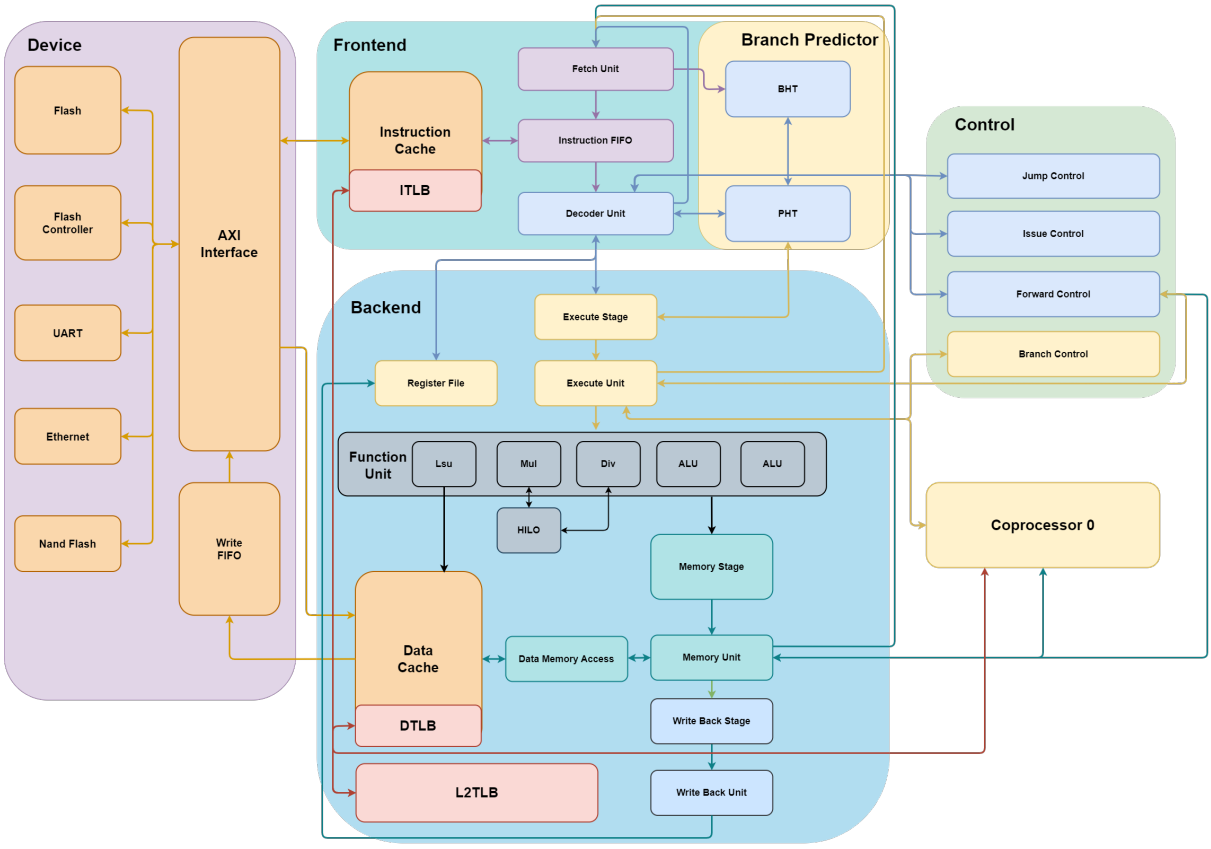
NSCSCC2023

杭州电子科技大学 PUA-MIPS 队
决赛设计报告



叶剑豪 奚力丰 胡致尧

目录

1 概述

我们基于Chisel语言设计并实现了一个MIPS32指令集架构的处理器:PUA-MIPS(Powerful Ultra Architecture MIPS)处理器。能够使用比赛方提供的大部分外设，并且成功通过了龙芯杯官方框架提供的功能测试、性能测试和系统测试。该处理器可以运行PMON引导程序，启动Linux操作系统。

