3.1 宏汇编语言的基本语法

3.1.1 常量与数值表达式

1.常量

2.数值表达式

算术运算

逻辑运算

关系运算

3.1.2 变量、标号与地址表达式

1. 变量

变量的属性

变量的定义

2. 标号

3. 地址表达式

①地址表达式的定义

②接触过的地址表达式

③地址表达式的属性

④地址表达式与数值表达式区别

⑤特殊运算符

(I)类型运算符PTR

使用PTR注意事项：

(II)属性分离算符

取段址算符SEG

取偏移算符OFFSE

取类型运算符TYPE

(III)定义类型运算符THIS

⑥使用地址表达式的注意事项

3.2 常用的机器指令语句

3.2.1 80X86指令集及其特点

80X86指令集

特点

注意事项

3.2.2 数据传送指令

1.一般数据传送指令

(1) 传送指令

A. 一般传送指令 MOV

B. 有符号数传送指令

C. 无符号数传送指令

(2) 数据交换指令

A. 一般数据交换指令

~~B.字节交换指令~~

~~C.字节加指令~~

(3) 查表转换指令XLAT

2.地址传送指令

(1) 传送偏移地址指令

(2) 传送偏移地址及数据段首址指令

3.2.3 算术运算指令

1. 加运算指令　ADD、INC

2. 减运算指令　SUB、DEC、NEG、CMP

3. 乘运算指令　IMUL、MUL

(1)有符号乘指令

①双操作数的有符号乘指令

②三个操作数的有符号乘指令

③ 单操作数的有符号乘指令

(2)无符号乘指令

4. 符号扩展指令 CBW、CWD、CWDE、CDQ

(1) 将字节转换成字指令
(2) 将字转换成双字指令
(3) CWDE
(4) CDQ
5. 除运算指令 IDIV、DIV
(1) 无符号除指令
(2) 有符号除指令
(3) 有符号数，无符号数除法的区别
3.2.4 位操作指令
1.逻辑运算指令
(0) 求反指令 NOT
(1) 逻辑乘指令AND
(2) 测试指令TEST
(3) 逻辑加指令OR
几点总结
(4) 按位加指令XOR——异或
(5) 位操作指令的特点
2.移位指令
有统一的语句格式
移位指令的特点
(1) 算术、逻辑移位指令
①算术左移和逻辑左移指令SAL/SHL（这两者是一样的）
②逻辑右移指令SHR
③算术右移指令SAR
(2) 循环移位指令
①循环左移指令
②循环右移指令
③带进位位的循环左移指令
④带进位的循环右移指令

3.3 伪指令语句

1. 伪指令的基本概念
问题：伪指令与机器指令的区别?
2. 处理器选择伪指令
3. 符号定义伪指令
4. 段定义伪指令
(1)段定义伪指令
(2)假定伪指令
(3)置汇编地址计数器伪指令
① 汇编地址计数器 $
② 设置汇编地址计数器的值
5. 源程序结束伪指令

3.4 常用的系统功能调用

1. 什么是系统功能调用?
2.DOS系统功能调用的一般过程
3.常用的输入/输出系统功能调用
(1)键盘输入(1号调用)
(2)显示输出(2号调用)
(3)输出字符串（9号调用）
(4)字符串输入（10号调用）
(5)结束调用，4CH

3.5 汇编过程

第一次扫描
第二次扫描

3.6 总结

一、学习目标与要求

1. 正确而熟练地使用地址表达式和数值表达式
2. 熟悉常用的机器指令的指令助记符、功能及使用格式
3. 区别**机器指令语句**和**伪指令语句**

> [机器指令]
> 每一条指令语句在源程序汇编时都要产生可供计算机执行的指令代码（即目标代码），所以这种语句又叫可执行语句。每一条指令语句表示计算机具有的一个基本能力，如数据传送，两数相加或相减，移位等，而这种能力是在目标程序（指令代码的有序集合）运行时完成的，是依赖于汁算机内的中央处理器（CPU）、存储器、I / O接口等硬件设备来实现的。
>
> [伪指令语句]
> 伪指令语句是用于指示汇编程序如何汇编源程序，所以这种语句又叫命令语句。例如源程序中的伪指令语句告诉汇编程序：该源程序如何分段，有哪些逻辑段在程序段中哪些是当前段，它们分别由哪个段寄存器指向；定义了哪些数据，存储单元是如何分配的等等。伪指令语句除定义的具体数据要生成目标代码外，其他均没有对应的目标代码。伪指令语句的这些命令功能是由汇编程序在汇编源程序时，通过执行一段程序来完成的，而不是在运行目标程序时实现的。

4. 常用的伪指令功能、使用方法
5. 熟练掌握常用的DOS系统功能调用(1,2,9,10号调用)

3.1 宏汇编语言的基本语法

3.1.1 常量与数值表达式

1.常量

汇编时已有确定的数值的量。

符号常量的定义：

- 等价伪指令 EQU
- 等号伪指令 =

注意：

1. 符号常量**不分配存贮单元**，只建立等价代换关系, 可出现在任何段。
2. 用**EQU语句定义**的符号常量在该程序中**不能再重新赋值**，而用**“=”定义**的符号常量**可多次重新赋值**，使用时，以最后一次定义的值为准。

符号常量特点：
① 在**汇编期间**被代换成相应等价的数据；
② 提高程序的可读性；
③ 便于随时修改程序中的参数。

2.数值表达式

算术运算

+、–、*、/、MOD(模除，取余数)、
SHR(右移)、SHL(左移)。
移位的特别说明：表示将二进制常量右移或左移运算符右边所规定的次数(正整数)，所空出的位数均补0

逻辑运算

逻辑乘：AND（与）
逻辑加：OR（或）
按位加：XOR（异或）
逻辑非：NOT（非）

关系运算

相等：EQ
不等：NE
小于：LT
大于：GT
小于等于：LE
大于等于：GE

> Little Great Equal，这是对有符号数的，对无符号数的是Above, Blow

3.1.2 变量、标号与地址表达式

1. 变量

一个数据存贮单元的名字, 是这个存储单元的地址的符号表示。**可代表一批存储单元的首址。**

变量的属性

- 段属性：定义变量所在段的段首址，当访问该变量时该段首址应在某一段寄存器中，即为CPU当前可访问段;
- 偏移地址：该变量所占存储单元到所在段的段首址的字节距离;
- 类型：类型是指存取该变量中的数据所需要的字节数, 变量的类型由定义该变量时所使用的伪指令确定;

变量的定义

DB、DW、DD、DQ和DT来定义

> BYTE (字节) DB
> WORD (字) DW
> DWORD (双字) DD
>
> FWORD (3个字) DF
> QWORD (4个字) DQ
> TBYTE (10个字节) DT

定义语句的格式为： [变量名] 数据定义伪指令 表达式[，...]

