8086常用指令

以下名词采用的缩写：

- 寄存器：`reg`、`r16`、`r8`
- 内存：`mem`、`mem16`
- 立即数：`imm`
- 段寄存器：`seg`

数据传输类指令

mov

把一个字节或字的操作数从源地址传送至目的地址

```
mov reg/mem, imm
mov reg/mem/seg, reg
mov reg/seg, mem
mov reg/mem, seg
```

注意：

1. 指令中的源操作数绝对不能是立即数和代码段CS寄存器
2. 指令中绝对不允许在两个存储单元之间直接传送数据
3. 指令中绝对不允许在两个段寄存器之间直接传送数据
4. 立即数不能直接送段寄存器
5. 指令不会影响标志位

xchg

交换数据

```
xchg reg, reg/mem
```

注意：

1. 不能在存储器和存储器之间交换数据

xlat

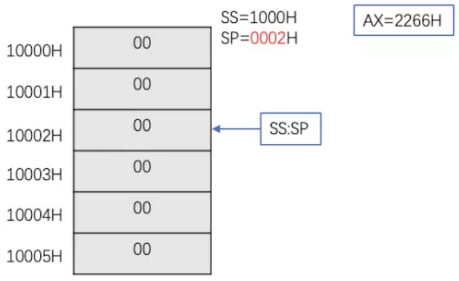
将`ds:[bx+al]`处的一个字节数据取出赋值给`al`。即`xlat`相当于`mov al, ds:[bx+al]`

push和pop

push

先使堆栈指针`sp - 2`，再把一个**字**操作数压入栈中

```
push r16/m16/seg
```



AX=2266H

CPU执行PUSH AX

第一步：SP=SP-2

SP=0004H-0002H=0002H

SS=1000H，SP=0002H
SS:SP指向1000:0002=1000H*10H+0002H=10002H

当前栈顶地址变为10002H

AX=2266H

CPU执行PUSH AX

第二步：将2266压入栈

将两个字节的数据2266H放入地址为10002H内存中
需要从10002H开始往高取两个字节的内存来存放数据

使用10002H、10003H来存放2266H

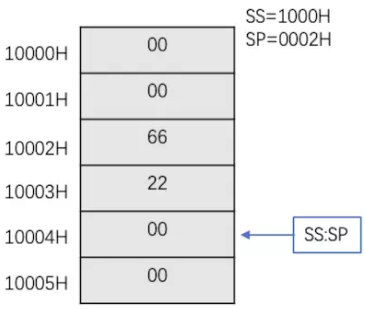
因为是小端模式，高字节放在高地址，低字节放在低地址

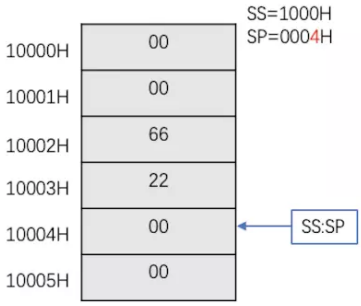
10002H放66
10003H放22

pop

先使栈中弹出一个**字**至目的操作数中，再让堆栈指针`sp + 2`

```
pop r16/m16/seg
```





CPU 执行pop bx

第一步

将SS:SP指向的内存单向的数据放入bx中

bx=2266H

CPU 执行pop bx

第二 步

SP=SP+2

注意：

1. 堆栈操作的单位是**字**
2. 压栈和弹出都是低地址送低地址，高地址送高地址
3. 堆栈的操作遵循**FILO**原则，但可用存储器寻址方式随机存去堆栈中的数据
4. 堆栈的用途
    - 临时存放数据

