第四章　汇编语言程序格式

本章主要内容：

- 汇编语言语句种类及其格式
- 汇编语言数据
- 符号定义语句
- 表达式与运算符
- 程序的段结构
- 过程定义伪指令
- 当前位置计数器$与定位伪指令
- 从程序返回操作系统的方法

不同的汇编程序有不同的汇编语言编程规定。目前支持**Intel 8086/8088**系列微机,常用的汇编程序有ASM、MASM、TASM、OPTASM等 。

本章主要介绍汇编语言程序设计中的一些基本书写格式与语法规则。

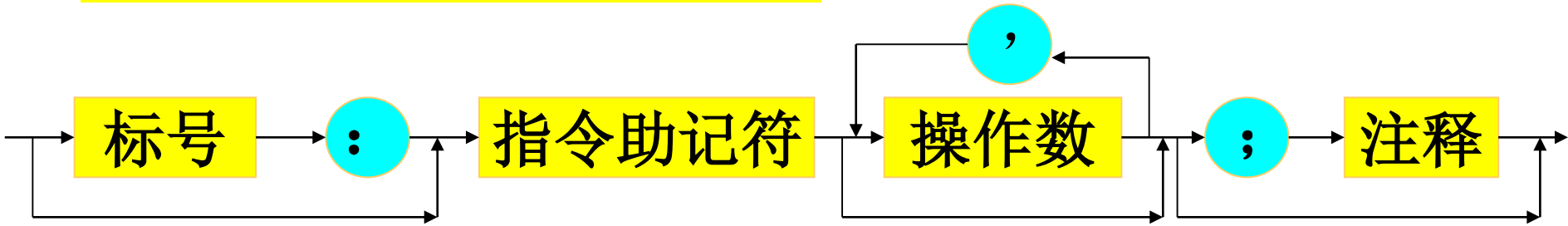
4.1 汇编语言语句种类及其格式

汇编语言的语句可以分为指令语句和伪指令语句

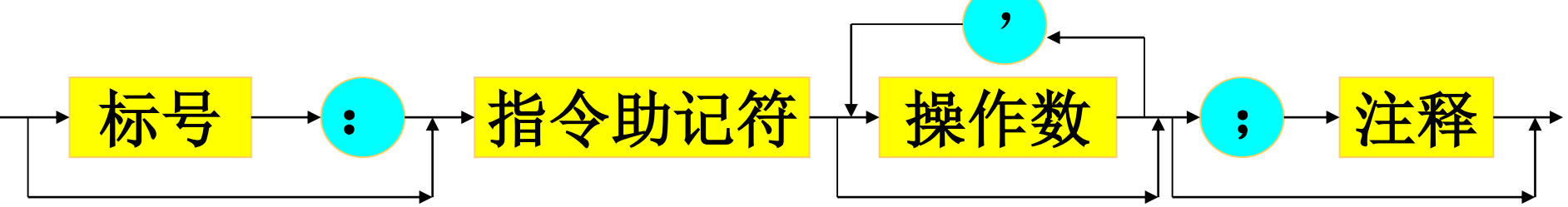
一、指令语句

每一条指令语句在汇编时都要产生一个可供CPU执行的机器目标代码，它又叫可执行语句。

指令语句的一般格式为：



一条指令语句最多可以包含4个字段



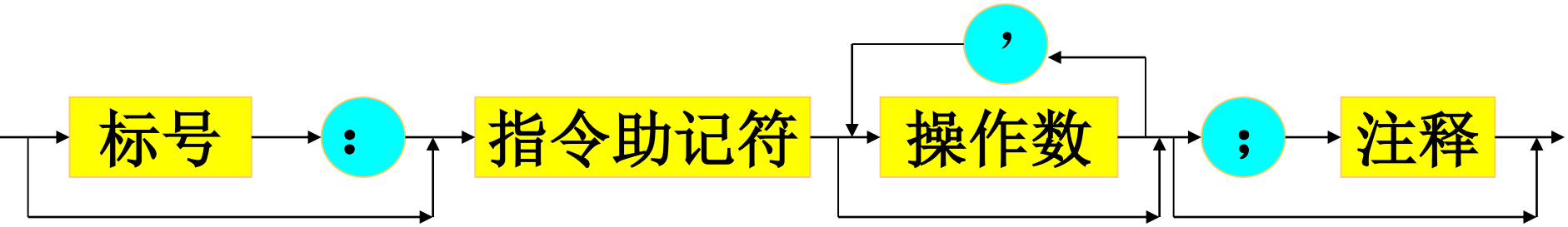
1.标号字段

标号是可选字段，它后面必须有“：”。标号是一条指令的符号地址，代表了该指令的第一个字节存放地址。

标号一般放在一个程序段或子程序的入口处，控制程序的执行转到该程序位置。

在转移指令或子程序调用指令中，可直接引用这个标号。

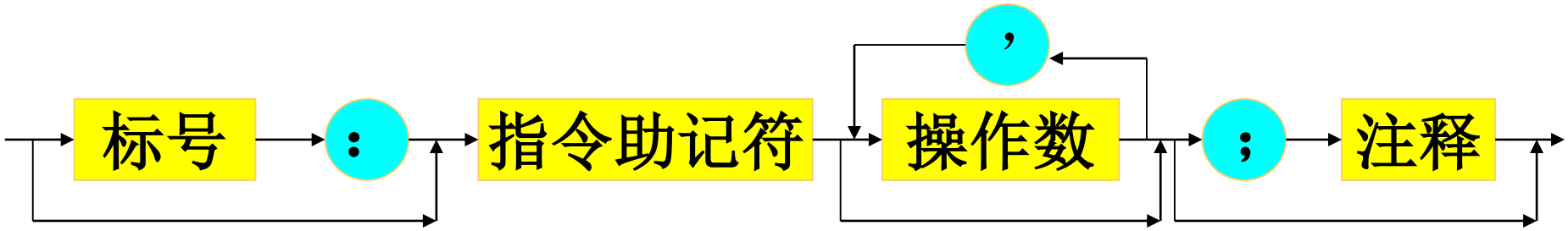
例　ADDR1：MOV　AL，100



2.指令助记符字段

该字段是一条指令的必选项，它表示这条语句要求CPU完成什么具体操作，如MOV、ADD、SHL等。

有些指令还可以在指令助记符的前面加上前缀，实现一定的附加操作。如串操作指令前所加的重复前缀REP（见第7章介绍）等。



3.操作数字段

一条指令可以有一个操作数、两个操作数或者无操作数。

如ADD、MOV指令需要两个操作数，INC、NOT指令只需一个操作数，而CLC指令不需要操作数。

4.注释字段

注释字段为可选项，该字段以分号“;”开始。

它的作用是为阅读程序的人加上一些说明性内容

注释字段不会产生机器目标代码，它不会影响程序和指令的功能。

注释字段可以是一条指令的后面部分，也可以是整个语句行。

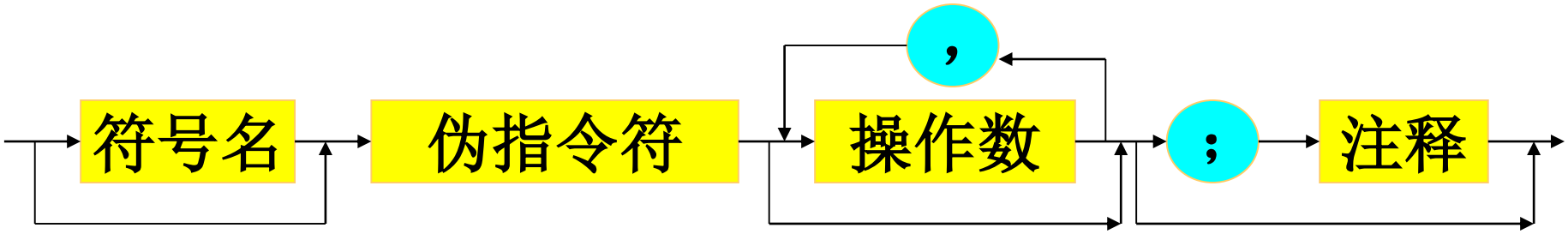
例：LABEL1：ADD AX，BX；功能为AX<=(AX)+(BX)
;后面的程序段将完成一次对存储器的访问

