寄存器 M 的不选择分支的指令来说是 valP 的值(存储在寄存器 M_valA 中)，或是当 ret 指令到达流水线寄存器 W(存储在 W_valM)时的返回地址的值。

4.5.5　流水线冒险

PIPE－结构是创建一个流水线化的 Y86-64 处理器的好开端。不过，回忆 4.4.4 节中的讨论，将流水线技术引入一个带反馈的系统，当相邻指令间存在相关时会导致出现问题。在完成我们的设计之前，必须解决这个问题。这些相关有两种形式：1)数据相关，下一条指令会用到这一条指令计算出的结果；2)控制相关，一条指令要确定下一条指令的位置，例如在执行跳转、调用或返回指令时。这些相关可能会导致流水线产生计算错误，称为冒险(hazard)。同相关一样，冒险也可以分为两类：数据冒险(data hazard)和控制冒险(control hazard)。我们首先关心的是数据冒险，然后再考虑控制冒险。

图 4-43 描述的是 PIPE－处理器处理 prog1 指令序列的情况。假设在这个例子以及后面的例子中，程序寄存器初始时值都为 0。这段代码将值 10 和 3 放入程序寄存器%rdx 和%rax，执行三条 nop 指令，然后将寄存器%rdx 加到%rax。我们重点关注两条 irmovq 指令和 addq 指令之间的数据相关造成的可能的数据冒险。图的右边是这个指令序列的流水线图。图中突出显示了周期 6 和 7 的流水线阶段。流水线图的下面是周期 6 中写回活动和周期 7 中译码活动的扩展说明。在周期 7 开始以后，两条 irmovq 都已经通过写回阶段，所以寄存器文件保存着更新过的%rdx 和%rax 的值。因此，当 addq 指令在周期 7 经过译码阶段时，它可以读到源操作数的正确值。在此示例中，两条 irmovq 指令和 addq 指令之间的数据相关没有造成数据冒险。

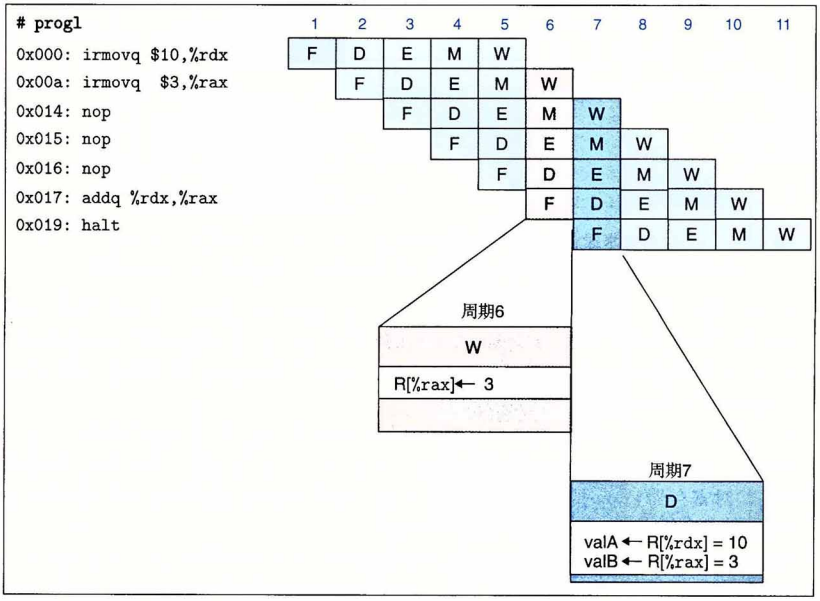

图 4-43　prog1 的流水线化的执行，没有特殊的流水线控制。在周期 6 中，第二个 irmovq 将结果写入寄存器%rax。addq 指令在周期 7 读源操作数，因此得到的是%rdx 和%rax 的正确值