

## 第四章 高级汇编语言的语法

- 4.1 汇编语句表达式
- 4.2 汇编伪指令语句
- 4.3 宏功能语句\*
- 4.4 模块连接



武汉大学

Wuhan University

School of computer, Aiping XU



### 4.1.1 汇编语言语句格式

- 汇编语言源程序中的语句可以由四项组成，格式如下：
  - [名字] 操作符 [操作数] [；注释]
- 名字项是程序设计人员自己定义的符号，用于代表内存单元的地址，表示本条语句的符号地址。一般来说，名字可以是标号和变量，统称为标识符
- 操作符可以是指令、伪指令或宏指令的助记符。
- 操作数字段是操作符的操作对象。
- 注释字段是以“；”开头的说明部分，可以用英文或者中文书写



- 伪指令语句格式如下：

[名字] 伪指令符 [参数, …参数] [; 注释]

- 名字：符号名是伪指令语句的一个可选项。
- 伪指令符：伪指令符指定汇编程序要完成的具体操作，如数据定义伪指令DB、DW、DD，段定义伪指令SEGMENT，假定伪指令ASSUME等。
- 参数表：伪指令后面的操作数可以是常数、字符串、变量、表达式等，其个数由具体的伪指令决定，各个操作数之间必须以“逗号”分隔。
- 注释：伪指令的注释必须以“;”开始，其作用同指令语句中的注释部分。



### 名字的定义规则

- 就是由用户按一定规则定义的标识符，可由英文字母(A~Z, a~z)，数字(0~9)和特殊符号(?、@、\_)等组成
- 名字的定义要满足如下规则：
  - (1) 数字不能作为名字的第一个符号；
  - (2) 单独的问号(?)不能作为名字；
  - (3) 一个名字的最大有效长度为31，超过31的部分计算机不再识别；
  - (4) 汇编语言中有特定含义的保留字，如操作码、寄存器名等，不能作为名字使用。
- 为了便于记忆，名字的定义最好能够见名知义，如用BUFFER表示缓冲区，SUM表示累加和等



## 操作码和操作数说明

### ➤ 操作码

- 操作码用来指明操作的性质或功能。指令中的助记符都是操作码。操作码与操作数之间用空格分开，如MOV、ADD等都是操作码

### ➤ 操作数

- 指令中的操作数用来指定参与操作的数据。对于一般指令，可以有1个或2个操作数，也可以没有操作数；对于伪指令和宏指令，可以有多个操作数。当操作数多于1个时，操作数之间用逗号分开
- 操作数可以是常数和表达式
  - 表达式分为数值表达式和地址表达式



## 注释项说明

### ➤ 注释项

- 注释是语句的说明部分。用来说明一条指令或一段程序的功能，由分号(;)开始
- 适当地加些注释内容，可以增加程序的可读性，便于阅读、理解和修改程序
- 汇编源程序时，注释部分不产生机器代码
- 一条语句可以写在多行上，续行符使用&



- P75 如:
- `VARN DB 1,9,7,5,2`
- `MOV AL,VARN`
- `MOV SI,1`
- `MOV AL,VARN[SI]`



#### 4.1.2 数值表达式

- 数值表达式是常量或运算符和常量的组合
- 1 常量
- 2 算术运算符
- 3 逻辑运算符
- 4 关系运算符



## 4.1.2 数值表达式

- 1 常量
  - 整常量
  - 字符串常量
  - 符号常量



## 4.1.2 数值表达式

### ■ 整常量

常量形式	格式	X 的取值	常例	说明
二进制常量	XX...XB	0 或 1	01110011B	数据类型后缀为 B
八进制常量	XX...XO	0~7	12537O	数据类型后缀为 O
十进制常量	XX...X XX...XD	0~9	1234D 1234	数据类型后缀为 D 后缀可省略
十六进制常量	XX...XH	0~9 A~F	0AB12H	如果第一位数是 A~F， 则必须在数的前面加 0

各种形式数字常量格式对照表



### 4.1.2 数值表达式

#### ➤ 字符串常量

- 字符串常量是用单引号或双引号引起来的一个或多个字符
- 字符串常量是以各字符的ASCII码表示的。如 ‘A’ 用41H 表示，字符串 ‘A1B2’ 用41H, 31H, 42H, 32H表示

➤ 例： `MOV DL, 'A' ;`            `DL= 'A' =41H`  
         `MOV AL, 'a' ;`            `AL= 'a' =61H`  
         `MOV AX, 'Aa' ;`        `AX= 'Aa' =4161H`



### 4.1.2 数值表达式

#### ➤ 符号常量

- 用等价语句EQU或者“=”语句定义的名字来表示常量
- 可以直接写在汇编与句中能提高程序的可读性，使用修改也很方便
- 例： `COUNT = 60 * 60 ; COUNT = 3600`

`MOV ECX , COUNT`



## 4.1.2 数值表达式

- 2. 算术运算符
  - + (加)、- (减)、\* (乘)、/ (除)、
  - MOD (求模)、SHL (逻辑左移)、SHR (逻辑右移)
- 既可以用于数值表达式又可用于地址表达式
- 如: VAL EQU 5
- MOV AL, VAL SHR 1



## 4.1.2 数值表达式

- 3. 逻辑运算符
- 逻辑运算符包括: 逻辑乘 (AND)、逻辑加 (OR)、按位加 (XOR)、逻辑非 (NOT) 四种运算
- 由于逻辑运算是按位操作, 且在汇编过程中完成, 因而运算的结果仍为整数常量
- 逻辑运算符只能用于数值表达式中, 不能用于地址表达式中
- 逻辑运算符和逻辑运算指令是有区别的
  - 逻辑运算符的功能在汇编阶段完成
  - 逻辑运算指令的功能在程序执行阶段完成
  - 如: PORT EQU 29H
  - AND DX, PORT AND 0F0H



### 4.1.2 数值表达式

#### 4. 关系运算符

- 关系运算符包括：相等（EQ），不等（NE），小于（LT），大于（GT），小于等于（LE）及大于等于（GE）
- 关系运算符要有两个运算对象
- 两个运算对象要么都是数值，要么都是同一个段内的地址
- 运算结果为真时，表示为0FFFFH；运算结果为假时，表示为0000H
- 如： PORT EQU 3
- MOV BX, ((PORT LT 5) AND 18 OR ((PORT GE 5) AND 69)
- 编译成： MOV BX, 18 或 MOV BX, 69



### 4.1.3 标号变量与地址表达式

- 汇编语言中用来表示存储器操作数的地址或程序指令的转移地址
  - 标号
  - 变量
  - 地址表达式
    - 可由变量、标号、常量、寄存器以及一些运算符组成