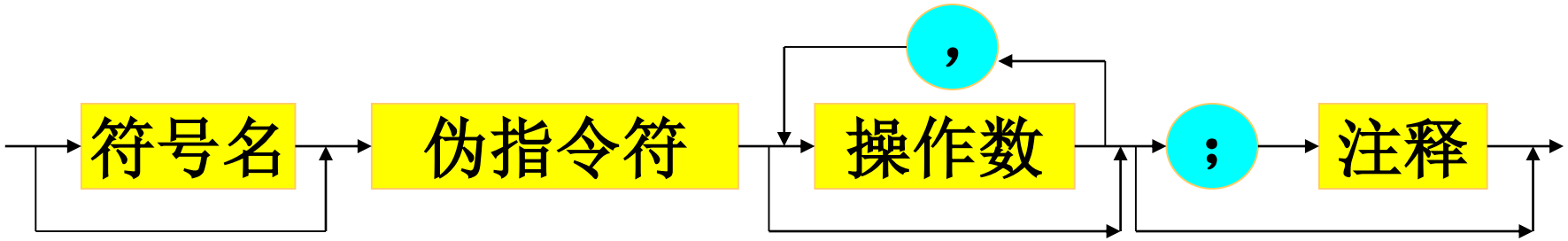
二、伪指令语句

伪指令语句又叫命令语句。

伪指令本身并不产生对应的机器目标代码。它仅仅是告诉汇编程序对其后面的指令语句和伪指令语句的操作数应该如何处理。

一条伪指令语句可以包含四个字段。如下所示：



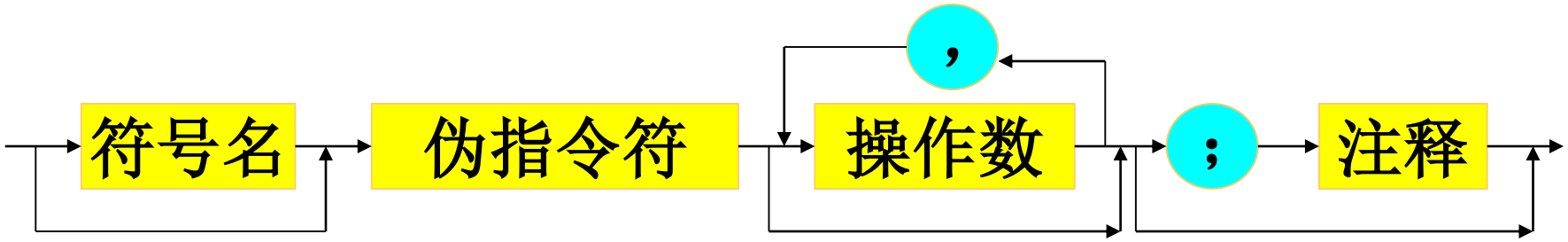


1.符号名字段

该字段为可选项。根据伪指令的不同，符号名可以是常量名、变量名、过程名、结构名和记录名等等。

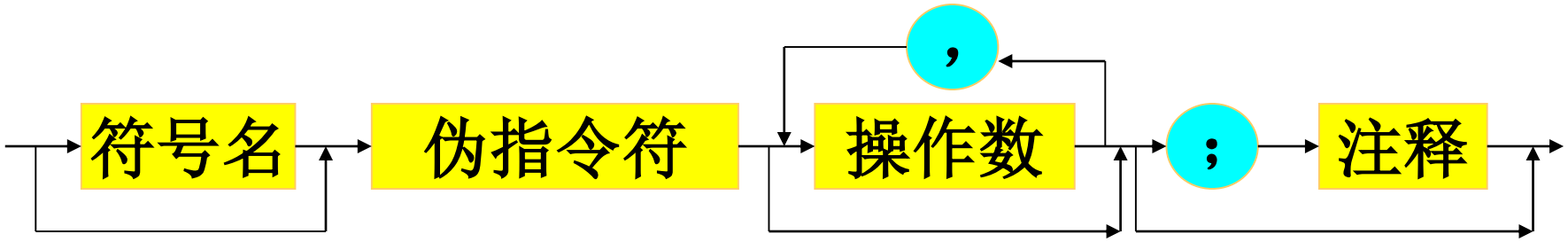
一条伪指令语句的符号名可以作其它伪指令语句或指令语句的操作数，这时它表示一个常 量或存储器地址

注意：符号名后面没有冒号“：”，这是与指令语句的重要区别。



2.伪指令符字段

该字段是伪指令语句的必选项，它规定了汇编程序所要完成的具体操作。本章后面的章节将对各种伪指令作详细介绍。



3.操作数字段

该字段是否需要，以及需要几个是由伪指令符字段来决定。

操作数可以是一个常数（二进制、十进制、十六进制等）、字符串、常量名、变量名、标号和一些专用符号（如**BYTE**、**FAR**、**PARA**等）。

4.注释字段

注释字段为可选项，该字段必须以分号开始。其作用与指令语句的注释字段相同。

三、标识符

指令语句中的标号和伪指令语句中符号名统称为标识符。标识符是由若干个字符构成的。

标识符构成规则：

1.字符的个数为1~31个；

2.第一个字符必须是字母、问号、@或下划线“_”这4种字符之一；

3.从第二个字符开始，可以是字母、数字、@ 、“_”或问号“？”；

4.不能使用属于系统专用的保留字。

保留字: CPU中各寄存器名（如AX、CS等），指令助记符（如MOV、ADD），伪指令符（如SEGMENT、DB）、表达式中的运算符（如GE、EQ）以及属性操作符（如PTR、OFFSET等）

4.2 汇编语言数据

数据是指令和伪指令语句中操作数的基本组成部分。一个数据由数值和属性两部分构成。

在说明数据时不仅要指定其数值，还需说明它的属性，比如是字节数据还是字数据。

在汇编语言中常用的数据形式有：常数、变量和标号。

一、常数

常数在汇编期间其值已完全确定，并且在程序运行过程中，其值不会发生变化。

常数有以下几种形式：

1.二进制数：以字母B结尾，如01001001B

2.八进制数：以字母O或Q结尾，如631Q 254O

3. 十进制数：以字母D结尾，或者没有结尾字母。如2007D、2007。

4. 十六进制数：以字母H结尾，如3FEH，如果常数的第一个数字为字母，为了与标识符加以区别，必须在其前面冠以数字“0”。

5.实数。一般格式为：

±整数部分• 小数部分E ±指数部分

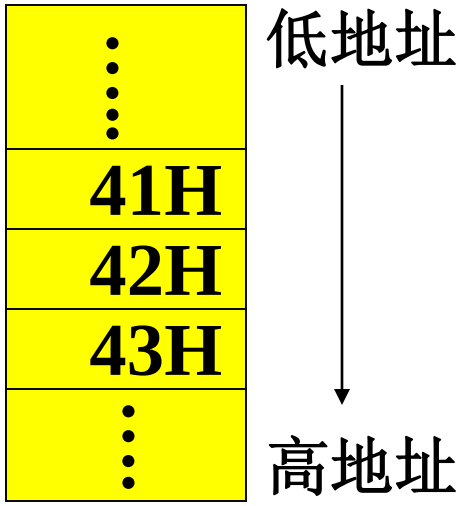
尾数

例　2.134 E +10

汇编程序在汇编源程序时，可以把实数转换为4字节、8字节或10字节的二进制数形式存放。

6．字符串常数：用引号（单引号或双引号）括起来的一个或多个字符，这些字符以它的ASCII码值存储在内存。

例如`B`在内存中为42H，`ABC`为41H 42H 43H。在内存中的存储如图所示。



常数在程序中可以用在以下几种情况：

（1）作指令语句的源操作数

```
MOV   AX，0B2F0H
ADD   AH，64H
```

（2）在指令语句的直接寻址方式、变址（基址）寻址方式或基址变址寻址方式中作位移量。

```
MOV   BX，32H [SI]
MOV   0ABH [BX]，CX
ADC   DX，1234H [BP][DI]
```

