制、十六进制,大家应熟练地掌握它们之间的相互转换的方法。

1. 十进制数与二进制数的转换

(1)二进制数→十进制数

如上所述,只要将二进制数中数码为1的每一位乘上它的权后加起来就可以得到二进制数对应的十进制数值。

例如,二进制数11011.01换算成十进制数为

$$(11011.01)_2 = 1\times2^4+1\times2^3+1\times2^1+1\times2^0+1\times2^{-2}$$
$$= 16+8+2+1+0.25 = 27.25$$

(2)十进制数→二进制数

十进制数转换为二进制数分两部分讨论,即整数部分和小数部分。

①十进制整数的转换方法

十进制整数转换为二进制整数的原则是“除2求余”。即对十进制整数连续除以2,每次相除所得的余数就构成所要转换的二进制整数,所得的整数商继续被2除,直到商为零为止,最后所得的余数,就是所转换的二进制数的最高位。

例如,将十进制整数187转换成二进制整数,过程如下:

相除	余数	
187÷2=93	1	最低位
93÷2=46	1	
46÷2=23	0	
23÷2=11	1	
11÷2=5	1	
5÷2=2	1	
2÷2=1	0	
1÷2=0	1	最高位

得到的二进制整数为187=10111011B。

②十进制小数的转换方法

十进制小数转换为二进制小数的原则就是“乘2求整”。即对十进制小数连续乘2,每乘一次后取出乘积整数部分上的0或1,并将小数部分继续乘2,直到相乘结果的小数部分为零或达到一定的精度为止,这时所取出的整数就构成了要转换的二进制小数。开始取得的整数为二进制小数的最高位,最后取得的整数就是二进制小数的最低位。

例如,将十进制小数0.71875转换成二进制小数,过程如下:

相乘	整数部分	
0.718 75 × 2 = 1.437 5	1	最高位
0.437 5 × 2 = 0.875	0	
0.875 × 2 = 1.75	1	
0.75 × 2 = 1.5	1	
0.5 × 2 = 1.0	1	最低位

得到的二进制小数为 0.718 75 = 0.10111B。

对于既有整数又有小数的十进制数，可以用“除 2 求余”和“乘 2 求整”的方法，分别对其整数、小数部分进行转换，然后合并。

例如，187.718 75 = 10111011.10111B。它的求解过程就是上面求出结果的过程组合。

2. 十六进制数与二进制数的转换

在微机中，为使二进制数书写形式简单，经常用十六进制数表示二进制数。例如，存储器的地址码一般由 16 或 20 位二进制数构成，内存地址单元的内容为 8 位二进制数，寄存器的内容也是 8 位、16 位二进制数。如果用二进制数表示的话，难记，难写，易错，一般都用十六进制数来表示二进制数。因此，要熟练地掌握十六进制数与二进制数之间的相互转换。

(1) 二进制数→十六进制数

二进制数转换为十六进制数非常简单、方便。因为 4 位二进制数的组合恰好等于 0 ~ 15 这 16 个数值，所以可用一位十六进制数表示 4 位二进制数。

一个二进制数的整数部分要转换成十六进制数时，可以从小数点开始向左，按 4 位一组分成若干组，最高一组不足 4 位时，在左边加 0 补足 4 位。

二进制数的小数部分可以从小数点开始向右，按 4 位一组分成若干组，最低一组不足 4 位时，在右边加 0 补足 4 位。然后将每一组的 4 位二进制数用相应的十六进制数表示，即为转换的十六进制数。

例如，二进制数 110100110.110101 转换为十六进制数的过程如下：

(整数)← | →(小数)

(000)1　1010　0110.　1101　01(00)

↓　↓　↓　↓　↓

1　A　6　D　4

即为 110100110.110101B = 1A6.D4H。

(2) 十六进制数→二进制数

十六进制数转换为二进制数时，只要将十六进制数的每一位用其等值的 4 位二进制数代替，连在一起就得到了所需要的二进制数。

例如，十六进制数 7FD3.5CH 转换为二进制数的过程如下：

7FD3.5CH = (0)111　1111　1101　0011.0101　11(00)B

注意在上面的结果中，已把出现在最高位前面或最低位后面的无效 0 舍去了。

3. 十进制数与十六进制数的转换

一个十进制数转换十六进制数与十进制数转换二进制数相似。其方法就是将十进制的整数部分采用“除 16 求余”的方法得到十六进制数的整数部分。而十进制的小数部分采用“乘 16 求整”的方法得到十六进制数的小数部分。