这个表达式可以是以下几种形式

1. 数值表达式

```
1 ARR   DW   10,-60,189          ; ARR的类型为字
```

2. ASCII字符串

```
BUF    DB    ‘ABCD12EF……’  ;  BUF的类型为字节,字符串只能为DB类型，只有为DB类型，才能长度超过两个
```

3. 地址表达式(只适用DW和DD两个伪指令)

   为一变量（或标号）名
   在**16位段**中，如果用**DW**，就是直接用**变量的偏移地址初始化**
   　　如果用**DD**，则是取**段首址和偏移地址初始化。**

   在32位段中，如果用DD就是取其偏移地址初始化。

   > 16位段中：
   >
   > > A　DW　B
   > > B为变量，则A的初始值为B的偏移地址
   >
   > > A　DD　B
   > > B为变量，则A的初始值为B的偏移地址，段首址。　偏移在上（低地址），段址在下（高地址），[1]

4. ？ 变量值不确定

   T DW？ ;这种

5. 重复子句: n　DUP(表达式)，表示定义了n个数据存储单元

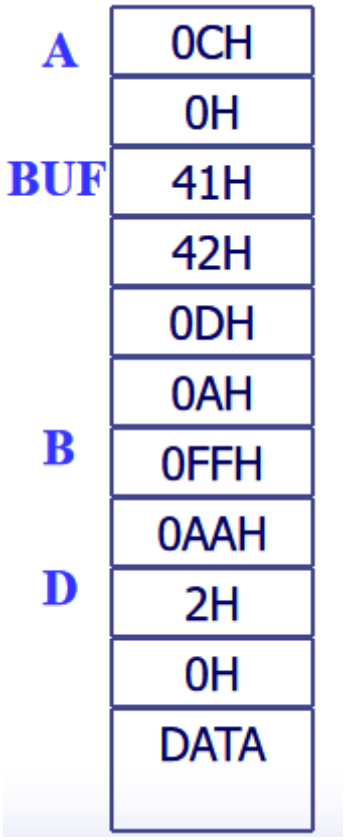
   DB　2 DUP(‘A’, 2 DUP(3), ‘B’)
   　　得到:　　**‘A’, 3, 3, ‘B’, ‘A’, 3, 3, ‘B’**

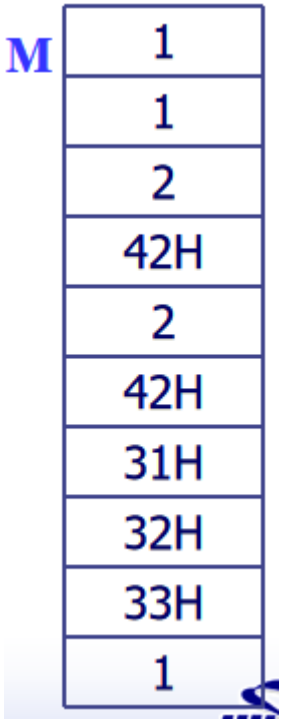
6. 上述(i)～(v)组成的系列，各表达式之间用逗号隔开。

类似`BUF　DB　‘ABCD12EF……’`这种语句

建立了一个以变量为首址的数据存储区或以变量为名的数组

```
例：数据段定义如下：
      DATA   SEGMENT   USE16
      A      DW    M   ;M是 0CH，也就是刚好偏移是12
      BUF    DB    'AB', 0DH, 0AH
      CON    EQU   500H ;不占空间，在段外存
      B      DW    0FFAAH
      MARK = 100H
      D      DD    BUF ;放段址+偏移，段址在下，为DATA，偏移在上，为BUF的偏移，2H
                       ;这个就是地址表达式
      M      DB   2 DUP(1), 2 DUP(2, 'B')
             DB   '123', 1    ;还可以这么连续定义的啊
      DATA   ENDS
画出其储存形式。
```





> 伪指令EQU及“=”不分配存贮单元；
>
> ORG 10H表示把 $+16。

2. 标号

- 标号：是**机器指令语句存放地址的符号表示**，也可以是子程序名，即子程序入口地址的符号表示；在代码段中定义和引用。
- 标号的属性：
  1. 标号的段属性：标号的段属性是指定义该标号所在段的段首址。
  2. 标号的偏移地址：标号的偏移地址是指它所在段的段首址到该标号所代表**存储单元**的字节距离。

3. 标号的类型: 分**NEAR**（近）和**FAR**（远）两类型，**近标号在定义该标号的段内使用**，远标号无此限制，一般也只用的到近标号。[2]

3. 地址表达式

①地址表达式的定义

- 数值表达式的运算结果是一数值常量，只有大小而无属性
- 地址表示式的值一般都是段内偏移地址，有段，偏移地址，类型三个属性。
- 地址表达式是由**变量、标号、常量、寄存器的内容**及一些**运算符**所组成的有意义的式子。
- 如果地址表达式内出现变量或者标号，就是取它们的EA参加运算，不可理解为取其存储单元中的内容。
  例如下面的BUF[BX]，显然是取BUF的EA+BX的内容作为整体的EA，再加上BUF所在段的首址作为地址的。

②接触过的地址表达式

存储器方式（四种）： 直接寻址方式、寄存器间接寻址方式、变址方式、基址加变址方式

```
MOV  AX,BUF[BX+SI]
MOV  AL,BUF+2
MOV  AL,BUF[BX]
MOV  WORD PTR DS:[1000H], 3000H
前三个源操作数为地址表达式
后一个目的操作数为地址表达式
```

这里的`MOV AL,BUF+2`的BUF显然是取EA

```
这MOV  AL,BUF+2等价于
MOV SI,OFFSET BUF
MOV AL,2[SI]
```

③地址表达式的属性

地址表达式的结果是一偏移地址，因此具备**段属性、偏移地址和类型。**

④地址表达式与数值表达式区别

- 地址表达式的结果：是一偏移地址，它具有段属性、偏移地址和类型，(一个表达式中一般只出现一个变量或标号)
- 数值表达式的结果：**只有大小，无属性。**
- 在特殊情况下(没有用到寄存器、不作为地址访问)，地址表达式的值也可能仅表示一个数值(没有属性)。

⑤特殊运算符

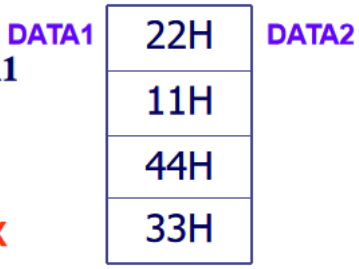
(I)类型运算符PTR

- 格式: 类型 PTR 地址表达式
- 类型可以是BYTE、WORD、DWORD、FWORD、**NEAR、FAR**
- 功能: 用来指明紧跟其后的地址表达式的类型属性，但保持它原来的段属性和偏移地址属性不变，或者使它们临时兼有与原定义所不同的类型属性。

- 作用1：使语句中类型模糊的操作数类型变得明确
  ADD BYTE PTR [SI]，5
  ADD WORD PTR [SI]，5
- 作用2：临时改变某一操作数地址的类型，使得类型不一致的两地址变为一致。

> DATA1 DW 1122H，3344H
> MOV AL，BYTE PTR DATA1 ；将变量DATA1临时改为字节类型
>
> 问题1：将最后一条语句改为：
> MOV EAX， DWORD PTR DATA1
> 执行该语句后，(EAX)=?
> 答：(EAX)=33441122H
>
> 问题2：上述最后一条指令中，改变了DATA1的类型是否从此DATA1变为BYTE类型?
> 不是，将变量DATA1临时改为字节类型
>
> 问题3：是否可以用该运算符改变寄存器的类型?
> MOV EAX， DWORD PTR SI
> 不行。不带方括号的寄存器符号不是地址表达式，**不能用PTR改变寄存器的类型。**

- 作用3：PTR运算符还可以与EQU或等号“=”等伪指令连用，用来**将同一存储区地址用不同类型的变量或标号来表示。**

> ```
> DATA1 DW 1122H，3344H
> DATA2 EQU BYTE PTR DATA1
> ⋮
> MOV AL，DATA2  ；22→AL
> MOV BX，DATA1  ；1122→BX
> ```
>
> 
>
>
> 用PTR算符建立了一个与变量DATA1有相同段首址和偏移地址的变量DATA2，**但它的类型为BYTE**

