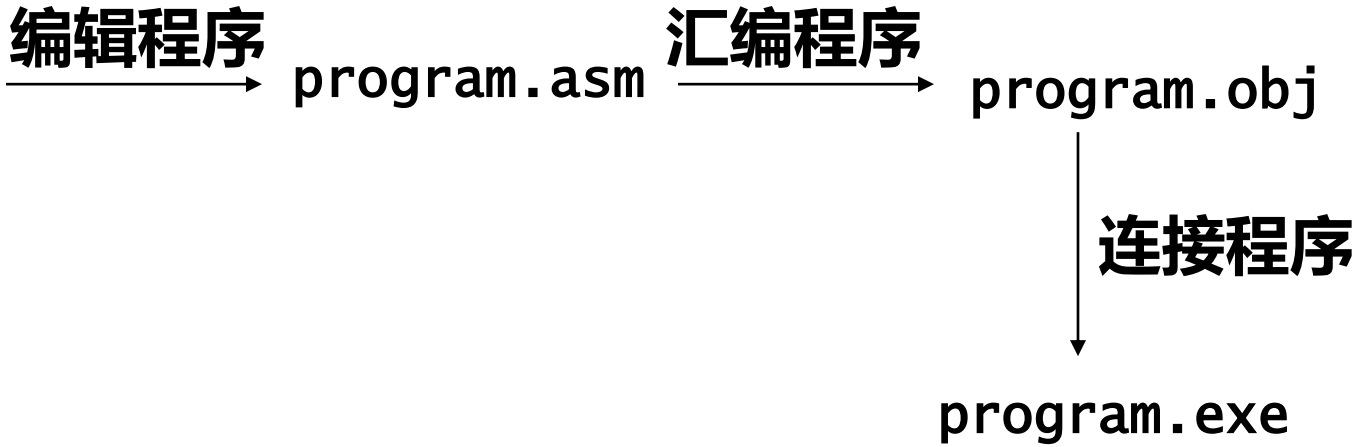
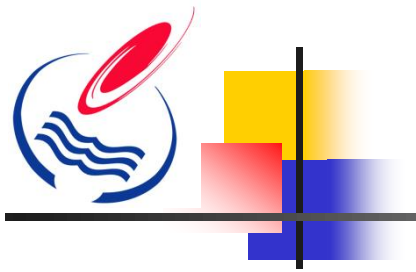
汇编语言程序设计-补充

§3.1 、汇编与连接

1、





2、编程调试过程

- 第一步：编辑　　EDIT　　文件名（.ASM）
- 第二步：汇编　　MASM　　文件名（.ASM）
- 第三步：连接　　LINK　　文件名（.OBJ）
- 第四步：运行　　文件名（.EXE）
- 第五步：调试　　DEBUG　文件名.EXE

3、DEBUG命令

DEBUG.COM是DOS外部命令，命令提示符：‘—’，

功能特点：

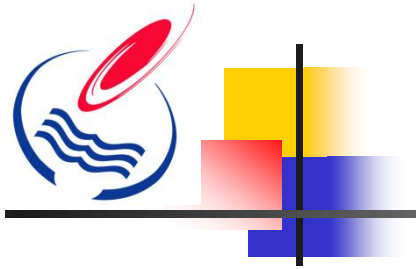
（1）可在最底层环境下运行，调试方便、速度快。

（2）也可直接输入汇编程序，然后调试运行该程序。。

（3）还可以直接用来检查和修改内存单元、装入、存储及启动运行程序、检查及修改寄存器，也就是说DEBUG可深入到计算机的基本级，可使用户更紧密地与计算机中真正进行的工作相联系。

