第三章 80X86的指令系统和寻址方式

指令的格式

数据的寻址方式

指令流的寻址方式

§3.1指令的格式

(机器）指令格式图示

操作码：指示计算机所要进行的动作或者运算
操作数：指示指令所涉及的数据或者数据的存储地址

根据操作数字段的个数，指令可以划分为：

三地址指令：含2个处理数据和一个结果的存储地址
多用于大型机指令系统中

二地址指令：两个操作数分别称为源操作数和目的操作数，
运算结果保存到目的操作数地址中。

单地址指令：只有一个运算数。

零地址指令：不需要操作数，一般为控制机命令。

对应于上面的指令格式，在当前的微型计算机系统中，为了缩短指令长度，提高指令编码效率，一般采用后面的三种方式编制指令系统。

计算机只能识别二进制代码指令，为方便编程，人们使用了助记符来书写指令代码，进而产生了助记符编程语言——汇编。

汇编语言是一种符号语言：

☆用助记符来表示操作码；

☆用符号或者符号地址来表示操作数或者操作数地址；

☆与机器指令是一一对应的

计算机内部的数据我们通称为量，描述一个量，我们从如下4个方面来考虑：

一个量

- 名字：为方便编程，以一个合法的字符串（标识符）来表征对该量的使用。
- 值/内容：某个时刻量的大小。根据值在程序执行过程中是否可以修改产生了常量和变量的概念。
- 存储地址：在内存的什么位置保存该量。对于一写深层低端编程人员或者特出程序而言，地址是不可缺少的一种工具。
- 类型： 决定了量的基本操作和性质。同地址结合，产生了空间大小和值域的概念。

对于量的访问，高级语言一般提供的是名字，而汇编中往往直接面对地址的概念来处理。

§3.2数据的寻址方式

寻址：计算并获得所需信息的存储地址
寻址方式：存储地址的计算方法

根据信息的不同，有两大类型信息的寻址

☻数据的寻址：即操作数的寻址，以什么样的方法获取或计算获得操作数或者操作数的地址。

☻指令的寻址：即程序流程的寻址，以什么样的方式产生下一条要执行的指令的地址

数据寻址

数据传送指令MOV

指令格式　MOV　DST，SRC

指令功能：将源操作数SRC的值传递给目的操作数保存

操作数字段可以是一个数据本身，也可以是数据的存储地址，因此，操作数的具体表述是不一样的，进而产生了不同的寻址方式，也就是是计算操作数地址的运算过程是不一样的。

由MOV指令的功能可知，DST必需是一个带有存储位置的单元的概念，若是一个数值本身做DST是不行的。目前我们已知道的可以存储数据的机制有辅存，内存和CPU中的寄存器，在汇编中，我们只关心内存单元和寄存器中的数据。辅存数据的应用作为虚拟存储系统的数据交换过程在操作系统课程中讲授。

数据的七种寻址方式

数据寻址的基本方式

- 直接：在指令中直接出现数据或数据地址
  - 立即数
  - 直接寻址
  - 寄存器方式
- 间接：在指令中出现的是数据地址的地址
  - 寄存器间接寻址方式
- 相对：指令中给定的是起点和相对的偏移量
  - 寄存器的相对寻址方式

在这三种基本方式基础上，变形衍生出了另外的两种方式

- 基址变址寻址方式
- 相对基址变址寻址方式

1、立即寻址方式--操作数在指令中给出

1）操作数的值直接存储在指令的操作数字段中，作为指令的一部分存放在代码段里，这种操作数称为立即数

2）常用于对变量赋值或初始化

例：

```
MOV   AX, 5
MOV  BH, 100B
MOV  CX, 1000H
MOV  DX, 0FFFFH
```

* 只能用于SRC字段
* SRC 和 DST的字长一致　　× MOV AH, 3064H

2、寄存器寻址方式--操作数在指定的寄存器中

指令中指定寄存器号（寄存器地址）作为操作数的保存地址。如前面的例子中的**DST**的寻址方式就是寄存器方式

MOV AX, BX

注意：

1）**SRC**源操作数寻址方式是立即数方式，其值的范围要同**DST**相一致。 **MOV AX, 70000**指令是错误的。

2）立即数用十六进制表征时，若第一个数字是**A～F**，则前面一定要补写一个**0**。如：

MOV AX，10（给**AX**赋值**10**）；如果写成十六进制就成了**MOV AX, AH**；（八位寄存器**AH**值传给**AX**？）类型不一致，显然是错误的，所以必需写成**MOV AX, 0AH**