使用PTR注意事项：

a. PTR临时赋予地址表达式的新类型**只能在本语句中有效。 临时的**
b. 不带方括号的寄存器符号不是地址表达式，**不能用PTR改变寄存器的类型**

> MOV EAX， DWORD PTR SI 就不行

(II)属性分离算符

取段址算符SEG

格式：SEG <变量或标号>
功能：分离出其后变量或标号的段首址。

取偏移算符OFFSE

格式：OFFSET <变量>
功能：分离出其后变量或标号的偏移地址。

取类型运算符TYPE

```
A DW 50,100
MOV CX,TYPE A;然后得到2->(CX)
```

(III)定义类型运算符THIS[3]

与PTR有类似的功能，了解就行。

```
B EQU THIS WORD ;使紧跟它的下个双字变量A，重定义为字类型，命名为B
A DD 44332211H

BB EQU THIS BYTE;使紧跟它的下个变量（没名字），重定义为字节类型，命名为BB，这操作完全没意义啊。
   DW 0FFFFH

C EQU BYTE PTR A ;这个是通过EQU给A起个类型为字节的别名，C，段属性和偏移都不变。
```

- THIS 不能像 PTR那样临时改变地址表达式的类型
- THIS运算符通常用于定义远标号。在模块化设计中经常用到

```
BEGIN EQU THIS FAR;定义远标号
      MOV CH,BL;BEGIN实际上是该语句的标号。
NEXT: MOV AL,[SI]
```

⑥使用地址表达式的注意事项

(1)指令中的地址表达式不允许出现不带方括号的寄存器符号；
例:　MOV AX，SI+4　——错误语句，SI又不是变量。MOV AX,BUF+4还可以。
　　MOV AX，[SI+4]　——正确语句
(2)在定义变量时，其后表达式不能带寄存器符号和方括号；
例:　A DW SI+4，[SI+4] ——错误
(3)数值表达式中如果有变量和标号，**均是取其EA参加运算。**

另外还有 Length，size， high ，low之类的指令在P56，一般也用不上，所以不讲了。

3.2 常用的机器指令语句

3.2.1 80X86指令集及其特点

80X86指令集

8086　100条基本指令
80386　170条指令
Pentium　300多条

特点

- 原8086的16位操作指令都可扩展支持32位操作数；
- 原有16位存储器寻址的指令都可以使用32位的寻址方式；
- 在实方式和虚拟8086方式中段的大小只能为64KB（16位段），只有在保护方式下才使用32位段。

注意事项[4]

⑴大多数双操作数的指令，具有相同的语句格式和操作规定

① 目的操作数与源操作数应有**相同的类型。**
② 目的操作数不能是立即操作数。
③ 操作结束后，运算结果送入目的地址中，源操作数并不改变。
④ 源操作数和目的操作数**不能同时为存储器操作数。**

(2)某些单操作数指令也有相同的语句格式和操作规定

① 操作对象为目的地址中的操作数，操作结束 后，将结果送入目的地址。
② 操作数不能是立即操作数。

3.2.2 数据传送指令

1.一般数据传送指令

(1) 传送指令

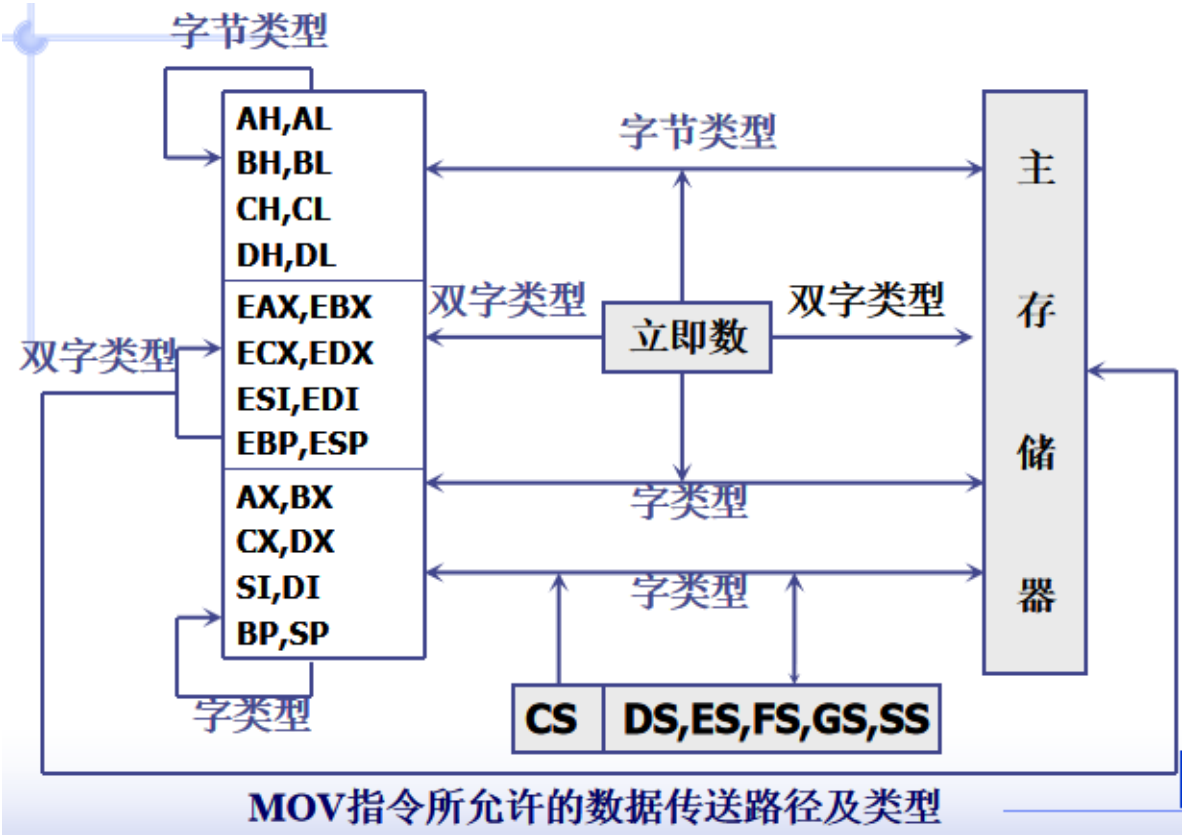
A. 一般传送指令 MOV

- 格式：MOV OPD, OPS
- 功能：(OPS)→OPD (字或字节)

注意：

1. 不能实现存贮单元之间的直接数据传送，因为OPS、OPD**不能同时采用存贮器寻址方式。**

> 例：将字变量BUF0中的内容传送至字变量BUF1中，只能用以下方式：
> MOV AX , BUF0
> MOV BUF1 , AX



2. 不能向CS送数据，从上面这个图就可以看出来，CS是自动设定的，只能传出，不能传入。
3. IP不能在任何语句中出现。
   例：“MOV CS , AX”、“MOV AX，IP”均为错误语句
   IP/EIP不能直接操作
4. **立即数不能直接传递至数据段或者附加数据段寄存器中**，要通过寄存器。

B. 有符号数传送指令

● 格式： MOVSX OPD，OPS（**mov**e with **s**ign-e**x**tend)
● 功能： 将源操作数的**符号向前扩展**成与目的操作数相同的数据类型再送入目的地址。
MOVSX ECX， BL

OPD必须是16位/32位的寄存器，不然没有扩展的必要。

C. 无符号数传送指令

● 格式：MOVZX OPD，OPS （move with **zero**-extend)
● 功能：将源操作数的**高位全部补0**，扩展成与目的操作数相同的数据类型再送入目的地址中

