高职高专“十一五”规划教材

汇编语言案例教程

HUIBIAN YUYAN
ANLI JIAOCHENG

张开成　钟文龙　编著

化学工业出版社

高职高专“十一五”规划教材

汇编语言案例教程

HUIBIAN YUYAN
ANLI JIAOCHENG

张开成　钟文龙　编著

化学工业出版社
·北京·

本书以 Intel 系列基本微处理器 8086/8088 为对象，主要介绍微型计算机的基础知识和工作过程、体系结构和寻址方式、指令系统及汇编语言程序设计的方法和技巧。

本书内容简明扼要、深入浅出、案例丰富、通俗易懂，融入作者从事多年教学及工程实践应用的体会和经验。本书首先引入了一个模型机执行 5 条指令构成的简短程序的教学案例，直观地描述了微型计算机的工作过程，为初学者学习汇编语言程序设计奠定基础；然后把寻址方式和微机硬件系统，即 CPU 的硬件结构放在一起讲解，更有利于初学者对问题的理解，便于突破难点。本书列举大量案例说明汇编语言程序设计的方法和技巧，强调应用，有利于培养初学者分析问题和解决问题的能力。通过本书学习，将为微机应用打下基础。本书还配有教学课件和案例库，教学内容丰富，课件功能完备，操作方便快捷，其中，微机工作过程和寻址方式等都已设计成动画。它将成为教师课堂授课和初学者学习汇编语言程序设计的有力帮手。

本书可作为高职高专计算机相关专业的教学用书，也可作为一般工程技术人员的参考用书。

图书在版编目 (CIP) 数据

汇编语言案例教程/张开成，钟文龙编著. —北京：化学工业出版社，2009.1
高职高专“十一五”规划教材
ISBN 978-7-122-04122-7

Ⅰ. 汇… Ⅱ. ①张…②钟… Ⅲ. 汇编语言-程序设计-高等学校：技术学院-教材 Ⅳ. TP313

中国版本图书馆 CIP 数据核字（2008）第 180549 号

责任编辑：王听讲　　文字编辑：丁建华　孙思晨
责任校对：周梦华　　装帧设计：刘丽华

出版发行：化学工业出版社（北京市东城区青年湖南街 13 号　邮政编码 100011）
印　　装：北京永鑫印刷有限责任公司
装　　订：三河市前程装订厂
787mm×1092mm　1/16　印张 13　字数 315 千字　2009 年 3 月北京第 1 版第 1 次印刷

购书咨询：010-64518888（传真：010-64519686）　售后服务：010-64518899
网　　址：http://www.cip.com.cn
凡购买本书，如有缺损质量问题，本社销售中心负责调换。

定　　价：23.00 元

前　言

汇编语言是一种面向机器的语言。它能够充分利用计算机的全部硬件特性，并且能够直接控制计算机的所有硬件，在微型计算机系统的开发应用和过程控制中得到广泛应用，因此特别受到重视。

考虑到国内当前广泛使用 8086/8088 和 80X86 系列微型计算机的实际情况，本书以 8086/8088 系统的汇编语言作为学习的对象。读者学习了 8086/8088 系统的汇编语言程序设计后，掌握更高层次的汇编语言也就有了基础。

本书在编写过程中突出了以下四大特点：1. 首先在第 1 章引入了一个模型机执行 5 条指令构成的一个简短程序的教学案例，直观地描述了微型计算机的工作过程，为初学者学习汇编语言程序设计奠定基础；2. 介绍指令与程序设计融为一体，由浅入深，循序渐进，打破了把指令集中在一章介绍，程序单独讲解的传统教法，避免了初学者单纯学习指令的枯燥无味；3. 把寻址方式和微机硬件系统，即 CPU 的硬件结构放在一起讲解，更有利于初学者对问题的理解，便于突破难点；4. 列举大量案例说明汇编语言程序设计的方法和技巧，强调应用，有利于初学者分析问题和解决问题能力的培养。