- 传递参数
- 保存和恢复寄存器

lahf、sahf、pushf和popf

标志寄存器传送指令用来传送标志寄存器`FLAGS`的内存，方便进行对各个标志位的直接操作

有2对4条指令：

- 低8位传送：`lahf`和`sahf`
- 16位传送：`pushf`和`popf`

lahf

将标志寄存器`FLAGS`的低字节送入寄存器`ah`中

其中`sf、zf、af、pf、cf`状态标志分别送入`ah`的第`7、6、4、2、0`位

sahf

将寄存器`ah`的值送入`FLAGS`的低字节中

pushf

先使堆栈指针`sp - 2`，再把标志寄存器`FLAGS`压入栈中

popf

先使栈中弹出一个**字**至标志寄存器`FLAGS`，再让堆栈指针`sp + 2`

lea

将存储器操作数的有效地址传送至指定的16位寄存器中

```
lea r16, mem
```

注意：

1. 获取的是主存储单元的有效地址而不是物理地址，也不是单元里的内容
2. 可实现计算功能

lds和les

lds

将主存中指定的`mem`指定的字送至16位寄存器中，并将`mem`的下一个字送`ds`寄存器

```
; r16 <-- mem
; ds <-- mem + 2
lds r16, mem
```

les

将主存中指定的`mem`指定的字送至16位寄存器中，并将`mem`的下一个字送`es`寄存器

```
; r16 <-- mem
; es <-- mem + 2
les r16, mem
```

in和out

8086通过输入输出指令与外设进行数据交换，呈现给程序员的外设是端口（port）即I/O地址

8086用于寻址外设端口的地址线为16条，端口最多为2^16=65536个，即64K个，端口号为`0000H ~ FFFFH`

没个端口用于传送一个字节的外设数据

8086的端口无需分段，有两种寻址方式：

1. 直接寻址：只用于寻址`00H ~ FFH`前256个端口，操作数`i8`表示端口号
2. 间接寻址：可用于寻址全部64k个端口，`dx`寄存器的值就是端口号

对于大于`FFH`的端口只能采用间接寻址方式

in

将外设数据传送给`al/ax`

```
in al, i8   ; 直接寻址
in al, dx   ; 间接寻址
in ax, i8   ; 直接寻址
in ax, dx   ; 间接寻址
```

out

将`al/ax`的数据传送给外设

```
out i8, al  ; 直接寻址
out dx, al  ; 间接寻址
out i8, ax  ; 直接寻址
out dx, ax  ; 间接寻址
```

算术运算类指令

add

将源与目的操作数相加，结果送到目的操作数

```
add reg, imm/reg/mem
add mem, imm/reg
```

adc

将源与目的操作数相加，再加上进位`CF`标志，结果送到目的操作数

主要与`add`指令配合，实现多精度加法运算

```
adc reg, imm/reg/mem
adc mem, imm/reg
```

inc

增量操作，对操作数加1，且不影响进位标志`CF`

```
inc reg/mem
inc bx
inc byte ptr [bx]   ; byte ptr 为修饰符，指明操作的是字节单位
```

sub

将目的操作数减去源操作数，结果送到目的操作数

```
sub reg, imm/reg/mem
sub mem, imm/reg
```

sbb

将目的操作数减去源操作数，再减去借位（进位）标志`CF`，结果送到目的操作数

主要与`sub`指令配合，实现多精度减法运算

```
sbb reg, imm/reg/mem
sbb mem, imm/reg
```

dec

减量操作，对操作数减1，且不影响进位标志`CF`

```
dec reg/mem
```

neg

对操作数执行求补运算：用0减去操作数，然后结果返回操作数

求补运算也可以表达成：将操作数按位取反后加1

该指令对标志的影响与用0做减法的`sub`指令一样

```
neg reg/mem
```

cmp

将目的操作数减去源操作数，结果并不会送到目的操作数

```
cmp reg, imm/reg/mem
cmp mem, imm/reg
```

mul和imul

乘法指令分为**无符号乘法**和**有符号乘法**

乘法指令的源操作数显示给出，隐式使用另一个操作数`ax`和`dx`

- 字节量相乘
  - `al`与`r8/m8`相乘，得16位结果，存入`ax`
- 字量相乘
  - `ax`与`r16/m16`相乘，得32位结果，高字存入`dx`，低字存入`ax`

