第6章 汇编语言高级编程

6.1 宏汇编

6.2 结构和记录

6.3 32位字长编程

6.4 汇编语言混合编程

✓宏类似于C语言中的预处理，在编译时进行宏展开。

✓条件汇编能避免使用过多的标号，使程序的结构更清晰。

✓C与汇编语言的联合编程以及W32下的汇编编程，将使汇编语言和高级语言的优点充分结合起来。

6.1 宏汇编

- ✓宏或称宏指令是源程序中的具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，所以使用宏可以加快编程速度和查错效率，从而为程序设计提供极具特色的模块化程序设计工具和手段，使汇编语言源程序同高级语言程序一样清晰、简洁，有利于阅读、修改和调试。

6.1.1 宏定义

<宏名> MACRO [<形式参数表>]

<语句1>

┆ 宏体

<语句n>

ENDM

说明：

①宏名即宏指令名是用户给某段指令定义的符号名，调用时可用该符号名来调用宏。

②宏名不能互相重复且使用宏汇编语言中的合法符号，并且允许与源程序中的其他变量、标号、指令、伪指令名相同。此时，宏名具有更高的优先级。

③MACRO和ENDM是一对伪操作符号，MACRO标识宏定义的开始，ENDM标识宏定义的结束。MACRO和ENDM之间的语句组称为宏体，宏体中允许有伪指令、注释语句“；”。

④形式参数表是可选项，因此宏可以不带参数；带参数时，参数表中的各形式参数（或称形参、哑元）用逗号分隔。

例如：CRLF MACRO

```
MOV AH,2
MOV DL,0AH    ；0AH是回车ASCII码
INT 21H
MOV DL,0DH    ；0DH是换行ASCII码
INT 21H
ENDM
```

该宏功能是实现回车换行。

6.1.2 宏调用与宏展开

宏调用的格式为：

<宏名> [<实参表>]

说明：

①宏名为已定义过的宏，即必须先定义、后调用。

②实参表中的参数将一一对应替换宏定义中形参表中的参数。两处的参数个数可以不同。若实参个数多于形式参数个数，则多余的实参忽略；若实参个数少于形式参数个数，则多余的形式参数用零或空格代替。

③汇编程序在汇编源程序时，若遇到宏调用，则用调用所提供的实参数替代相应的形参数，并把宏体中的指令嵌入到源程序中。这种嵌入操作称为宏展开。

例题6.1 应用宏功能，实现字符串的屏幕输出。

1.宏定义

```
PROMPT MACRO MESSAGE
    MOV AH,09H
    LEA DX,MESSAGE
    INT 21H
    ENDM
```

2.宏调用

```
DATA    SEGMENT
STRING1  DB “STRING  OUTPUT ！ $”
DATA    ENDS
CODE    SEGMENT
    ASSUME CS：CODE，DS：DATA
START：  MOV  AX，DATA
        MOV  DS，AX
        PROMPT STRING1   ；宏调用
        MOV AH, 4CH
        INT 21H
 CODE    ENDS
         END  START
```

3.宏展开

经汇编宏展开后，
代码段指令如下（+表示为宏展开后的语句）：

```
CODE    SEGMENT
        ASSUME  CS：CODE，DS：DATA
START：  MOV   AX，DATA
        MOV   DS，AX
+       mov ah, 09h；宏调用Prompt String1的宏展开
+       lea dx, String1
+       int  21h
        MOV AH, 4CH
        INT 21H
CODE    ENDS
        END  START
```

6.1.3 宏定义中参数使用

宏定义和宏调用中参数及参数间的代换应注意以下若干规定：

（1）宏定义中可以不带任何形式参数，在宏调用时，将用宏中整个宏体的全部指令嵌入到宏调用处的宏指令位置。

（2）宏定义中可以使用连接运算符“&”，以实现字符串的连接。“&”运算符只能出现在宏定义中，见下例

例题6.2 某宏定义如下：

```
ADL  MACRO  P1，P2
JMP   TA&P1      ；“&”出现在指令的操作数中
MOV   TB&P1, P2   ；同上
ENDM
```

当有宏调用时：

```
ADL   [BX+SI]，AX
```

则有宏展开：

```
+ JMP   TA[BX+SI]   ；宏展开时&不存在
+ MOV   TB[BX+SI]，AX
```

（3）在宏定义中，形式参数可以出现在宏体中任何位置，因此也可以出现在操作码位置，参见下例。

例题6.3 某宏定义如下：

```
NEWINST  MACRO  P1
        P1  AX
        ENDM
```

当有宏调用时：

```
NEWINST  INC
```

则有宏展开为：

```
  + INC  AX
```

（4）当形参出现在宏体的字符串中时，参数依然可以代替，参见下例。

例题6.4　某宏定义如下：

```
MSTRING  MACRO  LAB，NUM，XYZ
LAB&NUM  DB  “HELLO MY.&XYZ”
         ENDM
```

当有宏调用时有

```
MSTRING  MSG，1，TXT
```

则有宏展开为：

```
+ MSG1  DB  “HELLO MY.TXT”
```

（5）在宏调用中，有时实参是带间隔符（如空格、逗号等）的字符串，这样的实参必须用尖括号括起来，以避免混淆，参见下例。

例题6.5 在程序设计中，经常定义堆栈段，而且定义的语句基本相同，仅是堆栈段的大小和初值有所差别。为此，可先定义一个如下宏：

```
MSTACK  MACRO  XYZ
      STACK  SEGMENT  STACK
       DB  XYZ
 STACK  ENDS
       ENDM
```