缺点：不能使用浮号地址，也不能使用绝大多数ASM和MASM提供的伪指令。

命令： AU，DEFR，GPTQ，MC



汇编命令　A

格式：-A <ADDRESS>

功能：该命令允许键入汇编语言语句，并能把它们汇　编成机器代码，相继地存放在从指定地址开始的存储区中。

注意：DEBUG把键入的数字均看成十六进制数

例：

```
-A100
169C：0100    MOV     DL，33
169C：0102    MOV     AH，2
169C：0104     INT      21
```

显示内存命令　D

格式：(1)　- D <地址>

(2)　- D <范围>

(3)　- D

功能：以两种形式显示指定内存范围内容，一种形式为十六进制内容，一种形式为相应字符的ASCII码字符，对不可见字符以‘.’代替。

例：

-D100

169C：0100　B2 33 B4 02 CD 35 CD　61　. 3 . . . 5 . a

修改内存命令　E

格式：

(1)　- E <地址>　< 单元内容>

(2)　- E <地址>　<单元内容表>

其中<单元内容>是一个十六进制数,或用引号‘或“括起来的字符串；<单元内容表>是以逗号分隔的十六进制数，或用‘或“括起来的字符串，或者是二者的结合。

功能：

(1)将指定内容写入指定单元后显示下一地址，可继续键入修改内容，直至新地址出现后键入回车止。

(2)将<单元内容表>逐一写入由地址开始的一片单元。

- **填充内存命令　F**

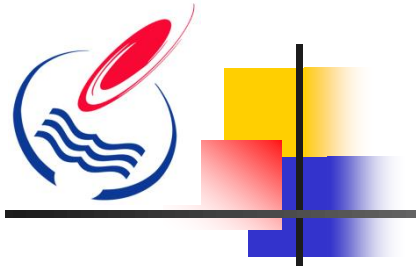
  格式： - F <范围> < 单元内容>

  功能： 将单元内容表中的值逐个写入指定范围，单元内容表内容用完后重复使用。

  例如：

  -F 1600：100 L 8 B2，’XYZ’

  说明：长度为8个字节，B2，’X’，’Y’，’Z’为四个字节，再重 复填充一次。



- **执行命令　G**

  格式：

  (1)　- G

  (2)　- G <=地址>

  (3)　- G <=地址>，<断点>

  功能：

  (1)　从CS：IP开始执行

  (2)　从指定地址开始执行

  (3)　从指定地址开始执行，到断点自动停止

- **执行命令**

  G、P、T三者的区别

  (1) G：执行整个程序结果

  (2) P：执行当前指令，但若当前指令是过程或函数或中断，则一次执行完

  (3) T：执行当前指令，并可进入过程或函数体

- **结束DEBUG，返回DOS命令　Q**

  格式：Q

  功能：返回DOS提示符下

• 反汇编命令 U

格式： (1) - U <地址>

(2) - U <范围>

功能： (1) 从指定地址开始反汇编

(2) 对指定范围内的存储单元进行反汇编

例：

```
-U100  106
169C：0100  B233        MOV   DL,33
169C：0102  B402        MOV   AH,02
169C：0104  CD21        INT    21
169C：0106  CD20        INT    20
```

- **十六进制算术运算指令H**

格式： H [值1] [值2]

功能：求十六进制数[值1]和[值2]的和与差并显式结果。

- **端口输入命令I**

格式： I [端口地址]

功能：将从指定端口输入 的内容显式出来。

- **端口输出命令O**

格式： O [端口地址] [字节]

功能：将该[字节]从指定[端口地址]输出。

例如：O 2F 4F 将4FH从2FH口输出

- **内存搬家命令M**

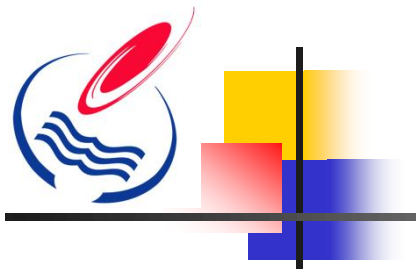
格式： M [源地址范围] [目标起始地址]

