微机汇编语言程序设计学习指导

许　远　何成彦　编

电子科技大学出版社

微机汇编语言
程序设计学习指导

许　远　何成彦　编

电子科技大学出版社
·1996·

内容简介

汇编语言是重要的程序设计语言，它是介于机器语言与高级语言之间的过渡性语言。本书以简洁的语言风格，剖析了汇编语言程序的基本结构、数据表示法、指令系统、伪指令、操作符和寻址方式，介绍了汇编语言中特有的语法内容，包括结构、记录、宏、堆栈。本书着重讲解了汇编语言程序设计的基本要素和高级的结构化程序设计方法。

本书是《微机汇编语言程序设计基础教程》的配套丛书，与该书配合使用能使读者在最短的时间内学会汇编语言程序设计，并且养成结构化编程的好习惯，为今后进一步学习奠定坚实的基础。本书既可作为广大计算机专业人员和普通高等学校计算机基础教育的参考用书，也可作为计算机等级考试(三级)的参考教材。

微机汇编语言程序设计学习指导

许　远　何成彦　编

*

电子科技大学出版社出版

(成都建设北路二段四号)邮编 610054

四川省自然资源研究所印刷厂印刷

新华书店经销

*

开本 787×1092　1/16　印张 11.75 字数 270 千字

版次　1996 年 6 月第一版　印次　1996 年 6 月第一次印刷

印数 1- 000 册

ISBN 7-81043-526-4/TP・204

定价：10.80 元

前　言

对于初学汇编语言的读者来说，要过好入门关是有一定难度的，因为与其他高级语言相比，汇编语言是属于低层次的程序设计语言，它更加接近机器的硬件系统。但是汇编语言即使是在当今其他高级语言相当普及的情况下仍是很有用处的，一些对时间要求特别严格的程序和直接操纵硬件的程序都要用汇编语言编写。

本书以简单明了的方式解释了汇编语言程序设计的方法和基本技能。主要内容包括汇编语言程序设计的要素、8088体系结构、数据表示法、伪指令、指令系统、寻址方式、堆栈、过程的基本概念和输入输出、中断程序设计、条件汇编和宏汇编、字符串的处理程序等基本技能。

本书不像其他汇编语言书籍那样系统地讲述汇编程序设计的各个细节，而是从易学、易模仿的角度出发，给出每个指令伪指令的应用例程和使用注意事项，同时根据读者对汇编语言的认识过程，调整了汇编语言程序设计的传统体系，力求做到学了就能用，学用结合。

本书的付梓与电子科技大学出版社许宣伟编辑的关心支持是分不开的，电子科技大学电子工程系的陈祝明副教授、电子科技大学电子实验中心李培茂同志也对本书的出版做出了重要贡献，在此一并致以衷心感谢。

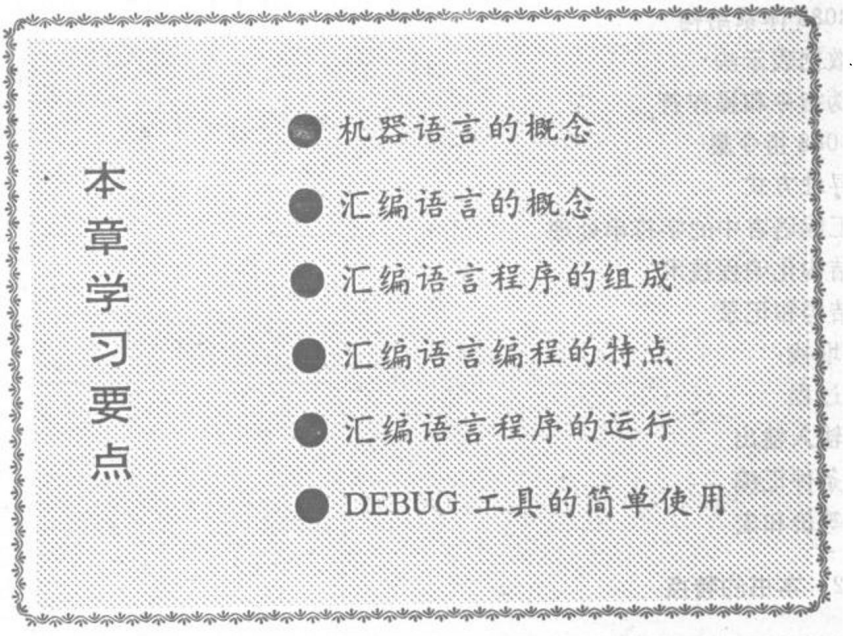
在本书的编写过程中，作者参考了许多参考资料，既有国内的，也有国外的，在本书的参考文献中已一并列出，谨在此向这些图书的作者表示诚挚的谢意。

