处理器调试专题

2024计算机组成原理竞赛班

涉浅水者见虾，其颇深者察鱼鳖，其尤甚者观蛟龙。

你有信心迅速定位处理器问题吗？

处理器调试问题分类

- 逻辑性正确性问题
- 物理性正确性问题
- 性能问题

在龙芯杯竞赛中，逻辑性正确性问题是开发前期经常遇到的问题。由于处理器设计并没有可供证明的“正确性理论”，因此我们永远无法保证处理器的逻辑性正确——在硬件开发中，这个问题会尤为突出。

在处理器开发的全流程中，逻辑性正确性问题的调试速度是“决胜关键”。只有能在压力测试中尽快解决这类问题，才能在后期的性能优化和功能增强中有更多的时间。

波形是否是唯一的解决方案?

是的，波形确实是唯一的解决方案，但我们始终缺乏有效的波形追踪分析手段。

- 调试器能否在发现问题时及时停下?
- 调试器能否输出便于我们分析的额外信息?
- 波形停下的位置距离错误真正发生的位置有多远?
- 波形中哪些信号是我们需要关注的?
- 如何分析波形指示错误的因果关系?

一个用来为难Arch人的例子

一个具有高速缓存的处理器系统——

- 一个测试程序，发现了结果并非所愿
- 经过追踪，发现第100000周期，一个load指令的结果错了，且本次load访问命中
- 向前追溯，发现第74502个周期，这个load指令所在的数据被换入高速缓存
- 再向前追溯，发现第39475个周期，这个load指令所在的数据被换出高速缓存
- 再向前追溯，发现第18272个周期，这个load指令所在的数据被换入高速缓存
- 再向前追溯，发现第9823个周期，这个load指令所在的数据被换出高速缓存
- 再向前追溯，发现第198个周期，原本一条应该store的指令写入了错误的数据，而这个数据应该被前递却没有被前递

波形应该怎么解读?

如果我们只有波形这一个工具，那么我们应该如何追溯错误呢?

- 找到调试工具报出的错误点，从流水线最后一级向前追溯，直到找到出现错误的元件
- 如果错误没有“时间相关性”，则直接根据当前指令的相关性问题解决问题
- 如果错误有“时间相关性”，则需要再向前定位，直到找到错误的根源

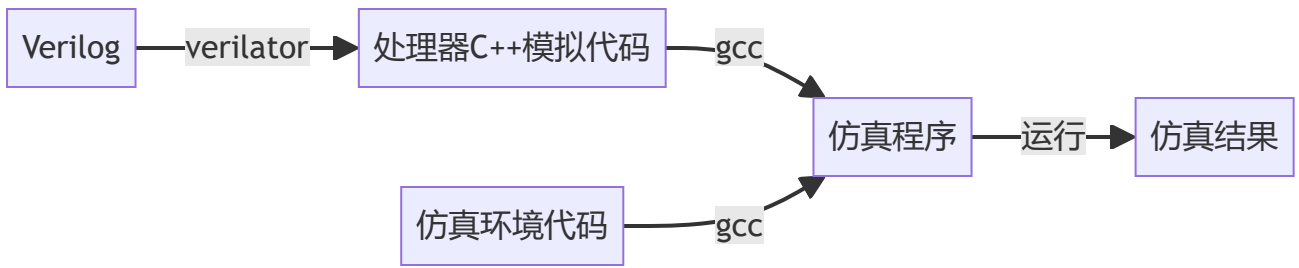
一个常见的误区是，我们经常会“格局小了”，始终盯在错误的位置上，而忽略了其“上游”的问题。

随着设计复杂度的提升，波形文件的大小也会急剧增加，这会占用大量的内存空间。因此，我们需要一种更加高效的波形分析方法。

更高效的仿真工具——Verilator

Verilator是一个开源的Verilog仿真工具，它能够将Verilog代码转换为C++代码，从而实现对Verilog代码的仿真。

- 由于Verilator是将Verilog转化为C++代码，因此可以使用C++来构建仿真环境，并使用代码获取到硬件中的各类信号，从而进行更加高效的仿真。
- 灵活的仿真环境也使得Verilator能整出更多“花活”，很多时候并不需要看波形就可以锁定到错误位置。



