第4章　数据传送指令

罗文坚
中国科大 计算机学院

http://staff.ustc.edu.cn/~wjluo/mcps/

1

本章内容

- MOV指令回顾
- PUSH/POP指令
- 装入有效地址
- 数据串传送
- 其他数据传送指令
- 段超越前缀
- 汇编程序详述

2

Assemble Language

- 汇编语言是机器语言的符号化描述。
  - 一种面向机器的程序设计语言，通常是为特定计算机或计算机系列专门设计的。
- 利用机器指令的助记符、符号地址和标号来编写程序。
  - 基本语句是机器指令系统中的指令。

3

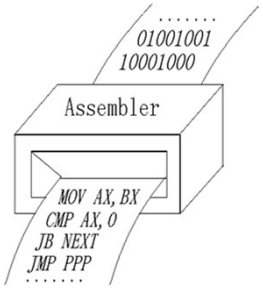
汇编语言程序设计的优点和缺点

- 优点
  - 可充分利用机器的硬件功能和结构特点，加快程序的执行速度，减少目标程序所占用的存储空间。
  - 常用来编写实时控制程序、实时通信程序，有时也用来编制某些系统软件程序。
- 缺点
  - 编程效率低（与人们描述计算过程的需要差距大）。
  - 与机器硬件的具体结构联系过于紧密。
    - 在一种结构的机器上开发的程序极难移植到另一种不同结构的机器上去。

4

Assemble Language

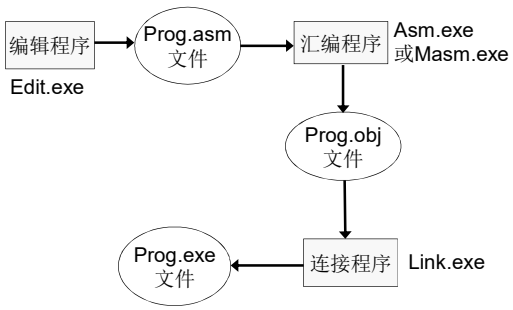
- 用汇编语言编写的程序不能由机器直接执行，而必须经汇编程序翻译成机器语言程序。
  - 汇编：汇编语言源程序由ASM.exe生成目标代码（*.obj，可能有多个）
  - 连接：由LINK.exe将.obj连接成可执行程序。



思考：什么是汇编？什么是汇编语言？

5

汇编语言程序的开发过程



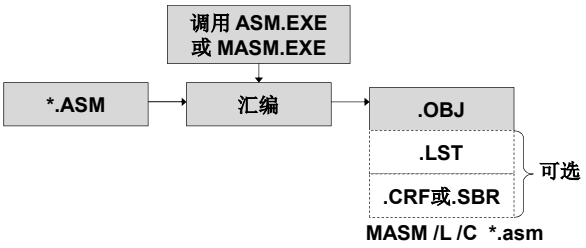
6

编译过程的目的

1. 检查源程序，测出源程序中的语法错误，并给出出错信息；
2. 展开宏指令。
3. 产生目标文件（.OBJ）。
   - 同时，也可给出：
   - ➢ 列表文件（同时列出汇编语言源程序和机器语言目标程序的文件，称之为.LST文件）；
   - ➢ 交叉索引文件（列出程序中使用的符号、变量和标号以及引用情况，称之为.CRF文件或.SBR文件）。

7

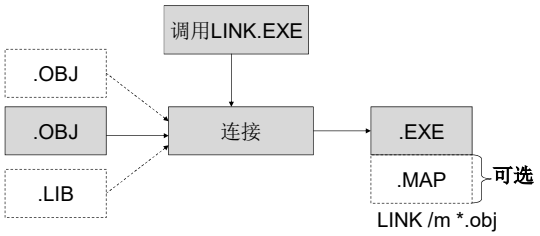
汇编示意图



- 汇编程序分两种：
  - ➢ 一种是基本汇编（ASM.EXE）。
  - ➢ 一种是宏汇编（MASM.EXE），宏汇编功能比较强。

8

连接示意图



- 汇编之后生成的OBJ文件必须经过链接过程，才能成为扩展名.EXE的可执行文件。
- 链接的过程就是调用连接程序（LINK.EXE），对OBJ文件进行定位、链接，最后生成扩展名为EXE的可执行文件。如果需要，也可生成MAP文件（地址分配文件）。

9

.lst文件

- 汇编表：源程序、目标代码、地址、错误信息
- 段（组）表：源程序中各段的名字、大小及其特征。
- 符号表：定义或引用的全部标号、变量、符号的属性值及其特征等。

```
Microsoft (R) Macro Assembler Version 5.00                  4/10/7


     1 0000                             DATA    SEGMENT 'DATA'
     2 0000  3031                       AREA1   DW  3031H
     3 = 3233                           NUM     EQU 3233H
     4 0002  24                                 DB  '$'
     5 0003                             DATA    ENDS
     6
     7 0000                             STACK   segment STACK 'STACK'
     8 0000  0032[                              db 50 DUP (?)
     9             ??
    10                        ]
    11
    12 0032                             STACK   ENDS
    13
    14 0000                             CODE    SEGMENT
```

（部分）

10

.map文件

- 地址分配文件（.map）用于描述文件中各段的浮动起始地址、结束地址、占用空间、段名和‘类别’，还列出了各模块中所定义的公共符号及其偏移地址，主要用于程序调试和资料归档。

```
 Start  Stop   Length Name                   Class
 00000H 00002H 00003H DATA                   DATA
 00010H 00041H 00032H STACK                  STACK
 00050H 0005FH 00010H CODE

  Address         Publics by Name


  Address         Publics by Value


Program entry point at 0005:0000
```

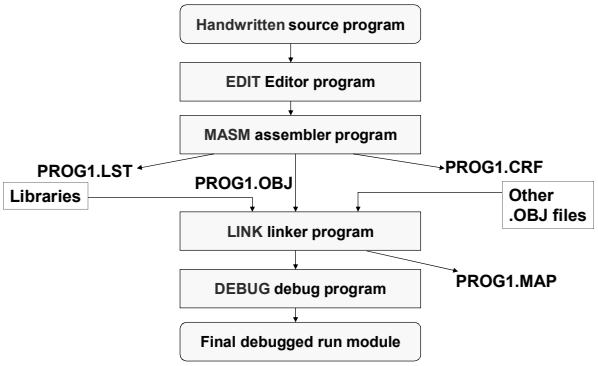
11

调试程序

- ➢ DEBUG PROG.EXE
  - – u (反汇编)
  - – t = 地址 指令条数
  - – 常用命令：http://staff.ustc.edu.cn/~wjluo/mcps/resource/dos/debug.pdf
- 集成开发环境
  - – 例如，MASM32，Visual C++

