ret 和 call

- 指令执行过程
  - 

通过栈中的数据来修改 **cs** 和 **ip** 同时还会 修改栈顶标志

ret（用栈中的数据）

- 弹栈

1. 近转移 ret 修改 IP ***pop ip***
   a. $(ip) = ((ss) * 16 + (sp))$
   b. $(sp) = (sp) + 2$
2. 远转移 retf 修改 cs:ip ***pop ip,pop cs***
   a. $(ip) = ((ss) * 16 + (sp))$
   b. $(sp) = (sp) + 2$
   c. $(cs) = ((ss) * 16 + (sp))$
   d. $(sp) = (sp) + 2$

call(不能实现短转移)

- 类似**jmp**
- call程序处理的数据一般要进行压栈

1.根据位移进行转移

```
push ip
jmp near ptr 标号
```

- 执行过程 原理
  - call下一条指令的IP压栈后，转到标号处

2.转移目的地址在指令中

```
call far ptr
```

- 执行过程 原理
  - call下一条指令的CS:IP压栈后，转到标号处

3.转移地址在寄存器中

```
call 16 位 reg
```

- 执行过程 原理
    - call下一条指令的IP压栈后，转到reg 处

4. 转移地址在内存中

1

```
call word ptr 内存单元地址
```

- 执行过程 原理
    - call下一条指令的IP压栈后，转到内存单元地址

2

```
call dword ptr 内存单元地址
```

- 执行过程 原理
    - call下一条指令的CS:IP压栈后，转到标号处

call 和 ret 共同应用

- 就像函数调用

批量数据处理

```
assume cs:code,ds:data,ss:stack

data segment
    db 'conversation'
data ends

stack segment
    db 16 dup(0)
stack ends

code segment
```

```
12
13
14      start:  mov ax,data
15              mov ds,ax
16              mov si,0
17              mov cx,12
18              call capital
19              mov ax,4c00h
20              int 21h
21
22      capital: and byte ptr ds:[si],11011111b
23               inc si;
24               loop capital
25               ret
26
27
28  code ends
29
30
31
32  end start
```

寄存器冲突问题

- 在子程序执行开头，把所需要用到的寄存器压栈
- 在子程序完成后，从栈中弹出各个寄存其的值

```
1   assume cs:code,ds:data,ss:stack
2
3   data segment
4       db 'word',0
5       db 'unix',0
6       db 'wind',0
7       db 'good',0
8   data ends
9
10  stack segment
11      db 128 dup(0)
12  stack ends
13
14  code segment
15
16
17      start:  mov ax,data
18              mov ds,ax
19
20              mov cx,4
21              mov bx,0
22
23      s:      mov di,bx
24              call capital
```

```
25              add bx,5
26              loop s
27
28              mov ax,4c00h
29              int 21h
30
31      capital: push cx;执行子程序前压栈
32               push si
33
34      change:  mov cl,ds:[si]
35               mov ch,0
36               jcxz ok
37               and byte ptr ds:[si],11011111b
38               inc si
39               jmp change
40
41           ok:  pop si;执行完后弹栈
42                pop cx
43                ret
44
45
46  code ends
47
48
49
50  end start
```

mul

1. 8位

> *一个默认放在AL，另一个放在内存字节单元或者8位reg。*
>
> *结果 默认 AX。*

2. 16位

> *一个默认放在AX，另一个放在内存字单元或者16位reg。*
>
> *结果 默认 高位在DX，低位在AX。*

模块化程序设计

- 通过ret,call.

参数和结果的传递

```
```

```
assume cs:code,ds:data,ss:stack

data segment
    dw 1,2,3,4,5,6,7,8
    dd 0,0,0,0,0,0,0,0
    db 'word',0
    db 'unix',0
    db 'wind',0
    db 'good',0
data ends

stack segment
    db 128 dup(0)
stack ends

code segment


    start:  mov ax,data
            mov ds,ax
            mov si,0
            mov bp,0
            call r_start


            mov ax,4c00h
            int 21h


    r_start:    mov bx,ds:[si]
                call cube
                mov ds:[16+bp],ax
                add si,2
                add bp,4
                loop r_start
                ret

        cube:   mov ax,bx
                mul bx
                mul bx
                ret

```