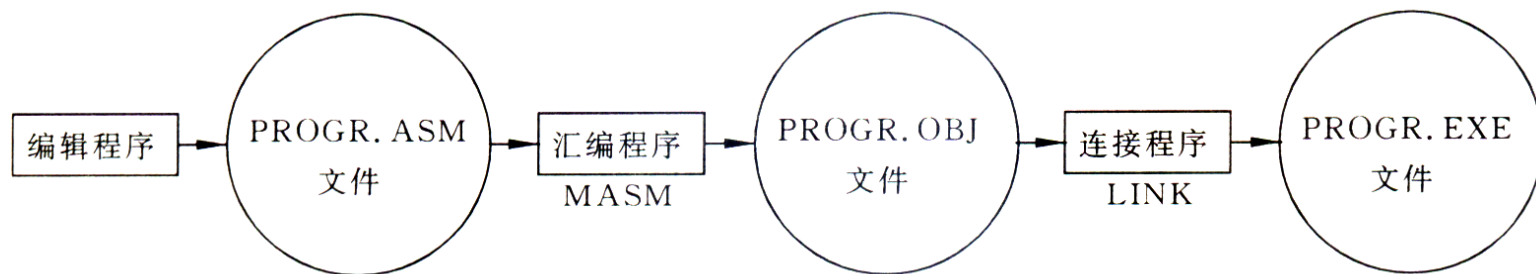


第6章 汇编语言程序格式

• 6.1 汇编语言概述

- **指令**：指使计算机完成某种操作的命令。
- **程序**：完成某种功能的指令序列。
- **软件**：各种程序总称。
- **机器语言**：计算机能直接识别的语言。用机器语言写出的程序称为机器代码。
- **汇编语言**：用字符记号代替机器指令。用汇编语言编写的程序叫汇编语言源程序。
- **汇编程序**：一种翻译程序，把助记符翻译成机器语言。
- 在计算机上运行汇编语言程序的步骤：



6.2 汇编语言语句格式

- 汇编语句格式有4个字段：

[名字] 操作符 操作数； [注释]

1. 一种标识符。

2. 组成：

A~Z, a~z;

0~9;

专用符号 ? . @ _ \$

3. 限制：

1) 第一个字符不能是数字；

2) “.” 必须是第一个字符；

3) 前31个字符有效；

4) 不能为关键字。

4. 类型： 标号-指令符号地址
 变量-数据符号地址

说明程序、指令的功能，
增加程序的可读性

指定参与操作的数据，
或数据所在单元地址。

组成： CPU指令、伪指令、宏指令。

6.3 伪指令

- **伪指令功能：**指示汇编程序完成规定的操作。如选择处理器、定义数据、分配存储区等。
-
- **6.3.1 程序开始和结束伪指令**
- **(3) END 源程序结束伪指令**
- **格式：**END [标号]
- **功能：**表示源程序结束，**不可缺**，源程序最后一条语句。
- **说明：**
 - 1) 标号指示程序开始执行的起始地址。
 - 2) 主程序缺省值为代码段的第一条指令的地址。
 - 3) 多个模块链接，主程序用标号，其他程序不用。

6.3.3 段定义伪操作

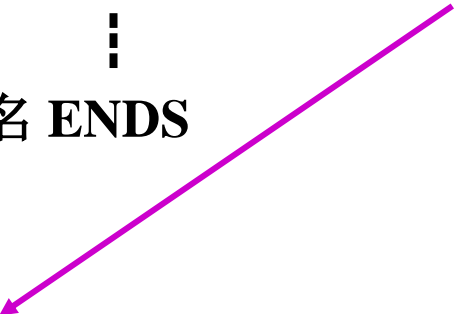
- **段定义：**确定代码组织与数据存储的方式。
- **有2种段定义伪指令：**完整的段定义、简化的段定义。
- 简化的段定义伪指令只能在MASM 5.0以上的汇编语言版本中使用。
- **1. 完整的段定义伪操作**
- **(1) SEGMENT、ENDS 段定义伪指令**
- **格式：**段名 SEGMENT [定位类型] [组合类型] [字长类型] [‘类别’]
• :
• 段名 ENDS
- **功能：**定义段名、段属性，并表示段的开始位置、结束位置。
- **说明：**
 - 1) SEGMENT和ENDS必须成对出现，而且伪指令前面的段名也要相同
 - 2) 段名是段的标识符，指明段的基址，由程序员指定。
 - 3) 一般情况下，选项可以不用，使用默认值。但若需链接程序，就必须使用这些说明。

定位类型

- (1) **SEGMENT、END** 段定义伪指令

- 格式:

段名 **SEGMENT** [定位类型] [组合类型] [字长类型] ['类别']
:
段名 **ENDS**



- 定位类型: 指定段起始地址边界。
- 5种定位类型:
 - 1) **BYTE**: 任意地址。
 - 2) **WORD**: 偶地址, 地址最低1位为0。
 - 3) **DWORD**: 4的倍数地址, 地址最低2位为0。
 - 4) **PARA**: 16的倍数地址, 地址最低4位为0。默认值。
 - 5) **PAGE**: 256的倍数地址, 地址最低8位为0。一页的起点。

组合类型

- (1) **SEGMENT、END** 段定义伪指令

- 格式:

段名 **SEGMENT** [定位类型] [组合类型] [字长类型] ['类别']

⋮

段名 **ENDS**

- **组合类型**: 表示本段与其他模块段之间, 具有相同段名的各段的组合关系。

- **注意3个前提**: 连接时, 类型相同, 段名相同。

- **6种组合类型**:

1) **PUBLIC**: 连接时, 把不同模块中类型相同、段名相同的段连接到同一物理存储段中, 共用一个段地址。连接次序由连接命令指定。即**同名段连接在一起**, 有共同段地址。

2) **STACK**: 用于说明堆栈段。与**PUBLIC**的处理方式一样, 长度为各原有段的总和。**LINK**自动将新段的段地址送**SS**, 长度送**SP/ESP**。若未定义**STACK**类型, 需在程序中用指令设置**SS、SP**。

3) **COMMON**: 连接时, 使类型相同、段名相同的段具有同一个起始地址, 即产生一个**覆盖段**。新段的长度是最长**COMMON**段的长度, 新段的内容取决于依次覆盖的最后内容。

4) **MEMORY**: 表示该段定位在所有段的最下面 (即**地址最大的区域**), 多个**MEMORY**段产生覆盖, 连接时与**PUBLIC**类型同等对待。

5) **PRIVATE**: **独立段**, 不与同名段合并。**默认值**。

6) **AT 表达式**: 指定本段起始地址为“表达式”, 偏移量为**0**, 不能用于代码段。

字长类型

- (1) **SEGMENT、END** 段定义伪指令

- 格式:

段名 **SEGMENT** [定位类型] [组合类型] [字长类型] ['类别']
:
段名 **ENDS**



- 字长类型: 386以后, 说明使用16位寻址还是32位寻址。
- 2种字长类型:
 - 1) **USE16**: 16位寻址, 段长≤64KB, 16位段地址, 16位偏移量。默认值。
 - 2) **USE32**: 32位寻址, 段长≤4GB, 16位段地址, 32位偏移量。

类别

- (1) **SEGMENT、END** 段定义伪指令

- 格式:

段名 **SEGMENT** [定位类型] [组合类型] [字长类型] [**类别**]
⋮
段名 **ENDS**

- **类别**: 引号括起来的字符串。连接时, 类别相同的段 (它们可能不同名) 放在**连续的存储空间**中, 但它们仍然是不同的段。
- 4种类别:
 - 1) **DATA**: 段类别是数据段。
 - 2) **CODE**: 段类别是代码段。
 - 3) **STACK**: 段类别是堆栈段。
 - 4) **EXTRA**: 段类别是附加数据段。

ASSUME

- (2) ASSUME 段分配伪指令

- 格式:

ASSUME 段寄存器名: 段名, 段寄存器名: 段名,

- 功能: 说明某个段使用哪一个段寄存器。

- 说明:

1) 程序段必须用CS, 堆栈段必须用SS。

2) 该语句一般放在代码段最前面。

3) 说明性语句, 除CS外(初始化赋值), 各段寄存器在程序中赋值。

4) 取消语句: ASSUME NOTHING

一般汇编格式举例

- 例6.3.1 段定义例

- DATA1 SEGMENT 'DATA'
- ⋮
- DATA1 ENDS
- DATA2 SEGMENT 'EXTRA'
- ⋮
- DATA2 ENDS
- DATA3 SEGMENT 'DATA'
- ⋮
- DATA3 ENDS
- CODE SEGMENT 'CODE'
- ASSUME CS:CODE, DS: DATA1, ES:DATA3, SS:DATA2
- **START:** MOV AX, DATA1
- MOV DS, AX ; 数据段地址赋给DS
- MOV AX, DATA2
- MOV SS, AX ; 堆栈段地址赋给SS
- MOV AX, DATA3
- MOV ES, AX ; 附加数据段地址赋给ES
- ⋮
- MOV AH, 4CH
- INT 21H ; 返回DOS
- CODE ENDS
- END **START**