12

汇编语言程序开发中的相关文件



13

汇编语言的书写格式

- 汇编语言的书写格式有两种：
  1. 完整的段定义方式
     - 可完全控制汇编的全过程，并且可用于所有的汇编程序。
  2. 针对一种特定的汇编程序的模型
     - 简化段定义的格式

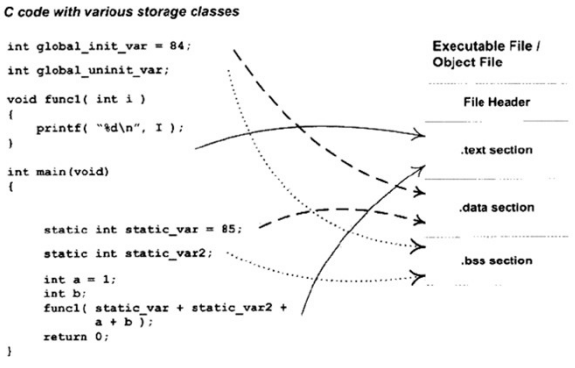
14

关于MASM

- MASM：Microsoft Macro Assembler
- MASM的版本
  - 5.0及其以前：MS-DOS；5.x~6.0：MS-DOS和OS/2
  - 6.1x：Windows，MASM 与 LINK 整合为 ML。
    - MASM.EXE和LINK.EXE，ML.EXE，ML64.EXE
  - MASM 6.15以后，微软不在单独发布MASM的开发包，而是包含在Visual Studio里面。例, …Visual Studio 9.0\VC\bin\ml.exe.
- MASM与IDE
  - Visual Studio，MASM32，WinAsm Studio，……
  - 写Windows汇编程序时用IDE比较合适。
- MASM源程序的书写格式
  - 完整段定义的格式 VS 简化段定义的格式

15

关于C程序与目标文件



16

汇编程序语法

- 常量、标识符和表达式
- 伪指令
- 存储器的组织
- 程序举例

17

常量

- 数字常量
  - 1001B，1037Q，166D（或166），6AH
- 字符常量
  - 字符：‘A’，
  - 字符串：’ abcd’
- 符号常量
  - COM_REG EQU 26H
  - COMREG=62H
    - MOV AX, COMREG等价于MOV AX, 62H

18

标识符

- 标识符是程序员在编程时建立的有特定意义的字符序列，可以用作常量名、变量名 、标号、名字（如过程名）等。

1. 组成标识符的字符：英文字母（A～Z，a～z），数字（0～9），特殊符号（？，@，_，$）等组成。
2. 数字不能作为名字的第一个符号。
3. 单独的问号（？）不能作为标识符。
4. 一个名字的最大有效长度为31，超过31的部分将不再被识别。
5. 不能用保留字，保留字包括指令和命令的助记符。
   - 如AX，MOV，SEGMENT等。

19

表达式

- 表达式由操作数和运算符组成。
  - 操作数：常量、变量、标号等。
  - 运算符：
    - 算术、逻辑、关系
    - 分析运算符：返回变量、标号等的属性。
      - offset、seg、type、length、size。
    - 合成运算符：修改变量、标号的属性。
      - PTR等。

20

表达式中的运算符

优先级从高到低

1. 括号中的项，即(…)和[…]。
2. LENGTH，SIZE，WIDTH，MASK
3. PTR，OFFSET，SEG，TYPE，THIS
4. *，/，MOD，SHL，SHR
5. ＋，－
6. EQ，NE，L，LE，GT，GE
7. NOT
8. AND
9. OR，XOR

21

例

- 设部分源程序如下。

```
DA    EQU 100
MOV AX, DA－80
MOV BX, DA MOD 10
MOV CX, DA/25
MOV DH, 01100100B SHR 2
MOV AL, DA  LT 120
MOV  AL, DA GT 120
```

- 汇编时，计算表达式形成指令如下。

```
➢ MOV AX, 20
➢ MOV BX, 0
➢ MOV CX, 4
➢ MOV DH, 19H
➢ MOV AL, 0FFH
➢ MOV AL, 00H
```

22

汇编程序语法

- 常量、标识符和表达式
- 伪指令
- 存储器的组织
- 程序举例

23

汇编语言程序的格式

- 两大特点：
  - 分段结构
  - 语句行
    - 指令性语句
    - 指示性语句

24

指令性语句和指示性语句

1. 指令性语句
   - 指令语句即由CPU提供的指令形成的语句，它能够被翻译成机器代码，并完成一定操作功能。
   - 例、LOP: ADD AL, [BX] ；把AL和[BX]相加
2. 指示性语句
   - 也叫伪指令语句，为汇编程序在翻译汇编语言源程序时提供有关信息，并不翻译成机器代码。
   - 伪指令没有对应的机器指令。
   - 例、DATA SEGMENT AT 2000H

- 注意：VC++内嵌汇编程序部分没有伪指令功能。

25

指令系统选择伪指令

- 默认情况下，汇编程序只接受8086/8088指令。如果要使用其它指令，需使用.686或.686P伪指令或其他的微处理器选择开关，放在程序前面。
  - .686伪指令：通知汇编程序按实模式使用Pentium Pro指令系统。
  - .686P伪指令：通知汇编程序使用Pentium Pro保护模式指令系统。
- 注意：多数现代软件都是假定微处理器是Pentium Pro或更新的微处理器，因此常用.686开关。

26

常用的指令系统选择伪指令

- .286
- .286P
- .386
- .386P
- .486
- .486P
- .586
- .586P
- .686 ；选择Pentium Pro~Pentium4指令系统
- .686P；选择Pentium Pro~Pentium4的保护模式指令系统
- .287
- .387

27

常用的伪指令

1) .CODE	14)DT	27)ORG
2) .DATA	15)DUP	28)QWORD
3) .EXIT	16)DWORD	29)PROC
4) .MODEL	17)END	30)PTR
5) .STACK	18)ENDM	31)QWROD
6) .STARTUP	19)ENDP	32)SEGMENT
7) ALGN n	20)ENDS	33)STACK
8) ASSUME	21)EQU	34)STRUC
9) BYTE	22)FAR	35)TITLE
10)DB	23)MACRO	36)USES
11)DW	24)NAME	37)USE16
12)DD	25)NEAR	38)USE32
13)DQ	26)OFFSET	39)WORD

