信息与电子学科百本精品教材工程

上海市教育委员会高等学校
重点课程建设教材

汇编语言程序设计简明教程

杨文显　主编　　宓　双　胡建人　副主编

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

上海市教育委员会高等学校重点课程建设教材

汇编语言程序设计简明教程

杨文显　主　编
宓　双　胡建人　副主编

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书以 80x86 指令系统和 Borland 公司 TASM5.x 软件为主要背景，系统介绍了汇编语言程序设计的基本概念和方法。内容包括：汇编语言程序设计基础、存储器数据定义与传送、数据运算与输入/输出、选择与循环、子程序、字符串与文件处理、显示程序设计、输入/输出与中断。

作者在长期的教学和科研实践中，以亲身积累的教学经验为基础，借鉴了许多国外的优秀教材，探索出“以程序设计为中心”，而不是“以语言为中心”展开本课程教学的方法，取得了显著的成效。读者学完前两章，就可以编写完整的汇编语言程序。此后，学习新知识的过程，就是不断地进行程序设计训练的过程，在多次“螺旋式”上升的过程中，牢牢地掌握汇编语言程序设计的基本方法。

本书是为计算机及相关专业本、专科的“汇编语言程序设计”课程而编写的，它也特别适合作为计算机工作者学习汇编语言程序设计的自学教材。

图书在版编目（CIP）数据

汇编语言程序设计简明教程 / 杨文显主编. —北京：电子工业出版社，2005.7
上海市教育委员会高等学校重点课程建设教材
ISBN 7-121-01251-0

I. 汇… II. 杨… III. 汇编语言－程序设计－高等学校－教材 IV. TP313

中国版本图书馆 CIP 数据核字（2005）第 049294 号

责任编辑：冉 哲
特约编辑：李 岩
印 刷：
装 订：北京牛山世兴印刷厂
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1092 1/16 印张：17.25 字数：438 千字
印 次：2007 年 2 月第 2 次印刷
印 数：3000 册 定价：23.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系电话：(010)68279077；邮购电话：(010)88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：(010)88258888。

关于汇编语言教与学的几点思考（代前言）

计算机程序设计语言课程的教学，历来存在着两种不同的方法。

方法之一是以程序设计语言为主线，从这种语言的“字、词、句、章”出发，系统地理解该语言的语法、语义规范，在这个基础上，展开程序设计的教学。在计算机应用的早期，计算机应用以科学计算为主，学透了一门语言就可以解决各种问题，走上这条路可以说是“顺理成章”的。但是，随着新的计算机程序设计语言的不断推出，这种方法逐渐被淘汰。

另一种方法则是以程序设计为主线，围绕着“怎样编写程序”这个中心展开。根据程序设计由易到难的次序，选择对应的语言元素进行教学，不再强调语言本身的完整性。这种方法可以较快地进入程序设计的主题，目前大多数程序设计语言的教学都采取这种模式。

但是，到目前为止，国内大多数汇编语言教材内容的组织可以归属为上面所叙述的第一种类型。产生这种局面的原因大致有两个方面：

首先是因为汇编语言是一种面向机器的低级语言。较之其他语言，它的语言元素“粒度”小，一个最简单的有意义的程序，也需要 20 行左右的代码，涉及到十余种符号指令和伪指令，各知识点之间的关联度较高。想“绕过”众多的语言成分，直接进入程序设计主题，教学组织的难度较大。

另一个原因则是早期的处理器相对简单，指令总量有限，相对完整地介绍汇编语言的语言元素还是能够做到的。

但是，现在的情况发生了许多的变化。

首先，我国的高等教育得到了快速的发展，在校学生总量成倍地增加，高等教育从“精英教育”向“大众教育”变迁。加上汇编语言自身的一些特点，使得以语言为中心展开教学的难度越来越大，学生普遍感到汇编语言程序设计难学，难掌握。

