21世纪高等学校计算机规划教材

21st Century University Planned Textbooks of Computer Science

# 80x86汇编语言程序设计（第2版）

## 80x86 Assembly Language Program (2nd Edition)

王成耀 编著

- 以Intel 80x86 CPU指令系统为背景
- 强调“为什么”、“是什么”和“怎么做”
- 概念讲解透彻，语言精练流畅
- 丰富实例都由编者精心设计挑选并亲手上机验证

精品系列

人民邮电出版社
POSTS & TELECOM PRESS

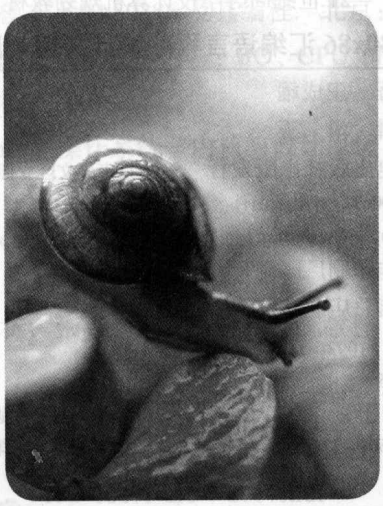
21世纪高等学校计算机规划教材

21st Century University Planned Textbooks of Computer Science

# 80x86汇编语言程序设计（第2版）

## 80x86 Assembly Language Program (2nd Edition)

王成耀 编著



精品系列

人民邮电出版社

北京

**图书在版编目（CIP）数据**

80x86 汇编语言程序设计 / 王成耀编著. —2 版. —北京：人民邮电出版社，2008.4
21 世纪高等学校计算机规划教材
ISBN 978-7-115-17501-4

Ⅰ. 8… Ⅱ. 王… 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（2008）第 005272 号

## 内 容 提 要

本书以满足“汇编语言程序设计”课程的教学为目标，以 Intel 80x86 CPU 指令系统与 Microsoft 宏汇编 MASM 6.1X 为背景，系统介绍了汇编语言程序设计的基本理论和方法。主要内容包括：汇编语言程序设计的基础知识、实模式下的 80x86 指令、常用伪指令、源程序格式、程序设计的基本技术、多模块程序设计、输入输出和中断程序设计等。此外，简要介绍了 32 位保护模式以及 Win32 汇编语言程序设计的基本方法；以 Microsoft Visual C++ 6.0 为背景，介绍了 Windows 9X/XP/2000/2003 等 32 位环境下汇编语言与 C/C++语言的混合编程以及并发程序设计的基本方法。

本书可作为高等院校计算机及相关专业本科生的教材，也可作为相关人员学习汇编语言的自学参考书。

21 世纪高等学校计算机规划教材

**80x86 汇编语言程序设计（第 2 版）**

◆ 编　　著　王成耀
责任编辑　滑　玉
◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鸿佳印刷厂印刷
新华书店总店北京发行所经销
◆ 开本：787×1092　1/16
印张：18.5
字数：444 千字　　2008 年 4 月第 2 版
印数：18 501－22 500 册　　2008 年 4 月北京第 1 次印刷

ISBN 978-7-115-17501-4/TP

定价：29.80 元

读者服务热线：(010)67170985　印装质量热线：(010)67129223
反盗版热线：(010)67171154

# 第 2 版前言

本书是编者在第 1 版的基础上，结合多年计算机专业教学与软件开发经验修订而成的。本书的修订主要基于如下考虑。

首先，作为一本计算机及相关专业本科生学习汇编语言的教材，不一定面面俱到。在涵盖基本内容的基础上，应突出重点、讲透难点、注重实用，特别是要强调“为什么”、“是什么”和“怎么做”。这也是本书第 1 版编写时的思路。因此，本书第 2 版仍保持第 1 版的总体风格，调整了部分内容的表述方式和次序，修订了部分例题，力求使概念和方法的阐述更加清楚和严密。