28

定义程序开始和结束的伪指令

- NAME伪指令
  - 主要作用：在程序的开始可以用NAME定义模块的名字。
  - 格式：

    NAME　MODULE_NAME
  - 汇编后，MODULE_NAME就成为模块的名字。
- TITLE伪指令
  - 主要作用：指定列表文件的每一页上打印的标题。
  - 格式：

    TITLE　TEXT
  - 如果程序中没有使用NAME伪指令，也可使用TITLE伪指令指定模块名。汇编程序将用TEXT中的前6个字符作为模块名。TEXT最多可有60个字符。

29

定义程序开始和结束的伪指令

- NAME及TITLE伪指令并不是不可缺少的。
- 如果程序中既无NAME又无TITLE伪指令，则用源文件名作为模块名。
  - 此时，直接由段定义语句SEGMENT开始编写程序。
- 一般经常使用TITLE，以便在列表文件中能打印出标题来。

30

定义程序开始和结束的伪指令

- 表示源程序结束的伪指令的格式为：

  END ［LABEL］

  – 标号LABEL指示程序开始执行的起始地址。

  – 如果多个程序模块相连接，则只有主程序要使用标号，其他子程序模块则只用END而不必指定标号。

- 注意：汇编程序将在遇END时结束汇编，而程序则将从主模块的第一个标号处开始执行。

31

在存储段中存储数据

- 命令：DB、DW、DD、DQ、DT
- 对应：1字节、2字节、4字节、8字节、10字节
- 格式：

  【变量】 命令 参数1，参数2，…【；注释】

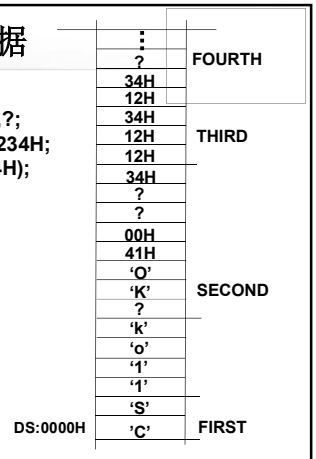
- 功能：
  – 分配变量的存储空间
  – 变量初始化

32

在存储段中存储数据

```
DATA  SEGMENT
   FIRST   DB  'CS11', 'o', 'k',?;
   SECOND  DW  'OK','A',?,1234H;
   THIRD   DB  2 DUP (12H,34H);
   FOURTH DB 100 DUP (?);
   …
   ALIGN  2
  DFLOAT   DD 300H
           DD 2.13
           DD 3.1E+12
DATA  ENDS
```

- 注意：
  - 变量的存储形式！
  - DUP的用法！



33

在存储段中存储数据

- 变量名：指针！

```
DATA  SEGMENT
   AREA1    DW  ?
   AREA2    DW  '$'
   AREA3    DW  10
DATA  ENDS
```

```
   MOV AX, AREA3
等价于
   MOV AX, [0004]
```

- 字符串定义
  – NOTES DB 'The result is:','$'
  – 除了DB定义的字符串常量外，单引号中ASCII字符的个数不得超过两个。
  – 思考：DW 'ABCD' 和 DW 0ABCDH 非法、合法？

34

ALIGN伪指令

- ALIGN伪指令的格式：

  ALIGN n

  – 其功能是控制下一个数据或指令的开始位置。
- 例，
  – ALIGN 2：按字的边界存储。
  – ALIGN 4：按双字的边界存储。
  – ALIGN 16：从可以被16整除的地址开始分配存储空间。

- ALIGN伪指令不能用于存储器模型，因为模型的大小确定了数据的对齐方式。

35

符号定义伪指令

1. 等值伪指令EQU
   – 格式：<符号名> EQU <表达式>
   – 功能：给符号名定义一个值，赋予一个符号名、表达式或助记符。
   – 例、 CONSTANT EQU 256

- 利用EQU命令可以为较复杂的表达式及源程序中的任何符号（如指令助记符、寄存器名、变量名、标号、段名、宏定义名等）定义一个替换名。
  – 在程序中，使用新替换名的作用与使用原来的表达式或符号名的作用完全等价。
  – 例、 ABC EQU [BX]
    MOV BX, ABC

36

符号定义伪指令

2. 解除定义伪指令PURGE
   - 格式：PURGE <符号1，符号2，…，符号N>
   - 功能：解除指定符号的定义；解除符号定义后，可用EQU重新进行定义。

- 例、

```
Y1      EQU  7     ；定义Y1的值为7
PURGE        Y1    ；解除Y1的定义
Y1      EQU  36    ；重新定义Y1的值为36
```

37

符号定义伪指令

3. 等号伪指令=
   - 与EQU相类似，也可以作为赋值操作使用。
   - 它们之间的区别：EQU伪指令中的表达式名不允许重复定义，而“=”伪指令则允许重复定义。

- 如下语句在程序中是允许使用的。

```
TEMP=7
TEMP=TEMP+1
```

- TEMP=6或TEMP EQU 6都可以使数6赋以符号名TEMP，然而不允许两者同时使用。

38

名字和变量

- 名字和变量
- 分析运算符
  - OFFSET，SEG，TYPE，LENGTH，SIZE
- 合成运算符
  - PTR，THIS，LABEL

39

名字

- 名字：文件名、标题名、段名、过程名、符号常量名等等。
- 段名：可用作段基值。
  - 例如，段定义语句DATA SEGMENT，段名为DATA。源程序在进行汇编连接时，系统分配给该段一个段基值，设为2000H。这时，段名就可作为段基值2000H被引用。
- 过程名：代表过程的入口地址，作为调用指令的目的地址使用。
- 符号常量：在源程序中由符号常量定义语句命名。
  - 例如，COUNT EQU 20，此后COUNT可当作20被使用。

40

变量

- 已定义的变量具有下列三种属性：
  - 段属性、偏移地址属性、类型属性。
- 段属性：定义变量的段基值。此值必须在一个段寄存器中。
- 偏移地址属性：变量的偏移地址是从段的起始地址到定义变量的位置之间的字节数。
- 类型属性：变量的类型属性定义该变量所保留的字节数。
  - 例如，BYTE(DB，1个字节长)、WORD(DW，2个字节长)、DWORD(DD，4个字节长)、QWORD(DQ，8个字节长)和TBYTE(DT，10个字节长)。