例如，将十进制数 47 632.781 25 转换十六进制数，其转换过程如下：

整数部分	余数
47 632 ÷ 16 = 2 977	0
2 977 ÷ 16 = 186	1
186 ÷ 16 = 11	10 (A)
11 ÷ 16 = 0	11 (B)
小数部分	整数
0.781 25 × 16 = 12.5	12 (C)
0.5 × 16 = 8.0	8

结果为47 632.781 25 =0BA10.C8H(因为十六进制数的第一个符号是B,所以前面必须加数字0)。

一个十六进制数转换为十进制数时,只要把十六进制数按权展开,然后相加即可。

例如,十六进制数9FE4.A7H转换十进制数过程如下:

$$\begin{aligned}9FE4.A7H &= 9\times16^3+15\times16^2+14\times16^1+4\times16^0+10\times16^{-1}+7\times16^{-2}\\ &= 36\,864+3\,840+224+4+0.625+0.027\,3\approx40\,932.652\end{aligned}$$

十六进制数和十进制数的转换也可以采用二进制作为中间媒介,即把十六进制数或十进制数先转换成二进制,再转换成其他进制。

4.八进制数与其他进制数的转换

由于八进制数与十六进制数的构成极为相似,因此八进制数与其他进制数的转换,只要参照十六进制数与其他进制数转换的方法,极易实现,在此不再赘述。

1.2 计算机中数的表示

1.2.1 无符号数

计算机中经常要处理的数分为两种:一种是无符号数,一种是带符号的数。

顾名思义,无符号数就是没有符号位的数。如果一个二进制数有N位,这N位数码均表示数值部分,没有任何一位是表示“+”“-”符号的,这就是无符号数。我们在前面介绍的都是二进制无符号数。二进制无符号整数的数值范围:$0\sim(2^N-1)$。其中,N为二进制无符号整数的位数。八位无符号整数的数值范围是0~255。

1.2.2 符号数的表示方法

我们把带符号位的数称为**符号数**。对于二进制符号数,其正负号如何表示呢?在计算机中,数学中的“+”“-”符号同样数值化。规定:一个二进制数的最高位为符号位,在符号位中,用“0”代表正号,“1”代表负号,而其余的位表示该数的数值部分。这样就把一个数的数值和符号全都数值化了。

我们把一个符号数(包括符号位)在机器中的一组二进制数表示形式,称为**机器数**。而把它所表示的数值称为该机器数的**真值**。

由于编码的不同,符号数表示的形式有三种方法:原码、反码、补码。目前计算机中实际使用的是补码。下面我们从了解原码和反码开始,目的还是为了更好地熟悉、掌握补码。

8 位二进制数是计算机运算和传送的基本单位，称为**字节**。因此，下面的举例都是用 8 位二进制数（字节）来表示符号数的各种编码。

1. 原码

原码表示：二进制数的最高位（符号位）用 0 表示正数，用 1 表示负数，其余的位表示数值。在下面的举例中，为了让大家看得更清楚，我们有意把符号位与其后面的部分拉开一些。

例如：

$$[+100]_{原}=0\ 1100100$$
$$[+127]_{原}=0\ 1111111$$
$$[-100]_{原}=1\ 1100100$$
$$[-127]_{原}=1\ 1111111$$

符号位 数值部分

若用 8 位二进制表示一个符号数原码时，最高位的 0 表示正号，而最高位的 1 表示负号。符号位后面的 7 位就是数值部分。例如，$[Y]_{原}=1\ 0000011$，则 Y 的真值为 -3；而 $[Z]_{原}=0\ 0000011$，则 Z 的真值为 $+3$。