其中源及目标地址若仅输入偏移量，则隐含相对DS。

功能：把[源地址范围]中的内容顺序搬至[目标起始地址]起的一片连续单元。

例如：M CS:100 110 600

把从CS:100起至CS:110止17个字节搬至DS:600至DS:610的一片单元。



- **比较命令C**

格式： C [源地址范围] ， [目标地址]

其中[范围]是由[起始地址] [终止地址]指出的一片连续单元，或[起始地址] L [长度]。

功能：从[源地址范围]的起始地址单元起逐个与目标起始地址以后的单元顺序比较单元的内容，直到源终止地址为止。遇到不一至时，以 [源地址] [源内容] [目标内容] [目标地址]的形式显式失配单元内容。

- **显示寄存器命令R**

格式： (1) R

(2) R [寄存器名]

功能：(1)显式当前所有寄存器内容、状态标志及将要执的下一指令的地址、代码及汇编语句形式。其中对状态标志FLAG以每位的形式显式。详见下页表3－1。

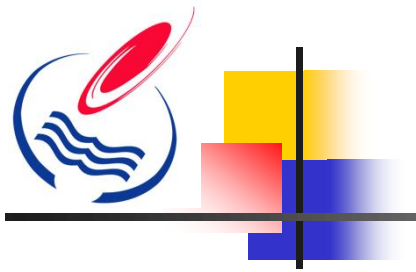
(2)显式指定寄存器内容

例如：

R AX

表3－1　状态标志显式形式

标志位	状态	显式形式
溢出标志OF	有/无	OV/NV
方向标志DF	减/增	DN/UP
中断标志IF	开/关	EI/DI
符号标志SF	负/正	NG/PL
零标志ZF	零/非	ZR/NZ
辅助标志AF	有/无	AC/NA
奇偶标志PF	偶/奇	PE/PO
进位标志CF	有/无	CY/NC



- **搜索指定内存命令S**

格式： S [地址范围] [表]

功能：在指定范围搜索表中内容，找到后显式表中元素所在地址。

例如：S CS:100 110 41

显式：04BA:0104

04BA:010D

表示在位移100H至110H间的上述两处有41H。

又如：S CS:100 L 11 41”AB”E

表示在当前代码段位移100H至111H处找连续4个字节内容为41H、41H、42H、0EH的地址。

- **执行并显式系统环境命令T**

格式： T [=地址] [条数]

功能： [地址]的缺省值是当前IP值，条数的缺省值是一条。执行由指定地址起始的，由[条数]指定的若干条命令。

例如：

T 执行当前指令并显式状态

T 10 从当前指令开始执行10H条指令

§3.2、源程序的书写格式和数据组织

一、字符的输入与输出

二、伪指令

三、数据的组织

汇编语言源程序书写结构形式；
如何安排程序中的数据，数据的分配和预置；
熟悉与程序基本结构密切相关的伪指令。

一、字符的输入与输出

DOS功能调用：汇编程序通过INT 21H软中断来调用DOS内部子程序完成特定操作。 调用前将功能号送AH寄存器，根据要求准备好各种参数，然后执行INT 21H。