编　者

1996.3 于北京

目　录

第一章 基础知识


本章学习要点
机器语言的概念
汇编语言的概念
汇编语言程序的组成
汇编语言编程的特点
汇编语言程序的运行
DEBUG 工具的简单使用

§1.1 引 言

在计算机发展的早期，用于程序设计的语言只有一种——机器语言。但是，用机器语言编程很不方便，于是出现了设计了一种以符号名称代表机器语言指令的编程语言。这种符号表示的编程语言就是汇编语言(Assembly Language)。后来计算机设计者又开发了诸如 Fortran 和 COBOL 等高级语言(High-level Language)，极大地简化了用户的编程工作。虽然目前完全使用汇编语言编写的应用程序很少见，但一些忠实的汇编语言程序员不会放弃选用汇编语言的机会。更常见的是，程序员在高级语言中嵌入汇编语言以加速某些对运行时间要求特别严格的程序。

1.1.1 本书主要内容

本书讲解了使用汇编语言编程的方法，书中涉及了汇编语言所有的基本特性，包括：

(1)汇编语言程序的要素
(2)8088 体系结构
(3)数据表示法
(4)伪指令和操作符
(5)8088 指令集
(6)寻址方式
(7)汇编语言中的字符串处理
(8)结构化编程技术
(9)结构和记录
(10)堆栈
(11)过程
(12)输入输出
(13)条件汇编
(14)等价和宏

1.1.2 本书的特点

本书不象其它讲解汇编语言的书那样，系统地讲述汇编语程序设计的各个细节，而是采用简明的方式讲解编程中要用到的知识，让读者能顺利地读完本书，对汇编语言有初步印象，能初步建立对汇编语言的总体认识，这样有助于消除学习的恐惧心理。更为系统的叙述在本书的姐妹篇《汇编语言程序设计基础教程》中进行讲述。

§1.2 机器语言与汇编语言

1.2.1 机器语言

在开始汇编语言编程的学习之前，必须了解机器语言编程。一般的计算机只明白两种不

同的状态:关和开。这两种数字状态通常以二进制数字 0 和 1 表示。而且,一位二进制数字习惯称之为一个位(Bit)。机器语言程序的表达式就由这些包含 0 和 1 的位串组成的。这些位串被称为位模式(Bit Patterns)。

【例 1-1】下面的二进制数表示一个机器语言位模式:

```
1011100000000010100000000
```

这个机器语言语句就是采用位模式表示指令、数据和指令数据的地址等信息的。前 8 位代表 8088 机器语言指令,后 16 位代表一个数据值。下面说明计算机是怎样理解上述模式的:

```
10111000 0000010100000000
MOV AX          03
```

位模式的前 8 位命令计算机把指令后面的 16 位数字送到名叫 AX 的寄存器(Registers)。在上述指令中,机器语言指令命令 CPU 在 AX 寄存器中存放数值 03(数字在位模式中,高位在后,低位在前)。

虽然机器语言运行速度快,能直接别计算机识别并执行,但毕竟太难以理解。因为,一个机器语言程序可以由大量位模式组成,而记住各种位模式完成的功能,是非常困难的。为了以更容易理解的方式表达这些机器语言指令,早期的程序设计者发明了一种方法,将机器语言中的各种位模式用相应的英文助记符号表示,这种语言就是汇编语言(Assembly Language)。

1.2.2 汇编语言

在汇编语言中,机器语言指令的专用名称为助记符(Memonics)。另外,数据可以用数字值(例如 5,16,0bfh,33,555,1)或者符号名称(例如 MAX,count,line,或 RESULT)表示。例1 1 中,将"101100"用助记符"MOV AX"表示,而后面的"000001010000000"表示数字 03,这就变成了易于理解的汇编语言形式了:

```
MOV AX ,03
```

