微机原理与应用实验教程

WEIJIYUANLIYUYINGYONGSHIYANJIAOCHENG

胡震伟 编

北京邮电大学出版社

微机原理与应用实验教程

胡震伟 编

北京邮电大学出版社

图书在版编目（CIP）数据

微机原理与应用实验教程/胡震伟编．－北京：北京邮电大学出版社，1998.2
ISBN 7－5635－0298－X

Ⅰ．微…　Ⅱ．胡…　Ⅲ．微型计算机－理论　Ⅳ．TP36

中国版本图书馆 CIP 数据核字（97）第 28626 号

TS272/15

内 容 简 介

计算机提供给用户最快而又最有效的语言是汇编语言，它是利用计算机所有的特性直接控制硬件的唯一语言。本教材着重介绍两方面的内容：①软件实验——宏汇编语言程序从建立经调试到运行的全过程；②硬件实验——在程序控制下硬件所完成的各种功能。

本书可作为工科院校电子类专业、机电自动控制专业及其它非计算机专业的实验教材及从事电子技术的工程技术人员参考。

微机原理与应用实验教程

编　　者　胡震伟

责任编辑　时友芬

*

北京邮电大学出版社出版

新华书店北京发行所发行　各地新华书店经售

河北省高碑店市印刷厂印刷

*

787×1092 毫米　1/16　印张 13.625　字数 348 千字

1998 年 2 月第一版　　1998 年 2 月第一次印刷

印数：1—1 000 册

ISBN 7－5635－0298－X/TP·19　　定价：19.00 元

前　言

汇编语言是计算机提供给用户的最快而又最有效的语言，能够利用计算机所有的特性直接控制硬件的唯一语言。本教材着重介绍两方面内容：1. 软件实验——宏汇编语言程序从建立经调试到运行的全过程。2. 硬件实验——在程序控制下硬件所完成的各种功能。

本教材共分五章。第一、二章简单介绍 INTEL 8088/8086 CPU 汇编语言程序、宏指令、文本编辑、汇编 MASM 和连接 LINK。第三章介绍汇编语言程序的调试与运行。第四章是软件上机实验。第五章介绍在程序控制下的硬件实验。

本教材宜作为“微机原理与应用”课程的配套教材，以充实与巩固原理的学习。微机原理与应用实验教程是一门实践性很强的课程，可供高等工科院校电子类专业、机电自动控制专业及其它非计算机专业作微机技术应用的实验教程，也可供从事电子技术工作的工程技术人员参考。

限于水平，本书的缺点和不足之处在所难免，敬请教材使用单位及读者批评指正，本人将不胜感激。

作　者

1997 年 12 月

目　录

第一章　宏汇编语言 …… (1)

1.1　宏汇编语言程序格式 …… (1)
1.2　宏指令 …… (6)

第二章　汇编语言程序的编辑、汇编与连接 …… (17)

2.1　文本编辑 …… (17)
2.2　汇编程序 MASM …… (19)
2.3　连接程序 LINK …… (21)

第三章　汇编语言程序的调试与运行 …… (38)

3.1　SYMDEB 命令及其应用 …… (38)
3.2　Code View 查错程序应用于调试汇编语言程序 …… (56)
3.3　Turbo Debugger 调试程序简介 …… (95)

第四章　汇编语言程序上机实验 …… (99)

4.1　数据传送实验 …… (99)
4.2　算术逻辑运算实验 …… (102)
4.3　乘法与循环指令应用实验 …… (103)
4.4　串操作运算实验 …… (108)
4.5　简单编程练习 …… (110)
4.6　代码转换实验 …… (112)
4.7　字符配匹实验 …… (114)
4.8　二进制数转换成十六进制数显示实验 …… (116)
4.9　排序实验 …… (118)
4.10　屏幕显示时钟实验 …… (120)
4.11　键盘中断实验 …… (122)
4.12　定时中断实验 …… (125)
附录　编程参考 …… (127)

第五章　硬件实验 …… (135)