本书共分 8 章。第 1 章，主要介绍汇编语言基础知识和微机的工作过程；第 2 章，讲述微机的硬件结构和寻址方式；第 3 章，介绍 3 种结构的汇编语言程序设计；第 4 章，介绍常用数据处理技术；第 5 章，讲述子程序设计；第 6 章，讲述汇编语言中的高级编程技术；第 7 章，介绍输入输出方法和中断技术；第 8 章，介绍计算机键盘和屏幕控制技术。为突出应用，从第 3 章起，每章都安排有适量的程序设计案例。这些程序设计案例经过调试生成可执行文件后，通过运行都能直观地看到结果，为读者分析理解问题提供方便。

本书配有教学课件和案例库，如有需要可与 wtjiang@sina.com联系，或到化学工业出版社网站（http://www.cip.com.cn）免费下载使用。

本书除第 8 章由张开成和钟文龙共同编写外，其他各章均由张开成编写。全书由张开成统稿、定稿。限于编者的水平，且时间仓促，书中如有不妥之处，恳请读者不吝赐教、指正。

编　者

2008 年 10 月

目　录

第1章 概 述

汇编语言是直接在计算机硬件之上工作的编程语言，首先要了解硬件系统的结构和计算机执行指令的过程，才能有效地应用汇编语言进行编程。在本章中，先讨论什么是汇编语言，然后学习数和编码的知识，最后简要说明微机的工作过程。

1.1 汇编语言简介

1.1.1 机器语言

计算机的所有操作都是在指令的控制下进行的。能够直接控制计算机完成指定动作的是机器指令。一条机器指令是一个由 0 和 1 组成的二进制代码序列，不同的机器指令对应的二进制代码序列也各不相同。一条机器指令通常由操作码和操作数两部分构成，操作码在前，操作数在后。

操作码部分用来指出这条指令要求计算机做什么样的操作，是做加法，做减法，还是完成数据传送，或者是其他的操作；操作数部分给出参与操作的数据值，或者指出操作对象在什么地方。下面的二进制代码序列就是一条 8086/8088 的机器指令：

10000000 00000110	01100100 00000000 00010010

这条指令的前 16 位是操作码部分，含义是要求计算机做两个数的加法操作；后 24 位是操作数部分。第 17 位至 32 位指出第一个加数在内部存储器的编号为 100 的那个字节中，最后 8 位指出另一个加数就在指令中，是 18。

对于同样的二进制序列，不同型号的 CPU 对它的“理解”是不一样的，比如上面的那一串二进制代码在 8086 和 8088 看来是要求做加法，换到另一种 CPU 中完全可能被当作是另一种操作，甚至是错误的指令，所以机器指令与机器本身有着紧密的联系。不同型号的计算机（准确地说是不同型号的 CPU）都有自己的一套指令，一种机型的所有机器指令的集合就是它的指令系统。指令系统及其使用规则构成这种计算机的机器语言。选择指令系统中的指令并排列起来，可以构成一个指令序列，用以告诉计算机完成一连串的动作，这就是一个机器语言程序。

1.1.2 汇编语言

早期的程序员们很快就发现了使用机器语言带来的麻烦，它是如此难于辨别和记忆，给整个产业的发展带来了阻碍。于是汇编语言产生了。

汇编语言的主体是汇编指令。汇编指令和机器指令的差别在于指令的表示方法上。汇编指令是机器指令便于记忆的书写格式，它是用英文单词的缩写符号来表示机器指令的操作码，用英文字母和数字符号来表示操作数。汇编语言是由汇编指令、伪指令（参见 4.2 节）及编程的语法规则所组成。

例如：机器指令 1000100111011000 表示把寄存器 BX 的内容送到 AX 中。汇编指令则写成 mov ax,bx。这样的写法与人类语言接近，便于阅读和记忆。

操作：寄存器 BX 的内容送到 AX 中

机器指令：1000100111011000

汇编语言：mov ax,bx

寄存器，简单地讲是 CPU 中可以存储数据的器件，一个 CPU 中有多个寄存器。AX 是其中一个寄存器的代号，BX 是另一个寄存器的代号。更详细的内容将在第 2 章中讲述。

