周期，导致取出 rrmovq 指令，但是在译码阶段就被替换成了气泡。这个过程在图 4-62 中的表示为，3 个取指用箭头指向下面的气泡，气泡会经过剩下的流水线阶段。最后，在周期 7 取出 irmovq 指令。比较图 4-62 和图 4-55，可以看到，我们的实现达到了期望的效果，只不过连续 3 个周期取出了不正确的指令。

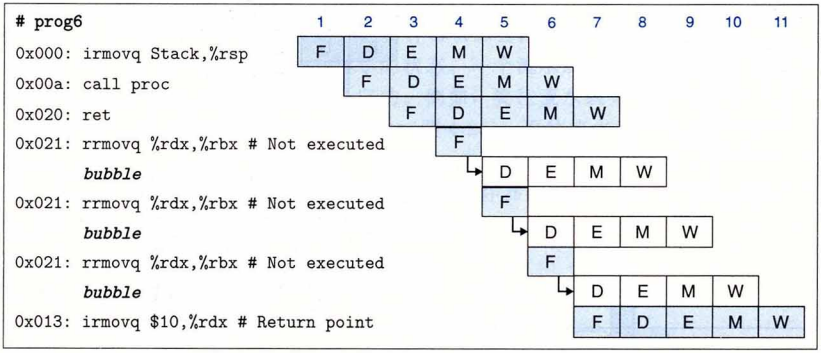

图 4-62　ret 指令的详细处理过程。取指阶段反复取出 ret 指令后面的 rrmovq 指令，但是流水线控制逻辑在译码阶段中插入气泡，而不是让 rrmovq 指令继续下去。由此得到的行为与图 4-55 所示的等价

当分支预测错误发生时，我们已经在 4.5.5 节中描述了所需的流水线操作，并用图 4-56 进行了说明。当跳转指令到达执行阶段时就可以检测到预测错误。然后在下一个时钟周期，控制逻辑就会在译码和执行段插入气泡，取消两条不正确的已取指令。在同一个时钟周期，流水线将正确的指令读取到取指阶段。

对于导致异常的指令，我们必须使流水线化的实现符合期望的 ISA 行为，也就是在前面所有的指令结束前，后面的指令不能影响程序的状态。一些因素会使得想达到这些效果比较麻烦：1）异常在程序执行的两个不同阶段(取指和访存)被发现的，2)程序状态在三个不同阶段(执行、访存和写回)被更新。

在我们的阶段设计中，每个流水线寄存器中会包含一个状态码 stat，随着每条指令经过流水线阶段，它会记录指令的状态。当异常发生时，我们将这个信息作为指令状态的一部分记录下来，并且继续取指、译码和执行指令，就好像什么都没有出错似的。当异常指令到达访存阶段时，我们会采取措施防止后面的指令修改程序员可见的状态：1）禁止执行阶段中的指令设置条件码，2）向内存阶段中插入气泡，以禁止向数据内存中写入，3）当写回阶段中有异常指令时，暂停写回阶段，因而暂停了流水线。

图 4-63 中的流水线图说明了我们的流水线控制如何处理导致异常的指令后面跟着一条会改变条件码的指令的情况。在周期 6，pushq 指令到达访存阶段，产生一个内存错误。在同一个周期，执行阶段中的 addq 指令产生新的条件码的值。当访存或者写回阶段中有异常指令时(通过检查信号 m_stat 和 W_stat，然后将信号 set_cc 设置为 0)，禁止设置条件码。在图 4-63 的例子中，我们还可以看到既向访存阶段插入了气泡，也在写回阶段暂停了异常指令——pushq 指令在写回阶段保持暂停，后面的指令都没有通过执行阶段。

对状态信号流水线化，控制条件码的设置，以及控制流水线阶段——将这些结合起来，我们实现了对异常的期望的行为：异常指令之前的指令都完成了，而后面的指令对程