### 4.1.3 标号与地址表达式

#### 1. 标号

- 标号是一条指令语句的符号地址
- 在汇编源程序中，只有在需要转向一条指令语句时，才为该指令语句设置标号，以便在转移类指令（含子程序调用指令）中直接引用这个标号。因此，标号可作为转移类指令的操作数，即转移地址
- NEAR(近程) 类型标号表示段内标号
  - 需要用2字节（16位方式）或者4字节（32位方式）给出标号所在的段内EA
- FAR（远程）类型标号表示段间标号
  - 需要用4字节（16位方式）或者6字节（32位方式）给出标号所在的段地址以及EA



### 4.1.3 标号与地址表达式

#### 2. 变量

- 变量是内存中一个数据区的名字，作为指令的存储器操作数来使用。具有三种属性：段地址，EA和类型。EA是汇编时汇编地址计数器LC的值。
- 变量的类型由DB, DW, DD, DQ和DT 来定义
- 例如：COUNT DW 5 ; 定义变量COUNT类型是字
  - VAR DB 'A', 'B' ;定义字符AB的ASCII码
  - MOV AL, VAR ;语句正确，AL和VAR类型相符
  - MOV EAX, VAR ;语句不正确，VAR 和EAX类型不符

### 3. 地址表达式

- 汇编语句的地址表达式形成的结果是存储器的地址，用来表示指令中的标号或者变量操作数，有三个属性：段地址，EA和类型
- 变量仅对应于数据区的第一个数据项，如果对其他数据项操作，必须要用地址表达式指出哪一个数据项是指令的操作数
- 例如：ARRAY DD 2, 0, 0, 2, 6, 1, 8, 9, 40, 4 ;定义10个双字类型的数据

MOV EAX, ARRAY + 36 ; 取第10 个元素

MOV ESI, 36 ;第10个元素的地址偏移量送ESI

SUB ESI, 4 ;ESI - 4 ->ESI

ADD EAX, ARRAY[ESI] ;EA = ARRAY + ESI ,EAX 与  
;第9个元素相加 =4+40=44H

### 4.1.4 特殊运算符

- 地址表达式中可使用特殊运算符
- 运算符只有在包含其自身的本条指令内才有效
  - 属性替换运算符
    - PTR, :, SHORT, THIS
  - 属性分离运算符
    - SEG, OFFSET, TYPE
  - 返回数值运算符
    - LENGTH, SIZE, \$, HIGH, LOW
    - 
    - 见图4.1

## 4.1.4 特殊运算符

### 1. 属性替换运算符

- (1) 强制类型转换PTR运算符
- 格式：类型 PTR 地址表达式
- PTR与EQU连用，可定义与PTR右边地址表达式类型不同的新变量名或新标号,但不另分配存储单元。 `JMP FAR PTR NEXTLAB[BX]`
- 例如：  
`NBYTE DB 0,2,4,6,8` ;定义NBYTE为字节型变量  
`MOV AX,WORD PTR NBYTE` ;字节变量临时作字使用  
`MOV AL,NBYTE` ;显式类型为字节由DB定义  
`MOV CH,NBYTE + 4` ;隐式类型，NBYTE + 4 伪字节  
`MOV WORD PTR [BX],5` ;PTR强制说明模糊型为字类型  
`MOV AX,[SI + 8]` ;两个操作数中以确定的类型为准

## 4.1.4 特殊运算符

- (2) 段地址取代运算符 “:”
- 又称为段地址运算或者跨段前缀
- 用于临时给变量，标号或者地址表达式指定一个段属性，地址表达式EA和类型保持不变。在指令中替代默认的段以形成物理地址。
- 例如： `MOV DX,ES:[BX + DI]`
- ;  $EA = BX + DI$ ,  $PA = ES * 16 + EA$ ，临时替换默认的DS



#### 4.1.4 特殊运算符

- (3) 短取代运算符SHORT
- 告诉汇编程序，目标标号在本条指令的-128~+127个字节的范围内，生成节省一个字节的机器指令。一般情况下可以不用
- 例如：

```
JMP SHORT NEXTBRACH
```

...

```
NEXTBRACH: MOV EAX, ECX
```



#### 4.1.4 特殊运算符

- (4) 定义类型运算符THIS
- 与PTR类似，**用于建立同一地址而类型不同的变量或者标号**，方便按照不同类型存取变量或程序发生转移。
- **不分配存储单元**，标号或变量的段属性为语句所在段的段地址，EA是该语句所在的下一个可用的存储单元地址
- **一般与EQU或“=”伪指令连用，产生一个变量或标号**
- 例如：

```
BYTE_VAR EQU THIS BYTE ;定义BYTE_VAR为字节
```

```
WORD_VAR DW 8800H
```

```
MOV AL,BYTE_VAR ;BYTE_VAR地址同WORD_VAR,AL = 00H
```

```
MOV AX,WORD_VAR ;AX = 8800H
```

## 4.1.4 特殊运算符

### 2. 属性分离运算符

➤ SEG分离出其后变量或标号所在段的段首址；OFFSET分离出其后变量或标号的偏移地址

➤ 例如：

➤ `ARRAYW DW 20 DUP(0)`

...

`START: MOV AX, SEG ARRAYW ;ARRAYW段址->AX`

`MOV DS, AX`

`MOV BX, OFFSET ARRAYW;ARRAYW的EA->BX`

上面一条可以用这条代替：`LEA BX, ARRAYW`

比较下面两条：

`LEA DX, ARRAYW[BX+SI]`

`MOV DX, OFFSET ARRAYW[BX+SI]` 错：只能用直接寻址

## 4.1.4 特殊运算符

➤ TYPE运算符

➤ 格式：TYPE 变量或标号

➤ 功能：分离出其后变量或标号的类型值

● 如果是变量，将返回该变量的类型对应字节数，如字节（1），字（2）等

● 如果是标号，则返回代表标号类型的数值

■ NEAR类型，取值为-1

■ FAR类型，取值为-2

■ 如：`MOV CL, TYPE ARRAYW`



#### 4.1.4 特殊运算符

##### 3. 返回值运算符

- (1) 取变量数据项个数LENGTH运算符
- 格式: LENGTH 变量
- 功能: 取出变量所含的数据存储单元个数
  - 带有一个DUP项时, 取DUP项前面的重复系数
  - 有多个DUP项时, 仍然取第一个DUP项前的数字
  - 其余情况, LENGTH变量的取值为1



#### 4.1.4 特殊运算符

- \*\*\* (3) 动态求数据项的个数
- 对字符串变量求数据项的个数, 或者表达式有多个DUP项嵌套, 或者表达式没有DUP项时, 使用SIZE达不到取数据项的个数的目的
- 在程序中使用“\$”符号来代表汇编程序下一个可用偏移值, 来达到动态求数据项的个数的目的



- 例如: `STRING DB 'Advance assembly'`  
`COUNTS EQU $ - STRING ;COUNTS = 16`
- `VARW DW 9,8,7,6,5,4,3,2,1,0`
- `COUNTW EQU ($-VARW)/2 ; COUNTW=10`
- `BUFFER DB 10 DUP(2 DUP(8),'A');`
- `COUNTL EQU $-BUFFER; COUNTL=30;`
- $10 * (2 + 1) = 30$
- `MOV CL,COUNTS ;16->CL`
- `MOV CX,COUNTW ;10->CX`
- `MOV ECX,COUNTL ; 30->ECX`