此后，程序员们就用汇编指令编写源程序。可是，计算机能读懂的只有机器指令，那么如何让计算机执行程序员用汇编指令编写的程序呢？这时，就需要有一个能够将汇编指令转换成机器指令的翻译程序，这样的程序被称为编译器。程序员用汇编语言写出源程序，再用编译器将其编译为机器码，由计算机最终执行。图 1-1 描述了这个工作过程。

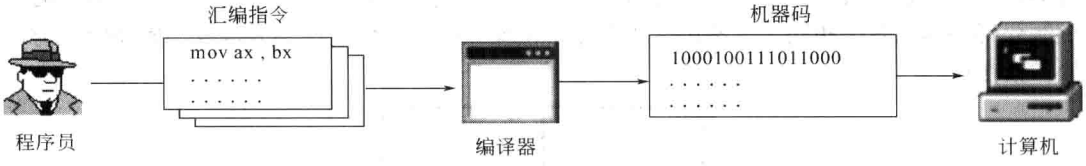

图 1-1　用汇编语言编写程序的工作过程

1.1.3　汇编语言的组成和特征

汇编语言由以下 3 类指令组成。

① 汇编指令：机器码的助记符，有对应的机器码。

② 伪指令：没有对应的机器码，由编译器执行，计算机并不执行。

③ 其他符号：如+、-、*、/等，由编译器识别，没有对应的机器码。

汇编语言的核心是汇编指令，它决定了汇编语言的特征：

① 与机器语言相比，汇编语言易于理解和记忆；

② 编写的源程序可读性较强；

③ 汇编语言能直接控制计算机的内存和外设。

1.1.4　编辑程序、汇编程序和连接程序

汇编语言编辑程序是一种把人们编写的汇编语言程序编辑输入到计算机中的编辑工具，这样的编辑工具很多，在此仅给大家推荐一种叫做 QE.EXE 的编辑软件，这种软件界面友好、操作简便、编辑功能较强，将随本书的教学课件和案例库一起提供给大家。汇编语言程序编辑输入到计算机以后必须存盘，存盘时一般以 ASM 作为文件扩展名，存盘后的这个文件被称作汇编语言源程序。

汇编程序是一种计算机软件，属于软件分类中的系统软件部分，它能够把人们编写的汇编语言源程序翻译成机器语言，这种翻译操作称为“汇编”。汇编程序还具有语法检查的功能，

交给汇编程序进行处理的源程序在翻译之前都必须经过语法检查这一关。这里所使用的汇编程序是 MASM.EXE。

汇编程序翻译的结果已具备机器语言的形式，称为“目标程序”，一般以 OBJ 作为文件扩展名。这种目标程序计算机还无法执行，还必须通过连接程序连接生成扩展名为 EXE 的可执行文件，计算机才能执行。这里所使用的连接程序是 LINK.EXE。

