第3节 跳转指令

1. jmp指令（无条件转移）

指令格式	示例
jmp 标号	- 段间转移（远转移）：`jmp far ptr 标号`
	- 段内短转移：`jmp short 标号`（8位的位移）
	- 段内近转移：`jmp near ptr 标号`（16位的位移）
jmp 寄存器	- `jmp bx`（等价于段内近转移，也是16位的位移）
jmp 内存单元	- `jmp word ptr 内存单元地址`（段内转移）
	- `jmp dword ptr 内存单元地址`（段间转移）

备注：在源程序中，不能直接使用"jmp 2000:0100"这样的转移指令来实现段间转移，这种方式在debug模式中使用的汇编指令，在源程序中写，编译器并不识别（编译报错）

也就是说你要跳转的地方可能是标号所在位置，或者寄存器中地址所指向的位置，或者内存单元中所指向的位置

本节简单起见，只使用jmp标号来进行跳转，

jmp s 默认是段内转移到s，如果要实现段间转移，需要使用 jmp far ptr s，无需指定s的段，编译器自动获取

实现原理：

对于段内转移，编译器会计算你要跳转的位置与当前的位置的差值，也就是偏移量，这个偏移量对于jmp short来说范围是-128-127，对于jmp near ptr来说范围是-32768-32767（向上32kb，向下32kb-1b），然后让IP加上这个偏移量就实现了跳转。

对于段间转移，编译器会同时得到段地址和偏移地址，送到cpu中同时修改cs和ip地址达到跳转的目的。

不过写程序的时候，使用jmp 标号可读性非常高，不需要繁琐的计算，繁琐的计算由编译器完成。

2. call与ret指令

call的用法和jmp几乎一样，也是上面3种，不过同样我们这里只讲call 标号这种段内转移。

与jmp不同的是，call跳转之前会把下一条指令的地址压入到栈顶，即执行了一步push ip，当执行ret指令时，

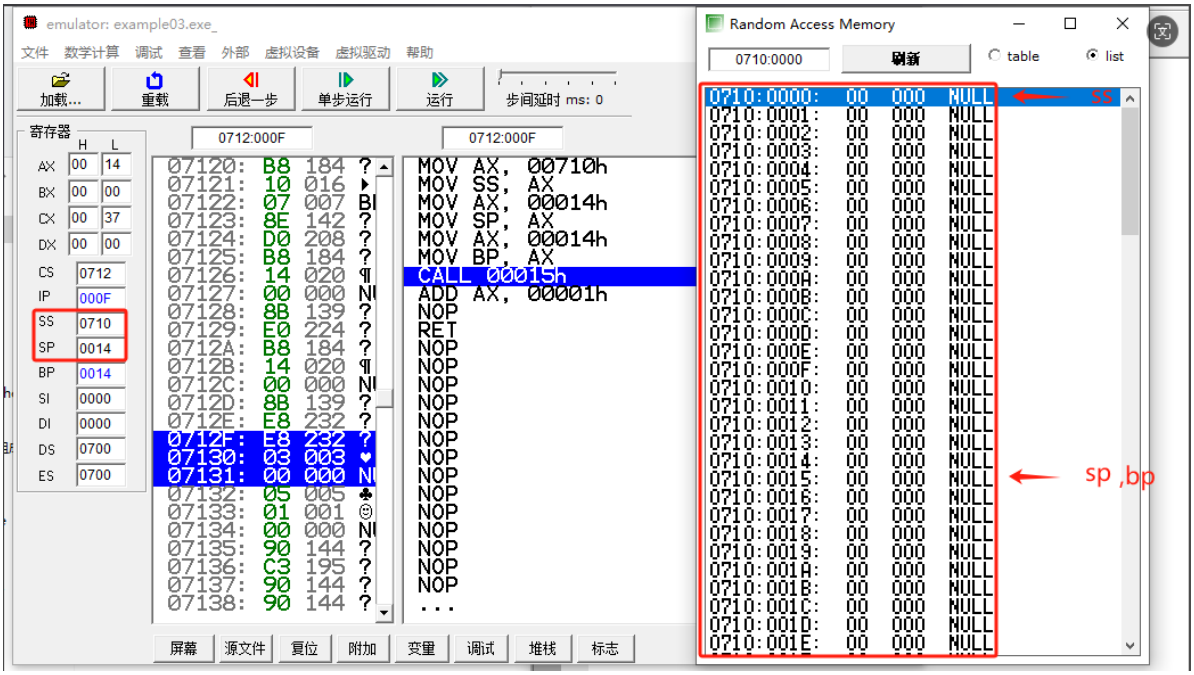
会自动把栈顶的ip弹出到指令寄存器IP，从而实现返回到记录的地址，下面我们使用jmp和ret模仿上述过程的实现