其次，随着计算机软件技术的发展和各种出色的高级语言开发工具的不断涌现，目前完全用汇编语言实现的软件系统已极为罕见；但同时，对于某些直接访问硬件或对性能要求较高的代码，高级语言又难以实现，至少目前高级语言还不能完全取代汇编语言。汇编语言与高级语言相结合广泛用于高性能软件特别是系统软件的开发。因此，第 2 版增加了汇编语言与 C/C++语言混合编程的内容，以便使读者了解汇编语言与 C/C++语言混合编程的基本方法，加深对 C/C++语言函数调用和参数传递实现机制的理解，提高程序设计水平。

最后，现代计算机系统广泛采用了支持多任务的环境，应用软件也日趋复杂，其中涉及资源的共享与竞争、多个任务之间的通信与同步等问题，所有这些问题的基础是并发。并发是程序员在开发复杂多任务应用时必然要面临的问题，而解决并发进程（或线程）的同步往往离不开汇编语言。因此，第 2 版增加了并发程序设计的内容，以便使读者了解设计并发程序必须考虑的问题以及并发环境下实现进程（或线程）同步的基本方法。

全书共分 10 章。第 1 ~ 8 章作为汇编语言的基本内容，以 Intel 80x86 CPU 指令系统为背景，基于 Microsoft 宏汇编 MASM 6.1X，重点讲述实模式下 80x86 指令系统与汇编语言程序设计的基本技术以及多模块程序设计与中断程序设计的基本方法。第 9 章简要介绍 32 位保护模式以及 Win32 汇编语言程序设计的基本方法。第 10 章介绍汇编语言程序设计的两个高级主题：汇编语言与 C/C++语言的混合编程以及并发程序设计。

本书含有丰富的实例，所有实例都是编者精心设计挑选并亲手上机验证的。每章后均有习题，并在配套的教辅资料中给出了所有习题的参考答案，以便读者复习和检查学习效果。读者只要有数制方面的基本知识，就可以通过学习本书的第 1 ~ 8 章，系统掌握 80x86 汇编语言程序设计的基本技术。如果具备高级语言（如 C 语言等）程序设计的基础，将有助于对部分内容的深入理解。

本教材计划讲授 50 学时左右，其中，带*的章节可选修，教师可根据自己的教学计划作相应取舍。

希望本书能对学习汇编语言的读者提供较大帮助。欢迎读者对书中的错误与不妥之处提出批评和改进意见。

编者

# 目　录

# 第1章 基础知识

本章介绍学习汇编语言程序设计所必须具备的基本知识，主要包括汇编语言的基本概念及计算机中数据的表示方法。通过本章的学习，读者应了解什么是汇编语言、汇编语言的特点和意义、数据的组织（字节、字和双字）、带符号数的二进制补码表示、BCD 以及基本位操作等。尤其要深刻理解：对于一个二进制数，其具体含义依赖于使用者的解释。

## 1.1 认识汇编语言

自然语言是具有特定语音和语法等规范的、用于人类表达思想并实现相互交流的工具。人与人之间只有使用同一种语言才能进行直接交流，否则就必须通过翻译。要使计算机为人类服务，人们就必须借助某种工具，告诉计算机“做什么”甚至“怎么做”，这种工具就是程序设计语言。

程序设计语言通常分为 3 类：机器语言（Machine Language）、汇编语言（Assembly Language）和高级语言（High Level Language）。其中，前两种语言是与机器密切相关的，统称为低级语言。

### 1.1.1 机器语言

计算机能直接识别并进行处理的是由 0、1 组成的二进制代码。因为构成计算机硬件本身的各个部件是基于二值逻辑的，这些部件只能识别 0 和 1 两个状态，其功能就是记忆、传输和加工二进制信息 0 或 1。计算机的工作就是传输和处理二进制信息的过程。

1. 机器指令

机器指令是指用二进制编码的指令，以指示计算机所要进行的操作及操作对象（数据或数据地址）。每条机器指令控制计算机完成一个操作。机器指令由指令译码器识别，并经过一定的时钟周期付诸实现，从而完成指令所规定的操作。

