汇编语言笔记(九)--jmp指令

2020-01-06 | 阅读：2626次

前言：

jmp指令 。

目录：

jmp指令：

jmp 是无条件转移指令，

无条件转移指令可转到内存中任何程序段，

转移地址可在指令中给出，也可以在寄存器中给出，或在储存器中指出。

它可以只修改 **IP** ，也可以同时修改 **CS** 和 **IP**

只修改IP的称为段内转移：`jmp ax` 相当于 `mov ax,ip`

同时修改 `cs:ip` 的叫 **段间转移**：`jmp 1000:0`

jmp 指令要给出两种信息：

1. 转移的目的地址
2. 转移的距离(段间转移、段内转移、段内近转移)

依据位移进行转移的jmp指令：

语法：`jmp short 标号` (转到标号处执行指令)

这种格式的 **jmp** 指令实现的是段内短转移

它对 **IP** 的修改范围为 `-128~127`

`short` 符号表示指令进行的是短转移

`标号` 指明了指令要转移的目的地

比如：

```
1   assume cs:codesg
2
3   codesg segment
4
5           start:mov ax,0
6                   jmp short s
7                   add ax,1
8           s:inc ax
9   codesg ends
10  end start
```

程序执行后，**ax** 中的值为 **1**

因为执行 `jmp short s` 后，越过了 `add ax,1`

IP 指向了 标号 **s** 处的 `inc ax`

也就是说，程序只进行了一次 **ax** 加 **1** 操作

此种转移方式并没有转移的目的地址，而是相对于当前 **IP** 的转移位移

另外，近转移ip修改范围： `-32768~32767`

转移的目的地址在指令中的jmp指令：

语法： `jmp far ptr 标号`

这种实现的是 **段间转移** ， 又称为远转移

(CS)=标号所在段的段地址； (IP)=标号在段中的偏移地址

far ptr 指明了指令用标号的段地址和偏移地址修改 **CS 和 IP**

比如：

```
1   assume cs:codesg
2
3   codesg segment
4
5           start:mov ax,0
6                 mov bx,0
7                 jmp far ptr s
8                 db 256 dup (0)
9           s:add ax,1
10                inc ax
11  codesg ends
12  end start
```

转移地址在寄存器中的jmp指令：

指令格式：`jmp 16 位 reg`

功能：`(IP)=(16 位 reg)`

比如：

```
1  jmp ax
2
3  指令执行前：  ax=1000H , CS=2000H , IP=0003H
4
5  指令执行后：  ax=1000H , CS=2000H , IP=1000H
```

`jmp ax`，相当于：`mov IP,ax`

转移地址在内存中的jmp指令：

转移地址在内存中的jmp指令有两种格式

(1) jmp word ptr 内存单元地址（段内转移）

功能：从内存单元地址处开始存放着一个字，是转移的目的偏移地址

内存单元地址可用寻址方式的任一格式给出。

比如：

```
1  mov ax,0123H
2
3  mov ds:[0],ax
4
5  jmp word ptr ds:[0]
```

执行后，(IP)=0123H

又比如：

```
1  mov ax,0123H
2
3  mov [bx],ax
4
5  jmp word ptr [bx]
```

执行后，(IP)=0123H

---

(2) jmp dword ptr 内存单元地址（段间转移）

功能：从内存单元地址处开始存放着两个字，

高地址处的字是转移的目的段地址，低地址处是转移的目的偏移地址

`(CS) = (内存单元地址+2)`

`(IP) = (内存单元地址)`

内存单元地址可用寻址方式的任一格式给出

比如：

```
1  mov ax,0123H
2
3  mov ds:[0],ax
4
5  mov word ptr ds:[2],0
6
7  jmp dword ptr ds:[0]
```

执行后，(CS)=0，(IP)=0123H，CS:IP 指向 0000:0123

再比如：

```
1  mov ax,0123H
2
3  mov [bx],ax
4
5  mov word ptr [bx+2],0
```

```
6
7  jmp dword ptr [bx]
```

执行后，(CS)=0，(IP)=0123H，CS:IP 指向 0000:0123

> 总结：

8086 转移指令分几类：

1. 无条件转移指令：如 jmp
2. 条件转移指令：如 jcxz
3. 循环指令：如 loop
4. 过程
5. 中断

`jmp short 标号` 功能为：段内短转移

(IP)=(IP)+8位位移

1. 8位位移 = 标号处的地址 - jmp指令后的第一个字节地址

2. short指明的此处是8位位移

3. 8位位移的范围为-128-127，用补码表示

4. 8位位移是编译程序时在编译时算出的

`jmp near ptr 标号` 功能为：段内近转移

(IP)=(IP)+16位位移

1. 16位位移 = 标号处的地址 - jmp指令后的第一个字节地址

2. short指明的此处是8位位移

3. 16位位移的范围为-32768-32767，用补码表示

4. 16位位移是编译程序时在编译时算出的

> 举例:

在x64汇编中，jmp指令用于无条件跳转。它的机器码取决于跳转的类型和距离。以下是一些例子：

短距离跳转（Short Jump）：

如果跳转目标与jmp指令的距离在-128到+127字节之间，那么jmp指令的机器码为EB xx，其中xx是一个有符号字节，表示跳转目标相对于下一条指令的偏移量。例如，如果我们有以下的汇编代码：

jmp short label

...

label:

假设label位于jmp指令后的10字节处，那么对应的机器码将是EB 0A。

近距离跳转（Near Jump）：

如果跳转目标与jmp指令的距离超过了短距离跳转的范围，那么jmp指令的机器码为E9 xxxx xxxx xxxx xxxx，其中xxxx xxxx xxxx xxxx是一个有符号的32位值，表示跳转目标相对于下一条指令的偏移量。

例如，如果我们有以下的汇编代码：

jmp near label

...

label:

假设label位于jmp指令后的1000字节处，那么对应的机器码将是E9 E8 03 00 00。

请注意，这些都是相对跳转，也就是说，跳转的目标地址是相对于当前指令的位置计算的。另外，这里的机器码都是以小端字节序表示的，也就是说，最低有效字节在前，最高有效字节在后。在实际的编程中，你可能需要使用反汇编工具来查看具体的机器码。