此外，由于 80x86 微处理器事实上的市场主导地位，目前汇编语言教学均围绕着该系列处理器进行。但是，不幸的是，80x86 处理器属于复杂指令系统计算机（CISC, Complex Instruction System Computer）类型，为了获取市场利益采取的“向下兼容”策略使它的指令系统越来越庞大，完整地介绍它的指令系统几乎已经没有可能。这对采用以语言为主线的教学模式提出了极大的挑战。

由于上述原因，沿袭多年来传统方法的教学变得越来越困难，汇编语言的教学改革势在必行。

目前国内的汇编语言教材，都是先学习寻址方式和指令系统，然后讲解汇编语言程序的语法结构和伪指令，此后才开始程序设计的教学，这种方法必然地造成了知识和应用的严重脱节。可以设想一下，对于一名初学者，要从几百条指令和几十条伪指令中挑选出适当的指令来编写程序，绝不是一件轻松的事情，因此感到茫然和不知所措就是不可避免的。

综上所述，在汇编语言教学中采用以程序设计为主线的教学模式是该课程教学改革的主要方向。在这方面，国外的一些教材已经有了许多成功的探索，如国内多次翻译和出版的《IBM PC Assembly Language and Programming》。

本教材是作者在多年进行教学研究和科研实践的基础上，广泛吸纳国内、外优秀教材的成功经验，历时两年，精心编写而成的。它的主要特点是：

- 明确了本课程的教学目标。把汇编语言程序设计作为硬件、软件知识的交汇点。通过本课程学习，建立完整的计算机工作模型，体验没有操作系统支持的、直接面向硬件的程序设计，进行程序设计能力的培养，而不是仅仅让学生多掌握一门软件开发工具。着重程序设计能力的培养，不强调语言体系的完整性。
- 按照程序设计逐步深入的过程，精心地分割汇编语言的元素和知识点，围绕着程序设计的需要学习相应的知识点，做到“学一点、用一点、巩固一点”。学习指令和相关知识的同时，就是不断进行程序设计训练的过程。课程的教学过程，呈现出知识结构的螺旋式上升形态，符合循序渐进的学习规律。
- 精选实例，通过典型例题使学生充分理解汇编语言程序设计的特点，把小粒度的众多知识点融化在应用实例中，避免喋喋不休式的注意事项。
- 精选 80x86 的核心指令集。计算机的出现尚不到 60 年，如果把立足点还停留在 26 年前出现、至今踪迹难觅的 16 位微处理器上，实在是有点说不通了。但是，出于本书的基本思想，没有对庞大的 80x86 指令系统做全面的介绍，而是选择了两类 32 位指令：由原 8086 的 16 位指令自然扩展得到的 32 位指令，对原 16 位指令功能进行改进和提升的 32 位指令。我们将它们称做“80x86 核心指令集”，这些指令完全能够满足培养在校学生汇编语言程序设计能力的需要。这样做，一方面不再把自己关在 32 位 CPU 的门外，同时也不要分散精力，把时间过多地消耗在讲解、记忆相对不太常用的指令上面。
- 提供了简单而又实用的输入输出库函数（YLIB.LIB），降低使用汇编语言进行程序开发的门槛，同时也避免了使用宏给初学者带来的程序调试困难。需要的读者可以通过 E-mial 与作者联系（xhywx@163.net 或 xhywxywx@163.com）。

本书由杨文显主编，胡建人、宓双副主编（排名不分先后）。杨文显在长期的本课程教学实践中，逐步摸索出一套以程序设计为主线的汇编语言教学体系，拟订了本书的大纲，编写了第 1～4 章，整理了大部分的附录。胡建人参加了大纲的讨论，编写了第 6，8 章，参加了部分附录的整理工作。宓双参加了大纲的讨论，编写了第 5，7 章。附录 C 由杨晶鑫提供。本书初稿完成后，主编对全书进行了多遍认真的统稿和修订。