上述的汇编语言语句比相应的机器语言位模式好记得多。遗憾的是,计算机不知道上述汇编语言语句的含义。为把上述语句翻译成计算机能够理解的格式,必须使用汇编程序。汇编程序功能是把汇编语言程序语句翻译成等价的机器语言位模式。

§1.3 汇编语言程序的组成

8088 汇编语言程序由以下四个基本部分组成:代码段、数据段、堆栈段、标号、注释。

1.3.1 代码段

代码段(Code segments)是由汇编指令组成的程序部分,这些指令完成各种任务,诸如传送数据、控制程序流程、实现算术运算功能等等。每条汇编语言指令由两个基本部分组成:操作符(Operation)和操作数(Operands)。

【例 1-2】说明指令"MOV AX ,03"的组成。

```
操作符          操作数
```

```
MOV        AX ,03
```

助记符 MOV 命令 CPU 传送一个数据。操作数 AX 与 03 告诉 CPU 要传送的数据是 03，要送到 AX 寄存器。

注意：

①所有的汇编语言指令都需要操作符，但不一定需要操作数；

②有的汇编语言指令需要 2 个操作数，而有的仅需要 1 个操作数。

代码段的定义形式如下：

〈段标识符〉SEGMENT [定位类型][组合类型][类别]

⋮

〈段标识符〉ENDS

[定位类型][组合类型][类别]在一般情况下可以省略不写。以下将要讲述的数据段，堆栈段的定义形式与上同。

有关代码段的具体例子见例 1-5。

1.3.2　数据段

虽然代码段是一个汇编程序必不可少的部分，但是几乎所有的 8088 汇编语言程序都至少需要一个数据段，用来存放有关数据。

【例 1-3】以下的数据段定义了 3 个 16 位变量(X,Y,Z)。

```
DATA SEGMENT
    X  DW  3
    Y  DW  4
    Z  DW  ?
DATA ENDS
```

上述语句中的三个 DW 是汇编伪指令(Assemble Directives)，指示汇编程序定义三个属性为字的存储单元。对于 8088，一个字是 16 位长。因此，上面的三个语句就定义了所需的三个 16 位变量。16 位存储单元 X 中存放数值 3，16 位存储单元 Y 中存放数值 4，而 16 位存储单元 Z 中存放不定值。因为上述数据段中的"?"告诉汇编程序 16 位存储单元 Z 的值尚未确定。

1.3.3　堆栈段

堆栈(stack)是 8088 体系中的一种特殊类型的存储器，它的特点是"先进后出，后进先出"，即最先入栈的数据最后才弹出。

下面举例说明堆栈的概念。作为一个简单的例子，可以把食堂里洗净的一摞碗看作一个栈。在通常情况下，最先洗净的碗总是放在最底下，后洗净的总是摞在先洗净的上面，最后洗净的总摞在最顶上。而在使用时，却是从顶上拿取，也就是说，后洗的先取用，后摞上的先取走。又如图 1-1 所示的铁路扳道站 B 也是一个栈，车辆由轨道 A 进入 B，B 上的车辆从轨道 C 离去，后进入 B 的车辆将先调出。如果我们把洗净的碗"摞上"和车辆由轨道 A"进入 B"称为进栈，把"取用碗"和车辆由 B"调出"称为出栈，那么，上述两例的共同特点是：后进栈的先出栈。我们常用图来形象地表示栈，其形式如图 1-2。