当有宏调用时：MSTACK <200 DUP（0）>

则有宏展开为：

```
  +  STACK  SEGMENT  STACK
  +  DB  200 DUP (0)
  +  STACK  ENDS
```

（6）数字参数

在某些情况下，需要以实参符号的值而不是符号本身来替换形参，这种参数的替换称为数字参数的替换。特殊宏操作符%用来将其后的表达式(通常是符号常数)转换成它所代表的数值，并将此数值的ASCII码字符嵌入到宏展开中。

例题6.6 某宏定义如下：

```
DATA1  MACRO  A，B，C，D
       DW  A，B，C
       DB  D  DUP(0)
       ENDM
```

当有宏调用时：

X = 10

Y = 20

```
DATAI %X+2，5，%X+Y，%Y－5
DATA1 X+2，5，X+Y，Y-5
```

则有宏展开为：

```
+  DW 12，5，30
+  DB 15 DUP(0)
+  DW X+2，5，X+Y
+  DB Y-5 DUP(0)
```

比较这两个宏调用语句的展开结果，就可以明显看出数字参数与一般实参的区别。

注意：%后的符号一定是直接用EQU或等号“=”赋值的符号常量，或者汇编时能计算出值的表达式，而不能是变量名和寄存器名。

6.1.4 宏定义中标号和变量处理

宏定义中允许使用标号和变量，但是由于一条宏指令每展开一次，将插入一组相同的指令或伪指令，多次宏调用经宏展开后就会出现相同标号或变量的多重定义，汇编时就会出错。解决此问题的方法是采用伪指令LOCAL将宏定义中出现的各个标号或变量作为形参，并安排在该伪指令的形参表中，其中各个标号或变量用逗号分隔。汇编程序遇到LOCAL伪指令时，将以实参??0000、??0001、…、??FFFF替代形参表中的各个标号或变量，从而避免标号或变量的重名。

其格式为：

LOCAL <形式参数表>

说明：如使用LOCAL伪指令，则该伪指令必须是宏定义中的第一条指令。

例题6.7 定义取绝对值的宏指令。

```
ABS   MACRO  OPS
      LOCAL  NEXT    ；定义NEXT为形参
      CMP  OPS，0
      JGE  NEXT
      NEG  OPS
NEXT：  MOV  AX，OPS    ；若无LOCAL说明，宏展开后将出现标号重名
      ENDM
```

当有宏调用：

```
ABS  CX
MOV  BX，AX
ABS  DX
```

经宏展开后的指令为：

```
+     CMP  CX，O
+     JGE  ？？0000
+     NEG  CX
+??0000：MOV  AX，CX
        MOV  BX，AX
+     CMP  DX，O
+     JGE  ？？0001
+     NEG  DX
+??0001：  MOV  AX，DX
```

6.1.5 取消宏定义伪指令PURGE

宏名可以与指令名或伪指令名相同，且宏名的优先级别高，此时与其同名的指令和伪指令将失去作用。为了恢复指令助记符和伪指令助记符的功能，就必须将宏取消，可用PURGE伪指令实现此功能。其格式为：

PURGE <宏名> [, <宏名>…]

作用：取消已经定义的宏，选用任选项时可同时取消多个宏。

例题6.8 取消例6.4和例6.7定义的宏指令

PURGE MSTRING，ABS

6.1.6 条件汇编

汇编的编译程序在对汇编源程序代码进行编译时，能够根据条件把一段源程序包括在汇编源程序内或把它排除在外，这就是**条件伪操作**，即**条件汇编**。

条件汇编伪指令的格式为：

```
IF <条件>
 <语句块1>
[ELSE
 <语句块2>]
  ENDIF
```

说明：**IF和ENDIF必须成对出现，ELSE为可选项**。汇编程序检测IF伪指令给定的条件，如果条件为真则汇编语句块1，否则汇编语句块2。条件汇编伪指令**可出现在源程序的任意位置上，但主要用于宏指令中，并允许任意次嵌套。**

例题6.9 设AL中存放了一个字母的ASCII码，下面的条件汇编决定程序是否将AL中的字母转换为大写字母还是小写字母。

```
CHANGE  DB  1
IF  CHANGE
OR  AL, 20H
ELSE
AND AL, 0DFH
ENDIF
```