总之，这是一本全新结构、全新思路的新教材。作者多年来按照这一思路组织教学，取得了比较理想的效果，希望也会对使用本教材的教师和学生有所裨益。

虽然本书的教学体系经过多年的教学实践已经证明是行之有效的，但是作为一本崭新体系的教材，难免存在着疏漏和缺陷，敬请使用本书的教师、读者不吝赐教，作者将不胜感激。

使用本书的教师如果需要本书习题参考答案和例题源程序可以与作者联系，或登录华信教育资源网（www.hxedu.com.cn）免费获取。

作者

2005 年 5 月

目　录

第 1 章　汇编语言基础

汇编语言是一种面向机器的“低级”语言。它和机器语言一样，是电子数字计算机的“母语”。要真正理解计算机的工作过程，理解计算机程序的执行过程，就必须学习汇编语言。学习汇编语言程序设计是走向优秀程序员的重要一步。

由于汇编语言是面向机器的语言，所以，使用汇编语言进行程序设计，就必须对处理的对象（各种数据）、处理的工具（计算机）有所了解。可以从几种不同的角度来描述计算机的结构，作为一名汇编语言程序员，必须了解的是计算机的“逻辑结构”。

本章介绍学习汇编语言必备的基础知识，包括计算机内数据的表示和计算机的逻辑结构。

1.1　计算机内数据的表示

现代计算机可以处理各种各样的信息：数值数据、文字、声音、图形等。这些信息在计算机内都是用一组二进制代码来表示的，统称为“数据”。这里首先介绍数值数据和文字在计算机内的表示方法。

1.1.1　进位计数制

进位计数制是现代人类普遍使用的表示数的方法。进位计数制具有三个基本特征：

① 有限个数字符号：0, 1, 2, …, R–1，“R”称为该进位计数制的“基数”；

② 数值达到“R”，不能用一位数字表示时，向左进位，称为“逢 R 进 1”；

③“权展开式”，不同位置上的数字具有不同的“权”：从小数点出发，向左各位数字的“权”分别是 $R^0, R^1, R^2, R^3, \cdots$；从小数点出发，向右各位数字的“权”分别是 $R^{-1}, R^{-2}, R^{-3}, \cdots$。

于是，一个“R 进制数” $D=d_{n-1}d_{n-2}d_{n-3}\cdots d_2d_1d_0.d_{-1}d_{-2}\cdots d_{-m}$

$$=d_{n-1}\times R^{n-1}+d_{n-2}\times R^{n-2}+\cdots+d_1\times R^1+d_0\times R^0+d_{-1}\times R^{-1}+d_{-2}\times R^{-2}+\cdots+d_{-m}\times R^{-m}$$

1．十进制计数法

十进制计数法是进位计数制的一种，它的基数为“十”。按照进位计数制的基本特征，它具有以下三个基本特征：

① 有十个基本数字符号：0, 1, 2, …, 9；

② 采用“逢十进一”规则；

③ 从小数点出发，向左各位数字的“权”分别是 $10^0, 10^1, 10^2, 10^3, \cdots$；从小数点出发，向右各位数字的“权”分别是 $10^{-1}, 10^{-2}, 10^{-3}, \cdots$。

例如，十进制数 $323.31=3\times10^2+2\times10^1+3\times10^0+3\times10^{-1}+1\times10^{-2}$。

在汇编语言中，十进制数用它原来的形式表示，如 123, –36 等；也可以在数值后面加上字母“D”或“d”，把十进制数和其他进制的数区分得更明显一些，如 123D, –36d 等。

2．二进制计数法

电子数字计算机内部用“开关电路”来表示数据，这些电路只有两种状态，分别表示“1”和“0”。由于这个原因，计算机内部采用“二进制”表示数据。和十进制数一样，二进制数也有三个基本特征：

① 两个数字符号：0, 1；

②“逢二进一”：用“进位”的方法表示大于 1 的数；

③“权展开式”：从小数点出发，向左各位数字的“权”分别是 $2^0, 2^1, 2^2, 2^3, \cdots$。从小数点出发，向右各位数字的“权”分别是 $2^{-1}, 2^{-2}, 2^{-3}, \cdots$。