（3）在数据定义伪指令中使用

```
DB   10H
DW   3210H
```

二、变量

变量用来表示存放数据的存储单元，这些数据在程序运行期间可以被改变 。

程序中以变量名的形式来访问变量，因此，可以认为变量名就是存放数据的存储单元地址。

1.变量的定义与预置

定义变量就是给变量在内存中分配一定的存储单元。也就是给这个存储单元赋与一个符号名，即变量名，同时还要将这些存储单元预置初值。

定义变量使用数据定义伪指令 **DB、DW、DD、DQ和DT等。**

变量定义的一般格式：

变量名	DB	表达式1，表达式2……；	定义字节变量
	DW		定义字变量
	DD		定义4字节变量
	DQ		定义8字节变量
	DT		定义10字节变量

其中表达式1、表达式2是给存储单元赋的初值。

例如：

```
VAR_DATA    SEGMENT
DATA1    DB     12H
DATA2    DB      20H,30H
DATA3    DW     5678H
VAR_DATA     ENDS
```

当变量被定义后，就具有了以下三个属性：

（1）段属性

它表示变量存放在哪一个逻辑段中。

例如上面例子中的变量DATA1、DATA2和DATA3三个变量都存放在VAR-DATA逻辑段中。

（2）偏移量属性（OFFSET）

它表示变量所在位置与段起始点之间的字节数。

如上述例子中，变量DATA1的偏移量为0，DATA2为1，DATA3为3。

段属性和偏移量属性就构造了变量的逻辑地址

（3）类型属性

它表示变量占用存储单元的字节数。其中DB伪指令定义的变量为字节，DW定义的变量为字，DD定义的为双字（4字节），DQ定义的为4字，DT定义的为5字。

在变量的定义语句中，给变量赋初值的表达式可以使用下面4种形式：

（1）数值表达式

例如：DATA1 DB 32，30H
DATA1的内容为32（20H），DATA1+1单元内容为30H.

（2）？表达式

不带引号的问号“？”表示可以预置任意内容。

例如：**DA-BYTE DB ？，？，？**

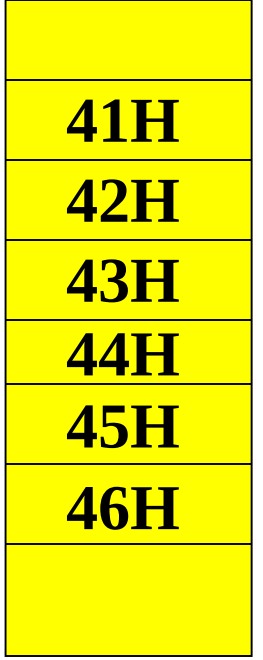
表示让汇编程序分配三个字节存储单元。这些存储单元的内容的值为任意值。

（3）字符串表达式

对于DB伪指令，字符串为用引号括起来的不超过255个字符。给每一个字符分配一个字节单元。字符串按从左到右，将字符的ASCII编码值以地址递增的排列顺序依次存放。

例如：STRING1 DB ‘ABCDEF’

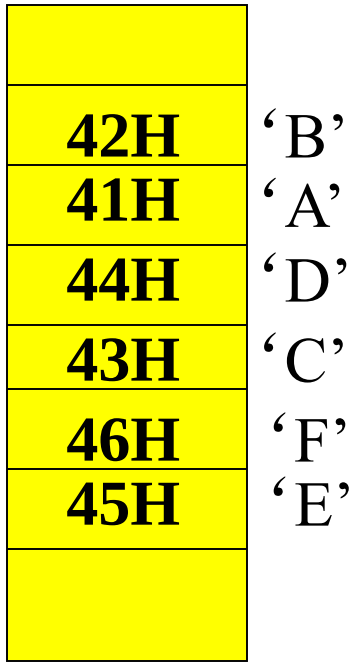
STRING1



对于DW伪指令可以给两个字符组成的字符串分配两个字字节存储单元。

注意：两个字符的存放顺序是前一个字符放在高地址，后一字符放低地址单元。

STRING2

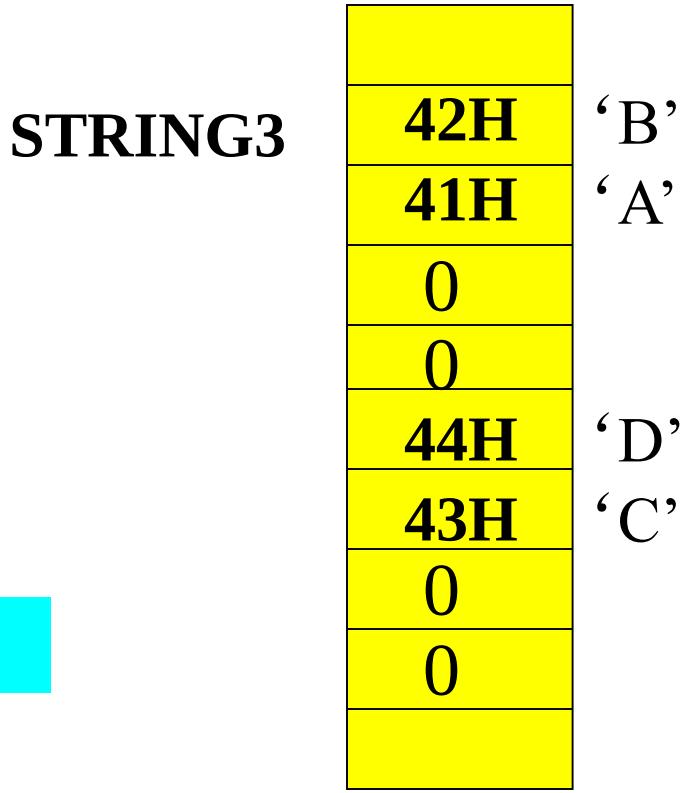


例如：STRING2 DW ‘AB’, ‘CD’, ’EF’

对于DD伪指令，只能给两个字符组成的字符串分配4个字节单元。

两个字符存放在较低地址的两个字节单元中。存放顺序与DW伪指令相同，而较高地址的两个字节单元存放0。

例如：STRING3 DD ‘AB’, ‘CD’



注意：DW和DD伪指令不能用两个以上字符构成的字符串赋初值，否则将出错。

（4）DUP表达式

DUP称为重复数据操作符。

使用DUP表达式的一般格式为：

变量名 { DB / DW / DD } 表达式1 DUP (表达式2)

其中：表达式1是重复的次数，表达式2是重复的内容。

例如：DATA_A DB 10H DUP(?)