上例中，汇编程序根据CHANGE的值是否为1，决定汇编哪一条指令。如果CHANGE等于1，条件为真，汇编OR指令，把AL中的字母转换为小写；否则汇编AND指令，把AL中的字母转换为大写。

例题6.10 已知两个数X、Y，若X=Y，则计算S=X+Y，否则，计算S=X-Y。

宏定义如下：

```
XY    MACRO A，B
IF    A-B          ；A-B不为0
      MOV  AX，A
      SUB  AX，B
ELSE               ；A-B为0
      MOV  AX，A
      ADD  AX，B
ENDIF
ENDM
```

程序代码如下：

```
DATA    SEGMENT
X       DW 4
Y       DW 5
S       DW ?
DATA    ENDS
CODE    SEGMENT
        ASSUME  DS：DATA，CS：CODE
START：  MOV  AX，DATA
        MOV  DS，AX
        XY   X，Y
        MOV  S，AX
        MOV  AH，4CH
INT  21H
CODE    ENDS
        END  START
```

6.1.7 宏库的使用

用汇编语言设计程序时，在不同的源程序中经常用到许多功能相同的程序块，可以把这些通用的程序块定义成宏，放在同一文件中，就形成了一个宏库。以后在设计汇编语言程序时，如果需要其中的某一个或某几个宏，则在源程序中使用INCLUDE伪指令指定所需的宏库，这样在程序中就可以直接调用宏库中的宏了。所以建立宏库，可以方便源程序的编写，提高工作效率。

1．宏库的建立

用任何一个文本编辑软件将所定义的宏组织到一个扩展名为.LIB的文件中。例如MY.LIB，该文件就构成了一个宏库。

宏库是实用的公用文件，所以在定义宏库中的宏时应遵循以下原则：

（1）宏尽量具有通用性；

（2）宏定义中的标号必须用LOCAL伪指令说明；

（3）要对宏中使用的每一个寄存器进行保护；

（4）附有必要的使用说明；

（5）宏库文件是文本文件，其扩展名无严格限制，可以由用户定义，例如.MAC、.ABC等等。

2. 宏库的使用

使用宏库只需在源程序中用INCLUDE伪指令把宏库打开（包含）即可。INCLUDE伪指令的使用格式如下：

INCLUDE <宏库文件名>

说明：INCLUDE伪指令必须在调用宏库中的宏指令之前，在源程序中使用一次INMY.LIB，然后在汇编源程序中调用宏库中的宏。

例题6.11 把清屏和在屏幕上显示字符的功能模块定义成宏，并形成一个宏库文件MY.LIB，然后在汇编源程序中调用宏库中的宏。

首先，建立宏库，设宏库的文件名为MY.LIB，分别定义CLS和PRINT两条宏指令，并用文本编辑软件将它插入到宏库MY.LIB中。这两条宏指令的代码如下：

```
；宏库MY.LIB中的宏PRINT
PRINT  MACRO  A，B
    MOV   AH，0AH
    MOV   AL，A
    MOV   CX，B
    MOV   BH，O
    MOV   BL，OFH
    INT   10H
    ENDM
    ；宏库MY.LIB中的宏CLS
CLS   MACRO
    MOV   AX，060OH
    MOV   CX，0
    MOV   DX，1849H
    MOV   BH，7
    INT   10H
    ENDM
```

接下来就可调用宏库MY.LIB中的这两条宏指令了，使屏幕上显示10个1。

其汇编源程序如下：

```
INCLUDE MY.LIB
CODE   SEGMENT
       ASSUME  CS: CODE
START:  CLS
       PRINT   31H, 10
       MOV    AX, 4C00H
       INT    21H
CODE   ENDS
       END    START
```

6.1.8 宏与子程序的比较

通过前面宏指令和子程序的学习，可知二者均可用来处理程序中重复使用的程序段，缩短源程序代码的长度，使源程序结构简洁、清晰，但它们是两个完全不同的概念，有着本质上的区别。

1. 处理的时间不同。
2. 处理的方式不同。
3. 目标程序的长度不同。
4. 执行速度不同。
5. 参数传递的方式不同。

宏指令扩展后占用的内存空间大，但执行速度快，在多参数时较子程序调用更为方便有效，但如果宏体较长且功能独立，采用子程序又比宏指令更能节省存储空间。

6.2 结构与记录

6.2.1 结构

结构是将逻辑上互相关联的一组数据，以某种形式组合在一起，使之成为一个整体，并可单独访问其中的某个数据元素，以便进行数据处理的一种数据组织形式。

1. 结构的定义

结构定义格式为：

<结构名> STRUC

<字段说明>

<结构名> ENDS

说明：结构名是用户对结构的命名，它必须是唯一的。字段说明由DB、DW、DD等伪操作给出任意数目的字段说明，字段名在程序中必须是惟一的，它表示结构开始到相应字段的偏移量。

例题6.12 定义一个学生成绩结构：

```
STUDENT    STRUC
  NUMBER   DB ?
  NAME     DB “ABCDEF”
  MATHS    DB ?
  ENGLISH  DB ?
 STRUDENT   ENDS
```