6.3.4 地址计数器伪指令

- (1) 地址计数器\$
- 功能：指出汇编地址，是偏移量，记载下一个变量或指令在当前段中的偏移量。
- 每个段开始时， $\$=0$ ，随着汇编过程的进行而增值。每处理一条指令，\$增加一个值，此值是该指令所需的字节数。
- 允许直接引用\$，如 `JNE $+6`，转向JNE指令的首地址加上6。
- \$用于指令时，表示本条指令的第一个字节的地址；
- \$用于其他情况，表示\$的当前值。
- (2) ORG 地址计数器设置（起始地址定义）
- 格式：ORG 数值表达式
- 功能：定义指令或数据的起始地址， $\$ \leftarrow$ 表达式的值。
- 说明：数值表达式取值范围在0~65535之间。

地址计数器伪指令

- (3) **EVEN** 偶数地址定义 (使地址计数器成为偶数)
- 格式: **EVEN**
- 功能: 使下一个变量或指令从偶地址开始。
- 说明: 便于字存储对准。 **EVEN**在代码段中可能多出一个**NOP**语句。

- (4) **ALIGN** 边界定义 (使地址计数器满足边界要求)
- 格式: **ALIGN** n ; n 必须是2的整数幂, $n = 2^i$
- 功能: 使下一个变量或指令从 n 的整倍数地址开始。
- 说明: 保证双字、四倍字对准。当 $n=2$ 时, 即**ALIGN 2**和**EVEN**是等价的。

6.3.5 数据定义伪指令

- 格式:

[变量名] 操作符 操作数 [; 注释]

- 功能: 为操作数分配存储单元, 用变量与存储单元相联系。

- 变量名和注释是可有可无的。

- 操作符: 定义操作数类型。

DB: 一个操作数占有1个字节单元 (8位), 定义的变量为字节变量。

DW: 一个操作数占有1个字单元 (16位), 定义的变量为字变量。

DD: 一个操作数占有1个双字单元 (32位), 定义的变量为双字变量。

DF: 一个操作数占有1个三字单元 (48位), 定义的变量为三字变量。

DQ: 一个操作数占有1个四字单元 (64位), 定义的变量为四字变量。

DT: 一个操作数占有1个五字单元 (80位), 定义的变量为五字变量。

- 操作数: 常数、表达式、字符串、? 等。

例6.3.6 常数与表达式

- 例6.3.6 操作数为常数与表达式
定义形式

```
ORG 200H
DATA1 DB 12H, 2+6, 34H
      EVEN
DATA2 DW 789AH
      ALIGN 4
DATA3 DD 12345678H
DATA4 DW $, 6699H
```

表6.1 例6.3.6的汇编结果

变量名	偏移量	存储单元内容
DATA1→	200H	12H
	201H	08H
	202H	34H
	203H	(即保留原值)
DATA2→	204H	9AH
	205H	78H
	206H	(即保留原值)
	207H	(即保留原值)
DATA3→	208H	78H
	209H	56H
	20AH	34H
	20BH	12H
DATA4→	20CH	0CH
	20DH	02H
	20EH	99H
	20FH	66H

例6.3.7 字符串

- 例6.3.7 操作数是字符串定义形式。
- 包括在单引号中的若干个字符形成字符串，字符串在存储中存储的是相应字符的ASCII码。

```
                ORG 100H  
DATA1          DB  'abcd'  
DATA2          DB  'AB'  
DATA3          DW  'AB'
```

- 当定义的字符串中字符多于2个时，只能使用DB定义，而不能使用DW。汇编程序将DW 'AB'按一个字来处理。

表6.2 例6.3.7的汇编结果

变量名	偏移量	存储单元内容
DATA1→	100H	61H
	101H	62H
	102H	63H
	103H	64H
DATA2→	104H	41H
	105H	42H
DATA3→	106H	42H
	107H	41H

例6.3.8 “？”

- 例6.3.8 操作数是“？”定义形式，
- 此时只分配存储空间，但不定义初值。

```

ORG 400H

DATA1 DB 1, 2, ?, 4
DATA2 DW 5, ?, 6
DATA3 DF ?
DATA4 DB 8
    
```

表6.3 例6.3.8的汇编结果

变量名	偏移量	存储单元内容
DATA1→	400H	01H
	401H	02H
	402H	(即保留原值)
	403H	04H
DATA2→	404H	05H
	405H	00H
	406H	(即保留原值)
	407H	(即保留原值)
	408H	06H
	409H	00H
DATA3→	40AH	(即保留原值)
	40BH	(即保留原值)
	40CH	(即保留原值)
	40DH	(即保留原值)
	40EH	(即保留原值)
	40FH	(即保留原值)
DATA4→	410H	08H