汇编语言源程序、汇编程序、目标程序、连接程序、执行文件的关系如图 1-2 所示。

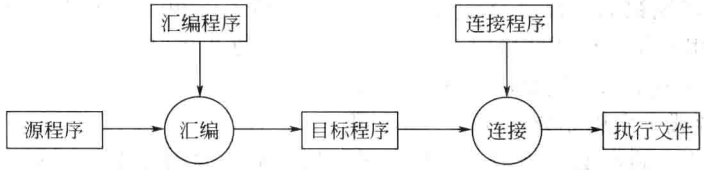

图 1-2 由汇编语言源程序到执行文件的处理过程

1.2 计算机中的数和编码

1.2.1 计算机中的数制

1. 常用进位计数制

1）十进制

十进制数人们最熟悉，但机器实现起来困难。

2）二进制

二进制数由于每位只需两个状态“0”或“1”，机器实现容易，因而二进制是数字系统唯一认识的代码，但二进制书写太长。

3）八进制

$2^3=8$，所以三位二进制数可用一位八进制数表示。

4）十六进制

$2^4=16$，所以四位二进制数可用一位十六进制数表示。

在计算机应用系统中，二进制主要用于机器内部的数据处理，八进制和十六进制主要用于书写程序，十进制主要用于运算最终结果的输出。

2. 数制转换

1）非十进制数转换成十进制数

不同数制之间的转换方法有若干种，把非十进制数转换成十进制数采用按权展开相加法。具体步骤是首先把非十进制数写成按权展开的多项式，然后按十进制数的计数规则求和。

2）十进制数转换成二进制数

对于既有整数部分又有小数部分的十进制数转换成二进制数，首先要把整数部分和小数部分分开转换，再把两者的转换结果相加。具体方法介绍如下。

（1）整数转换

方法：连续除以 2 取余数，直到商为 0，并按照和运算过程相反的顺序把各个余数排列起

来，即为二进制整数。

（2）小数转换

方法：连续乘 2 取整数，并按照和运算过程相同的顺序把各个整数排列起来，即为二进制小数。

3）二进制数转换成八进制数或十六进制数

二进制数转换成八进制数或十六进制数时，其整数部分和小数部分可以同时进行转换。其方法是：以二进制数的小数点为起点，分别向左、向右，每三位或四位分一组。对于小数部分，最低位一组不足三位或四位时，必须在有效位右边补 0，使其足位。然后把每一组二进制数转换成八进制或十六进制数，并保持原顺序。

4）八进制数或十六进制数转换成二进制数

八进制数或十六进制数转换成二进制数时，只要把八进制数或十六进制数的每一位数码分别转换成三位或四位二进制数，并保持原排序即可。整数最高位一组左边的 0 及小数最低位一组右边的 0 可以省略。

1.2.2 符号数的表示

1. 机器数与真值

二进制数与十进制数一样有正有负。在计算机中，常把数的符号和数值部分一起编码后表示带符号数。常用的带符号数编码方法有原码、反码和补码表示法。这几种表示法都将数的符号数码化。通常正号用“0”表示，负号用“1”表示。为了区分一般书写时表示的数和机器中编码表示的数，称前者为真值，后者为机器数，即数值连同符号数码“0”或“1”一起作为一个数称为机器数，而它的数值连同符号“+”或“-”称为机器数的真值。

为了表示方便，常把 8 位二进制数称为字节，把 16 位二进制数称为字，把 32 位二进制数称为双字。对于机器数应将其用字节、字或双字表示，因此只有 8 位、16 位或 32 位机器数的最高位才是符号位。

1）原码

如上所述，数值部分用该数的绝对值，符号部分，用“0”表示正数，用“1”表示负数，这样表示数的方法称带符号数的原码表示法，所表示的数称原码。

y1=+127=+1111111B　　[y1]原=01111111B

y2=−127=−1111111B　　[y2]原=11111111B

其中最高位为符号，后面 7 位是数值。用原码表示时，+127 和−127 的数值部分相同而符号位相反。

8 位整数原码表示的数值范围为 FFH～7FH，即−127～+127。16 位整数原码表示的数值范围为 FFFFH～7FFFH，即−32767～+32767。

原码表示简单易懂，而且与真值的转换方便。但若是两个异号数相加，或两个同号数相减，就要做减法。为简化计算机的结构，常把减法运算转换为加法运算，因而引入了反码和补码。