1、从键盘输入：

```
MOV     AH，01H
INT       21H
```

等待从键盘输入一个字符，输入字符的ASCII码存在AL中。

2、输出到显示器：

```
MOV     DL，41H
MOV     AH，02H
INT       21H
```

输出ASCII码对应的字符在屏幕上。

3、输入字符串：

DS：DX指向内存缓冲区首址，然后执行

```
MOV  AH，0AH
INT  21H
```

结果：以回车键结束的字符串输入到内存缓冲区。

4、输出显示字符串：

DS：DX指向内存缓冲区首址，然后执行

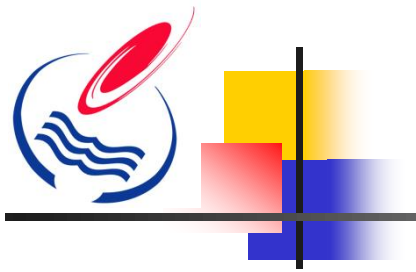
```
MOV  AH，09H
INT  21H
```

结果：内存缓冲区中以$结束的字符串显示在屏幕上。

二、 伪指令

伪操作是汇编程序对源程序进行汇编时处理的操作，完成处理器选择、存储模式定义、数据定义、存储器分配、指示程序开始结束等功能。

伪指令本身，并不是程序码，而是在进行汇编时，汇编程序(MASM)会去看得懂这些伪指令，而加以处理我们所编写的程序码，常用到的伪指令如下：



- 处理器选择伪操作
- 段定义伪操作
- 程序开始和结束伪操作
- 数据定义及存储器分配伪操作
- 表达式赋值伪操作
- 地址计数器与对准伪操作
- 基数控制伪操作

1、处理器选择伪操作：

.8086　选择 8086 指令系统

.286　选择 80286 指令系统

.286P　选择保护模式下的 80286 指令系统

.386　选择 80386 指令系统

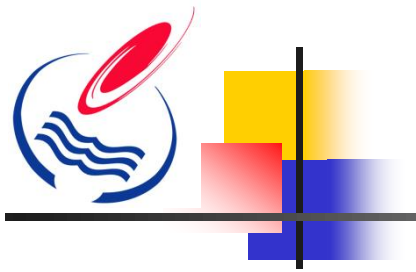
.386P　选择保护模式下的 80386 指令系统

.486　选择 80486 指令系统

.486P　选择保护模式下的 80486 指令系统

.586　选择 Pentium 指令系统

.586P　选择保护模式下的 Pentium 指令系统



注：若指定.286，那麼程序在汇编时，就无法汇编386以上的组合语言指令码，例如 EAX与EBX 等32位元的寄存器，只有386以上CPU才有，所以若程序里有 mov eax,ebx 等指令时，汇编时会发生汇编错误的信息，只有将微处理器模式改回 .386 或 .486 才会汇编成功。

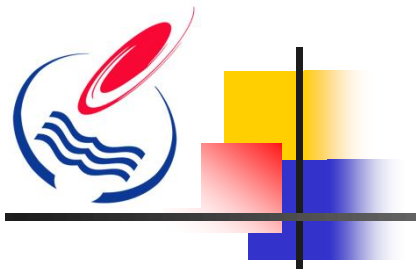
2、段定义伪操作：

- 完整的段定义伪操作：适用于所有版本的汇编语言。

```
data    segment             ;定义数据段
        …
data    ends

extra   segment             ;定义附加段
        …
extra   ends

code    segment             ;定义代码段
        assume cs:code, ds:data, es:extra
start:
        mov    ax, data
        mov    ds, ax       ;段地址 → 段寄存器
        …
        mov    ah, 4ch
        int    21h          ;返回DOS
code    ends
        end    start
```



（1）段定义伪指令：SEGMENT/ENDS

格式：段名　SEGMENT[定位类型][组合类型][寻址方式][类别]