2. 结构变量的定义及初始化

定义一个结构只是通知汇编程序构造一种新的数据类型，并不分配存储单元，只有定义了结构类型变量后，才能分配实际的存储单元，同时可将具有该结构类型的数据存入结构变量存储单元中。

定义结构变量的格式：

<结构变量名> <结构名> <<字段值表>>

例题6.13 定义2个学生成绩结构变量。

S1 STUDENT <01, ‘WANG’，90，85>
S2 STUDENT <02, ‘ZHAO’，100，70>

结构变量S1和S2可供程序直接引用，每一个变量占9B，该值由结构STUDENT决定，字段表中的4个字段值对应结构类型中定义的4个字段，并用逗号分隔。该表需用“< >”括起来。

3．结构变量中字段的访问

结构变量中字段的访问形式：

<结构变量名>.<字段名>

注意：结构变量名和字段名之间的“.”号不可省，该形式可以作为指令中的操作数使用。

例题6.14:

将两个复数X和Y相加，和存入复数变量Z中，复数定义为结构类型。

```
DATA    SEGMENT
COMP    STRUC
 RE     DW 0
 IM     DW 0
COMP    ENDS
X       COMP <10, 20>
Y       COMP <30, 40>
Z       COMP <0, 0>
DATA    ENDS
CODE    SEGMENT
    ASSUME  CS: CODE, DS: DATA
START:  MOV  AX, DATA
    MOV  DS, AX
    MOV  BX, OFFSET Z
    MOV  AX, X.RE
    ADD  AX, Y.RE
    MOV  [BX].RE, AX
    MOV  AX, X.IM
    ADD  AX, Y.IM
    MOV  [BX].IM, AX
    MOV  AX，4C00H
    INT  21H
CODE ENDS
END  START
```

例题6.15 采用结构伪指令编写计算N！的程序。

```
DATA    SEGMENT
N       DW 3
RESULT  DW ?
DATA    ENDS
STACK   SEGMENT
        DW  128 DUP （0）
TOS     LABEL  WORD
STACK   ENDS
CODE1   SEGMENT
MAIN    PROC  FAR
        ASSUME CS：CODE1，DS：
DATA，SS：STACK
START：  PUSH  DS
        SUB  AX，AX
        PUSH  AX
        MOV  AX，DATA
        MOV  DS，AX
        MOV  AX，STACK
        MOV  SS，AX
        MOV  SP，OFFSET TOS
        MOV  BX，OFFSET RESULT
        PUSH BX
        MOV  BX，N
        PUSH BX
        CALL FAR PTR FACT
        ；RET
        MOV AH，4CH
        INT 21H
MAIN    ENDP
```

```
CODE1   ENDS
CODE2   SEGMENT
PAGE1   STRUC ；定义结构
PAGE1
        SAVEBP    DW ?
        SAVECSIP  DW 2 DUP（? ）
        NN        DW ?
        ARESULT   DW ?
PAGE1   ENDS
        ASSUME CS：CODE2
FACT    PROC  FAR
        PUSH  BP
        MOV  BP，SP
        PUSH  BX
        PUSH  AX
        MOV   BX，[BP].ARESULT
        MOV   AX，[BP].NN
        CMP   AX，0
        JE  DONE
        PUSH  BX
        DEC   AX
        PUSH  AX
        CALL  FAR PTR FACT
        MOV   BX，[BP].ARESULT
        MOV   AX，[BX]
        MUL   [BP].NN
        JMP   SHORT EXIT
DONE：  MOV   AX，1
EXIT：  MOV   [BX]，AX
        POP  AX
        POP  BX
        POP  BP
        RET  4
FACT    ENDP
CODE2   ENDS
        END  START
```

6.2.2 记录

记录与结构相似，记录是以二进制数位为基本单位组成的字段，使用记录类型可将若干二进制位信息紧凑地存放在一个字节或字中，并可对这些信息按位处理。记录类型能有效地节省存储空间。

1．记录的定义

记录定义的格式：

<记录名> RECORD <字段名>：<宽度> [= <表达式>]

[，<字段名>：<宽度>][= <表达式> ……]

说明：在格式中，记录名和字段名不能省略。字段的宽度是相应字段占有的二进制位数，且所有的字段宽度之和不能大于16。如果所有字段的宽度之和大于8位，该记录按字处理，否则按字节处理。可在记录定义时，用表达式给字段名赋初值。

例题6.16 将职工基本情况定义为一个记录类型。

WORKER RECORD NO: 8, SEX: 1, MARRY: 1, YEAR: 5, HEL:1

记录中：职工编号NO占8位，性别SEX占1位（0表示男，1表示女），婚姻状态MARRY占1位（0表示已婚，1表示未婚），工龄YEAR占5位，健康状况HEL占1位（0表示健康，1表示不健康）。