DATA_B DB 20H DUP('AB')

分配16个字节单元

分配20H*2=40H个 字节，其内容为重复字符串 'AB'。

DUP还可以嵌套使用，即表达式2又可以是一个带DUP的表达式。

例如：DATA_C DB 10H DUP(4 DUP(2),7)
重复10H个数字序列“2，2，2，2，7”，共占用10H*5=50H个字节。

2.变量的使用

（1）在指令语句中引用

在指令语句中直接引用变量名就是对其存储单元的内容进行存取

```
例如：DA1  DB  0FEH
      DA2  DW  52ACH
      ……
      MOV  AL,DA1   ;将0FEH传送到AL中
      MOV  BX,DA2   ;将52ACH传送到BX中
```

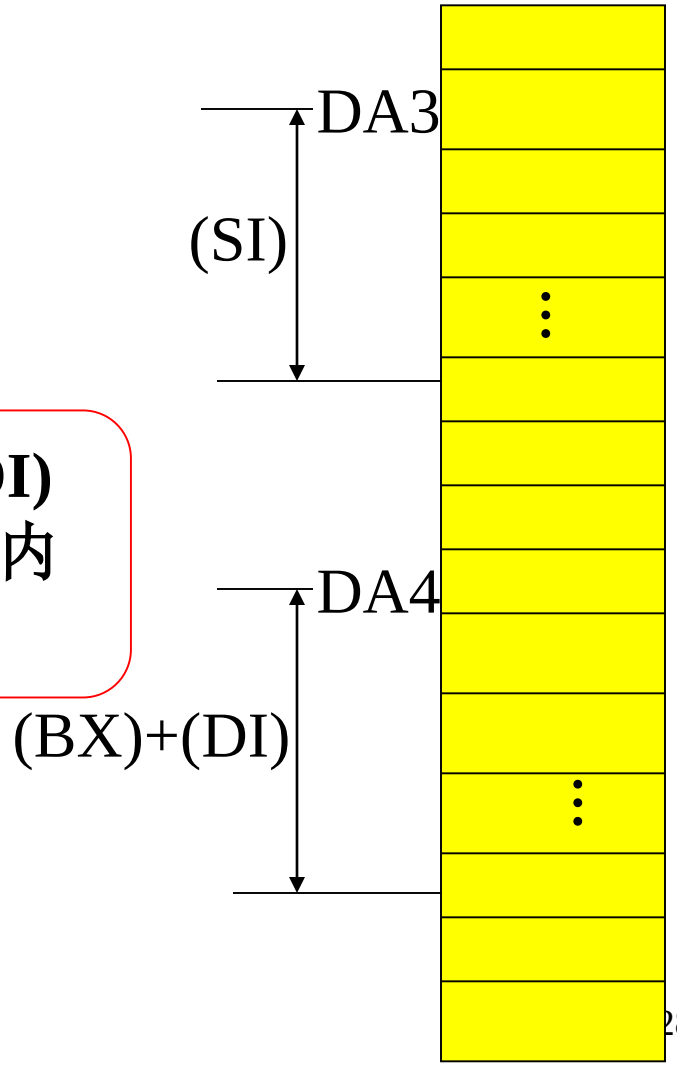
当变量出现在变址（基址）寻址或基址变址寻址的操作数中时表示取用该变量的偏移量。

例如：

```
DA3  DB  10H  DUP(?)
DA4  DW  10H  DUP（1）
MOV  DA3[SI]，AL
ADD  DX，DA4[BX][DI]
```

将从DA4开始再偏移(BX)+(DI)的字存储单元的内容与DX的内容相加，结果送回DX中。

将AL的内容送入从DA3 开始再偏移(SI)的存储单元中



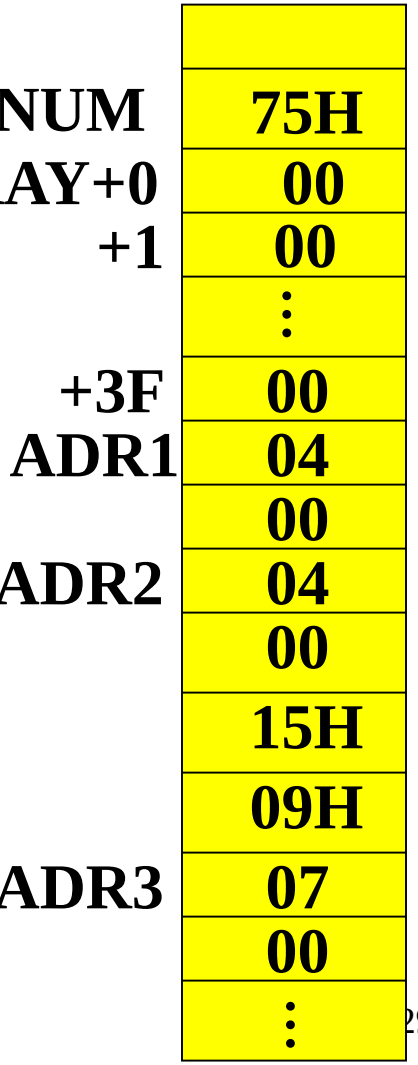
（2）在伪指令语句中引用

```
NUM     DB   75H
ARRAY   DW   20H DUP(0)
ADR1    DW   NUM
ADR2    DD   NUM
ADR3    DW   ARRAY[2]
```

它表示取变 量地址的偏移量

后面三条伪指令的操作数中都包含了前面定义的两个变量

取变量段基值和偏移量。前两个字节存偏移量，后两个字节存段基值



设上述语句所在段的段基值为0915H，NUM的偏移量为0004H，则存储单元的分配情况如图所示。

三、标号

标号写在一条指令的前面，它就是该指令在内存的存放地址的符号表示，也就是指令地址的别名。

标号主要用在程序中需要改变程序的执行顺序时，用来标记转移的目的地，即作转移指令的操作数。

例如：
```
        MOV   CX, 100
LAB:    MOV  AX, BX
        ……
        LOOP  LAB
        JNE    NEXT   ;不为零转移
        ……
NEXT:   ……
```

每个标号具有三属性

（1）段属性（SEG）

它表示该标号所代表的地址在哪个逻辑段中，即段基值。

（2）偏移量属性（OFFSET）

它表示该标号所代表的地址在段内与段起点间的字节数，即地址的偏移量。

（3）距离属性（也叫类型属性）

它表示该标号可以被段内还是段间的指令调用。

NEAR（近）： 该标号只能作段内转移，也就是说只能是与该标号所指指令同在一个逻辑段的转移指令和调用指令才能使用它。

FAR（远）： 该标号可以被非本段的转移和调用指令使用。

标号的距离属性可以有两种方法来指定：

a.隐含方式

当标号加在指令语句前面时，它隐含为NEAR属性。

例 SUB1：MOV AX，30H

SUB1的距离属性为NEAR也就是它只能被本段的转移指令和调用指令访问。

b.用LABEL伪指令给标号指定距离属性

格式： 标号名 LABEL 类型

类型为NEAR或FAR。该语句应与指令语句连用。

例如：SUB1_FAR LABEL FAR
SUB1: MOV AX,30H
……

SUB1_FAR与SUB1两个标号具有相同的段属性和偏移量属性，即相同的逻辑地址。被转移指令或调用指令访问时，是指同一个入口地址，但SUB1-FAR可以被其它段的指令调用。