(2) 数据交换指令

A. 一般数据交换指令

●格式：XCHG OPD，OPS （exchange)
●功能：(OPD) ⇔ (OPS)，可作八位或十六位交换。

例：XCHG AX，DI
执行前：(AX)=0001H (DI)=0FFFFH
执行后：(DI)=0001H (AX)=0FFFFH

源操作数不能是立即操作数[5]

~~B.字节交换指令~~

BSWAP OPD

将32位通用寄存器第一个字节与第4个交换，第二个与第三个交换。

基本不考。

~~**C.字节加指令**~~

XADD OPD,OPS

先交换，再相加之后送回OPD，更加不考

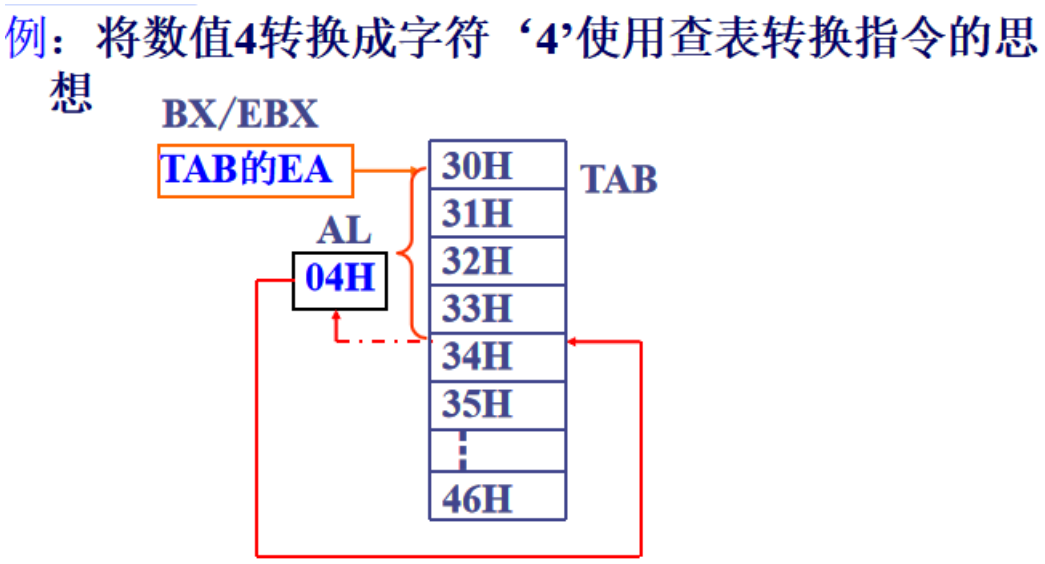
(3) 查表转换指令XLAT

格式：**XLAT** OPS　　或者
　　XLAT　　不带操作数

这玩意儿也不怎么用

功能：([BX+AL])→ AL或([EBX + AL])→AL
　　将(BX)或(EBX)为首址，**(AL)为位移量**的字节存贮单元中的数据→AL

> 那那个XLAT OPS的OPS有啥用？书上也没讲。不过我猜测是([OPS + AL])→AL，不用设定EBX了？



XLAT指令**简化了变址寻址和基址加变址寻址方式的使用**

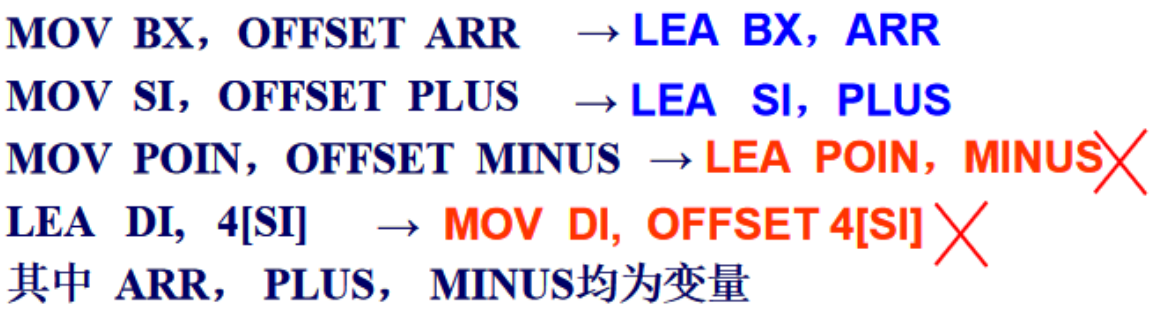
> 注意下列语句
>
> ```
> MOV EBX，OFFSET ASCII
> MOV AL，[DI]；//DI是0，1，2，3，4，5的ASCII码
> XLAT ASCI ；//即([EBX+AL])=(ASCII+AL)=数字相应的ASCII码
> ```

2.地址传送指令

(1) 传送偏移地址指令

●格式：LEA OPD，OPS (load effective address)
●功能：**按OPS的寻址方式计算EA**，将EA送入指定的**通用寄存器**（所以OPD要是寄存器）
●注意：

1. OPD一定要是16位/32位的通用寄存器
2. OPS一定是一个存储器地址，可是寄存器间接寻址、基址加变址、变址寻址、直接寻址。
3. 如果偏移地址为32位而OPD为16位寄存器，**取低16位**→OPD；
4. 如果偏移地址为16位而OPD为32位寄存器，**高16位补0**后→OPD



> POIN 不是寄存器，所以是错的。
>
> LEA DI , 4[SI] 的偏移不能OFFSET或取。**OFFSTE只能用在变量上**，前面也有提到

(2) 传送偏移地址及数据段首址指令

LDS,LES,LFS,LGS,LSS

> 比如 LDS OPD , OPS
> 功能 (OPS)->OPD , (OPS+2/4)->DS

当程序中使用多个数据段时比较方便，可惜快速转换段址。

这里就不写了，不怎么用，见书P61

3.2.3 算术运算指令

1. 加运算指令 ADD、INC

● 语句格式：ADD OPD，OPS
● 功 能： (OPD) + (OPS)→OPD
该指令对标志寄存器的**标志位**有影响。

2. 减运算指令 SUB、DEC、NEG、CMP

● 格式：SUB OPD，OPS (subtract)
● 功能：(OPD) — (OPS) →OPD

DEC是自减1

NEG是单操作数，将OPD的每一位取反（包括符号位）后加1→OPD
算得上是求相反值了，因为符号位变了

> 注意和NOT区别，NOT是直接全部取反，不会加1的。

3. 乘运算指令 IMUL、MUL

(1)有符号乘指令

①双操作数的有符号乘指令

- IMUL OPD, OPS
- 功能：（OPD）*（OPS）→OPD

OPD可为16/32的**寄存器**，OPS为同类型的**寄存器、存储器操作数或立即数。**

②三个操作数的有符号乘指令

- 语句格式：IMUL OPD，OPS，n
- 功能：（OPS）* n →OPD
  其中，OPD可为16/32的**寄存器**，OPS可为同类型的**寄存器、存储器操作数, n为立即数**

③ 单操作数的有符号乘指令

- 语句格式： IMUL OPS
- 功 能：看OPS的类型来选。

字节乘法：(AL)*(OPS)→AX（即AH，AL)
字乘法： (AX)*(OPS)→DX，AX
双字乘法：(EAX)*(OPS)→EDX，EAX

说明：

1. 只需指定**源操作数**，另一个操作数是隐含的，被乘数和乘积都在规定的寄存器中。源操作数只能是存储器操作数或寄存器操作数而**不能是立即数**，乘法类型由OPS的类型决定。
2. 如果乘积的**高位**(字节相乘指AH，字相乘指DX，双字相乘指EDX)**不是低位的符号扩展**，**即在AH**(或DX/EDX)**中包含有乘积的有效位**，**则CF = 1、OF = 1**（有溢出和进位）；否则，CF = 0，OF = 0。

