第3章　汇编语言程序设计

- 机器语言：用二进制表示指令和数据。
- 汇编语言：由指令助记符、符号常量和标号等符号书写程序的设计语言称为汇编语言（**Assemble Language)**。

汇编语言需要翻译为机器语言，**CPU**才能执行，这个过程称为汇编。汇编程序分为小汇编（**ASM**），和宏汇编（**MASM**）。

- 高级语言：通用性强，编程简捷，易读，易维护。最终翻译为机器语言才能执行。

汇编程序(MASM.EXE)的主要功能是：

1、将汇编语言源程序翻译成机器语言；

2、根据程序分配存储区域（程序区，数据区，堆栈区）；

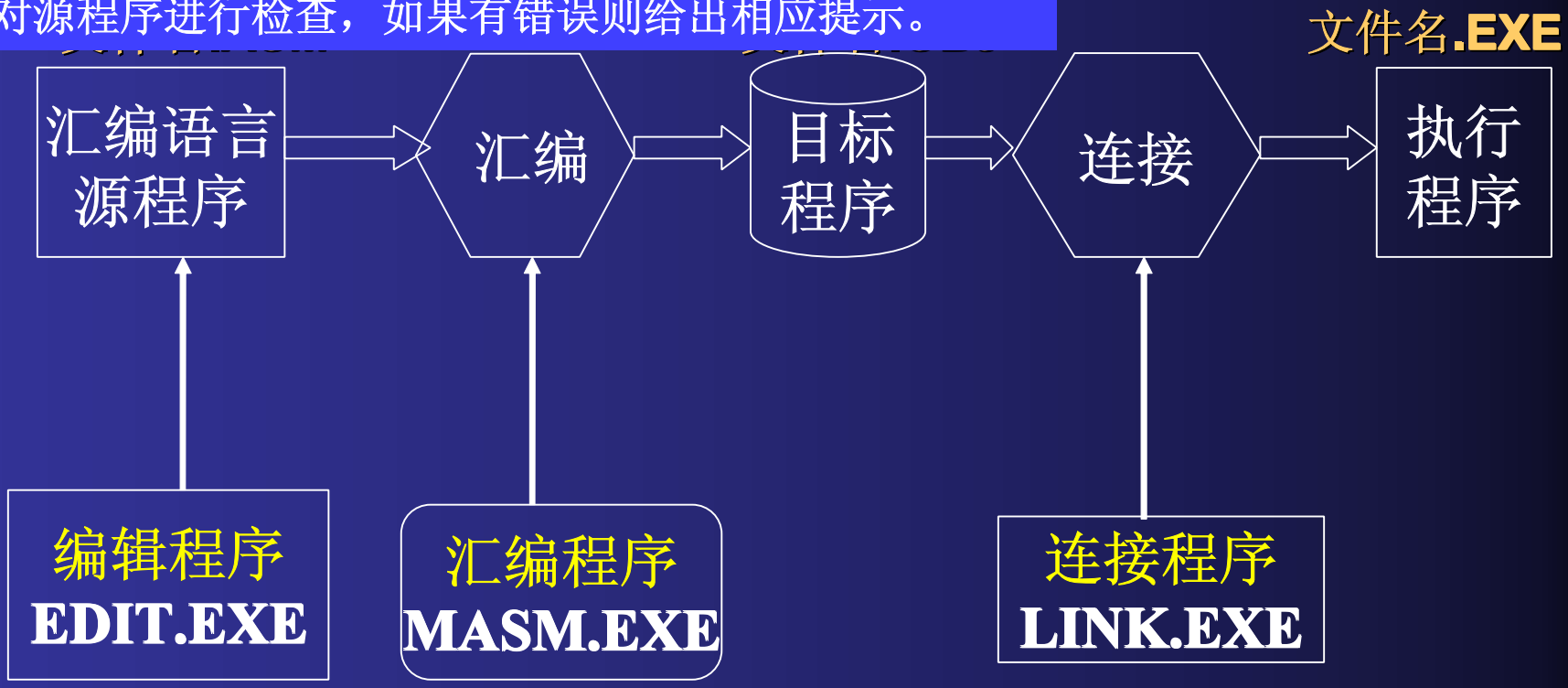
3、将各种进位制数据转换成二进制数；

4、把字符转换成ASCII码；

5、计算出数值表达式的值；

6、对源程序进行检查，如果有错误则给出相应提示。

程序和连接程序生



3.1 汇编语言程序格式

一、示例程序

```
data1    segment 'data'                          ;数据段
         NUM    DW    0011101000000111B          ;即3A07H
data1    ends
stack1   segment stack 'stack'                   ;堆栈段
         SAVE DW 100 DUP(?)
stack1   ends
code     segment ‘code’                          ;代码段
         assume cs:code,ds:data1,ss:stack1
begin:   mov  ax, data1                          ;建立数据段段地址
         mov  ds, ax
         mov  ax, stack1                         ;建立堆栈段
         mov  ss, ax
         ……                                      ;程序代码
code     ends
         end begin
```

汇编语言中有两类指令：

一类是执行性指令，另一类是说明性指令。

1、 执行性指令

即指令系统中包括的指令。

执行性指令的格式

[标号:] [前缀] 指令助记符 [操作数表] [；注释]

1）标号代表“:”后指令的存储地址，供**JMP**，**CALL**和**LOOP**等指令操作使用。

2）前缀是 **8086/8088**中一些特殊指令，它们同其他指令配合使用，如“串操作指令”的重复指令**REP**。

3）指令助记符包括**8086/8088**指令助记符。

4）操作数表是由逗号分隔开的多个操作数。包括目标操作数和源操作数，

5）注释以“；”开始，用来简要说明该指令在程序中的作用，以提高程序的可读性。

2、说明性指令（伪指令）

说明性指令在汇编时不产生任何代码。

表示源程序的起始终止信息、分段情况、内存结构和变量说明等信息。

说明性指令的格式如下：

[名字] 伪指令 [操作数表] [；注释]

3.2 汇编语言语句的组成

1、指令语句

在汇编时产生目标代码，与机器指令对应的语句。

如：**MOV　AX, CX**

ADD　AX, datal

2、伪指令语句

在汇编时不产生目标代码，只为汇编程序提供汇编时所
需要信息的语句。

如：**datal　　DW　　12abH**

3、宏指令语句

为了书写方便，把一组汇编语句序列用一条指令代替，这种指令称为宏指令。在汇编时，凡是有宏指令的地方将用相应的汇编语句序列取代，所以宏指令可以产生目标代码。

宏指令定义 如下，

```
funcal   macro   x          ; x是形式参数
         mov     ah,  x
         int     21h        ; 宏定义
         endm
```

funcal 是宏指令，使用时直接写成：

```
funcal   2          ; 其中2是宏参数，
```

汇编时产生：

```
mov   ah,   2
int    21h
```

两条语句，并将它们汇编成目标代码。

汇编语言中的几个基本概念

1、标识符

标识符即标号和名字，标号后面必须跟冒号。

名字可以是变量名、段名、过程名等。

2、保留字

保留字是汇编语言中预先保留下来的具有特殊含义的符号，不能滥用。

3、数的表示

⑴ 常数

二进、八进、十进和十六进制数，注意十六进制数若以字母开头，前面要加数字**0**。

⑵ 实数（有数值协处理器的机器）

±整数部分**.** 小数部分 **E±**指数部分

例如：实数**5.213E－6＝5.213**×10^{-6}

⑶ 字符串常数：用单引号括起来的一个或多个字符组成一个字符串常数，例如：**‘The rezult is:’**在内存中。以**ASCII**码值存放，一个空格也是字符