可利用`OF`溢出标志和`CF`进位标志判断乘积的高一半是否具有有效数值

mul

无符号乘法

```
mul r8/m8   ; 结果存入ax中
mul r16/r16 ; 结果高字存入dx，低字存入ax
```

乘积的高一半（`ah`或`dx`）为0，则`OF=CF=0`，否则`OF=CF=1`

imul

有符号乘法

```
imul r8/m8      ; 结果存入ax中
imul r16/m16    ; 结果高字存入dx，低字存入ax
```

乘积的高一半是低一半的符号扩展，则`OF=CF=0`，否则`OF=CF=1`

div和idiv

除法指令分为**无符号除法**和**有符号除法**

除法指令的除数显示给出，隐式使用另一个操作数`ax`和`dx`作为被除数

- 字节量除法

    - `ax` 除以 `r8/m8`，8位的商存入 `al`，8位余数存入 `ah`
- 字量除法
    - `dx ax` 除以 `r16/m16`，16位的商存入 `ax`，16位余数存入 `dx`

除法指令会产生结果溢出

div

无符号除法

```
div r8/m8       ; 商在al，余数在ah
div r16/m16     ; 商在dx，余数在dx
```

idiv

有符号除法

```
idiv r8/m8      ; 商在al，余数在ah
idiv r16/m16    ; 商在dx，余数在dx
```

cbw和cwd

符号扩展时用一个操作数和的符号位（最高位）形成另一个操作数，后一个操作数的各位全是0（正数）或全是1（负数）。符号扩展不改变数据大小。

cbw

令 `al` 的符号扩展至 `ah`

```
mov al, 80h
cbw ; ax = ff80h
```

cwd

令 `ax` 的符号扩展至 `dx`

```
; 实现 ax / bx
cwd ; dx ax <-- ax  利用符号扩展获得被长于除数的被除数
idiv bx ; ax <-- dx ax / bx
```

BCD码

二进制编码的十进制数：一位十进制数用4位二进制编码来表示

例如：

真值	8	64
二进制编码	08h	40h
压缩BCD码	08h	64h
非压缩BCD码	08h	0604h

对照表如下：

十进制	二进制	十六进制	BCD 码（8421 码）	5421 码	2421 码
0	0000	0	0000	0000	0000
1	0001	1	0001	0001	0001
2	0010	2	0010	0010	0010
3	0011	3	0011	0011	0011
4	0100	4	0100	0100	0100
5	0101	5	0101	1000	1011
6	0110	6	0110	1001	1100
7	0111	7	0111	1010	1101
8	1000	8	1000	1011	1110
9	1001	9	1001	1100	1111

位操作类指令

and、or、not和xor

and

对两个操作数执行逻辑与运算，结果送到目的操作数

```
and reg, imm/reg/mem
and mem, imm/reg
```

指令设置`cf = of = 0`，根据结果设置`sf`、`zf`、`pf`

or

对两个操作数执行逻辑或运算，结果送到目的操作数

```
or reg, imm/reg/mem
or mem, imm/reg
```

not

对一个操作数执行逻辑非运算

```
not reg/mem
```

指令不影响标志位

xor

对两个操作数执行逻辑异或运算，结果送到目的操作数

```
xor reg, imm/reg/mem
xor mem, imm/reg
```

指令设置`cf = of = 0`，根据结果设置`sf`、`zf`、`pf`

test

对两个操作数执行逻辑与运算，结果不送到目的操作数

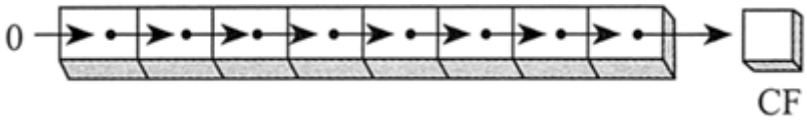
```
test reg, imm/reg/mem
test mem, imm/reg
```