> 进位判断方法：加减时**最高位**产生**进位或借位**，代表结果超出无符号数的范围，单字节就是0~255；
>
> 溢出判断方法：加减时最高位和次高位中有且仅有一个产生进位或借位，代表**结果超出有符号数的范围**，单字节就是-128~127；

(2)无符号乘指令

- 语句格式： MUL OPS
- 功 能：

字节乘法：(AL)*(OPS)→AX
字乘法： (AX)*(OPS)→DX，AX

> 猜猜为啥不用EAX，因为这是8086时候就有的，当时没有EAX这32位的。
>
> 另外，无符号乘指令只有单操作数格式的。

双字乘法：(EAX)*(OPS)→EDX、EAX

1. 与有符号乘法指令之间的区别：参与运算的操作数和运算后的结果均为无符号数。
2. 如果乘积的**高位不为0**（因为无符号肯定是正数），即在AH(或DX/EDX)中包含有乘积的有效位，则CF = 1、OF = 1；否则，CF = 0，OF = 0。

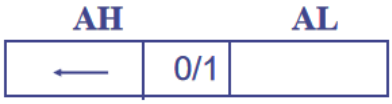
4. 符号扩展指令 CBW、CWD、CWDE、CDQ

全都是针对 AL,AX的，感觉像专门为后面的除法设计的

(1) 将字节转换成字指令

语句格式：CBW（convert byte to word)

功能：将**AL**中的符号扩展至**AH**中, 操作数隐含且固定



(2) 将字转换成双字指令

语句格式：CWD（convert word to double word)

功 能： 将AX中的符号扩展至DX中，由 DX，AX组成双字

> 和乘法那儿一样

注意：上述指令的操作数是隐含且固定的。

(3) CWDE

将AX中的有符号数扩展为32位数→EAX

(4) CDQ

将EAX中的有符号数扩展为64位数→EDX、EAX

5. 除运算指令 IDIV、DIV

(1) 无符号除指令

● 语句格式： DIV OPS （unsigned divide)
● 功 能:
字节除法：(AX)/(OPS)→AL(商) 、AH(余数)
字除法： (DX、AX)/(OPS)→AX(商)、DX(余数)
双字除法：(EDX、EAX)/(OPS)→EAX(商)、EDX(余数)

说明

1. 除法类型由OPS的类型决定。**OPS不能是立即操作数**，且指令执行后，(OPS)不变。
2. 如果**除数为0**或**运算结果溢出**，则会产生溢出中断，立即中止程序的运行。但系统未定义除法指令影响条件标志位。
3. 除法指令的**被除数是隐含的**，隐含在AX/DX/EDX中。

(2) 有符号除指令

●语句格式： IDIV OPS （signed integer divide)
●功 能:
字节除法:(AX)/(OPS)→AL(商)、AH(余数)
字除法:（DX，AX)/(OPS)→AX(商)，DX(余数)
双字除法:(EDX、EAX)/(OPS)→EAX(商)、EDX(余数)

说明:
1，2两点与DIV语句相同

3、余数与被除数同号

(3) 有符号数，无符号数除法的区别[6]

例：写出计算4001H÷4的程序段。

```
MOV AX，4001H
CWD                    ——符号位扩展到DX
MOV CX，4
IDIV CX                ——（DX，AX）/（CX）

结果：（AX）=1000H ，（DX）=1
```

如果将被除数改为 -4001H，程序段为：

```
MOV  AX，－4001H
CWD
MOV  CX，4
IDIV CX

运算的结果为：（AX）＝0F000H，就是商（AX）=-1000H，  （DX）＝0FFFFH，你看余数（DX）=-1，与被除数同号的。
```

假如被除数为-4001H，除数为-4，相除后的余数也为0FFFFH，就是-1，不会是3。余数一定与被除数同号。商就是1000H

3.2.4 位操作指令

1.逻辑运算指令

(0) 求反指令 NOT

格式：NOT OPD

逐位取反再送到OPD

(1) 逻辑乘指令AND

格式：AND OPD，OPS

功能：（OPD）∧（OPS）—> OPD

就是与运算，1^1=1，其他全是0
该指令主要用来将目的操作数中清除与源操作数置0的对应位

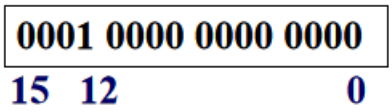
(2) 测试指令TEST

格式：TEST OPD，OPS

功能：(OPD)∧(OPS) ——根据结果设置标志位

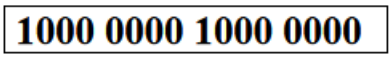
做**逻辑乘运算**，根据测试结果**置OF、CF、SF、ZF位**

例：要测试AX中第12位是否为0，为0转L，则要使用如下指令：




```
TEST AX, 1000H
JZ   L
```

如果要同时测试第15位和第7位是否同时为0，为0转L




```
TEST AX, 8080H
JZ   L
```

(3) 逻辑加指令OR

格式：OR OPD, OPS

功能：(OPD) ∨ (OPS) —> OPD

就是并操作，在目的操作数中置位与源操作数为1的对应位，其余位不变。

几点总结

①如果要将目的操作数中某些位清0，用AND
②如果要将目的操作数中某些位置1，用OR
③用来测试目的操作数中某一位或某几位是否为0或1，而操作数不变，用TEST
④操作数自身相或、相与结果不变。

(4) 按位加指令XOR——异或

格式：XOR OPD, OPS (e**x**clusive **or**)

功能：(OPD) ∀ (OPS) → OPD

运算法则：0 ∀ 1=1，1 ∀ 0=1，1 ∀ 1=0，0 ∀ 0=0

不同取1，相同取0

```
XOR AX, 0AAAAH
执行前：(AX)=0FFFFH
1111  1111  1111  1111
1010  1010  1010  1010
0101  0101  0101  0101
结果：  (AX)=5555H
```

(5) 位操作指令的特点

1. 自身相或相与结果不变
2. 自身按位加结果为0，“XOR AX, AX”之后(AX) = 0；
3. 可用于测试。例如，测试SI中的第3、7、11、15为是否同时为0，为0 转ERR：

```
TEST  SI，8888H
JE    ERR
显然，有一位不为0，结果都不是0，ZF都不为1，都不会跳转
```

1011 0111 1000 0111
1000 1000 10001000

1000 0000 1000 0000
15 11 7 3 0

2.移位指令

有统一的语句格式

操作符 OPD，n
算术逻辑移位操作符： SXL， SXR (X=H, A)
循环移位操作符：RXL，RXR (X=O, C)

X代表	含义
H	逻辑移位(logical)
A	算术移位 (arithmetic)
O	循环移位(Rotate)
C	带进位的循环移位 carray

移位指令的特点

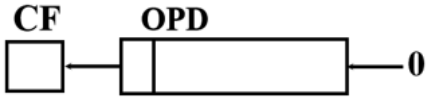
不管哪种方式的移位都会将所**移出的最后一位放入CF位**

(1) 算术、逻辑移位指令

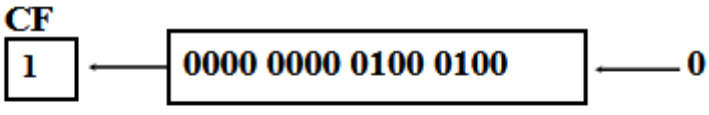
①算术左移和逻辑左移指令SAL/SHL（这两者是一样的）

格式：SHL OPD，n 或 SAL OPD，n （**sh**ift **l**ogical left/**s**hift **a**rithmetic **l**eft)

