


条件	流水线寄存器				
	F	D	E	M	W
处理 ret	暂停	气泡	正常	正常	正常
预测错误的分支	正常	气泡	气泡	正常	正常
组合	暂停	气泡	气泡	正常	正常

也就是说,组合情况 A 的处理与预测错误的分支相似,只不过在取指阶段是暂停。幸运的是,在下一个周期,PC 选择逻辑会选择跳转后面那条指令的地址,而不是预测的程序计数器值,所以流水线寄存器 F 发生了什么是没有关系的。因此我们得出结论,流水线能正确处理这种组合情况。

组合 B 包括一个加载/使用冒险,其中加载指令设置寄存器 %rsp,然后 ret 指令用这个寄存器作为源操作数,因为它必须从栈中弹出返回地址。流水线控制逻辑应该将 ret 指令阻塞在译码阶段。

 **练习题 4.38** 写一个 Y86-64 汇编语言程序,它能导致出现组合 B 的情况,如果流水线运行正确,以 halt 指令结束。

合并组合 B 条件的控制动作(图 4-66),我们得到以下流水线控制动作:

条件	流水线寄存器				
	F	D	E	M	W
处理 ret	暂停	气泡	正常	正常	正常
预测错误的分支	暂停	暂停	气泡	正常	正常
组合	暂停	气泡+暂停	气泡	正常	正常
期望的情况	暂停	暂停	气泡	正常	正常

如果同时触发两组动作,控制逻辑会试图暂停 ret 指令来避免加载/使用冒险,同时又会因为 ret 指令而往译码阶段中插入一个气泡。显然,我们不希望流水线同时执行这两组动作。相反,我们希望它只采取针对加载/使用冒险的动作。处理 ret 指令的动作应该推迟一个周期。

这些分析表明组合 B 需要特殊处理。实际上,PIPE 控制逻辑原来的实现并没有正确处理这种组合情况。即使设计已经通过了许多模拟测试,它还是有细节问题,只有通过刚才那样的分析才能发现。当执行一个含有组合 B 的程序时,控制逻辑会将流水线寄存器 D 的气泡和暂停信号都置为 1。这个例子表明了系统分析的重要性。只运行正常的程序是很难发现这个问题的。如果没有发现这个问题,流水线就不能忠实地实现 ISA 的行为。

5. 控制逻辑实现

图 4-68 是流水线控制逻辑的整体结构。根据来自流水线寄存器和流水线阶段的信号,控制逻辑产生流水线寄存器的暂停和气泡控制信号,同时也决定是否要更新条件码寄存器。我们可以将图 4-64 的发现条件和图 4-66 的动作结合起来,产生各个流水线控制信号的 HCL 描述。

遇到加载/使用冒险或 ret 指令,流水线寄存器 F 必须暂停:

```
bool F_stall =
    # Conditions for a load/use hazard
    E_icode in { IMRMVQ, IPO PQ } &&
    E_dstM in { d_srcA, d_srcB } ||
    # Stalling at fetch while ret passes through pipeline
    IRET in { D_icode, E_icode, M_icode };
```