5.1　TPC－1 型微机实验台 …… (135)
5.2　可编程定时/计数器 8253 实验 …… (148)
5.3　在 SYMDEB 环境下调试 LED（七段）显示器 …… (151)

5.4 用 8253 作中断源实现分秒电子钟实验 …… (156)
5.5 实时时钟实验（EX331.ASM） …… (161)
5.6 可编程并行接口芯片 8255A 实验 …… (168)
5.7 8259A 中断控制器实验 …… (181)
5.8 8251A 串行口实验 …… (185)
5.9 DMA 实验 …… (194)
5.10 D/A 实验 …… (198)
5.11 A/D 实验 …… (203)
参考文献 …… (212)

第一章　宏汇编语言

宏汇编语言是计算机提供给用户的最快且最有效的语言，能利用计算机所有硬件特性及直接控制硬件的唯一语言，因而适用于对于程序的空间和时间要求上有很高的场合。例如，由于宏汇编语言有着任何高级语言所不能比及的快速优越性，所以适合直接控制硬件的应用场合。

1.1　宏汇编语言程序格式

宏汇编语言程序(以下称汇编程序)主要由指令、数据、堆栈、注释等内容组成。

指令：CPU 指令，即 8086/8088 CPU 指令，有机器码。伪指令，即无机器码由 MASM 执行。宏指令，由 MASM 及 CPU 来执行。其表示格式由以下几种：逻辑段分段定义、整个程序定义在一个逻辑段中和简化的段定义。

1.1.1　逻辑段分段定义

逻辑段有数据段、附加段、堆栈段、代码段四种，每个逻辑小于或等于 64 K。

1. 定义数据段或附加段格式

```
段名    SEGMENT (连接方式)
        ⋮| DB, DW, DD  变量定义语句
段名    ENDS
```

例 1.1

```
DATA1   SEGMENT
  A1    DB    7, 8
  B1    DB    25H
  A2    DW    1, 2
        DW    $ +4, $ +3
DATA1   ENDS
DATA2   SEGMENT
  ORG   12H
  B2    DB    'BOY'
  B3    DB    2 DUP (0)
  B4    DW    1234H, 2236H
DATA2   ENDS
```

这两个数据段其内存分布见图 1.1。

地址	
DATA1→A1 = 0000H	7
0001H	8
B1 = 0002H	25H
A2 = 0003H	01
0004H	00
0005H	02
0006H	00
0007H	0BH
0008H	00
0009H	0CH
000AH	00
000BH	
000CH	
	⋮
B2 = 0012H	'B'
0013H	'O'
0014H	'Y'
B3 = 0015H	00
0016H	00
B4 = 0017H	34H
0018H	12H
0019H	36H
001AH	22H

图 1.1　内存分布

2. 定义堆栈格式

```
段名    SEGMENT    STACK
```

```
        DW      20H
    TOP     LABLE   WORD
段名        ENDS
```

或定义为

```
段名    SEGMENT         STACK
        DW      20H
    TOP DW  ?
段名    ENDS
```

以上堆栈其内存分布见图 1.2。

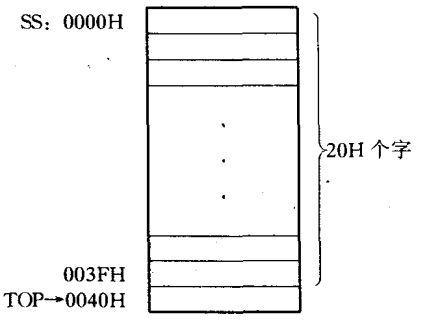

图 1.2　堆栈分布

3. 定义代码段格式：

```
段名  SEGMENT   [组合类型]
ASSUME  CS:段名,DS:数据段名,ES:附加段名,SS:堆栈段名
START:
            :   代码（指令）段内容
段名  ENDS
    END     START
```

例 1.2　一个简单程序用分段定义方式的书写格式如下：

```
            ; DISCH
DATA    SEGMENT
  BUF   DB  'IBM MICRO COMPUTER $'
DATA    ENDS
STACK   SEGMENT     STACK
        DW      10H
  TOP   DW  ?
STACK   ENDS
CODE    SEGMENT
  ASSUME CS:CODE, DS:DATA, SS:STACK
START:      MOV     AX, DATA
            MOV     DS, AX      ; DS 初始化
            MOV     AX, STACK
            MOV     SS, AX      ; SS 初始化
            MOV     SP, OFFSET TOP
            LEA     DX, BUF
            MOV     AH, 9       ; 显示提示符
            INT     21H
            MOV     AH, 4CH
            INT     21H         ; 返回 DOS
CODE    ENDS
    END     START
```

