東北大學
Northeastern University

汇编语言程序设计

主讲：东北大学计算机学院　刘松冉

第三章 微型计算机的结构

四. 寻址方式

1. 操作数的种类
2. 寻址方式
3. 段更换与段跨越
4. 有效地址的计算时间
5. 指令系统

寻址方式 ： 指令中提供操作数或操作数地址的方法。寻址方式是规定如何对指令代码中操作数字段进行解释以找到操作数的方法。

计算机是进行数据处理、运算的机器，寻址解决了“要处理的数据在什么地方？”这一问题。

四. 寻址方式

1. 操作数（Operands）的种类

1) 立即操作数：指令要操作的数据在指令代码中，MOV AL, 10H （指令代码B010）

2) 存储器操作数：指令要操作的数据在存储器(内存)中，MOV AL, [1234H]（指令代码A03412）

3) 寄存器操作数：指令要操作的数据在CPU的寄存器中，MOV AL, BL（指令代码88D8）

4) I/O端口操作数：指令要操作的数据来自或送到I/O， IN AL，20H（指令代码E420）

2. 寻址方式 – 固定寻址

- 定义：指令要操作的数据在指令中并没有明确给出，但**隐含**在指令中
- 例子： MUL BL ; AL*BL → AX

在该指令中，AL和AX并未给出。

2. 寻址方式 – 立即寻址

- 定义：指令要操作的数据包含在指令码中
- 例子： MOV AX, 1234H （其指令代码B83412）

2. 寻址方式 – 寄存器直接寻址

- 定义：指令(码)中给出的寄存器的名字(编号), 要操作的数据在该寄存器中
- 例子：

```
INC   CX        ; 指令码 41
INC   DX        ; 指令码 42
INC   BX        ; 指令码 43
INC   SP        ; 指令码 44
INC   BP        ; 指令码 45
MOV CX, BX      ; 指令码 89D9H
```

2. 寻址方式 – 存储器寻址

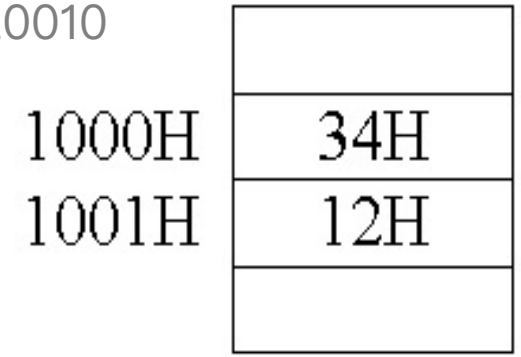
- 定义：要寻址的数据位于存储器(内存)中，在指令中是直接或间接的给出的存储器操作数的地址
- **存储器寻址包括以下几类：**
  1) 存储器直接寻址
  2) 寄存器间接寻址
  3) 基址寻址
  4) 变址寻址
  5) 基变址寻址

2. 寻址方式 – 存储器寻址

- 存储器寻址包括以下几类：
  1) **存储器直接寻址：**在存储器直接寻址中，指令直接给出的是操作数在内存中存放的地址

MOV AL, [1000H] ；指令码：A0010

MOV BX, [1000H] ；指令码：8B1E0010

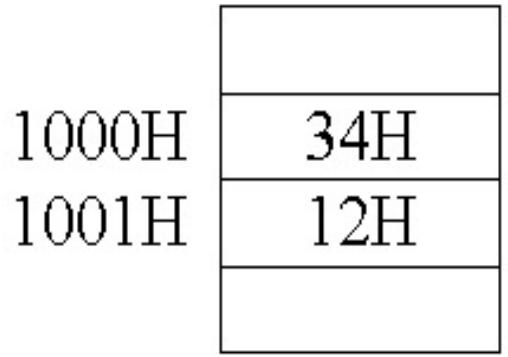


数据段寄存器DS=0000H

2. 寻址方式 – 存储器寻址

- 存储器寻址包括以下几类：
  1) 存储器直接寻址
  2) **寄存器间接寻址**：在寄存器间接寻址中，操作数位于内存中，操作数的地址位于某个寄存器中，在指令(码)中给出的是该寄存器的名字(编号)。

```
MOV  AL, [BX]  ；指令码8A07，偏移地址在BX中
MOV  BL, [SI]  ；指令码8B04
```



