```
1       irmovq $5, %rdx
2       irmovq $0x100,%rsp
3       rmmovq %rdx,0(%rsp)
4       popq %rsp
5       nop
6       nop
7       rrmovq %rsp,%rax
```

两个 nop 指令会导致当 rrmovq 指令在译码阶段中时，popq 指令处于写回阶段。如果给予处于写回阶段中的两个转发源错误的优先级，那么寄存器%rax 会设置成增加了的程序计数器，而不是从内存中读出的值。

4.34　这个逻辑只需要检查 5 个转发源：

```
word d_valB = [
        d_srcB == e_dstE : e_valE;    # Forward valE from execute
        d_srcB == M_dstM : m_valM;    # Forward valM from memory
        d_srcB == M_dstE : M_valE;    # Forward valE from memory
        d_srcB == W_dstM : W_valM;    # Forward valM from write back
        d_srcB == W_dstE : W_valE;    # Forward valE from write back
        1 : d_rvalB;  # Use value read from register file
];
```

4.35　这个改变不会处理条件传送不满足条件的情况，因此将 dstE 设置为 RNONE。即使条件传送并没有发生，结果值还是会被转发到下一条指令。

```
1           irmovq $0x123,%rax
2           irmovq $0x321,%rdx
3           xorq %rcx,%rcx          # CC = 100
4           cmovne  %rax,%rdx       # Not transferred
5           addq %rdx,%rdx          # Should be 0x642
6           halt
```

这段代码将寄存器%rdx 初始化为 0x321。条件数据传送没有发生，所以最后的 addq 指令应该把%rdx 中的值翻倍，得到 0x642。不过，在修改过的版本中，条件传送源值 0x123 被转发到 ALU 的输入 valA，而 valB 正确地得到了操作数值 0x321。两个输入加起来就得到结果 0x444。

4.36　这段代码完成了对这条指令的状态码的计算。

```
## Update the status
word m_stat = [
        dmem_error : SADR;
        1 : M_stat;
];
```

4.37　设计下面这个测试程序来建立控制组合 A(图 4-67)，并探测是否出了错：

```
1   # Code to generate a combination of not-taken branch and ret
2           irmovq Stack, %rsp
3           irmovq rtnp,%rax
4           pushq %rax          # Set up return pointer
5           xorq %rax,%rax      # Set Z condition code
6           jne target          # Not taken (First part of combination)
7           irmovq $1,%rax      # Should execute this
8           halt
9   target: ret                 # Second part of combination
10          irmovq $2,%rbx      # Should not execute this
11          halt
12  rtnp:   irmovq $3,%rdx      # Should not execute this
13          halt
14  .pos 0x40
15  Stack:
```