2. 记录变量的定义及初始化

与定义结构类型相似，定义一个记录也只是通知汇编程序构造一种新的数据类型，并不分配存储单元，而只有定义了记录变量，才能分配实际的存储单元并给其赋值。

定义记录变量并赋值的语句格式如下：

<记录变量名> <记录名> < <字段值表> >

说明：记录变量名是记录变量的符号地址，该记录变量具有记录名指定的记录类型。字段值表中的值是赋给记录变量相应字段的初值，各字段值用逗号分离，且排列顺序与记录的定义一致。对不需要赋值的字段，数值可不写，但逗号不能省略。字段值表必须用尖括号括起来。

例题6.17：

用职工基本情况记录类型定义一个职工情况变量P1。其中编号1的职工为女职工，已婚，工龄为12年，健康。

P1 WORKER <00000001B，1B，0B，01011B，0B>

3．记录运算符

1）记录宽度运算符WIDTH

格式：　WIDTH <记录名> | <记录字段名>

功能：返回记录长度或记录字段在记录中所占的二进制位数。

例题6.18

```
MOV  AX，WIDTH WORKER　；把职工基本情况记录的　长度16送到AX
MOV  DL，WIDTH SEX　　；把性别字段所占的宽度1送到DL
```

2）记录屏蔽运算符MASK

格式：MASK <记录字段名>

功能：返回一个8位或16位的二进制数。在这个数中，记录字段名所指定的字段的对应位为1，其他位为0。

例题6.19

```
MOV  AX，MASK  SEX
        ；把0000000010000000B送到AX
```

4．对记录中字段的访问

已定义的记录中的字段名可以作为常数直接被引用，值为该字段中最低位在记录单元中的位置。

例题6.20 定义50个职工的记录变量，通过输入数据得到职工情况信息，然后统计工龄满20年的女职工人数。

```
DATA    SEGMENT
WORKER  RECORD  NO: 8, SEX: 1, MARRY: 1, YEAR: 5, HEL:1
ARRAY   WORKER  50  DUP  （< 00000001b,1b,1b,11010b,0b>）
COUNT   DW  0
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS：CODE，DS：DATA
START：  PUSH  DS
        MOV  AX，O
        PUSH  AX
        MOV  AX，DATA
        MOV  DS，AX
        MOV  CH，50    ；计数器
        MOV  BX，OFFSET ARRAY
NEXT：   MOV  AX，[BX]
        TEST AX，MASK SEX
        JZ  NEXT1
    ；是男性，转移
        MOV  CL，YEAR
  ；CL←1，记录字段名可直接引用，返回该
  ；字段移到记录最右边所需的移位次数
        MOV  DX，MASK YEAR
  ；DX←0000000000111110
        AND  AX，DX
  ；AX←取出工龄字段内容
        SHR  AX，CL    ；AX右移1位
        CMP  AL，20    ；工龄≥20吗？
        JL  NEXT1
        INC  COUNT
  ；工龄≥20则存放统计数加1
NEXT1：  ADD  BX，2
        DEC  CH        ；计数器减1
       JNE  NEXT
        MOV  AH, 4CH
        INT  21H
CODE    ENDS
        END  START
```

6.3 32位字长编程

32位字长编程实质是在386及其后继机型中的编程，是首先使用伪指令选择处理器型号，然后使用32位寄存器来存储数据的编程。

32位字长编程并不等同于Win32下的汇编编程。Win32汇编编程是在Windows操作系统下，调用的是Windows 系统的API函数，是保护模式下的编程。

32位字长编程是指在DOS操作系统下，应用的是386以上系统指令，采取的是实模式或虚拟8086模式编程，调用的是系统中断服务，发挥的是系统32位字长的优势。

6.3.1处理器选择伪指令

由于80x86的所有处理器都支持8086/8088指令系统，但每一种高档机型又都增加一些新的指令，因此在编写程序时要对所用处理器有一个确定的选择。也就是说，要告诉汇编程序应该选择哪一种指令系统。

处理器选择伪指令一般放在整个程序的最前面。如不给出，则汇编程序默认为处理器型号为.8086。当然它们也可以放在程序中间，如程序中使用了一条80486所增加的指令，则可在该指令的上一行加上.486。

6.3.2简化伪指令

80x86汇编语言从MASM5.0开始提供简化段定义伪指令，主要用于实现汇编代码与高级语言代码之间的连接。定义当前段的简化段定义伪指令可作为前一段的结束，而且还隐含使用ASSUME伪指令。

1．简化段定义伪指令

在前面的伪指令中已经讲过，段定义采用的格式为：

<段名> SEGMENT [<定位方式>] [<定位类型>] ['<类别>']

<段名> ENDS

这种格式称为标准段模式，但在32位字长编程中，经常使用简化段定义伪指令来定义标准段。对应代码段、数据段、堆栈段的简化定义如下：

1）堆栈段定义伪指令

格式：.STACK [<长度>]