2）反码

正数的反码与原码相同；负数的反码为它的原码的数值部分按位取反，而符号位不变，即仍为“1”。

y1=+127=+1111111B　　[y1]反=01111111B

y2=−127=−1111111B　　[y2]反=10000000B

3）补码

正数的补码与原码相同；负数的补码为它的原码的数值部分按位取反，最末位再加 1，而符号位不变，即仍为“1”。

y1=+127=+1111111B　　[y1]补=01111111B

y2=−127=−1111111B　　[y2]补=10000001B

8 位补码数表示的数值范围为 80H～7FH，即−128～+127。16 位补码表示的数值范围为 8000H～7FFFH，即−32768～+32767。

2. 原码、反码、补码和真值的相互转换关系

前面已经讨论了符号数的真值表示和在机器中的表示，下面总结一下机器数和真值的相互转换关系。

正数的原码、反码、补码相同，即正号“0”加上绝对值的数值位表示。

负数的机器码的相互转换关系，如图 1-3 所示。

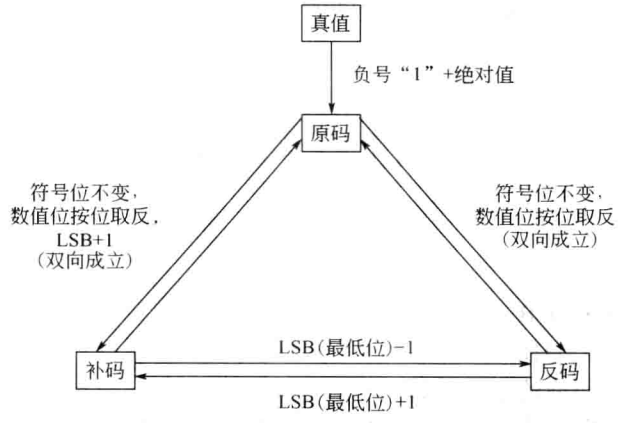

图 1-3　负数的机器码的相互转换关系

下面举例说明符号数的机器表示和真值的相互转换关系。以负数情况为例进行分析。

1）已知原码，求补码

【例 1-1】已知某数 X 的原码为 10110100B，试求 X 的补码。

【解】如图 1-4 所示。由[X]原=10110100B，X 为负数。求其补码表示时，符号位不变，数值部分按位取反，再在末位加 1。

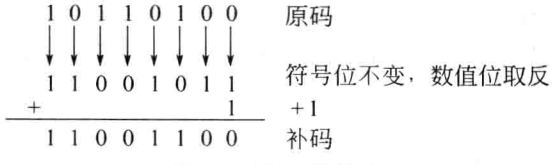

图 1-4　求 X 的补码

2）已知补码，求原码。

【例 1-2】已知某数 X 的补码为 11101110B，试求其原码。

【解】如图 1-5 所示。由[X]补=11101110B，知 X 为负数。求其原码表示时，符号位不变数值部分按位求反后，再在末位加 1。

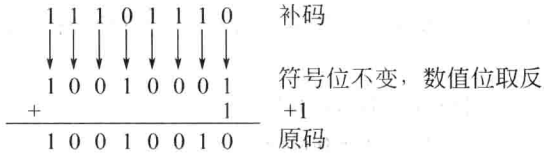

图 1-5 已知补码，求原码

所以[X]原码=10010010B

3）求补

已知[X]补，求[−X]补的过程称为求补。所谓求补，就是将[X]补的所有位（包括符号位）一起逐位取反，然后在末位加 1，即可得到−X 的补码，亦即[−X]补。不管 X 是正数还是负数，都应按此方法操作。

【例 1-3】试求+97、−97 的补码。

【解】+97=1100001，于是[+97]补=01100001B

求[−97]补的方法如图 1-6 所示。

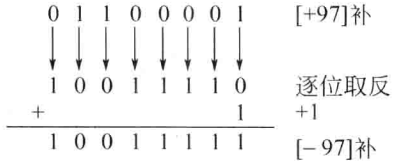

图 1-6 求补

所以[+97]补=01100001B

[−97]补=10011111B

4）已知补码，求对应的十进制数。

【例 1-4】已知某数 X 的补码为 10101011B，试求其对应的十进制数。

【解】由[X]补=10101011B，知 X 为负数，故符号位不变，数值部分按“求反加 1”法得到[X]原：

[X]原=11010101B，X= −1010101B，X= −85

1.2.3 二进制数的运算

1. 二进制数的加减运算

计算机把机器数均当作无符号数进行运算，即符号位也参与运算。运算的结果要根据运算结果的符号及运算有无进位（借位）和溢出等来判别。计算机中设置有这些标志位，标志位的值由运算结果自动设定。