功能：
将OPD的内容向左移动n指定的位数，**低位补入相应个数的0。**CF的内容为最后移入位的值，移动方式为：



例：SAL AX，1
执行前：(AX) = 0044H，CF = 1



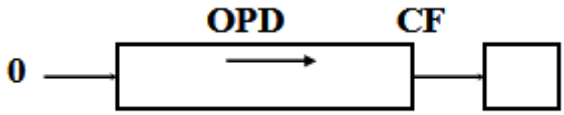
结果：CF = 0，(AX) = 0088H

注意：

当一个数**乘2的N次方**时，可以用**算术左移或逻辑左移N位**的方法实现，**比用乘法指令效率高。**但若**发生溢出则可能得不到正确的结果。**

②逻辑右移指令SHR

格式：SHR OPD，n （**sh**ift logical **r**ight)
功能：将(OPD)向右移动n规定的次数，**最高位补入相应个数的0**，CF的内容为最后移入位的值。

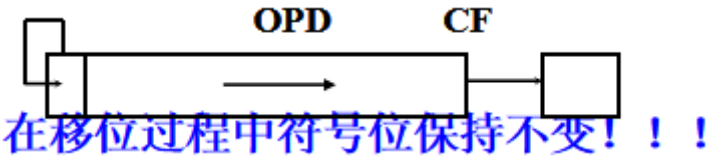


功能：

1. SHR指令右移n位，实现**无符号数**除2n运算
2. 将一个字(或字节/双字)中的某一位(或几位)移动到指定的位置，从而达到分离出这些位的目的.

③算术右移指令SAR

格式：SAR OPD, n （shift arithmetic right)
功能：将OPD中的操作数移动n所指定的次数。且最高位（符号位）保持不变。就是说移动的时候最高位补符号位，若符号位是1就补1，符号位是0就补0CF的内容为最后移入位的值。



功能：

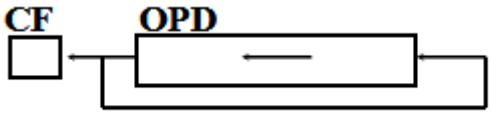
利用算术右移指令可以方便地实现对**有符号数**除2的N次方的运算。 向右移位N次即可，注意不保留余数哦。

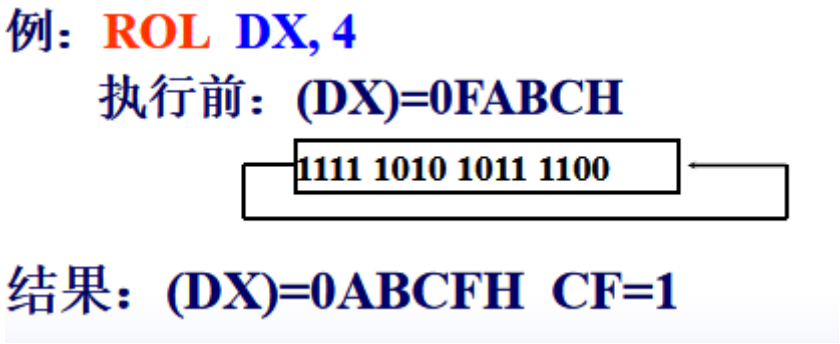
(2) 循环移位指令

①循环左移指令

格式：ROL OPD, n (rotate left)
功能：将目的操作数的最高位与最低位连接起来，**组成一个环**，将环中的所有位一起向左移动n所规定的次数。**CF的内容为最后移入位的值。**

> 相当与CF每次都得到一个移入数的副本



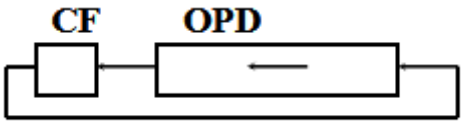


②循环右移指令

格式：ROR OPD，n (rotate right)
功能：该指令的移动方式完全同ROL，只是向右移动。

③带进位位的循环左移指令

格式：RCL OPD，n (rotate left through carry)
功能：将目的操作数**连同CF标志一起**向左循环移动所规定的次数。相当于把CF和OPD当一个整体了。



④带进位的循环右移指令

格式：RCR OPD，n

移动方式完全同RCL，只是向右移动。

3.3 伪指令语句

汇编语言用助记符代替操作码，用符号代替操作数，这就是汇编语言中的机器指令语句。但仅有这些，汇编程序分辨不出段的定义，区分不了数据和指令，以及数据类型。所以需要一些固定格式的约定符号，来告诉汇编程序如何工作，这就是汇编控制命令。

1. 伪指令的基本概念

定义：汇编源程序中控制汇编程序应如何工作的命令是**伪指令**，或称**汇编控制命令。**

> 实际告诉机器怎么做的是机器指令。

工作原理:

- 只为汇编程序所识别
- 每一条汇编控制命令都对应着一段处理程序，
- 汇编程序每遇到汇编控制命令，即转入对应的处理程序执行，执行完该处理程序，也就实现了这条汇编控制命令的功能。

结果:

- 可以申请分配一部分存贮空间用作数据区和堆栈
- **没有对应的机器代码**
- 在**将源程序翻译成目标程序后，伪指令就不存在了。**可以看到源程序反汇编之后就看不到那些伪指令了。

指令分类:

- 处理器选择伪指令
- 数据定义伪指令
- 符号定义伪指令
- 段定义伪指令
- 过程定义伪指令——就是子程序
- 程序模块的定义与通讯伪指令
- 宏定义伪指令
- 条件汇编伪指令——IF,ELSE，符合条件的才参与汇编
- 格式控制、列表控制及其它功能伪指令

问题：伪指令与机器指令的区别?

1. **功能不同**，机器指令控制CPU的工作，**伪指令控制汇编程序工作。**
2. **格式不同**，机器指令标号后面带冒号，而伪指令的名字后面没有。
3. 被执行时CPU所处状态不同，用户程序在运行时执行机器指令，**汇编程序运行时，执行伪指令。**
4. 机器指令是用**硬件线路**来实现其功能的，它**有目的代码。**
   而伪指令是用来控制汇编程序操作的，是用程序来实现其功能的，它在**汇编期间被执行，在目的代码中已不存在了。**

2. 处理器选择伪指令

缺省情况下是.8086

- 在源程序中，告诉汇编程序选择何种CPU所支持的指令系统
- 一般放在程序的开始处，表示后面的段使用该处理器所支持的指令系统
- 不同版本的宏汇编程序对指令系统的支持不同

一般都是用.386 接受.80386的指令

3. 符号定义伪指令

=，EQU这两个，详细使用见P78

还有个LABLE指令，和EQU THIS功能差不多。和 EQU PTR BYTE(或者其它的类型) 也差不多，前面有讲

```
DWBUF LABEL WORD;和BUF一样的段首址和偏移，就是类型不同
BUF DB 100 DUP(?)

DWBUF EQU THIS WORD;和BUF一样的段首址和偏移，就是类型不同
BUF DB 100 DUP(?)

BUF DB 100 DUP(?)
DWBUF EQU WORD PTR BUF;这个顺序可以再下面或者上面，上面两个方式不行
```

4. 段定义伪指令

(1)段定义伪指令

格式:

段名 SEGMENT [使用类型] [定位方式] [组合方式] ['类别']