41

变量（续）

- 要把变量的属性取出来作为操作数，需采用“分析运算符”。
  - ① SEG 变量名：取段基值。
  - ② OFFSET 变量名：取偏移地址。
  - ③ TYPE 变量名：取变量的数据类型。
    - 字节数据，TYPE 变量名＝1
    - 字数据， TYPE 变量名＝2
    - 双字数据，TYPE 变量名＝4
    - 8字节数据，TYPE 变量名＝8
    - 10字节数据，TYPE 变量名＝10

42

变量（续）

- 要把变量的属性取出来作为操作数，需采用“分析运算符”。
  - ① SEG 变量名：取段基值。
  - ② OFFSET 变量名：取偏移地址。
  - ③ TYPE 变量名：取变量的数据类型。
  - ④ LENGTH 变量名：表示变量所在数组的数据元素个数。
    - 注意：只有当数据用复制符DUP定义时，LENGTH才等于数组的元素个数，否则LENGTH就等于1。
  - ⑤ SIZE 变量名：表示变量所在数组的字节总数，且满足公式SIZE = LENGTH×TYPE。

43

例

```
DATA SEGMENT
  BUF1 DB N1, N2, N3, …, N10 ; N1~N10为10个字节数据
  BUF2 DB 10 DUP (0)
  BUF3 DW 10 DUP (?)
DATA ENDS
```

设段的段基值为2000H，则

SEG	BUF1=2000H
OFFSET	BUF1=0000H
TYPE	BUF1=1
LENGTH	BUF1=1
SIZE	BUF1=1

SEG	BUF2=2000H
OFFSET	BUF2=000AH
TYPE	BUF2=1
LENGTH	BUF2=10
SIZE	BUF2=10

SEG	BUF3=2000H
OFFSET	BUF3=0014H
TYPE	BUF3=2
LENGTH	BUF3=10
SIZE	BUF3=20

44

变量和常量

```
AREA1 DW  0867H
……
MOV AX, AREA1
```

- DW伪指令语句用来定义变量，变量用作表示存储器中的数据。程序运行过程中，变量是可以被修改的运算对象。
- 变量名为AREA1，表示内存中一个数据区的名字，也就是符号地址，该地址单元存放一个字数据0867H。
- MOV AX, AREA1 指令执行后，AX=0867H。

```
AREA1 EQU  0867H
……
MOV AX, AREA1
```

- 等值伪指令EQU给常数0867H定义了一个符号名AREA1，程序中可以使用符号AREA1代表立即数0867H。
- 这两段程序效果相同，但符号AREA1的含义不同。

45

修改属性运算符

- BUFW DW 1234H，5678H
- 合法：MOV AX, BUFW
- 非法：MOV AL, BUFW

- 合成运算符（又称修改属性运算符），可作用于变量和标号。
  - 类型（重新）指定运算符：PTR
  - 属性指定运算符：THIS
  - 类型指定命令：LABEL

46

修改属性运算符

- 类型（重新）指定运算符PTR
  - 格式：类型 PTR 变量/标号
    - 其中类型是BYTE, WORD, DWORD, NEAR, FAR
  - 功能：用新类型取代表达式默认的数据类型
- 例、 BUFW DW 1234H, 5678H
  MOV AL, BYTE PTR BUFW

- 可以用EQU和PTR定义一个新的变量，它与原变量具有相同的段属性和偏移地址属性，但类型属性不同。
- 例、BUFW DW 1234H, 5678H
  BUFB EQU BYTE PTR BUFW
  则 MOV AL, BUFB 合法。

47

修改属性运算符

- 属性运算符THIS
  - 格式：变量/标号 THIS 类型
  - THIS和EQU一起用来定义一个变量，其功能是将右边的类型属性赋给左边的变量/标号，该变量或标号的段地址和偏移量与下一个存储单元的地址相同。

```
BUFB  EQU  THIS  BYTE
BUFW  DW  1234H, 5678H
则 MOV AL, BUFB 合法。

FADDR  EQU  THIS  FAR
       MOV  AX, 100
```

此时MOV AX, 100前有标号FADDR，允许其它段的JMP指令跳转到本段FADDR标号地址上。

48

修改属性运算符

- 类型指定命令LABEL
  - 格式：变量/标号 LABEL 类型
  - LABEL伪指令定义一个指定的符号名，该符号名的段地址和偏移量与下面紧跟存储单元的相应属性相同，但该符号的类型是新指定的。使同一变量或标号在不同的地方引用时，可以采用不同的名字。

```
BUFB   LABEL  BYTE
BUFW  DW 1234H, 5678H
```

则MOV AL, BUFB合法。

```
DISF   LABEL FAR
DISN:  MOV AX, [SI]
```

则DISF和DISN指向同一条指令，DISF是DISN的别名。

49

修改属性运算符

- 以下两行等价：
  - tag1 EQU THIS BYTE
  - tag1 LABEL BYTE

50

地址计数器

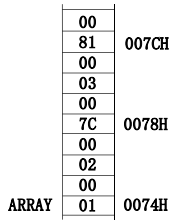
- 地址计数器$

  功能：保存当前正在汇编的指令(或数据)的地址。

  说明：可参加构成表达式。

示例1：ARRAY DW 1,2,$+4,3,$+5

示例2：BUFFER DB 1,2,3,4,5
COUNT EQU $-BUFFER
；COUNT的值为BUFFER的长度



51

定位伪指令

- 定位伪指令ORG

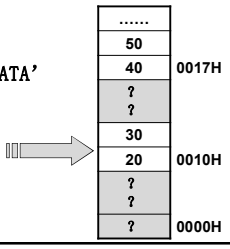
  格式：ORG 表达式

  操作：将表达式的值送入汇编地址计数器，即命令汇编程序从表达式值指定的位置生成目标代码。

  说明：若需相对$移动，则表达式=$+偏移量。

示例：

```
D_SEG SEGMENT PARA 'DATA'
   ORG 10H
   X   DB 20H, 30H
   ORG $+5
   Y   DB 40H, 50H
D_SEG ENDS
```



52

过程定义：PROC和ENDP

```
过程名  PROC  [类型]
        ------;
        RET ；或RET n
过程名  ENDP
```

- 过程名：在汇编语言程序中可作为调用指令的目的地址使用。
  - 例如，CALL SORT，其中SORT就表示过程名为SORT的过程的入口地址。
- 类型：far，near。
- 可以嵌套调用。
- 可以递归。

53

过程定义：PROC和ENDP

```
过程名  PROC  [类型] [USES …]
        ------;
        RET ；或RET n
过程名  ENDP
```

- 在MASM6.0版本的汇编程序中，PROC伪指令可以自动保持过程中使用的寄存器。这些寄存器用USES语句指示。
- 例，ADDS PROC NEAR USES BX CX DX