例6.3.9 DUP

- 例6.3.9 操作数用复制操作符DUP定义形式
- 此时表示操作数重复若干次。

ORG 300H

DATA1 DB 2 DUP (12H, 34H, 56H)

表6.4 例6.3.9的汇编结果

变量名	偏移量	存储单元内容
DATA1→	300H	12H
	301H	34H
	302H	56H
	303H	12H
	304H	34H
	305H	56H

例6.3.10 DUP嵌套

- 例6.3.10 操作数用复制操作符
DUP嵌套定义形式。

```
ORG 100H  
DATA1 DB 12H,34H,  
        2 DUP(56H,3 DUP(9AH),78H)
```

表6.5 例6.3.10的汇编结果

变量名	偏移量	存储单元内容
DATA1→	100H	12H
	101H	34H
	102H	56H
	103H	9AH
	104H	9AH
	105H	9AH
	106H	78H
	107H	56H
	108H	9AH
	109H	9AH
	10AH	9AH
	10BH	78H

6.3.6 PROC和ENDP过程定义伪指令

- 格式:

过程名 **PROC** [属性]

⋮ (过程体)

过程名 **ENDP**

- 功能: 用于定义子程序结构, 定义一段程序的入口(过程名)及属性。
- 说明: 过程名是该过程(子程序)的入口, 是CALL的操作数。
- 属性: **FAR**、**NEAR** (默认)

6.3.7 模块连接伪指令

- 用于定义各模块之间的共享信息。
- 共享信息是全局变量，包括：常量（ABS）、变量（BYTE、WORD）、标号（FAR、NEAR）或过程名等。
- **1. PUBLIC 全局符说明**
- 格式：PUBLIC 符号1[, 符号2, ...]
- 功能：说明本模块定义，而其他模块引用的共享信息。
- 例：PUBLIC VAR1, VAR2
- **2. EXTRN 外部符说明**
- 格式：EXTRN 符号1: 类型[, 符号2: 类型, ...]
- 功能：说明其他模块定义，而本模块引用的共享信息。
- 说明：EXTRN说明的信息应是在PUBLIC中已经定义的，否则出错。
- 例：EXTRN VAR1:WORD, VAR2:BYTE

6.4 操作数字段

- 操作数包括：寄存器、变量、标号、常数、表达式。
- 6.4.1 常数
 - 常数：主要用作指令中的直接操作数，也可作为存储变量操作数的组成部分，或者在伪指令中用于给变量赋初值。
 - 常数包括：数值常数、符号常数、字符串常数。
 - 1. 数值常数
 - 数值常数：2进制数B、8进制数Q、10进制数D（默认）、16进制数H。
 - 格式：.RADIX 数值表达式；基数控制伪指令
 - 功能：把默认的基数改变为2~16范围内的任何基数。
 - 2. 字符串常数
 - 字符串常数：在单引号中的若干个字符。
 - 字符串在存储中存储的是相应字符的ASCII码。

6.4.1 常数

3. 符号常数

• (1) EQU赋值伪指令

- **格式:** 符号常数名 EQU 表达式 ; 将表达式的值赋给符号常数
- **说明:** 表达式可以是有效的操作数格式, 也可以是任何可求出数值常数的表达式, 还可以是任何有效的符号 (如操作符、寄存器名、变量名等)。
- **注意:** 只能定义一次。

• 例6.4.1

- DATA1 EQU 88
- AAA1 EQU CX
- DATA2 EQU DATA1+12

• (2) =伪指令

- **格式:** 符号常数名 = 表达式 ; 同EQU伪指令
- **注意:** 可重复定义。

• 例6.4.2

- DATA1 = 88
- ⋮
- DATA1 = DATA1+99

6.4.2 表达式

- 表达式是一个组合序列，包括常数、寄存器、标号、变量。
- **两种形式：**数字表达式、地址表达式。
- **1. 运算符**
- **6种：**算术运算、逻辑运算、移位运算、关系运算、返回值运算、属性运算。
- **(1) 算术运算符：** +、-、*、/、MOD
- **(2) 逻辑运算符：** AND、OR、XOR、NOT
- **(3) 移位运算符：** 左移SHL、右移SHR
- **(4) 关系运算符：** 等于EQ、不等NE、小于LT、大于GT、小于等于LE、大于等于GE。
- 关系运算符的两个操作数的**计算结果应为逻辑值**：结果为真（即关系成立），表示为0FFFFH；结果为假（即关系不成立），则表示为0。