Case1: BX=1001H, SI = 1000H

Case2: BX=1000H, SI = 1000H

2. 寻址方式 – 存储器寻址

- 存储器寻址包括以下几类：
  1) 存储器直接寻址
  2) **寄存器间接寻址**：在寄存器间接寻址中，操作数位于内存中，操作数的地址位于某个寄存器中，在指令(码)中给出的是该寄存器的名字(编号)。
     注意：可以用于寄存器间接寻址的寄存器有BX, SI, DI

```
MOV  AL, [BX]  ；指令码8A07
MOV  BL, [SI]  ；指令码8B04
```

1000H	34H
1001H	12H

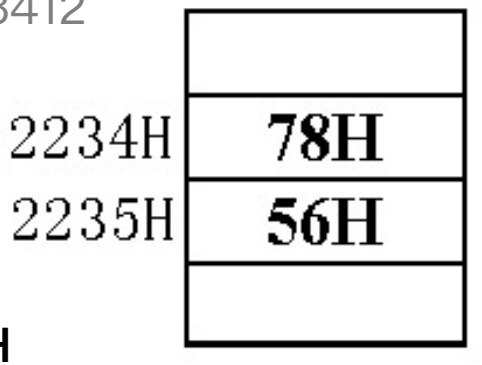
Case1: BX=1001H, SI = 1000H

Case2: BX=1002H, SI = 1000H

2. 寻址方式 – 存储器寻址

- 存储器寻址包括以下几类：
  1）存储器直接寻址
  2）寄存器间接寻址
  3）**基址寻址**：在基址寻址中，操作数位于位于内存中，操作数的地址由基址寄存器BX或BP与一个位移量相加给出，在指令(码)中给出的是该基址寄存器的名字(编号)及位移量

MOV AL, [BX+1234H] ;指令码:8A873412



BX=1000H

数据段寄存器DS=0000H

2. 寻址方式 – 存储器寻址 – 基址寻址

- 格式：[BX + idata]表示一个内存单元，它的偏移地址是 (BX)+idata (这里idata表示一个立即数)；该偏移地址对应的段基址存储在DS中
- 例1：用Debug查看内存，结果如下：

  2000:1000 BE 00 06 00 00 00 ……

  写出下面的程序执行后，ax、bx、cx中的内容。

1) **mov ax, [bx];** 访问的字单元的段地址在ds中（ds）=2000H，偏移地址在bx中（bx）=1000H；物理地址为2000:1000；**(ax) = 00BEH**
2) **mov cx, [bx+1];** 访问的字单元的段地址在ds中（ds）=2000H，偏移地址在 (bx)+1=1001H；物理地址为2000:1001；**(cx) = 0600H**
3) add cx, [bx+2]; 访问的字单元的段地址在ds中（ds）=2000H，偏移地址在 (bx)+2=1002H；物理地址为2000:1002；**(cx) = 0600H+0060H = 0606H**

```
mov ax,2000H
mov ds,ax
mov bx,1000H
mov ax,[bx]
mov cx,[bx+1]
add cx,[bx+2]
```

2. 寻址方式 – 存储器寻址 – 基址寻址

- 格式：[BX + idata]表示一个内存单元，它的偏移地址是 (BX)+idata (这里idata表示一个立即数)；该偏移地址对应的段基址存储在DS中
- 例2：用[BX + idata]的方式进行数组的处理（将字符串a[]转化为大写，b[]转化为小写）

```
1   char a[5]="BaSiC";
2   char b[5]="MinIX";
3   main(){
4       int i;
5       i=0;
6       do{
7           a[i]=a[i] & 0xDF;
8           b[i]=b[i] | 0x20;
9           i++;
10      }while(i<5);
11  }
```