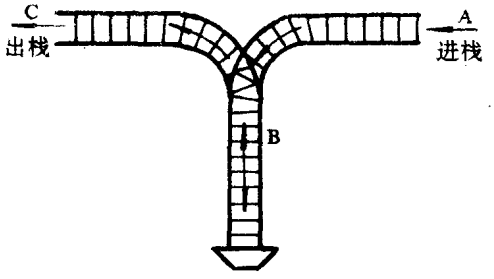


图 1-1

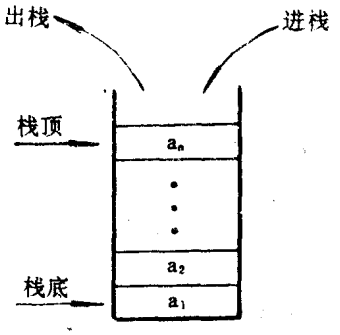


图 1-2

堆栈段在汇编程序设计中，一般用来保存程序的返回地址，在调用子程序前保存当前各个寄存器的值(也称保存程序运行现场)。也可以用来临时保存一些变量值。

堆栈段的定义形式如下：

〈段名〉SEGMENT STACK

⋮

〈段名〉ENDS

在 SEGMENT 后加上 STACK 就可以定义一个段为堆栈段，堆栈段定义例子见例 1-5。

1.3.4 标　号

例 1-3 数据段中的符号名 X、Y 和 Z 是汇编语言标号(Labels)的一个极好实例。除了用作变量名以外，汇编语言标号还可用作名称，例如段名或过程名。合法的 8088 汇编语言标号由数字、字母和字符?、·、@、_和 $ 组成。另外，名称不能以数字打头，点号(·)只能用作标号的第一个字符。标号的长度不限，但只有前 31 个字符有效。除非标号用于连接汇编伪指令，否则后面必须跟一个冒号(:)。

【例 1-4】标号使用示范。

```
x1     DW     0        (标号 x1,用于伪指令 DW 中,不用冒号)
exit1: MOV    AX,0     (标号 exit1,它表示一个汇编语句的存储首地址,由于不用于伪指令,后面
                        要有":")
1a DW 0                 (1a 是非法标号,以数字打头)
_TX                     (合法的标号)
.A2                     (合法的标号)
a#b                     (非法标号,含有不允许出现的字符)
```

1.3.5 注　释

汇编语言程序的第四部分是注释(Comment)。顾名思义，注释描述程序所做的工作而并不影响程序的实际操作。虽然注释看起来不是必需的，但它们是任何程序的一个重要组成部分。通常在以后程序员需要调试或修改程序时，利用注释会明显地减少重新读懂完成某些功能的程序所需的时间。在 8088 汇编语言程序中，编写注释只需以分号(;)开头即可。下例列举了一些汇编语言注释：

```
MOV AX,3;AX is set to 3
    ;The previous instruction sets register AX to 3
```

1.3.6 一个汇编语言程序范例

【例 1-5】下面列举了一个非常短小的 8088 汇编语言程序。虽然相当简单，它的功能是求两个数 x，y 的和，并把结果存在 z 中，但是它指出了组成一个汇编语言程序的要素。首先，查看下面的程序，然后逐行解释。

```
;
;文件名:First.asm
;以下是数据段
data SEGMENT
  x DW 7
  y DW 9
  z DW ?
data ENDS           ;数据段结束
;以下是堆栈段
st1 SEGMENT STACK
  DB 16 DUP (0)
st1 ENDS            ;堆栈段结束
;以下是程序开头的固定语句
PROG SEGMENT
      ASSUME CS:PROG,DS:data,SS:st1
start:
    PUSH DS         ;寄存器 ds 的原值进入堆栈
    SUB AX,AX       ;寄存器 ax 的值清零
    PUSH AX         ;寄存器 ax 的值进入堆栈
    MOV AX,DATA
    MOV DS,AX       ;寄存器 ds 的值置为 data 的段地址
    ;准备完毕,以下是求 x+y 和的程序
    MOV AX,X        ;将 x 的值送到 ax
    ADD AX,Y        ;将 y 的值与 x 求和,结果送到 ax
    MOV Z,AX        ;结果送到 z
    MOV AX,4C00H
    INT 21H         ;程序结束,返回 DOS
PROG ENDS           ;代码段结束
END start           ;汇编结束
```

从上述程序可以看出，用低级语言编写程序，是很繁琐的，它不能像高级语言编程序那样只指明“做什么”，而必须详细描述“如何做”，这就是用汇编这样的低级语言编程的困难所在。汇编语言是介于计算机能够理解的代码语言(即机器语言)与使用者容易理解的高级语言之间的一种语言。它以助记符代替二进制码帮助使用者记忆和使用代码语言，所以又称为符号语言，用汇编语言编出的程序与高级语言源程序比较，难读，容易出错，而且因为每种系