寄存器寻址方式* — 操作数在指定的寄存器中

MOV AX, BX

MOV AL, BH

MOV AX, 3064H

* 字节寄存器只有 **AH AL BH BL CH CL DH DL**

* **SRC 和 DST 的字长一致** **MOV AH, BX** ×

* **CS 不能用 MOV 指令改变** **MOV CS, AX** ×

前面已经讲过，一个20位的物理地址的计算表示为：
物理地址PA＝段地址×10H＋段内偏移地址。
这里为了同指令系统的寻址方式相结合，我们一般将最终参与计算PA所用的那个段内偏移地址量称为有效地址EA

$$PA = 段地址 \times 10H + EA$$

有效地址EA可由4个部分构成（可选项）：
1）位移量DISP：一个指定的固定地址值
2）基址BASE：起点地址的有效值
3）变址INDEX：相对起点的位移量或者说偏移地址
4）比例因子SCALE：386以后CPU的一个术语，
可以为1，2，4，8(即数据的字节宽度）

$$EA = BASE + (INDEX \times SCALE\ ) + DISP$$

3、直接寻址方式--有效地址EA由指令直接给出

指令的操作数字段是操作数的地址。具体将就是段内的偏移地址或者位移量。即EA只有DISP构成。

例：

MOV AX， 2000H 是立即数寻址，2000H代表数值

如果现在我们要把偏移地址2000H的数据传递给AX，有

MOV AX， [2000H] 这里[2000H]代表了地址2000H

1)书写程序中，为了方便，我们用一个有效的标识符代表地址2000H，如假设为VALUE，则上面的指令可写为

MOV AX，VALUE 等价为 MOV AX，[VALUE]

称VALUE为地址2000H的符号地址。（地址的符号常量表示）

2）直接寻址方式中，有效地址所关联的段地址默认为DS，可采用段跨越前缀的方法改变默认段。

MOV AX，VALUE → PA = (DS) × 10H ＋ 2000H
MOV AX，ES：[VALUE] →PA = (ES)×10H ＋ 2000H。

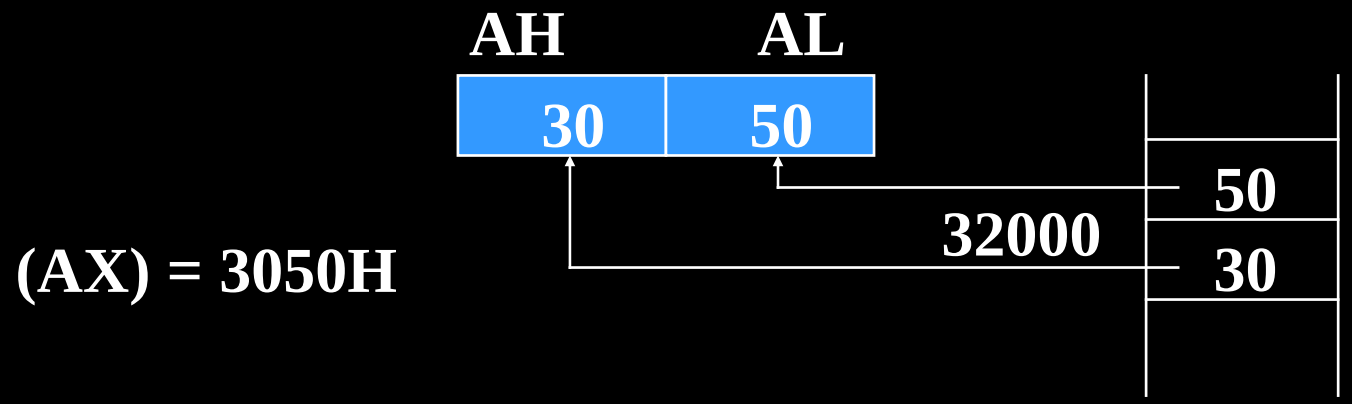
3）符号地址所代表的量一定是在内存中存储的。若用高级语言来描述，我们称为变量VALUE。