作用：定义一个堆栈段。

说明：长度的默认值为1KB，隐含段名为@STACK，定位方式为PARA，定位类型为STACK，类别为‘STACK’。

2）代码段定义伪指令

格式：.CODE [<名字>]

作用：定义代码段。

说明：隐含段名为@CODE，定位方式为WORD，定位类型为PUBLIC，类别为‘CODE’。若需定义多个代码段时，可用不同的名字区分。

3）数据段定义伪指令

格式：.DATA | .FARDATA |.CONST

作用：3种定义数据段格式的作用是不同的。

.DATA定义一个NEAR类型的数据段，隐含段名为@DATA，定位方式为WORD，定位类型为PUBLIC，类别为‘DATA’。

.FARDATA定义一个FAR类型的数据段。而用.CONST格式，则是定义一个常数数据段。

2. 内存模式选择伪指令

格式：.MODEL<模式选择符> [，<高级语言>]

作用：指明简化段所使用的内存模式，指示数据和代码允许使用的长度。

说明：模式选择符有TINY、SMALL、MEDIUM、COMPACT、LARGE、HUGE、FLAT等。

6.3.3编程实例

例题6.21 有两个4字长数据分别存放在Buf1和Buf2存储单元中，请用8086指令系统编写汇编程序求出它们的和，并将结果存放在Buf3中。

为了得到4字长数据之和，在8086处理器中需要分4段分别计算，每段一个字长（16位），用4次循环可得到4字长数据之和。考虑到每次求和可能有进位值，要用ADC（而不是ADD）指令求和，而且在进入循环体之前应先清除CF位。在循环中修改地址指针时用INC指令而不用ADD指令，以免影响求和时得到的进位值。

①8086指令编码：

```
STACK  SEGMENT
    DB  200 DUP(0)
STACK  ENDS
DATA  SEGMENT
BUF1  DQ  1234567890ACDED2H
BUF2  DQ  17FEDCA937543219H
BUF3  DQ  ?
DATA  ENDS
CODE  SEGMENT
      ASSUME CS:CODE, DS:DATAT,
SS:STACK
START: MOV AX, DATA
      MOV DS, AX
      CLC
      LEA  SI,  BUF1
      LEA  DI,  BUF2
LEA  BX,  BUF3
      MOV  CX,  4
NEXT:   MOV  AX,  [SI]
      ADC  AX,  [DI]
      MOV  [BX], AX
      INC SI
      INC SI
      INC DI
      INC DI
      INC BX
      INC BX
LOOP NEXT
     MOV AX,4C00H
    INT  21H
CODE  ENDS
         END  START
```

分析：把INC改为ADD指令则如何操作？

②80386指令编码：

```
    .MODEL SMALL
    .386
    .STACK 200H
.DATA
BUF1  DQ  1234567890ACDED2H
BUF2  DQ  17FEDCA937543219H
BUF3  DQ  ?
.CODE
START: MOV AX, @DATA
       MOV DS, AX
       MOV EAX, DWORD PTR
BUF1
     ADD EAX, DWORD PTR BUF2
    MOV EDX, DWORD PTR
BUF1+4
    ADC EDX, DWORD PTR BUF2+4
    MOV DWORD PTR BUF3, EAX
    MOV DWORD PTR BUF3+4, EDX
    MOV AX, 4C00H
    INT  21H
    END  START
```

经过测试执行上述程序所需要的时钟周期数可知：在80386及其以上机型运行上述程序，用32位字长编码计算比用16位字长编码计算快5-7倍。所以利用高档机的32位字长编程可以提高程序的执行效率。这里只是以加法运算为例，实际上，对于其他指令也有类似的效果，对于如乘/除法等更复杂的指令，收到的效果会更好。

6.4 汇编语言的混合编程

汇编语言距硬件最近，受硬件的限制移植性较差。高级语言相对汇编语言开发效率高、可移植性好，所以高级语言比汇编语言使用范围更加广泛。但在要求执行速度快、占用空间少、可直接控制硬件时，仍然用汇编语言编程。对于一个具体的任务，编程者经常使用高级语言和汇编语言混合编程，同时发挥二者的程序设计优势。一般情况下是程序的主体部分用高级语言编写，以便缩短开发周期，而程序的关键部分及高级语言不能完成的部分用汇编语言编写。

混合编程的关键是两种语言的接口问题。解决方法有在高级语言程序中直接嵌入汇编代码，或者高级语言主程序调用汇编子程序，汇编语言也可调用高级语言函数。常用方法是高级语言主程序调用汇编子程序。

6.4.1 直接嵌入方式

在C语言程序中直接嵌入汇编语句，但在汇编语句前应用关键字_ ASM说明。

语句格式：_ASM 汇编语句

内嵌汇编语句的操作码必须是有效的80x86指令。不能使用BYTE，WORD，DWORD等语句定义数据，可以使用OFFSET，TYPE，SIZE，LENGTH等属性操作符。