⋮

段名 ENDS

> 这边定位方式和组合方式在5.3节有详细讲。

功能：
定义一个以SEGMENT伪指令开始、ENDS伪指令结束的、给定段名的段。
其中，段名为该段的名字，用来指出汇编程序为该段分配存贮区的起始位置。

- 一个程序模块可以由若干段组成，段名可以各不相同，也可以重复，汇编程序将一个程序中的**同名段处理成一个段**；
- 段的定义还可以嵌套，但不能交叉；
- “使用类型”只有对使用.386及以上处理器选择伪指令的段才起作用。 —— USE16　USE32，默认32位段
- 在实方式和虚拟8086方式中段的大小只能为64KB。

(2)假定伪指令

格式：ASSUME 段寄存器：段名[，段寄存器：段名] ...
功能：用来设定段寄存器与段之间的对应关系，即告诉汇编程序，该段中的变量或标号用哪个段寄存器作段首址指示器。哪些段是当前CPU可访问段。

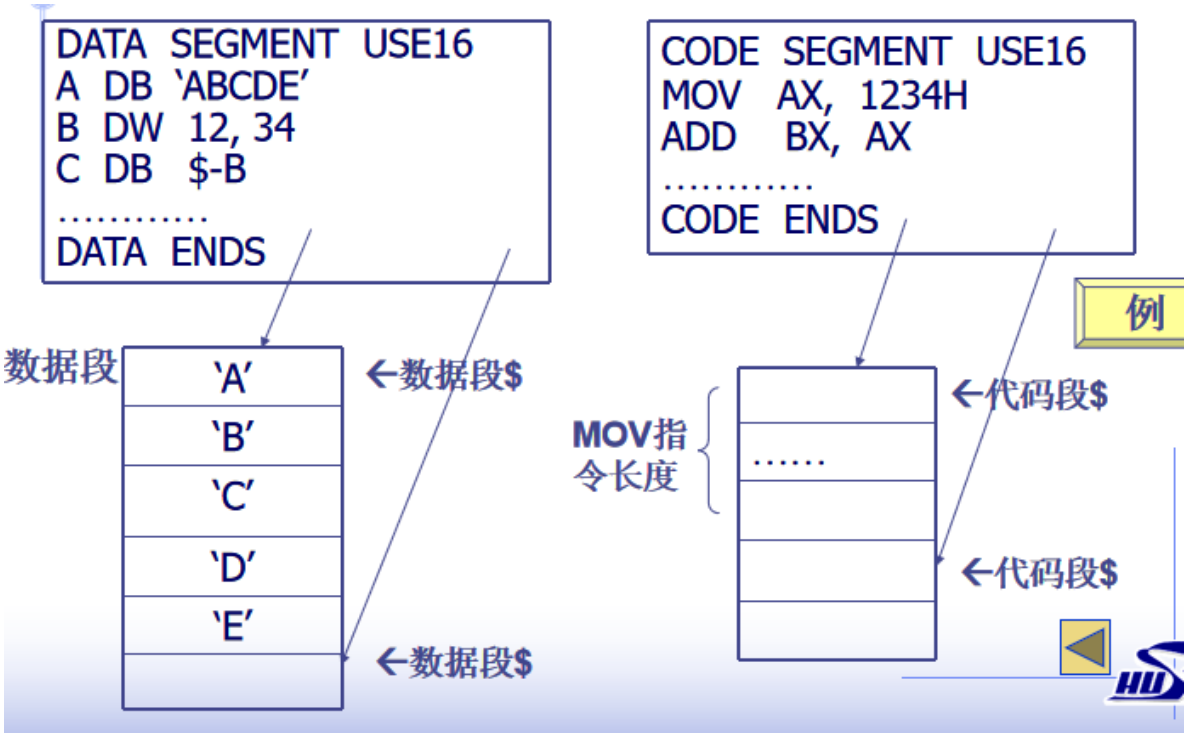
注意：

- 在代码段的开始，就要用ASSUME语句建立CS、SS与代码段、堆栈段的对应关系，否则就会出错
- ASSUME语句不可能将段首址置入对应的段寄存器中，这一工作要到目标程序最后投入运行时CS和SS的内容将由**系统自动设置**，不用用户程序处理
- 对于数据段和附加数据段，**若用ASSUME语句建立它们与DS、ES的关系**，则其后语句如需访问这些段内的变量，**均可直接使用段内寻址，而不必带跨段前缀;** ——ASSUME语句之后用户设置DS、ES段
- 对于数据段和附加数据段，若**不用ASSUME语句建立它们与DS、ES的对应关系**，则其后语句如需访问这些段内的变量，**都必须带跨段前缀**才可使用段内寻址

> 如果定义了DATA1 ,DATA2两个段
> 一开始ASSUME DS:DATA1
> 那么之后如果要把数据段换到DATA2的时候
>
> 需要：
> ASSUME DS:DATA2
> MOV AX,DATA2
> MOV DS,AX
>
> 要先把数据段换成DATA2，再把正确的段首值送到DS，**两者缺一不可**

(3)置汇编地址计数器伪指令

① 汇编地址计数器 $



$的内容标示了汇编程序当前的工作位置

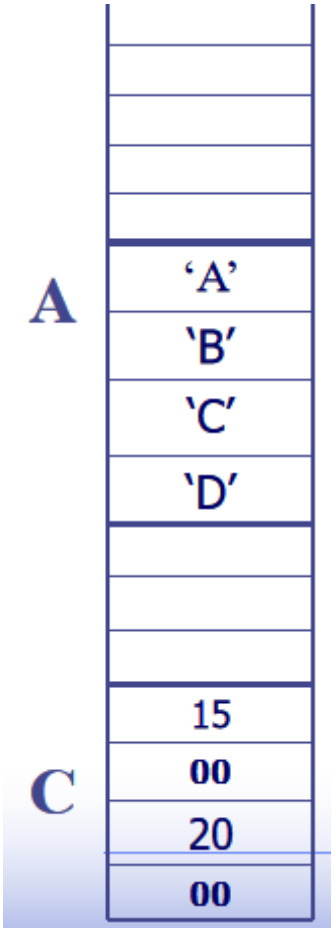
```
DATA SEGMENT USE16
BUF DB '12345ABCD'
COUNT EQU $-BUF ;COUNT的值就是BUF数据区所占的字符数，9
DATA ENDS
```

② 设置汇编地址计数器的值

> 很少用

●格式：ORG <数值表达式>　表达式的值为0 ~ 65535(16位段)0~ 4G（32位段)
●功能：将$的值置成数值表达式的值
例：

```
DATA    SEGMENT  USE16
ORG       5          ;空了五个字节
A   DB    'ABCD'     ;A的EA为5
B   EQU   $-A        ;设B的值为4
ORG       $+3        ;空三个字节
C   DW    15，20，…… ;C的EA为12
DATA      ENDS
```



5. 源程序结束伪指令

格式：END［表达式］
功能：该语句为汇编源程序的最后一个语句，用标志源程序的结束。即**告诉汇编程序翻译到此为止。**

表达式为可选项，其值必须是一个存储器地址（标号是一个直接寻址地址？），该地址为程序的启动地址。

3.4 常用的系统功能调用

1. 什么是系统功能调用?