机器指令一般由操作码（Opcode）和操作数（Operand）构成。操作码指出指令所要执行的操作，如加、减、乘、除和传送等。操作数指出操作的数据对象。

2. 指令系统与机器语言

指令系统（Instruction Set）是指特定计算机上机器指令的集合。机器语言是由指令系统以及机器指令的使用规则构成的。

机器语言是计算机唯一能够识别的语言，只有用机器语言描述的程序，计算机才能直接执行。下面是用 Intel 8086 CPU 机器语言编写的一段代码（十六进制）：

```
B024
B344
F6E3
050A00
```

区区几行代码，若没有注解，很少有人能直接看出它的功能就是计算表达式 36 × 68+10 的值。即使对 Intel 8086 机器语言非常了解，也许还得借助于手册。然而，在计算机诞生的初期，人们也就是这样设计程序的。

3. 机器语言的主要特点

机器语言主要具有下列两个特点。

（1）机器语言与机器密切相关

机器语言与计算机的 CPU、内存管理机制和 I/O 机制有着十分密切的关系。通常，不同型号 CPU 的指令系统往往有较大差异，但同一系列 CPU 的指令系统常常具有向上兼容性，即较高级别的 CPU 指令系统是较低级别 CPU 指令系统的超集。例如，Intel 80486 指令系统包含 80386 指令系统。

（2）用机器语言设计程序非常困难，但容易实现高性能

正是与机器的密切相关性，使得用机器语言设计程序能最大限度地利用和发挥计算机的硬件功能和优势，容易得到时间和空间上的最优代码。然而，这种代码可移植性差，不仅难以理解和掌握，而且不易调试和维护，其开发效率也很低，难以胜任大型软件的开发需要。

## 1.1.2 汇编语言

为了克服机器语言难以记忆、表达和阅读的缺点，人们将机器指令符号化，以直观、便于记忆的符号来表示机器指令，这些符号被称作指令助记符（Mnemonic）。例如，用 ADD 表示加法指令，MOV 表示传送指令，MUL 表示乘法指令等。

以助记符描述的指令称作汇编格式指令或符号指令，通常简称指令。指令和伪指令的集合及其程序设计规则便构成了汇编语言。伪指令的概念将在第 4 章介绍。用汇编语言编写的程序就是汇编语言源程序。

利用汇编语言，计算表达式 36 × 68+10 的程序代码如下：

```
MOV    AL, 36
MOV    BL, 68
MUL    BL
ADD    AX, 10
```

显然，用汇编语言编写的程序要比机器代码更易理解。

然而，汇编语言仅仅是机器语言的符号化，每条汇编语言指令均对应唯一的机器指令，因而与机器语言并无本质区别，即具有机器语言“与机器的密切相关性”等特点，只是在直观和记忆

方面有了改进。

1. 汇编与汇编器

由于计算机只能识别机器语言，故必须将用汇编语言编写的源程序翻译成机器语言。把汇编语言源程序翻译成机器语言描述的目标程序的过程称作汇编。完成汇编任务的程序称作汇编器（Assembler）或汇编程序。

汇编器的主要功能是对汇编语言源程序进行语法检查，并生成相应的目标文件（.obj 文件）。汇编器类似于高级语言的编译器（Compiler）。

2. 连接与连接器

虽然目标文件已是机器语言程序，是二进制代码文件，但还不能直接运行，需要经过连接器（Linker，或称连接程序）将其与其他目标文件或库文件连接在一起，生成可执行文件（.exe 文件）后，方可在计算机上运行。连接器的主要功能是实现多个目标文件及库文件的连接，并完成浮动地址的重定位。

从汇编语言源程序到可执行程序的生成过程如图 1-1 所示。


图 1-1 汇编与连接过程

## 1.1.3 高级语言

虽然汇编语言较机器语言在记忆和直观等方面有了很大改进，但并无本质上的飞跃。人们迫切希望有一种接近自然语言或数学表达形式的程序设计语言，使程序设计工作能避开与机器硬件相关的细节，而着重于解决问题的算法本身，因此便产生了高级语言。例如，可以在程序中直接使用表达式 36*68+10。目前，常用的高级语言有数十种，如 Pascal，C，C++，Basic，Java 等。