对于连续多个汇编语句，可采用如下形式：

```
_ASM {
        汇编语句
        汇编语句
         ……
      }
```

例题6.22 C程序中内嵌汇编语句举例，文件名E622.C

```
#include "stdio.h"
#pragma warning (disable:4101)
struct Misc
{
        char misc1;
        short misc2;
        int  misc3;
        long misc4;
};
char mychar;
short myshort=-5;
int main()
{
        int myint =20;
        long mylong;
        _int64 mylonglong;
        int mylongarray[10];
        struct Misc mymisc;
        _asm mov mychar,'9'
        _asm {
                mov eax,LENGTH myint
                mov eax,TYPE myint
                mov eax,SIZE myint
                mov eax,LENGTH
mylongarray
mov eax,TYPE mylongarray
                mov eax,SIZE
mylongarray
                mov eax,LENGTH
mymisc
                mov eax,TYPE mymisc
                mov eax,SIZE mymisc
        mov eax,TYPE mychar
                mov eax,TYPE myshort
                mov eax,TYPE mylong
                mov eax,TYPE
mylonglong
                add myint,30
                mov ax,myshort
                mov mymisc.misc2,ax
                movsx eax,mymisc.misc2
                lea ebx,mymisc
                mov [ebx].misc3,eax
                mov
mylongarray[2*4],200
        }
        printf("mychar=%c myint=%d
mymisc.misc3=%d mylongarray[2]=%d\n",

mychar,myint,mymisc.misc3,mylongarray[2]);
        return 0;
}
```

执行结果为：

```
mychar=9 myint=50 mymisc.misc3=-5
mylongarray[2]=200
Press any key to continue
```

6.4.2 C调用汇编子程序

在进行程序设计时，常采用C语言编写主程序，汇编语言编写子程序，然后C主程序调用汇编子程序的方法进行模块化设计，要求C源程序中的所有语句必须符合C的语法规则，汇编源程序中的所有语句要符合汇编的语法规则。与直接嵌入汇编语句相比，这种方法实现的功能更多，还可以绕开嵌入汇编的限制。

6.4.3 汇编调用C函数

在C和汇编联合编程时，一般采用C模块调用汇编模块子程序的方法，但是汇编模块也可作为主程序调用C模块中的函数。从汇编模块的角度看，这种方法与调用C库函数及Windows API没有太大区别，只是在汇编模块中，使用PROTO声明C函数的名称、调用方式、参数类型，使用INVOKE调用C函数。

例题6.24 汇编主程序调用C子程序举例

1. 汇编主程序模块代码，文件为E624_1.ASM

```
 .386
  .MODEL FLAT
INPUT  PROTO C PX:PTR SDWORD,PY:PTR
   SDWORD,PZ:PTR  SDWORD
VERIFY PROTO C X:DWORD,Y:DWORD,Z:DWORD
OUTPUT PROTO C
   X:DWORD,Y:DWORD,Z:DWORD,RESULT:DWORD
.DATA
    X  DWORD ?
    Y  DWORD ?
    Z  DWORD ?
```

```
RESULT DWORD ?
.CODE
MAIN PROC C
   INVOKE INPUT,OFFSET X,OFFSET Y,OFFSET Z
   INVOKE VERIFY,X,Y,Z
   MOV RESULT,EAX
   INVOKE OUTPUT,X,Y,Z,RESULT
   RET
MAIN ENDP
    END
```

2．C子程序模块代码，文件名E624_2.C

```
#include "stdio.h"
extern void input(int *px,int *py,int *pz);
extern int verify(int x,int y,int z);
extern void output(int x,int y,int z,int result);
void input(int *px,int *py,int *pz)
 {
     printf("input x,y,z:");
     scanf("%d,%d,%d",px,py,pz);
     printf("x=%d,y=%d,z=%d\n",*px,*py,*pz);
  }
int verify(int x,int y,int z)
  {
    if(x*x+y*y==z*z)  return 1;
     else   return 0;
  }
void output(int x,int y,int z,int result)
 {
    printf("verify(%d,%d,%d)=%d\n",x,y,z,result);
  }
```

3．编译链接过程

```
ML /C /COFF E624_1.ASM
CL /C E624_2.C
Link E624_1.obj E624_2.obj /out:E624.exe
/subsystem:console
```

4．汇编模块的执行过程相当于：

```
input(&x,&y,&z);
result=verify(x,y,z);
output(x,y,z,result);
```

6.4.4 C++与汇编

在C++与汇编的混合编程中，对汇编模块没有特殊要求。对于C++而言，则应该将与汇编模块共享的变量以及函数等用EXTERN “C”进行说明。常用方法同样是C++语言编写主程序，汇编语言编写子程序，C++主程序调用汇编子程序。

