导致我们在 ctest 测试脚本中增加了附加的情况。

形式化验证仍然处在发展的早期阶段。工具往往很难使用，而且还不能验证大规模的设计。我们能够验证 Y86-64 处理器的部分原因就是因为它们相对比较简单。即使如此，也需要几周的时间和精力，多次运行那些工具，每次最多需要 8 个小时的计算机时间。这是一个活跃的研究领域，有些工具成为可用的商业版本，有些在 Intel、AMD 和 IBM 这样的公司使用。

**网络旁注 ARCH:VLOG　流水线化的 Y86-64 处理器的 Verilog 实现**

正如我们提到过的，现代的逻辑设计包括用硬件描述语言书写硬件设计的文本表示。然后，可以通过模拟和各种形式化验证工具来测试设计。一旦对设计有了信心，我们就可以使用逻辑合成(logic synthesis)工具将设计翻译成实际的逻辑电路。

我们用 Verilog 硬件描述语言开发了 Y86-64 处理器设计的模型。这些设计将实现处理器基本构造块的模块和直接从 HCL 描述产生出来的控制逻辑结合了起来。我们能够合成这些设计的一些，将逻辑电路描述下载到字段可编程的门阵列(FPGA)硬件上，可以在这些处理器上运行实际的 Y86-64 程序。

### 4.5.9　性能分析

我们可以看到，所有需要流水线控制逻辑进行特殊处理的条件，都会导致流水线不能够实现每个时钟周期发射一条新指令的目标。我们可以通过确定往流水线中插入气泡的频率，来衡量这种效率的损失，因为插入气泡会导致未使用的流水线周期。一条返回指令会产生三个气泡，一个加载/使用冒险会产生一个，而一个预测错误的分支会产生两个。我们可以通过计算 PIPE 执行一条指令所需要的平均时钟周期数的估计值，来量化这些处罚对整体性能的影响，这种衡量方法称为 CPI(Cycles Per Instruction，每指令周期数)。这种衡量值是流水线平均吞吐量的倒数，不过时间单位是时钟周期，而不是微微秒。这是一个设计体系结构效率的很有用的衡量标准。

如果我们忽略异常带来的性能损失(异常的定义表明它是很少出现的)，另一种思考 CPI 的方法是，假设我们在处理器上运行某个基准程序，并观察执行阶段的运行。每个周期，执行阶段要么会处理一条指令，然后这条指令继续通过剩下的阶段，直到完成；要么会处理一个由于三种特殊情况之一而插入的气泡。如果这个阶段一共处理了 $C_i$ 条指令和 $C_b$ 个气泡，那么处理器总共需要大约 $C_i+C_b$ 个时钟周期来执行 $C_i$ 条指令。我们说“大约”是因为忽略了启动指令通过流水线的周期。于是，可以用如下方法来计算这个基准程序的 CPI：

$$\text{CPI} = \frac{C_i + C_b}{C_i} = 1.0 + \frac{C_b}{C_i}$$

也就是说，CPI 等于 1.0 加上一个处罚项 $C_b/C_i$，这个项表明执行一条指令平均要插入多少个气泡。因为只有三种指令类型会导致插入气泡，我们可以将这个处罚项分解成三个部分：

$$\text{CPI} = 1.0 + lp + mp + rp$$

这里，$lp$(load penalty，加载处罚)是当由于加载/使用冒险造成暂停时插入气泡的平均数，$mp$(mispredicted branch penalty，预测错误分支处罚)是当由于预测错误取消指令时插入气泡的平均数，而 $rp$(return penalty，返回处罚)是当由于 ret 指令造成暂停时插