



# 汇编语言程序设计

## *Assembly Language Programming*

主讲：徐娟

计算机与信息学院 计算机系 分布式控制研究所

E-mail: [xujuan@hfut.edu.cn](mailto:xujuan@hfut.edu.cn),

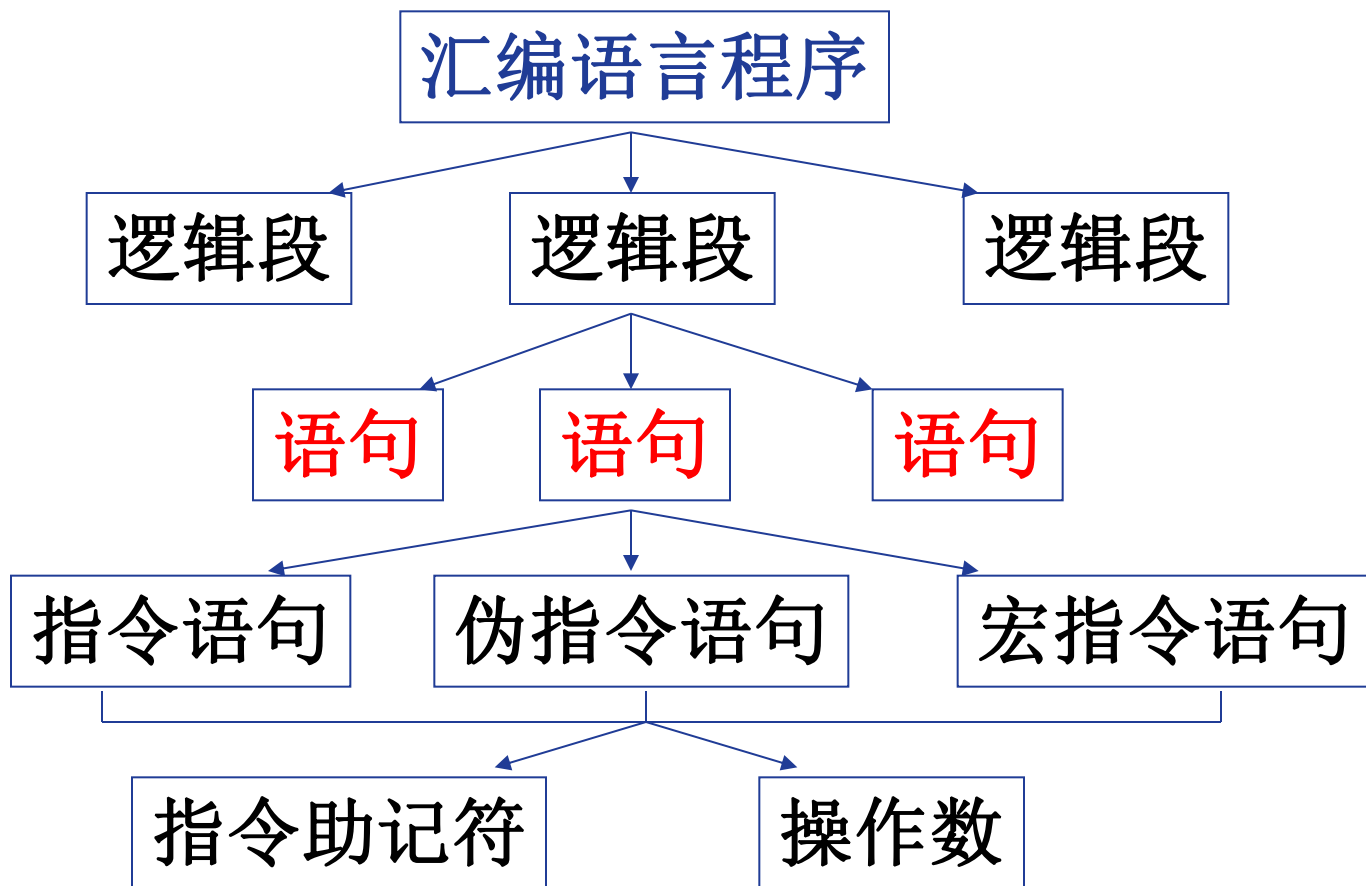
Mobile: 18055100485



# 第三章

## 汇编语言程序格式

## 3.1 汇编语言程序格式





## 典型完整段定义格式 MASM 5.x支持

```
stack    segment stack
dw 512 dup(?)
stack    ends
```

```
data     segment
```

```
...
```

```
data     ends
```

;在数据段定义数据

```
code     segment
```

```
assume cs:code, ds:data, ss:stack
```

```
start:   mov ax, data
```

```
mov ds, ax
```

```
...
```

```
mov ah, 4ch
```

```
int 21h
```

```
...
```

```
code     ends
```

```
end start
```

;在代码段填入指令序列

;子程序代码

# 1 逻辑段

❖ 汇编语言源程序由一个或多个逻辑段组成。

- 一个程序中可以有几个同一类型的逻辑段。
- 需独立运行的程序必须至少有一个代码段。
- 源程序的结束标志——“END”语句

[注] 源程序分段的目的在于程序结构清晰、便于内存分配，寻址方便

一个源程序需要设置几个段应根据具体问题来定。

# 1 逻辑段

- ❖ 程序段（代码段）——主要由指令语句组成，完成源程序的功能。
- ❖ 数据段——定义数据及符号的伪指令组成。
- ❖ 附加段——定义数据及符号的伪指令组成。
- ❖ 堆栈段——定义堆栈伪指令组成。