1.1.2 整个程序定义在一个逻辑段中

若整个程序各逻辑段长度之和(总长度)不大于 64 K，则可将整个程序定义在一个逻辑段中。例 1.2 可编写为

```
                ; DISCH
CODE            SEGMENT
  BUF           DB  'IBM MICRO COMPUTER $'
  STACK         DW    10H
  TOP           DW ?
  ASSUME  CS:CODE, DS:CODE, SS:CODE
    START:MOV     AX, CODE
          MOV     DS, AX
          MOV     SS, AX
          LEA     SP, TOP
                : 最后 5 条指令同上
CODE    ENDS
    END   START
```

1.1.3 简化的段定义

MASM 5.0 版本提供一种简化系统以说明段。简化的段伪指令采用段名和 Microsoft 高级语言约定。接受这些约定，则能自动控制段定义中较难把握的方面。

用户编写独立的汇编语言程序，其中的段名、排列和其它定义因素不重要，则简化的段伪指令使程序设计更容易。Microsoft 约定很灵活，适用于大部分程序。

用户编写与 Microsoft 高级有语言连接汇编例行程序时，采用简化的段伪指令，能够自动地定义一致且正确的名字，可保证用户模块的兼容性。

使用简化的段伪指令时，可自动产生与 Microsoft 约定一致的 ASSUME 和 Group 语句，不过，大多数程序不需要这些伪指令。读者可按本章例中的格式使用简化的段伪指令。

注意：

简化的段伪指令不能用在 .COM 格式的程序中，必须针对这种格式特别定义所需要的单个的段。

1. 存储模型

使用简化的段伪指令时，必须在程序中描述“存储模型”。存储模型说明了程序所用数据和代码的缺省大小。

Microsoft 高级语言要求每个程序有一个存储模型。由高级语言程序调用的任何汇编语言例行程序应当具有与调用程序相同的存储模型。以下介绍常用的存储模型。

小型 SMALL	全部数据限制在单个 64 K 段内，全部代码限制在 64K 段内，因此，所有代码和数据能按 NEAR 访问。这是独立汇编语言程序最常用的模型。C 是唯一支持这种模型的 Microsoft 语言。

中型 MEDIUM	全部数据限制在 64K 段内，但代码可大于 64K。因此，数据寻址是 NEAR，代码寻址是 FAR。大多数最新的 Microsoft 语言版本支持这种模型。
紧致型 COMPACT	代码限制在 64K 段内，数据可大于 64K，C 语言支持这种模型。
大型 LARGE	代码和数据均可大于 64 K，但数组不大于 64K，Microsoft 高级语言支持这种模型。
巨型 HUGE	代码段、数据段均可大于 64K，数组也可大于 64K，多数高级语言支持这种方式。

一般程序选用小型（Small）就足够了，数据与代码的访问均是 NEAR，所以速度较快，这种存储模型效率通常最高。

2. 指定 DOS 的段排列

DOSSEG 伪指令说明段依据 DOS 的段排列约定来排列。这是 Microsoft 高级语言程序所用的约定。

格式：DOSSEG

3. 定义存储模型

格式：.MODEL　存储模型

存储模型可选择 SMALL，MEDIUM，COMPACT，LARGE 或 HUGE。大型与巨型模型的段定义相同，但@date size 等式的值不同。若用户为一种高级语言写汇编语言例行程序，则存储模型应与编译程序或解释程序的存储模型匹配。