4、变量

变量在程序运行期间可以修改。

定义变量就是给变量分配一个或多个存储单元，并且对该存储单元赋于一个名字——变量名，同时预置初始值。

定义变量用数据定义伪指令**DB**、**DW**、**DD**、**DQ**、**DT**。定义变量是在逻辑段——数据段、附加段和堆栈段内完成的。

例：
```
DATA1   SEGMENT 'data'
        x1    DB    12H
        x2    DW    5678H
DATA1   ENDS
```

变量定义后，变量具有三个属性：段属性、偏移属性和类型属性。

- 段属性（SEG)：表示变量存放在那一个逻辑段中，段的基地址是多少。如上面定义的变量名x1和x2存放在DATA1逻辑段中。

对它们进行存取时要先将它们所在段的段基址放在DS中，段基地址就是DATA1表示的数值，执行下面语句：

```
mov    ax, DATA1
mov    ds,  ax
```

汇编后

```
mov  ax, 1292H
```

- 偏移属性(OFFSET)：表示变量在逻辑段中离段基址的偏移字节数，即偏移地址。如上面的x1的偏移地址为0000H，x2的偏移地址为0001H。

变量的段属性和偏移属性——构成了变量的逻辑地址（段基地址：偏移地址）。

程序中用变量名代表对存储单元的访问。

```
mov    al, x1
mov    bx, x2
```

```
mov  al, [0000H]
mov  bx, [0001H]
```

- 类型属性(type)：表示变量占用存储单元的字节数

  DB　1字节，DW　2字节，DD　4字节，

  DQ　8字节，DT　10字节

例如，上面**X1**占有**1**个字节，初始值是**12H**；**X2**占用两个字节，初始值**5678H**

(2) 变量定义伪指令

格式： 变量名 伪指令(db,dw,dd,…) 表达式1，表达式2，‥‥

表达式有以下几种情况：

① 数值表达式

例2： **DA_BYTE DB 50H , 50 , 0caH**

DA_WORD DW 0a3f1H , 498dH

② ? 表达式，不带引号的？表示可预置任何内容

例3： **DA_B DB ? , ?** ； 要求分配两个字节单元

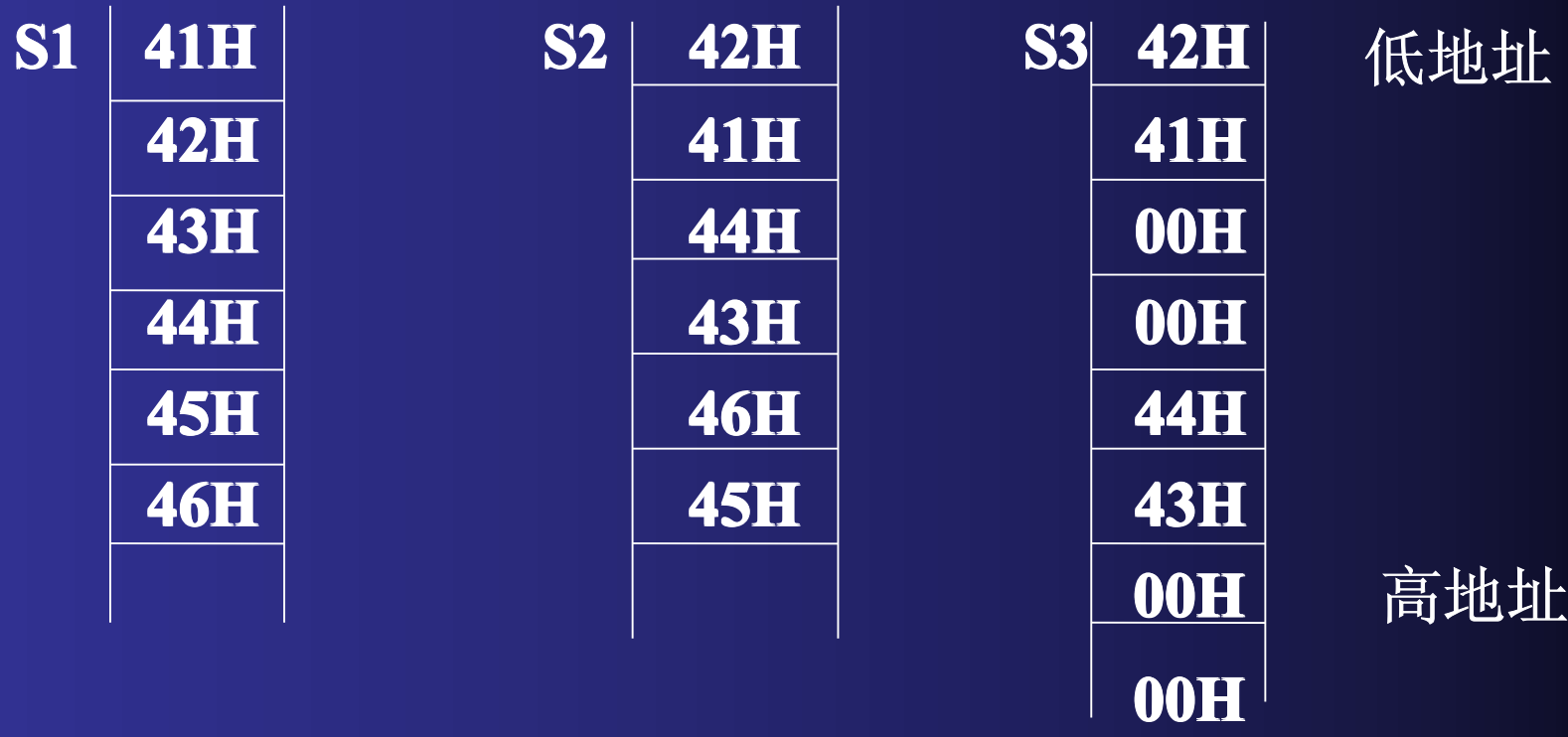
DA_W DW ? , ? ； 要求分配两个字单元

③ 字符串表达式

表达式可以写成字符串形式，只能用**DB**、**DW**、**DD**定义，而且**DW**、**DD**语句定义的串只允许包含两个字符。

例4：　S1　　DB　‘ABCDEF’
　　S2　　DW　‘AB’, ‘ CD’, ‘ EF’
　　S3　　DD　‘AB’, ‘ CD’