## 2 语句格式

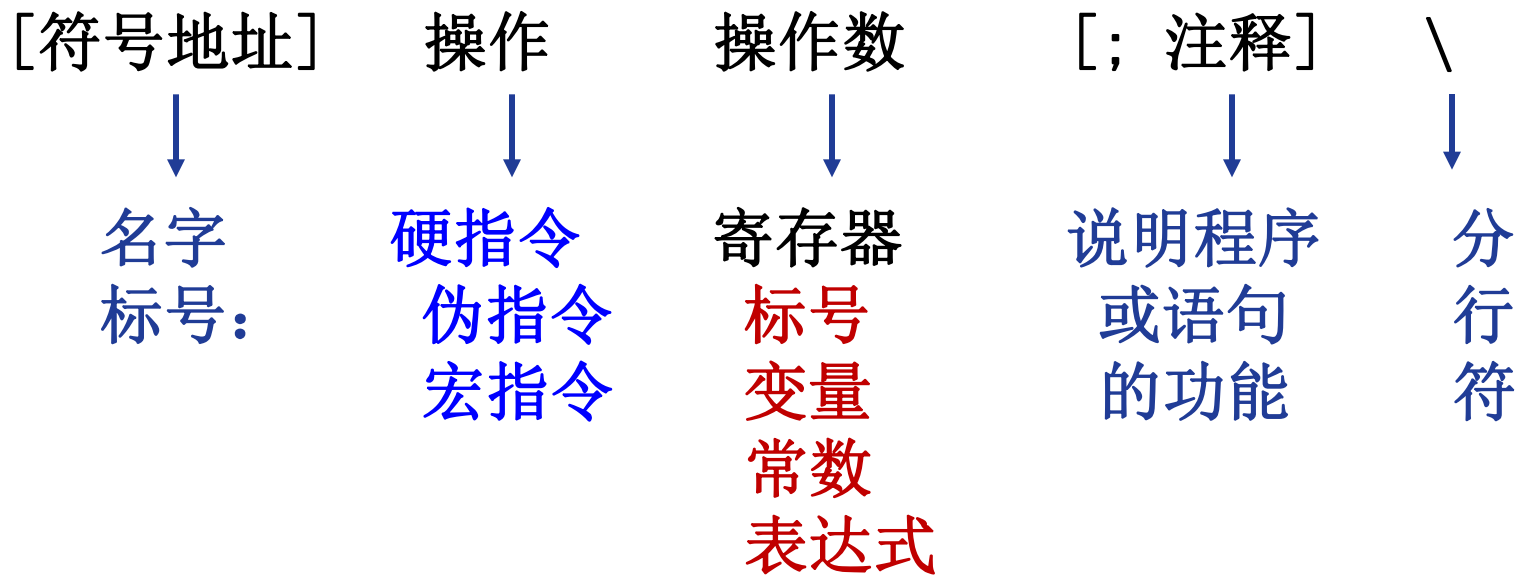
(1) 执行性语句——由硬指令构成的语句，它通常对应一条机器指令，出现在程序的代码段中：

标号：硬指令助记符 操作数, 操作数 ; 注释

(2) 说明性语句——由伪指令和宏指令构成的语句，它通常指示汇编程序如何汇编源程序：

名字 伪/宏指令助记符 参数, 参数, ... ; 注释

## 2 语句格式





# 标号、名字与标识符

❖ **标号**是反映**硬指令**位置（逻辑地址）的标识符，后跟一个冒号分隔

❖ **名字**是反映**伪指令**位置（逻辑地址）和属性的标识符，后跟空格或制表符分隔，**没有一个冒号**

❖ **名字+标号=标识符（Identifier）**

一般最多由31个字母、数字及规定的特殊符号（如 \_、\$、?、@）组成，不能以数字开头。

默认汇编程序**不区别标识符中的字母大小写**

一个程序中，每个标识符的定义是唯一的，还不能是汇编语言采用的**保留字**

# 保留字

❖ **保留字** (Reserved Word) 是汇编程序已经利用的标识符,

主要有:

- 硬指令助记符——例如: MOV、ADD
- 伪指令助记符——例如: DB、EQU
- 操作符——例如: OFFSET、PTR
- 寄存器名——例如: AX、CS
- 预定义符号——例如: @data



**汇编语言大小写不敏感**

# 助记符

- ❖ 硬指令助记符可以是任何一条处理器指令，也可以是一条宏指令
- ❖ 伪指令助记符将在本章和下章学习
- ❖ 定义字节数据和字符串的DB就是伪指令





- ❖ 语句中由分号 “; ” 开始的部分为注释内容，用以增加源程序的可读性
- ❖ 必要时，一个语句行也可以由分号开始作为阶段性注释
- ❖ 汇编程序在翻译源程序时将跳过该部分，不对它们做任何处理

# 分隔符



- ❖ 语句的4个组成部分要用分隔符分开
- ❖ 标号后用冒号，注释前用分号
- ❖ 操作数之间和参数之间使用逗号分隔
- ❖ 其他部分通常采用空格或制表符
- ❖ 多个空格和制表符的作用与一个相同
- ❖ MASM支持续行符 “\”



# 语句格式

[符号地址]



名字  
标号:

操作



硬指令  
伪指令  
宏指令

操作数



寄存器  
标号  
变量  
参数

[; 注释]



说明程序  
或语句  
的功能

\



分行  
符



## 3.1 汇编语言程序格式

## 3.2 参数、变量和标号

### 3.2.1 数值型参数

### 3.2.2 地址型参数

### 3.2.3 变量定义伪指令

### 3.2.4 变量与标号属性

## 3.3 程序段的定义和属性

## 3.2 参数、变量、标号

### ❖ 参数（包括硬指令中的立即数）

- 数值型参数和地址型参数
- 汇编时求值

### ❖ 变量与标号

- 均为程序中定义的符号地址，以名标识并按名访问，汇编时确定偏移地址



## 3.2.1 数值型参数

- ❖ 在源程序语句格式的4个组成部分中，  
参数是指令的操作对象（硬指令时被称为操作数），  
参数之间用逗号分隔
- ❖ 参数根据指令不同可以没有，可以有1个、2个或多个

# 1 常数

❖ 常数（常量）表示一个固定的数值

❖ 它又分成多种形式：

（1）十进制常数：以字母D或d结尾或缺省

（2）十六进制常数：以字母H或h结尾，以字母A~F 开头的十六进制数，前面要用0表达。

（3）二进制常数：由0或1两个数字组成，以字母B或b结尾

（4）八进制常数：以字母Q或q结尾

# 1 常数

## (5) 符号常数（常量）

### ❖ 定义：

- 等价伪指令EQU：给符号名定义一个数值或字符串，

不可重复定义

符号名 EQU 数值表达式

符号名 EQU <字符串>

- 等号伪指令=，可重复赋值

符号名 = 数值表达式

### ❖ 直接用符号名

## 符号定义

```
DosWriteChar equ 2
```

```
CarriageReturn = 13
```

```
CallDOS equ <int 21h>
```

## 符号应用（左边程序段等价右侧的符号形式）

```
mov ah, 2          ;mov ah, DosWriteChar
```

```
mov dl, 13         ;mov dl, CarriageReturn
```

```
int 21h           ;CallDOS
```

## 2. 数值表达式

- ❖ 数值表达式：用常数、符号常数和**算术、逻辑、关系运算符**组成的表达式。如： $(75*2+X)/Y$
- ❖ 汇编程序在汇编过程中计算表达式，最终得到一个数值
- ❖ 程序运行之前，就已经计算出了表达式；所以，程序运行速度没有变慢，但增强程序的可读性

## 2. 数值表达式



### 运算符

#### ❖ 算术运算符

+      -      \*      /      MOD

#### ❖ 逻辑运算符

AND    OR    XOR    NOT

#### ❖ 移位运算符

SHL      SHR

#### ❖ 关系运算符

EQ    NE    GT    LT    GE    LE

#### ❖ 高低分离符

HIGH    LOW    HIGHWORD    LOWWORD

# 算术运算符

- ❖ 算术运算符：+（加）、-（减）、\*（乘）、/（除）、MOD（求模）几种，
- ❖ 它既可以用于数值表达式又可用于地址表达式。
- ❖ 例如：
  - `MOV AL, 25*4 - 50`
  - `MOV DL, 10 MOD 3`

