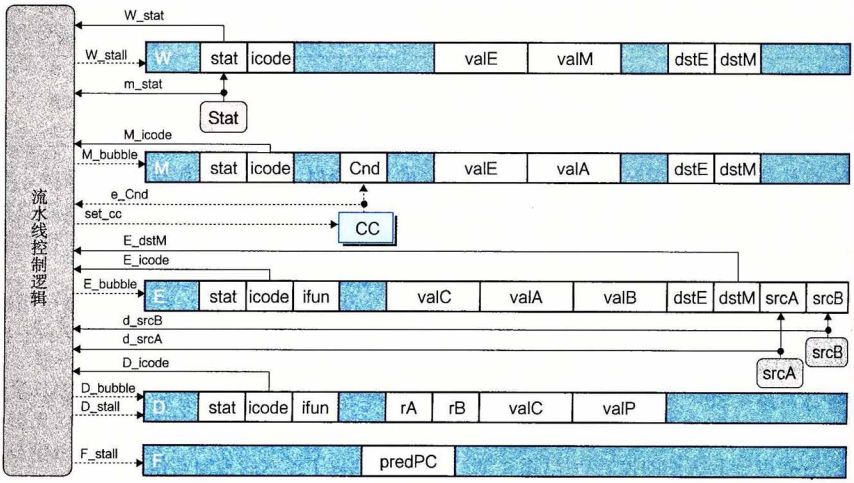

图 4-68　PIPE 流水线控制逻辑。这个逻辑覆盖了通过流水线的正常指令流，以处理特殊条件，例如过程返回、预测错误的分支、加载/使用冒险和程序异常

练习题 4.39　写出 PIPE 实现中信号 D_stall 的 HCL 代码。

遇到预测错误的分支或 ret 指令，流水线寄存器 D 必须设置为气泡。不过，正如前面一节中的分析所示，当遇到加载/使用冒险和 ret 指令组合时，不应该插入气泡：

```
bool D_bubble =
        # Mispredicted branch
        (E_icode == IJXX && !e_Cnd) ||
        # Stalling at fetch while ret passes through pipeline
        # but not condition for a load/use hazard
        !(E_icode in { IMRMOVQ, IPOPQ } && E_dstM in { d_srcA, d_srcB }) &&
          IRET in { D_icode, E_icode, M_icode };
```

练习题 4.40　写出 PIPE 实现中信号 E_bubble 的 HCL 代码。

练习题 4.41　写出 PIPE 实现中信号 set_cc 的 HCL 代码。该信号只有对 OPq 指令才出现，应该考虑程序异常的影响。

练习题 4.42　写出 PIPE 实现中信号 M_bubble 和 W_stall 的 HCL 代码。后一个信号需要修改图 4-64 中列出的异常条件。

现在我们讲完了所有的特殊流水线控制信号的值。在 PIPE 的完整 HCL 代码中，所有其他的流水线控制信号都设为 0。

旁注　测试设计

正如我们看到的，即使是对于一个很简单的微处理器，设计中还是有很多地方会出现问题。使用流水线，处于不同流水线阶段的指令之间有许多不易察觉的交互。我们看到一些设计上的挑战来自于不常见的指令(例如弹出值到栈指针)，或是不常见的指令组合(例如不选择分支的跳转指令后面跟一条 ret 指令)。还看到异常处理增加了一类全新的可能的流水线行为。那么怎样确定我们的设计是正确的呢？对于硬件制造者来说，这