图 4-49～图 4-51 的扩展图还表明译码阶段逻辑能够确定是使用来自寄存器文件的值，还是要用转发过来的值。与每个要写回寄存器文件的值相关的是目的寄存器 ID。逻辑会将这些 ID 与源寄存器 ID srcA 和 srcB 相比较，以此来检测是否需要转发。可能有多个目的寄存器 ID 与一个源 ID 相等。要解决这样的情况，我们必须在各个转发源中建立起优先级关系。在学习转发逻辑的详细设计时，我们会讨论这个内容。

图 4-52 给出的是 PIPE 的结构，它是 PIPE－的扩展，能通过转发处理数据冒险。将这幅图与 PIPE－的结构(图 4-41)相比，我们可以看到来自五个转发源的值反馈到译码阶段中两个标号为“Sel＋Fwd A”和“Fwd B”的块。标号为“Sel＋Fwd A”的块是 PIPE－中标号为“Select A”的块的功能与转发逻辑的结合。它允许流水线寄存器 E 的 valA 为已增加的程序计数器值 valP，从寄存器文件 A 端口读出的值，或者某个转发过来的值。标号为“Fwd B”的块实现的是源操作数 valB 的转发逻辑。

3. 加载/使用数据冒险

有一类数据冒险不能单纯用转发来解决，因为内存读在流水线发生的比较晚。图 4-53 举例说明了*加载/使用冒险*(load/use hazard)，其中一条指令(位于地址 0x028 的 mrmovq)从内存中读出寄存器%rax 的值，而下一条指令(位于地址 0x032 的 addq)需要该值作为源操作数。图的下部是周期 7 和 8 的扩展说明，在此假设所有的程序寄存器都初始化为 0。addq 指令在周期 7 中需要该寄存器的值，但是 mrmovq 指令直到周期 8 才产生出这个值。为了从mrmovq“转发到”addq，转发逻辑不得不将值送回到过去的时间！这显然是不可能的，我们必须找到其他机制来解决这种形式的数据冒险。(位于地址 0x01e 的 irmovq 指令产生的寄存器%rbx 的值，会被位于地址 0x032 的 addq 指令使用，转发能够处理这种数据冒险。)

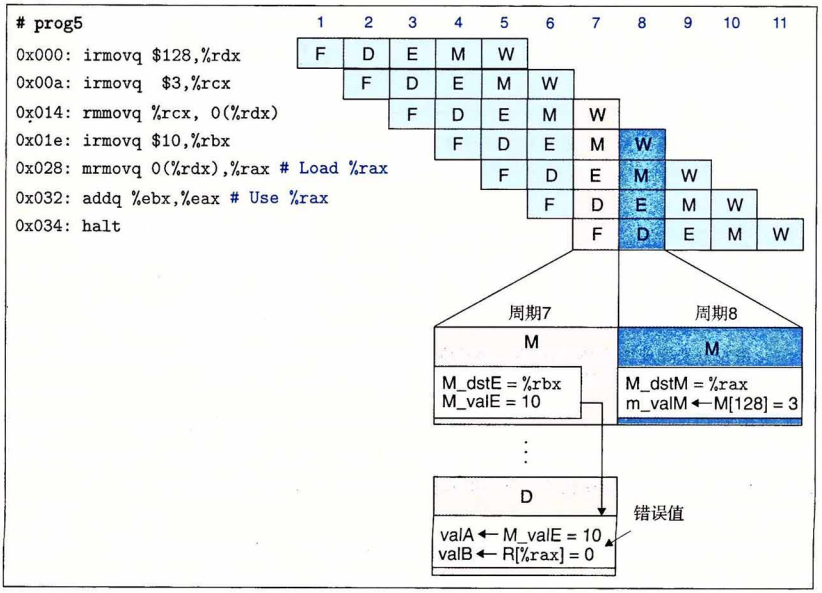

图 4-53　加载/使用数据冒险的示例。addq 指令在周期 7 译码阶段中需要寄存器%rax 的值。前面的 mrmovq 指令在周期 8 访存阶段中读出这个寄存器的新值，这对于 addq 指令来说太迟了