# 逻辑运算符

- ❖ 逻辑运算符：逻辑乘（AND）、逻辑加（OR）、按位加（XOR）、逻辑非（NOT）、
- ❖ 移位运算符：SHL（左移）、SHR（右移）运算。

❖ 例如：

- MOV AL, 34H AND 0FH
- MOV BL, 05H OR 30H
- MOV CX, NOT 00FFH
- MOV DX, 789AH XOR 000FH
- MOV DH, 1 SHL 4
- MOV DL, 0FFH SHR 1



# 关系运算符

## ❖ 关系运算符包括：

相等（EQ），不等（NE），  
小于（LT），小于等于（LE）  
大于（GT），大于等于（GE）

- 关系成立——0FFFFH，如：5 GT 0=0FFFFH
- 关系不成立——0000H，如：5 LE 0=00H

MOV AX, 2 EQ 2 ;0FFFFH→AX

MOV BL, 1 GT 1 ;00H→BL

mov bx, ((PORT LT 5) AND 20) OR ((PORT GE 5) AND 30)

;当PORT<5时，汇编结果为mov bx, 20

;否则，汇编结果为mov bx, 30

# 高低分离符

❖ 取数值的高半部分或低半部分

❖ HIGH、LOW从一个字数值或符号常量中得到高、低字节

`mov ah, HIGH 8765h` ;等价于`mov ah, 87h`

❖ 从MASM 6.0引入的HIGHWORD、LOWWORD取一个符号常量（不能是其他常数）的高字或低字部分

`dd_value equ 0ffff1234h` ;定义一个符号常量

`mov ax, LOWWORD dd_value` ;等价于`mov ax, 1234h`

## 3.2.2 地址型参数

- ❖ 指令参数：数值型 + 地址型
- ❖ 形式：标号和名字（变量名、段名、过程名等）  
或 “+”、“-” 常数或数值表达式组成。
- ❖ 如：DATA+5, VARY[BX]；
  - [注] 含有变量的地址表达式其类型与该变量一致，  
如VARY[BX]，VARY+4与VARY类型一样；



## 3.1 汇编语言程序格式

## 3.2 参数、变量和标号

### 3.2.1 数值型参数

### 3.2.2 地址型参数

### 3.2.3 变量定义伪指令

### 3.2.4 变量与标号属性

## 3.3 程序段的定义和属性

### 3.2.3 变量定义伪指令 (Define)

#### ❖ 功能:

- 定义变量;
- 在内存中分配一组存储单元;
- 并对单元进行初始化。

#### ❖ 格式:

变量名    伪指令助记符    初值表

# 变量名

格式：变量名 伪指令助记符 初值表

- ❖ 变量名为用户自定义标识符，表示初值表首元素的逻辑地址；
- ❖ 用这个符号表示地址，常称为符号地址
- ❖ 变量名可以没有。这种情况，汇编程序将直接为初值表分配空间，无符号地址
- ❖ 设置变量名是为了方便存取它指示的存储单元

# 变量定义伪指令助记符

格式：变量名 **伪指令助记符** 初值表

## ❖ 变量定义伪指令根据申请的主存空间单位分类

- DB——定义字节伪指令
- DW——定义字伪指令
- DD——定义双字伪指令
- DF——定义3字伪指令
- DQ——定义4字伪指令
- DT——定义10字节伪指令

# 初值表

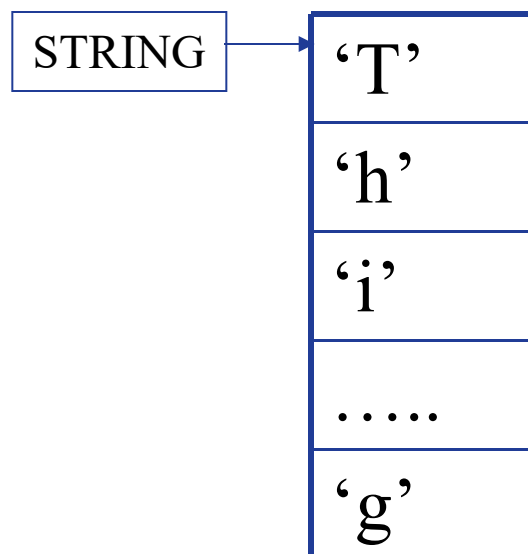
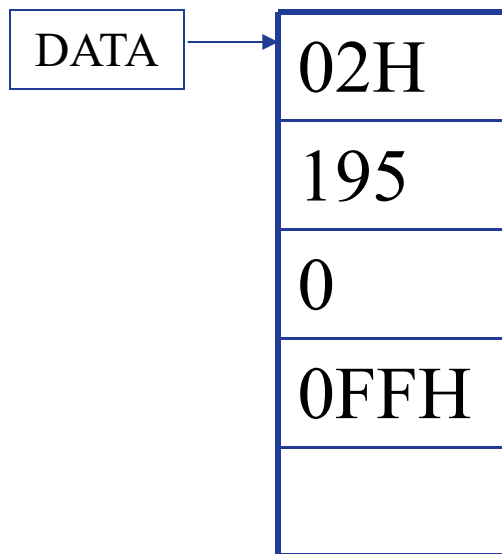
格式：变量名 伪指令助记符 初值表

❖ 常数或一组常数或数值表达式；

- 如：DATA DB 2, 100\*2-5, 0, -1

❖ 一组字符串

- 如：STRING DB 'This is a string' ,
- 其数值是每个字符对应的ASCII码的值
- 注意比较DB “ab” 和 DW “ab”





# 初值表

格式：变量名 伪指令助记符 初值表

## ❖ 变量名或标号名

- ADDR1 DW BLOCK ; BLOCK偏址放在ADDR1单元
- ADDR2 DD BLOCK ; BLOCK的偏址和段址依次存放在ADDR2四字节单元中。

## ❖ 一组“?”

- 只分配空间，不进行初始化
- 例如：BLOCK DW ?, ?  
; 分配两个字，但为随机值

# 初值表

格式：变量名 伪指令助记符 初值表

## ❖ 重复DUP语句 (duplicate)

- 格式： n DUP (重复内容)
- 功能：将DUP后的内容重复定义n次。
- 如：BLOCK DB 3 DUP (0, 1, -1)