Difftest——龙芯杯决胜密码

> 如果你拥有了一个完全正确的CPU，那么能不能利用这个CPU迅速定位到你的CPU的问题呢？

在之前的《计算机程序设计A》课程中，或许有部分同学用过这样一种Debug手段：

- 将其他同学的能通过所有测试点的程序“借”过来
- 额外写一个程序和大量测试用例，将你写的程序和“借”来的程序的输出结果进行逐一比对
- 找到输出不同的测试用例，从而定位到你的程序的问题

换个思路想，我们把每条“指令”看做每个“测试点”，再把一个“程序”看做一个“处理器”，那么我们就可以用同样的方法来定位处理器的问题。不过，为了真正实现这个“伟大理想”，我们需要解决两个问题：

- “正确”的处理器哪里来？
- 如何实现“逐条比对”？

模拟器——一个软件实现的CPU

在开发成本上，软件开发的难度和时间成本要远远低于硬件。因此，我们可以通过软件实现一个“正确”的处理器。

模拟器的定义很广泛，主要包括两种：

- 正确性验证模拟器：这种模拟器的目的是验证处理器的逻辑性正确性，因此它模仿的就是一个简单的单周期CPU
- 性能验证模拟器：这种模拟器的目的是探索硬件算法对处理器的性能的影响，因此它模仿了一个完整的流水线CPU

Difftest本身是为了解决逻辑性正确性问题而生的，因此我们主要关注正确性验证模拟器。而在开发后期，对于分支预测、高速缓存等性能问题，我们可以用性能验证模拟器来快速探索算法的影响。

标准比对——如何定义“处理器状态”