#### 4.1.4 特殊运算符

- (4) 字节分离运算符
- 字节分离运算符包括: HIGH和LOW
- 格式: `HIGH 常量或地址表达式`
- `LOW 常量或地址表达式`
- 功能: HIGH 用来分离出其后16位常量或地址表达式的偏移量的高字节; LOW用来分离出其后16位常量或地址表达式偏移量的低字节

例如:

```
WVAL EQU 0CA19H
MOV AH,HIGH 8162H ;取高位字节81H->AH
MOV AL,LOW WVAL ;取低位字节19H->AL
```



## 4.2 汇编伪指令语句

- 汇编语言程序的语句除指令以外还可以由伪指令和宏指令组成。伪指令又称为伪操作，它们不像机器指令那样是在程序运行期间由计算机来执行的，而是在汇编程序对源程序汇编期间由汇编程序处理的操作，它们可以完成如处理器选择、定义程序模式、定义数据、分配存储区和指示程序结束等功能。伪指令形式上与一般指令相似，但伪指令只是为汇编程序提供有关信息，不产生相应的机器代码。



## 4.2 汇编伪指令语句

- 4.2.1 方式伪指令
- 4.2.2 数据与符号定义伪指令
- 4.2.3 段定义与段管理伪指令SEGMENT/ENDS
- 4.2.4 过程定义伪指令
- 4.2.5 模块连接伪指令





### 4.2.1 方式伪指令

➤ 32位pc系列微机有386、486、Pentium等多种CPU也就有多种指令系统。这些指令虽然向上兼容，却是不断进行扩充的，增加了许多功能丰富的新指令。为了告诉汇编程序怎样进行汇编，识别哪种类型的CPU指令系统，以保证扩充功能的指令能够正确汇编，于是采用方式伪指令来制定具体的指令系统

➤ 如果没有指定，则masm默认的是：8086/8088 cpu和8087协处理器的指令系统以及浮点变量的IEEE格式

➤ 方式伪指令主要有：

. 8086    . 386    . 386P    . 486    . 486P  
. 586    . 686    . MMX    等



### 4.2.1 方式伪指令

- |          |                     |
|----------|---------------------|
| ■ . 8086 | 选择8086指令系统          |
| ■ . 286  | 选择80286指令系统         |
| ■ . 286P | 选择保护方式下的80286指令系统   |
| ■ . 386  | 选择80386指令系统         |
| ■ . 386P | 选择保护方式下的80386指令系统   |
| ■ . 486  | 选择80486指令系统         |
| ■ . 486P | 选择保护方式下的80486指令系统   |
| ■ . 586  | 选择Pentium指令系统       |
| ■ . 586P | 选择保护方式下的Pentium指令系统 |

### 4.2.1 方式伪指令

- 这类伪指令一般放在整个程序的最前面。如不给出，则汇编程序默认值为 .8086指令系统。它们可放在程序中，如程序中使用了一条80486所增加的指令，则可以在该指令的上一行加上 .486
- 例如：
- .386  
CODES SEGMENT 'CODE' ;设置Pentium兼容的32位386方式  
...  
CODES ENDS

### 4.2.2 数据与符号定义伪指令

#### 1. 定义符号伪指令

- 伪指令EQU、= 和 LABEL用于定义名字(符号)，不分配存储单元；
- (1) 格式：符号名 EQU 表达式
- 它们将右边的表达式赋予左边的一个名字，以后在程序中任何需要此表达式的地方，均可用该名字来引用代替；
- (2) 格式：符号名 = 表达式
- = 伪指令与EQU相类似，也可以作为赋值操作使用，它们之间的区别是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



- 注意:
- ① EQU的表达式可以是常量、变量名、标号、指令助记符、寄存器名等;而=的表达式只能是数字表达式,主要用来定义符号常量。
- ②同一个源程序,EQU的符号名不允许重复定义,否则引起同名错;=的符号名允许重复定义,相关语句使用该符号值是以其最接近的事先定义的值。
- ③同一个源程序中同一符号名,不能同时用EQU和=语句定义。



- 例如:
- **NUMBER EQU 60 ;NUMBER =60**
- **TIMES EQU NUMBER \* 24 ; TIMES = 60 x 24 = 1440**
- **BYTES DB 10 DUP(?)**
- **WORDF EQU WORD PTR BYTES ;定义新名字新类型**
- **COUNT EQU CX ;COUNT为CX寄存器的同义名字**
- **DEC COUNT ;意指CX的内容-1→CX**
- **VALUE = 88**
- **VALUE = VALUE + 1 ;88+1=89**
- **MOV CL, VALUE ;CL = 89**

## 4.2.2 数据与符号定义伪指令

- (2) 格式: 变量名或标号 LABEL 类型
- 功能: 定义与原有变量类型不同的新变量或为指令语句定义有指定类型的标号
- 通常与数据定义伪指令连用, 其功能类似语句“变量名或标号 EQU THIS 类型”
- 例如:
  - BARRA Y LABEL BYTE ; 类似: BARRA Y EQU THIS BYTE
  - WARRAY DW 50 DUP(0)
  - MOV AX, WARRAY[48H]; 取数组的第25个字元素→AX
  - MOV AL, BARRA Y[49H];取数组的第50个字节值→AL
  - 后一条语句也可改写成: MOV AL, BYTE PTR WARRAY[49H]
  - ⋮

- LABEL语句还可以定义指向同一指令的NEAR和FAR类型的标号。由其他程序段来调用FAR标号, 在同一段内调用NEAR标号。例如:
  - FARSUB LABEL FAR
  - NEARSUB: MOV EBX, ECX
  - . . . . .
  - JZ NEARSUB; 结果为0则转标号NEARSUB
- 另一程序段可用:JMP FARSUB 来执行MOV EBX, ECX指令。



## 4.2.2 数据与符号定义伪指令

### 2. 程序模块名与结束语句

- 为了提供模块程序的设计功能，汇编语言提供了划分模块并命名的能力
- (1) 模块命名语句 **NAME**
- 格式: **NAME** [模块名]
- 源程序模块开始的一个语句，有模块名则为模块的名字，名字不要使用程序的变量、标号和保留字。**NAME**语句在原程序中也可以不必写出。
- (2) 源程序模块结束语句 **END**
- 格式: **END** [表达式]
- 源程序模块的最后一个语句。如果有表达式的值，则为程序开始执行目标代码的地址。汇编结束后，指令指针**IP**或**EIP**指向这个入口地址。