2. 寻址方式 – 存储器寻址 – 基址寻址

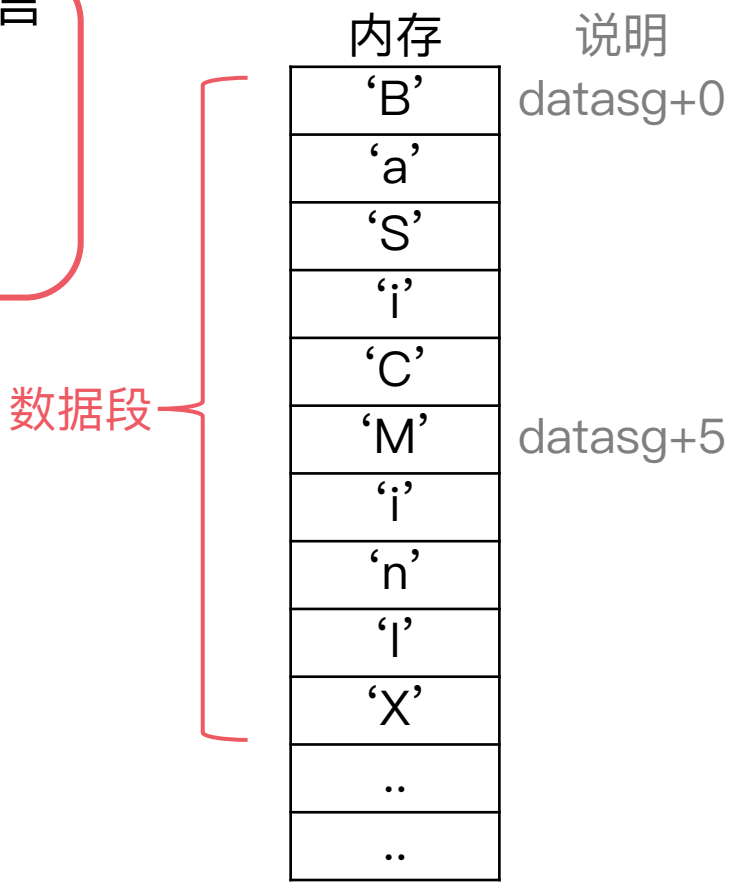
```
1   assume cs:codesg,ds:datasg
2   datasg segment
3   db 'BaSiC'
4   db 'MinIX'
5   datasg ends
6
7   codesg segment
8   start:  mov     ax, datasg
9           mov     ds, ax
10          mov     bx, 0
11
12          mov     cx, 5
13  s:      mov     al, [0+bx]      ;定位第一个字符串的字符
14          and     al, 1101111b
15          mov     [bx], al
16          mov     al, [5+bx]      ;定位第二个字符串的字符
17          mov     al, 0010000b
18          mov     [5+bx], al
19          inc     bx
20          loop    s
21  codesg ends
22  end start
```

[bx+idata]的寻址方式为高级语言实现数组提供了便利机制

C语言　　：　a[i], b[i]

汇编语言：　[0+bx], [5+bx]

　　0[bx], 5[bx]



2. 寻址方式 – 存储器寻址

- 存储器寻址包括以下几类：
  1）存储器直接寻址
  2）寄存器间接寻址
  3）**基址寻址**：在基址寻址中，操作数位于位于内存中，操作数的地址由基址寄存器BX或BP与一个位移量相加给出，在指令(码)中给出的是该基址寄存器的名字(编号)及位移量