这几个变量在存储器中存放情况如下：



注意：定义多于两个以上字符的字符串时，只能使用**DB**伪指令，不能使用**DW**和**DD**等伪指令。

④ 带DUP表达式， DUP是定义重复数据操作符，格式：

变量名 数据定义伪指令(db,dw,…) 重复次数 DUP（重复内容）

例5:

```
D_B1    DB   20H  DUP(?)          ；保留20H个字节
D_B2    DB   10H  DUP('ABCD')     ；字符串'ABCD' 重复10H次
D_W1    DW   10H  DUP(4)          ；字4重复10H次
```

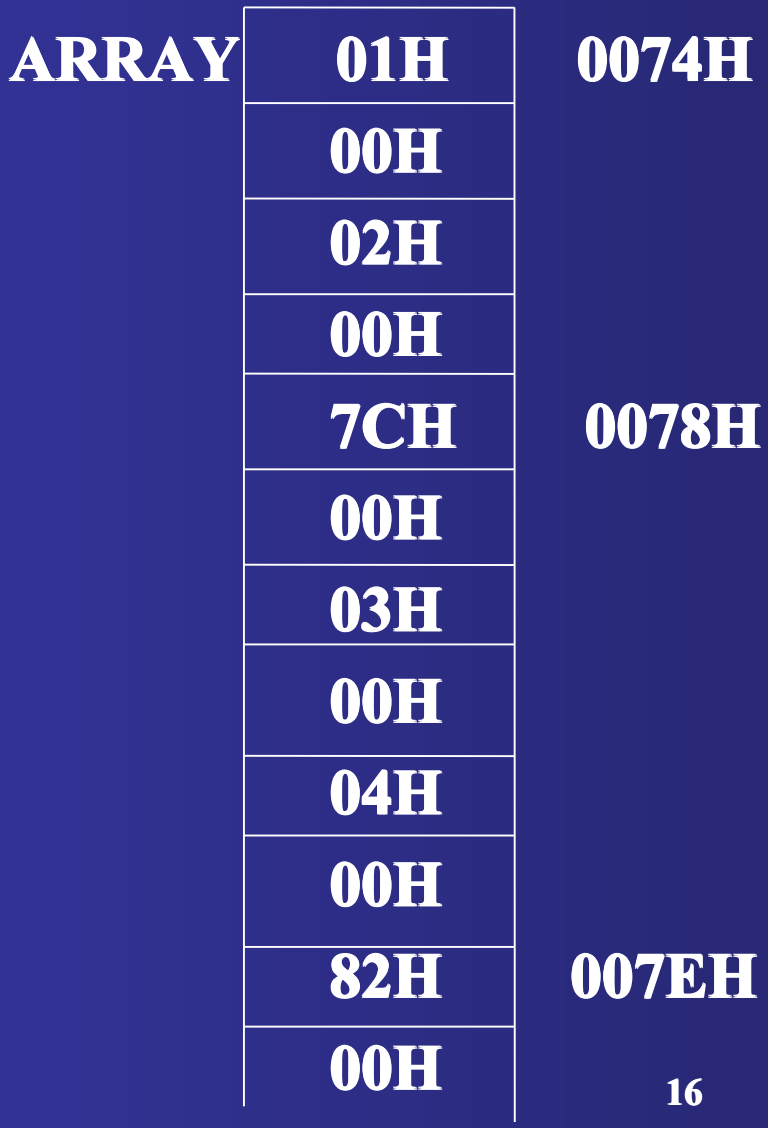
⑤ '$'符号，伪指令中表示地址计数器的当前值

例6:

```
ARRAY   DW  1，2，$+4，  3，4，  $+4
```

如果在汇编时，ARRAY的偏移地址是0074H，则在ARRAY数组中，两个$+4得到的结果是不同的，这是由于$的值是在不断变化的。

ARRAY DW 1，2，$+4， 3，4， $+4



$用在伪指令和用在执行指令中的情况是不同的：

$用在指令中只表示该指令的首地址。

例如指令 **JNE $+6,** 表示满足条件时转移到该指令的首地址加6以后所在的单元。

1000:2543 JNE $+6

则转移地址是**2549H**

例7：下面的数据段定义，COUNT 表示什么？

```
DATA    SEGMENT
        BUF         DB    '0123456789'
        COUNT       EQU   $ - BUF
DATA    ENDS
```

这里，COUNT的值就是数据区的长度，所以COUNT=10

```
MOV  BX, 0
MOV AX, BUF[BX]

MOV  DX,  OFFSET  BUF

LEA   DX,    BUF
```

变量定义伪指令功能：在变量名所对应的地址内存开始依次存入各项数据。当同时有几个定义语句时，由低地址到高地址给每个变量语句中的表达式分配存储单元。

例：
```
DATA1  DB   23H，34H
DATA2  DW   13A5H
DATA3  DD   8975H
DATA4  DB   （-1），3*4
DATA5  DB   ’23’
DATA6  DW ’AB’
DATA7  DW ?
```

段地址：偏移地址		
2000：0000H	23H	DATA1
0001H	34H	
2	A5H	DATA2
3	13H	
4	75H	DATA3
5	89H	
6	00H	
7	00H	
8	FFH	DATA4
9	0CH	
A	32H	DATA5
B	33H	
C	42H	DATA6
D	41H	
E	?	DATA7
F	?	

? 为可在内存单元中放任意值

四、表达式和运算符

表达式由常数、操作数、操作符和运算符组成。有六种运算符，即算术运算符、逻辑运算符和关系运算符、分析运算符、综合运算符和分离运算符。

1、算术运算符

+、－、*、/、MOD(取余）、SHL（左移）、SHR(右移）

例1 ： 32 MOD 5 ； 结果为2

21H SHL 2 ； 结果为84H

2、逻辑运算符（按位操作）

AND(与)	24H AND 0FH＝04H
OR (或)	24H OR 0FH＝2FH
XOR (异或)	24H XOR 0FH＝2BH
NOT (非)	NOT 24H ＝ 0DBH

3、关系运算符

关系运算是逻辑判定，当为真时结果为全1（0FFFFH)，为假时结果为全0。

EQ（等于）　　；若PP=25，则25 EQ PP = 0FFFFH

NE（不等于）　　；25 NE PP = 0

LT（小于）　　；25 LT 26 = 0FFFFH

LE（小于等于）　；25 LE PP = 0FFFFH

GT（大于）　　；26 GT PP = 0FFFFH

GE（大于等于）　；24 GE PP = 0

4、分析运算符

1）OFFSET（求偏移地址）

格式： **OFFSET** < 符号名 >

2） SEG (求段地址）

格式： **SEG** < 符号名 >

例：
```
MOV  BX，OFFSET  DATA3  ；取DATA3的偏移地址给BX
MOV  AX，SEG  DATAZ     ；取DATAZ的段基址给AX
MOV  DS，AX
```

3）TYPE （求符号名类型值）

格式：**TYPE** < 符号名 >

类型	byte	word	dword	qword	tbyte	NEAR	FAR
类型值	1	2	4	8	10	-1 (FFH)	-2 (FEH)