表达式-1

- (5) 返回值运算符
- **返回值运算符**: 返回变量或标号的段地址 (SEG)、返回变量或标号的偏移地址 (OFFSET)、返回变量或标号的类型值 (TYPE)、返回变量的单元数 (LENGTH)、返回变量的字节数 (SIZE)。
- 操作数 SEG 变量/标号 ; 段地址值赋给操作数。
- 操作数 OFFSET 变量/标号 ; 偏移量值赋给操作数。
- 操作数 TYPE 变量/标号 ; 代表变量/标号类型的值赋给操作数。
- 操作数 LENGTH 变量 ; 第一个数占用的单元数赋给操作数。
- 操作数 SIZE 变量 ; 第一个数占用的字节数赋给操作数。
- **TYPE**: 变量DB返回1、DW返回2.....标号NEAR返回-1、FAR返回-2。
- **LENGTH**: 只对**DUP定义**的变量有意义, 返回分配给该变量的元素的个数 (不是字节数), 其他情况均送1。
- **SIZE**: 只对**DUP定义**的变量有意义, 返回分配给该变量的字节数。

$$\text{SIZE} = \text{LENGTH} \times \text{TYPE}.$$

例6.4.3

```

DATA SEGMENT
    ORG 3000H
    AA1 DW 100 DUP(0)
    BB1 DW 1, 2
    CC1 DB 'ABCD'
    DD1 DW 1000 DUP(2, 3)
    EE1 DB 50 DUP(5, 6)
    FF1 DW 1, 2, 100 DUP(?)
    GG1 DD 5 DUP(6 DUP(?))
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
    MOV AH, 4CH
    INT 21H
CODE ENDS
END HH1

```

```

HH1: MOV AX, DATA ; 1000H
      MOV DS, AX
      MOV AX, TYPE BB1 ; 2
      MOV BX, OFFSET AA1 ; 3000H
      MOV CL, TYPE AA1 ; 2
      MOV CH, TYPE CC1 ; 1
      MOV AL, TYPE GG1 ; 4
      MOV DX, LENGTH AA1 ; 100
      MOV AX, SIZE AA1 ; 200
      MOV DX, LENGTH BB1 ; 1
      MOV AX, SIZE BB1 ; 2
      MOV DX, LENGTH CC1 ; 1
      MOV AX, SIZE CC1 ; 1
      MOV DX, LENGTH DD1 ; 1000
      MOV AX, SIZE DD1 ; 2000
      MOV DX, LENGTH EE1 ; 50
      MOV AX, SIZE EE1 ; 50
      MOV DX, LENGTH FF1 ; 1
      MOV AX, SIZE FF1 ; 2
      MOV DX, LENGTH GG1 ; 5
      MOV AX, SIZE GG1 ; 20

```

表达式-2

- (6) 属性运算符
- 1) PTR 属性说明运算符
- 格式: 类型 PTR 变量/标号 ; 临时改变类型属性
- 说明: 变量: 字节BYTE、字WORD、双字DWORD、三字FWORD、四字QWORD、五字TWORD ;
标号: 近类型NEAR、远类型FAR。

• 例6.4.4

```
DATA1 DW    1234H, 5678H
DATA2 DB    99H, 88H, 77H, 66H
DATA3 EQU   BYTE PTR DATA1
MOV  AX, WORD PTR DATA2
MOV  BL, BYTE PTR DATA1
MOV  BL, DATA3           ; 34H→(BL)类型正确
MOV  DX, DATA1+2         ; 5678H→(DX)
MOV  [BX], 8              ; ×
MOV  BYTE PTR [BX], 8     ; 存入字节单元
MOV  WORD PTR [BX], 8     ; 存入字单元
```

表达式-3

- 2) **THIS** 指定类型属性运算符
- **格式**: 变量/标号 **EQU THIS** 类型
- **功能**: 将变量或标号定义成指定的类型。
- **说明**: **THIS** 指定的变量或标号本身并不分配存储单元, 它与紧跟其后的变量或标号只有类型不同, 而段地址和偏移量均相同, 从而便于程序设计。该指令可以指定的类型与**PTR**相同。
- **例6.4.5**

DATA1 EQU THIS BYTE

DATA2 DW 1234H, 5678H ; DATA1与DATA2具有相同的段地址和偏移量, 但他们的类型值分别为1、2。

MOV AX, DATA2 ; 1234H→(AX)

MOV BL, DATA1 ; 34H→(BL)

MOV BH, DATA1+1 ; 12H→(BH)

AA1 EQU THIS FAR

AA2: MOV AX, 100H ; AA1与AA2具有相同的段地址和偏移量, 但他们的类型值分别为FAR、NEAR。

表达式-4

- 3) LABEL定义类型属性运算符
- 格式: 变量/标号 LABEL 类型
- 功能: 将变量或标号定义成指定的类型。
- 说明: LABEL 指定的变量或标号本身并不分配存储单元, 它与紧跟其后的变量或标号只有类型不同, 而段地址和偏移量均相同, 从而便于程序设计。该指令可以指定的类型与PTR相同。