从二进制数的“权展开式”出发，很容易把一个二进制数转换成和它大小相同的十进制数。例如：

$$(11011.101)_2 = 1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3} = 27.625$$

把一个十进制数转换成大小相同的二进制数稍微复杂一些，它需要把十进制数的整数部分和小数部分用不同的方法分别处理。整数部分不断地被 2 除，得到的各位余数顺序排列就是它对应的二进制数。小数部分不断地乘以 2，保留每次乘法得到的整数，不再参加下一次的乘法。把保留的整数数字顺序排列，就是小数部分对应的十进制数值。小数部分转换的结果可能是“无限循环小数”，可以根据需要进行取舍。图 1.1 展示了十进制数 215.35 向二进制数的转换过程，经转换，$(215.35)_{10}=(11010111.01011)_2$。

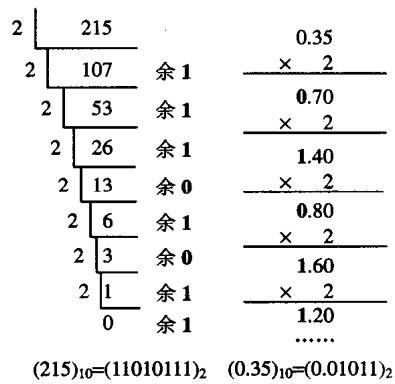

图 1.1　十进制数向二进制数的转换

在汇编语言中使用二进制数，需要在它的后面加上字母“B”，以与十进制数加以区别，如 10011010B。

3．八进制和十六进制计数法

用二进制表示的数据位数多，不便于书写和阅读。借助于等式 $8=2^3$，$16=2^4$，常常用八进制数和十六进制数来表示二进制数。

八进制数使用我们熟悉的 8 个数字符号：0, 1, 2, 3, 4, 5, 6, 7。

十六进制数除了使用 0～9 这十个数字符号，还借用了字母 A～F 来表示大于 9 的 6 个数字符号，它们分别等于十进制数的 10, 11, 12, 13, 14, 15。

1 位八进制数可以方便地转换成 3 位二进制数，反之亦然。1 位十六进制数则和 4 位二进制数相对应。这样，二进制数和八进制数、十六进制数之间可以方便地相互转换：

$$(1101100.0101)_2=(1'101'100.010'1)_2=(001'101'100.010'100)_2=(154.24)_8$$

$$(1101100.0101)_2=(110'1100.0101')_2=(0110'1100.0101')_2=(6C.5)_{16}$$

在汇编语言中使用十六进制数，需要在它的后面加上字母“H”。如果一个十六进制数用数字符号“A～F”开始，为了和变量名等名字区分，需要在这个数的前面补加一个数字“0”，这个“0”不应该看做该数有效数字的一部分。例如，0E6H 代表一个 8 位二进制代码表示的无符号数“1110 0110B”，而不是 12 位二进制数。

许多汇编语言版本不支持使用八进制数。

十进制数和八进制数、十六进制数之间的转换可以通过二进制数间接进行，也可以采用对十进制数的整数部分“除 8/16 取余”，对小数部分“乘 8/16 保留整数”的方法进行。

1.1.2 数据组织

计算机内的信息按一定的规则组织存放。

1．位（bit）

位（bit）是信息的最小表示单位，用小写字母“b”表示。1 位二进制数可以表示一个开关的状态（称为**开关量**），例如，用“1”表示“接通”，“0”表示“断开”。

大多数的数据无法用 1 位二进制数表示，不能从计算机内单独取出 1b 的信息进行处理。

2．字节（Byte）

字节（Byte）是计算机内信息读/写、处理的基本单位，由 8 位二进制数组成，用大写字母“B”表示。1 个字节可以表示 256（2^8）个不同的值，可以用来存放一个范围较小的整数，一个西文字符，或者 8 个开关量。