4）LENGTH，求为符号名分配的项数。

格式：　**LENGTH** < 符号名 >

这里为符号名定义的数据项必须是用重复格式**DUP**（ ）定义的。而对于其他情况则回送**1**。

5）SIZE ，求为符号名分配的字节数

格式：　**SIZE** < 符号名 >

返回分配给该符号名的字节数，此值是**LENGTH**的值和**TYPE**的值的乘积。

例2：K2　DW　10 DUP （？）

则　MOV　AX, LENGTH　K2　　;AX = 10

MOV　BX, TYPE　K2　　;BX = 2

MOV　CX, SIZE　K2　　;CX = 20

例3：AARR DW 2，4，6　　;不是DUP,返回1

则 LENGTH AARR = 1，

TYPE　AARR = 2

SIZE　AARR = 2

5、分离运算符

(1) 取低字节 格式：

LOW < 符号名 >

(2) 取高字节 格式：

HIGH < 符号名 >

例7：设**SSY=2050H**

```
mov   al , LOW  3080H   ;   al = 80H
mov   ah , HIGH  SSY    ;   ah = 20H
mov   cl , LOW   3a4bH  ;   cl = 4bH
```

6、合成运算符PTR

格式： 类型 **PTR** 表达式

可以是标号，变量或用各种寻址方式表示的地址表达式

可以是**BYTE**，**WORD**，**DWORD**，**NEAR**，**FAR**

功能：**PTR** 不分配存储单元，可以给已分配的存储单元赋予新的类型属性。**PTR**只是临时改变变量的属性，它不能改变这些变量在定义时的永久属性。

例： **MOV WORD PTR [BX]，0**

JMP NEAR PTR ROAT

INC BYTE PTR[BX][SI]

例：

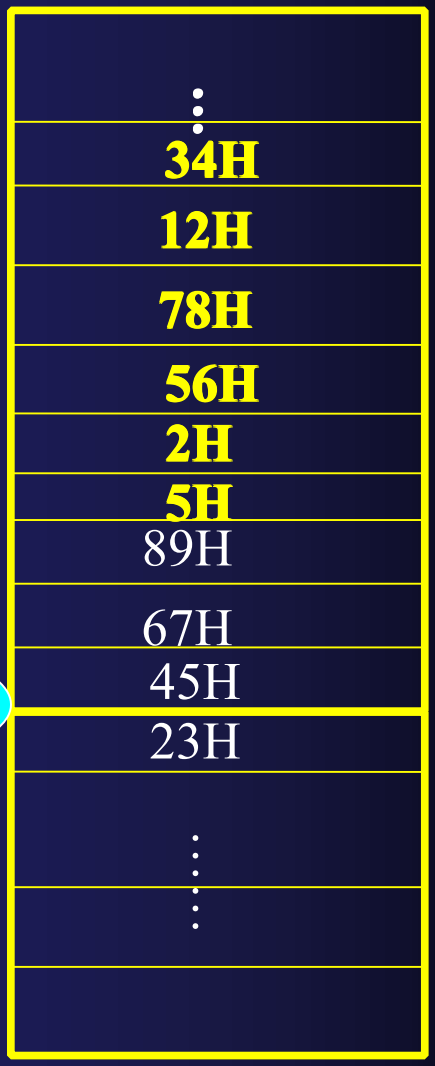
```
W1    DW    1234H, 5678H
B1    DB    2
      DB    5
 D1   DD    23456789H
 …
MOV AX, word ptr B1 ;
MOV BH, byte ptr W1  ;
MOV CH, byte ptr W1+1 ;
MOV word ptr D1, 12H  ;
```

AX=0502H

BH=34H

CH=12H

(D1)=23450012H



7、合成运算符THIS

格式：**THIS** 数据类型

功能：它为同一存储单元取另一别名，该别名可具有其自身的数据属性，但其段地址和偏移量是不变的。该操作数的段地址和偏移地址与下一个存储单元的地址相同。

注：**THIS** 运算符与**PTR** 一样可以改变存储区的类型，但用法不同

PTR用在指令语句中，临时改变表达式的类型；

而**THIS** 则将新类型赋予一个新的操作数，

该操作数的段地址和偏移地址与下一个存储单元地址相同

7、合成运算符THIS

例： FIRT EQU THIS BYTE

SECO DW 100 DUP （0）

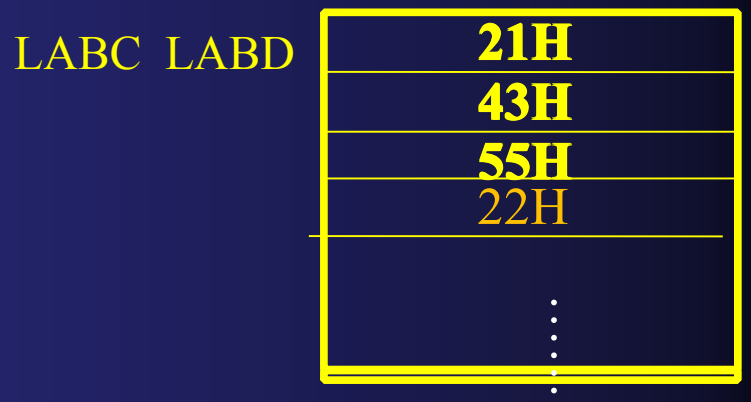
FIRT 和SECO的地址相同，但FIRT为字节类型，SECO为字类型

例 LABC EQU THIS BYTE

LABD DW 4321H,2255H

MOV AL, LABC

MOV BX, LABD



AL=21H

BX=4321H

3.3 伪操作命令

1、符号常量定义伪指令

< 符号名 > EQU < 表达式 >

< 符号名 > = < 表达式 >

例1：

```
port1     EQU   78
port2     EQU   port1+2
counter   EQU   cx          ；定义为寄存器
cbd       EQU   DAA         ；定义为助记符
A         =     6
A         =     A+2
```

用 = 定义的符号名在同一程序中可以重复定义，而用EQU定义的符号名在同一程序中不允许重复定义。

BUF EQU 55H ； BUF等价于数值55H

定义之后，则指令

MOV AL，BUF

等价于

MOV AL，55H

LABEL伪指令

功能：定义标号或变量的类型

格式：变量/标号名　　**LABEL**　　类型

变量类型：**BYTE、WORD、DWORD**

标号类型：**NEAR、FAR**

LABLE　可使同一数据区兼有**BYTE、WORD**两种属性

```
AREAW    LABEL   WORD;   AREAW的类型是WORD
AREAB    DB  100 DUP（？）；  AREAB的类型是BYTE
  ¦
              MOV    AREAW，AX
```