• 例6.4.6

```
AA1    LABEL FAR    ; AA1为段间（即远）转移或调用入口
AA2:    ...          ; AA2为段内（即近）转移或调用入口
        ⋮
AA3    LABEL BYTE
AA4    DW    1234H, 5678H
        MOV  AX, AA4
        MOV  BH, AA3+1
```

表达式-5

- （7）运算符的优先级

- 1) 先计算括号内的表达式。
- 2) 然后计算优先级高的运算符。
- 3) 最后从左到右地对优先级相同的运算符进行计算。

表6.6 运算符的优先级别

运算符的优先级		运 算 符
高→ 低→	1	()、[]、<>、LENGTH、SIZE、WIDTH、MASK
	2	段跨越前缀符:
	3	PTR、OFFSET、SEG、TYPE、THIS
	4	HIGH、LOW
	5	单目+、单目-
	6	*/、MOD、SHL、SHR
	7	+、-
	8	EQ、NE、LT、LE、GT、GE
	9	NOT
	10	AND
	11	OR、XOR
	12	SHORT

表达式-6

- 2. 数字表达式

- 数字表达式**：由常数、变量、标号与一些运算符相组合的序列。它的结果必须是常数。

- 例6.4.7

AAA1 EQU X

DATA1 DB 12H, 34H, 56H, 78H

AAA2: MOV AX, LENGTH DATA1 ; 1

MOV BX, TYPE AAA2 ; -1

MOV DX, (AAA1 LT 3 AND 10H OR AAA1 GE 3 AND 0AFH)

SHL 2

- 当AAA1的值小于3时，AAA1 LT 3=FFH（成立）、AAA1 GE 3=0（不成立），汇编程序将例6.4.7中最后一条指令汇编成：

MOV DX, 40H

- 当AAA1的大于等于3时，AAA1 LT 3=0（不成立）、AAA1 GE 3=FFH（成立），汇编成：

MOV DX, 2BCH

表达式-7

- 3. 标号
- 标号：指令的符号地址。标号后跟着冒号“：”。
- 这个地址一定在代码段中，它可用作转移、调用指令的目标操作数。
- 标号有3种属性：段值、偏移值及类型。
- 段值属性：段值属性是标号所在段的段地址。
- 偏移值属性：偏移值属性是标号的偏移地址，它是从段起始地址到定义标号的位置之间的字节数。
- 类型属性：指出该标号是在本段内引用还是在段间引用。段内引用则类型属性为NEAR，段间引用则类型属性为FAR。

表达式-8

- 4. 变量
- 变量：数据单元的符号地址。
- 5种属性：段值、偏移值、类型、单元数（也称长度）、字节数。
- 段值属性：变量所在段的段地址。
- 偏移值属性：变量的偏移地址。
- 类型属性：变量一个数据的字节数。
- 长度属性：变量在数据区中的单元数。
- 字节数属性：在数据区中分配给该变量的字节数。
- 在同一个程序中，相同的变量或标号名只允许定义一次，否则，汇编时汇编程序指出错误。

表达式-9

- 5. 地址表达式

- 地址表达式：其值是存储器地址，即EA的计算。

- 地址表达式使用规则：

- (1) 变量和标号可以加上或减去某个结果为整数的数值表达式。
- (2) 同一段内的变量和变量、标号和标号之间可以相减。
- (3) 寄存器的内容可作为地址使用，用“[]”表示。方括号“[]”内可以出现一个或两个寄存器，也可加、减某个数值表达式，但不能有变量或标号出现。
- (4) 可以使用段超越前缀，其格式为：

段寄存器/段名：变量/标号/地址表达式

- (5) 地址表达式的书写格式：

MOV AX, [BX+SI+8]

MOV AX, [BX+SI]8

MOV AX, [BX][SI]8

MOV AX, 8[BX+SI]

6.5 汇编语言源程序的结构

- 80x86汇编语言源程序结构特点：

- (1) 一个程序由若干逻辑段组成，各逻辑段由伪指令语句定义和说明。
- (2) 整个源程序以**END**伪指令结束。
- (3) 每个逻辑段由语句序列组成，语句可以是：