一个字节内的 8 个“位”从右（低位）向左（高位）从 0 开始编号，依次为 $b_0, b_1, \cdots, b_7$，如图 1.2（a）所示。其中，b_0 称为最低有效位（LSB, Least Significant Bit），b_7 称为最高有效位（MSB, Most Significant Bit）。

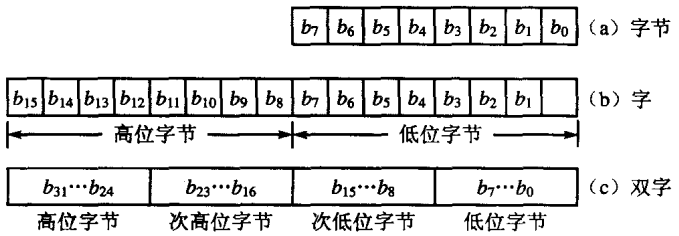

图 1.2 数据组织

3．字（Word）和双字（Double Word）

字（Word）和**双字**（Double Word）分别由 16/32 位二进制数组成，或者说，由 2/4 个字节组成。

1 个字由 16 位二进制数（2 个字节）组成，可以存放一个范围较大的整数或一个汉字的编码。它的 16 个二进制位仍然从右（低位）向左（高位）从 0 开始编号，依次为 $b_0, b_1, \cdots, b_{15}$，如图 1.2（b）所示。其中，b_0～b_7 称为“低位字节”，b_8～b_{15} 称为“高位字节”。

1 个“双字”由 32 位二进制数组成（4 个字节），可以存放范围更大的整数或者用“浮点格式”表示的单精度实数。它的 32 个二进制位中，b_0～b_7, b_8～b_{15}, b_{16}～b_{23}, b_{24}～b_{31} 分别称为“低位字节”、“次低位字节”、“次高位字节”、“高位字节”，如图 1.2（c）所示。

1.1.3　无符号数的表示

所谓**无符号数**是正数和零的集合。储存一个正数或零时，所有的“位”都用来存放这个数的各位数字，无须考虑它的符号，“无符号数”因此得名。

可以用字节、字、双字或者更多的字节来存储和表示一个无符号数。

用 N 位二进制数表示一个无符号数时，最小的数是 0，最大的数是 2^N-1。一个字节、字、双字无符号数的表示范围分别是 0～255, 0～65 535, 0～4 294 967 295。

一个无符号数需要增加它的位数时，只需要在它的左侧添加若干个“0”，称为**零扩展**。例如，用一个字来存储 8 位无符号数 1011 0011 时，低位字节置入这个无符号数，高位字节填“0”，结果为 0000 0000 1011 0011（注：插入空格是为了阅读和区分，书写时没有这个要求）。

两个 N 位无符号数相加时，如果最高位产生了“进位”，表示它们的和已经超出了 N 位二进制数所能够表示的范围，需要向更高位“进位”。同样，两个 N 位无符号数相减时，如果最高位产生了“借位”，表示当前数据“不够减”，需要向更高位“借位”。

计算机内用**进位标志**（CF, Carry out Flag）表示两个无符号数运算结果的特征。如果 CF = 1，表示它们的加法有**进位**，或者它们的减法有**借位**；CF = 0，则表示没有产生进位或借位。

1.1.4　有符号数的表示

可以用字节、字、双字或者更多的字节来存储和表示一个**有符号数**。

表示一个有符号数有多种不同的方法，如原码、反码、补码表示法。同一个有符号数在不同的表示法中可能有不同的形式。

1．原码

表示有符号数最简单的方法是采用**原码**。用原码表述一个有符号数时，最左边一位二进制数表示这个数的符号，“0”代表正，“1”代表负，后面是它的“有效数字”。例如，用字节存储一个有符号数时，$[+3]_{原}$ = 0 000 0011，$[-3]_{原}$= 1 000 0011（注：在二进制代码中插入空格是为了阅读方便，原码本身没有这个规定）。为了和计算机内的数据组织相协调，通常用 8 位、16 位或者 32 位二进制数表示一个数的原码。