注意：用户在定义任意段前必须使用 .MODEL 伪指令。若在 .MODEL 伪指令前，使用了其它简化的段伪指令如 .CODE 或 .DATA 等将会出错。

使用次序如下：

```
DOSSEG
.MODEL SMALL
```

4. 定义简化的段

格式：.STACK　[大小] 堆栈段

.DATA　初始化的近数据段

.DATA?　未初始化的近数据段

.FARDATA　初始化的远数据段

.FARDATA?　未初始化的远数据段

.CODE　代码段

.CONST　常数数据段

其“大小”为堆栈的字节数，若不给出大小值，则段自动定义 1K 字节作为堆栈的大小。

.CODE 伪指令初始化的段内，不考虑存储模型。通常在一个源模块中只定义一个代码

段。若有多个代码段就必须说明“名字”来区分这些段。“名字”只能在允计有多个代码段的模型(中型和大型)中说明。在 SMALL 和 COMPACT 两种模型中即使给出“名字”也将被忽略。常数数据是必须在数据段中说明而在程序执行时不需要修改的数据。编写高级语言调用的汇编语言例行程序时,可用 Const 伪指令说明串、实数和其它必须按数据分配的常数数据。

例 1.3

```
    DOSSEG
   .MODEL      SMALL
   .STACK      100H
   .DATA
IVAR        DB    5
IARR        DW    50    DUP (?)
String      DB    'THIS  IS  A  STRING'
   EXTRN    XVAR:WORD
   .CODE
START:      MOV    AX, @DATA
            MOV    DS, AX
                   ⋮
            END    START
```

5. 使用预定义等式

段等式对每个主区段伪指令，相同的名字下存在一个对应的等式，只是等式以“@”号开头，而伪指令以“.”号开头。例如，@Code 等式表示由 .Code 伪指令定义的段，@data 表示由所有近数据段分享的组名，它可用来访问由 .data，.data?，.const 和 .STACK 段伪指令建立的段。这些等式可用在 ASSUME 语句中，在任何其它的时间，一个段必须通过名字访问。例如：

```
ASSUME es: @fardata; ASSUME ES to fardata
  MOV AX, @data
  MOV ds, AX
  MOV AX, @fardata
  MOV ES, AX
```

例 1.4

```
; DISCH
   DOSSEG
  .MODEL SMALL
  .STACK    10H     ; 定义堆栈长度为 16 个字节
  .DATA
  BUF  DB  'IBM MICRO COMPUTER $'
  .CODE
START: MOV    AX, @DATA
       MOV    DS, AX
```

```
        LEA     DX, BUF
        MOV     AH, 9       ; 显示字符串
        INT     21H
        MOV     AH, 4CH     ; 返回 DOS
        INT     21H
END     START
```

从以上例子可看出，如采用“分段定义”，该程序共用了 22 行，如采用“定义在一个逻辑段中”则共使用了 17 行，如采用“简化的段定义”则只使用了 15 行，所以它是最简单的一种方法。

1.2 宏 指 令

我们已经了解使用子程序结构具有很多优点，可以节省存储空间及程序设计所化的时间，可提供模块化程序设计的条件，便于程序的调试及修改等。但是，使用子程序也有一些缺点：为转子及返回、保存及恢复寄存器以及参量的传送等都要增加程序的开销，这些操作所消耗的时间以及它们所占用的存储空间，都是为取得子程序结构使程序模块化的优点而增加的额外开销。因此，有时特别在子程序本身较短或者是需要传送的参量较多的情况下使用宏汇编就更加有利。

1.2.1 宏定义和宏调用

宏是源程序中一段有独立功能的程序代码。它只需要在源程序中定义一次，就可以多次调用，调用时只需要用一个宏指令语句就可以了。

宏定义是用一组伪操作来实现的。其格式是：

```
MACRO NAME MACRO [DUMMY PARAMETER LIST]
        ⋮       (宏定义体)
    ENDM
```

其中 MACRO 和 ENDM 是一对伪操作。这对伪操作之间是宏定义体——一组有独立功能的程序代码。宏指令名（MACRO NAME）给出该宏定义的名称，调用时就使用宏指令名来调用该宏定义。宏指令名的第一个符号必须是字母，其后可以跟字母、数字或下划线字符。其中哑元表（DUMMY PAROMETER LIST）给出了宏定义中所用到的形式参数（或称虚参），每个哑元之间用逗号隔开。