- 顺序双发射六级流水线处理器PUA-MIPS(Powerful Ultra Architecture MIPS)。
- 处理器参数
  - IPC几何平均值 1.25
  - 极限频率 88M
  - 六级流水（2取指，1译码，1执行，1访存，1写回）
  - 几何平均双发率 59.42%
  - 分支预测准确度 94.48%
- 处理器设计
  - 指令集：MIPS32 Release1的除BranchLikely和浮点指令以外的所有指令。
  - 流水线结构：乘法4级流水，除法8级流水，两级访存。非对称双发射。
  - 缓存
    - ICache为两路组相连8KB设计。命中率99.72%
    - DCache为两路组相连8KB设计。命中率95.00%
- Soc
  - CPU：PUA-MIPS CPU（含Cache）
  - DRAM支持：使用板载 DDR3 SDRAM作为主存
  - 串口：实现串口控制器以调整波特率
  - 以太网：使用IP核构建以太网控制器
  - GPIO：使用confreg组件控制LED，数码管等组件
  - 图像输出：实现VGA传输协议
- OS
  - 支持PMON引导程序
  - 支持最新Linux主线v6.5-rc3版本
  - 支持OS驱动Soc
- Chisel语言
  - 参数化设计，支持流水线长度、取指宽度、访存宽度等参数的配置。
  - KISS设计理念加持，单个文件大小不超过200行。
  - 面向对象带来更高的自由度，同一种算法，四种不同实现随意组合。
  - 高性能模拟器带来更快速的设计空间探索。

2 处理器详细设计

PUA-MIPS是顺序双发射结构设计，支持MIPS32 Release1扩展，处理器前端流水级包括取指单元、指令队列等单元，顺序取指。后端包括译码、分支预测、运算单元、寄存器堆、访存控制和写回等单元。顶端有一个控制单元，负责控制各个流水级之间的同步和冲突。缓存分为ICache、DCache等模块。