对于 0，可以认为它是（+0），也可以认为它是（-0）。因此，0 在原码中有两种表示：

$$[+0]_{原}=0\ 0000000 \qquad [-0]_{原}=1\ 0000000$$

显然，8 位二进制数的原码可表示的数值范围：$-127 \sim +127$。N 位二进制数的原码可表示的数值范围：$-(2^{N-1}-1) \sim +(2^{N-1}-1)$。

原码表示简单易懂，而且与真值的转换也很方便。但计算机采用原码进行加、减运算的话，这个运算是相当麻烦的。譬如，两个符号数相加，首先必须判断这两个数的符号是否相同，如果相同，则做加法，否则做减法。做减法时，还得比较两个数绝对值的大小，再用大数减小数，而差的符号又要与绝对值大的数的符号相同。从理论上讲，设计这样的计算机是没有问题的，但是运算器逻辑结构设计复杂，运算、判断的时间增长，因此在微型机问世前就基本不为人们所用了。

2. 反码

反码表示：正数的反码与原码相同，即反码的最高位（符号位）用 0 表示正数，其余的位表示数值（数字位）。

例如：

$$[+0]_{反}=0\ 0000000$$
$$[+100]_{反}=0\ 1100100$$
$$[+127]_{反}=0\ 1111111$$

符号位 数值部分

负数的反码是将它的正数按位求反得到的。

例如：

$$[-0]_{反}=1\ 1111111$$
$$[-100]_{反}=1\ 0011011$$
$$[-127]_{反}=1\ 0000000$$

符号位 数字位

若用8位二进制表示一个符号数反码时，最高位为符号位。若符号位为0（正数）时，后面的7位就是数值部分；若符号位为1（负数）时，后面的7位数字位表示的并不是该数的数值（因此写成数字位），必须把它们按位求反后，才能得到该数的数值部分。

例如，$[Y]_{反}=0\ 0000100$，则 Y 的真值为 $+4$；$[Z]_{反}=1\ 0000001$，Z 的真值并不是 -1，而是 -126。

同原码一样，对于0，其反码也有两种表示：

$$[+0]_{反}=0\ 0000000 \qquad [-0]_{反}=1\ 1111111$$

显然，8位二进制数的反码可表示的数值范围：$-127 \sim +127$。N 位二进制数的反码可表示的数值范围：$-(2^{N-1}-1) \sim +(2^{N-1}-1)$。

计算机中符号数也很少采用这种编码。

3. 补码

补码表示：正数的补码与原码、反码相同，即补码的最高位（符号位）用0表示正数，其余的位表示数值。

例如：

$$[+100]_{补}=0\ 1100100$$
$$[+127]_{补}=0\ 1111111$$
$$\uparrow \quad \uparrow$$
符号位 数值部分

负数的补码是将它的对应正数按位求反加1形成的（即求反加1）。

例如：

$$[-100]_{补}=1\ 0011100$$
$$[-127]_{补}=1\ 0000001$$
$$[-128]_{补}=1\ 0000000$$
$$\uparrow \quad \uparrow$$
符号位 数字位

若用8位二进制表示一个符号数补码时，最高位为符号位。若符号位为0（正数）时，后面的7位就是数值部分；若符号位为1（负数）时，后面的7位数字表示的并不是该数的数值（也写成数字位），必须把它们按位求反加1后，才能得到该数的数值部分。

例如，$[Y]_{补}=0\ 0000110$，则 Y 的真值为 $Y=+6$；$[Z]_{补}=1\ 0000110$，Z 的真值并不是 -6。Z 的数值这样求：将数字位0000110按位求反后得到1111001，再加1，即为1111010。因此，$Z=-1111010=-122$。

在补码表示中，0的补码只有一种表示（与原码、反码不同）：

$$[+0]_{补}=[-0]_{补}=0\ 0000000$$

显然，8位二进制数的补码可表示的数值范围：$-128 \sim +127$。N 位二进制数的补码可表示的数值范围：$-2^{N-1} \sim +(2^{N-1}-1)$。