LABEL伪指令还可以用来定义变量的属性，即改变一个变量的属性，如把字变量的高低字节作为字节变量来处理。

例如：DATA_BYTE LABEL BYTE
DATA_WORD DW 20H DUP（？）

DATA_BYTE与DATA_WORD具有相同的段基值和偏移量。 DATA_BYTE可以被用来存取一个字节数据，而DATA_WORD则不能。

4.3 符号定义语句

在源程序设计中，使用符号定义语句可以将常数或表达式等内容用某个指定的符号来表示。在8086/8088汇编语言中有两种符号定义语句。

一、等值语句

语句格式：符号名　EQU　表达式

功能：用符号名来表示EQU右边的表达式。后面的程序中一旦出现该符号名，汇编程序将把它替换成该表达式。

表达式可以是任何形式，常见的有以下几种情况。

1.常数或数值表达式

```
COUNT  EQU  5
NUM    EQU  COUNT+5
```

2.地址表达式

ADR1 EQU DS：[BP+14]

ADR1被定义为在DS数据段中以BP作基址寻址的一个存储单元。

3.变量、寄存器名或指令助记符

例如：CREG EQU CX；在后面的程序使用CREG就是使用CX
CBD EQU DAA；DAA为十进制调整指令。

注意：在同一源程序中，同一符号不能用EQU定义多次。

例：CBD EQU DAA
CBD EQU ADD

} 错误用法

二、等号语句

格式：符号名=表达式

等号语句与等值语句具有相同的作用。但等号语句可以对一个符号进行多次定义。

例如：

CONT=5

NUM=14H

NUM=NUM+10H

下面是错误用法：

CBD=DAA

……

CBD=ADD

等号语句不能为助记符定义别名

注意：等值语句与等号语句都不会为符号分配存储单元。因此所定义的符号没有段、偏移量和类型等属性。

4.4 表达式与运算符

表达式是指令或伪指令语句操作数的常见形式。它由常数、变量、标号等通过操作运算符连接而成。

注意：任何表达式的值在程序被汇编的过程中进行计算确定，而不是到程序运行时才计算。

8086/8088宏汇编语言中的操作运算符非常丰富，可以分为以下五类。

一、算术运算符

+、—、＊、 / 、MOD、SHL、SHR、[ ]

1.运算符“+”和“-”也可作单目运算符，表示数的正负。

2.使用“+”、“-”、“＊”、和“/”运算符时，参加运算的数和运算结果都是整数。

3.“/”运算为取商的整数部分，而“MOD”运算取除法运算的余数。

例如：

NUM=15＊8；NUM=120
NUM=NUM/7；NUM=17
NUM=NUM MOD 3；NUM=2
NUM=NUM+5；NUM=7
NUM= -NUM - 3；NUM= - 10
NUM=-NUM-NUM；NUM=20

4. “SHR ”和“ SHL ”为逻辑移位运算符

“SHR”为右移，左边移出来的空位用0补入。
“SHL”为左移，右边移出来的空位用0补入。

注意：移位运算符与移位指令区别。 移位运算符的操作对象是某一具体的数（常数），在汇编时完成移位操作。而移位指令是对一个寄存器或存储单元内容在程序运行时执行移位操作。

例如

```
NUM=11011011B
…….
MOV   AX , NUM SHL  1
MOV   BX ,  NUM SHR 2
ADD   DX ,  NUM SHR 6
```

不能改成：
SHL NUM,1

上面的指令序列等效下面三条指令。

```
MOV   AX ，110110110B
MOV   BX ，  00110110B
ADD   DX ，  3
```

5.下标运算符“[ ]”具有相加的作用

一般使用格式：　表达式1 [表达式2]

作用：将表达式1与表达式2的值相加后形成一个存储器操作数的地址。

下面两个语句是等效的。

```
MOV   AX，DA_WORD[20H]
MOV   AX，DA_WORD+20H
```

可以用寄存器来存放下标变量

例：下面几个语句是等价的

```
MOV  AX , ARRAY[BX][SI]; 基址变址寻址
MOV   AX, ARRAY[BX+SI]
MOV   AX , [ARRAY+BX][SI]
MOV   AX , [ARRAY+SI][BX]
MOV   AX , [ARRAY+BX+SI]
```

下面是几个错误的语句。

```
MOV   AX，ARRAY+BX+SI
MOV   AX，ARRAY+BX[SI]
MOV   AX，ARRAY+DA_WORD
```

二、逻辑运算符

逻辑运算符有NOT、AND、OR和XOR等四个，它们执行的都是按位逻辑运算。

例如

```
MOV  AX，NOT 0F0H   =>MOV  AX ,0FF0FH
MOV   AL，NOT 0F0H   =>MOV  AL , 0FH
MOV   BL，55H AND 0F0H =>MOV BL,50H
MOV   BH，55H OR 0F0H  =>MOV BH, 0F5H
MOV   CL，  55H XOR 0F0H =>MOV CL, 0A5H
```

三、关系运算符

关系运算符包括：EQ（等于）、NE（不等于）、LT（小于）、 LE（小于等于）、GT（大于）、 GE（大于等于）

关系运算符用来比较两个表达式的大小。关系运算符比较的两个表达式必须同为常数或同一逻辑段中的变量。

如果是常量的比较，则按无符号数进行比较；如果是变量的比较，则比较它们的偏移量的大小。

关系运算的结果只能是“真”（全1）或“假”（全0）

例1：

```
MOV   AX，0FH  EQ  1111B =>MOV  AX ,0FFFFH
MOV   BX , 0FH  NE  1111B =>MOV   BX, 0
```

例2

```
VAR    DW    NUM  LT  0ABH
```

该语句在汇编时，根据符号常量NUM的大小来决定VAR存储单元的值，当NUM<0ABH时，则变量VAR的内容为0FFFFH，否则VAR的内容为0。

四、数值返回运算符

该类运算符有5个，它们将变量或标号的某些特征值或存储单元地址的一部分提取出来。

1.SEG运算符

作用：取变量或标号所在段的段基值。

例如：

```
DATA   SEGMENT
       K1   DW  1，2
       K2   DW  3，4
        ……
MOV   AX , SEG   K1
MOV   BX , SEG  K2
```

设DATA逻辑段的段基值为1FFEH，则两条传送指令将被汇编为：

```
MOV   AX ，1FFEH
MOV    BX ，1FFEH
```

2.OFFSET运算符

该运算符的作用是取变量或标号在段内的偏移量。

例如：

```
DATA SEGMENT
VAR1  DB   20H DUP(0)
VAR2  DW  5A49H
ADDR  DW  VAR2 ;将VAR2的偏移量20H存入ADDR中
        …….
        MOV  BX，VAR2；(BX)=5A49H
        MOV   SI , OFFSET  VAR2  ;(SI)=20H
        MOV   DI, ADDR   ;DI的内容与SI相同
        MOV   BP，OFFSET  ADDR ;(BP)=22H
```

3.TYPE运算符

作用:取变量或标号的类型属性，并用数字形式表示。对变量来说就是取它的字节长度。

变量
- BYTE 1
- WORD 2
- DWORD 4
- QWORD 8
- TWORD 10

标号
- NEAR -1
- FAR -2

例如：

```
V1   DB    'ABCDE'
V2   DW    1234H, 5678H
V3   DD    V2
     ……
     MOV   AL, TYPE  V1
     MOV   CL, TYPE  V2
     MOV   CH, TYPE  V3
```

经汇编后的等效指令序列如下：

```
MOV  AL，01H
MOV  CL，02H
MOV  CH，04H
```

4.LENGTH运算符

该运算符用于取变量的长度。

- 如果变量是用重复数据操作符DUP说明的,则LENGTH运算取外层DUP给定的值。
- 如果没有用DUP说明，则LENGTH运算返回值总是1。

```
K1  DB   10H DUP（0）
K2  DB    10H，20H，30H，40H
K3  DW  20H DUP（0，1，2 DUP（0）
K4  DB   ‘ABCDEFGH’
……..
```

```
MOV   AL,  LENGTH   K1; (AL)=10H
MOV   BL,  LENGTH   K2 ; (BL)=1
MOV   CX,  LENGTH   K3 ;  (CX)=20H
MOV   DX,  LENGTH   K4 ;  (DX)=1
```

