设计这个程序是为了出错(例如如果实际上执行了 ret 指令)时，程序会执行一条额外的 irmovq 指令，然后停止。因此，流水线中的错误会导致某个寄存器更新错误。这段代码说明实现测试程序需要非常小心。它必须建立起可能的错误条件，然后再探测是否有错误发生。

4.38 设计下面这个测试程序用来建立控制组合 B(图 4-67)。模拟器会发现流水线寄存器的气泡和暂停控制信号都设置成 0 的情况，因此我们的测试程序只需要建立它需要发现的组合情况。最大的挑战在于当处理正确时，程序要做正确的事情。

```
1   # Test instruction that modifies %esp followed by ret
2           irmovq mem,%rbx
3           mrmovq  0(%rbx),%rsp # Sets %rsp to point to return point
4           ret                  # Returns to return point
5           halt                 #
6   rtnpt:  irmovq $5,%rsi       # Return point
7           halt
8   .pos 0x40
9   mem:    .quad stack          # Holds desired stack pointer
10  .pos 0x50
11  stack:  .quad rtnpt          # Top of stack: Holds return point
```

这个程序使用了内存中两个初始化了的字。第一个字(mem)保存着第二个字(stack——期望的栈指针)的地址。第二个字保存着 ret 指令期望的返回点的地址。这个程序将栈指针加载到 %rsp，并执行 ret 指令。

4.39 从图 4-66 我们可以看到，由于加载/使用冒险，流水线寄存器 D 必须暂停。

```
bool D_stall =
        # Conditions for a load/use hazard
        E_icode in { IMRMOVQ, IPOPQ } &&
         E_dstM in { d_srcA, d_srcB };
```

4.40 从图 4-66 中可以看到，由于加载/使用冒险，或者由于分支预测错误，流水线寄存器 E 必须设置成气泡：

```
bool E_bubble =
        # Mispredicted branch
        (E_icode == IJXX && !e_Cnd) ||
        # Conditions for a load/use hazard
        E_icode in { IMRMOVQ, IPOPQ } &&
         E_dstM in { d_srcA, d_srcB};
```

4.41 这个控制需要检查正在执行的指令的代码，还需要检查流水线中更后面阶段中的异常。

```
## Should the condition codes be updated?
bool set_cc = E_icode == IOPQ &&
       # State changes only during normal operation
       !m_stat in { SADR, SINS, SHLT } && !W_stat in { SADR, SINS, SHLT };
```

4.42 在下一个周期向访存阶段插入气泡需要检查当前周期中访存或者写回阶段中是否有异常。

```
# Start injecting bubbles as soon as exception passes through memory stage
bool M_bubble = m_stat in { SADR, SINS, SHLT } || W_stat in { SADR, SINS, SHLT };
```

对于暂停写回阶段，只用检查这个阶段中的指令的状态。如果当访存阶段中有异常指令时我们也暂停了，那么这条指令就不能进入写回阶段。

```
bool W_stall = W_stat in { SADR, SINS, SHLT };
```

4.43 此时，预测错误的频率是 0.35，得到 $mp=0.20\times0.35\times2=0.14$，而整个 CPI 为 1.25。看上去收获非常小，但是如果实现新的分支预测策略的成本不是很高的话，这样做还是值得的。

4.44 在这个简化的分析中，我们把注意力放在了内循环上，这是估计程序性能的一种很有用的方法。只要数组足够大，花在代码其他部分的时间可以忽略不计。

A. 使用条件转移的代码的内循环有 9 条指令，当数组元素是 0 或者为负时，这些指令都要执行，