高级语言在程序设计的简易性与代码的可移植性等方面有了质的提高。当然，用高级语言编写的源程序必须经过编译和连接，将其转换为可执行程序或借助于解释程序方可运行。

## 1.1.4 对汇编语言的评价

高级语言简单、易学且开发效率高，而汇编语言复杂、难懂、开发效率低。那么，是否就意味着没有必要学习和使用汇编语言了呢？对于这一问题，也存在着不同看法。支持使用汇编语言的观点认为，汇编语言具有如下优势。

（1）用汇编语言容易得到高时空效率的程序。由于汇编语言本质上就是机器语言，可直接、有效地控制计算机硬件，因而与高级语言相比，容易得到运行速度快、执行代码短、占用内存空间少的高时空效率的目标程序。

（2）用汇编语言能设计出高级语言无法实现的程序。正是由于与机器的密切相关性，使得汇编语言能充分利用计算机的硬件特性，编写出与硬件紧密相关而高级语言又无法实现的程序来。

另一方面，对汇编语言持相反观点的人们则认为：

（1）汇编语言难学、难理解、难调试、难维护等；

（2）汇编语言程序可移植性差；

（3）随着计算机运行速度的提高和内存容量的增加，人们对时空效率的需求已不再迫切，因而汇编语言的优势也就不再突出。

尽管如此，汇编语言还是在某些方面具有高级语言无法比拟的独特优势。因此，学习和使用汇编语言的意义可归纳为如下四个方面。

（1）速度：对于同一个问题，用汇编语言设计出的程序能达到“运行速度最快”。

（2）空间：对于同一个问题，用汇编语言设计出的程序能达到“占用空间最少”。

（3）功能：汇编语言可以实现高级语言难以胜任甚至不能完成的任务。

（4）知识：掌握汇编语言知识，有助于加深对计算机系统特别是程序执行逻辑的理解，有助于写出更好的高级语言程序来。很难想象，一个没有汇编语言知识的程序员能写出高质量的程序来。至少对于计算机专业人员来说，汇编语言是非常重要的。

当然，我们不能期望用汇编语言开发大型的应用软件系统，而应尽可能扬长避短。汇编语言正是由于其“面向机器”的特点，广泛用于高性能软件的开发中。例如，操作系统的核心代码要求有快速响应的实时系统等。

虽然汇编语言不具有通用性，不同类型 CPU 的指令系统可能有较大差异，但其原理和方法是具有普遍性的。只要熟练掌握一种汇编语言，再学习其他汇编语言是相当容易的。

## 1.2 数 据 表 示

现代计算机系统采用二进制（Binary）来表示数据。然而，二进制书写太冗长，而十进制又与二进制的计算机系统不相吻合。因此，为了描述方便，引入了十六进制（Hexadecimal）。十六进制数简短、易读，而且与二进制数之间的转换非常容易。下面介绍一些数制方面的知识。

十进制数用数字 0～9 描述，基数是 10，运算规则是逢 10 进 1。

二进制数用数字 0～1 描述，基数是 2，运算规则是逢 2 进 1。

十六进制数用数字 0～9 和 A～F（或 a～f）描述，其中 A～F（a～f）表示 10～15，基数是 16，运算规则是逢 16 进 1。为了与标识符区分，若以字母打头，则前面补 0。

八进制数用数字 0～7 描述，基数是 8，运算规则是逢 8 进 1。

实质上，对于 $X$ 进制数来说，用数字 0～$X$-1 描述，基数是 $X$，运算规则是逢 $X$ 进 1，实际值为各位数字与权值乘积之和。$X$ 进制数

$$a_n a_{n-1}\cdots a_0\, a_{-1}a_{-2}\cdots a_{-m}$$

等价于十进制数

$$a_n X^n + a_{n-1}X^{n-1} + \cdots + a_0 X^0 + a_{-1}X^{-1} + a_{-2}X^{-2} + \cdots + a_{-m}X^{-m}$$