MOV　AREAB，AX　错

2、内存数据定义伪指令

DB 定义字节（前面已讲过定义方法）

DW 定义字（**2**字节）

DD 定义双字（**4**字节）

DF 定义**6**字节

DQ 定义**8**字节

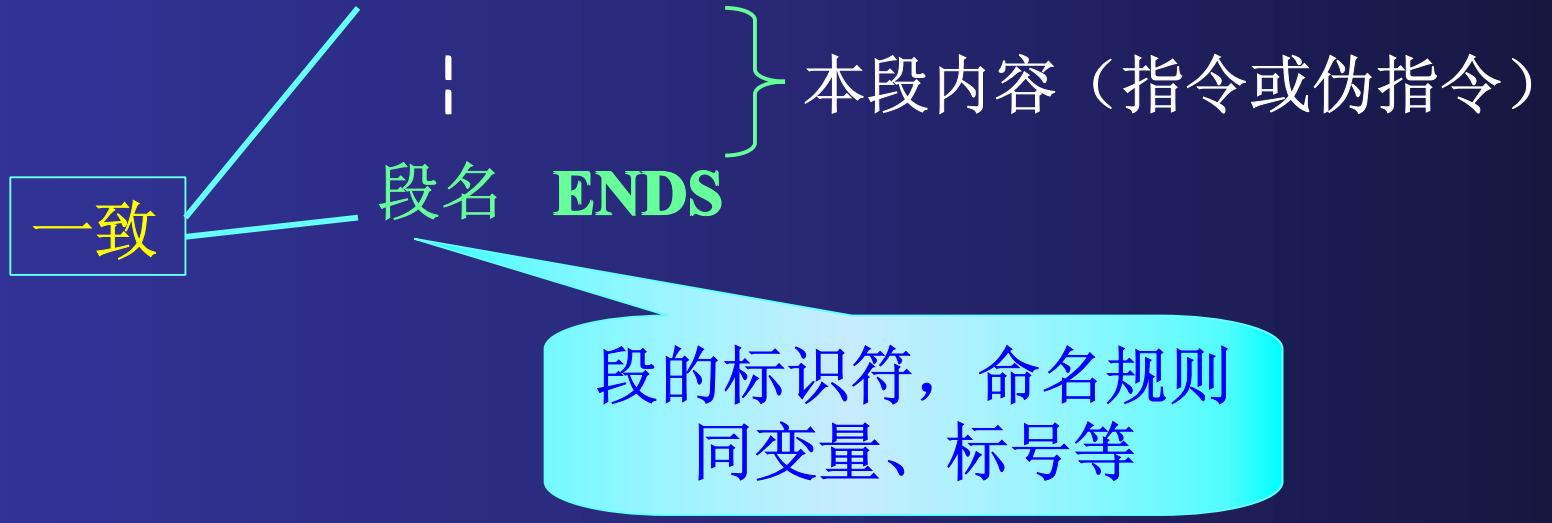
DT 定义**10**字节

3、段定义伪指令

1）SEGMENT / ENDS 伪指令

成对使用，用来指定段的名称和范围。

格式：　段名　**SEGMENT**　[定位方式] [连接方式] [‘类别名’]



<段名> **SEGMENT** [定位方式] [连接方式] [‘类别名’]

…

<段名> **ENDS**

（1）定位方式

BYTE 指定起始地址是任意值

WORD 指定起始地址的最低位是**0**

PARA 指定起始地址的低**4**位是**0**，******0H**（隐含）

PAGE 指定起始地址的低**8**位是**0**，*****00H**

⑵组合类型（连接方式）

组合类型告诉连接程序本段与其它段的关系。有默认、**PUBLIC**、 **COMMON**、 **STACK**、 **MEMORY** 和 **AT** **6**种。

默认：不组合，表示 本段与其他段逻辑上不发生关系
但同一模块的同名逻辑段则集中为一个逻辑段

PUBLIC：同名逻辑段组合

COMMON：对同名逻辑段从同一个地址开始装入

STACK：堆栈区同名逻辑段组合

MEMORY：连接时本段在地址最高处

AT 表达式：根据表达式值定位段地址；

例： **AT 8A00H**

；本段段地址为**8A00H**，从物理地址 **8A000H**开始装入。

(3) ‘类别名’

类别名是用单引号括起来的字符串，连接时决定各逻辑块的装入顺序。

当几个程序模块进行连接时，具有相同类别名的逻辑段被装入连续的内存区，按出现的先后顺序排列。

所有没有·类别名的逻辑段一起连续装入内存。

编程一般采用默认方式：

段名　**SEGMENT**

　　⋮

段名　**ENDS**

段起始地址的低**4**位是**0**

不组合，

无类别

对**8086**　只有**4**个段是当前段！！

一个段一经定义，其中指令、变量等在段内的偏移地址就已确定，整个段占用的存储空间大小也就确定。

```
DATA  SEGMENT 'DATA'
      ┆
DATA  ENDS
STACK  SEGMENT  STACK'STACK'
      ┆
STACK  ENDS
EXTRA  SEGMENT
      ┆
EXTRA  ENDS
CODE  SEGMENT
      ┆
START:  PUSH  DS
      ┆
CODE  ENDS
END  START              ；源程序模块结束
```

定义数据段

定义堆栈段

定义附加段

定义代码段

用**SEGMENT---ENDS** 定义的段称为逻辑段，各段内容如下：

数据段：　预置初值，预留内存单元

　　只能是伪指令

定义之后，如何判断是那种类型的段？

堆栈段：设置堆栈空间

　　只能是伪指令

附加段：预置初值，预留内存单元

　　只能是伪指令（目标串）

ASSUME ！ ！

代码段：存放程序指令，也可兼当数据段

4、段寄存器说明伪指令

格式：

ASSUME 段寄存器：段定义名**1[**，段寄存器：段定义名**2**，**...]**

例**3**：**ASSUME CS**：**CODE**，**DS**：**DATA**，**ES**：**DATA**，**SS**：**STACK**

ASSUME 伪指令用来告诉汇编程序，在指令执行期间内存的哪一段是数据段，哪一段是堆栈段，哪一段是代码段。

- **ASSUME**语句习惯上作为代码段的第一条指令
- **ASSUME**语句是非执行性的伪指令，段寄存器的初值必须在程序中用指令设置。

- 对**CS**的赋值由操作系统装入**EXE**文件时自动完成。

例：

```
MYDATA    SEGMENT
XX2       DW ?
XX3       DD ?
MYDATA    ENDS
MYEXTRA   SEGMENT
YY1       DB 25H，36H
YY2       DW ?
MYEXTRA   ENDS
MYSTACK   SEGMENT
STA       DW 100 DUP（？）
TOP       EQU LENGTH STA
MYSTACK   ENDS
MYCODE    SEGMENT
          ASSUME CS：MYCODE
          ASSUME DS：MYDATA
          ASSUME ES：MYEXTRA
          ASSUME SS：MYSTACK
START：   MOV AX，MYDATA
          MOV DS，AX
          MOV AX，MYSTACK
          MOV SS，AX
          MOV AX，MYEXTRA
          MOV ES，AX
          MOV AX，TOP*2
          MOV SP，AX      ；设堆栈指针

MYCODE    ENDS
          END START
```

数据段

附加段

堆栈段

说明各逻辑段的性质

为各段寄存器装入实际值

7、定位伪指令

格式： **ORG** < 表达式 >

这里表达式是一个无符号数，表示该指令以下的程序或数据的开始地址。

例4：下面程序段，指出变量**BUF**和**NUM**的偏移地址为多少？

```
DATA    SEGMENT
        ORG   10h
        BUF   DB    'ABCD'
        ORG   100h
        NUM   DW    50
DATA    ENDS
```