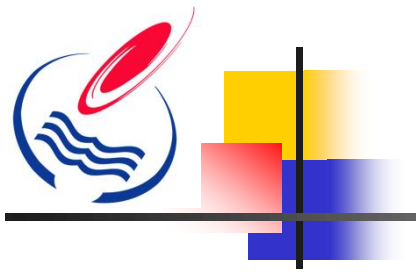
……

段名　ENDS

定位类型（align）：用来指示各段起始边界的性质。可以有如下五钟，缺省时是PARA类型。

BYTE：　字节，任何地址均可。

WORD：　字，　段地址必须为偶数。
DWORD：双字，4的倍数
PARA：　节，　16的倍数
PAGE：　页，　256的倍数

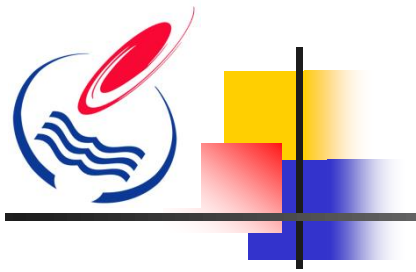


组合类型（combine)：是用来 描述连接时各段间的关系的，可以是下列六种类型之一。

AT表达式：连接程序把本段装在表达式的值所指定的地址上，（程序代码段不允许用AT指定）。

PRIVATE：表示本段与其它逻辑段没有关系，每段均各自设置地址。这是缺省时的设置方式。

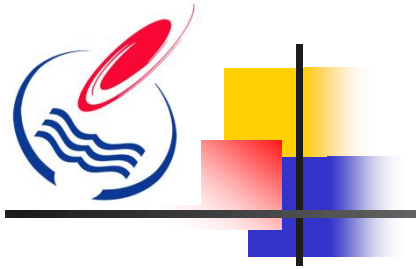
PUBLIC：连接程序把所有同名的PUBLIC段邻接存放，这样，同名的段可以连接在一起形成一个更大的段，段的总长度是所有同名段的长度和，段的总长度不能超过64KB。



COMMON：连接程序把所有同名同类COMMON段指向共同的段址，因而各段相互重叠。段的总长度是同名段中最长的段所具有的长度。这样冲突吗？其实COMMON的含义是共同的段，当多人是共同设计一个大软件时，对于某个相同的数据段可以设置成COMMON类型，这样各个模块均可在自己的程序中对该COMMON段方便地寻址。

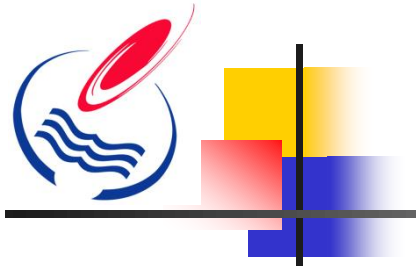
MEMORY：连接程序把本段定位在所有段的上面，即程序的最高地址处。如有多个MEMORY段，则遇到的第一个是MEMORY段，其它的作COMMON段处理。

STACK：连接程序把所有的STACK段连接成一个堆栈段。



寻址方式（USE）：主要用于80386，它可以设置为USE16或USE32，分别代表16位寻址和32位寻址，当处理器是80386时，对于80386保护模式缺省为USE32,其它模式缺省为USE16，对于8086和80286，由于不具备32位寻址能力，所以只能是16位寻址，不用设置这一项。

类别：可以是任何名字，用引号括起来，连接时用于组成段组名。属于一个给定类的所有段均被安放在一个内存邻接块中，而不管其在源模块中的顺序如何。



（2）指定段址伪指令：ASSUME

格式：ASSUME 段寄存器名：段名[，…]

说明：本伪指令只是指示各逻辑段使用寄存器的对应关系，告诉汇编器MASM各个段分别分配给了哪一个段寄存器，并不把段地址装入段寄存器。

段寄存器的装填：

```
MOV     AX，数据段名
MOV     DS，AX
```

装填数据段

```
MOV     AX，附加数据段名
MOV     ES，AX
```

装填附加数据段

```
MOV     AX，堆栈段名
MOV     DS，AX
```

装填**堆栈**段

说明：DS，ES必须在程序段中进行人工装填，

CS、SS由OS根据文件头中的信息自动装填。

源程序结束伪指令：END

格式：END［标号］

功能：表示源程序结束，并可指定程序从标号指向的指令开始执行。

外部名申明：EXTRN

格式：EXTRN　［外部名］

说明本程序文件所用到的外部模块中的过程名、标号或变量名。

过程定义：PROC/ENDP

过程的起始和结束

3、汇编语言程序的正常结束(三种方式)

方法1：

在.COM文件中或DEBUG下所写程序可用

INT 20H 或 INT 3H 结束程序。

方法2：在程序最后加

```
MOV     AX, 4C00H
MOV     AH,4CH  ；功能模块号
INT     21H     ;中断功能调用
```

方法3：过程结束

```
<程序名>    PROC    FAR
            PUSH    DS
            MOV     AX，0
            PUSH    AX
            ……
            RET
<程序名>    ENDP
```