54

宏指令的定义、调用和扩展

- 宏（MACRO）是源程序中一段有独立功能的程序代码。它只需在源程序中定义一次，就可以多次用一条宏指令来调用。
- 宏定义是用伪指令来实现的。其格式为：

其中MACRO和ENDM是一对伪指令，这对伪指令之间是宏定义体，即“一组有独立功能的程序代码”。

- 宏指令名必须是合法的标识符，其第一个符号必须是字母。

55

宏指令的定义、调用和扩展

- 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宏调用。
- 宏调用的格式为：

  macro_name [actual parameter list]

  (实参数，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，而且用实参数一一取代宏定义的形式参数。

56

例1、宏定义可以无变元

宏定义：SAVEREG MACRO　　宏定义开始

```
        PUSH AX
        PUSH BX
宏指令  PUSH CX
        PUSH DX
        PUSH SI
        PUSH DI
        ENDM
```

宏定义体

宏定义结束

宏调用： SAVEREG

宏展开： 将宏定义体的内容（具有独立功能的一段代码）全部列出。

57

例2、宏定义带形式参数

宏定义： FOO MACRO P1, P2, P3
MOV AX, P1
P2 P3
ENDM

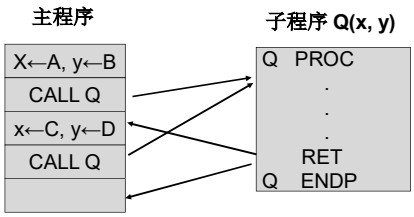
宏调用： FOO WORD_VAR, INC, AX

宏展开： ＋ MOV AX, WORD_VAR
＋ INC AX

58

过程调用和宏调用工作方式的区别

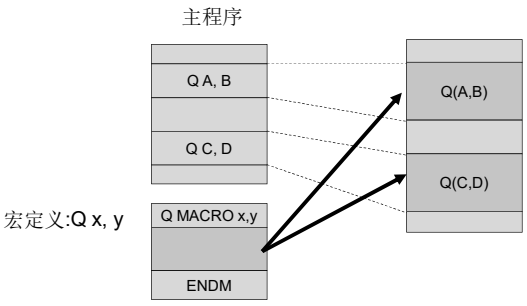
- 过程调用工作方式：执行时调用。



59

过程调用和宏调用工作方式的区别

- 宏调用工作方式：汇编时展开。



60

过程调用和宏调用工作方式的区别

1. 在处理时间上不同。
2. 传递参数的方式不同。
3. 用宏指令得到的目标代码长，占内存空间大，而且宏调用的次数越多，所占内存空间越大；用子程序占内存空间小，而且不会随调用次数的增加而增加，但执行时间长。

61

指令性语句中的标号

- 指令性语句的格式：
  [标号]：操作码 [操作数] [;注释]
- 标号具有三种属性。
  - 段属性、偏移地址属性和类型属性。
- 段属性：定义标号所在段的段基值，此值在一个段寄存器中。
- 偏移地址属性：标号的偏移地址是从段起始地址到定义标号的位置之间的字节数。

62

指令性语句中的标号

- 类型属性：用来指出该标号是供段内转移用，还是供段间转移用。
  - 如果是用于段内转移的，则类型为NEAR。
  - 如果是用于段间转移的，则类型为则称为FAR。
  - 标号为NEAR时，TYPE＝－1；
  - 标号为FAR时，TYPE＝－2。
  - 指令性语句中的标号，其类型一般为NEAR。
  - 标号类型的改变可以应用算符PTR、THIS和LABEL命令。
- 过程名与标号一样具有段属性、偏移地址属性和类型属性。

63

指令性语句中的操作数

- 指令中的操作数可以按寻址方式表示。
- 指令中的操作数也可以用段名、符号常量、变量、属性、过程名和标号来表示。
- 例如：
  - MOV AX, DATA；设DATA是段名，代表段基值，立即寻址方式
  - MOV SI, OFFSET ARRAY; OFFSET ARRAY是属性，立即寻址方式
  - MOV CX, COUNT；COUNT是符号常量，立即寻址方式
  - MOV BL, BUFFER; BUFFER是变量，直接寻址方式
  - CALL SBRT1; SBRT1是过程名，直接寻址方式
  - JMP DONE; DONE是标号，直接寻址方式
  - LOOP AGAIN; AGAIN是标号，直接寻址方式

64

汇编程序语法

- 常量、标识符和表达式
- 伪指令
- 存储器的组织
- 程序举例

65

存储器的组织

- 模型
- 完整段定义
- 简化段

66

存储模型

- 使用简化段的程序，每个模块必须用.MODEL伪指令说明存储模型，并且该伪指令必须在所有段定义之前。
- .MODEL伪指令格式如下：
  - .MODEL <模式> [选择项]
- MASM可以使用从小到大多种多种模型：
  - TINY、SMALL、MEDIUM、COMPACT、LARGE、HUGE、FLAT
- 选择项有三种：
  - 语言关键字、堆栈距离

67

存储模式的含义

存储模式	符号	意义
微模式	空或TINY	全部数据和代码放在一个物理段内（64KB），执行程序可以转化为.COM程序，该模式不能使用简化段。
小模式	SMALL	数据和代码各放在一个物理段内（64KB），所有代码和数据都用NEAR方式访问。这是最方便常用的方式。
中模式	MEDIUM	数据放在一个物理段内（64KB），但是代码可以大于64KB而放在不同的物理段内。所有数据都用NEAR方式访问，所有代码都用FAR方式访问。

68

存储模式的含义

存储模式	符号	意义
紧缩模式	COMPACT	代码段在一个物理段内（64KB），但是数据可以大于64KB而放在不同的物理段内。所有代码段都用NEAR方式访问，所有数据都用FAR方式访问。注意，每个数组本身不能大于64KB。
大模式	LARGE	代码和数据都可以大于64KB而放在不同的物理段内。所有代码和数据都用FAR方式访问。注意：每个数组本身不能大于64KB。预定义常数@DATASIZE值为1.