计算机中的符号数都采用补码表示，这样同一个运算器不但可以完成符号数的加法运算，而且两个数的减法运算也可用加法代替。这可使运算器设计简化，速度提高。

二进制数可以表示无符号数，也可以表示符号数。表1-1列出了8位二进制数表示无符号数、符号数的对照表。

表 1-1　8 位二进制数表示无符号数、符号数对照表

二进制数表示	无符号数	原码	反码	补码
00000000	0	+0	+0	+0
00000001	1	+1	+1	+1
⋮	⋮	⋮	⋮	⋮
01111110	126	+126	+126	+126
01111111	127	+127	+127	+127
10000000	128	-0	-127	-128
10000001	129	-1	-126	-127
⋮	⋮	⋮	⋮	⋮
11111110	254	-126	-1	-2
11111111	255	-127	-0	-1

1.2.3　符号数的加、减法运算

对于一个数，它究竟是符号数还是无符号数是人为规定的。对于计算机来讲，当进行二进制数计算时，它并不知道这些二进制数是进行符号数运算还是无符号数运算。可是，当用无符号数的概念或用补码来解释符号数时，运算的结果都是正确的。

例如，有两个数 10000100 和 00001110，若规定是无符号数时，它们分别代表 132 和 14。将这两个数相加：10000100 + 00001110 = 10010010，得到一个无符号数 10010010，该数为 146。这就是说，按照无符号数的概念去解释，运算结果是正确的。

如果把上面两个数规定为符号数的话，其数值分别为 -124 和 +14。两个符号数相加，结果仍为 10010010。该结果若用符号数解释，因为 $[-110]_{补} = 1\ 0010010$，所以结果为 -110，可见运算结果也是正确的。

在计算机中，当确定为符号数运算时，符号数一律用补码表示，运算时符号位和数字位一起参加运算。同样，运算结果也用补码表示。

两符号数作加法运算时，按下面公式进行：

$$[X+Y]_{补} = [X]_{补} + [Y]_{补}$$

上式表明，两数和的补码等于两数的补码之和。

例 1-1　已知 $X=64$，$Y=8$，求 $X+Y=?$

X，Y 用补码表示：$[X]_{补} = 0\ 1000000$；$[Y]_{补} = 0\ 0001000$

$$\begin{array}{rr} [X]_{补} = 0\ 1000000 & 64 \\ +\quad [Y]_{补} = 0\ 0001000 & +\ \ 8 \\ \hline [X]_{补} + [Y]_{补} = 0\ 1001000 & 72 \end{array}$$

即　$[X+Y]_{补} = [X]_{补} + [Y]_{补} = 0\ 1001000$

因为和的补码为正数，所以，$X+Y = +1001000 = +72$，结果是正确的。

例 1-2　已知 $X=78$，$Y=-100$，求 $X+Y=?$

X，Y 用补码表示：$[X]_{补} = 0\ 1001110$；$[Y]_{补} = 1\ 0011100$

$$\begin{array}{rr} [X]_{补}=0\ 1001110 & +78 \\ +\quad [Y]_{补}=1\ 0011100 & +\ -100 \\ \hline [X]_{补}+[Y]_{补}=1\ 1101010 & -22 \end{array}$$

即 $$[X+Y]_{补}=[X]_{补}+[Y]_{补}=1\ 1101010$$

因为和的补码为负数，即$[-22]_{补}=11101010$，所以，$X+Y=-0010110=-22$，结果也是正确的。

两数作减法运算时，按下面公式进行：

$$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$$

从上式中可以看出，两数作减法运算时，先求$[X]_{补}$，再求$[-Y]_{补}$，然后进行补码的加法运算。具体的运算过程与前述的补码的加法过程一样。

例 1-3 已知 $X=-78$，$Y=-15$，求 $X-Y=?$

X，Y 用补码表示：$[X]_{补}=1\ 0110010$；$[Y]_{补}=1\ 1110001$；

对$[Y]_{补}$再求补，就得到$[-Y]_{补}$，因此$[-Y]_{补}=0\ 0001111$

$$\begin{array}{rr} [X]_{补}=0\ 0110010 & -78 \\ +\quad [-Y]_{补}=0\ 0001111 & -\ -15 \\ \hline [X]_{补}+[-Y]_{补}=1\ 1000001 & -63 \end{array}$$