## 4.2.2 数据与符号定义伪指令

### 2. 程序模块名与结束语句

例如:

```
NAME    Pentium_MASM61X
CODES   SEGMENT
START:  ...
        ...
CODES   ENDS
        END      START
```



## 4.2.2 数据与符号定义伪指令

### 3. \*\*\*\* 数据定义以及存储分配伪指令

- 常用的数据定义伪指令有DB(8位)，DW(16位)，DD(32位)，DQ(64位)，DT(80位)
- 格式：[变量名] 数据定义伪指令 表达式[, 表达式]
- 功能：定义数据存储区，类型由数据定义伪指令确定，初值由表达式给定



## 4.2.2 数据与符号定义伪指令

- 表达式可以有以下几种类型：
  - (1) 数值表达式；
  - (2) ? 定义的数据项没有确定的初值，只分配存储单元
  - (3) ASCII字符串；多于2个以上字符的字符串可以用DB来定义
  - (4) n DUP(?)；重复定义n个数据项，内容不确定；
  - (5) n DUP(表达式[, 表达式, ...])；定义n个数据项
  - (6) n DUP(m DUP(表达式1)) 多层嵌套DUP项

## 4.2.2 数据与符号定义

### 例4.1

```
VARN DB 88          ; 定义一个字节88H
VARM DW 60 * 60      ; 定义一个双字3600
STRC  DW 'AB'        ; 定义一个双字 'AB'
MAXN DD -2147483848
STRS DD 'AB' ; 相当00004142, 然后倒着存放;
ERROR DB 'syntax error',13,10
; 定义一个字符串和数字
TABB DB 2 DUP(3 DUP(8),'A') ; 定义2个字符串, 每个字符串 3个8和一个 'A'
```

V A R N	0 0 0 0 H	8 8
V A R M	0 0 0 1 H	
	0 0 0 2 H	
S T R C	0 0 0 3 H	4 2 H
	0 0 0 4 H	4 1 H
M A X N	0 0 0 5 H	
	0 0 0 6 H	
	0 0 0 7 H	
	0 0 0 8 H	
S T R S	0 0 0 9 H	4 2 H
	0 0 0 A H	4 1 H
	0 0 0 B H	0 0
	0 0 0 C H	0 0
E R R O R	0 0 0 D H	' s '
	0 0 0 E H	' y '
	0 0 0 F H	' n '
	0 0 1 0 H	' t '
	0 0 1 1 H	' a '
	0 0 1 2 H	' x '
	0 0 1 3 H	' '
	0 0 1 4 H	' e '
	0 0 1 5 H	' r '
	0 0 1 6 H	' r '
	0 0 1 7 H	' o '
	0 0 1 8 H	' r '
	0 0 1 9 H	0 d h
	0 0 1 A H	0 a h
T A B B	0 0 1 B H	8
	0 0 1 C H	8
	0 0 1 D H	8
	0 0 1 E H	' A '
	0 0 1 F H	8
	0 0 2 0 H	8
	0 0 2 1 H	8
	0 0 2 2 H	' A '

## 示例2-4

### 例2

- DATA1 DB 11H, 33H ; 定义包含两个元素的字节变量DATA
- NUM DW 100\*5+88; 定义一个字类型变量NUM其初值为表达式的值
- SUM DD 44556677H ; 2字存入变量SUM。

DATA1	11H
	33H
NUM	低位
	高位
SUM	77H
	66H
	55H
	44H

## 示例4.2

### 例3

- **STR1 DB 'COMPUTER' ;** 定义一个字符串，字符串的首地址为**STR1**
- **STR2 DW 'AA', 'BC', 'DE' ;** 给两个字符组成的字符串分配两个字节存储单元
- **DATA1 DW ?, ?, ? ;** 为**DATA**预留6个存储单元

里面放的是随机数

STR1	'C'
	'O'
	'M'
	'P'
	'U'
	'T'
	'E'
	'R'
STR2	41H
	41H
	'C'
	'B'
	'E'
	'D'
DATA1	?
	?
	?
	?
	?
	?

### 例4

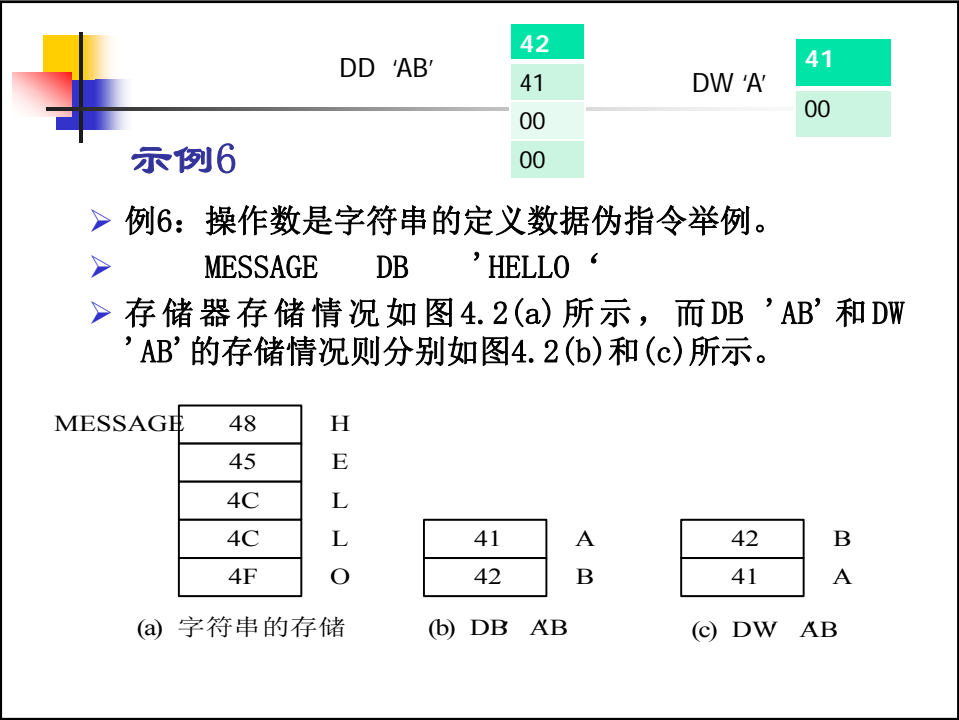
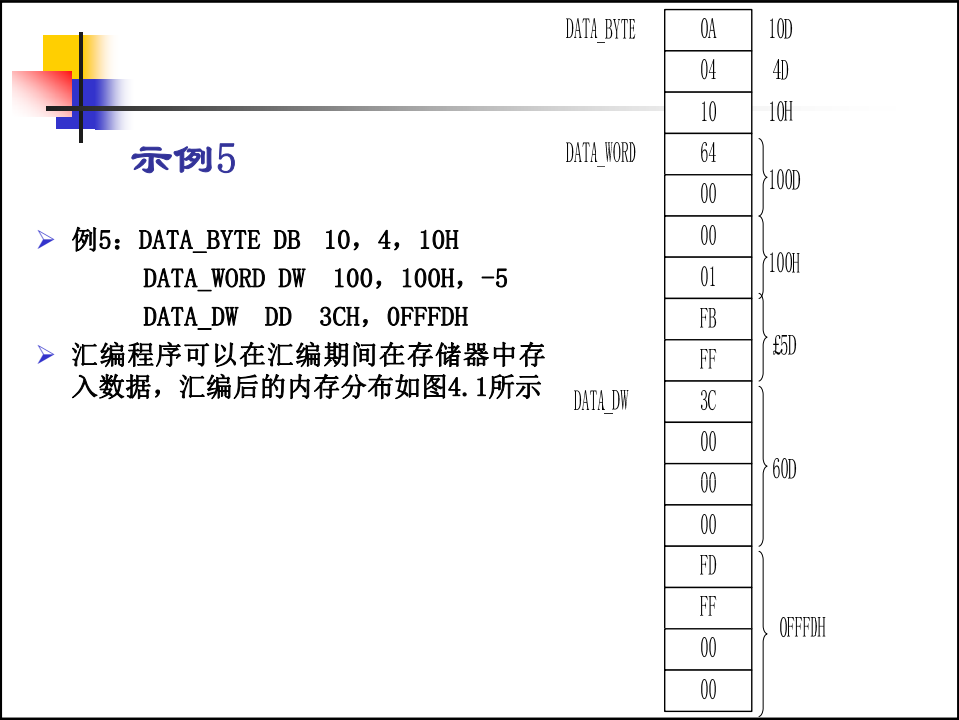
- **DATA1 DB 20 DUP (?) ;** 为变量**DATA1**分配20个字节的空间，初值为任意值
- **DATA2 DW ? ;** 为变量**DATA2**分配2个字节的空间，初值为任意值
- **DATA3 DB 20 DUP (30H) ;** 为变量**DATA3**分配20个字节的空间，初值均为30H