用一个字节存储有符号数原码时，可以表示 127 个正数（1～127），127 个负数（–1～–127）和两个“0”（正“0”：0 000 0000，负“0”：1 000 0000）。

原码的表示规则简单，但是运算规则比较复杂，不利于计算机高速运算的实现。

2．反码

反码仍然用最高位“0”表示符号为正，“1”表示符号为负。

符号位之后的其他二进制位用来存储这个数的有效数字。正数的有效数字不变，负数的有效数字取反。例如，用字节存储一个有符号数时，$[+11011]_{反}$ = [+001 1011]$_{反}$ = 0 001 1011，$[-11011]_{反}$ = [–001 1011]$_{反}$ = 1 110 0100。

对于正数 $X = d_{n-2}d_{n-3}\cdots d_2d_1d_0$ 来说，$[X]_{反} = X = 0\ d_{n-2}d_{n-3}\cdots d_2d_1d_0$。

对于一位二进制数而言，$\overline{b}=1-b$。所以，对于负数 $Y=-d_{n-2}d_{n-3}\cdots d_2d_1d_0$，$[Y]_{反}=1\overline{d_{n-2}}\,\overline{d_{n-3}}\cdots\overline{d_2}\,\overline{d_1}\,\overline{d_0}=1111\cdots111-|Y|=2^n-1-|Y|=2^n-1+Y$。

用一个字节存储有符号数反码时，可以表示 127 个正数（1～127）、127 个负数（−1～−127）和两个“0”（正“0”：0 000 0000，负“0”：1 111 1111）。

反码的运算规则仍然比较复杂，可以用做原码和常用的补码之间的一个过渡。

3. 补码

补码表示法仍然用最高有效位（MSB）表示一个有符号数的符号，“1”表示符号为负，“0”表示符号为正。

符号位之后的其他二进制位用来存储这个数的有效数字。正数的有效数字不变，负数的有效数字取反后最低位加 1。用字节存储一个有符号数时，$[+11011]_{补}=[+001\ 1011]_{补}=0\ 001\ 1011$，$[-11011]_{补}=[-001\ 1011]_{补}=1\ 110\ 0100+1=1\ 110\ 0101$。

对于正数 $X=d_{n-2}d_{n-3}\cdots d_2d_1d_0$ 来说，$[X]_{补}=X=0\,d_{n-2}d_{n-3}\cdots d_2d_1d_0$。

对于负数 $Y=-d_{n-2}d_{n-3}\cdots d_2d_1d_0$，$[Y]_{补}=1\overline{d_{n-2}}\,\overline{d_{n-3}}\cdots\overline{d_2}\,\overline{d_1}\,\overline{d_0}+1=1111\cdots111-|Y|+1=2^n-|Y|=2^n+Y$。表 1.1 列出了用 8 位二进制代码表示的部分数值的原码、反码和补码。

表 1.1　部分数值的原码、反码和补码

真值（十进制）	二进制表示	原　码	反　码	补　码
+127	+111 1111	0 111 1111	0 111 1111	0 111 1111
+1	+000 0001	0 000 0001	0 000 0001	0 000 0001
+0	+000 0000	0 000 0000	0 000 0000	0 000 0000
−0	−000 0000	1 000 0000	1 111 1111	0 000 0000
−1	−000 0001	1 000 0001	1 111 1110	1 111 1111
−2	−000 0010	1 000 0010	1 111 1101	1 111 1110
−127	−111 1111	1 111 1111	1 000 0000	1 000 0001
−128	−1000 0000	无	无	1 000 0000

用一个字节存储有符号数补码时，可以表示 127 个正数（1～127）、128 个负数（−1～−128）和一个“0”（0000 0000）。其中，$[-1]_{补}=1\ 111\ 1111$，$[-128]_{补}=1\ 000\ 0000$。

如果把一个数补码的所有位（包括符号位）取反加 1，将得到这个数相反数的补码。把“取反加 1”这个操作称为**求补**，$[[X]_{补}]_{求补}=[-X]_{补}$。例如，