等价 BLOCK DB 0, 1, -1, 0, 1, -1, 0, 1, -1

- DUP语句中可以包含DUP语句。

BLOCK

00H

01H

0FFH

00H

01H

0FFH

00H

01H

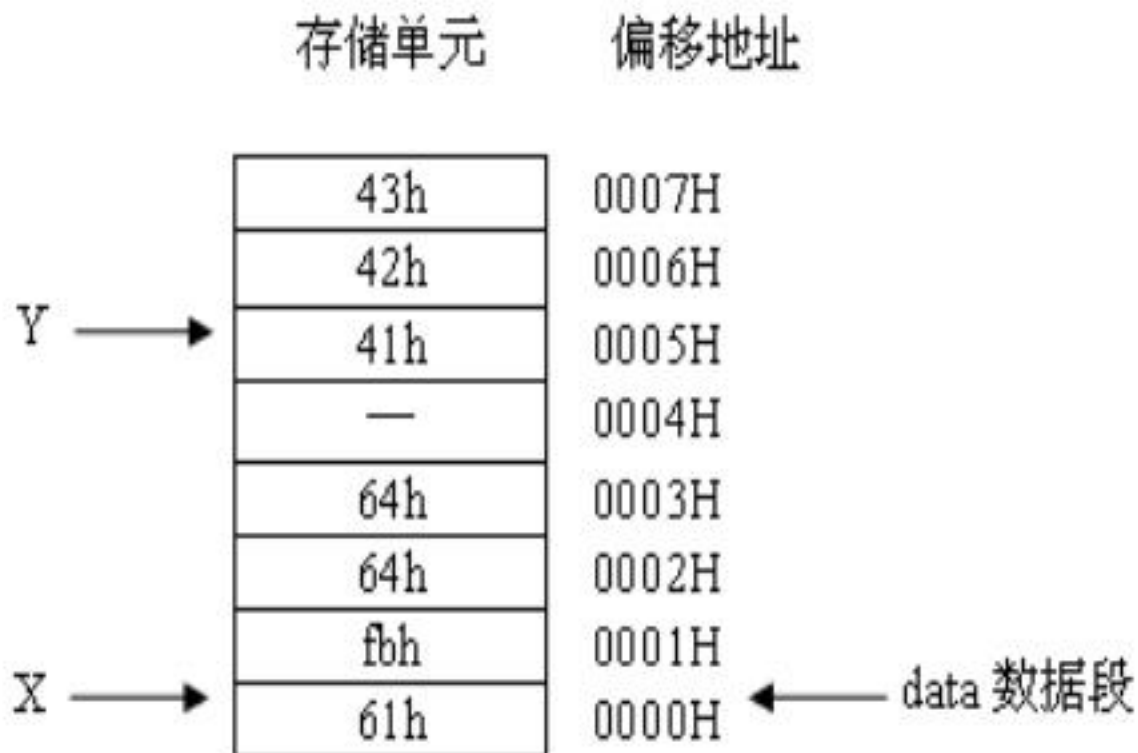
0FFH



.data ; 数据段

字节单元定义实例

X db 'a', -5  
db 2 dup(100), ?  
Y db 'ABC'



❖ 汇编语言强类型！

❖ 变量有类型！

**BUFFER DW 1234H**

- MOV AL, BUFFER
- MOV AL, BYTE PTR BUFFER
- MOV AX, BUFFER



## 3.1 汇编语言程序格式

## 3.2 参数、变量和标号

### 3.2.1 数值型参数

### 3.2.2 地址型参数

### 3.2.3 变量定义伪指令

### 3.2.4 变量与标号属性

## 3.3 程序段的定义和属性

## 3.2.4 变量和标号属性

### ❖ 变量的属性：

- 段属性——变量的段地址
- 偏移属性——变量的偏移地址
- 类型属性——变量所指单元的类型，字节变量、字变量、双字变量等

### ❖ 标号的属性：

- 段属性——是指定义标号所在段的段地址。
- 偏移属性——是指定义标号处到段地址的距离。
- 类型属性——NEAR型（段内）和FAR型（段间）。

MASM提供操作符对变量和标号的属性进行操作！

## ❖ 取得名字或标号的段地址和偏移地址两个属性

- [ ] 将括起的表达式作为存储器地址
- \$ 当前偏移地址
- : 采用指定的段地址寄存器, cs:[0000]
- OFFSET 名字/标号
  - 返回名字或标号的偏移地址
- SEG 名字/标号
  - 返回名字或标号的段地址

# 地址操作符

## SEG 运算符

- 格式：SEG 变量或标号
- 功能：分离出其后变量或标号所在段的段首址。

例如：

- MOV AX, SEG ARR
- MOV DS, AX

## OFFSET运算符

- 格式：OFFSET 变量或标号
- 功能：分离出其后变量或标号的偏移地址。

例如：

- MOV BX, OFFSET BUF



## ❖ 对名字或标号的类型属性进行有关设置

- 类型名 PTR 名字/标号
- THIS 类型名
- TYPE 名字/标号
- sizeof 变量名
- LENGTHOF 变量名

## ❖ 类型（合成） 操作符

- 类型 PTR 表达式 （使变量具有指定的类型）

例： MOV WORD PTR [BX], 5

JMP FAR PTR LA

- THIS 类型 （操作数具有汇编时的当前逻辑地址，但具有指定类型）

b\_var equ THIS byte

;按字节访问变量b\_var, 但与w\_var的地址相同

w\_var dw 10 dup(0)

;按字访问变量w\_var

f\_jump equ THIS far

;用f\_jump为段间转移



## 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。
- 如果是变量，将返回该变量的类型对应字节数；  
字节、字和双字变量依次返回1、2和4；
- 如果是标号，则返回代表标号类型的数值。  
短、近和远转移依次返回ff01h、ff02h和ff05h

## LENGTHOF运算符

- 格式：LENGTHOF 变量
- 功能：取出变量所含的数据存储单元个数。

## SIZEOF运算符

- 格式：SIZEOF 变量
- 功能：取出变量所含的数据存储区大小。
- 其返回值为： $\text{LENGTHOF 变量} * \text{TYPE 变量}$

# 操作符

例1: ARRAY DW 100 DUP (?)

TABLE DB 'ABCDE'