Difftest需要对指令的“结果”进行比对，那么什么是“指令的结果”呢？首先我们来定义一下“处理器状态”：

> 广义的处理器状态，是整个处理器系统中所有寄存器、内存的值组成的集合。

如果每个周期，我们都能把这些内容和一个完全正确的处理器进行比较，那么一定能迅速发现问题。只不过，这样存在两个问题：

- 太耗时：内存单元逐个比较是一件非常恐怖的事情
- 架构问题：这种思路要求你的设计必须和标准处理器完全一致，至少寄存器要完全一样。

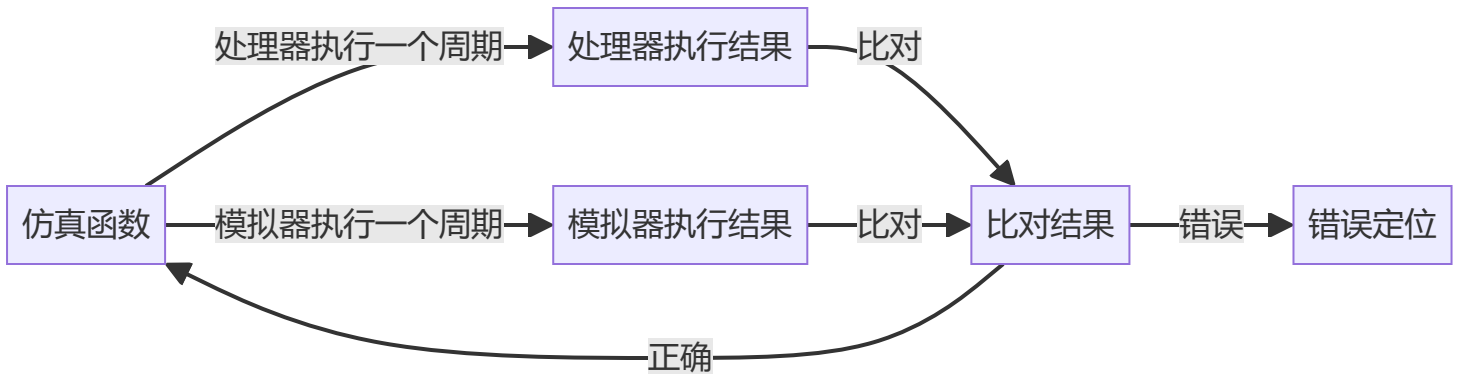
所以，我们需要提炼出一个最精简的集合，来表示处理器的状态：

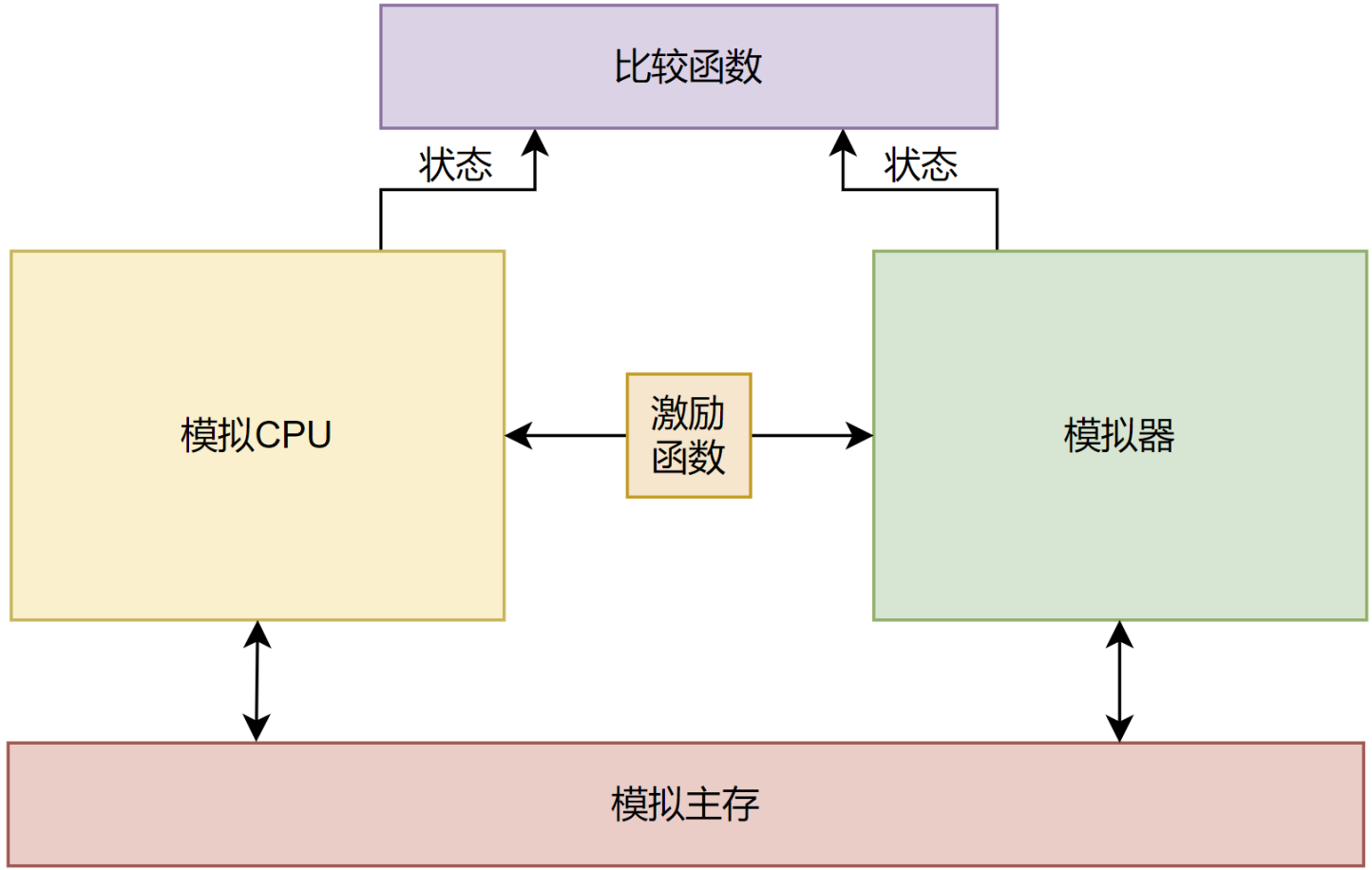
> 处理器状态 = 通用寄存器的全体值 + pc值 + 控制状态寄存器的全体值

这些信息是每一个处理器都必须有的（即便是最基本的模拟器），因此我们可以通过这些信息来进行比对。

Difftest+Verilator——一个高效的调试工具

将Difftest的理念和Verilator的高效仿真工具结合起来，我们就可以实现一个高效的调试工具。我们将上面的软件架构图做一个进一步的补充：




比较函数
状态
状态
模拟CPU
激励
函数
模拟器
模拟主存

Chiplab

- Chiplab 代码仓库
- Chiplab 说明文档