$[5]_{补}=0\ 000\ 0101$，$[[5]_{补}]_{求补}=[0000\ 0101]_{求补}=1\ 111\ 1011=[-5]_{补}$

已知一个负数的补码，求这个数自身时，可以先求出这个数相反数的补码。例如，已知 $[X]_{补}=1\ 010\ 1110$，求 X 的值（**真值**）可以遵循以下步骤：

① $[-X]_{补}=[[X]_{补}]_{求补}=[1\ 010\ 1110]_{求补}=0\ 101\ 0001+1=0\ 101\ 0010$。

② 于是，$-X=[+101\ 0010]_2=+82D$。

③ 于是，$X=-82$。

4. 补码的扩展

一个补码表示的有符号数需要增加它的位数时，对于正数，需要在它的左侧添加若干个“0”；对于负数，则需要在它的左侧添加若干个“1”。上述操作实质上是用它的符号位

来填充增加的高位（无论该数是正是负），称为**符号扩展**。例如，$[-5]_{补}$ = 1 111 1011（8 位）= 1 111 1111 1111 1011（16 位），$[+5]_{补}$ = 0 000 0101（8 位）= 0 000 0000 0000 0101（16 位）。

5. 补码的运算

补码的运算遵循以下规则：

$[X+Y]_{补}=[X]_{补}+[Y]_{补}$

$[X-Y]_{补}=[X]_{补}-[Y]_{补}$，或者，$[X-Y]_{补}=[X]_{补}+[-Y]_{补}=[X]_{补}+[[Y]_{补}]_{求补}$

例如，$[15+23]_{补}=[15]_{补}+[23]_{补}$= 0 000 1111+0 001 0 111 = 0 010 0 110 = $[38]_{补}$

$[15-23]_{补}=[15]_{补}-[23]_{补}$= 0 000 1111−0 001 0 111 = 1 111 1 000 = $[-8]_{补}$（舍去借位）

或者，$[15-23]_{补}=[15]_{补}+[-23]_{补}$= 0 000 1111+$[0\ 001\ 0\ 111]_{求补}$

= 0 000 1111+1 110 1 001=1 111 1 000 = $[-8]_{补}$

进行补码加法时，最高位如果有进位/借位，将其自然抛弃，不会影响结果的正确性。例如，$[(-3)+(-5)]_{补}=[-3]_{补}+[-5]_{补}$= 1 111 1101+ 1 111 1011 = 1…1111 1000 =1 111 1000 = $[-8]_{补}$。

计算机内用**溢出标志**（OF, Overflow Flag）表示两个有符号数运算结果的特征。如果补码表示的有符号数的运算结果超过了该长度数据的表示范围，称为**溢出**，OF=1；反之，OF=0，则没有产生溢出。

计算机自身用“双进位法”判断是否产生溢出：一次运算中，如果补码最左边 2 个位上的进位相等，表示没有溢出；反之则表示有溢出发生。

例如，$[(-103)+(-105)]_{补}$=1 001 1001+1 001 0111=1…0 011 0000。

上面运算中，b_7 位上的进位 c_7=1，b_6 位上的进位 c_6=0，$c_7 \neq c_6$，运算产生了溢出。

程序员可以用下面更简便的方法来进行判断：

① 两个同号数相加，结果符号位与原来相反，则产生了溢出。异号数相加不会产生溢出；

② 两个异号数相减，结果符号位与被减数符号位不同，则产生了溢出。同号数相减不会产生溢出。

上面运算中，两个负数相加，它们的和却为正数，可以断定运算产生了溢出。

从上面叙述可以看出，补码的运算规则具有突出的优点：

① 同号数相加和异号数相加使用相同的规则；

② 有符号数加法和无符号数加法使用相同的规则；

③ 减法可以用加法实现（对于电子计算机内的开关电路，求补是十分容易实现的）。

上述特性可以用来简化运算器电路，简化指令系统（如果有符号数和无符号数的运算规则不同，则两者运算需要使用不同的指令，用不同的电路来实现）。由于这个原因，计算机内的有符号数一般都用补码表示，除非特别说明。

