16位的间接寻址方式：
[bx],[bp],[si],[di]（只能用这四个寄存器）
[bx+2],[bp-2],[si+2],[di-3]；[bx+si],[bx+di],[bp+si],[bp+di]
（注意：只有这几种相加形式,b开头的加上i结尾的）
（注意：32位中取消了这种限制，任意两个寄存器都一样相加，甚至[ebx+ebx]可以自己相加）
[bx+si+2],[bx+di-2],[bp+si+1],[bp+di-1]

1． 32 位间接寻址方式

(1) 32 位比 16 位多了以下这种寻址方式：

[寄存器+寄存器*n+常数]

其中 n=2、4、8。

例如：

```
mov eax, [ebx+esi*4+6]
```

VC 里面要查看当前 C 代码对应的机器语言，可以在按 F10 开始调试后选菜单：

View->Debug Windows->Disassembly

TC 里面要查看当前 C 代码对应的机器语言：

先把 ary.c(http://10.71.45.100/bhh/ary.c)拷到 dosbox86\tc，

集成环境中选菜单 File->Dos Shell->

```
cd \tc
tc
File->Load->ary.c
Compile->Compile
Compile->Link
File->Quit
td ary.exe
View->CPU
```

这种寻址方式的应用：

```
long a[10]={...};
int i, n=10, sum=0;
for(i=0; i<n; i++)
   sum += a[i];
```

设 ebx=&a[0], esi=0, eax=0, 则上述 C 代码可转化

成以下汇编代码：

```
again:
add eax, [ebx+esi*4]
add esi, 1
cmp esi, 10
jb again
```

(2) 32 位寻址方式里面，对[]中的两个寄存器几乎不加限制

例如：但是，段寄存器不能放在方括号内，ip寄存器（32位中为eip）、fg寄存器（32位为efg）不能放在方括号中

ebx, ebp, esi, edi,

eax, ecx, edx, esp 都可以放到[]里面；

mov eax, [ebx+ebx*4]；两个寄存器可以任意组合

32位比16位多了以下这种寻址方式：[寄存器+寄存器*n+常数]（其中n=2、4、8）

（为什么n=2、4、8？因为这与不同类型的sizeof值有关系）

2．段跨越(segment overriding)

例如:mov eax,[ebx+esi*4+6]

对于long类型数组a，&a[i]的地址一定等于a的首地址+i*4

通过在操作数前添加一个段前缀(segment prefix)如 CS:、DS:、ES:、SS:来强制改变操作数的段址，这就是段跨越。

段地址的隐含：

```
mov ax, [bx]
mov ax, [si]
mov ax, [di+2]
mov ax, [bx+si]
mov ax, [bx+di+2]
mov ax, [1000h]
```

上述指令的源操作数都省略了段地址 ds。

[bp], [bp+2], [bp+si+2], [bp+di-1]

等价于

ss:[bp], ss:[bp+2], ss:[bp+si+2], ss:[bp+di-1]

当[]中包含有寄存器 bp 时，该变量的段地址一定是 ss。

例如：

mov ax, [bp+2] 相当于

```
mov ax, ss:[bp+2]
```

默认的段地址是可以改变的，例如：

```
mov ax, ds:[bp+2]
```

这条指令的源操作数段地址从默认的 ss 改成了 ds。

同理，

`mov ax, [bx+si+2]`改成 `mov ax, ss:[bx+si+2]`的话，默认段地址就从 ds 变成了 ss。

3．通用数据传送指令：MOV，PUSH，POP，XCHG

```
mov byte ptr ds:[bx], byte ptr es:[di]
```

错误原因:两个操作数不能同时为内存变量

以下为正确写法：

```
mov al, es:[di]
mov ds:[bx], al
```

32 位 push、pop 过程演示：

http://10.71.45.100/bhh/stk1.txt 代码

http://10.71.45.100/bhh/stk2.txt 堆栈布局

push/pop 后面也可以跟变量,例如：

```
push word ptr ds:[bx+2]
pop word ptr es:[di]
```

8086 中，push 不能跟常数，但 80386 及以后的 cpu 允许 push 一个常数。

push/pop 后面不能跟一个 8 位的寄存器或变量。

```
mov ax, 1
mov bx, 2
xchg ax, bx; 则 ax=2, bx=1
```