  注意：基址寻址的格式为 [BX+位移量] 或 [BP+位移量]
  位移量范围为补码表示的16位（ –32768~+32767 ）

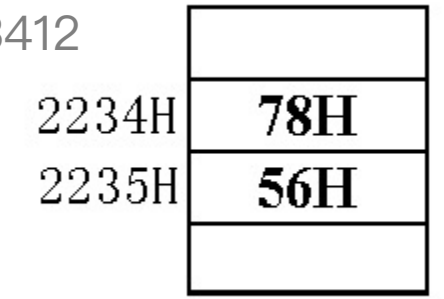
MOV AL, [BX+100H]
MOV AL, 100H[BX]

2. 寻址方式 – 存储器寻址

- 存储器寻址包括以下几类：
  1）存储器直接寻址
  2）寄存器间接寻址
  3）基址寻址
  4）**变址寻址**：在变址寻址中，操作数位于位于内存中，操作数的地址由变址寄存器SI或DI与一个位移量相加给出，在指令(码)中给出的是该变址寄存器的名字(编号)及位移量

MOV AL, [SI+1234H] ;指令码:8A843412

SI = 1000H



2. 寻址方式 – 存储器寻址

- 存储器寻址包括以下几类：
  1）存储器直接寻址
  2）寄存器间接寻址
  3）基址寻址
  4）**变址寻址**：在变址寻址中，操作数位于位于内存中，操作数的地址由变址寄存器SI或DI与一个位移量相加给出，在指令(码)中给出的是该变址寄存器的名字(编号)及位移量

  注意：– SI和DI是与BX功能相近的寄存器，但SI和DI不能分成两个8位寄存器来使用
  – 变址寻址的格式为[SI+位移量] 或 [DI+位移量]
  – 位移量范围为补码表示的16位（ –32768~+32767 ）

MOV AL, [SI+100H]
MOV AL, 100H[DI]

2. 寻址方式 – 存储器寻址 – 变址寻址

- 例3：用寄存器SI和DI实现将字符串‘welcome to masm!’复制到它后面的数据区中。
  - 我们编写的程序大都是进行数据的处理，而数据在内存中存放，所以我们在处理数据之前首先要搞清楚数据存储在什么地方，也就是说数据的内存地址。
  - 现在我们要对datasg 段中的数据进行复制，我们先来看一下要复制的数据在什么地方，datasg:0，这是要进行复制的数据的地址。
  - “welcome to masm!”从偏移地址0开始存放，长度为 16 个字节，所以，它后面的数据区的偏移地址为 16 ，就是字符串所要存放的空间。
  - 我们用ds:si 指向要复制的源始字符串，用 ds:di 指向复制的目的空间，然后用一个循环来完成复制。

```
1  assume cs:codesg,ds:datasg
2  datasg segment
3  db 'welcome to masm!'
4  db '................'
5  datasg ends
```

2. 寻址方式 – 存储器寻址 – 变址寻址

- 例3：用寄存器SI和DI实现将字符串‘welcome to masm!’复制到它后面的数据区中。
  - 我们编写的程序大都是进行数据的处理，而数据在内存中存放，所以我们在处理数据之前首先要搞清楚数据存储在什么地方，也就是说数据的内存地址。
  - 现在我们要对datasg 段中的数据进行复制，我们先来看一下要复制的数据在什么地方，datasg:0，这是要进行复制的数据的地址。
  - “welcome to masm!”从偏移地址0开始存放，长度为 16 个字节，所以，它后面的数据区的偏移地址为 16 ，就是字符串所要存放的空间。
  - 我们用ds:si 指向要复制的源始字符串，用 ds:di 指向复制的目的空间，然后用一个循环来完成复制。

```
1   assume cs:codesg,ds:datasg
2   datasg segment
3   db 'welcome to masm!'
4   db '................'
5   datasg ends
6
7   codesg segment
8   start:  mov     ax,datasg
9           mov     ds,ax
10          mov     si,0
11          mov     di,16
12          mov     cx,8
13  s:      mov     ax,[si]
14          mov     [di],ax
15          add     si,2
16          add     di,2
17          loop    s
18
19          mov     ax,4c00h
20          int     21h
21  codesg ends
22  end start
```