例：下面是一个完整段定义的简单例子，该程序实现显式字符串‘Hello，world’.源程序如下：

```
DATA      SEGMENT
HM     DB    ‘Hello,world’,0DH,0AH,’$’
DATA      ENDS
CODE      SEGMENT
          ASSUME    CS:CODE,DS:DATA
          MOV     AX,DATA
          MOV     DS,AX
          MOV     DX,OFFSET   HM
          MOV     AH,9
          INT       21H
          MOV     AH,4CH
          INT       21H
CODE  ENDS
          END
```

4、表达式赋值伪操作：

表达式名　**EQU**　表达式

```
B        EQU   [BP+8]
ALPHA    EQU   9
BETA     EQU   ALPHA+18
```

5、地址计数器伪操作：

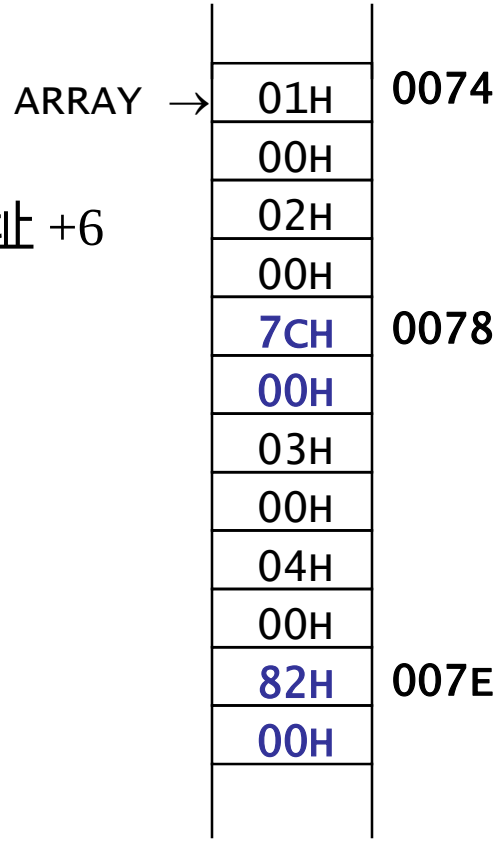
地址计数器 $ ：**保存当前正在汇编的指令的地址**

```
        ORG    $+8          ; 跳过8个字节的存储区
        JNE    $+6          ; 转向地址是 JNE 的首址 +6
```

$ 用在伪操作的参数字段：

表示地址计数器的当前值

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



ORG 伪操作：

格式：**ORG** 常数表达式

功能：将地址计数器的值置为常数表达式的值

例：

```
DSEG1    SEGMENT
         ORG   10
         VAR1  DW  1234H
         ORG   20
         VAR2  DW  5678H
         ORG   30
         VAR3  DW  1357H
DSEG1    ENDS
```