5.SIZE运算符

该运算符只能作用于变量，SIZE取值等于LENGTH和TYPE两个运算符返回值的乘积。

例如，对于上面例子，加上以下指令：

```
MOV  AL，SIZE  K1；  （AL）=10H
MOV   BL，SIZE  K2；  （BL）=1
MOV   CL，SIZE  K3；  （CL）=20H*2=40H
MOV   DL，SIZE  K4；  （DL）=1
```

五、属性修改运算符

这一类运算符用来对变量、标号或存储器操作数的类型属性进行修改或指定。

1.PTR运算符

使用格式：　类型　PTR　地址表达式

作用：将地址表达式所指定的标号、变量或用其它形式表示的存储器地址的类型属性修改为“类型”所指的值。

类型可以是BYTE、WORD、DWORD、NEAR和FAR。
这种修改是临时的，只在含有该运算符的语句内有效。

例如：

```
DA_BYTE   DB   20H  DUP(0)
DA_WORD  DW   30H  DUP(0)
     …….
MOV   AX , WORD  PTR  DA_BYTE[10]
ADD    BYTE   PTR   DA_WORD[20], BL
INC    BYTE  PTR  [BX]
SUB    WORD  PTR  [SI], 100
JMP    FAR   PTR  SUB1;指明SUB1不是本段中的地址
```

2.HIGH/LOW运算符

使用格式：

```
HIGH 表达式
LOW  表达式
```

这两个运算符用来将表达式的值分离出高字节和低字节。

如果表达式为一个常量，则将其分离成高8位和低8位；
如果表达式是一个地址（段基值或偏移量）时，则分离出它的高字节和低字节。

例如：

```
DATA   SEGMENT
CONST  EQU   0ABCDH
DA1    DB    10H DUP （0）
DA2    DW    20H DUP （0）
DATA   ENDS
       …….
MOV    AH ,HIGH  CONST
MOV    AL, LOW   CONST
MOV    BH, HIGH  (OFFSET DA1)
MOV    BL,  LOW  (OFFSET  DA2)
MOV    CH,  HIGH (SEG  DA1)
MOV    CL,  LOW  (SEG   DA2)
```

设DATA段的段基值是0926H，则上述指令序列汇编后的等效指令为:

```
MOV   AH ，  0ABH
MOV   AL ，  0CDH
MOV   BH ，  00H
MOV   BL ，  10H
MOV   CH ，  09H
MOV   CL ，  26H
```

注意：HIGH/LOW运算符不能用来分离一个变量、寄存器或存储器单元的高字节与低字节。

下面语句使用是错误的用法。

```
DA1   DW  1234H
……
MOV   AH，HIGH  DA1
MOV   BH，  LOW  AX
MOV   CH，HIGH [SI]
```

3、THIS运算符

THIS运算符一般与等值运算符EQU连用，用来定义一个变量或标号的类型属性。所定义的变量或标号的*段基值*和*偏移量*与紧跟其后的变量或标号相同。

例如：

```
DATA_BYTE  EQU   THIS  BYTE
DATA_WORD  DW    10 DUP (0)

           ……
           MOV   AX, DATA_WORD
           MOV   BL, DATA_BYTE
           ……
```

又如：

```
LFAR     EQU   THIS  FAR
LNEAR：  MOV   AX，  B
```

标号LFAR与LNEAR具有相同的逻辑地址值，但类型不同。LNEAR只能被本段中的指令调用，而LFAR可以被其它段的指令调用。

六、运算符的优先级

在一个表达式中如果存在多个运算符时，在计算时就有先后顺序问题。不同的运算符具有不同的运算优先级别。

优先级别	运算符
（最高）1	LENGTH，SIZE，圆括号
2	PTR，OFFSET，SEG，TYPE，THIS
3	HIGH，LOW
4	*，/，MOD，SHR，SHL
5	+，-
6	EQ，NE，LT，LE，GT，GE
7	NOT
8	AND
（最低）9	OR，XOR

汇编程序在计算表达式时，按以下规则进行运算。

- 先执行优先级别高的运算，再算较低级别运算；
- 相同优先级别的操作，按照在表达式中的顺序，从左到右进行；
- 可以用圆括号改变运算的顺序。

例如：

K1= 10 OR 5 AND 1 ；结果为K1=11
K2=（10 OR 5） AND 1；结果为K2=1

4.5 程序的段结构

8086/8088在管理内存时，按照逻辑段进行划分，不同的逻辑段可以用来存放不同目的的数据。在程序中使用四个段寄存器**CS,DS,ES**和**SS**来访问它们。

在源程序设计时，使用伪指令来定义和使用这些逻辑段。

一、段定义伪指令

伪指令SEGMENT和ENDS用于定义一个逻辑段。使用时必须配对，分别表示定义的开始与结束。

一般格式：

```
段名   SEGMENT  [定位类型] [组合类型] ['类别名']
          ……
          … …      本段语句序列
          … …
段名   ENDS
```

段定义伪指令语句各部分的作用如下：

1、段名

段名是由用户自己任意选定的，符合标识符定义规则的一个名称。

最好选用与该逻辑段用途相关的名称。如第一个数据段为DATA1,第二个数据为DATA2等。

一个段的开始与结尾用的段名必须一致。

2、定位类型

定位类型用于决定段的起始边界，即第一个可存放数据的位置（不是段基址）。它可以有4种取值。

（1）**PAGE**：表示该段从一个页面的边界开始

由于一个页面为256个字节，并且页面编号从0开始，因此，PAGE定位类型的段起始地址的最后8位二进制数一定为0，即以00H结尾的地址。

（2）**PARA**:表示该段从一个小节的边界开始

如果用户未选定位类型，则缺省为PARA。

（3）**WORD**:表示该段从一个偶数字节地址开始，即段起始单元地址的最后一位二进制数一定是**0**。

（4）**BYTE**:表示该段起始单元地址可以是任一地址值。

注意：定位类型为**PAGE**和**PARA**时，段起始地址与段基址相同。定位类型为**WORD**和**BYTE**时，段起始地址与段基址可能不同。

3、组合类型

组合类型说明符用来指定段与段之间的连接关系和定位。它有六种取值选择。

（1）若未指定组合类型，表示本段与其它段无连接关系。在装入内存时，本段有自己的物理段，因此有自己的段基址

（2）**PUBLIC**:在满足定位类型的前提下，将与该段同名的段邻接在一起，形成一个新的逻辑段，共用一个段基址。段内的所有偏移量调整为相对于新逻辑段的段基址。

（3）**COMMON**: 产生一个覆盖段。在多个模块连接时，把该段与其它也用COMMON说明的同名段置成相同的段基址，这样可达到共享同一存储区。共享存储区的长度由同名段中最大的段确定。

（4）**STACK**:把所有同名段连接成一个连续段，且系统自动对SS段寄存器初始化为该连续段的段基址。并初始化堆栈指针**SP**。

用户程序中应至少有一个段用STACK说明，否则需要用户程序自己初始化SS和SP。

（5）**AT表达式**：表示本段可定位在表达式所指示的小节边界上。表达式的值也就是段基值。

（6）**MEMORY**:表示本段在存储器中应定位在所有其它段之后的最高地址上。如果有多个用MEMORY说明的段，则只处理第一个用MEMORY说明的段。其余的被视为COMMON