2. 寻址方式 – 存储器寻址

- 存储器寻址包括以下几类：
  1）存储器直接寻址
  2）寄存器间接寻址
  3）基址寻址
  4）变址寻址
  5）**基变址寻址**：在基变址寻址中，操作数位于位于内存中，操作数的地址由基址寄存器BX或BP与变址寄存器SI或DI及一个位移量相加给出，在指令(码)中给出的是寄存器的名字(编号)及位移量

MOV AL, [BX+SI+1234H] ；机器码8A803412

BX = 1000H，SI = 2000H

4234H	78H
4235H	56H

2. 寻址方式 – 存储器寻址 – 基变址寻址

- 正确格式：[BX+SI+idata]［BX+DI+idata]
  [BP+SI+idata]［BP+DI+idata]
  [BX+SI]［BX+DI]［BP+SI]　[BP+DI]
- 错误格式：[BX+BP]［SI+DI]
- 位移量（idata）范围为补码表示的16位（ –32768 ~ +32767
- 例4： 用Debug查看内存，结果如下：

  2000:1000 BE 00 06 00 00 00 ……

```
1    mov ax,2000H
2    mov ds,ax
3    mov bx,1000H
4    mov si,0
5    mov ax,[bx+si]
6    inc si
7    mov cx,[bx+si]
8    inc si
9    mov di,si
10   mov ax,[bx+di]
```

2. 寻址方式 – 存储器寻址 – 基变址寻址

- 正确格式：[BX+SI+idata]［BX+DI+idata]
  [BP+SI+idata]［BP+DI+idata]
  [BX+SI]［BX+DI]［BP+SI]　[BP+DI]
- 错误格式：[BX+BP]［SI+DI]
- 位移量（idata）范围为补码表示的16位（ –32768 ~ +32767
- 例4： 用Debug查看内存，结果如下：

  2000:1000 BE 00 06 00 00 00 ……

  - mov cx,[bx+si]

    访问的字单元的段地址在ds中，(ds)=2000H；

    偏移地址=(bx)+(si)=1001H；

    指令执行后(cx)=0600H。

```
1    mov ax,2000H
2    mov ds,ax
3    mov bx,1000H
4    mov si,0
5    mov ax,[bx+si]
6    inc si
7    mov cx,[bx+si]
8    inc si
9    mov di,si
10   mov ax,[bx+di]
```

2. 寻址方式 – 存储器寻址 – 基变址寻址

- 正确格式：[BX+SI+idata]［BX+DI+idata]
  [BP+SI+idata]［BP+DI+idata]
  [BX+SI]［BX+DI]［BP+SI]　[BP+DI]
- 错误格式：[BX+BP]［SI+DI]
- 位移量（idata）范围为补码表示的16位（ –32768 ~ +32767
- 例4： 用Debug查看内存，结果如下：

  2000:1000 BE 00 06 00 00 00 ……

  - add cx,[bx+di]

    访问的字单元的段地址在ds中，
    (ds)=2000H；
    偏移地址=(bx)+(di)=1002H；
    指令执行后(cx)=0606H。

```
1    mov ax,2000H
2    mov ds,ax
3    mov bx,1000H
4    mov si,0
5    mov ax,[bx+si]
6    inc si
7    mov cx,[bx+si]
8    inc si
9    mov di,si
10   mov ax,[bx+di]
```