例题6.25 用多种方法求数组元素的和

1．C＋＋主程序模块代码，文件名E625_1.CPP

```
#include <stdlib.h>
#include <memory.h>
#include <iostream.h>
extern "C" int _cdecl arraysum2(int array[],int count);
extern "C" int _stdcall arraysum3(int array[],int count);
extern "C" int initvals[];
int arraysum(int array[],int count)
{
    int sum=0;
    for(int i=0;i<count;i++)
        sum+=array[i];
    return sum;
}
#define array_size 5
int main()
{
    int array[array_size];
    memcpy(array,initvals,array_size*sizeof(int));
    for(int i=0;i<array_size;i++)
    {
        cout<<"array["<<i<<"]="<<array[i];
        if(i<array_size-1)
            cout<<",";
    }
```

```
cout<<"\n";
    cout<<"the sum is"<<arraysum(array,array_size)<<".\n";
    cout<<"the sum is"<<arraysum2(array,array_size)<<".\n";
    cout<<"the sum is"<<arraysum3(array,array_size)<<".\n";
    return 0;
}
```

2．汇编子程序模块代码，文件名E625_2.ASM

```
.386
.model flat
public _initvals
.data
_initvals dword -2,-1,0,10,20
.code
arraysum2  proc  C array:ptr sdword,count:sdword
local sum:dword
push esi
mov sum,0
mov esi,array
mov ecx,count
cld
addelement:lodsd
add sum,eax
loop addelement
mov eax,sum
pop esi
ret
arraysum2  endp
```

```
arraysum3  proc  stdcall array:ptr sdword,count:sdword
local sum:dword
push esi
mov sum,0
mov esi,array
mov ecx,count
cld
addelement:lodsd
add sum,eax
loop addelement
mov eax,sum
pop esi
ret
arraysum3  endp
end
```

3．编译链接过程

```
ML /C /COFF E625_2.ASM
CL /C E625_1.C
Link E625_1.obj E625_2.obj /out:E625.exe /subsystem:console
```

6.4.5 控制台编程

控制台程序是指那些仅仅显示文本字符的程序。控制台程序可以在DOS状态下执行，执行后自动转入Windows保护模式下运行执行。控制台程序不是DOS程序，它能够使用Windows的高级功能，是32位程序；而DOS程序不能调用Windows的函数，是16位程序。

例题6.26 在控制台输入数据125，然后显示该数据并输出字符串“Hello World”。

1．汇编程序源代码文件E626_1.ASM

```
.386
.model flat, stdcall
option casemap:none
; 说明程序中用到的库、函数原型和常量
includelib      msvcrt.lib
scanf           proto c :dword, :vararg
printf          proto c :ptr sbyte, :vararg
; 数据区
.data
szinput         byte   "%d",0
szoutput        byte   "%d",0ah,0
szmsg           byte    "Hello World!", 0ah, 0
a               sdword   ?
; 代码区
```

```
.code
start:
            invoke  scanf,offset szinput,offset a
            invoke  printf,offset szmsg
            invoke  printf,offset szoutput,a
            ret
end             start
```

2．与该程序功能相同的C程序源代码文件E626_2.C。

```
#include <stdio.h>
int main()
{
    int a;
    Scanf(“%d”,&a);
    Printf(“hello world!\n”;
    Printf(“%d\n”,a);
Return 0;
    }
```

6.4.6 Windows界面编程

Windows界面编程是在Windows环境下的程序设计，显著特点是具有友好的图形界面，在图形方式下实现用户交互。控制台编程实质也是在Windows环境下的编程，只是程序被限制在文本方式下实现用户交互。二者都能够使用Windows的高级功能，都是32位程序设计，都可以调用Windows操作系统中核心动态链接库的Win32 API函数。

例题6.27 应用Widows消息框显示“Hello World！”字符串。

1．源程序代码文件名E627_1.ASM

```
.386
.model  flat, stdcall
option  casemap:none
MessageBoxA     PROTO :dword, :dword, :dword, :dword
MessageBox      equ                 <MessageBoxA>
includelib      user32.lib
NULL            equ                 0
MB_OK                   equ                 0
.stack  4096
.data
szTitle                 byte        'Hi!', 0
szMsg           byte        'Hello World!' ,0
.code
start:
                invoke  MessageBox,
                        NULL,           ；HWND hWnd
                        offset szMsg,   ；LPCSTR lpText
                        offset szTitle, ；LPCSTR lpCaption
                        MB_OK           ；UINT uType
                ret
end             start
```

程序中使用的MessageBox是Windows核心动态链接库User32.Dll中的API函数，功能是显示消息对话框。第1个参数是窗口句柄，第2是字符串指针，指向消息框中显示的正文，第3个参数也是字符串指针，指向消息框的窗口标题，第4个参数指定消息框的类型。

2．编译链接过程：

```
Ml /coff e626.asm /link /subsystem:windows
```

3．运行结果：

4．与该程序的功能相同的VC程序源代码文件E627_2.C：

```
#include <windows.h>
char szTitle[]="Hi!";
char szMsg[]="Hello World!";
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
    // TODO: Place code here.
    MessageBox(NULL,szTitle,szMsg,MB_OK);
    return 0;
}
```