DATA1	?
	?
	⋮
	?
	?
	?
	?
DATA2	?
	?
DATA3	30H
	30H
	30H
	⋮
	30H
	30H
	30H
	30H

20 个任意值单元

20 个值为 30H 的单元





### 示例7

- 例7： 操作数？可以保留存储空间，但不存入数据。如
- ABC DB 0, ?, ?, 0
- DEF DW ?, 52, ?
- 经汇编后的存储情况如图4.3所示

ABC	00
	-
	-
	00
DEF	-
	-
	34
	00
	-
	-

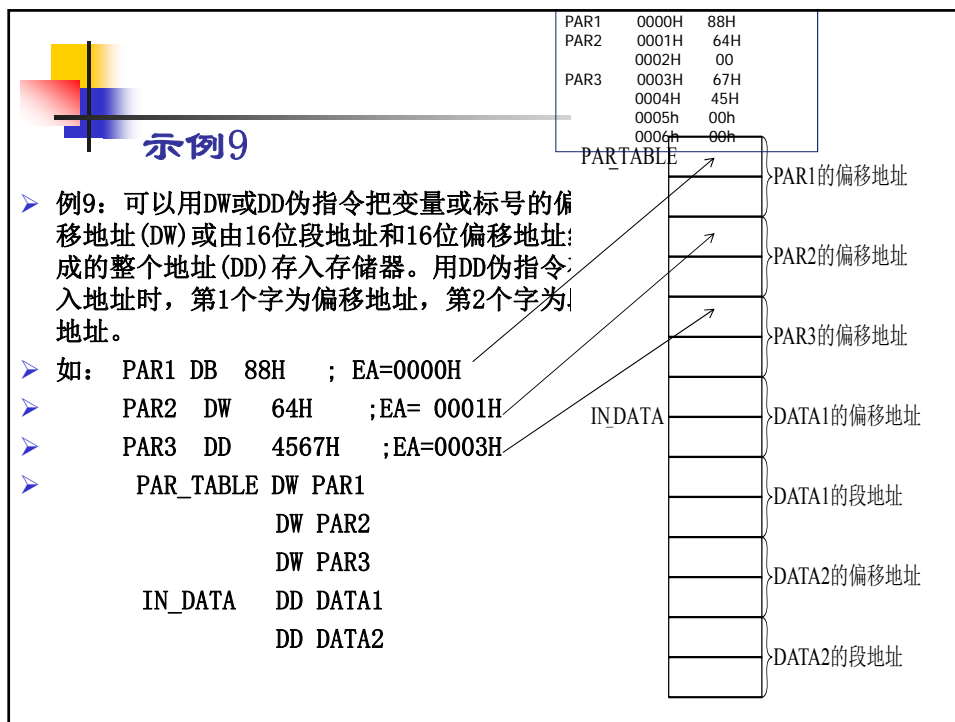
52d

### 示例8

- 例8： DUP操作可以嵌套，例如：
- ARRAY3 DB 100DUP(0, 2 DUP(1, 2), 0, 3)
- 汇编结果如图4.5所示

ARRAY3	00
	01
	02
	01
	02
	00
	03
	⋮
	00
	01
	02
	01
	02
	00
	03

重复100次  
共700个字节



- 例4.1 定义单项变量。**
- **VARN DB 88** ;为变量VARN分配一字节存储单元, 赋的初值是**88**
  - **VARM DW 60 \* 60** ;为变量VARM分配一个字, 初值是**3600**
  - **STRC DW 'AB'** ;为变量STRC分配一个字, 初值是**'AB' = 4142H**
  - **MAXN DD - 2147483648** ;MAXN为双字(2个字), 初值是**- 2147483648**
  - **STRS DD 'AB'** ;STRS的逻辑值= **00004142H**, 物理存储为 **42410000H**
  - **QVAR DQ ?** ;变量QVAR分配4个字单元, 初值不定, 需要时由程序填充
  - **TVAR DT 1024D** ;为变量TVAR分配10个字节, 初值为整数, 即:  
 ;**00000000000000000000400H = 1024**



- 例4.2 定义字符串变量
- **STR1 DB 'A', 'B', 'C', 'D'** ;分配4个字节，初值为字符**ASCII**码值
- **STR2 DB 'ABCDE'** ;等价上面变量**STR1**，仅两者书写形式不同
- **ERROR DB 'syndaxerror!', 13, 10** ;定义一串字符、数字



- |  |              |     |
|--|--------------|-----|
|  | <b>VPACK</b> | 01H |
|  |              | 12H |
|  |              | 46H |
|  |              | 19H |
|  |              | 00H |
|  |              | 00H |
|  |              | 00H |
|  |              | 00H |
|  |              | 00H |
|  |              | 00H |
|  | <b>VUNPK</b> | 01H |
|  |              | 00H |
|  |              | 02H |
|  |              | 01H |
|  |              | 06H |
|  |              | 04H |
- 例4.3定义BCD变量
  - **VPACK DT 19461201H** ;分配10个字节的压缩BCD数  
**19461201**
  - ;即压缩BCD数逻辑形式为  
**:00000000000019461201**
  - **VUNPK DB 1, 0, 2, 1, 6, 4**  
;分配6个字节未压缩BCD数  
**461201**
  - ;即未压缩BCD数逻辑形式为  
**:040601020001**