ADD SI, TYPE ARRAY

; ADD SI, 2

ADD SI, TYPE TABLE

; ADD SI, 1

MOV CX, LENGTHOF ARRAY

; MOV CX, 100

MOV CX, LENGTHOF TABLE

; MOV CX, 5

MOV CX, SIZEOF ARRAY

; MOV CX, 200

MOV CX, SIZEOF TABLE

; MOV CX, 5

# 第3章 汇编语言程序格式



## 3.1 汇编语言程序格式

## 3.2 参数、变量和标号

### 3.2.1 数值型参数

➤ DOS支持的exe程序和com程序

### 3.2.2 地址型参数

➤ 完整段定义格式

### 3.2.3 变量定义伪指令

➤ 简化段定义格式

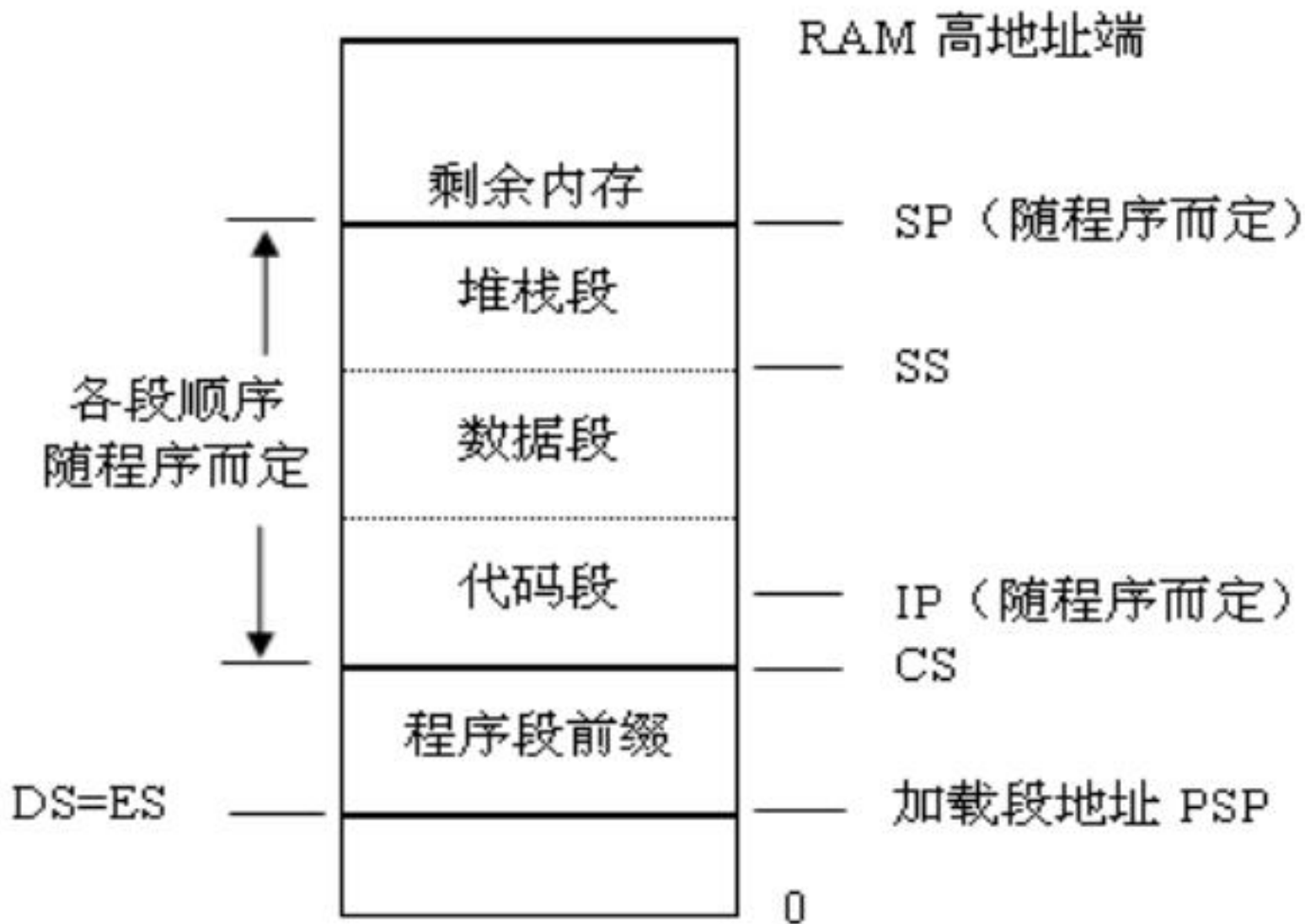
### 3.2.4 变量与标号属性

## 3.3 程序段的定义和属性

## 3.3.1 DOS 程序结构-EXE

- ❖ 利用程序开发工具，生成EXE结构的可执行程序 (\*\*\*.EXE)
- ❖ 它可以有独立的代码、数据和堆栈段，还可以有多个代码段或多个数据段，程序长度可以超过64KB，执行起始处可以任意指定
- ❖ 当DOS装入或执行一个程序时，DOS确定当时主存最低的可用地址作为该程序的装入起始点。此点以下的区域称为程序段。在程序段内偏移0处，DOS为该程序建立一个程序段前缀控制块PSP (Program Segment Prefix)，它占256 (=100h) 个字节；而在偏移100h处才装入程序本身