则VAR1的偏移地址为0AH，而VAR2的偏移地址为14H，VAR3的偏移地址是1EH。

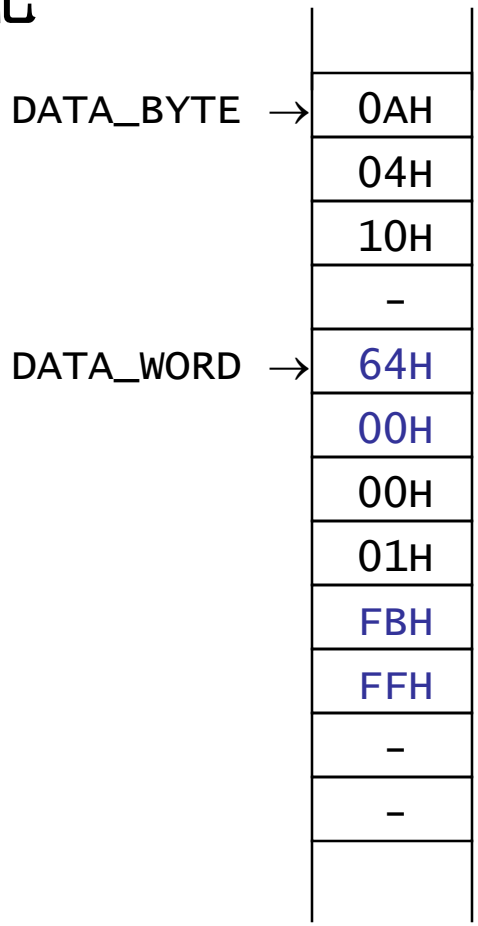
三、数据的组织

数据定义及存储器分配

[变量] 助记符 操作数[,操作数,...] [;**注释**]

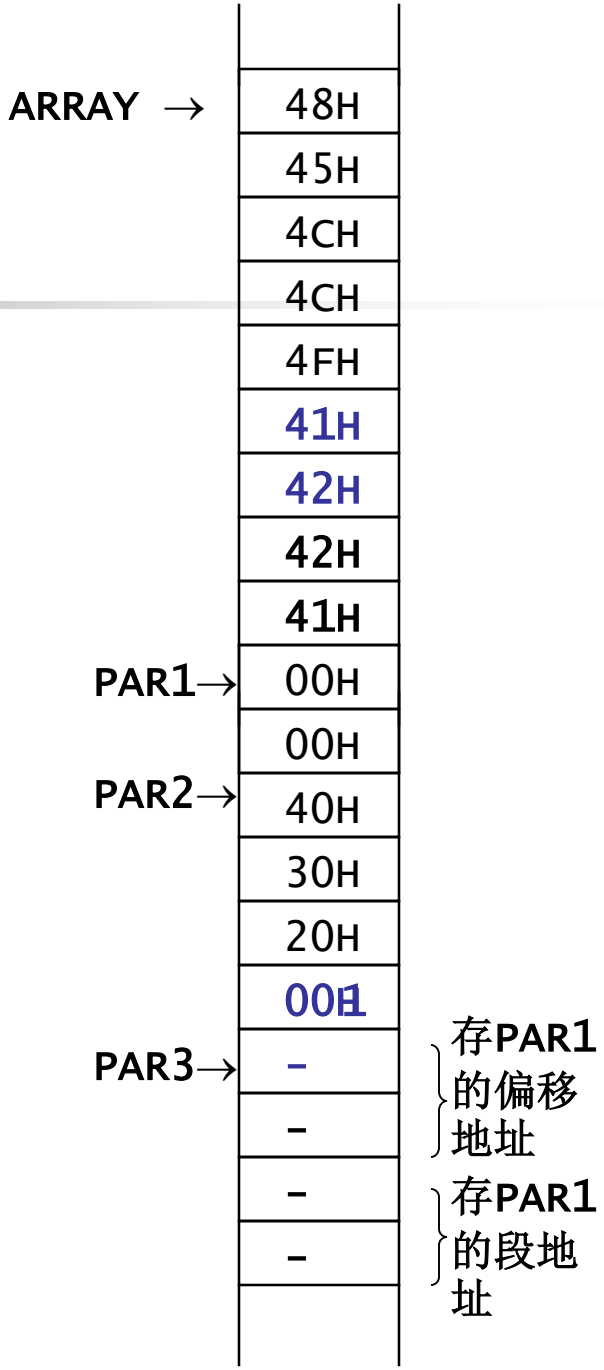
助记符：**DB DW DD**

```
DATA_BYTE   DB   10,4,10H,?
DATA_WORD   DW   100,100H,-5,?
```



```
ARRAY   DB   'HELLO'
        DB   'AB'
        DW   'AB'

PAR1    DB   2 DUP(0)
PAR2    DD   203040H
PAR3    DD   PAR1
```