- 例4.4 定义数组或缓冲区变量
- **BUF DB 98 DUP(?)** ;为BUF分配98个字节，初值不定，可由程序填充
- **TABB DB 2 DUP(3 DUP(8), 'A')** ;定义2个字节串，每串3个8和一个 'A'
- **ARRA DW 26 DUP(41H)** ;定义26个字，每个字的内容为41H



- 例4.5 定义协处理器使用的浮点变量
- **FLOAS DD 45.21** ; Microsoft二进制格式的短(32位)浮点数45.21
- **FLOAL DQ 4.521E1** ; IEEE格式的长(64位)浮点数 $4.521 \times 10^1$
- **FLOAR DT 4521.0E - 2** ; 10字节实数 $4521.0 \times 10^{-2}$



- 例4.6 定义指针变量
- `ARRAY DW 4 DUP(4096 DUP(0))` ;定义一片内存区
- `P16EA DW ARRAY` ;定义变量ARRAY的偏移地址
- `P32ES DD ARRAY` ;.8086方式，低位字为ARRAY偏址，高位字为ARRAY段址
- `P32EA DD ARRAY` ;.386方式32位寻址，定义变量ARRAY的32位偏移地址
- `P48EA DF ARRAY` ;.386方式32位寻址，ARRAY的48位指针(偏移:段址)



## 4.2.3 段定义与段管理伪指令

### 1. 定义段语句SEGMENT和ENDS

- 格式:
- 段名 `SEGMENT` [使用类型] [边界类型][连接类型][‘类别’]
- 段名是为该段起的名字，用来指出汇编程序为该段分配的存储区起始位置
- (1) 使用类型
- 只有使用了.386等方式伪指令，使用类型才有用。有两种类型：
  - `USE16`-----该段按照16位寻址，与8088寻址方式相同
  - `USE32`-----该段按照32位寻址，地址指针48位：段地址16位，EA32位
  - 若在段定义是没有给出使用类型，使用了.386后默认的是`USE32`.

### 4.2.3 段定义与段管理伪指令

#### ➤ (2) 边界类型

- 无：即省略边界，默认为节类型**PARA**
- **PARA**：表示本段必须从能被16整除的地址处开始存放，即**段起始地址最低四位必须是0**，称为节地址
- **BYTE**：表示本段起始地址可以从任一地址处开始存
- **WORD**：表示本段要从一个偶数地址处开始存放，即段起始地址的**最低一位必须是0**，称为字地址
- **DWORD**：表示本段要从一个4的倍数地址处开始存放，即**段起始地址的最低两位必须是0**，双字地址，常用于32位寻址
- **PAGE**：表示本段要从能被256整除的地址处开始存放，即**起始地址的最低八位必须是0**，称为页地址。

### 4.2.3 段定义与段管理伪指令

#### ➤ (3) 连接类型:组合方式有六种类型可供选择

- 无 省略连接类型。本段有自己的段地址，与其他同名段独立
- **PUBLIC** 同名同类段连接。程序连接时将本段与其他同名同类型（可以包括不同的模块）的段连接在一起，组成同一个物理段，具有同一个段地址
- **STACK** 指定堆栈段。Link程序连接后的段为堆栈段，与**PUBLIC**处理相同
- **COMMON** 指明覆盖段。连接时产生一个覆盖段，类别相同的同名段有相同的起始段地址，段的长度选取含**COMMON**段的最大长度
- **MEMORY** 指定存储方式段
- **AT表达式** 指定段地址值表达式。将给定表达式的值作为16位寻址的非代码段的段地址值，可作为变量赋予绝对地址

### 4.2.3 段定义与段管理伪指令

- (4) ‘类别’说明：定义逻辑段时若类别选择项缺省，则表明该类别为空。如果有‘类别’说明，则必须用单引号将类别括起来，类别是任何合法的名字。连接时，LINK程序将类别名相同的段依次连续存放在内存中。

例 4.7 将两个或更多个同名的代码段或者两个到多个同名的数据段组成同一个物理段时，程序中可使用 PUBLIC 连接方式，进行段的连接。

程序段 1	程序段 2
DATAS SEGMENT PUBLIC	DATAS SEGMENT PUBLIC
VAR1 DB 5	VAR2 DW 8
DATAS ENDS	DATAS ENDS
CODES SEGMENT PUBLIC	CODES SEGMENT PUBLIC
⋮	⋮
CODES ENDS	CODES ENDS

高级汇编连接后，将源代码段 1 和源代码段 2 的内容依次存放，并组成一个代码段；同时将数据段 1 和段 2 组成同一个数据段，同样将数据内容依次存放。

例 4.8 将几个模块共享的同名堆栈段变成一个堆栈段，用 STACK 连接方式。

堆栈段 1	堆栈段 2
STACKS SEGMENT STACK	STACKS SEGMENT STACK
DB 128 DUP(?)	DB 256 DUP(?)
STACKS ENDS	STACKS ENDS

经汇编和连接后，初始栈顶 SP 指向两个堆栈的大小值之和。



## 4.2.3 段定义与段管理伪指令

### 2. 置汇编计数器语句ORG

- 格式：ORG 数值表达式
- 功能：将数值表达式的值赋给汇编地址计数器。  
数值表达式的值须为0~65545之间的非负整数。

**\$ 表示当前汇编地址。**

- 例:给汇编地址计数器赋值。

DATA SEGMENT

ORG 10 ; 置\$值为10

VAR1 DW 100H, 200H

ORG \$ + 5 ; 置\$的值为14+5, 即为19

VAR2 DB 1, 2, \$ + 1, \$ + 2

N EQU \$ - VAR2 ; 0017H - 0013H = 04, 不占  
单元

DATA ENDS

000AH	VAR1	00H
000BH		01H
000CH		00H
000DH		02H
000EH		
000FH		
0010H		
0011H		
0012H		
0013H	VAR2	1
0014H		2
0015H		16H
0016H		18H
0017H		

### ORG示例

例1：ORG伪指令示例一。

VECTORS SEGMENT

ORG 10

VECT1 DW 47A5H

ORG 20

VECT2 DW 0C596H

VECTORS ENDS

VECT1	000AH	A5H
		47H
VECT2	0014H	96H
		C5H

则VECT1的偏移地址值为0AH，而VECT2的偏移地址值为14H。

伪指令 ORG 将数值表达式的值置入汇编地址计数器(LC),指定其后的程序或数据存放的起始 EA。

#### 例 4.9

```
VIDEO_SEG SEGMENT USE32 AT 0B800H ;视频缓冲区段址
            ORG      400H
EQUEIP     LABEL    WORD           ;EQUEIP 的绝对地址为 B800:0400H
VIDEO_SEG ENDS
```

#### 例 4.10

```
.386
DATAS_SEG SEGMENT USE16 DWORD PUBLIC 'DATA'
:
ORGAREA   DB      ?
            ORG     $ + 100H ;表示从当前偏移地址再跳过 256 字节处
                                ;开始存放下一变量 VAR 的值
VAR        DB      128 DUP(99)
DATAS_SEG ENDS
CODES_SEG SEGMENT USE16 PARA PUBLIC 'CODE'
:
CODES_SEG ENDS
```

### 3. 指示段寄存器语句(假定语句) ASSUME

格式 1: ASSUME 段寄存器:段名/组名[,段寄存器:段名/组名[,...]]

