对这些多路复用器的控制是由 dstE 确定的——当它表明有某个寄存器时，就选择端口 E 的值，否则就选择端口 M 的值。

在模拟模型中，我们可以禁止寄存器端口 M，如下面这段 HCL 代码所示：

```
## Disable register port M
## Set M port register ID
word w_dstM = RNONE;

## Set M port value
word w_valM = 0;
```

接下来的问题就是要设计处理 popq 的方法。一种方法是用控制逻辑动态地处理指令 popq rA，使之与下面两条指令序列有一样的效果：

```
iaddq  $8, %rsp
mrmovq -8(%rsp), rA
```

(关于指令 iaddq 的描述，请参考练习题 4.3)要注意两条指令的顺序，以保证 popq %rsp 能正确工作。要达到这个目的，可以让译码阶段的逻辑对上面列出的 popq 指令和 addq 指令一视同仁，除了它会预测下一个 PC 与当前 PC 相等以外。在下一个周期，再次取出了 popq 指令，但是指令代码变成了特殊的值 IPOP2。它会被当作一条特殊的指令来处理，行为与上面列出的 mrmovq 指令一样。

文件 pipe-lw.hcl 包含上面讲的修改过的写端口逻辑。它将常数 IPOP2 声明为十六进制值 E。还包括信号 f_icode 的定义，它产生流水线寄存器 D 的 icode 字段。可以修改这个定义，使得当第二次取出 popq 指令时，插入指令代码 IPOP2。这个 HCL 文件还包含信号 f_pc 的声明，也就是标号为“Select PC”的块(图 4-57)在取指阶段产生的程序计数器的值。

修改该文件中的控制逻辑，使之按照我们描述的方式来处理 popq 指令。可以参考实验资料获得如何为你的解答生成模拟器以及如何测试模拟器的指导。

••4.59　比较三个版本的冒泡排序的性能(家庭作业 4.47、4.48 和 4.49)。解释为什么一个版本的性能比其他两个的好。

练习题答案

4.1　手工对指令编码是非常乏味的，但是它将巩固你对汇编器将汇编代码变成字节序列的理解。在下面这段 Y86-64 汇编器的输出中，每一行都给出了一个地址和一个从该地址开始的字节序列：

```
1   0x100:                      | .pos 0x100  # Start code at address
0x100
2   0x100: 30f30f00000000000000 |     irmovq $15,%rbx
3   0x10a: 2031                 |     rrmovq %rbx,%rcx
4   0x10c:                      | loop:
5   0x10c: 4013fdffffffffffffff |     rmmovq %rcx,-3(%rbx)
6   0x116: 6031                 |     addq   %rbx,%rcx
7   0x118: 700c01000000000000   |     jmp loop
```

这段编码有些地方值得注意：

- 十进制的 15(第 2 行)的十六进制表示为 0x000000000000000f。以反向顺序来写就是 0f 00 00 00 00 00 00 00。
- 十进制－3(第 5 行)的十六进制表示为 0xfffffffffffffffd。以反向顺序来写就 fd ff ff ff ff ff ff ff。
- 代码从地址 0x100 开始。第一条指令需要 10 个字节，而第二条需要 2 个字节。因此，循环的目标地址为 0x0000010c。以反向顺序来写就是 0c 01 00 00 00 00 00 00。

4.2　手工对一个字节序列进行译码能帮助你理解处理器面临的任务。它必须读入字节序列，并确定要执行什么指令。接下来，我们给出的是用来产生每个字节序列的汇编代码。在汇编代码的左边，你可以看到每条指令的地址和字节序列。