4.类别名

类别名为某一个段或几个相同类型段设定的类型名称。系统在进行连接处理时，把类别名相同的段存放在相邻的存储区，但段的划分与使用仍按原来的设定。

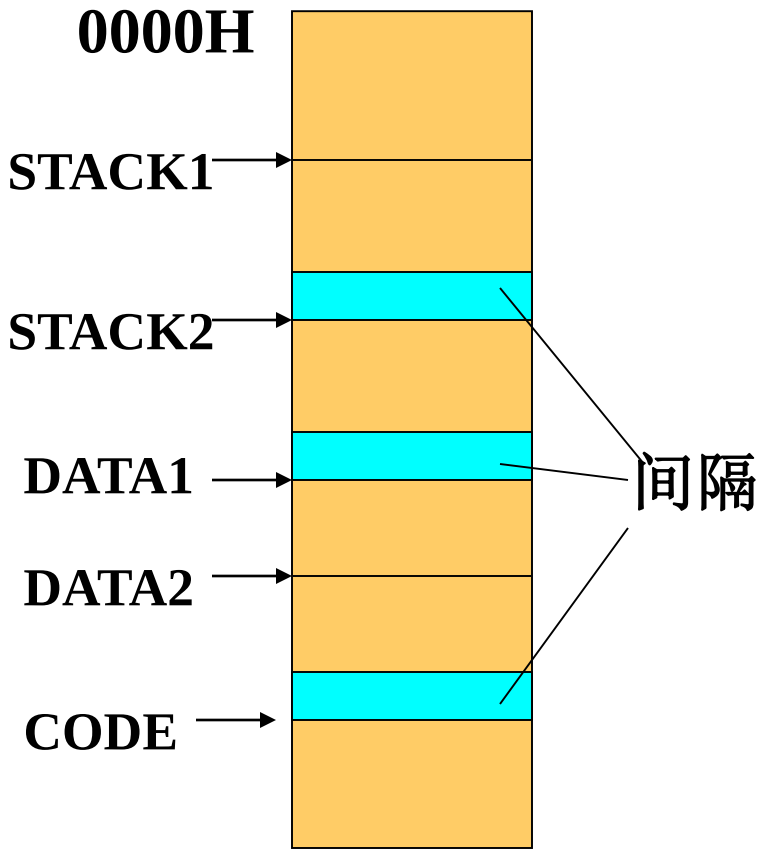
类别名必须用单引号引起来。所用字符串可任意选定，但它不能使用程序中的标号、变量名或其它定义的符号。

在定义一个段时，段名是必须有的项，而定位类型、组合类型和类别名三个参数是可选项。各个参数之间用空格分隔。各参数之间的顺序不能改变。

下面是一个分段结构的源程序框架。

```
STACK1   SEGMENT  PARA  STACK  'STACK0'
         .....
STACK1   ENDS
DATA1    SEGMENT  PARA  'DATA'
         ......
DATA1    ENDS
STACK2   SEGMENT  PARA  'STACK0'
         ......
STACK2   ENDS
CODE     SEGMENT  PARA  MEMORY
       ASSUME  CS:CODE,DS:DATA1,SS:STACK1
MAIN:    ......
         ......
CODE     ENDS
DATA2    SEGMENT  BYTE  'DATA'
         ......
DATA2    ENDS
              END  MAIN
```

上述源程序经LINK程序进行连接处理后，程序被装入内存的情况如图所示。



如果在段定义中选用了PARA定位类型说明，则在一个段的结尾与另一个段的开始之间可能存在一些空白，图中以兰色框表示。CODE段的组合类型为MEMORY,因此被装入在其它段之后。

在进行程序设计时，如果程序不大，一般只需要定义三个段就可以了。

二、段寻址伪指令

段寻址伪指令**ASSUME**的作用是告诉汇编程序,在处理源程序时，定义的段与哪个寄存器关联。

ASSUME并不设置各个段寄存器的具体内容，段寄存器的值是在程序运行时设定的。

一般格式：

ASSUME 段寄存器名：段名，段寄存器名：段名，.....

其中段寄存器名为**CS,DS,ES**和**SS**四个之一，段名是用**SEGMENT/ENDS**伪指令定义的段名。

例如：

```
DATA1  SEGMENT
VAR1    DB  12H
DATA1  ENDS
DATA2  SEGMENT
 VAR2    DB  34H
 DATA2  ENDS
 CODE   SEGMENT
 VAR3    DB  56H
      ASSUME  CS:CODE,DS:DATA1,ES:DATA2
START: .....
        .....
        INC  VAR1
        INC  VAR2
        INC  VAR3
        ......
CODE   ENDS
        END START
```

该指令汇编时，VAR1使用的是DS

该指令被汇编时，VAR2使用的是ES,即指令编码中有段前缀ES

该指令汇编时，VAR3使用的是CS，即指令编码中有段前缀CS

- 在一个代码段中可以有几条ASSUME伪指令，对于前面的设置，可以用ASSUME改变原来的设置。

- 一条ASSUME语句不一定设置全部段寄存器，可以选择其中一个或几个段寄存器。

- 可以使用关键字NOTHING将前面的设置删除。

例如：

```
ASSUME  ES:NOTHING ;删除前面对ES与某个定义段的关联
ASSUME  NOTHING    ;删除全部4个段寄存器的设置
```

三、段寄存器的装入

段寄存器的初值（段基值）装入需要用程序的方法来实现。四个段寄存器的装入方法略有不同。

1、DS和ES的装入

在程序中，使用数据传送语句来实现对DS和ES的装入。

例如：

```
DATA1    SEGMENT
DBYTE1  DB  12H
DATA1    ENDS
DATA2    SEGMENT
DBYTE2  DB 14H DUP(?)
DATA2    ENDS
CODE      SEGMENT
            ASSUME  CS:CODE,DS:DATA1
START: MOV  AX,DATA1
            MOV  DS,AX
            MOV  AX,DATA2
            MOV  ES,AX
            MOV  AL,DBYTE1
            MOV  DBYTE2[2],AL
            ……
CODE   ENDS
```

该指令在汇编时出错，因为在ASSUME指令中未指定ES与DATA2的联系。

为了改正上述程序中的错误，可以在变量DBYTE2前加一个段前缀说明即可。即：

MOV ES:DBYTE2[2], AL

2、SS的装入

SS的装入有两种方法

（1）在段定义伪指令的组合类型项中，使用STACK参数，并在段寻址伪指令ASSUME语句中把该段与SS段寄存器关联。

例如：

```
STACK1  SEGMENT  PARA  STACK
        DB  40H DUP(?)
STACK1  ENDS
    ……
CODE    SEGMENT
        ASSUME  CS:CODE,SS:STACK1
    ……
```

SS将被自动装入STACK1段的段基值，堆栈指针SP也将指向堆栈底部+2的存储单元。上例中（SP）=40H。

（2）如果在段定义伪指令的组合类型中，未使用STACK参数，或者是在程序中要调换到另一个堆栈，这时，可以使用类似于DS和ES的装入方法。

例如：

TOP变量的偏移量为40H

```
DATA_STACK   SEGMENT
             DB  40H DUP(?)
       TOP   LABEL  WORD
DATA_STACK   ENDS
          ……
CODE         SEGMENT
          ……
             MOV  AX,DATA_STACK
             MOV   SS,AX
             MOV   SP,OFFSET TOP
          ……
```