69

存储模式的含义

存储模式	符号	意义
巨型模式	HUGE	代码和数据都可以大于64KB而放在不同的物理段内。所有代码和数据都用FAR方式访问。注意，每个数组本身不能大于64KB。预定义常数@DATASIZE值为2。
平展模式	FLAT	允许用户使用32位位移量，但DOS下不允许使用这种模型，只能在OS/2下或其他保护模式的操作系统下使用。MASM 5.0版不支持该模型，但MASM 6.0支持。

70

.MODEL伪指令的选项

- 语言关键字选项：决定了过程和公共符号的调用约定及命名约定。可以使用C、FORTRAN、BASIC、PASCAL、stdcall等关键字。
- 堆栈距离选项：可以用NEARSTACK（默认）或FARSTACK来说明。
  - NEARSTACK是指把堆栈段和数据段组合到一个DGROUP段中，DS和SS均指向DGROUP段。
  - FARSTACK是指堆栈段和数据段不合并。

71

存储器的组织

- 模型
- 完整段定义
- 简化段

72

完整段定义

```
段名 SEGMENT [定位类型][组合类型] [字长类型][‘类别’ ]
————————————;
段名 ENDS
```

```
STACK SEGMENT PARA STACK ‘STACK’
  STA DB    50 DUP (?)
  TOP EQU   LENGTH STA
STACK ENDS
```

- 段名：通常段名确定了段的首地址，整个逻辑段存放在首地址开始的一片连续存储单元中。
- 源程序在进行汇编连接时，系统分配给段一个段基值，设为2000H。这时段名STACK就可以作为段基值2000H被引用。

73

完整段定义－定位类型

- 定位类型：说明段的起始地址应有怎样的边界值，取值为：
  - ① PARA：指定段的起始地址必须从节的边界开始，即段起始地址最低4位必须为0。这样，偏移地址可从0开始。
    - MASM把1M字节存储空间从0开始，每16个存储单元叫一节。
  - ② BYTE：该段可以从任何地址开始。这样，段起始地址的偏移地址可能不是0。
  - ③ WORD：该段必须从字的边界开始，即段起始地址必须为偶数（形如xxxx xxxx xxxx xxxx xxx0B）。
  - ④ PAGE：该段必须从页的边界开始，即段起始地址的最低两个十六进制数位必须为0(该地址能被256整除)。
- 定位类型的默认项是PARA。

74

完整段定义－组合类型

- 组合类型：多模块设计时如何分配各个段的空间。
  - NONE（“不选择”）：该段与其它同名段不进行连接，独立分配，为缺省参数。
  - PUBLIC：同名段顺序邻接，由低地址到高地址连接起来，连接成一个逻辑段。
  - COMMON：同名段相互覆盖。连接时该段与其它同名段有相同的基地址，连接长度为各分段的最大长度。
  - AT exp：指定段基址位置（段基值为按表达式exp计算所得的16位数）。CS不允许。
  - STACK：指示此段为堆栈，系统自动对SS和SP初始化。
    - 多模块只需设置一个堆栈，采用“覆盖”方式，容量按最大者。
  - MEMORY：指定该段在同名段的最后，其他段按PUBLIC处理。若有多个MEMORY段，则最前面（最先遇到）的段按MEMORY处理，其它按PUBLIC处理。

75

例1

- 同一模块中同名段顺序合并在一起。
- 以下两种写法等价。

```
DATA SEGMENT
 AREA1    DW 3031H
 COM_REG=20H
DATA ENDS

DATA SEGMENT
 BUF0     DW ?
DATA ENDS
```

⇔

```
DATA     SEGMENT
 AREA1   DW 3031H
 COM_REG=20H
 BUF0    DW ?
DATA     ENDS
```

76

例2

- 同一模块中同名段应具有相同的组合类型。

同一模块中，汇编出错：

```
DATA  SEGMENT PUBLIC
   AREA1     DW 3031H
   COM_REG=20H
DATA  ENDS

DATA  SEGMENT COMMON
   BUF0      DW ?
   BUF1      DW ?
DATA  ENDS
```

同一模块中，汇编正确：

```
DATA  SEGMENT COMMON
   AREA1     DW 3031H
   COM_REG=20H
DATA  ENDS

DATA  SEGMENT COMMON
   BUF0      DW ?
   BUF1      DW ?
DATA  ENDS
```

思考：右边的写法中，DATA段的长度？BUF0的偏移地址？

77

例3

同一模块中，汇编出错：

```
DATA  SEGMENT PUBLIC
AREA1 DW 3031H
COM_REG=20H
DATA  ENDS

DATA  SEGMENT COMMON
BUF0  DQ ?
DATA  ENDS
```

不同模块中，汇编正确：

```
DATA  SEGMENT COMMON
AREA1 DW 3031H
COM_REG=20H
DATA  ENDS

DATA  SEGMENT PUBLIC
BUF0  DQ ?
DATA  ENDS
```

- 可通过查看.LST文件和.MAP文件分析段的组合情况。

78

例4

```
; test0.asm
DATA    SEGMENT COMMON
        AREA1     DW  3031H
        AREA2     DW  '$'
DATA    ENDS
STACK   SEGMENT STACK
        db 50 DUP (?)
STACK   ENDS
CODE    SEGMENT
  ASSUME  CS:CODE, DS:DATA, SS:STACK
START:  MOV AX, DATA
        MOV DS, AX
        MOV DX, offset AREA1
        MOV AH, 9
        INT 21H
        MOV AH, 4CH
        INT 21H
CODE    ENDS
        END START
```

```
; test1.asm
DATA   SEGMENT PUBLIC
    BUF0      DW  ?
    BUF1      DW  ?
    BUF2      DW  ?
    BUF3      DW  ?
    BUF4      DW  ?
DATA   ENDS
    … …
END
```

- 思考：link test0.obj+test1.obj和link test1.obj+test0.obj后，DATA段的长度分别为：26，10。为什么？(MASM5.0)

79

完整段定义－字长类型

- 字长类型用来说明16位寻址方式还是32位寻址方式。
  - USE16、USE32
- USE16：16位寻址方式，段长不超过64KB，地址的形式是16位段基址和16位偏移地址。
- USE32： 32位寻址方式，段长不超过4GB，地址的形式是16位段基址和32位偏移地址。

80

完整段定义－类别

- 类别必须用单引号括起来。
  - 主要作用是指示汇编程序链接时将所有分类名相同（它们的段名不一定相同）的逻辑段组成一个段组，存放在连续的存储区中。

- 例：

```
A SEGMENT 'DATA'
B SEGMENT 'CODE'
C SEGMENT 'TO'
D SEGMENT 'DATA'
E SEGMENT 'TO'
```

- 链接后，在生成的EXE文件中，各段的相对位置为：

```
A SEGMENT 'DATA'
D SEGMENT 'DATA'
B SEGMENT 'CODE'
C SEGMENT 'TO'
E SEGMENT 'TO'
```