example03.asm

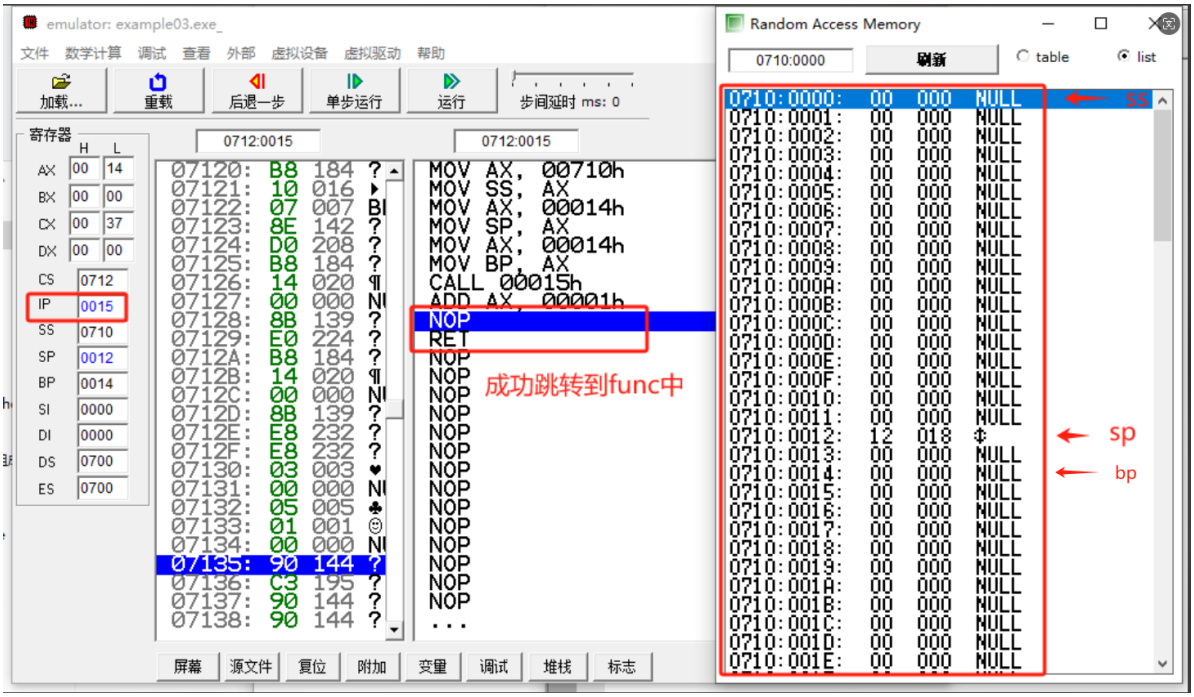
```
assume cs:code,ss:stack

stack segment
    db 20 dup(0)      ; 预留的栈空间
```

```
stack ends

code segment
main:
    mov ax,stack       ; 初始化栈寄存器
    mov ss,ax

    mov ax,20          ; 初始化栈顶
    mov sp,ax

    mov ax,20          ; 初始化栈底
    mov bp,ax

    call func
    add ax,1           ; 程序返回的地方

func:
    nop                ; 啥也不做
    ret

end main
code ends
```