段名是由 SEGMENT 语句定义的名字,或者组名是由伪指令 GROUP 定义的名字。用来通知汇编程序,源程序中定义的段或组由对应的段寄存器去进行段的寻址。段寄存器可以是 CS、DS、ES、SS 或 FS 及 GS。此格式对于存储器有隐含约定的访问有效。

格式 2: ASSUME 段寄存器:NOTHING

格式 3: ASSUME NOTHING

这两种格式取消前面由 ASSUME 指定的段寄存器与段或组的联系。

#### 例 4.11

```
CODES SEGMENT USE16 PARA PUBLIC 'CODE'
ASSUME CS:CODES, DS:DATAS,ES:DATAS,SS:STACKS
START: MOV     AX,DATAS
        MOV     DS,AX ;实现段址 DATAS 加载到 DS 或 ES
        MOV     ES,AX
        :
ASSUME ES:NOTHING ;取消 ES 段寄存器寻址,再用 ES 则出错
        :
```



## 4.3 宏功能语句

### 4.3.1 宏汇编

- 对程序中多次重复使用的指令序列，可给它取个名字，定义成一个宏，在源程序中书写宏名就代表这个指令序列。汇编时把宏名展开成该指令序列，从而减少了程序的编写量，使源程序更加简练、清晰
- 重点：
  - 宏定义
  - 宏调用
  - 宏展开



## 1. 宏定义

- 格式：

```
    <宏名>  MACRO  [形式参数表]
                ...
                {宏体}
    ENDM
```
- 宏名必须唯一，宏调用时用来代替所定义的宏体(具体内容)，宏体由汇编语言所允许的任何语句(指令或伪指令)组成，它决定了宏的功能
- 形式参数根据宏体需要可以没有，或者有一个或多个；有多个形式参数时相互之间要用逗号或空格或制表符分隔开。调用宏时要依次用对应的实参去取代；
- 宏体中可以定义或调用另一个宏。定义一个宏，相当于增加了一条伪指令，汇编时告诉汇编程序用宏体来代替该名字；
- 宏要先定义，后调用；

## 2. 宏的调用与展开

例 4.12 定义用“INT 21H”软中断指令调用,从键盘输入一个字符带回显(AH=1)或不带回显(保密,AH=7)的宏指令 INPUT1、INPUT7 和 INPUT。

带回显输入

不带回显输入

宏指令 INPUTC 定义

INPUT1 MACRO

INPUT7 MACRO

INPUTC MACRO X ;X 是形参

MOV AH,1

MOV AH,7

MOV AH,X

INT 21H

INT 21H

INT 21H

ENDM

ENDM

ENDM

宏指令必须先定义后调用(引用)。宏指令可以重新定义,也可以嵌套定义。嵌套定义是指在宏指令体内还可以再定义宏指令或调用另一宏指令定义。

INPUTC 1

INPUTC 7

## 2. 宏的调用与展开

- 格式: <宏名> [实际参数表]
- 汇编源程序时,汇编程序要对源程序中所有的宏调用进行展开,即将宏名所定义的指令序列插入到宏调用处
- 宏与子程序(即过程)有所不同
  - 宏调用是把宏体展开,程序中有几处宏调用就展开几次,源程序代码长,不节省存储空间,但展开后执行速度快,节省运行时间
  - 子程序调用不展开子程序代码(子程序仍存储在原处),但改变程序流程,由主程序调用处转入子程序去执行,执行完毕再返回主程序调用处继续执行,子程序调用前后要保护和恢复现场,因此执行时间长,但节省存储空间

## 2. 宏的调用与展开

- 在参数的类型与顺序应与宏定义时的形式参数一致。
- 宏展开时，若实参数的个数多于形式参数，则忽略多余的实在参数；若实参数的个数少于形式参数，则多余的形式参数为空
- 宏定义只是告诉MASM用一个名字来表示一段语句序列，其本身不被汇编
- 宏指令的参数非常灵活，可以出现在指令的操作数或操作码部分，MASM在宏展开时对参数进行文本替换。
- 宏定义可以写在程序的任何地方，但习惯上总是把宏定义写在程序的最前面

**例 4.13** 定义一条 INOUT 宏指令,在程序需要的地方既可以引用它输入一串字符,也可以引用它显示一串提示字符。此外程序也可调用上例中的宏指令。

```
INOUT MACRO X,Y
    MOV     AH,X
    LEA     DX,Y
    INT     21H
ENDM

DATAS SEGMENT
FNOEND DB      ' End of file,no END directive. ',13,10,' $ '
KEYBUF DB      10,11 DUP(?)
DATAS ENDS
CODES SEGMENT
    :
    INOUT  10,KEYBUF    ;输入一串字符的宏指令调用
    INOUT  9,FNOEND     ;显示一串提示符的宏指令调用
    INPUT7                      ;键盘输入一个字符不带回显
    INPUTC 7
```

(2) 宏展开 汇编程序在遇到宏指令定义时将其登录名字并加入宏指令定义库。当汇编程序检查到在程序中有引用该宏指令时,才把对应的宏指令体调出来在该宏指令的位置进行汇编处理,这个过程称为宏扩展(或宏展开)。宏指令扩展后,在宏指令调用的地方自动留下用实参替换形参的宏指令体语句。

注意:在高级汇编 MASM6.1X 汇编生成列表(.LST)时,每行中间列用符号“1”作为标志。本节为说明是宏展开生成的语句,故在语句的左边仍用符号“1”标志。例如上述INOUT宏指令调用后,宏展开生成的语句如下:

```
1    MOV    AH,10
1    LEA    DX,KEYBUF
1    INT    21H
1    MOV    AH,9
1    LEA    DX,FNOEND
1    INT    21H
```

这里实参是以整体去替换形参的整体(即对应符号的整体代替)。如果只希望代替某一符号或以数值(值参)代替形参,则可用特殊宏运算符“&”和“%”。

### 3. 宏运算符&

➤ (1)宏运算符& 用来连接文本或符号。

➤ 程序设计中较常用的是带&宏运算符的形式参数。

➤ 例4.14

➤	<b>FRRGER</b>	<b>MACRO W</b>	<b>FRRGER B</b>
➤	<b>ERROR&amp;W:</b>	<b>PUSH BX</b>	<b>+ ERRORB: PUSH BX</b>
➤		<b>MOV BX, 'A&amp;W'</b>	<b>+ MOV BX,'AB'</b>
➤		<b>JMP ERROR&amp;W</b>	<b>+ JMP ERRORB</b>
➤		<b>ENDM</b>	