4）为了缩短指令平均长度，我们规定：
对于双操作数指令（双目运算），除了源操作数为立即数方式以外，一定要有一个寄存器寻址方式。

5）立即数不能做目的操作数。

MOV AX, [2000H]

EA=2000H, 假设(DS)=3000H, 那么(PA)=32000H



* 隐含的段为数据段 DS
* 可使用段跨越前缀　　MOV AX, ES: [2000H]
* 操作数地址可由变量（符号地址）表示，但要注意

变量的属性

```
VALUE  DB  10

MOV    AH, VALUE
MOV    AX, VALUE  ×
MOV    AX,  WORD PTR  VALUE √
```


描述下列指令的功能：

mov ax,[0]

mov al,[0]

将一个内存单元的内容送入**ax**
内存单元长度为**2**字节（字）
偏移地址为**0**，段地址在**DS**中

将一个内存单元的内容送入**al**
内存单元长度为**1**字节（字节）
偏移地址为**0**，段地址在**DS**中

- 例题1：内存中的情况如图，写出下面指令执行后寄存器ax，bx，cx中的值。

代码情况：

```
mov ax, 1000H
mov ds, ax
mov ax, [0]
mov bx, [2]
mov cx, [1]
```

内存情况示意图

物理地址	内存单元
10000H	23
10001H	11
10002H	22
10003H	66

问题分析

内存情况示意图

物理地址	内存单元
10000H	23
10001H	11
10002H	22
10003H	66

指令	执行后相关寄存器中的内容	说明
mov ax,1000H	ax=1000H	这两条指令的目的是将ds设为1000H
mov ds, ax	ds=1000H	
mov ax, [0]	ax=1123H	物理地址1000:0处存放的字型数据送入ax； 物理地址1000:1单元处存放的字型数据的高8位为11H； 物理地址1000:1单元处存放的字型数据的低8位为23H； 所以1000:0处存放的字型数据为1123H。 指令执行时，字型数据的高8位送入ah,字型数据的低8位送入al，则ax中的数据为1123H
mov bx, [2]	bx=6622H	原理同上。
mov cx, [1]	cx=2211H	原理同上。

- 例题2：内存中的情况如图，画出下面指令执行后内存单元的数据存储情况。

代码情况：

```
mov ax, 1000H
mov ds, ax
mov ax, 11316
mov [0], ax
mov bx, [0]
mov  [1], bx
```

内存情况示意图

物理地址	内存单元
10000H	23
10001H	11
10002H	22
10003H	11

问题分析

内存情况示意图

物理地址	内存单元
10000H	34
10001H	22
10002H	11
10003H	11

指令	执行后相关寄存器中的内容	说明
mov ax,1000H	ax=1000H	这两条指令的目的是将ds设为1000H
mov ds, ax	ds=1000H	
mov ax, 11316	ax=2C34H	十进制11316，十六进制为2C34H
mov [0], ax	10000H 34 10001H 2C 10002H 22 10003H 11	ax中的字型数据送到物理地址1000:0单元处存放： ax中的字型数据为2C36H， 高8位为2CH，在ah中， 低8位为34H，在al中， 指令执行时，高8位送入物理地址中的高地址1000:1单元，低8位送入低地址1000:0单元
mov bx, [2]	bx=1122H	原理同上。
mov [1], bx	10000H 34 10001H 22 10002H 11 10003H 11	bx中的字型数据送到物理地址1000:1单元处存放： bx中的字型数据为1122H， 高8位为11H，在bh中， 低8位为22H，在bl中， 指令执行时，高8位送入物理地址中的高地址1000:2单元，低8位送入低地址1000:1单元

AH AL

AH	AL
2C	34

BH BL

BH	BL
11	22

4）寄存器间接寻址

指令中出现的是数据地址的存储地址。为了加快程序的执行，我们规定将数据地址存放在寄存器中，指令中指定该寄存器进行寻址。有：操作数的有效地址在基址寄存器BX、BP或者变址寄存器SI、DI中，而操作数则在存储器中。

