第九章 转移指令

基本介绍

可以修改IP，或同时修改CS和IP的指令被称为转移指令。概括的讲，转移指令就是可以控制CPU执行内存中某段代码的指令。

转移指令的分类

- 只修改IP时，被称为**段内转移**。如 `jmp ax`
- 同时修改CS和IP时，被称为**段间转移**。如`jmp 1000:0`

段内转移又分为一下的两类

- 短转移：IP的修改范围为**-128~127**(一个字节)
- 近转移：IP的修改范围为**-32768~32767**（一个字）

操作符offset

`offset`是由编译器识别的符号，他的功能是**取得标号的偏移地址**。

例如：

```
assume cs:code

code segment
start:
        mov ax, offset start
s:
        mov ax, offset s
code ends
end start
```

其中`mov ax, offset start`，因为start标号的偏移地址为`0`，所以该语句等价于`mov ax, 0`。同样的下面的标号s的偏移地址为3（`mov ax, 1`的指令占3个字节），因而`mov ax, offset s`等价于`mov ax, 3`

分析以下的程序：

```
assume cs:code

code segment
start:
        mov ax, bx ; 该指令占两个字节
        mov si, offset start
        mov di, offset s
        mov ax, cs:[si]
        mov cs:[di], ax
s:
        nop ; nop表示什么都不干，占一个字节
        nop
code ends
end start
```

通过上面的程序，我们将start开始的第一条指令复制到了s中。上面的程序也就相当于。

```
assume cs:code

code segment
start:
        mov ax, bx
s:
        mov ax, bx
code ends
end start
```

jmp指令

jmp为**无条件转移指令**，可以只修改IP，也可以同时修改CS和IP。

根据不同条件的转移，有不同的格式。

根据位移进行转移

```
jmp short 标号（转到标号处执行指令）
```

例：

```
assume cs:code

code segment
start:
        mov ax, 0
        jmp short s
        add ax, 1 ; 该指令被跳过
s:
        inc ax

        mov ax, 4c00h
        int 21h
code ends
end start
```

`jmp short xx`是短转移指令，该指令占用两个字节，在机器码中不包含任何地址，只包含IP的位移，范围为-128~127。

还有相应的近转移指令，`jmp near xx`，该指令占用三个字节，IP变化的范围为-32768~32767，

地址在指令中的jmp指令

使用`jmp short xx`或者`jmp near xx`在指令中并不包含，转移之后的地址，而是相对于当前的IP的转移位移。

`jmp far ptr 标号`实现的是段间转移，又被称为远转移。

```
assume cs:code

code segment
start:
        mov ax, 0
        mov bx, 0
        jmp far ptr s ;段间转移
        db 256 dup (0)
s:
        add ax, 1
        inc ax
code ends
end start
```

使用段间转移指令的时候，机器码中包含转移的目的地址的段地址和偏移地址，因而该转移指令占用5个字节。

转移地址在寄存器中

格式：**jmp 16位寄存器**

作用：**IP = 该寄存器**

转移地址在内存中

只修改IP，段内转移

格式： **jmp word ptr 内存单元地址**

作用： **IP = 该内存单元内容**

例:

```
mov ax, 0123h
mov ds:[0], ax
jmp word ptr ds:[0]

mov ax, 0123h
mov [bx], ax
jmp word ptr, [bx]
```

修改CS和IP，段间转移

格式：**jmp dword ptr 内存单元地址**

租用：**CS = 高地址的字， IP = 低地址的字**

例：

```
mov ax, 0123h
mov ds:[0], ax
mov word ptr ds:[2], 4567h
jmp dword ptr ds:[0]
```

此时用到的内存单元为`ds:[0] ~ ds:[3]`。高地址存放的字是`word ptr ds:[2]`，低地址存放的字为`word ptr ds:[0]`，因而执行完之后， cs = 4567h, ip = 0123h。

jcxz指令

`jcxz`为有条件转移指令。**所哟的条件转移指令都是短转移。**在对应的机器码中只包含IP转移的位移，没有目的地址。

格式：**jcxz 标号**

功能：**如果cx为0，则跳转到标号处执行，否则继续向下执行。**

使用C语言的方式，可以说`jcxz 标号`相当于`if (cx == 0) jump short 标号`。

loop指令

loop是循环指令，也是条件转移指令，之前已经介绍过。

使用C语言的方式，可以说`loop 标号`相当于`if (--cx != 0) jmp short 标号`。

程序分析

该程序能否正确返回？

```
assume cs:code

code segment
        mov ax, 4c00h ; 3个字节
        int 21h ; 2个字节
start:
        mov ax, 0 ; 3个字节
s:
        nop
        nop

        mov di, offset s
        mov si, offset s2
        mov ax, cs:[si]
        mov cs:[di], ax

s0:
        jmp short s

s1:
        mov ax, 0 ; 3个字节
        int 21h ; 2个字节
        mov ax, 0 ; 3个字节
s2:
        jmp short s1
```

```
        nop

code ends
end start
```

在不明白`jmp short 标号`的实际函数的情况下，很多人会讲程序翻译成这样。

```
assume cs:code

code segment
        mov ax, 4c00h
        int 21h
start:
        mov ax, 0
s:
        jmp short s1; 讲s2处的2字节指令复制到s处
        ...
        ...
start
code ends
end start
```

然后发现死循环了，无法返回。

但是这是错误的，`jmp short s1`其实和`s1`并没有绑定的关系，只是表示一个位移。从源代码可以看出，该`jmp short s1`表示的含义是`ip -= 10`，程序是可以正常返回的。

注：

CPU执行指令的过程为：

1. 从`cs:ip`处读取指令。
2. `ip`指向下一条指令
3. 执行指令
4. 回到1

在屏幕中显示字符

代码如下：

```
assume cs:code, ds:data, ss:stack
; 在显存中显示一个字符需要两个字节，其中奇数地址的字节表示要显示的字符的内容，偶数地址的字节表示要显示的字符的格式，如白底蓝字等，具体的显示的格式如下。
data segment
    db 'Welcome to MASM!' ; 要显示的字符数据
    ; 要显示的字符格式
    ; BL（闪烁） R G B(background) I R G B(foreground)
    db 10000010b ;  绿字
    db 10100100b ; 绿底红字
    db 11110001b ; 白底蓝字
data ends

stack segment stack
    db 128 dup (0)
stack ends

code segment

start:
    mov ax, stack
    mov ss, ax
    mov sp, 128

    mov ax, data
    mov ds, ax

    mov bx, 0b800h
    mov es, bx ; 设置开始显示的段地址

    mov si, 0
```

```
    mov di, 1600 + 80 ; 显示的位置
    mov bx, 16
    mov dx, 0


    mov cx, 3 ; 三种形式的字，三次循环
showMasm:
    push bx ; 保存寄存器
    push cx
    push si
    push di

    mov cx, 16
    mov dh, ds:[bx] ; dh 中存放显示的字符的格式，如白底蓝字


showRow:
    mov dl, ds:[si] ; dl 中存放显示的字符
    mov es:[di], dx ; 放入到显存中
    add di, 2 ; 显存中下一个字符
    inc si ; 要显示的下一个字符
    loop showRow

    pop di ;取出保存的寄存器内容
    pop si
    pop cx
    pop bx

    add di, 160 ; 显示下一列
    inc bx ; 显示下一种格式
    loop showMasm


    mov ax, 4c00h ; 结束程序
    int 21h
code ends
```

```
66 end start
```