### 3.3.1 DOS 程序结构-EXE

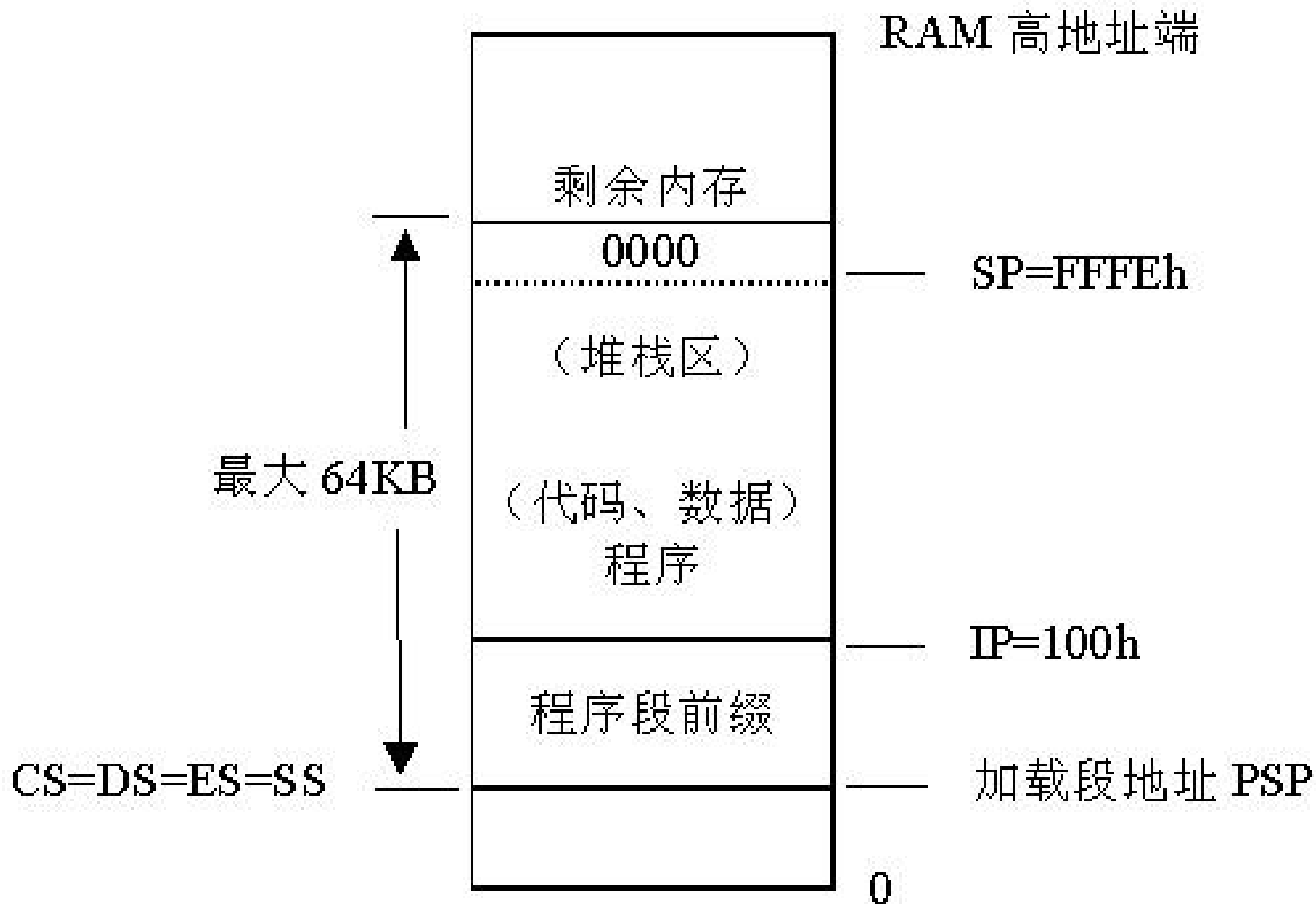




## 3.3.1 DOS 程序结构- com程序

- ❖ COM程序是一种将代码、数据和堆栈段合一的结构紧凑的程序，所有代码、数据都在一个逻辑段内，**不超过64KB**
- ❖ COM文件存储在磁盘上是主存的完全影象，不包含重新定位的加载信息，与EXE文件相比**其加载速度更快，占用的磁盘空间也少**
- ❖ 尽管DOS也为COM程序建立程序段前缀PSP，但由于两种文件结构不同，所以加载到主存后各段设置并不完全一样

### 3.3.1 DOS 程序结构- com程序



## 3.3.2 完整段的定义和属性

- ❖ 起始地址和对准语句
- ❖ 段定义语句
- ❖ 段类型说明语句
- ❖ 汇编结束语句

# (1) 起始地址和对准语句

## ❖ ORG (original)

- 格式：ORG 表达式
- 功能：指定随后指令或者定义数据的偏移地址
- 说明：
  - “ORG” 伪指令可设置程序段、数据段任何位置。
  - 若程序中没有设置“ORG”语句，一般情况每个逻辑的起始地址为0000H。

## (2) 起始地址和对准语句

### ❖ EVEN

- 格式：EVEN
- 功能：偶地址对齐指令。若当前地址是奇数，则加1；

### ❖ ALIGN

- 格式：ALIGN n
- 功能：使随后的数据或者指令起始于n(2, 4, 8...)的倍数地址

### (3) 段定义伪操作

段名    segment        定位    组合    段字    '类别'  
         ...            ;语句序列  
段名    ends

---

- ❖ 完整段定义由SEGMENT和ENDS这一对伪指令实现，SEGMENT伪指令定义一个逻辑段的开始，ENDS伪指令表示一个段的结束
- ❖ 如果不指定，则采用默认参数；但如果指定，注意要按照上列次序
- ❖ 段名对外表现为立即数，为该段的段地址

## (4) 段类型说明伪操作

- ❖ 在代码段开始必须用ASSUME伪操作声明段和寄存器之间的关系，格式为：

**ASSUME 段寄存器：段名 [, 段寄存器名：段名, ...]**

- ❖ 通知MASM，建立段寄存器与段的缺省关系；在需要时自动插入段超越前缀。这是ASSUME伪指令的主要功能。
- ❖ 实际上，数据段之所以成为数据段，是由于DS指向它。由于程序运行时可以改变DS的值，使得任何段都可以成为数据段。



**DATA1 SEGMENT**

**X DB 1**

**DATA1 ENDS**

**DATA2 SEGMENT**

**Y DB 2**

**DATA2 ENDS**

**CODE SEGMENT**

**ASSUME CS:CODE,DS:DATA1,ES:DATA2**

**START:**

**MOV AX,DATA1**

**MOV DS,AX**

**MOV AL,X** → **MOV AL,DS:[0000]**

**MOV AX,DATA2**

**MOV ES,AX**

**MOV AH,Y** → **MOV AH,ES:[0000]**

**MOV AH,4CH**

**INT 21H**

**CODE ENDS**

**END START**



- ❖ ASSUME语句位于在程序段开始位置
- ❖ 在ASSUME语句中并没有给段寄存器赋值。

## ❖ DS、ES的初值必须在程序中设置：

- `MOV AX, <段名>`
- `MOV DS/ES/SS, AX`

## ❖ CS与IP的初值不能在程序中显式设置，由系统自动设置为END后指定的起始地址

## ❖ SS与SP的初值

- 可在程序中显式设置：SS同上，
- SP用 `MOV SP, St_TOP`
- 堆栈段定义时给出了属性STACK，则由系统自动设置。
- 其他，则由系统指定堆栈，编译时给出警告错误

## (5) 汇编结束伪指令

### ❖ 格式:

- END [标号]

### ❖ 功能:

- 指示源码到此结束;
- 指示程序开始执行点（标号处）。



- ❖ 源程序分别用两种格式书写
- ❖ 第一种格式从MASM 5.0开始支持
  - 简化段定义格式
- ❖ 第二种格式MASM 5.0以前就具有
  - 完整段定义格式



## 典型完整段定义格式

### MASM 5.x支持

```
stack      segment stack
dw 512 dup(?)

stack
data       segment
...
data       ends
code       segment
assume cs:code, ds:data, ss:stack
start:     mov ax, data
           mov ds, ax
           ...
           mov ah, 4ch
           int 21h
           ...
code       ends
end start
```

;在数据段定义数据

;在代码段填入指令序列

;子程序代码

### 3.3.3 简化段的定义格式

#### 简化段定义格式 MASM 6.x支持

`;example.asm`

`.model small`

`.stack`

`.data`

`...`

**;在数据段定义数据**

`.code`

`.startup`

`...`

**;在代码段填入指令序列**

`.exit 0`

`...`

**;子程序代码**

`end`

### 3.3.3 简化段的定义格式

#### 简化段定义格式 MASM 5.x支持

```
; exampleb.asm
```

```
.model small
```

```
.stack
```

```
.data
```

```
...
```

;在数据段定义数据

```
.code
```

```
start: mov ax, @data
```

```
mov ds, ax
```

```
...
```

;在代码段填入指令序列

```
mov ax, 4c00h
```

```
int 21h
```

```
...
```

;子程序代码

```
end start
```

### 3.3.3 简化段的定义格式

程序存储模型伪指令的格式如下：

.MODEL 存储模型[, 语言类型] [, 操作系统类型] [, 堆栈类型]

存储模型	功 能	适用系统
Tiny（微型）	所有数据和代码都放在一个段内，其访问都为NEAR型，整个程序小于或等于64 KB，并会产生.COM文件	MS-DOS
Small（小型）	所有代码在一个64KB的段内，所有数据在另一个64 KB的段内（包括数据段、堆栈段和附加段）	MS-DOS Windows
Medium（中型）	所有代码大于64KB时可放在多个代码段中，转移或调用可为FAR型。所有数据限在一个段内，DS可保持不变	MS-DOS Windows
Compact（紧凑型）	所有代码限在一个段内，转移或调用可为NEAR型。数据大于64 KB时，可放在多个段中	MS-DOS Windows
Large（大型）	代码段和数据段都可超过64 KB，被放置在有多个段内，所以数据和代码都是远访问	MS-DOS Windows
Huge（巨型）	单个数据项可以超过64KB，其他同Large模型	MS-DOS Windows
Flat（平展型）	所有代码和数据放置在一个段中，但段地址是32位的，所以整个程序可为4 GB。MASM 6.0支持该模型	OS/2 Windows NT