3、CS的装入

CPU在执行指令之前根据CS和IP的内容来从内存中提取指令,即必须在程序执行之前装入CS和IP的值。因此,CS和IP的初始值就不能用可执行语句来装入。

装入CS和IP一般有下面两种情况。

(1)由系统软件按照结束伪指令指定的地址装入初始的CS和IP

任何一个源程序都必须以END伪指令来结束。

其格式为：　END　起始地址

起始地址可以是一个标号或表达式，它与程序中第一条指令语句前所加的标号必须一致。

END伪指令的作用是标识源程序结束和指定程序运行时的起始地址。当程序被装入内存时，系统软件根据起始地址的段基值和偏移量分别被装入CS和IP中。

例如：

```
    ......
CODE   SEGMENT
       ASSUME  CS:CODE,.....
START: ......
       ......
CODE   ENDS
       END START
```

(2)在程序运行期间，当执行某些指令时，CPU自动修改CS和IP，使它们指向新的代码段。

例如：

执行段间过程调用CALL和段间返回指令RET;
执行段间无条件转移指令JMP;
响应中断及中断返回指令;
执行硬件复位操作。

4.6 过程定义伪指令（PROC/ENDP)

在程序设计过程中，常常将具有一定功能的程序段设计成一个子程序。在MASM宏汇编程序中，用过程(PROCEDURE)来构造子程序。

过程定义伪指令格式如下：

```
过程名   PROC  [NEAR/FAR]
          ......
          RET
          ......
过程名   ENDP
```

过程名是子程序的名称，它被用作过程调用指令CALL的目的操作数。它类同一个标号的作用。具有段、偏移量和距离三个属性。而距离属性使用NEAR和FAR来指定，若没有指定，则隐含为NEAR。

NEAR过程只能被本段指令调用，而FAR过程可以供其它段的指令调用。

每一个过程中必须包含有返回指令RET,其作用是控制CPU从子程序中返回到调用该过程的主程序。

4.7 当前位置计数器$与定位伪指令ORG(Origin)

汇编程序在汇编源程序时，每遇到一个逻辑段，就要为其设置一个位置计数器，它用来记录该逻辑段中定义的每一个数据或每一条指令在逻辑段中的相对位置。

在源程序中，使用符号$来表示位置计数器的当前值。因此，$被称为当前计数器。它位于不同的位置具有不同的值。

位置计数器$在使用上完全类似变量的使用.

定位伪指令**ORG**——用来改变位置计数器的值。

格式：　**ORG　数值表达式**

作用：将数值表达式的值赋给当前位置计数器**$** 。**ORG**语句为其后的数据或指令设置起始偏移量。

表达式的值必须为正值。表达式中也可以包含有当前位置计数器的现行值**$**。

```
DATA1    SEGMENT
            ORG  30H
    DB1   DB  12H,34H  ;DB1在DATA1段内的偏移量为30H
          ORG  $+20H;保留20H个字节单元，其后再存放'ABCD....
  STRING  DB  'ABCDEFGHI'
  COUNT   EQU   $-STRING;计算STRING的长度
  DB2  DW  $;  取$的偏移量,类似变量的用法
  DB3  DB   $     ;此语句错误!
DATA1    ENDS
  CODE      SEGMENT
            ASSUME  CS:CODE.....
      ORG  10H
  START:  MOV  AX,DATA
             MOV  DS,AX
             ......
  CODE    ENDS
            END START
```

4.8 标题伪指令TITLE

语句格式：**TITLE 标题名**

作用：给所在程序指定一个标题。以便在列表文件的每一页的第一行都显示这个标题。其中标题是用户任意选用的字符串，字符个数不能超过60。

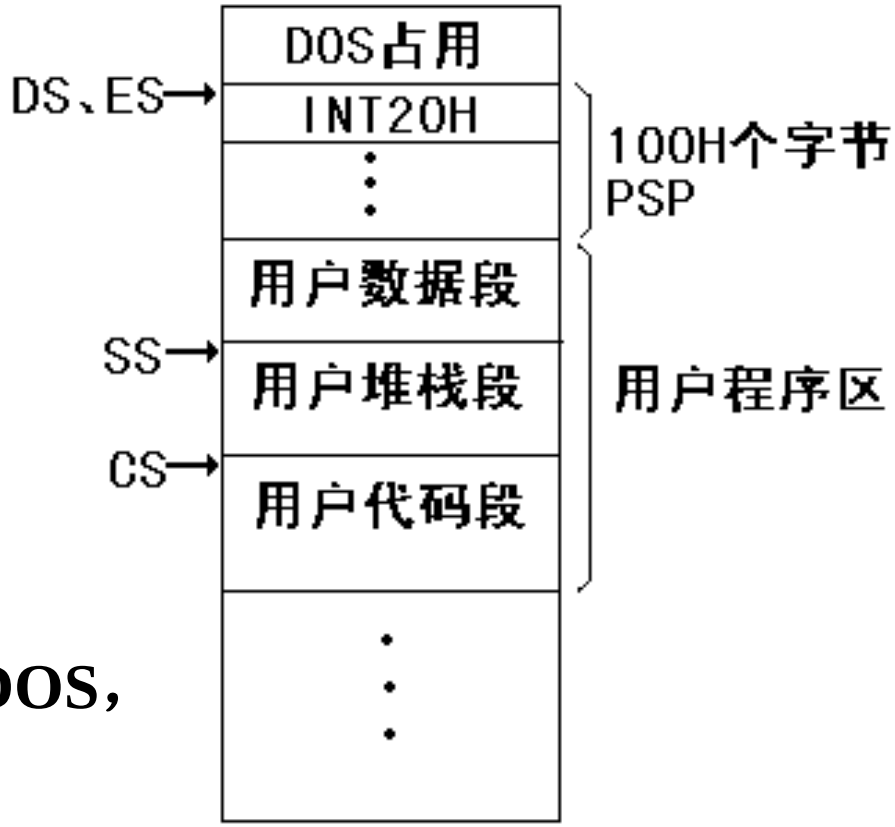
4.9 从程序返回操作系统的方法

为了使程序运行结束后，能够正确地返回到操作系统，需要在程序中加上一些必要的语句。一般有以下两种方法。

一、使用程序段前缀PSP（Program Segment Prefix)实现返回

DOS系统将一个.EXE文件（可执行文件）装入内存时，在该文件的前面生成一个程序段前缀PSP，其长度为100H字节。同时让DS和ES都指向PSP的开始，而CS指向该程序的代码段，即第一条可执行指令。

如图所示。PSP中一开始就是一条中断指令 INT 20H，执行该指令将终止用户程序，返回DOS系统。



为了使程序执行完后，正确返回DOS，需要做以下三个操作：

1. 将用户程序编制成一个过程,类型为FAR;

2. 将PSP的起始逻辑地址压栈,即将INT 20H指令的地址压栈;

3. 在用户程序结尾处,使用一条RET指令。执行该指令将使保存在堆栈中的PSP的起始地址弹出到CS和IP中。

程序结构：

```
DATA    SEGMENT
          …
DATA    ENDS
STACK1  SEGMENT STACK
          …
STACK1  ENDS
CODE    SEGMENT
BEGIN   PROC FAR
        ASSUME CS:CODE,DS:DATA,SS:STACK1
        PUSH DS
        MOV AX,0
        PUSH AX
        MOV AX,DATA
        MOV DS,AX
          …
        RET
BEGIN   ENDP
CODE    ENDS
        END BEGIN
```

二、使用DOS系统功能调用实现返回

执行DOS功能调用4CH，也可以控制用户程序结束，并返回DOS操作系统。

在程序结束时，使用两条指令：

```
MOV AH，4CH
INT  21H
```

代码段的结构为：

```
CODE   SEGMENT
       ASSUME  CS:CODE.....
BEGIN:MOV  AX,DATA
       MOV  DS,AX
       …
       MOV  AH,4CH
       INT  21H
CODE   ENDS
       END  BEGIN
```