经宏定义定义后的宏指令就可以在源程序中调用，这种对宏指令的调用称为宏调用。宏调用的格式是：

```
MACRO NAME   [ACTRAL PARAMETER LIST]
```

实元表（ACTRAL PARAMETER LIST）中的每一项为实元，相互之间用逗号隔开。

当源程序被汇编时，汇编程序将对每个宏调用作宏展开。宏展开就是用宏定义体取代源程序中的宏指令名，而且用实元取代宏定义中的哑元。在取代时，实元和哑元是一一对应的，即第一个实元取代第一个哑元，第二个实元取代第二个哑元……依此类推。一般说来，

实元的个数应该和哑元的个数相等，但汇编程序并不要求它们必须相等。若实元个数大于哑元个数，则多余的实元不予考虑；若实元个数小于哑元个数，则多余的哑元作“空”处理。另外，应该注意，宏展开后即用实元取代哑元后，所得到的语句应该是有效的，否则汇编程序将会指示出错。

下面用一个例子来说明宏定义、宏调用和宏展开的情况。

例 1.5 用宏指令定义两个字操作数相乘，得到一个 16 位的第三个操作数，作为结果。

宏定义：

```
MULTY    MACRO  OP1, OP2, RES
  PUSH   DX
  PUSH   AX
  MOV    AX, OP1
  IMUL   OP2
  MOV    RES, AX
  POP    AX
  POP    DX
     ENDM
```

宏调用：

```
         ⋮
MULTY    CX, VAR, XYZ [BX]
         ⋮
MULTY    240, BX, SAVE
```

宏展开：

```
         ⋮
+ PUSH   DX
+ PUSH   AX
+ MOV    AX, CX
+ IMUL   VAR
+ MOV    XYZ [BX], AX
+ POP    AX
+ POP    DX
         ⋮
+ PUSH   DX
+ PUSH   AX
+ MOV    AX, 240
+ IMUL   BX
+ MOV    SAVE, AX
+ POP    AX
+ POP    DX
         ⋮
```

宏展开后在指令前加上“+”号以示区别。从上面例子可以看出：宏指令可以带哑元，调用时可以用实元取代，而且实元可以是常数、寄存器、存储单元名以及用寻址方式能找到的地址或表达式等。

例 1.6 宏定义可以无变元。

宏定义：

```
SAV   MACRO
      PUSH      AX
      PUSH      BX
      PUSH      CX
      PUSH      DX
      PUSH      SI
      PUSH      DI
ENDM
```

宏展开则将宏定义体的内容全部列出。

例 1.7 变元可以是指令助记符或操作数。

宏定义：

```
FOO   MACRO     P1, P2, P3
      MOV       AX, P1
      P2        P3
      ENDM
```

宏调用：

```
FOO             VAR, INC, AX
```

宏展开：

```
+MOV            AX, VAR
+INC            AX
```

例 1.8 & 操作符在宏定义体中可以作为哑元的前缀，展开时可以把 & 前后的两个符号连在一起，这个符号可以是操作码（助记符）、操作数或是一个字符串。

宏定义：

```
FO    MACRO     P1
      JMP       TA&P1
      ENDM
```

宏调用：

```
FO              WORD-VAR
```

宏展开：

```
+ JMP           TAWORD-VAR
```

在这里如果宏定义写为

```
FO    MACRO     P1
      JMP       TAP1
      ENDM
```

则在展开时，汇编程序把 TAP1 看作一个独立的标号，并不把 TAP1 中的 P1 作为哑元看待，否则就不能得到预期的结果。

例 1.9 变元是 ASCII 串的情况。

宏定义：

```
MSG     MACRO     LAB, NUM, XYZ
  LAB&NUM  DB  'HELLO  MR.&XYZ'
        ENDM
```

宏调用：

```
MSG  MSG, 1, TAYLOR
```

宏展开：

```
+ MSG1  DB  'HELLO  MR.TAYLOR'
```

例 1.10 宏指令名可以与指令助记符或伪操作名相同，在这种情况下，宏指令的优先级最高，而同名的指令或伪操作就失效了。伪操作 PURGE 可以用来在适当的时候取消宏定义，以便恢复指令的原始含义。本例说明 PURGE 的用法。