### 3.3.3 简化段的定义格式

简化段伪指令	功 能	注 释
.CODE [段名]	创建一个代码段	段名为可选项，如不给出段名，则采用默认段名。对于多个代码段的模型，则应为每个代码段指定段名
.DATA	创建一个数据段	段名是：_DATA
.DATA?	创建无初值变量的数据段	段名是：_BSS
.FARDATA [段名]	建立有初值的远调用数据段	可指定段名，如不指定，则将以FAR_DATA命名
.FARDATA? [段名]	建立无初值的远调用数据段	可指定段名，如不指定，则将以FAR_BSS命名
.CONST	建立只读的常量数据段	段名是CONST
.STACK [大小]	创建一个堆栈段并指定堆栈段大小	段名是STACK。如不指定堆栈段大小，则默认值为1 KB

# 与简化段定义有关的预定义符号

- ❖ @CODE: 由 . CODE 伪指令定义的段名或段组名。
- ❖ @DATA: 由 . DATA 伪指令定义的段名, 或由 . DATA、  
. DATA?、. CONST 和 . STACK 所定义的段组名。
- ❖ @STACK: 堆栈段的段名或段组名。

❖ 利用MASM 6. x的简化段定义格式，可以非常容易地创建一个COM程序

❖ 遵循的规则：

- 采用TINY模型
- 源程序只设置代码段，无数据、堆栈等段
- 程序必须从偏移地址100h处开始执行
- 数据只能安排在代码段中，注意不能与可执行代码相冲突，通常在程序最后

## 【例】 简化段定义实例

```
. MODEL    SMALL
. STACK    100H           ; 定义堆栈段及其大小
. DATA           ; 定义数据段
. CODE           ; 定义代码段
START:           ; 起始执行地址标号
MOV     AX, @DATA      ; 数据段地址
MOV     DS, AX          ; 存入数据段寄存器
MOV     AX, 4C00H
INT     21H
END      START          ; 程序结束
```