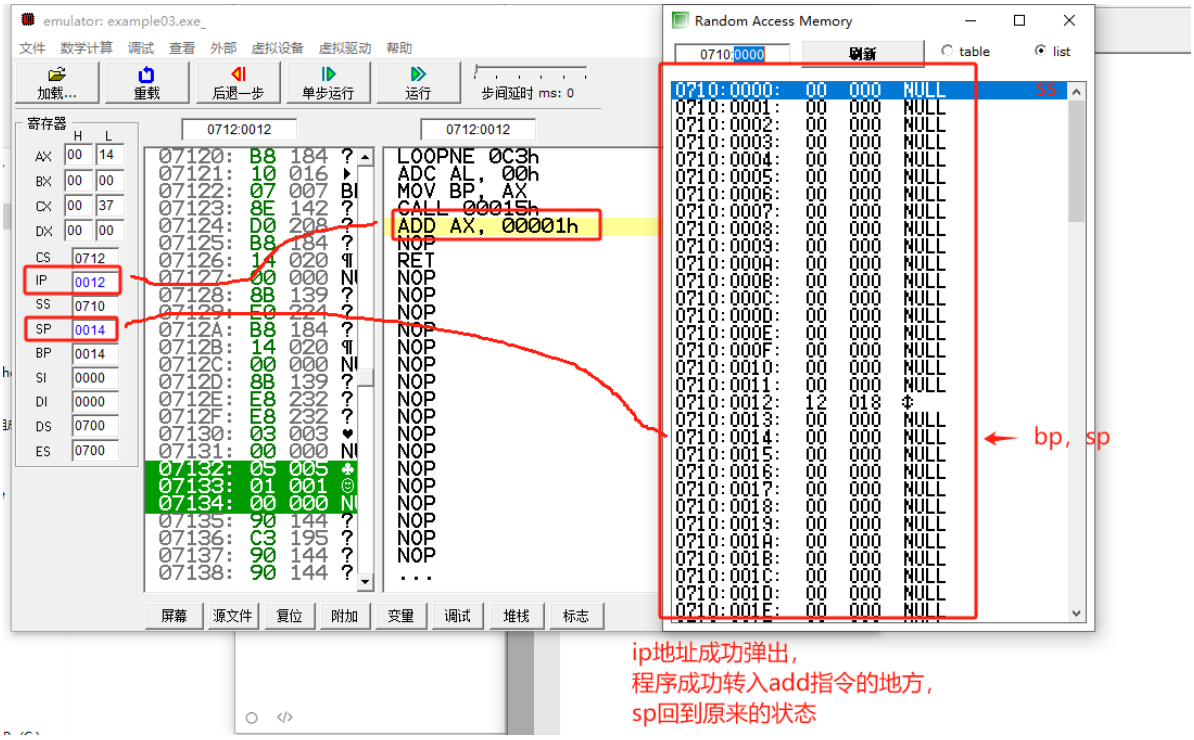
执行call之前栈的状态



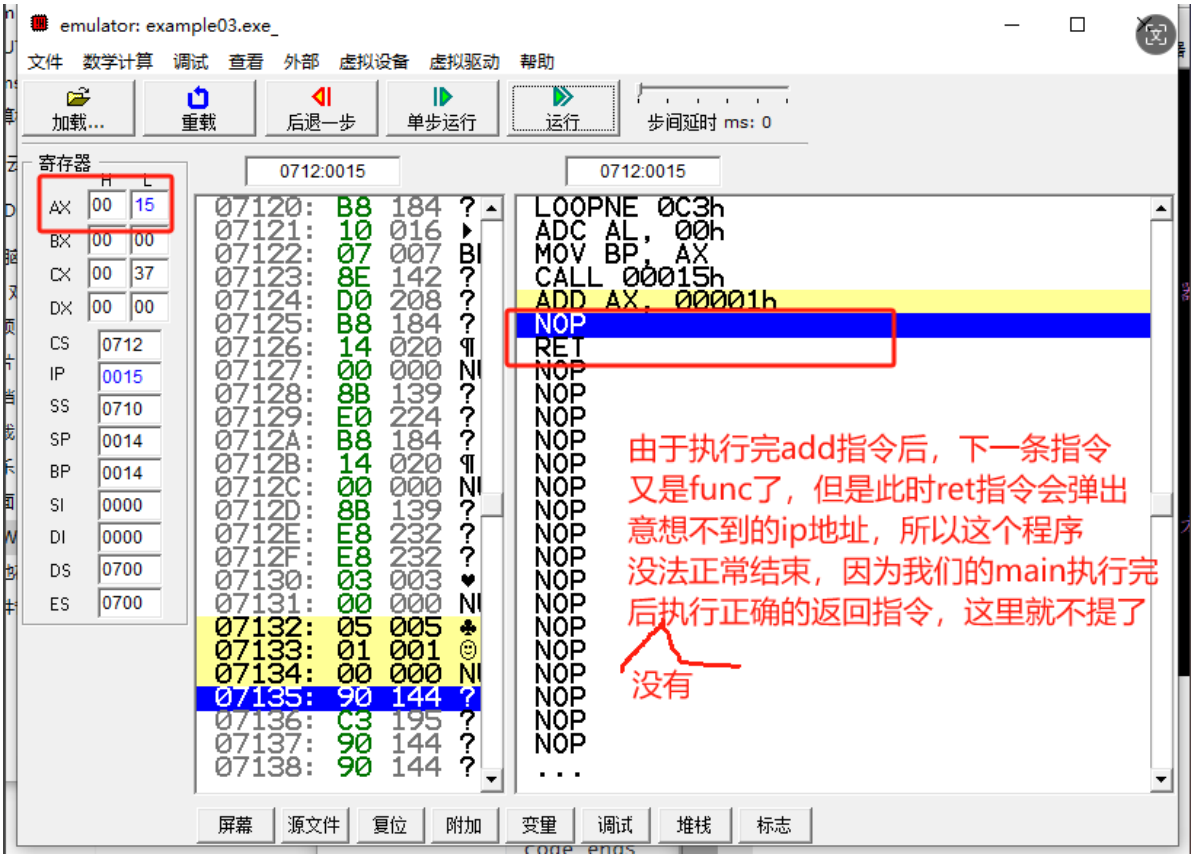
刚执行call时栈的状态



执行ret后的状态



程序执行完add之后



3. flags寄存器与运算指令

mul指令

格式：
mul 寄存器
mul 内存单元
默认将al或ax的内容作为被乘数，寄存器和内存单元的内容作为乘数，结果存放在ax或者dx和ax中

比如：

1.mul bl（8位乘法）
结果 ax = al*bl

2.mul bx（16位乘法）
结果 concat（dx，ax）= ax*bx
dx用于存放高8位，ax用于存放低8位

3.mul byte ptr [si]
4.mul word ptr [si]
注意，直接操作内存单元时，如果没有寄存器参与，需要说明操作的字节数

标志寄存器

flags寄存器也是16位，只有个别的位用于存放标志，用于存放运算指令的一些执行结果，非运算的指令不影响flags寄存器

OF：
将运算看成有符号运算，然后看运算是否溢出，比如16位的有符号数范围是-32768-32767，如果运算结果不在这个范围，OF=1

CF：
将运算看成无符号运算，然后看运算时最高位是否进位或者借位了。
比如对于8位数做加法，如果运算结果超出8位了，说明进位了。如果做8位减法，最高位不够减了，需要向不存在的第9位借位

ZF：
看运算结果是否为0

SF：
看运算结果的符号

其他的暂时不作了解，遇到时再看。

既然知道了各个标志位的含义，那么如何使用这些标志位的结果呢，这就是接下来需要介绍的各种条件转移指令

cmp**指令和条件转移指令**