2. 寻址方式 – 存储器寻址 – 基变址寻址

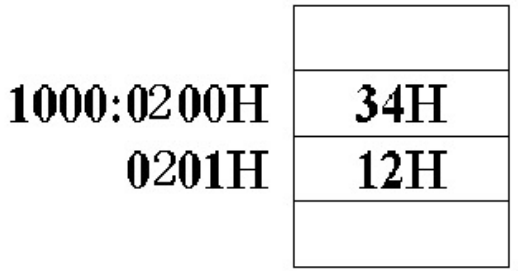
- 正确格式：[BX+SI+idata]［BX+DI+idata]
  [BP+SI+idata]［BP+DI+idata]
  [BX+SI]［BX+DI]［BP+SI]　[BP+DI]
- 错误格式：[BX+BP]［SI+DI]
- 位移量（idata）范围为补码表示的16位（ –32768 ~ +32767 ）
- 指令mov ax,[bx+si+idata]， 也可以写成如下格式（常用）：
  - mov ax, [bx+200+si]
  - mov ax, [200+bx+si]
  - mov ax, 200[bx][si]
  - mov ax, [bx].200[si]
  - mov ax, [bx][si].200

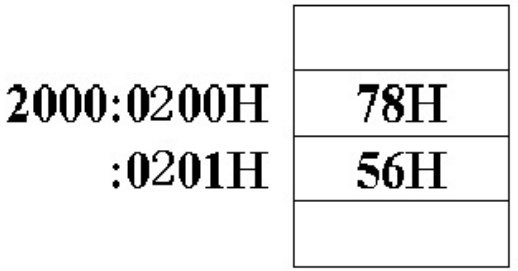
2. 寻址方式 – 存储器寻址

- 存储器寻址方式中的段地址
  - ➢ 在存储器寻址方式中只给出了偏移地址，其段地址是隐含的，一般情况下，是DS，只有特殊情况下是SS
  - ➢ 例子：假定 DS=1000H, SS=2000H, BP=0100H, BX=0100H, 如下指令在执行完后的结果分别是什么？

MOV AX, [BX+100H]

MOV AX, [BP+100H]





2. 寻址方式 – 存储器寻址（总结）

- 如果我们比较一下前而用到的几种定位内存地址的方法（可称为寻址方式），就可以发现有以下几种方式：
  1) [idata] 用一个常量来表示地址，可用于直接定位一个内存单元；
  2) [bx]用一个变量来表示内存地址，可用于间接定位一个内存单元；
  3) [bx+idata] 用一个变量和常量表示地址，可在一个起始地址的基础上用变量间接定位一个内存单元；
  4) [bx+si]用两个变量表示地址；
  5) [bx+si+idata] 用两个变量和一个常量表示地址。
- 从[idata]一直到[bx+si+idata]，我们可以用更加灵活的方式来定位一个内存单元的地址。这使我们可以从更加结构化的角度来看待所要处理的数据。