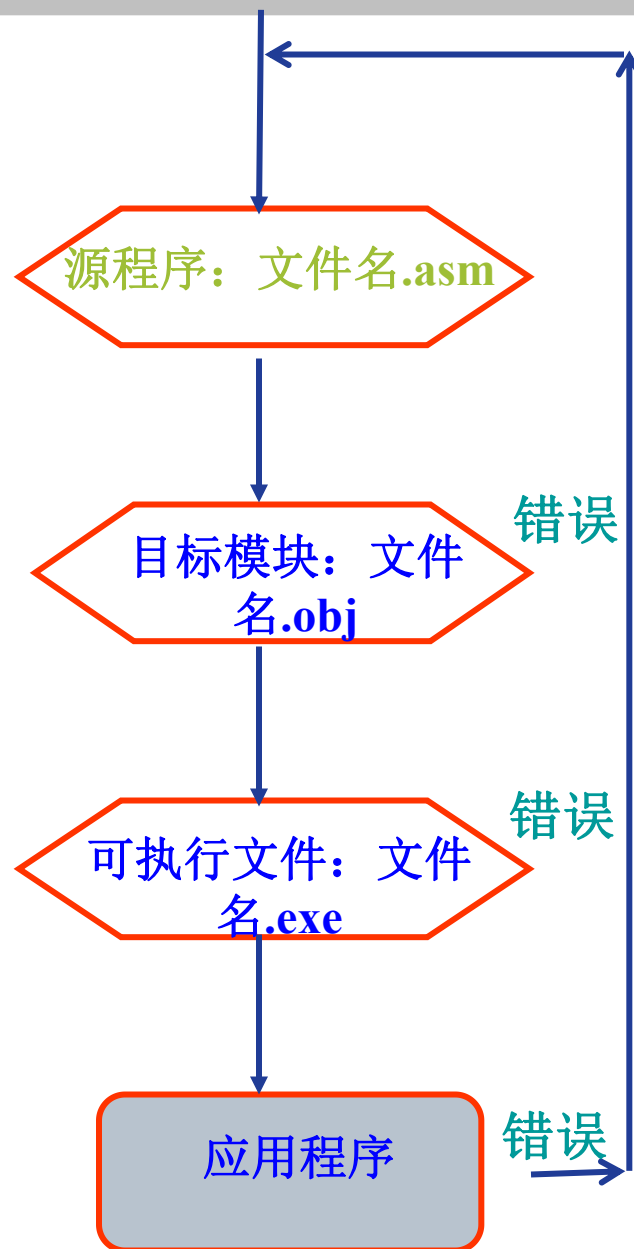
## 3.4 汇编语言的上机过程

汇编语言程序的上机过程

- 1、编写源程序
- 2、将源程序编译为目标程序
- 3、把目标程序连接为 DOS可执行程序

汇编程序的主要功能是：

- 1、报告源程序中的语法错误
- 2、形成目标程序



# 建立汇编语言的工作环境-MASM 5.x



MASM.EXE

LINK.EXE

DEBUG.COM

**使用： MASM myfile.asm ;编译**

**LINK myfile.obj ;连接**

**ML.EXE**

**LINK.EXE**

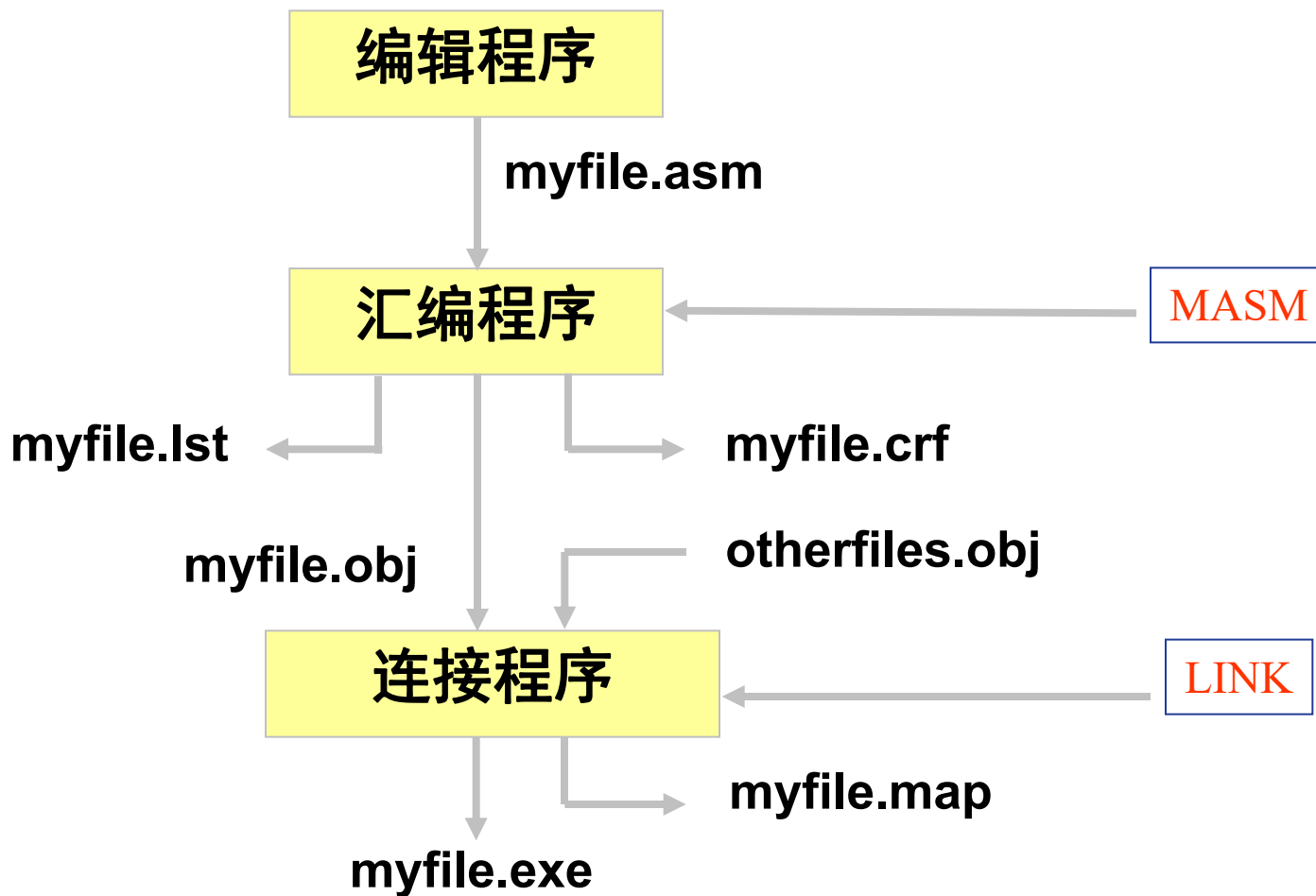
**CodeView.EXE**

**使用： ML myfile.asm ;编译连接**

**ML /c myfile.asm ;只编译**

**ML /Zi myfile.asm ;加入调试信息**

# 编译和连接





## ❖ 程序错误

- 语法错误
- 逻辑错误

## ❖ 用debug/CodeView调试程序

- Debug \*.exe
- -U (机器代码翻译成汇编语言)
  - MOV AX, \*\*\*\*H
- -G (执行指令, 直到遇到断点或程序返回)
- -D \*\*\*\*:0000 (显示)



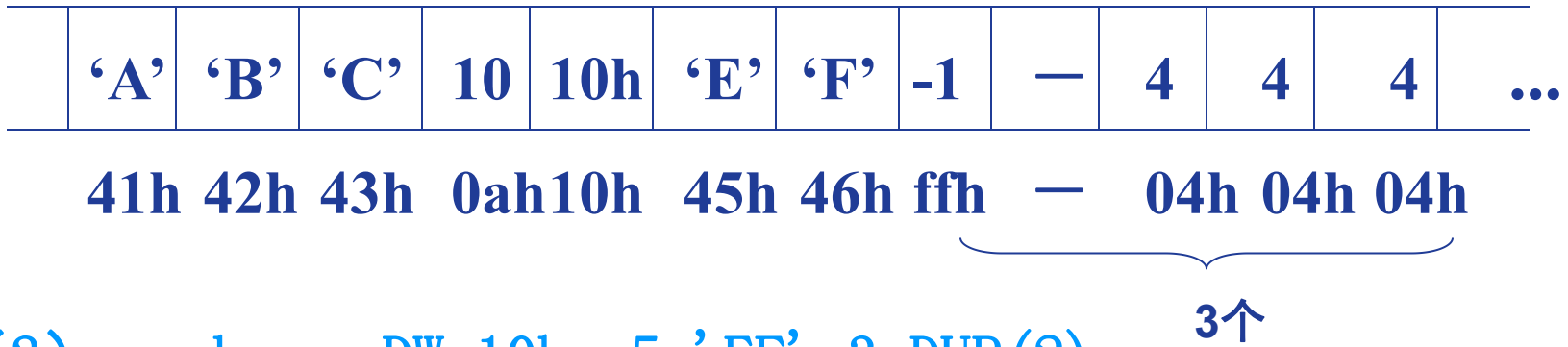
## ❖ 作业

3. 7/3. 15/3. 21 (1) /3. 22 (3)

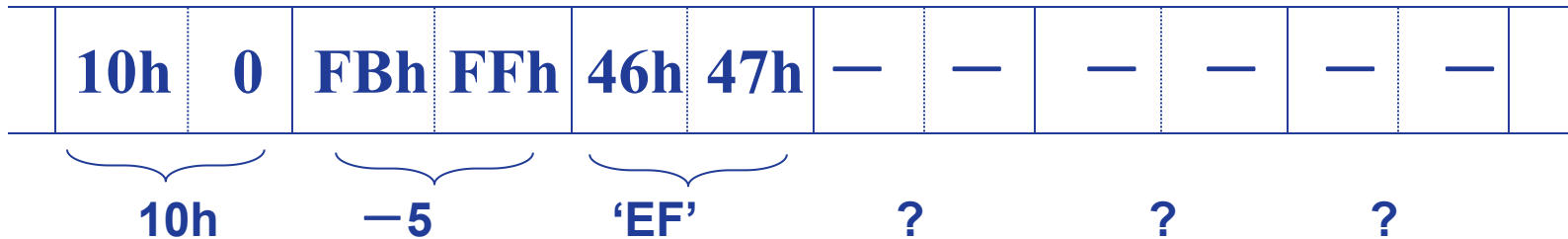



### 习题3.10: 画图说明下列语句分配的存储空间及初始化数据值

(1) `byte_var DB 'ABC', 10, 10h, 'EF', 3 DUP(-1, ?, 3 DUP(4))`



(2) `word_var DW 10h, -5, 'EF', 3 DUP(?)`





习题3.11：请设置一个数据段mydataseg，按照如下要求定义变量：

- (1) my1b为字符串变量：Personal Computer
- (2) my2b为用十进制数表示的字节变量：20
- (3) my3b为用十六进制数表示的字节变量：20
- (4) my4b为用二进制数表示的字节变量：20
- (5) my5w为20个未赋值的字变量
- (6) my6c为100的常量
- (7) my7c表示字符串：Personal Computer

### 习题3.11解答

```
mydataseg    segment
my1b         db 'Personal Computer'
my2b         db 20
my3b         db 14h                ;20h
my4b         db 00010100b
my5w         dw 20 dup(?)
my6c         equ 100                ;my6c = 100
my7c         equ <Personal Computer>
mydataseg    ends
```