变量BUF的偏移地址为10h，变量NUM的偏移地址为100h

四、过程定义语句

过程是定义的独立的程序段，可被其它程序段调用。当过程执行完毕，需要返回到断点地址。

过程调用及返回均分为两种情况：段内及段间。

调用指令 **CALL**，返回指令 **RET**

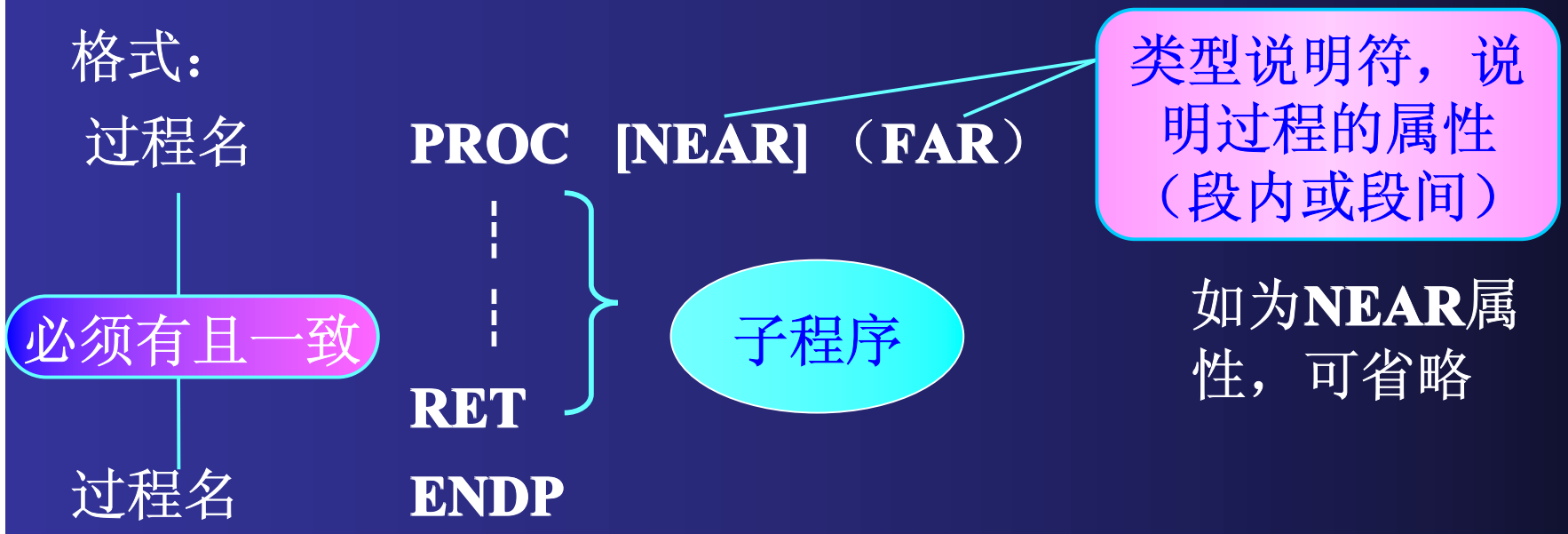
段内的调用及返回指令只将段内的偏移地址入栈或弹出；

段间的调用与返回指令需将段基址和段内偏移地址都入栈保护或弹出。

过程定义伪指令PROC 和ENDP

成对出现，两条伪指令之间的内容是过程定义，即一个子程序。

格式：



过程定义语句限定了一个过程，并指出它是一个**NEAR** 或 **FAR** 过程，帮助汇编程序明确 **CALL**（调用）和 **RET**（返回）指令的性质，是段内还是段间。

例：

```
MYCOCD    SEGMENT
XX        PROC  NEAR
          DEC  CX
          RET
XX        ENDP
START:    MOV  CX，0FFAH
            ⋮
          CALL  XX
            ⋮

MYCOCD    ENDS
          END  START
```

定义XX为NEAR属性

汇编为段内返回

汇编为段内调用

五、结束语句

标志着整个源程序的结束。

格式为：**END** <表达式>

这个地址是当程序执行时，第一条要执行的指令地址

功能：源程序结束

汇编结束

例：内存数据区有2个字数据，求其差并存入同一数据区SULT单元中

```
DATASEG   SEGMENT
    XX1       DW  1234H
    YY1       DW  4567H
   SULT       DW  ?
DATASEG   ENDS
```

```
CODESEG   SEGMENT
          ASSUME CS: CODESEG
          ASSUME DS: DATASEG
START:    MOV AX, DATASEG
          MOV DS, AX
          MOV AX, XX1
          SUB AX, YY1
          MOV SULT, AX
          MOV AH, 4CH
          INT 21H
CODESEG   ENDS
          END START
```

返DOS！

3.2.3 汇编语言程序设计

一、程序设计步骤

1、分析问题

2、确定算法——化繁为简

3、画流程图，简单程序也可以不画流程图，直接编程。

4、内存空间分配，根据题目定义变量

5、编写程序

6、静态检查

7、上机调试

二、顺序、循环与分支程序设计

1、简单程序（顺序结构）

例1：用数据运算指令对两个两字节数做加法，这两个数从地址10050H开始连续存放，结果放在这两个数之后。

分析题目，两字节数相加，用字加法指令，本题可以不考虑进位问题。

确定算法，可以用不带进位的加法指令；若用带进位加法指令，事前先清除进位。

二、顺序、循环与分支程序设计

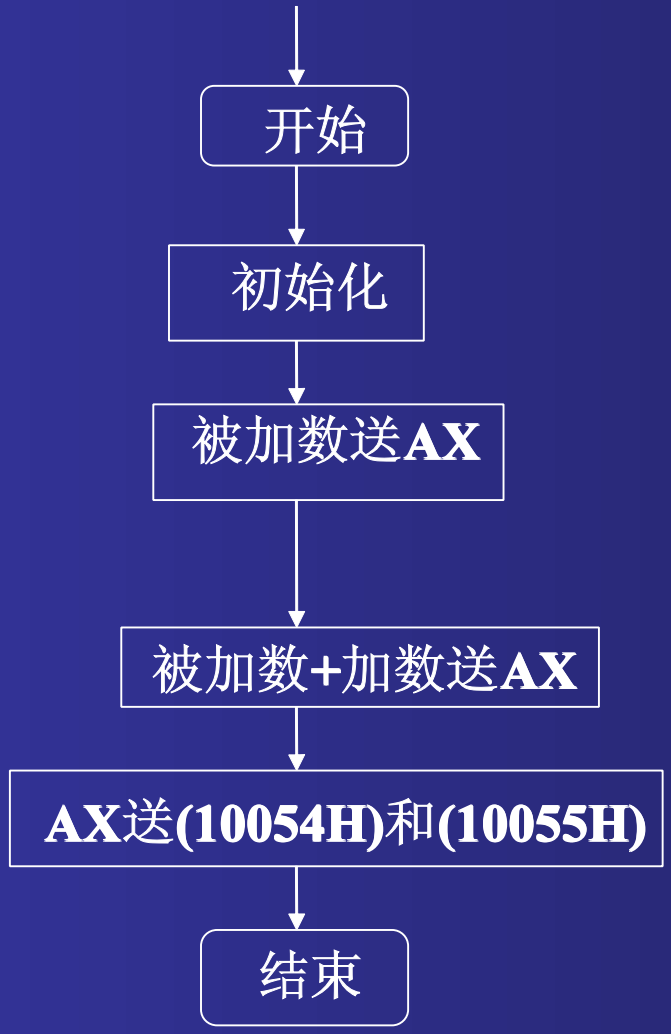
1、简单程序（顺序结构）

例1：用数据运算指令对两个两字节数做加法，这两个数从地址10050H开始连续存放，结果放在这两个数之后。

画流程图；

内存空间分配：

内存地址	内容
10050H	被加数低8位
10051H	被加数高8位
10052H	加数低8位
10053H	加数高8位
10054H	和数低8位
10055H	和数高8位

