目 录

1 ret 和 retf **2**

2 call 指令 **2**

2.1 根据位移进行转移的 call 指令 2

2.2 转移的目的地址在指令中的 call 指令 3

2.3 转移的目的地址在寄存器中的 call 指令 3

2.4 转移的目的地址在内存中的 call 指令 3

3 call 和 ret 指令的配合使用 **3**

4 mul 指令 **4**

1 ret 和 retf

ret 指令用栈中的数据，修改 IP 的内容，从而实现近转移。
retf 指令用栈中的数据，修改 CS 和 IP 的内容，从而实现远转移。
CPU 执行 ret 指令时进行以下两个步骤：

```
(IP)=((ss)*16+(sp))
(sp)=(sp)+2
```

所以 ret 指令相当于 pop IP。
CPU 执行 retf 指令时进行以下四个步骤：

```
(IP)=((ss)*16+(sp))
(sp)=(sp)+2
(CS)=((ss)*16+(sp))
(sp)=(sp)+2
```

所以 retf 指令相当于 pop IP；pop CS。

2 call 指令

CPU 执行 call 指令时进行以下两个步骤：

```
将当前的IP或CS和IP压入栈中
转移
```

2.1 根据位移进行转移的 call 指令

“call 标号”的功能为：

```
(sp)=(sp)-2
((ss)*16+(sp))=(IP)
(IP)=(IP)+16位位移
```

其中 16 位位移 =“标号”处的地址-call 指令后的第一个字节的地址。
CPU 执行“call 标号”时，相当于执行：

```
push IP
jmp near ptr 标号
```

2.2 转移的目的地址在指令中的 call 指令

“call far ptr 标号”实现的是段间转移，相当于进行如下操作：

```
push CS
push IP
jmp far ptr 标号
```

2.3 转移的目的地址在寄存器中的 call 指令

“call 16 位寄存器”相当于进行如下操作：

```
push IP
jmp 16位寄存器
```

2.4 转移的目的地址在内存中的 call 指令

“call word ptr 内存单元地址”相当于进行如下操作：

```
push IP
jmp word ptr 内存单元地址
```

“call dword ptr 内存单元地址”相当于进行如下操作：

```
push CS
push IP
jmp dword ptr 内存单元地址
```

3 call 和 ret 指令的配合使用

call 和 ret 可以实现子程序的机制，子程序的框架如下：

```
assume cs:code
code segment
main:
    ; ...
    call sub1
    ; ...
    mov ax, 4c00h
    int 21h
sub1:
    ; ...
    call sub2
    ; ...
    ret
```

```
14      sub2:
15          ; ...
16          ret
17      code ends
18      end main
```

4 mul 指令

mul 指令是乘法指令，乘法指令的介绍如下：

- 两个相乘的数：如果两个相乘的数都是 8 位，那么一个默认放在 AH 中，另一个放在 8 位寄存器或内存字节单元中。如果两个都是 16 位，那么一个默认放在 AX 中，另一个放在 16 位寄存器或内存字单元中。
- 结果：如果是 8 位乘法，结果默认放在 AX 中。如果是 16 位乘法，结果的高位默认放在 DX 中，低位放在 AX 中。

mul 指令的格式如下：

```
1   mul reg
2   mul 内存单元
```