序列出的，而不是按照它们出现在程序中的顺序。因为预测跳转指令会选择分支，所以周期 3 中会取出位于跳转目标处的指令，而周期 4 中会取出该指令后的那条指令。在周期 4，分支逻辑发现不应该选择分支之前，已经取出了两条指令，它们不应该继续执行下去了。幸运的是，这两条指令都没有导致程序员可见的状态发生改变。只有到指令到达执行阶段时才会发生那种情况，在执行阶段中，指令会改变条件码。我们只要在下一个周期往译码和执行阶段中插入气泡，并同时取出跳转指令后面的指令，这样就能取消(有时也称为指令排除(instruction squashing))那两条预测错误的指令。这样一来，两条预测错误的指令就会简单地从流水线中消失，因此不会对程序员可见的状态产生影响。唯一的缺点是两个时钟周期的指令处理能力被浪费了。

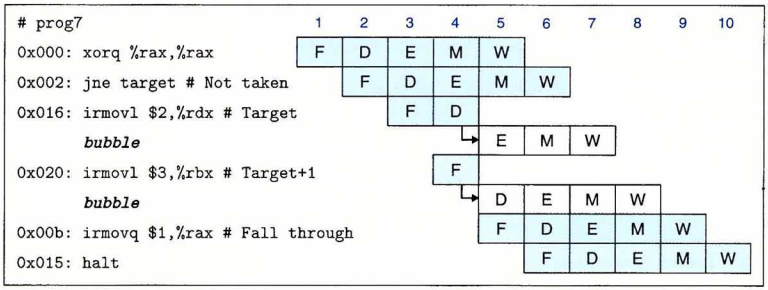

图 4-56　处理预测错误的分支指令。流水线预测会选择分支，所以开始取跳转目标处的指令。在周期 4 发现预测错误之前，已经取出了两条指令，此时，跳转指令正在通过执行阶段。在周期 5 中，流水线往译码和执行阶段中插入气泡，取消了两条目标指令，同时还取出跳转后面的那条指令

对控制冒险的讨论表明，通过慎重考虑流水线的控制逻辑，控制冒险是可以被处理的。在出现特殊情况时，暂停和往流水线中插入气泡的技术可以动态调整流水线的流程。如同我们将在 4.5.8 节中讨论的一样，对基本时钟寄存器设计的简单扩展就可以让我们暂停流水段，并向作为流水线控制逻辑一部分的流水线寄存器中插入气泡。

4.5.6　异常处理

正如第 8 章中将讨论的，处理器中很多事情都会导致异常控制流，此时，程序执行的正常流程被破坏掉。异常可以由程序执行从内部产生，也可以由某个外部信号从外部产生。我们的指令集体系结构包括三种不同的内部产生的异常：1)halt 指令，2)有非法指令和功能码组合的指令，3)取指或数据读写试图访问一个非法地址。一个更完整的处理器设计应该也能处理外部异常，例如当处理器收到一个网络接口收到新包的信号，或是一个用户点击鼠标按钮的信号。正确处理异常是任何微处理器设计中很有挑战性的一方面。异常可能出现在不可预测的时间，需要明确地中断通过处理器流水线的指令流。我们对这三种内部异常的处理只是让你对正确发现和处理异常的真实复杂性略有了解。

我们把导致异常的指令称为异常指令(excepting instruction)。在使用非法指令地址的情况中，没有实际的异常指令，但是想象在非法地址处有一种“虚拟指令”会有所帮助。在简化的 ISA 模型中，我们希望当处理器遇到异常时，会停止，设置适当的状态码，如图 4-5 所示。看上去应该是到异常指令之前的所有指令都已经完成，而其后的指令都不应该对程序员可见的状态产生任何影响。在一个更完整的设计中，处理器会继续调用异常处理