#### 4. 宏指令体的特殊处理

- (1) **LOCAL伪指令—用来指定宏内局部符号**
- LOCAL 符号名, . . . , 符号名
- 作用: 在宏展开时, MASM将源文件中的所有局部符号统一按出现次序替换为唯一的标识符(??0000 ~ ??FFFF), 以避免符号的重复定义

例 4.16

```
MACRO X,Y
LOCAL LOOPI,NEXT
LOOPI: MOV AX,Y-1
NEXT:  SUB AX,X      ;AX-X→AX
      JZ  LOOPI      ;若 AX=X,则转向标号 LOOPI
ENDM

:
:
LABELS 0FFH,80H ;宏指令调用
1 ?? 0000: MOV AX,80H-1 ;宏指令扩展
1 ?? 0001: SUB AX,0FFH ;宏指令扩展
      JZ  ?? 0000
```

- (2) **EXITM 伪指令**
- 用来立即终止宏展开, 通常与条件汇编结合使用。MASM忽略EXITM与ENDM之间的所有语句
- (3) **PURGE宏的删除**
- 当宏在程序中调用完之后而再也用不到时, 可用PURGE伪指令删除之, 以释放它所占用的存储空间。
- 格式: PURGE <宏名>[, 宏名…]

## 5. 宏库

- 为避免重复定义的麻烦，可将通用的宏定义集中放在一个单独的文件中，即放在后缀为 .LIB 的宏库文件中，宏库中可以有多宏定义
- 可用任一个编辑程序(如EDIT或NE)，像建立源程序那样来建立宏库。宏库中的宏必须是通用的例行程序或程序段，对宏中使用的寄存器及状态标志要入栈保存，宏定义中的标号要先用Local伪指令定义(限定只在宏内使用)

例 4.17

D:\MASM6>EDIT MACROIO.LIB

```
PRINTP MACRO X,Y
LOCAL LPRINT
PUSHA
MOV CX,X ;X 为打印字符个数
LEA DI,Y ;Y 为打印字符的缓冲区首址
LPRINT: MOV DX,0 ;打印字符
MOV AH,0 ;打印字符
MOV AL,[DI] ;取打印字符
INT 17H ;调用 ROM BIOS 的打印程序
INC DI ;字符缓冲区变址
LOOP LPRINT ;若 CX≠0,则转向 LPRINT 继续打印
POPA
ENDM

INPUT1 MACRO
MOV AH,1
INT 21H
ENDM
```

按 ALT + F 键,再按 S 键将文件存盘。





- 在程序中可用INCLUDE伪指令将宏库中的宏定义复制到该指令所在的位置：
  - INCLUDE [盘符:\路径\] <文件名>[. 扩展名]
  - INCLUDE功能是通知汇编程序把指定的文件拷贝一份，插入到该语句下方，供汇编使用。如INCLUDE MLIB.LIB语句将宏库中的全部宏定义复制出来
  - 一般INCLUDE伪指令也放在源程序的开始部分

### ● 4.3.2 与 4.3.3 不看

...




## 4.4 模块连接伪指令


- 如果一个模块引用了另一个模块定义的符号，则在使用时应说明这些符号是外部符号，在定义时应说明这些符号是全局符号
- 1. 全局符号伪指令
  - 格式：PUBLIC 符号名1，符号名2，...
  - 用来说明该模块定义的可被其他模块引用的符号常量、变量和标号等
- 2. 引用伪指令
  - 格式：EXTRN 符号名1：类型，符号名2：类型，...
  - 引用符号的类型必须与原定义类型一致
  - 类型可为字节、字和常量等

➤ 例4. 23 主模块程序M1由键盘输入字符串至BUF缓冲区后，调用子模块M2来显示BUF的内容。  
M1和M2的源模块程序分别调用宏指令库MACROLIB.LIB中的INOUT宏指令。程序如下：

```
➤ ; MACROLIB.LIB 宏指令库
➤ INOUT MACRO X,Y
➤     LEA DX, X
➤     MOV AH, Y
➤     INT 21 H
➤ ENDM
➤ M1.ASM主模块源程序
➤ .586
➤ INCLUDE MACROLIB.LIB
➤ STACKS SEGMENT USE16 PARA STACK 'STACK'
➤     DW 128 DUP(0)
➤     TOPS LABEL WORD
➤ STACKS ENDS
```

```
➤ EXTRN DISPLAY FAR
➤ PUBLIC BUF
➤ DATAS SEGMENT USE16 PARA PUBLIC 'DATA'
➤     BUF DB 21, 22 DUP(' ')
➤     CR DB 13,10,'$'
➤ DATAS ENDS
➤ CODES SEGMENT USE16 PARA PUBLIC 'CODE'
➤ ASSUME CS:CODES, DS:DATAS, SS:STACKS
➤ START: MOV AX, DATAS
➤         MOV DS, AX
➤         LEA SP, TOPS
➤         INOUT BUF, 10
➤         XOR BX, BX
➤         MOV BL, BUF+1 ;取输入字符的实际个数
➤         MOV BUF[BX+2], 20H ; 用空格代替输入串后的回车符
➤         INOUT CR, 9
➤         CALL FAR PTR DISPLAY ; call调用M2子模块段间过程display
➤         MOV AH, 4CH ; 自动返回MSDOS
➤         INT 21H
➤ CODES ENDS
➤ END START
```

- 
- M2.ASM子模块源程序
  - .586
  - INCLUDE MACROLIB.LIB
  - EXTRN BUF:BYTE
  - PUBLIC DISPLAY
  - CODES SEGMENT USE16 PARA PUBLIC 'CODE'
  - ASSUME CS:CODES,ES:SEG BUF
  - DISPLAY PROC FAR ; 伪指令PROC定义FAR段间过程DISPLAY
  - MOV AX,SEG BUF
  - MOV ES, AX
  - INOUT ES:BUF+2,9
  - RET , 指令RET返回主模块中CALL指令的下一条返回DOS的指令
  - DISPLAY ENDP : 伪指令ENDP结束DISPLAY过程定义
  - CODES ENDS
  - END ;子模块源程序结束

- 
- 多模块的连接操作见书本 P102
  - 上机实习该操作
  - D:\MASM> MASM M1.ASM
  - D:\MASM> MASM M2.ASM
  - D:\MASM> LINK M1+M2;
  - D:\MASM> M1
  - 4.4.3 与 4.4.4 不看



## 本章结束

---

习题：4.2, 4.3, 4.11,  
4.13, 4.14, 4.17



武汉大学

Wuhan University

School of computer, Aiping XU