指令设置`cf = of = 0`，根据结果设置`sf`、`zf`、`pf`

shl和shr

`shl`逻辑左移，最高位进入`cf`，最低为补0

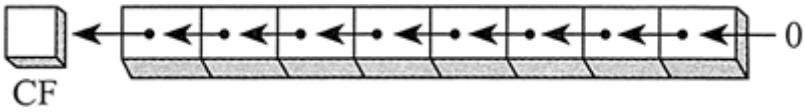
`shr`逻辑右移，最低位进入`cf`，最高为补0



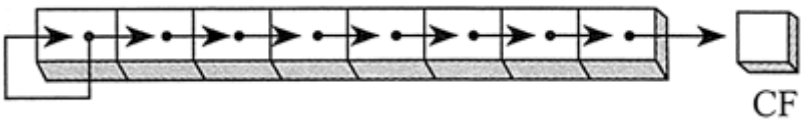

```
shl reg/mem, 1/cl
shr reg/mem 1/cl
```

sal和sar

`sal`算术左移，最高位进入`cf`，最低位补0



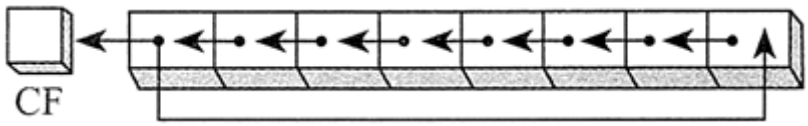
`sar`算术右移，最低位进入`cf`，最高位不变



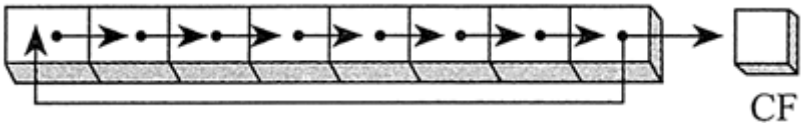
```
sal reg/mem, 1/cl
sar reg/mem 1/cl
```

rol和ror

`rol`不带进位的循环左移



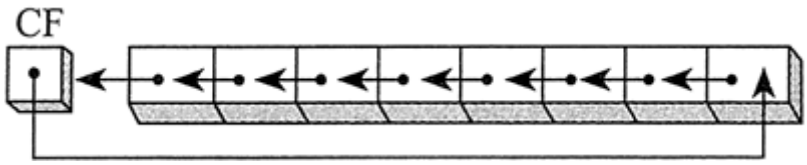
`ror`不带进位的循环右移



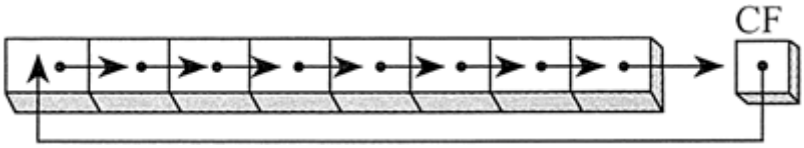

```
rol reg/mem, 1/cl
ror reg/mem, 1/cl
```

rcl和rcr

`rcl`带进位的循环左移



`rcr`带进位的循环左移右移




```
rcl reg/mem, 1/cl
rcr reg/mem, 1/cl
```

串操作类指令

串操作指令的操作数是内存中连续存放的数据串，即在连续的内存区域中的字节或字的序列

串操作指令的操作对象以字（w）为单位，或是以字节（b）为单位

- 源操作数使用`si`，默认在`ds`中，允许段超越
- 目的操作数使用`di`。默认在`es`，不允许段超越
- 每次执行一次串操作，`si`和`di`将自动修改（以操作的单位）
    - 执行`cld`指令后，`df = 0`，地址指针增加