前导 0 可以忽略，不影响取值。

在汇编语言中，为了区分各种不同进制数，在结尾以一个字母表示。其中，十进制数书写时结尾用 D 或 d，二进制数用 B 或 b，十六进制数用 H 或 h，八进制数用 Q 或 q，缺省为十进制数。

## 1.2.1 数据组织

1. 位（bit）

计算机中数据的最小单位是一个二进制位。除非特别指出，位即指二进制位。一位可以表示0和1两个值，当然也可用来描述任意两个对象，如真与假、开与关、对与错等，甚至可以表示236与5 623，依赖于使用者赋予的含义。

尽管在理论上，我们可以使用任意位的数，但计算机是在特定位数下工作的，一般是8或16的倍数，如8位、16位和32位等。

2. 字节（Byte）

1个字节是8位。字节是Intel 80x86 CPU可寻址的最小数据单位，基于80x86的内存与I/O空间均以字节编址。也就是说，80x86程序可以存取的最小数据单位是字节。例如，若要读取的位数不足8位，则只能先读出一个完整字节，再屏蔽掉其他位。

位编号从右到左依次为0～7，如图1-2所示。其中，第0位被称作最低（Low Order）位或最低有效（Least Significant）位，第7位被称作最高（High Order）位或最高有效（Most Significant）位。一个字节可以表示$2^8$（即256）个不同值。例如，0～255。

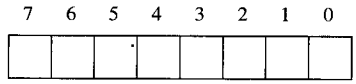



图1-2 字节的位编号

3. 字（Word）

一个字是16位。位编号从右到左为0～15。其中，第0位被称作最低位，第15位被称作最高位。0～7位称作低字节（Low Order Byte），8～15位称作高字节（High Order Byte），如图1-3所示。一个字可以表示$2^{16}$（即65 536）个不同值。例如，0～65 535。

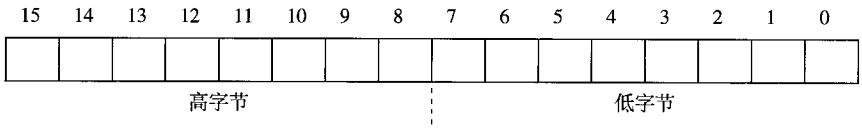



图1-3 字的位编号

4. 双字（Double Word）

一个双字是32位。位编号从右到左为0～31。其中，第0位被称作最低位，第31位被称作最高位。0～15位称作低字（Low Order Word），16～31位称作高字（High Order Word），如图1-4所示。一个双字可以表示$2^{31}$（即4 294 967 296）个不同值。例如，0～4 294 967 295。

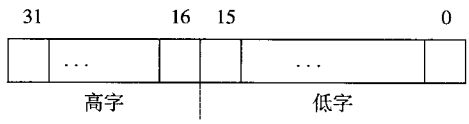



图1-4 双字的位编号

## 1.2.2 无符号数与带符号数

1. 无符号数

到目前为止，我们把二进制数看作无符号数（Unsigned Number），因此，$n$位二进制数可以表示的无符号数范围为0～$2^n-1$。例如，8位二进制数00H～0FFH表示0～255，16位二进制数

0000H～0FFFFH 表示 0～65 535。那么，如何表示负数呢？

2. 带符号数的补码表示

80x86 CPU 采用二进制补码表示带符号数（Signed Number），以最高位作为符号位（0 表示正数，1 表示负数）。例如，对于 16 位数来说，8000H 是负数，0FFFH 是正数。具体表示规则为：

（1）正数的补码是其本身；

（2）负数的补码是对其正数“各位求反、末位加 1”后形成的。

我们把“各位求反、末位加 1”的操作称作求补。

$n$ 位二进制补码数可以表示的带符号数范围为$-2^{n-1}$～$2^{n-1}-1$。例如，8 位二进制数可以表示−128～127，16 位二进制数可以表示−32 768～32 767，32 位二进制数可以表示−2 147 483 648～2 147 483 647。

