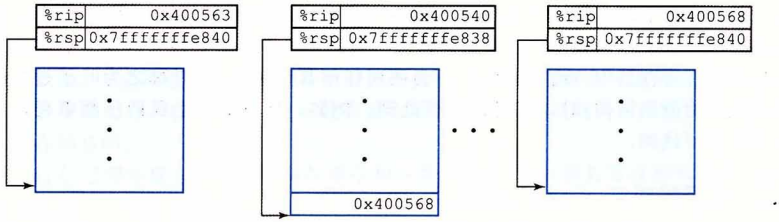
```
     Call to multstore from main
5     400563:  e8 d8 ff ff ff          callq  400540 <multstore>
6     400568:  48 8b 54 24 08          mov    0x8(%rsp),%rdx
```

在这段代码中我们可以看到，在 main 函数中，地址为 0x400563 的 call 指令调用函数 multstore。此时的状态如图 3-26a 所示，指明了栈指针%rsp 和程序计数器%rip 的值。call 的效果是将返回地址 0x400568 压入栈中，并跳到函数 multstore 的第一条指令，地址为 0x0400540(图 3-26b)。函数 multstore 继续执行，直到遇到地址 0x40054d 处的 ret 指令。这条指令从栈中弹出值 0x400568，然后跳转到这个地址，就在 call 指令之后，继续 main 函数的执行。




a）执行call　　b）call执行之后　　c）ret执行之后

图 3-26　call 和 ret 函数的说明。call 指令将控制转移到一个函数的起始，而 ret 指令返回到这次调用后面的那条指令

再来看一个更详细说明在过程间传递控制的例子，图 3-27a 给出了两个函数 top 和 leaf 的反汇编代码，以及 main 函数中调用 top 处的代码。每条指令都以标号标出：L1～L2(leaf 中)，T1～T4(main 中)和 M1～M2(main 中)。该图的 b 部分给出了这段代码执

```
     Disassembly of leaf(long y)
     y in %rdi
1    0000000000400540 <leaf>:
2     400540:  48 8d 47 02             lea    0x2(%rdi),%rax  L1: y+2
3     400544:  c3                      retq                   L2: Return

4    0000000000400545 <top>:
     Disassembly of top(long x)
     x in %rdi
5     400545:  48 83 ef 05             sub    $0x5,%rdi       T1: x-5
6     400549:  e8 f2 ff ff ff          callq  400540 <leaf>   T2: Call leaf(x-5)
7     40054e:  48 01 c0                add    %rax,%rax       T3: Double result
8     400551:  c3                      retq                   T4: Return

     . . .
      Call to top from function main
9     40055b:  e8 e5 ff ff ff          callq  400545 <top>    M1: Call top(100)
10    400560:  48 89 c2                mov    %rax,%rdx       M2: Resume
```

a）说明过程调用和返回的反汇编代码

图 3-27　包含过程调用和返回的程序的执行细节。使用栈来存储返回地址使得能够返回到过程中正确的位置