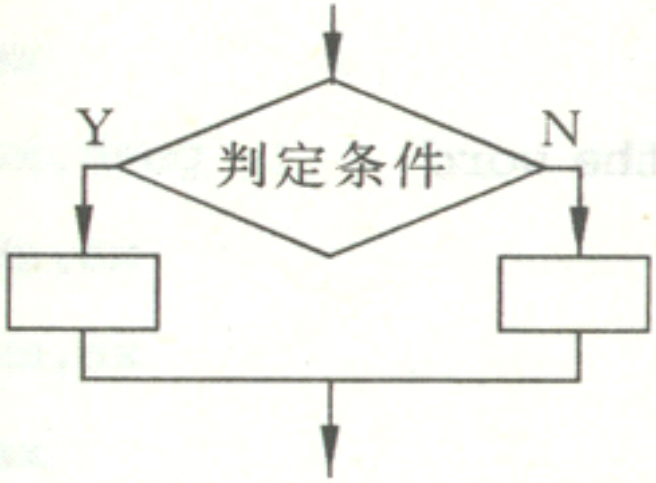


编写程序：

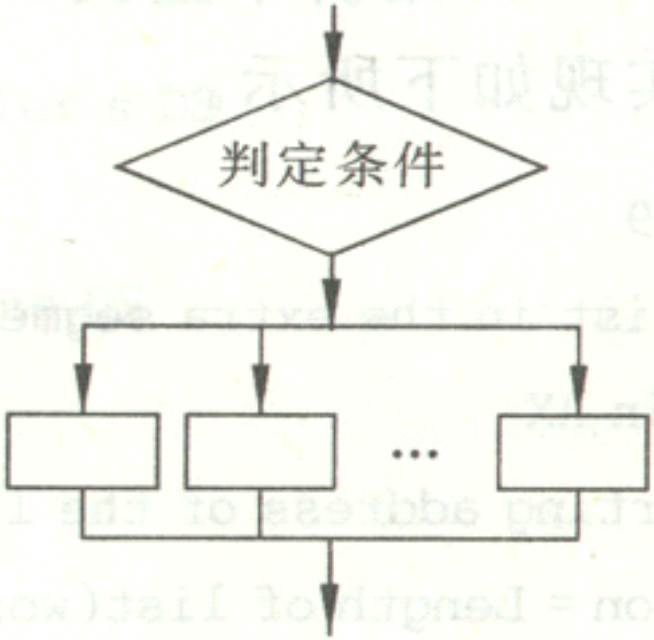
```
mov ax, 1000h；初始化
mov ds, ax
mov si, 50h
mov di, 52h
mov bx, 54h
clc              ；清除进位
xor ax, ax       ；ax清零
mov ax, [si]
adc ax, [di]     ；带进位加法
mov [bx], ax
hlt
```

2、分支程序

分支程序的结构形式



(1) IF_THEN_ELSE结构



(2) CASE结构

图 5.10　分支程序的结构形式

例3：求AX和BX中两个无符号数之差的绝对值，结果放在内存2800H单元。

```
        clc                 ;  清除进位
        mov   di,  2800H
        sub   ax,  bx
        jc    aa            ;  有借位转移
        mov  [di], ax       ;  ax大于等于bx
        JMP bb
aa:  neg  ax                ;  ax小于bx
        mov  [di], ax
bb:  hlt
```

例4：比较数据段中两个无符号字节数的大小，把大数存入MAX单元

解：比较两个无符号数，可采用CF标志位来判断大小

```
DATA      SEGMENT
SOURCE    DB      12，34              ；两个无符号数X1和X2
MAX       DB      ？                  ；保留存放大数的单元
DATA      ENDS
CODE      SEGMENT
          ASSUME CS：CODE，DS：DATA
BEGIN：   MOV   AX，DATA
          MOV   DS，AX
          MOV   AL，SOURCE            ；X1送AL
          CMP   AL，SOURCE+1          ；X1-X2
          JNC   BRANCH                ；若X1>X2，转BRANCH
          MOV   AL，SOURCE+1          ；否则，X2送AL
BRANCH：  MOV   MAX，AL               ；大数送MAX单元
     CODE ENDS
          END BEGIN
```