1）无符号数的运算

无符号数实际上是指参加运算的数均为正数，且全部数位都用于表示数值。n 位无符号二进制数能表示的数值范围为 0～(2^n−1)。

两个无符号数相加，由于两个加数均为正数，因此其和也是正数。当和超过其位数所允许表示的数值范围时，就向更高位进位。如图 1-7 所示。

两个无符号数相减，被减数大于或等于减数，无借位，结果为正；被减数小于减数，有借位，结果为负。如图 1-8(a) 所示。

127+150=7FH+96H

```
  0 1 1 1 1 1 1 1
+ 1 0 0 1 0 1 1 0
-----------------
1 0 0 0 1 0 1 0 1 =115H=256+16+5=277
↑
└──── 进位
```

图 1-7 两个无符号数相加

128−10=80H−0AH

```
  1 0 0 0 0 0 0 0                    0 0 0 0 1 0 1 0
- 0 0 0 0 1 0 1 0                  - 1 0 0 0 0 0 0 0
-----------------                  -----------------
  0 1 1 1 0 1 1 0=76H=112+6=118      1 1 0 0 0 1 0 1 0  [补]−01110110=−118
                                     ↑
                                     └──── 借位
        (a)                                (b)
```

图 1-8 两个无符号数相减

反过来相减，即 10−128，运算过程如图 1-8 (b) 所示。

由此可见，对无符号数进行减法运算，其结果的符号用借位标志位来判别：若 CF=0，则无借位，结果一定为正；若 CF=1，则有借位，结果一定为负，对 8 位数值位求补便可求得它的绝对值。

2）符号数的加法运算和溢出判断

（1）补码的加减法运算

由于符号数在引入补码表示法以后，补码的减法运算可以转换为加法运算来进行。其运算法则为：

[x±y]补=[x]补+[±y]补

补码加法运算规则：

① 参加运算的数均为补码；

② 符号位和数值位一起参加运算，符号位向前的进位自然丢掉，若不发生溢出，留下的结果一定是正确的；

③ 运算结果为补码；

④ 若运算结果为负数的补码，且未发生溢出，应转换为原码，才能知道结果的真值。

【例 1-5】用 8 位补码完成下列计算。

（1）63−127　　（2）−67−15

【解】如图 1-9 所示。

```
[−63]补：     0 0 1 1 1 1 1 1
[−127]补：  + 1 0 0 0 0 0 0 1
            -----------------
            [补] 1 1 0 0 0 0 0 0
            [原] 1 1 0 0 0 0 0 0

所以 63−127=−64
```

```
[−67]补：     1 0 1 1 1 1 0 1
[−15]补：   + 1 1 1 1 0 0 0 1
            -----------------
            [1] 1 0 1 0 1 1 1 0  [补]
                1 1 0 1 0 0 1 0  [原]

所以 −67−15=−82
```

图 1-9 【例 1-5】图

（2）补码加减法运算的溢出判断

两个补码加数相加时，在什么情况下才有可能发生溢出呢？很显然，只有当它们都是正

数或当它们都是负数补码相加时才有可能发生溢出。

【例 1-6】两个带符号数（01000001）B（+65）和（01000011）B（+67）相加，如图 1-10 所示。

```
  0 1 0 0 0 0 0 1
+ 0 1 0 0 0 0 1 1
-----------------
  1 0 0 0 0 1 0 0
```

图 1-10 【例 1-6】图

上例中是两个正数相加，但相加结果却是一个负数，显然这个结果是错误的，出现这种情况的原因就在于这两个加数相加结果超过了 8 位二进制带符号数补码所能表示的范围（−128～+127）。

【例 1-7】现在，再来看两个负数（10001000）B（−120）和（11101110）B（−18）的相加情况，如图 1-11 所示。

```
  1 0 0 0 1 0 0 0
+ 1 1 1 0 1 1 1 0
-----------------
[1] 0 1 1 1 0 1 1 0
```

图 1-11 【例 1-7】图

由于规定用 8 位二进制数来表示带符号数，故忽略作为进位位的第九位。按 8 位二进制数来解释这两个符号数的相加，其结果为一个正数，很明显，结果是错误的。错误的原因如上所述。