2. 寻址方式 – 存储器寻址（总结）

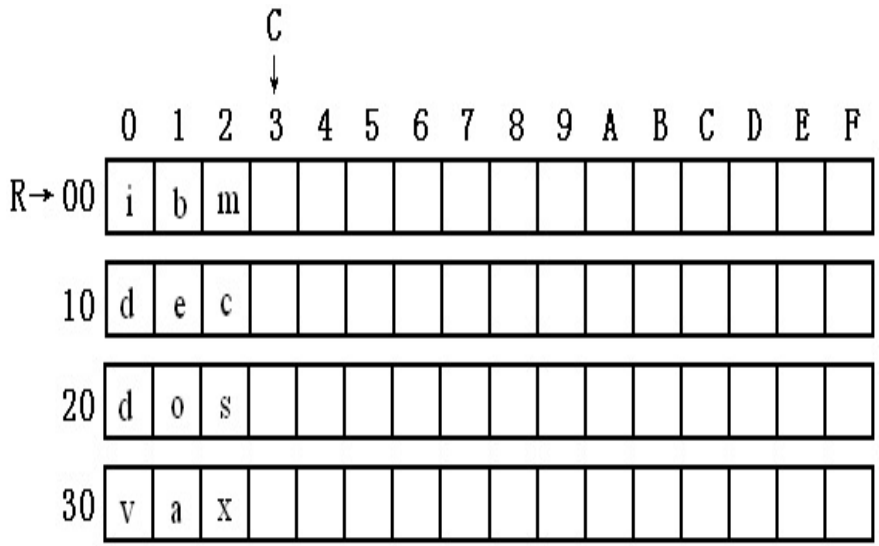
寻址方式	含义		名称	常用格式举例
[idata]	EA=idata;	SA=(DS)	存储器直接寻址	[idata]
[BX]	EA=(BX);	SA=(DS)	寄存器间接寻址	[BX]
[SI]	EA=(SI);	SA=(DS)		[SI]
[DI]	EA=(DI);	SA=(DS)		[DI]
[BP]	EA=(BP);	SA=(SS)		[BP]
[BX + idata]	EA=(BX)+idata;	SA=(DS)	基址寻址	用于结构体：[BX].idata 用于数组：idata[SI], idata[DI] 用于二维数组：[BX][idata]
[BP + idata]	EA=(BP)+idata;	SA=(SS)	基址寻址	
[SI + idata]	EA=(SI)+idata;	SA=(DS)	变址寻址	
[DI + idata]	EA=(DI)+idata;	SA=(DS)	变址寻址	
[BX + SI]	EA=(BX)+(SI);	SA=(DS)	基变址寻址	用于二维数组：[BX][SI]
[BX + DI]	EA=(BX)+(DI);	SA=(DS)		
[BP + SI]	EA=(BP)+(SI);	SA=(SS)		
[BP + DI]	EA=(BP)+(DI);	SA=(SS)		
[BX + SI + idata]	EA=(BX)+(SI)+idata;	SA=(DS)	相对基变址寻址	用于表格（结构）中的数组项： [BX].idata[SI] 用于二维数组： idata[BX][SI]
[BX + DI + idata]	EA=(BX)+(DI)+idata;	SA=(DS)		
[BP + SI +idata]	EA=(BP)+(SI)+idata;	SA=(SS)		
[BP + DI +idata]	EA=(BP)+(DI)+idata;	SA=(SS)		

在8086CPU 中，只有这4个寄存器（bx、bp、si、di）可以用在“[...]”中来进行内存单元的寻址。

2. 寻址方式 – 存储器寻址（例子）

- 编程： 将datasg段中每个单词改为大写字母 。
- 分析：datasg中的数据的存储结构，如图

```
1  assume cs:codesg,ds:datasg
2  datasg segment
3      db 'ibm             '
4      db 'dec             '
5      db 'dos             '
6      db 'vax             '
7  datasg ends
```