MOV　AX, BX
SRC寄存器方式

MOV　AX, [BX]
SRC寄存器间接寻址方式

PA＝（DS）×10H＋（BX）/（SI）/（DI）
PA＝（SS）×10H＋（BP）

寄存器间接寻址* — EA 在基址寄存器(BX/BP) 或变址寄存器(SI/DI) 中

```
MOV   AX, [BX]          PA = 16d × (DS) + (BX)
MOV   AX, ES:[BX]       PA = 16d × (ES) + (BX)
MOV   AX, [BP]          PA = 16d × (SS) + (BP)
```

* 不允许使用AX、CX、DX 存放 EA

```
MOV   AX, [CX]    ×
```

* SRC 和 DST 的字长一致

```
MOV   DL, [ BX ]     ; [BX]指示一个字节单元
MOV   DX, [ BX ]     ; [BX]指示一个字单元
```

* 适于数组、字符串、表格的处理

描述下列指令的功能：

mov ax, [bx]

mov al, [bx]

将一个内存单元的内容送入**ax** 内
存单元长度为**2**字节（字）
偏移地址在**bx**中，段地址在**DS**中

将一个内存单元的内容送入**al** 内
存单元长度为**1**字节（字节）
偏移地址在**bx**中，段地址在**DS**中

[bx]功能描述

- mov ax,[bx]：
  - 将SA:EA处的数据送入ax中
  - bx中存放的数据作为一个偏移地址EA，段地址SA 默认在ds 中
- mov [bx],ax：
  - 将ax中的数据送入内存SA:EA处
  - bx中存放的数据作为一个偏移地址EA，段地址SA默认在ds中

问题

程序和内存中的情况如右图，写出程序执行后，21000H~21007H单元中的内容

说明：inc意为累加器加1

```
mov ax, 2000H
mov ds, ax
mov bx, 1000H
mov ax, [bx]
inc bx
inc bx
mov [bx], ax
inc bx
inc bx
mov [bx], ax
inc bx
mov [bx], al
inc bx
mov [bx], al
```