3、循环程序

(1) 循环程序的结构形式

循环程序的结构有两种形式：

DO_WHILE结构和**DO_UNTIL**结构。

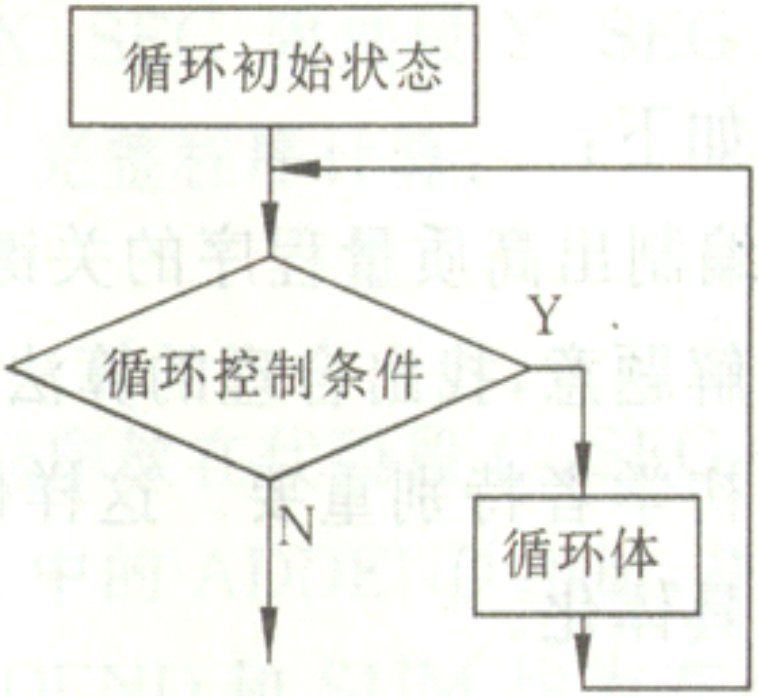
① DO_WHILE结构

对循环控制条件的判断放在循环的入口，先判断条件，满足条件就执行循环体，否则就退出循环。

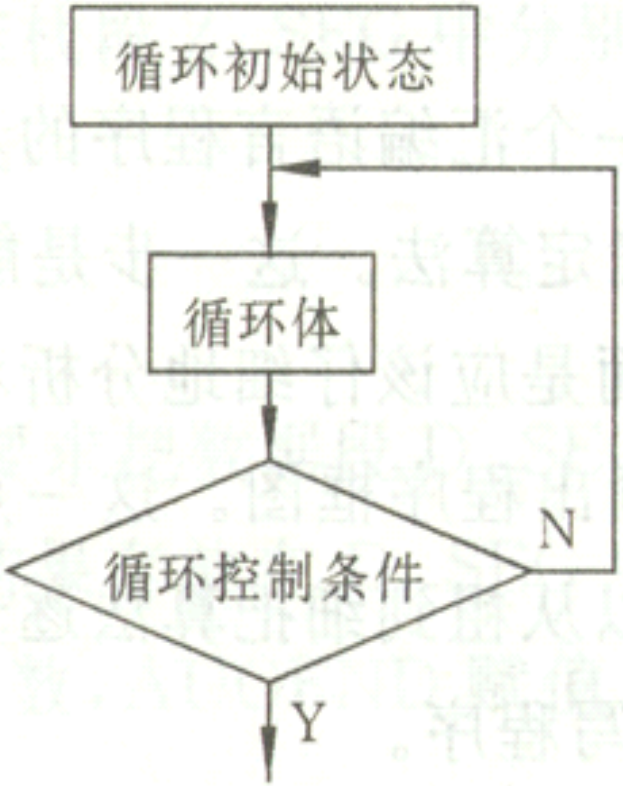
② DO_UNTIL结构

先执行循环体，然后再判断控制条件，不满足条件则继续执行循环操作，一旦满足条件就退出循环。

如下图所示。



(1) DO_WHILE结构



(2) DO_UNTIL结构

图 5.1　循环程序的结构形式

例5：求两个多字节数之和，这两个数在10050H单元开始的内存中连续存放，结果存放在两数之后，设这两个数均为8个字节长。

```
Start:   mov   ax, 1000H        aa:  mov    ax, [si]
         mov   ds, ax                adc    ax, [di]
         mov   si, 50H               mov    [bx], ax
         mov   di, 58H               pushf
         mov   bx, 60H               add    si，  2
         mov   cx, 4                 add    di，  2
         clc                         add     bx，  2
                                     popf
                                     loop   aa
                                     hlt
；第一个数在10050H-10057H，第二个数在10058H-1005FH
；结果数据在10060H-10067H
```

例6：用循环结构编写程序完成

$$SUM = \sum_{i=1}^{10} a_i = a_1 + a_2 + \cdots + a_{10}$$

```
DATA    SEGMENT
BUFFER  DW   a1,a2,…,a10   ；原始10个字数据
SUM     DW   ?             ；存放和数
DATA    ENDS
CODE    SEGMENT
    ASSUME  CS：CODE，DS：DATA
START：MOV   AX，DATA
       MOV   DS，AX
       MOV   AX，0         ；AX中为和数，初始为0
       MOV   DI，OFFSET SUM；存放结果地址送DI
       MOV   BX，OFFSET BUFFER
                           ；数据缓冲区首地址送BX
```

```
        MOV     CX，10          ；循环次数送CX
LOP:    ADD     AX，[BX]        ；累加
        INC     BX              ；修改数据缓冲区地址
        INC     BX
        LOOP    LOP             ；到10次了吗？不到，转LOP
        MOV     [DI]，AX        ；到10次，存放结果和
        INT     20H             ；返回DOS

CODE    ENDS
        END     START
```

四、程序设计举例

(1) 起泡算法排序（从大到小或从小到大）

从第一个数开始依次对相邻两个数比较，若次序对，则不做任何操作，如次序不对，则使这两个数交换位置。这种算法称为起泡算法。

序号	数	第一遍	第二遍	第三遍	第四遍
1	8	8	16	32	84
2	5	16	32	84	32
3	16	32	84	16	16
4	32	84	8	8	8
5	84	5	5	5	5

第一遍进行4（N-1次）比较，最小数已到最下面，第二遍进行3（N-2次）比较，依次类推，最多进行N-1遍即可完成排序。

例：首地址为ARER的字数组，从大到小排序。

```
Data    segment
    area            dw   n dup(?) ；  n个数
Data    ends
Code    segment

    assume cs:code,ds:data
Start:
        mov     ax, data
        mov     ds, ax
        mov     cx, n
        dec     cx
```

```
Loop1:mov    di, cx
      mov    bx, 0
Loop2:mov    ax, area[bx]
       cmp   ax, area[bx+2]
      jge    con；大于等于，转con
      xchg   ax, area[bx+2]
      mov    area[bx], ax
Con:  add     bx, 2
                loop   loop2
                mov    cx, di
                loop   loop1
Code  ends
                end    start
```

注：loop2循环进行每遍中的比较

loop1循环控制比较多少遍

八、简化段定义

在高版本的宏汇编语言中增加了许多新的伪指令，下面只介绍有关模块定义和段定义伪指令。

1、模块定义伪指令（.MODEL）

格式：

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

（1）存储模式

微模式**tiny**、小模式**small**、中模式**medium**、紧缩模式**compact**、 大模式**large**、巨模式**huge**

各种存储模式的比较见下表：

存储模式	代码段	数据段	容量限制	代码的距离属性	数据的距离属性
tiny	代码和数据组合成一个段		代码+数据≤64K	**Near**	**near**
small	**1**	**1**	代码≤ 64K 数据≤ 64K	**near**	**near**
medium	多个	**1**	数据≤ 64K	**far**	**near**
compact	**1**	多个	代码≤ 64K	**near**	**far**
large	多个	多个	无	**far**	**far**
huge	多个	多个	无	**far**	**far**

2、段的定义

一旦使用了**.MODEL**伪指令定义了存储模式，就可以使用简化段定义伪指令定义段。

代码段由**. CODE**伪指令定义，其格式为：

. CODE [段名**];**

堆栈段由**.STACK**伪指令定义，其格式为：

. STACK [大小**]**；缺省为**1024**字节，段名**STACK**

数据段定义有三条伪指令：

（1）**.DATA**：定义初始化的数据段

格式： **.DATA**

（2） **.DATA**？：定义未初始化的数据段

格式： **.DATA**？

（3）**.CONST**：定义常数数据段

格式： **.CONST**

例8：使用简化段定义，调用DOS的字符串显示输出中断向屏幕输出一个字符串。

```
.MODEL      small    ；定义存储模式：小模式
.STACK      256      ；定义堆栈段，256字节
.DATA                ；定义数据段
msg    db    "This program illustrates the usage of"
       db    "complete segment control directives."
       db    0dh，0ah
       db    "$"
.CODE                ；定义代码段
       mov   ax，@data
       mov   ds，ax
       mov   ah，09h
       mov   dx，offset msg
       int   21h
       mov   ax，4c00h
       int   21h
       END
```