宏定义：

```
ADD     MACRO     OPR1, OPR2, RESULT
        ⋮
        ENDM
```

宏调用：

```
ADD     X, Y, Z
PURGE       ADD
        ⋮
```

在宏调用后，用 PURGE 伪操作取消宏定义，以便恢复 ADD 指令的原始含义，在 PURGE ADD 后面所用的 ADD 指令，则服从机器指令的定义。

PURGE 伪操作可同时取消多个宏定义，此时各宏指令名之间需用逗号隔开。

例 1.11 LOCAL 伪操作的使用。宏定义体内允许使用标号，如

宏定义：

```
ABSOL    MACRO      OPER
         CMP        OPER, 0
         JGE        NEXT
         NEG        OPER
NEXT:
         ENDM
```

如果程序中多次调用该宏定义时，展开后会出现标号的多重定义，这是不能允许的。为此系统提供了 LOCAL 伪操作，其格式是

```
LOCAL     LIST OF LOCAL LABELS
```

其中局部标号表内的各标号之间用逗号隔开。汇编程序对 LOCAL 伪操作的局部标号表中的每一个局部标号建立唯一的符号(用?? 0000～?? FFFF)以代替在展开中存在的每个局部标号。必须注意，LOCAL 伪操作只能用在宏定义体内，而且它必须是 MACRO 伪操作后的第

一个语句，在 MACRO 和 LOCAL 伪操作之间还不允许有注释和分号标志。

本例中的 ABSOL 宏定义在考虑到有多次调用可能性的情况下，应定义为：

```
ABSOL   MACRO   OPER
        LOCAL   NEXT
        CMP     OPER, 0
        JGE     NEXT
        NEG     OPER
NEXT:
        ENDM
```

宏调用：

```
        ⋮
        ABSOL   VAR
        ⋮
        ABSOL   BX
        ⋮
```

宏展开：

```
        ⋮
+       CMP     VAR, 0
+       JGE     ?? 0000
+       NEG     VAR
+ ?? 0000:
        ⋮
+       CMP     BX, 0
+       JGE     ?? 0001
+       NEG     BX
+ ?? 0001:
        ⋮
```

宏定义允许嵌套，下面两个例子说明宏嵌套的情况。

例 1.12 宏定义中允许使用宏调用，其限制条件是：必须先定义后调用。

宏定义：

```
DIF     MACRO   X, Y
        MOV     AX, X
        SUB     AX, Y
        ENDM
DIFSQR  MACRO   OPR1, OPR2, RESULT
        PUSH    DX
        PUSH    AX
        DIF     OPR1, OPR2
        IMUL    AX
```

```
        MOV     RESULT, AX
        POP     AX
        POP     DX
        ENDM
```

宏调用：

```
    DIFSQR   VAR1, VAR2, VAR3
```

宏展开：

```
    +   PUSH    DX
    +   PUSH    AX
    +   DIF     VAR1, VAR2
    +   MOV     AX, VAR1
    +   SUB     AX, VAR2
    +   IMUL    AX
    +   MOV     VAR3, AX
    +   POP     AX
    +   POP     DX
```

在汇编后形成的 LST 清单中，可以看到上述宏展开的结果。但是，其中的 DIF VAR1, VAR2 语句是为用户方便而提供的，它并不占有存储单元，也就是说，在 OBJ 文件中，这一语句是不存在的。

例 1.13 宏定义体内不仅可以使用宏调用，也可以包含宏定义。

宏定义：

```
DEFMAC  MACRO   MACNAM,   OPERATOR
        MACNAM  MACRO     X, Y, Z
                PUSH      AX
                MOV       AX, X
                OPERATOR  AX, Y
                MOV       Z, AX
                POP       AX
                ENDM
        ENDM
```

其中 MACNAM 是内层的宏定义名，但又是外层宏定义的哑元，所以当调用 DEFMAC 时，就形成一个宏定义。

宏调用：

```
    DEFMAC   ADDITION, ADD
```

宏展开：

```
    +  ADDITION   MACRO   X, Y, Z
                  PUSH    AX
                  MOV     AX, X
                  ADD     AX, Y
```