例如，对于 8 位数来说：

$[5]_{补}$ = 00000101B

$[-5]_{补}$ = 11111011B

$[0]_{补}$ = 00000000B

$[-0]_{补}$ = 00000000B

那么，−5 的补码是如何得到的呢？计算过程如下：

| | |
|---|---|
| 0000 0101 | 5 的二进制表示 |
| 1111 1010 | 各位求反 |
| 1111 1011 | 末位加 1，得−5 |

反之，对补码表示的−5 进行求补操作，可得到 5。计算过程如下：

| | |
|---|---|
| 1111 1011 | −5 的 8 位二进制补码表示 |
| 0000 0100 | 各位求反 |
| 0000 0101 | 末位加 1，得 5 |

实质上，求补就是求相反数。例如下列求补操作。

| | | | |
|---|---|---|---|
| 7FFFH: | 0111 1111 1111 1111 | +32 767 | （16 位带符号数的最大值） |
| | 1000 0000 0000 0000 | 各位求反 | （8000H） |
| | 1000 0000 0000 0001 | 末位加 1 | （8001H 或−32 767） |
| 4000H: | 0100 0000 0000 0000 | +16 384 | |
| | 1011 1111 1111 1111 | 各位求反 | （0BFFFH） |
| | 1100 0000 0000 0000 | 末位加 1 | （0C000H 或−16 384） |
| 8000H: | 1000 0000 0000 0000 | −32 768 | （16 位带符号数的最小值） |
| | 0111 1111 1111 1111 | 各位求反 | （7FFFH） |
| | 1000 0000 0000 0000 | 末位加 1 | （8000H 或−32 768） |

8000H 求补后仍得 8000H。难道−32 768 的相反数是−32 768？问题在于，+32 768 超出了 16 位带符号数的表示范围，因而出现溢出。

补码具有如下特性。

求补

（1）$[x]_{补} \Longleftrightarrow [-x]_{补}$

例如，在8位二进制表示下，

$[10]_{补}$ = 00001010B

求补后得

$[-10]_{补}$ = 11110110B

反之亦然。

（2）$[x+y]_{补} = [x]_{补} + [y]_{补}$

（3）$[x-y]_{补} = [x]_{补} + [-y]_{补}$

例如，在8位二进制表示下，实现25减32，可用减法：

$$\begin{array}{rr} & 0001\ 1001\text{B}\ \ (25) \\ - & 0010\ 0000\text{B}\ \ (32) \\ \hline & 1\ \nwarrow 1111\ 1001\text{B}\ \ (-7) \end{array}$$

其中，向高位的借位丢失。或者转换为如下的加法：

$$\begin{array}{rr} & 0001\ 1001\text{B}\ \ (25) \\ + & 1110\ 0000\text{B}\ \ (-32) \\ \hline & 1111\ 1001\text{B}\ \ (-7) \end{array}$$

结果相同。

因此，在计算机内部，补码减法是通过对减数求补后，再将减法转换为加法进行的。为什么能保证结果的正确性呢？请看补码的物理意义。

3. 补码的物理意义

以现实生活的钟表对时为例。假设钟表目前指向10点整，而正确的时间应该是6点整。那么，要使钟表指向正确位置，通常有下列两种拨法：

① 按逆时针方向拨4小时，即

$$10 - 4 = 6$$

② 按顺时针方向拨8小时，即

$$10 + 8 = 6$$

这里，意味着存在下列等式：

$$10-4 = 10 + 8 = 6$$

为什么这样呢？因为钟表的表示范围是0～11，12即0，因此

$$10 + 8 = 12 + 6 = 6$$

我们说，-4与8关于12互为补数，即在钟表这种环境下，-4等同于8。

推而广之，考虑8位二进制数，其表示范围为0～255，即256等同于0。若将其想象为一个环，以0为基点，向顺时针方向移246个单位，则得246。然而，若按逆时针方向移动，则该位置就是-10。即

$$-10 = 0F6H = 246$$

因此，在8位二进制表示下，对于负数$x$（-128～-1）来说，存在下列等式：

$$-x = 256 - |x|$$