交换指令XCHG是两个寄存器，寄存器和内存变量之间内容的交换指令，两个操作数的数据类型要相同，可以是一个字节，也可以是一个字，也可以是双字

4. 除法指令：div

(1) 16 位除以 8 位得 8 位

ax / 除数 = AL..AH

商默认放在al中，余数默认放在ah中，
也就是说在除法运算中，被除数会被损坏

例如：div bh

设 AX=123h, BH=10h

div bh; AL=12h, AH=03h

(2) 32 位除以 16 位得 16 位

dx:ax / 除数 = ax..dx

注意在这里dx:ax并不表示段地址和偏移地址，只是表示dx为高16位，ax为低16位，两者组合成为32位

例如：div bx

设 dx=123h, ax=4567h, bx=1000h

```
div bx ; 1234567h/1000h
       ; AX=1234h, DX=0567h
```

(3) 64 位除以 32 位得 32 位

edx:eax / 除数 = eax..edx

例如：div ebx

假定要把一个 32 位整数如 7FFFFFFFh 转化成十进制格式

则一定要采用(3)这种除法以防止发生除法溢出。

代码：http://10.71.45.100/bhh/val2decy.asm

5. 地址传送指令：LEA, LDS, LES

(1) lea dest, src

```
lea dx, ds:[1000h] ; DX=1000h
mov dx, 1000h; 上述 lea 指令的效果等同于 mov 指令

lea dx, abc ; 效果等价于以下指令
mov dx, offset abc

lea dx, ds:[bx+si+3]; dx=bx+si+3
mov dx, bx+si+3; 错误
mov dx, bx; \
add dx, si;  | 效果等同于上述 lea 指令
add dx, 3 ; /
```

mov dx,1+2+3;正确　　mov ((1*2+3) shl 2) xor 5;正确　　只要右侧纯常数，mov就不会错

这些计算是在编译阶段由编译器完成计算，而不是在程序执行过程中进行计算的，下面展示了lea的特殊用法

```
lea eax, [eax+4*eax]; EAX=EAX*5 用 lea 做乘法
lea eax, [eax+eax*2]; EAX=EAX*3
```

(2) 远指针(far pointer)

16 位汇编中，远指针是指 16 位段地址+16 位偏移地址；

32 位汇编中，远指针是指 16 位段地址+32 位偏移地址。

近指针(near pointer):

16 位汇编中，近指针是指 16 位的偏移地址；

32 位汇编中，近指针是指 32 位的偏移地址；

在C语言中编写程序时，一般不会涉及到远指针

远指针(far pointer)包括段地址及偏移地址两个部分；

近指针(near pointer)只包括偏移地址，不包含段地址。

假定把一个远指针 1234:5678 存放到地址 1000:0000 中，则内存布局如下：

```
1000:0000 78h
1000:0001 56h
```

```
1000:0002 34h
1000:0003 12h
```

假定要把 1000:0000 中存放的远指针取出来，存放到 es:bx 中，则

LES(load ES)指令的功能是：把内存中指定位置的双字操作数的低位字装入指令中指定的寄存器、高位字装入ES寄存器。

```
mov ax, 1000h
mov ds, ax
les bx, dword ptr ds:[0000h]
        ; es=1234h, bx=5678h
```

远指针的汇编语言例子：

http://10.71.45.100/bhh/les.asm

近指针的汇编语言例子：

http://10.71.45.100/bhh/nearptr.asm

远指针的 C 语言例子：

http://10.71.45.100/bhh/farptr.c