以上两种情况叫做补码运算的溢出。当两个同符号的数据相加时，如果相加的结果超过了微处理器所能表示的数值范围，就将发生溢出，其结果就是错误的。因此，在微处理机中设有专门的电路用以判断运算结果是否产生溢出，并以某种标志告诉人们这次运算的结果是否存在溢出。只要溢出没有发生，运算的结果总是正确的。这在 8086 微处理机中是用 OF 来标志的。若运算结果发生溢出，则置 OF 为 1；否则 OF 为 0。

在机器中如何判别溢出呢？如何设置 OF 标志位呢？

设符号位向进位位的进位为 CY，数值部分向符号位的进位为 CS，则有无溢出的判别式如下：

$OF=CY \oplus CS$

式中，OF=1 表示有溢出；OF=0 表示无溢出。

【例 1-8】再来看 105+50、−105−50 和−50−5，其运算结果有无溢出，如图 1-12 所示。

2. 二进制数的逻辑运算

1）逻辑非

亦称“求反”。对二进制数进行逻辑非运算，就是按位求它的反。

2）逻辑与

对两个二进制数进行逻辑与运算，就是按位求它们的逻辑积。又称逻辑“乘”，常用记号“∧”或“·”来表示。

3）逻辑加

对两个二进制数进行逻辑加运算，就是按位求它们的“或”，所以逻辑加又称“逻辑或”，常用记号“∨”或“+”来表示。

```
  0 1 1 0 1 0 0 1          1 0 0 1 0 1 1 1
+ 0 0 1 1 0 0 1 0        + 1 1 0 0 1 1 1 0
-----------------        -----------------
  1 0 0 1 1 0 1 1          1 0 1 1 0 0 1 0 1
  CY=0, CS=1               CY=1, CS=0
```

(a) OF=0⊕1=1,发生正溢出，结果错误　　(b) OF=1⊕0=1,发生负溢出，结果错误

```
  1 1 0 0 1 1 1 0
+ 1 1 1 1 1 0 1 1
-----------------
  1 1 1 0 0 0 0 1
  CY=1, CS=1
```

(c) OF=1⊕1=0,无溢出，结果正确

图 1-12 【例 1-8】图

4）逻辑异或

对两个二进制数进行逻辑异或运算，就是按位求它们的模 2 和，所以逻辑异或又称“按位加”，常用“⊕”来表示。

注意：按位加与普通整数加法的区别是它仅按位相加，不产生进位。

1.2.4 二进制编码

计算机中除了处理数值信息外，还要处理大量的字符数据，如字母、符号、汉字、图像、语音等。在计算机中常用的字符编码有 BCD（Binary Coded Decimal）码、ASCII 码、格雷码、奇偶校验码等，这些编码在机器内部都是用二进制表示的，所以统称二进制编码。

1. BCD 码

二进制编码的十进制码（BCD 码）有压缩和非压缩两种存储形式。压缩的 BCD 码是用半个字节存放一位十进制数，即一个字节存放两位十进制数；而非压缩的 BCD 码则以一个字节存放一位十进制数。BCD 码在指令中是常用的一种编码。

所谓 BCD 码就是由四位二进制数表示一位十进制数，十进制数只有 10 个基数符号，而四位二进制数有 16 种取值组合，这样就出现了多种编码方案。下面仅介绍 8421 码、余 3 码和格雷码。这三种编码与十进制数的对应关系如表 1-1 所示。

1）8421 码

8421 码是一种最简单的二进制自然编码，它以 4 位二进制数的前 10 个代码分别对应十进制数的 10 个数码，1010～1111 为无效编码。这种编码按权求和，和就是对应的十进制数。这就是说，编码中的每位仍保留着二进制数所具有的位权，而且 4 位代码从左向右的位权依次为 8、4、2、1。8421 码就是由此而命名的。

如十进制数 72 的压缩 BCD 码用 8421 码可表示为 01110010；十进制数 54 对应的编码为 01010100。

2）余 3 码

余 3 码是无权码，它具有良好的代码校验性。这种编码转换成十进制数后，每个代码的值比相应的十进制数多 3。例如，表 1-1 中的十进制数 4，它对应的二进制代码是 0111=7；十进制数 9 对应的二进制代码是 1100=12；它们都比相应的十进制数大 3。因此，称这种编码为余 3 码。