81

完整段：指定段寄存器伪指令

格式：ASSUME 段寄存器:段名 {[,段寄存器:段名]}
操作：明确段和段寄存器的关系。
说明：①代码段中必须至少有一个ASSUME语句，ASSUME可以出现在源程序中的任何地方；
②对同一段寄存器重复指定时最后一个有效。

（1）CS的指定

必须指定CS，最后一次指定必须在开始执行的段中，且段名为开始执行段的段名。

CS＝END指令中标号/过程名对应的段。

IP＝END指令中标号/过程名在段内的偏移地址。

82

完整段：指定段寄存器伪指令

（2）DS、ES的指定

- 对DS、ES的指定仅仅指明设置方案，未赋值，需要在程序中显式赋值。

```
D_SEG SEGMENT                    ; 数据段
      X DW ?, 4000H, 100 DUP(?)
D_SEG ENDS
E_SEG SEGMENT                    ; 附加段
      STRING DB  'EXAMPLE'
E_SEG ENDS
C_SEG SEGMENT PARA 'CODE'
      ASSUME CS:C_SEG,DS:D_SEG,ES:E_SEG
START:
      MOV  AX, D_SEG
      MOV  DS, AX        ; 数据段基址→DS
      MOV  AX, E_SEG
      MOV  ES, AX        ; 附加段基址→ES
       ……
```

83

完整段：指定段寄存器伪指令

（3）SS的指定

自动指定：当有组合类型为STACK的段时，SS自动指向该段，SP=段长。

自动指定示例：

```
S_SEG  SEGMENT  PARA  STACK
         DB 100 DUP(?)
S_SEG  ENDS
…
ASSUME CS:??, DS:??, SS: S_SEG
```

- 思考：若有多个STACK属性的堆栈段，结果如何？

84

完整段：指定段寄存器伪指令

- 缺省指定
  - 如果程序没有定义堆栈段，会出现“LINK: warning L4021: no stack segment”警告信息。
  - 该警告信息可以不必理会，因为在操作系统装入程序时会自动为其添加一个默认的堆栈段，即“缺省指定”。
- 缺省指定：当无组合类型为STACK的段时，无论是否有ASSUME设定SS对应的段，都采用缺省指定。

85

完整段：指定段寄存器伪指令

- 显式指定：使用MOV指令使SS及SP指向目的处。
- 示例：

```
S_SEG  SEGMENT
   STAK   DB 100 DUP(?)
   TOP    EQU LENGTH STAK
S_SEG  ENDS
C_SEG  SEGMENT PARA 'CODE'
       ASSUME CS:C_SEG,SS:S_SEG,DS:...,ES:...
START: ……
       MOV  AX, S_SEG  ；填入段基址
       MOV  SS, AX
       MOV  SP, TOP    ；填入栈顶指针
       ……
C_SEG  ENDS
    END START
```

- 思考：若有STACK属性的堆栈段，又用指令赋值后结果如何？

86

存储器的组织

- 模型
- 完整段定义
- 简化段

87

简化段

- 新版本的汇编程序（MASM5.0及MASM6.0以上）除支持SEGMENT伪指令外，还提供了一种新的较简单的段定义方法。
- 简化段的定义方法虽然不像SEGMENT伪指令那样具有较完整的表达能力，但简单易用。
- 简化段有利于实现汇编语言程序模块与高级语言程序模块的连接，可以有操作系统自动安排段序，自动保证名字、定义的一致性。

88

简化段

- 简化段程序的形式是：
  1) 开始先用.MODEL伪指令定义存储模型；
  2) 然后再用简化段伪指令定义段；
  3) 每一个新段的开始就是上一个段的结束，而不必用ENDS作为段的结束符。

89

简化段的定义

- 简化段定义没有段结束符ENDS。
  - 每个段用下一个段的段首部或模块尾部的END作为本段的结束。
- 简化段的段首部格式也与完整段不同，格式是：

  段描述 段名
  - 段名是本段的名字，往往是默认的。
  - 段描述给出该段的类别等信息。

90

简化段的定义

描述符	SMALL模式下的默认段名	段属性及意义
.CODE	_TEXT	代码段
.STACK	_STACK	堆栈段，格式为：.STACK 长度值
.DATA	_DATA	数据段，近访问，变量初始化
.FARDATA	_FARDATA	数据段，远访问，变量初始化
.DATA?	_DATA?	数据段，近访问，变量不初始化
.FARDATA?	_FARDATA?	数据段，远访问，变量不初始化
.CONST	_CONST	常数数据段。常量定义与数据段定义带初值的变量相同，但程序执行期间不可改变这些值。这种方法定义的常量分配存储空间，与用EQU、“=”定义的常量不同。

91

简化段的定义

- 定义堆栈段的伪指令格式是：

  .STACK [长度]

  – 其功能是定义一个堆栈段，并形成SS及SP的初值，SP=长度。

  – 如果省略长度，SP=1024。

- 例，

  – .STACK 1000H

92

简化段的定义

- 定义代码段的伪指令格式是：

  .CODE [名字]

  – 其功能是定义一个代码段。

  – 如果有多个代码段，则用名字区别。

- 定义数据段的伪指令包括4种：

  .DATA

  .DATA?

  .FARDATA

  .FARDATA?

  – 在简化段中，说明变量的格式与完整段一致。

93

简化段的定义

- 常数段是在简化段程序中的一种特殊的数据段。定义常数段的格式是：

  .CONST

  – 其功能是定义一个常数段，该段是近访问的。程序执行期间，段中数据不能改变。

94

简化段的定义

- 从简化段定义伪指令可以看出，简化段把数据段分得更细：

  1) 常数段和数据段分开；

  2) 初始化数据段和未初始化数据段分开。

  3) 近和远的数据段分开。

- 简化段的目的：不在于简化书写，而是有利于实现汇编语言程序模块与Microsoft高级语言程序模块的连接。在这种情况下，汇编语言的内存模式、段结构、段属性必须高级语言编译系统产生的段结构一致。采用简化段定义伪指令的目的正在于此。

95

预定义符号

- 汇编程序还给出了与简化段定义有关的一组预定义符号，可以再程序中出现，并由汇编程序识别使用。

- @MODEL：以数值表示当前使用的存储模型。

  – TINY=1

  – SMALL或FLAT=2

  – COMPACT=3

  – MEDIUM=4

  – LARGE=5

  – HUGE=6

96

预定义符号

- @CODE：由.CODE伪指令定义的段名。
- @CODESIZE：以数值表示当前代码段情况。
  - 在指定的存储模型中，只有一个代码段时（TINY，SMALL，MEDIUM，FLAT），值为0；有多个代码段时，值为1.
- @DATA：由.DATA伪指令定义的段名，或有.DATA、.DATA?、CONST和.STACK定义的段组名。