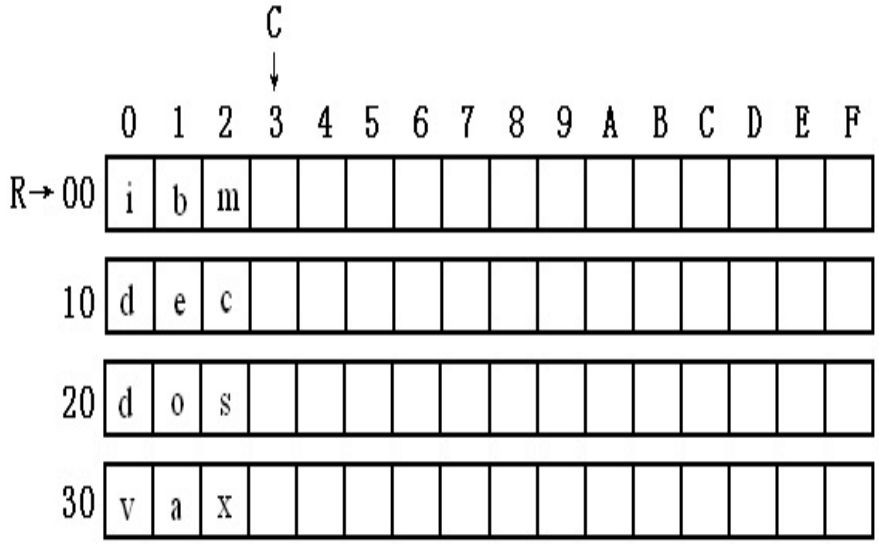


2. 寻址方式 – 存储器寻址（例子）

- 编程： 将datasg段中每个单词改为大写字母 。
- 分析：datasg中的数据的存储结构，如图

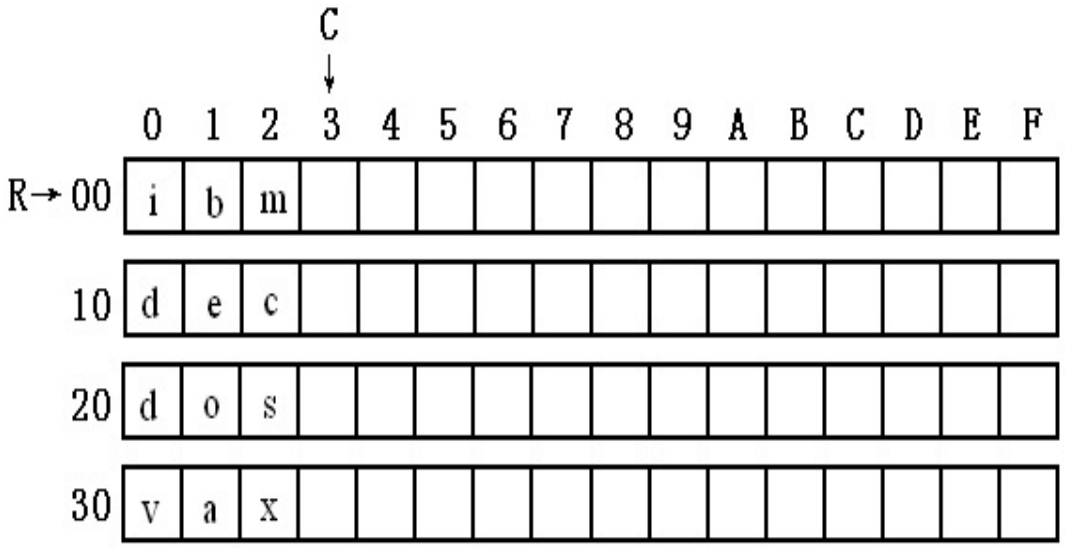
1) 在datasg中定义了4个字符串，每个长度为16字节 。
2) 因为它们是连续存放的，我们可以将这 4 个字符串看成一个 4行16列的二维数组 。
3) 按照要求，我们需要修改每一个单词，即二维数组的每一行的前3列 。
4) 我们需要进行4x3次的二重循环，用变量R 定位行，变量C定位列。
5) 我们首先用R 定位第1行，然后循环修改R行的前3列；然后再用R定位到下一行，再次循环修改R行的前3列……，如此重复直到所有的数据修改完毕。

```
1   assume cs:codesg,ds:datasg
2   datasg segment
3       db 'ibm             '
4       db 'dec             '
5       db 'dos             '
6       db 'vax             '
7   datasg ends
```



2. 寻址方式 – 存储器寻址（例子）

- 结构： 我们用bx来作变量，定位每行的起始地址，用si 定位要修改的列，用 [bx+si] 的方式来对目标单元进行寻址




```
 9   codesg segment
10   start:  mov     ax, datasg
11           mov     ds, ax
12           mov     bx, 0
13
14           mov     cx, 4
15   s0:     mov     si, 0
16           mov     cx, 3
17   s:      mov     al, [bx+si]
18           and     al, 11011111b
19           mov     [bx+si], al
20           inc     si
21           loop    s
22
23           add     bx, 16
24           loop    s0
25   codesg ends
26   end start
```