1.1.5 字符编码

计算机处理的对象除了数值数据之外，还有大量的文字信息。文字信息以“字符”为基本单元，每个字符用若干位二进制表示。

计算机内常用的字符编码是 ASCII（American Standard Code for Information Interchange，美国信息交换标准编码）。它规定用 7 位二进制数表示一个字母、数字或符号，包含 128 个不同的编码。由于计算机用 8 位二进制数组成的字节作为基本存储单位，一个字符的 ASCII 码一般占用一个字节，低 7 位是它的 ASCII 码，最高位置“0”，或者用做“校验位”。

ASCII 编码的前 32 个（编码 00H～1FH）用来表示控制字符，例如 CR（“回车”，编码 0DH），LF（“换行”，编码 0AH）。

ASCII 编码 30H～39H 用来表示数字字符 0～9。它们的高 3 位为 011，低 4 位就是这个数字字符对应的二进制数表示。例如，'5' = 011 0000B + 0101B= 35H。

ASCII 编码 41H～5AH 用来表示大写字母 A～Z。它们的高 2 位为 10。

ASCII 编码 61H～7AH 用来表示小写字母 a～z。它们的高 2 位为 11。小写字母的编码比对应的大写字母大 20H。例如：'A'=41H，'a'=61H，'a'–'A'='b'–'B'= …= 20H。

1.1.6 BCD 码

十进制小数和二进制小数相互转换时可能产生误差，这对于某些应用会带来不便。计算机内部允许用一组四位二进制数来表述一位十进制数，组间仍然按照“逢十进一”的规则进行。这种用二进制数表示的十进制数编码称为 **BCD**（Binary Coded Decimal）**码**。

有两种不同的 BCD 码。

压缩的 BCD 码用一个字节存储 2 位十进制数，高 4 位二进制数表示高位十进制数，低 4 位二进制数表示低位十进制数。例如，$[25]_{压缩BCD}$=0010 0101B。需要使用压缩 BCD 数时，可以用相同数字的十六进制数表述。例如，用 25H 表示十进制数 25 的压缩 BCD 码。

非压缩的 BCD 码用一个字节存储 1 位十进制数，低 4 位二进制数表示该位十进制数，对高 4 位的内容不做规定。例如，数字字符 '7' 的 ASCII 码 37H 就是数 7 的非压缩 BCD 码。

从上面的叙述可以看出，计算机内的一组二进制数编码和它们的原型之间存在着“一对多”的关系。有符号数+65 的补码、无符号数 65、大写字母 'A' 的 ASCII 码在计算机内的表示都是 41H，它甚至还可以是十进制数 41D 的压缩 BCD 数编码。所以，面对计算机内的一组二进制数编码，可能无法准确地知道它究竟代表什么。知道它“真面目”的应该是这组二进制数信息的“主人”——汇编语言程序员。

1.2 计算机组织

使用汇编语言进行程序设计时，除了需要考虑求解问题的过程或者算法，安排数据在计算机内的存储格式，同时还要根据程序/算法的要求对计算机内的资源进行调度和分配。因此，作为一名汇编语言程序员，必须了解计算机的基本结构，了解有哪些可供使用的资源，以及不同资源在使用上的区别。但是，他无须精确地了解上述资源的电子线路，以及它们工作时的电气特性。不同的计算机具有不同的结构，本节主要结合 80x86 系列微型计算机来介绍程序员需要掌握的计算机逻辑结构。

1.2.1 计算机组成

迄今为止，电子计算机的基本结构仍然属于冯·诺依曼体系结构。这种结构的特点可以概要归结如下。

① 存储程序原理。把程序事先存储在计算机内部，计算机通过执行程序实现高速数据处理。

② 五大功能模块。电子计算机由运算器、控制器、存储器、输入设备、输出设备这些功能模块组成。