即 $$[X-Y]_{补}=[X]_{补}+[-Y]_{补}=1\ 1000001$$

因为和的补码为负数，所以，$X-Y=-0111111=-63$，结果也是正确的。

1.2.4 溢出及符号数扩展

我们先看一个符号数加法的例子。

例 1-4 已知 $X=64$，$Y=68$，求 $X+Y=?$

X，Y 用补码表示：$[X]_{补}=0\ 1000000$；$[Y]_{补}=0\ 1000100$

$$\begin{array}{rr} [X]_{补}=0\ 1000000 & 64 \\ +\quad [Y]_{补}=0\ 1000100 & +\ 68 \\ \hline [X]_{补}+[Y]_{补}=1\ 0000100 & 132 \end{array}$$

即 $$[X+Y]_{补}=[X]_{补}+[Y]_{补}=1\ 0000100 \leftarrow (-124\ 的补码)$$

在例 1-4 中，由于是两个正数相加，其结果应该也是正数，应为 132 才对，但运算结果却为负数(-124)，这显然是错误的。究其原因是和数 +132 大于 +127 所致，即超过了 8 位二进制补码所能表示的最大值，使数值部分占据了符号位的位置，产生了溢出错误。

如果运算结果超过了补码所能表示的数值范围就称为**溢出**。溢出后，其结果是错误的。计算机设有专门的电路用于判断运算结果，并以标志 OF(溢出标志位)告诉人们这次运算的结果是否溢出。只要结果不溢出(OF=0)，运算结果就是正确的；要是结果溢出(OF=1)，运算结果就是错误的。

判断结果是否溢出的方法有多种，在这里我们只介绍一种最直观的方法。

如果两个异号数相加或两个同号数相减，那么其结果肯定不会溢出(OF=0)。

因此，溢出只能发生在同号数相加(或异号数相减)的时候。譬如，两个同号数相加，若结果的符号与参加运算的数字符号相反，则表明溢出(出错的原因是相加后，最高数值位向符号位产生了进位，使结果的符号变反)；两个异号数相减实际是转换为两个同号数相加来计算的。因而加减法的溢出判断方法可以统一为下列的逻辑表达式：

$$OF = \overline{A}_n \overline{B}_n S_n + A_n B_n \overline{S}_n$$

该表达式中，OF 代表溢出；A_n，B_n，S_n 分别代表相加的两个加数和结果的符号位。该式说明：两个同号数相加后，所得结果的符号如果与加数的符号数不相同，则表明发生了溢出（这时 OF = 1）。在例 1 - 4 中，两个加数都是正数，但结果却是负数，说明已溢出，结果错误。

为了避免溢出，最好的办法就是增加符号数的位数，使数值范围加大。如在例 1 - 4 中，若用 9 位二进制来表示符号数的话，结果就不会溢出了。但考虑到数据的存储与运算都是与字节的位数相关的，因此，增加符号数的位数一般都采用字节（8 位）的倍数，即 8 位、16 位、24 位等。

例如，符号数 123，用 8 位二进制补码表示：$[+123]_{补} = 0\ 1111011$；若扩展为 16 位，则 $[+123]_{补} = 0\ 0000000\ 01111011$。

再如，符号数 - 123，用 8 位二进制补码表示：$[-123]_{补} = 1\ 0000101$；若扩展为 16 位，则 $[-123]_{补} = 1\ 1111111\ 10000101$。

上面两个例子中，数字下面画杠的部分为原 8 位二进制补码。可以看出，原为正数的补码，其扩展的位数全部加 0；原为负数的补码，其扩展的位数全部加 1。

1.3 十进制数与字符的编码

1.3.1 BCD 码

因为二进制数容易用器件实现，运算规律也十分简单，所以计算机都采用二进制数。但就人们习惯来说，对十进制数更熟悉，用起来更直观、方便，遗憾的是计算机无法直接进行操作。为此，人们发明一种特殊的二进制编码，它兼有二进制和十进制计数的特点，既符合人们的习惯，计算机又能直接进行操作。人们把这种特殊的编码叫做**二—十进制编码**，简称 **BCD**（Binary Coded Decimal）**码**。