指令语句： CPU指令，可执行语句。汇编时译成目标码。

伪指令语句： CPU不执行，提供汇编信息，不产生目标代码。

宏指令语句： 一个指令序列，汇编时产生对应的目标代码序列。

注释语句： 以分号“；”开始，说明性语句，汇编程序不予处理。

空行语句： 保持程序书写清晰，仅包含回车换行符。

- 每个源程序必须有返回操作系统的指令语句，使程序执行完后能自动返回系统。

- DOS环境下有2种程序结构：

- (1) **.COM文件：** 长度限制为一个段长（64KB），在加载过程中没有段重定位，结构紧凑、装入速度快，适合小型程序。
- (2) **.EXE文件：** 长度仅受内存空间限制，在加载过程中需要段重定位，占用盘空间大，装入速度慢，适合中、大型程序。

6.5.1 COM文件结构

- 1. .COM文件结构特点:

- (1) 整个程序（包括数据和代码）在一个段（64KB）内，不准建立堆栈段。
- (2) 段的偏移量从100H开始，且在偏移量100H处是一条可执行指令。
- (3) 必须由END说明起始地址。
- (4) .COM文件连接，所有目标模块必须具有同一代码段名、类别名、公共属性，主模块应具有100H的入口指针并优先连接。
- (5) 程序中子程序属性必须为NEAR。

2. .COM源程序结构 -例6.5.1

- 例6.5.1: 编制在CRT上显示ABCDE的源程序。

```
CODE    SEGMENT                                ; 只指定一个段
        ORG    100H                            ; 设置起始偏移量
        ASSUME  CS: CODE, DS:CODE              ; 只指定一个段
START:  JMP     AA1                            ; 第一条可执行指令存入100H单元
BUF     DB      0AH, 0DH, ' ABCDE'            ; 被显示的数据串
AA1:    MOV     AX, CS
        MOV     DS, AX                        ; 设置数据段地址
        LEA     BX, BUF
        MOV     CX, 7
AA2:    MOV     DL, [BX]
        MOV     AH, 2
        INT     21H
        INC     BX
        LOOP    AA2
        MOV     AH, 4CH; 返回
        INT     21H
CODE    ENDS
        END     START
```

2. .COM源程序结构-例6.5.2

- 例6.5.2: 编制在CRT上显示ABCDE的源程序。

```
CODE SEGMENT ; 只指定一个段
    ORG 100H ; 设置起始偏移量
    ASSUME CS:CODE, DS:CODE ; 只指定一个段
START: MOV AX, CS
    MOV DS, AX ; 设置数据段地址
    LEA BX, BUF
    MOV CX, 7
AA2: MOV DL, [BX]
    MOV AH, 2
    INT 21H
    INC BX
    LOOP AA2
    MOV AH, 4CH; 返回
    INT 21H
BUF DB 0AH, 0DH, 'ABCDE' ; 被显示的数据串
CODE ENDS
    END START
```

3. 加载.COM文件的过程

表6.7 实模式加载.COM文件后的内存映像

工作区		地址	内存内容
		0000: 0000H ⋮ 0000: 03FFH	中断向量表
		0000: 0000H ⋮	DOS常驻区
最大允许 64K字节	程序段前缀 控制块PSP区	XXXX: 0000H	INT 20H, 返回DOS
		⋮ XXXX: 00FFH	⋮
	.COM文件 工作区	XXXX: 0100H	可执行指令
			.COM文件段
			⋮
			堆栈段
			0
		XXXX: 0FFFFH	0
			剩余内存
		RAM最高端	DOS常驻区

加载.COM文件的过程

- .COM 文件由DOS操作系统EXEC功能加载。
- 文件加载时，命令处理程序COMMAND.COM将确定当时内存可用空间的最低端作为程序段起点。在该起点首先设置一个程序段前缀控制块PSP。
- PSP控制块长度为256个字节，首字节处是程序终止退出指令INT 20H。
- 加载.COM文件之后，各个内部寄存器的初值设置：
 - (1) CS、ES、DS、SS = PSP段基址（DOS分配的段起始地址）。
 - (2) IP = 0100H
 - (3) SP = 0FFFEH，或当前可用内存字节数减2（此时可用内存空间不足64K）。
 - (4) 栈顶字含有0000H。
 - (5) BX:CX = .COM文件的长度。
- 注意：因规定.COM文件不能自行设定堆栈，故在目标模块连接时，系统会显示下列信息：