- 变量的访问：

类型一致（及跟随）原则。

1、指令中两个操作数的类型必须一致，否则错；

2、若一个操作数类型确定，另一个不确定，则跟随前者而确定；

3、存储器操作数的类型可修改，使其保持一致；

4、若两个操作数类型均不确定，则错。

** 寄存器操作数类型固定；

立即操作数无类型；

定义变量时类型即已指明其类型，并可修改；

其他方式访问的存储器操作数，类型不确定，并可修改。

数据（变量）的属性及取值操作符：

OFFSET、SEG、TYPE、LENGTH、SIZE

OFFSET / SEG

变量 及标号的偏移地址 / 段地址

TYPE 变量 的类型

DB DW DD

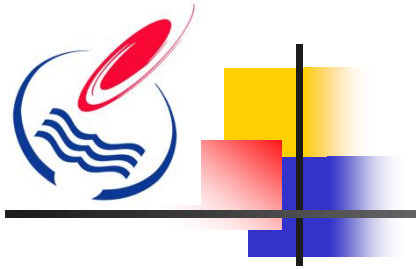
1 2 4

LENGTH 变量的长度

功能：由DUP定义的变量的单元数，其它情况为1

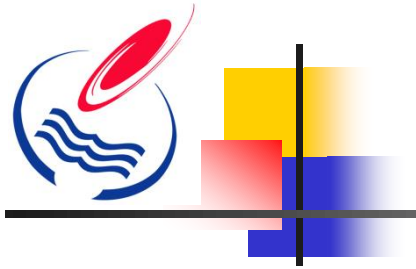
SIZE 变量的大小

功能：LENGTH * TYPE



```
ARRAY   DW   100 DUP (?)
TABLE   DB   'ABCD'

ADD   SI,   TYPE     ARRAY   ;  ADD  SI, 2
ADD   SI,   TYPE     TABLE   ;  ADD  SI, 1
MOV   CX,   LENGTH   ARRAY   ;  MOV  CX, 100
MOV   CX,   LENGTH   TABLE   ;  MOV  CX, 1
MOV   CX,   SIZE     ARRAY   ;  MOV  CX, 200
MOV   CX,   SIZE     TABLE   ;  MOV  CX, 1
```



属性的修改： PTR 操作符、THIS 操作符

类型 PTR 表达式：

```
MOV   WORD PTR [BX], 5
```

变量的双重定义：

```
TA    EQU  THIS  BYTE
NEXT  DW   12,3344H
```

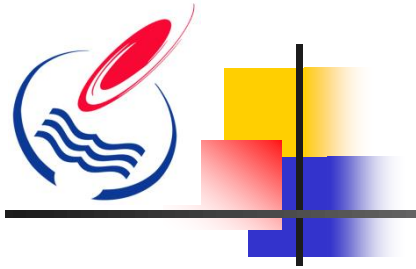
表达式：

(1) 算术操作符： +、－、*、/、mod

```
ARRAY    DW    1,2,3,4,5,6,7
ARYEND   DW    ?
MOV   CX,  (ARYEND-ARRAY)/2

MOV   DX,  BLOCK+(6-1)*2

ADD   AX,  BLOCK+2      ;   符号地址±常数 有意义
                        ; * / 时意义不明确
MOV   AX,  BX+1         ; ×
```



(2) 逻辑和移位操作符： AND、OR、XOR、NOT、SHL、SHR

```
OPR1   EQU   1
OPR2   EQU   7
AND   AX, OPR1 AND OPR2                ; AND AX,1

MOV    AX,   0FFFFH SHL 2        ; MOV AX,0FFFCH

IN     AL, PORT_VAL
OUT    PORT_VAL AND 0FEH, AL
```

(3) 关系操作符： EQ、NE、LT、LE、GT、GE

计算结果为**逻辑值**：**真** 0FFFFH

假 0000H

```
MOV FID, (OFFSET Y - OFFSET X) LE 128
```

若 ≤128 (真) **汇编结果：** `MOV FID, -1`

若 >128 (假) **汇编结果：** `MOV FID, 0`