1. 8421 BCD 码

用二进制编码来表示十进制数的方案有多种。它们都是用多位二进制数编码来表示一位十进制数的。经常使用的一种方案是 8421 BCD 码，其编码原则：每位十进制数用 4 位等值的二进制数来表示，因从左到右各位二进制的权为 8421，故名 8421 BCD 码。表 1 - 2 列出了十进制数与 8421 BCD 码的对应关系。

表 1 - 2 十进制数与 BCD 码的对应关系

十进制数	8421 BCD 码	十进制数	8421 BCD 码
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

（1010 ~ 1111 为非法码）

从表 1 -2 中可以看到,BCD 码仅仅利用了 4 位二进制编码的 10 种组合。而代表十进制数 10 ~ 15 的二进制编码,对于 BCD 码来说是非法的。很明显,BCD 码只利用了二进制中对应的 0 ~ 9 的 10 种组合,余下的 6 种组合是不允许使用的。

根据上述定义,一个十进制数,能够很方便地用 BCD 码来表示。例如,十进制数 9 521 用 BCD 码可表示为

$$9\ 521 = (1001\ 0101\ 0010\ 0001)_{BCD}$$

为了避免 BCD 码与纯二进制码混淆,在 BCD 码表示中,每位 BCD 码(4 位二进制数)写成一组,中间留有空隙,而且要标明此数为 BCD 码。

BCD 码的优点:10 个 BCD 码组合形式容易记忆。一旦熟悉了 4 位二进制数的表示,BCD 码就可以像十进制数一样迅速地读出。同样,也可以很快地得到以 BCD 码表示的十进制数。

例如,将下面的 BCD 码转换成相应的十进制数:

$$(1000\ 0111\ 0100\ 0101.\ 0001\ 0110\ 0011)_{BCD} = 8\ 745.\ 163$$

可见十进制数与 BCD 码之间的转换是非常方便的。而二进制数与 BCD 码之间的转换却不能直接实现,必须先转换成十进制。

例如,将二进制数 1101.01 转换成相应的 BCD 码。

首先,将二进制数 1101.01 转换成十进制数 13.25,再把十进制数 13.25 转换成相应的 BCD 码:

$$1101.01B = 13.25 = (0001\ 0011.\ 0010\ 0101)_{BCD}$$

BCD 码在计算机中常用两种方法表示。

第一种如上所述,用 4 位二进制编码来表示一位十进制数,称为**压缩十进制编码**(或**压缩 BCD 码**)。用这种表示方法,8 位二进制数(一个字节)可以表示两位十进制数。例如,$68 = (0110\ 1000)_{BCD}$。

十进制数的另一种表示方法称为**非压缩 BCD 码**(也称**扩展 BCD 码**)。它利用 8 位二进制数表示一位十进制数。这种编码规定:它的低 4 位以 8421 码表示十进制数(与压缩 BCD 码相同),而高 4 位没有意义,可为任意值。例如,$5 = (\times\times\times\times 0101)_{BCD}$;$68 = (\times\times\times\times 0110\ \times\times\times\times 1000)_{BCD}$。其中"×"表示任意值。

2. BCD 码运算及调整

因为计算机只能进行二进制运算,所以在计算机中设置的加、减法指令都是二进制的运算指令。为了便于 BCD 码的运算,在计算机的指令系统中又提供了一组十进制调整指令,即将二进制的运算结果直接进行调整,得到十进制数的正确结果。

为什么用普通的二进制运算指令对 BCD 码运算时,需对其结果进行十进制调整呢?如何调整?下面通过一个简单的例子,说明十进制的调整原理(以压缩 BCD 码为例)。

例 1 -5 计算 9 +6 =?

计算如下:

```
    0000 1001      09H     ;9 的压缩 BCD 码
+   0000 0110      06H     ;6 的压缩 BCD 码
    0000 1111      0FH     ;二进制数
```

运算结果为 0FH。在 BCD 编码中,0FH 属于非法码,必须加以调整。那么如何调整呢?我们知道 BCD 码运算是"逢 10 进 1",但加减法指令进行的运算却是二进制运算,对 4 位二