Warning NO Stack Segment

4. .COM文件返回到DOS的方法

- .COM文件执行完毕时，在程序结束处可使用不同的返回DOS方法。

(1) **INT 20H**

(2) **MOV AH, 0**
 INT 21H

(3) **RET**

(4) **MOV AH, 4CH** ; 建议用此方法返回DOS
 INT 21H

(5) **MOV AH, 31H** ; 需给出驻留字节数送DX
 INT 21H

(6) **INT 27H** ; 需给出驻留字节长度加1送DX

6.5.2 .EXE文件结构

- 1. .EXE文件的结构的特点

(1) **多个段**，允许建立若干个不同名的代码段、数据段、附加段、堆栈段。

(2) **空间不限**，程序长度仅受当前内存可用空间的限制。

(3) 程序的**入口随应用而定**，只需起始标号与END语句说明的起始地址一致即可。

(4) 子程序属性可为NEAR、FAR，随段内、段间调用而定。

(5) **可重定位**。连接程序LINK根据被连接的目标模块的连接参数，相应生成一个“控制信息块”（或称“重定位信息块”），并将其安装在程序的前头（俗称“文件头”）。该文件头的大小依程序加载时需重定位的段的指令条数而变化，通常是512字节的整数倍。换言之，至少占1个扇区的长度。

(6) 只有主模块的END语句指出程序入口的起始标号，并至少有一个具有STACK属性的堆栈段（**有堆栈段**）。

2. .EXE源程序结构-例6.5.3

DATA1	SEGMENT	； 数据段定义开始
BUF	DB 0AH, 0DH, ' ABCDE'	； 被显示的
数据串		
DATA1	ENDS	； 数据段定义结束
DATA2	SEGMENT	； 附加数据段定义开始
⋮		
DATA2	ENDS	； 附加数据段定义结束
STACK	SEGMENT	； 堆栈段定义开始
	DW 88 DUP(0)	
STACK	ENDS	； 堆栈段定义结束
CODE	SEGMENT	； 段定义开始
	ASSUME CS:CODE, DS:DATA1, ES:DATA2, SS:STACK ;	
START:	MOV AX, DATA1	
	MOV DS, AX	； 设置DS的值
	MOV AX, DATA2	
	MOV ES, AX	； 设置ES的值
	LEA BX, BUF	
	MOV CX, 7	
AA2:	MOV DL, [BX]	
	MOV AH, 2	
	INT 21H	
	INC BX	
	LOOP AA2	
	MOV AH, 4CH	； 返回
	INT 21H	

• 例6.5.3: 编制在CRT上显示ABCDE的源程序

2. .EXE源程序结构-例6.5.4

DATA	SEGMENT		； 段定义开始
BUF	DB	0AH, 0DH, 'ABCDE'	； 被显示的数据串
DATA	ENDS		； 段定义结束
CODE	SEGMENT		； 段定义开始
	ASSUME	CS:CODE, DS:DATA	； 规定段的性质
ASUM	PROC	FAR	
START:	PUSH	DS	； DS值压栈
	MOV	AX, 0	
	PUSH	AX	； 0压栈
	MOV	AX, DATA	
	MOV	DS, AX	； 设置DS的值
	LEA	BX, BUF	
	MOV	CX, 7	
AA2:	MOV	DL, [BX]	
	MOV	AH, 2	
	INT	21H	
	INC	BX	
	LOOP	AA2	
	RET		； 返回
ASUM	ENDP		； 过程定义结束
CODE	ENDS		； 段定义结束
	END	START	； 源程序结束

• 例6.5.4: 编制在CRT上显示ABCDE的源程序

3. 加载.EXE文件的过程

- .EXE文件由DOS操作系统EXEC功能加载。
- 加载.EXE文件经历三个过程：确定被装模块的长度、段重定位、初始化段和指针寄存器。
- 加载.EXE文件后，对寄存器的初始化：
 - （1）取“文件头”中的栈顶指针值送入SP，并将开始段值送入堆栈段寄存器SS；
 - （2）DS、ES = 程序段前缀PSP的段值；
 - （3）取“文件头”中的代码段开始段值和指令指针初值，分别送入CS、IP。到此，控制转向CS:IP，被加载的程序开始执行。
- 了解.EXEC子功能加载.EXE文件的初始化段和指针寄存器的过程后，就不难明白，在编制程序时，为何无需对CS、IP、SS和SP进行初始化，而要对DS或ES等数据段寄存器进行预置，使其指向程序设置的数据段或附加段。

4. .EXE文件返回到DOS的方法

- DOS加载.EXE文件时，只是DS、ES指向程序段前缀PSP，而CS和SS则分别指向程序定义的代码段和堆栈段。因此，若不做相应的处理，.EXE文件不能直接调用INT 20H或INT 27H来处理程序的结束。
- .EXE文件执行完毕时，可使用不同的返回到DOS方法：
 - (1) MOV AH, 0
 INT 21H
 - (2) RET
 - (3) MOV AH, 4CH ; 建议用此方法返回DOS
 INT 21H
 - (4) MOV AH,31H
 INT 21H

第6章 结 束