97

预定义符号

- @DATASIZE：以数值表示当前数据段的情况。
  - 在指定的存储模型中，只有一个数据段时（TINY，SMALL，MEDIUM，FLAT），此值为0；
  - 有多个数据段的COMPACT和LARGE时，此值为1；
  - 有多个数据段，且有超过64KB的大数据段的HUGE时，此值为2。
- @FARDATA、@FARDATA？：段名。
- @CURSEG：当前段名。

98

预定义符号

- @STACK：堆栈段的段名或段组名。
- @FILECUR：当前文件名（包括扩展名）。
- @FILENAME：当前文件名（不包括扩展名）。
- @WORDSIZE：表明段时16位还是32位的数值回送符。
  - 16位段，值为2。
  - 32位段，值为4。

99

关于简化段的两点说明

- 凡是与高级语言程序连接的程序，必须把常数与变量分开，变量中又要把赋初值、不赋初值的分开，并分别在.CONST、.DATA、.FARDATA、.DATA?、.FARDATA？中定义。远访问数据段只能在压缩模式、大模式和巨型模式中使用，其他数据段和代码段可在任何模式下使用。
- 独立的汇编语言程序（即不与高级语言连接的源程序），只需使用.MDOEL、.CODE、.STACK、.DATA这5中简化语句，并且不区分常数与变量、是否赋初值。在.DATA段中，所有数据定义语句均可使用。

100

.STARTUP和.EXIT

- 汇编程序MASM中，还提供了二组简化的代码伪指令：.STARTUP和.EXIT，用于程序的启动和结束。
- .STARTUP：在代码段的开始，用于自动初始化寄存器DS、SS和SP。
- .EXIT：用于结束程序的运行，它等价于下列二条语句：

```
MOV   AH, 4CH
INT    21h
```

101

.STARTUP和.EXIT

- 以下两段代码等价。

```
.MODEL  SMALL
.STACK  128
.DATA
   MSG DB   "CS11$"
.CODE
   MOV  AX, @DATA
   MOV DS, AX
   MOV DX, offset MSG
   MOV AH, 9H
   INT 21h
   MOV AX, 4C00H
   INT 21h
END
```

```
 .MODEL  SMALL
.STACK  128
.DATA
    MSG DB   “CS11$"
.CODE
.STARTUP
   MOV DX, offset MSG
   MOV AH, 9H
   INT 21h
   .EXIT
END
```

102

汇编程序语法

- 常量、标识符和表达式
- 伪指令
- 存储器的组织
- 程序举例

103

示例程序1

```
DATA SEGMENT
    NUM   DW 0011101000000111B
    NOTES  DB 'The result is :','$'
DATA  ENDS
CODE  SEGMENT
    ASSUME  CS:CODE, DS:DATA
BEGIN:
        MOV  AX, DATA
        MOV  DS, AX
        MOV  DX, OFFSET NOTES
        MOV AH, 9H
        INT  21H; 显示字符串
        MOV  BX, NUM
        MOV  CH, 4
ROTATE :
        MOV  CL,  4
        ROL   BX, CL
        MOV  AL, BL
        AND   AL, 0FH
        ADD  AL, 30H
        CMP  AL, '9'
        JLE   DISPLAY
        ADD   AL, 07H
DISPLAY:
        MOV  DL, AL
        MOV   AH, 2
        INT  21H;  显示一个字符
        DEC  CH
        JNZ   ROTATE
        MOV  AX, 4C00H
        INT  21H;  终止并退出
CODE  ENDS
        END   BEGIN
```

104

示例程序2

```
.MODEL SMALL

.DATA
    NUM   DW 0011101000000111B
    NOTES  DB 'The result is :','$'

.CODE
.STARTUP
        MOV  DX, OFFSET NOTES
        MOV AH, 9H
        INT  21H; 显示字符串
        MOV  BX, NUM
        MOV  CH, 4
ROTATE :
        MOV  CL,  4
        ROL   BX, CL
        MOV  AL, BL
        AND   AL, 0FH
        ADD  AL, 30H
        CMP  AL, '9'
        JLE   DISPLAY
        ADD   AL, 07H
DISPLAY:
        MOV  DL, AL
        MOV   AH, 2
        INT  21H;  显示一个字符
        DEC  CH
        JNZ   ROTATE
.EXIT
END
```

105

示例程序3

```
;程序功能：显示一个信息框。
;ex1.asm (…\base);程序名
;编译链接方法：
;ml /c /coff  ex1.asm
;link /subsystem:console ex1.obj

.386                    ;指明指令集
.model flat,stdcall  ;程序工作模式，flat为Windows程序使用
                        ;的模式(代码和数据使用同一个4GB段),
                        ; stdcall为API调用时右边的参数先入栈
option casemap:none     ;指明大小写敏感
```

106

示例程序3

```
include windows.inc
include user32.inc
includelib user32.lib
include kernel32.inc
includelib kernel32.lib

.data                   ;数据段
    szCaption  db       '抬头串',0
    szText              db       'Hello！',0

 .code                  ;代码段
start:
```

107

示例程序3

```
invoke MessageBox,      ;显示信息框
        NULL,           ;父窗口句柄
        offset szText,  ;正文串的地址
        offset szCaption,  ;抬头串的地址
        MB_OK           ;按钮

invoke ExitProcess,     ;终止一个进程
        NULL            ;退出代码

end    start            ;指明程序入口点
```

108

本章小结

- MOV指令回顾
  - 机器指令的格式
- 数据传输指令
  - PUSH/POP、装入有效地址、数据串传送、其他数据传送指令
- 段超越前缀
- 汇编程序语法
  - 程序结构、伪指令、存储模型

109

作业

- 习题11、习题21、习题25、习题43
- 【补充题1】指令AND AX, 7315H AND 0FFH中，两个AND有什么区别？这两个AND操作分别在什么时候执行？
- 【补充题2】 设计指令序列，将字符$送入附加段中偏移地址为0100H的连续10个单元中。
- 【补充题3】设VAR是一个DATE类型的结构变量。 DATE结构有3个成员： 字节变量MONTH用于保存月、字节变量DAY保存日，字变量YEAR用于保存年。要求用伪指令STRUC定义该结构体，并给出将2000年1月1日赋值为变量VAR的代码段。
- 【补充题4 】对于指令“MOV AX, DATA”，设DATA是段名，代表段基值，属于立即寻址方式。为什么？

110