2.1 前端

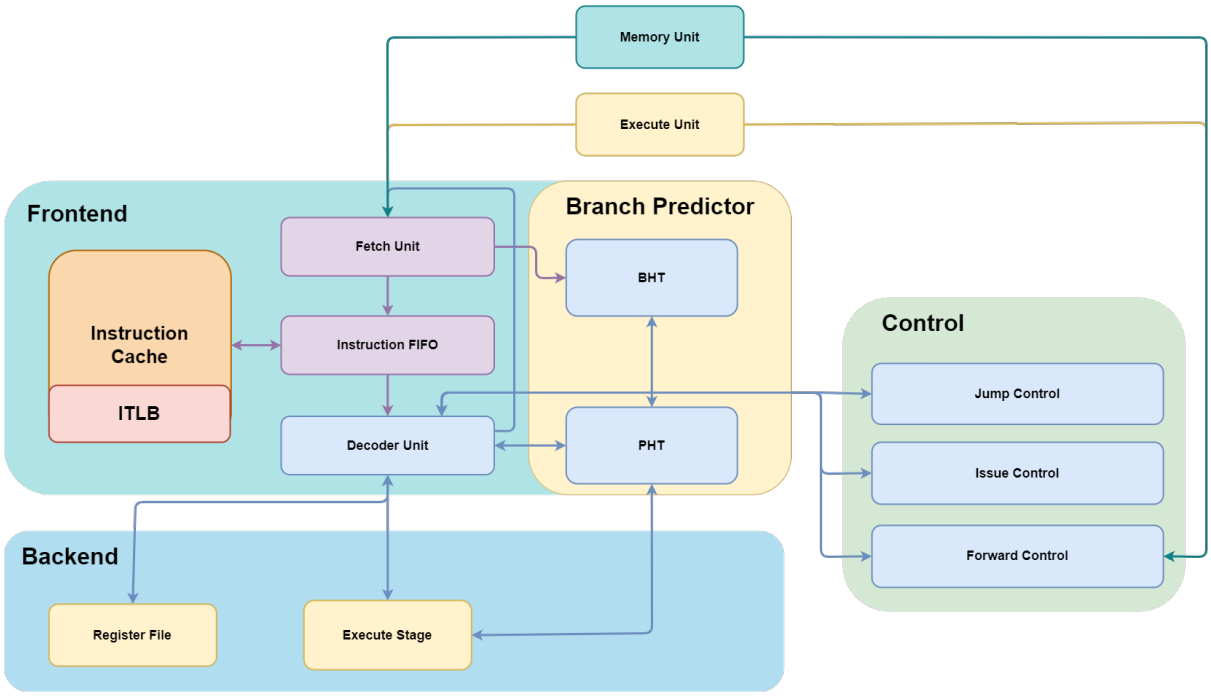


图 2: Frontend Design

2.1.1 取指（Fetch Unit）

前端流水级包括取指单元、指令缓存等单元。取指单元（FetchUnit）用于产生下两条指令的PC并发送到指令缓存（Instruction Cache)。我们通过取指队列（Instruction Fifo）实现了前端（Frontend）和后端（Backend）的充分解耦，使得开发过程能够尽量保持独立。指令队列是一个深度为8的先进先出队列： 对于前端它负责存储从指令缓存发来的指令；对于后端它负责维护转移延迟槽解，与发射控制模块（Issue Control）一同管理指令的发射与保留。在PC发往指令缓存的同时也会并行访问分支预测器获取PHT的下标，并将PHT下标与指令一起存入取指队列中，这个操作将大大优化访问分支预测器的时延，使得我们的分支预测器容量可以尽可能增加。

2.1.2 译码（Decoder Unit）

由于我们是双发射处理器，译码级每周期需要译码两条指令。它会对取指队列的首两条指令进行译码并通过发射控制模块决定是否发射指令。

- 对于简单的Jump指令，我们进行直接跳转，倘若存在JR等与寄存器相关的跳转指令，若在译码级没有成功获得寄存器值，我们将把该跳转指令延迟到Execute级进行处理。
- 对于Branch相关的指令，我们将PHT下标发往分支预测器，获取是否跳转的信息；当分支预测器选择跳转时，我们的Decoder级会发出跳转请求，这时将清空Decoder级前的流水线，而取值队列会帮助我们处理延迟槽相关的问题。
- 在译码阶段还将使用发射控制（Issue Control）模块动态处理指令的双发，这部分信息会反馈给取指队列以控制取指队列的出队。
- 寄存器堆的访问也在这一级进行。这一级所有待执行的指令会得到所有流水级的目的操作数相关信息，通过数据前递模块（Forward Control）处理数据前递。