cmp ax,bx

用于计算ax-bx，但是运算结果不保存在任何一个寄存器中，只保留flags寄存器的运算结果。
之所以不直接使用减法指令比较两个数的大小，就是因为会影响ax寄存器的结果，而我们并不是真的需要做减法，只需要比较大小。

8086处理器中的条件跳转指令包括：

1. `je` - 如果等于（ZF=1）
2. `jne` - 如果不等于（ZF=0）
3. `jz` - 如果为零（ZF=1）
4. `jnz` - 如果不为零（ZF=0）
5. `js` - 如果为负（SF=1）
6. `jns` - 如果不为负（SF=0）
7. `jo` - 如果溢出（OF=1）
8. `jno` - 如果不溢出（OF=0）
9. `jc` - 如果进位（CF=1）
10. `jnc` - 如果没有进位（CF=0）
11. `ja` / `jnbe` - 如果无符号大于（CF=0 且 ZF=0）
12. `jae` / `jnb` - 如果无符号大于等于（CF=0）
13. `jb` / `jnae` - 如果无符号小于（CF=1）
14. `jbe` / `jna` - 如果无符号小于等于（CF=1 或 ZF=1）
15. `jl` / `jnge` - 如果有符号小于（SF ≠ OF）
16. `jge` / `jnl` - 如果有符号大于等于（SF = OF）
17. `jle` / `jng` - 如果有符号小于等于（ZF = 1 或 SF ≠ OF）
18. `jg` / `jnle` - 如果有符号大于（ZF = 0 且 SF = OF）

这些条件跳转指令根据标志寄存器（如零标志位ZF，符号标志位SF，溢出标志位OF，进位标志位CF等）的状态来决定是否跳转到目标地址。