BE	21000H
00	21001H
	21002H
	21003H
	21004H
	21005H
	21006H
	21007H

- 前三条指令：

  mov ax,2000H

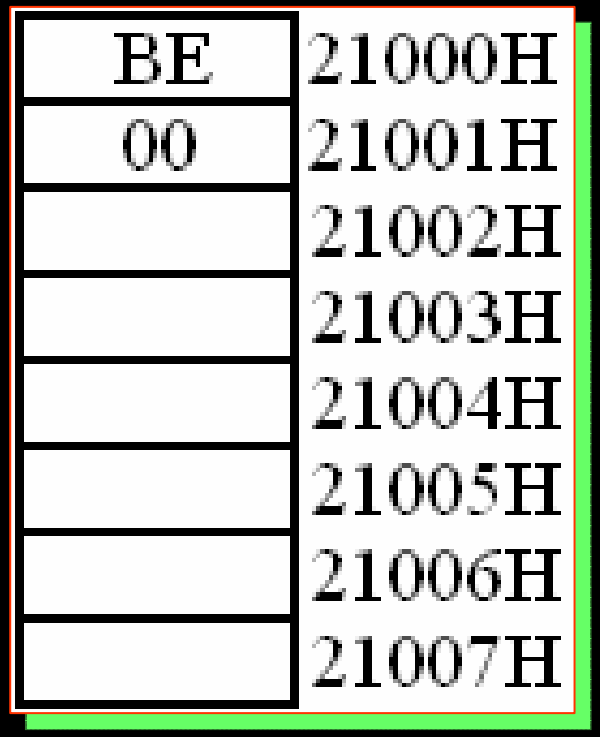
  mov ds,ax

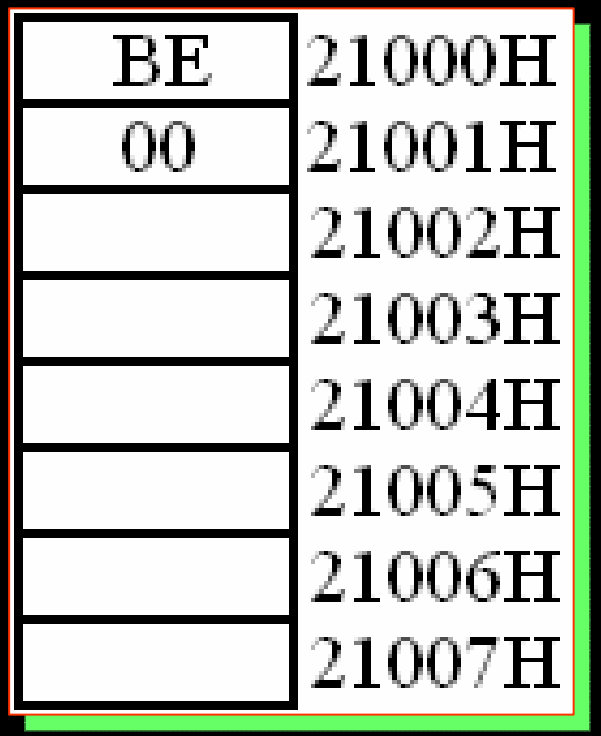
  mov bx,1000H

  这三条指令执行后

  ds=2000H，

  bx=1000H；





- 第4条指令：

  mov ax,[bx]

  指令执行前：

  ds=2000H，bx=1000H，

  则**mov ax,[bx]**将把内存**2000:1000**处的字型数据送入**ax**中。

  该指令执行后:

  ax=00beH。

- 第5、6条指令：

  inc bx

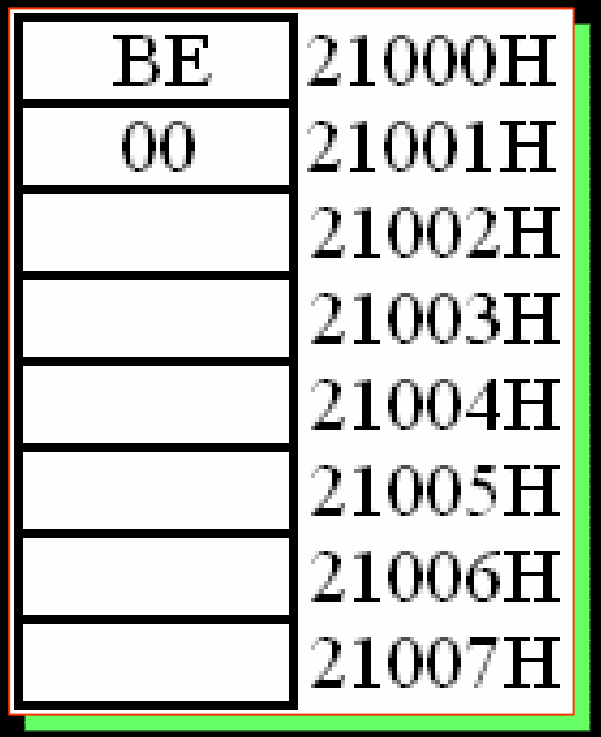
  inc bx

  指令执行前：

  bx=1000H。

  执行后：

  bx=1002H。



- 第7条指令：

  mov [bx],ax

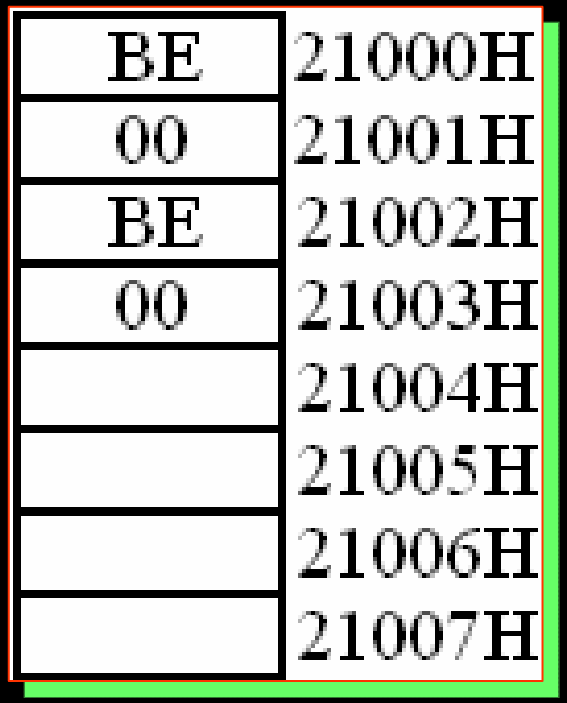
  指令执行前：

  ds=2000H，bx=1002H，则

  mov [bx],ax将把ax中的数据送入内存2000:1002处。

  指令执行后:

  2000:1002单元的内容为BE，2000:1003单元的内容为00。



- 第8、9条指令：

  inc bx

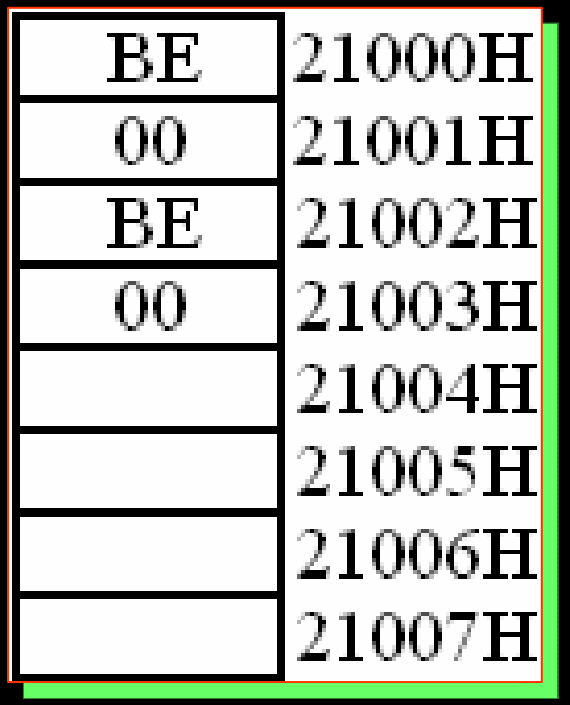
  inc bx

  指令执行前：

  bx=1002H。

  执行后：

  bx=1004H。



- 第10条指令：

  mov [bx],ax

  指令执行前：

  ds=2000H，bx=1004H，则

  mov [bx],ax将把ax中的数据

  送入内存2000:1004处。

  指令执行后：

  2000:1004单元的内容为BE，

  2000:1005单元的内容为00。

BE	21000H
00	21001H
BE	21002H
00	21003H
BE	21004H
00	21005H
	21006H
	21007H

- 第11条指令：

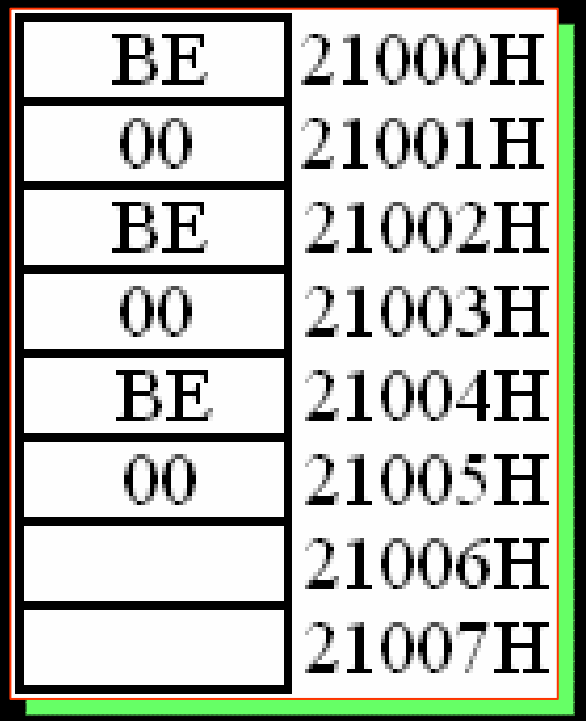
  inc bx

  指令执行前：

  bx=1004H。

  执行后：

  bx=1005H。



- 第12条指令：

  mov [bx],al

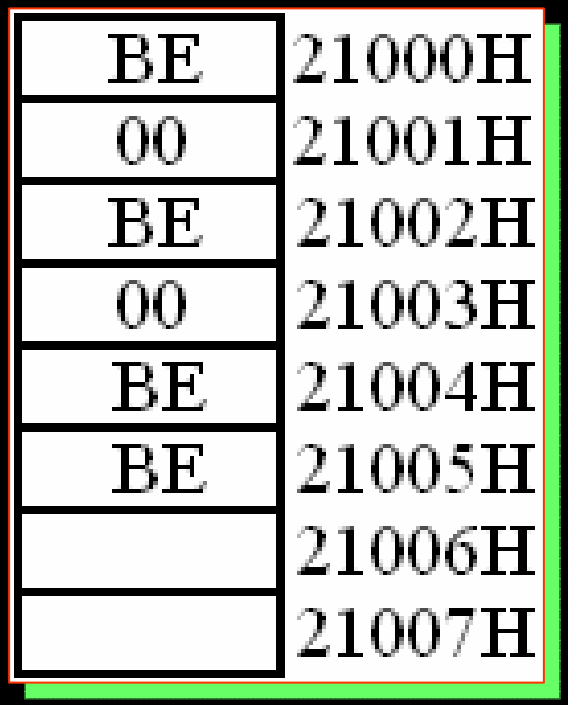
  指令执行前：

  ds=2000H，bx=1005H，则

  mov [bx],ax将把al中的数据送入内存2000:1005处。

  指令执行后：

  2000:1005单元的内容为BE。



- 第13条指令：

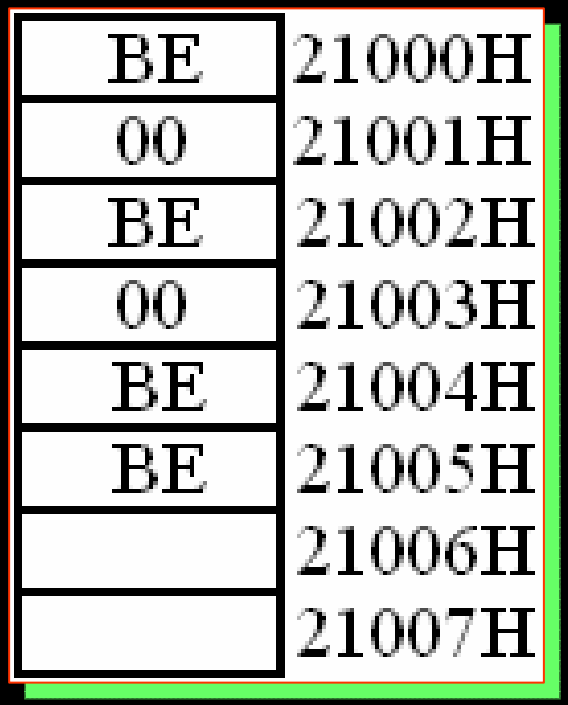
  inc bx

  指令执行前：

  bx=1005H，

  指令执行后：

  bx=1006H。



- 第14条指令：

  mov [bx],al

  指令执行前：

  ds=2000H，bx=1006H，则mov [bx],ax将把al中的数据送入内存2000:1006处。

  指令执行后：

  2000:1006单元的内容为BE。

BE	21000H
00	21001H
BE	21002H
00	21003H
BE	21004H
BE	21005H
BE	21006H
	21007H

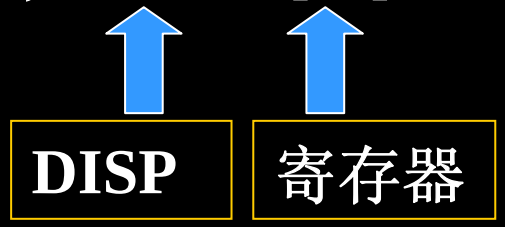
5、寄存器相对寻址

有效地址是一个基址或变址寄存器的内容和指令中指定的一个8位或者16位位移量（displacement)之和。

EA＝（BX)/(BP)/(SI)/(DI) ＋ 8/16位位移DISP

PA= (DS）×10H＋(BX)/(SI)/(DI)+DISP
PA= (SS）×10H＋(BP)+DISP

MOV AX, COUNT[SI]