DOS的系统功能调用：系统将**对计算机外部设备**(键盘输入、屏幕输出)的**控制过程**编写成程序，事先存放在系统盘上，用户在需要时只要按规定的格式设置好参数，直接调用。

2.DOS系统功能调用的一般过程

(1) 置入口参数
(2) 子程序编号(功能号)→AH
(3) INT 21H
(4) 成功调用返回，CF=0； 否则CF=1且错误码 → AX

3.常用的输入/输出系统功能调用

(1)键盘输入(1号调用)

- 格式：MOV AH，1
  INT 21H
- 功能：等待从键盘输入**一个字符的ASCII码**→AL，同时将此字符在屏幕上显示出来。

(2)显示输出(2号调用)

格式：MOV DL，待显示字符的ASCII码
MOV AH，2
INT 21H

功能：将DL中的字符在屏幕上显示出来。
例如：

```
MOV  DL，0AH ;输出换行符
MOV  AH，2
INT  21H
```

(3)输出字符串（9号调用）

格式：LEA DX，**字符串首址偏移地址**
MOV AH，9
INT 21H

功能：将当前数据段中指定的**(DS:DX)字符串**输出(该字符串必须以'$'为结束符，且字符'$'不输出)

```
例：DATA  SEGMENT USE16
      BUF   DB ‘A字符的ASCII码是41H’，0AH，0DH
            DB ‘B字符的ASCII码是42H’，0AH，0DH，‘$’
      DATA  ENDS
              ┆
      LEA   DX，  BUF  （或者MOV  DX，OFFSET BUF）
      MOV   AH，9
      INT   21H
              ┆

则输出：A字符的ASCII码是41H，光标回车换行，再显示
      B字符的ASCII码是42H，光标回车换行。
```

(4)字符串输入（10号调用）

格式：

```
LEA DX,缓冲区首址
MOV AH,10
INT 21H
```

```
DATA  SEGMENT  USE16
BUF   DB  50          ；缓冲区大小
      DB   ?          ；记录填入实际的字符个数
      DB  50 DUP(0) ；初始化为0
DATA  ENDS
        ┆
      LEA  DX，BUF
      MOV  AH，10
      INT  21H
```

功能：从**键盘**接收一个**以回车为结束**的字符串到当前数据段输入缓冲区中，并在屏幕上回显示，同时将**输入字符实际个数n填入缓冲区第二字节中。**

> 例如缓冲区大小50个，如n>49则多余字符被丢掉。因为会有一个回车

注意：一串字符的最后必须要输入回车作结束，但该结束只表示输入工作的结束，**回车符ASCII码送入到缓冲区内但不计入字符个数（就是说实际字符个数里不包括回车，但缓冲区里它又占了一个位置）**，且光标只回到本行行首。

例如：`BUF DB 10,?,10 DUP(0)`

1. 第一个字节的数值在程序设计时确定，它决定允许输入的字符串最大长度（**含回车符在内**）。

   > 这里最多放9个字符，一个回车符

2. 第二个字节在功能调用完成后由DOS填写，**记录实际输入的字符个数**（不计回车）。
3. 第三个字节开始往后的区域用来存放程序运行过程中实际输入的字符串内容。

例如，程序运行过程中执行到这一次的0AH号功能调用时，你**最多可以输入9个字符，再输入字符DOS系统将不予理睬，等你敲回车。**你敲了回车以后，**第2个字节记录你输入的字符，最多是9。**系统所接受的字符加上回车符会保存在第3个字节开始往后的数据区中。

```
例：阅读下列程序，并指明它所完成的功能
DATA   SEGMENT
BUF   DB   50
      DB   0
      DB   50 DUP（0）
CRLF   DB   0DH，0AH，'$';0DH是回车，0AH是换行
DATA   ENDS
        ┆
START: MOV   AX，DATA
       MOV   DS，AX
       LEA   DX，BUF       ;输入到BUF缓冲区
       MOV   AH，10        ;十号调用，输入字符串
       INT   21H
       LEA   DX，CRLF
       MOV   AH，9         ;9号调用，输出回车换行
       INT   21H
       MOV   BX，BUF+1     ;缓冲区的第二个字节，取实际存了多少个字符，不计算回车，放入BX
       MOV   BYTE   PTR   BUF+2[BX]，'$' ;再输入字符的末尾补$，
       ;显然BUF+(BX)+2即字符末尾，可见书P87，我画的有解析
       LEA   DX，  BUF+2   ;+2是因为前两个字节不放字符串，放缓冲区大小和实际长
       MOV   AH，9         ;准备输出之前的字符串
       INT   21H
```

```
        MOV   AX，4C00H   ;结束本程序
        INT   21H         ;返回DOS操作系统
CODE    ENDS
        END   START
```

(5)结束调用，4CH

```
INT  21H 称为 DOS 中断调用。
```

```
MOV  AH，4CH   ;结束本程序
INT  21H       ;返回DOS操作系统
```

3.5 汇编过程

见书P89，这部分挺有趣的。

MASM 本身提供两个表，称固定符号表

1. 机器指令表
   指令助记符（mov，add这种）对应的目标机械代码
2. 伪指令表
   记录例如DW,EQU等**伪指令**的相应处理程序的入口地址

第一次扫描

建立**用户自定义符号表**和**宏定义表**

- 用户自定义符号表
  - 段表
    登记各段段名，字节数，定位，组合方式，类别等
  - 一般符号表（变量，标号，符号常量等）
    记录程序定义和引用的全部符号常量，变量，标号的名字及其属性值
- 宏定义表
  记录全部宏定义的名字，内容，性质等

扫描过程

扫到宏定义即填入宏定义表，扫到新段名，就把相关信息填入段表，同时配一个初值为0的**汇编计数器$**。对段内语句进行汇编时，当前$的值也就是该变量（或标号）的偏移。MASM会将它们的名字和属性填入到一般的符号表中。如果扫描遇到同名段，就接着之前的$继续累计长度，并和同名段的目标代码连续存放，组合成一个段。

书上P90还有个图。

第二次扫描

MASM利用**固定符号表**，和**用户自定义符号表**，可以把源程序翻译成目标程序，同时产生相关文件。

3.6 总结

本章主要掌握的内容：

1. 数值表达式和地址表达式
    1. **存贮器寻址方式均属地址表达式形式**(直接、间接、变址、基址加变址)
    2. 在地址表达式不允许出现不带方括号的寄存器符号。
    3. 地址表达式中的标号和变量均是取其偏移地址参加运算。
    4. 算符： PTR SEG OFFSET
2. 机器指令语句
    1. 数据传送指令
        - 一般数据传送指令： MOV、XCHG、**XLAT**、MOVSX、MOVZX、BSWAP、XADD
        - 堆栈操作指令： PUSH、POP、PUSHF、POPF[7]
        - 标志传送命令： SAHF、LAHF
        - 地址传送指令： LEA、LDS、LES
    2. 算术运算指令：
        - 加指令： ADD、INC
        - 减指令： DEC、SUB、CMP
        - 乘除法指令：MUL、IMUL、CBW、CWD、CWDE、CDQ、DIV、IDIV
    3. 位操作指令：
        - 逻辑运算指令：NOT、AND、TEST、OR、 XOR
        - 移位指令： SHL/SAL、SHR、SAR、ROL、ROR、RCL、RCR
3. 伪指令语句
    1. 机器指令语句的区别
    2. 数据定义伪指令： DB、DW
    3. 符号定义伪指令： EQU、=
    4. 段定义伪指令：SEGMENT、ENDS、 ASSUME
    5. 源程序结束伪指令：END
    6. 汇编地址计数器： $ 和ORG <表达式>
4. DOS功能调用

    1号、2号、9号、10号DOS功能调的格式、使用条件。
5. 在IBM-PC机上建立、调试、运行汇编源程序的方法。

---

1. 谁先谁后？书上好像偏移地址先。P49 ↩
2. 在后面子程序的时候会讲 ↩
3. 好像不考，PPT上没有，见书P54 ↩
4. 常识了 ↩
5. 书上写的，P59 ↩
6. 书P67 ↩
7. 这个我怎么记得不在第三章啊。见书P11 ↩