练习题 4.36　在这个阶段中，通过检查数据内存的非法地址情况，我们能够完成状态码 Stat 的计算。写出信号 m_stat 的 HCL 代码。

4.5.8　流水线控制逻辑

现在准备创建流水线控制逻辑，完成我们的 PIPE 设计。这个逻辑必须处理下面 4 种控制情况，这些情况是其他机制(例如数据转发和分支预测)不能处理的：

加载/使用冒险：在一条从内存中读出一个值的指令和一条使用该值的指令之间，流水线必须暂停一个周期。

处理 ret：流水线必须暂停直到 ret 指令到达写回阶段。

预测错误的分支：在分支逻辑发现不应该选择分支之前，分支目标处的几条指令已经进入流水线了。必须取消这些指令，并从跳转指令后面的那条指令开始取指。

异常：当一条指令导致异常，我们想要禁止后面的指令更新程序员可见的状态，并且在异常指令到达写回阶段时，停止执行。

我们先浏览每种情况所期望的行为，然后再设计处理这些情况的控制逻辑。

1. 特殊控制情况所期望的处理

在 4.5.5 节中，我们已经描述了对加载/使用冒险所期望的流水线操作，如图 4-54 所示。只有 mrmovq 和 popq 指令会从内存中读数据。当这两条指令中的任一条处于执行阶段，并且需要该目的寄存器的指令正处在译码阶段时，我们要将第二条指令阻塞在译码阶段，并在下一个周期往执行阶段中插入一个气泡。此后，转发逻辑会解决这个数据冒险。可以将流水线寄存器 D 保持为固定状态，从而将一个指令阻塞在译码阶段。这样做还可以保证流水线寄存器 F 保持为固定状态，由此下一条指令会被再取一次。总之，实现这个流水线流需要发现冒险的情况，保持流水线寄存器 F 和 D 固定不变，并且在执行阶段中插入气泡。

对 ret 指令的处理，我们已经在 4.5.5 节中描述了所需的流水线操作。流水线要停顿 3 个时钟周期，直到 ret 指令经过访存阶段，读出返回地址。通过图 4-55 中下面程序的处理的简化流水线图，说明了这种情况：

```
0x000:    irmovq stack,%rsp  #   Initialize stack pointer
0x00a:    call proc          #   Procedure call
0x013:    irmovq $10,%rdx    #   Return point
0x01d:    halt
0x020: .pos 0x20
0x020: proc:                 # proc:
0x020:    ret                #   Return immediately
0x021:    rrmovq %rdx,%rbx   #   Not executed
0x030: .pos 0x30
0x030: stack:                # stack: Stack pointer
```

图 4-62 是示例程序中 ret 指令的实际处理过程。在此可以看到，没有办法在流水线的取指阶段中插入气泡。每个周期，取指阶段从指令内存中读出一条指令。看看 4.5.7 节中实现 PC 预测逻辑的 HCL 代码，我们可以看到，对 ret 指令来说，PC 的新值被预测成 valP，也就是下一条指令的地址。在我们的示例程序中，这个地址会是 0x021，即 ret 后面 rrmovq 指令的地址。对这个例子来说，这种预测是不对的，即使对大部分情况来说，也是不对的，但是在设计中，我们并不试图正确预测返回地址。取指阶段会暂停 3 个时钟