COUNT是16位移量的符号地址，代表起点。寄存器SI中的值就是操作数相对起点位置的偏移距离。有个相当相近的数据表述形式，C语言中的数组元素A[i]

$$\text{有效地址} = \begin{cases} (BX) \\ (BP) \\ (SI) \\ (DI) \end{cases} + \begin{matrix} 8\text{位} \\ 16\text{位} \end{matrix} \text{位移量}$$

例： MOV AX, COUNT[SI] 或 MOV AX, [COUNT+SI]

假设(DS)=3000H, (SI)=2000H, COUNT=3000H

那么 PA = 35000H

假设(35000H) = 1234H, 那么 (AX) = 1234H

* 适于数组、字符串、表格的处理

6. 基址变址寻址方式

$$\text{有效地址} = \begin{Bmatrix} (BX) \\ (BP) \end{Bmatrix} + \begin{Bmatrix} (SI) \\ (DI) \end{Bmatrix}$$

MOV AX, [BX][DI]

或 **MOV AX, [BX+DI]**

MOV AX, ES:[BX][SI]

* 适于数组、字符串、表格的处理

* 必须是一个基址寄存器和一个变址寄存器的组合

× **MOV AX, [BX][BP]**

× **MOV AX, [SI][DI]**

7. 相对基址变址寻址方式

$$\text{有效地址} = \left\{\begin{matrix}(BX)\\(BP)\end{matrix}\right. + \left\{\begin{matrix}(SI)\\(DI)\end{matrix}\right. + \left\{\begin{matrix}8\text{位}\\16\text{位}\end{matrix}\right. \text{位移量}$$

MOV AX, MASK[BX][SI]

或 **MOV AX, MASK[BX+SI]**

或 **MOV AX, [MASK+BX+SI]**

* 适于堆栈处理和数组处理

段寄存器的使用规定

访问存储器的方式	默认的段寄存器	可跨越的段寄存器	偏移地址
取指令	CS	无	IP
堆栈操作	SS	无	SP
一般数据访问	DS	CS ES SS	有效地址EA
BP作为基址的寻址	SS	CS DS ES	BP
串操作的源操作数	DS	CS ES SS	SI
串操作的目的操作数	ES	无	DI

bx、 bp、si、di

(1) 在8086CPU 中，只有bx、 bp、si、di这4个寄存器可以在“[…]” 中进行内存单元的寻址

正确的指令

```
mov ax, [bx]
mov ax, [bx + si]
mov ax, [bx + di]
mov ax, [bp]
mov ax, [bp + si]
mov ax, [bp + di]
```

错误的指令

```
mov ax, [cx]
mov ax, [ax]
mov ax, [dx]
mov ax, [ds]
```


(2) 在“[…]” 中，bx、bp、si、di可单个出现，或只能以下列四种组合出现：

bx与si、bx与di、bp与si、bp与di

正确的指令

```
mov ax, [bx]
mov ax, [si]
mov ax, [di]
mov ax, [bp]
mov ax, [bx+si]
mov ax, [bx+di]
mov ax, [bp + si]
mov ax, [bp + di]
mov ax, [bx + si + idata]
mov ax,[bx + di + idata]
mov ax,[bp + si + idata]
mov ax,[bp + di + idata]
```

错误的指令

```
mov ax, [bx + bp]
mov ax, [si + di]
```


bx、 bp、si、di

（3）只要在[...]中使用寄存器bp，而指令中没有显性的给出段地址，段地址就默认在ss中，如：

- mov ax, [bp]　　// (ax)=((ss)*16+(bp))
- mov ax, [bp+idata]　　// (ax)=((ss)*16+(bp)+idata)
- mov ax, [bp+si]　　// (ax)=((ss)*16+(bp)+(si))
- mov ax, [bp+si+idata]　　// (ax)=((ss)*16+(bp)+(si)+idata)