- 执行`std`指令后，`df = 1`，地址指针减少

movs

把字节或字操作数从内存的源地址送至目的地址

```
movsb   ; 字节 es:[di] <-- ds:[si]
movsw   ; 字 es:[di] <-- ds:[si]
```

stos

把`al`或`ax`数据传送至目的地址

```
stosb   ; 字节 es:[di] <-- al
stosw   ; 字 es:[di] <-- ax
```

lods

把指定内存单元的数据传送给`al`或`ax`

```
lodsb   ; 字节 al <-- ds:[si]
lodsw   ; 字 ax <-- ds:[si]
```

cmps

将内存中的源操作数减去至目的操作数，以便设置标志，进而比较两操作数之间的关系

```
cmpsb   ; 字节 ds:[si] - es:[di]
cmpsw   ; 字 ds:[di] - es:[di]
```

scas

将`al`减去至目的操作数，以便设置标志，进而比较`al/ax`与操作数之间的关系

```
scasb   ; 字节 al - es:[di]
scasw   ; 字 ax - es:[di]
```

rep

串操作指令执行一次，仅对数据串的一个字节或字进行操作。通过重复前缀指令可实现串操作的重复执行。

重复次数保存在`cx`中，每次执行`cx`减一

重复前缀分为两类：

- 配合不影响标志的`movs`、`stos`、`lods`指令的`rep`
  - `rep`仅当`cx != 0`的时候执行
- 配合影响标志的`cmps`、`scas`指令的`repz/repe`和`repnz/repne`
  - `repz/repe`在`cx != 0`且`zf = 1`时执行，即未到结尾且相等
  - `repnz/repne`在`cx != 0`且`zf = 0`时执行，即未到结尾且不相等

控制转移类指令

jmp

无条件跳转，程序转向标号指定的地址

```
    jmp label   ; 跳转到label标号处
    ; ...
label:  ; label标号
    ; ...
```

目标地址的寻址方式：

- 直接寻址
  - 转移地址像立即数一样，直接在指令的机器代码中
- 间接寻址
  - 转移地址在寄存器或内存单元中，通过寄存器或存储器的间接寻址访问

目标地址的范围：

> 在实际编程时，汇编程序会根据目标地址的距离，自动处理
>
> 可用操作符`near ptr`、`short`、`far ptr`强制修饰

1. 段内
   - 近转移`near ptr`
     - 在当前代码段64K范围内（±32K）
     - 不需要修改`cs`，只修改`ip`
   - 短转移`short`
     - 转移范围可以用一个字节表达，在段内-128 ~ +127范围的转移
2. 段间
   - 远转移`far ptr`
     - 从当前代码段跳到另一个代码段，可以在1MB范围
     - 需要修改`cs`和`ip`
     - 目标地址必须用一个32位表达式，叫做32位远指针（逻辑地址）

jcc

条件跳转，满足条件发生跳转，其中`cc`代表条件

```
    jcc label   ; 跳转到label标号处
    ; ...
label:  ; label标号
    ; ...
```

属于短转移，不影响标志位

具体指令为下表：

助记符	标志位	说明	助记符	标志位	说明
JZ/JE	ZF=1	等于零/相等	JC/JB/JNAE	CF=1	进位/低于/不高于等于
JNZ/JNE	ZF=0	不等于零/不相等	JNC/JNB/JAE	CF=0	无进位/不低于/高于等于
JS	SF=1	符号为负	JBE/JNA	CF=1 或 ZF=1	低于等于/不高于
JNS	SF=0	符号为正	JNBE/JA	CF=0 且 ZF=0	不低于等于/高于
JP/JPE	PF=1	“1” 的个数为偶	JL/JNGE	SF≠OF	小于/不大于等于
JNP/JPO	PF=0	“1” 的个数为奇	JNL/JGE	SF=OF	不小于/大于等于
JO	OF=1	溢出	JLE/JNG	ZF≠OF 或 ZF=1	小于等于/不大于
JNO	OF=0	无溢出	JNLE/JG	SF=OF 且 ZF=0	不小于等于/大于

loop

循环指令利用`cx`计数器自动减一，`label`操作数采用相对寻址方式

```
loop label  ; cx != 0 执行
loopz label ; cx != 0 且 zf = 1 执行
loopnz lable ; ; cx != 0 且 zf = 0 执行
```