- 因为chisel会帮助我们自动删除无用的信号，所以我们在译码级尽可能产生后面所有可能用到的控制信号，并包装在inst info结构体中向后传递。

2.2 后端

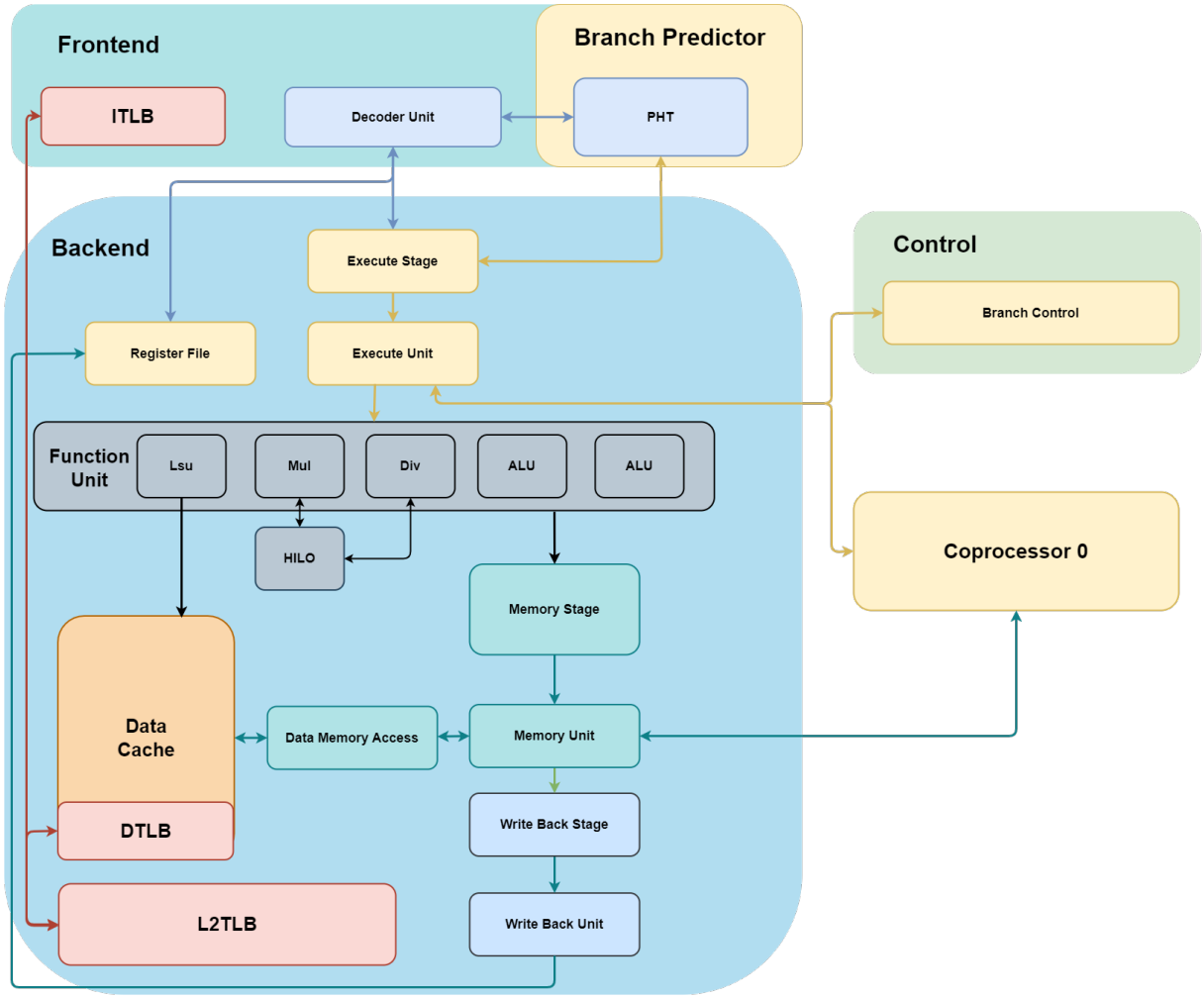


图 3: Backend Design

2.2.1 执行（Execute Unit）

执行阶段中使用译码生成的信号对数据进行运算处理。

- 对于普通的运算指令，我们使用两个ALU运算单元来实现双发射极限情况下的普通运算，对于特殊运算，我们使用不同的运算单元。
- 针对乘除法指令，我们使用Xilinx ip核进行运算，其中乘法两周期，除法四周期。通过握手信号与执行单元进行通信，并在该级将运算内容写回HILO寄存器。
- 针对分支指令，我们在该阶段处理寄存器相关的分支指令，同时也对分支预测器的结果进行修正，当结果不一致时将清空流水线，重新取指运行，并将判断的结果返回分支预测单元，对BHT和PHT的值进行更新。
- 针对访存指令，由于访存需要两个周期，因此我们在执行级同时需要向数据缓存（Data Cache）发送访存信号。
- 我们还需要进行TLB的第一级地址转换来减少TLB翻译带来的时延。
- MTC0和MFC0指令将在这一级进行。
- 在该级实例化了HILO寄存器，MTHI、MTLO、MFHI和MTLO指令将在这一级进行。
- 在该级实例化了LLBit寄存器，用于处理LL和SC相关指令。

2.2.2 访存（Memory Unit）

- 对于Load指令，访存级接受数据缓存返回的数据。
- 对于Store指令，访存级向数据缓存发出写请求。

- 由于所有的例外都会在这一级得到结果，我们的CP0的访问也在这一级发生，倘若遇到例外或者碰到可能修改TLB的指令，我们都会清空流水线，保证执行的正确性。

2.2.3 写回（Write Back Unit）

- 写回阶段进行双发射写回，若遇到了读写冲突则进行前递操作。我们还实现了一个双进一出的提交队列，用于在Vivado上使用官方提供的差分测试。

2.3 双发策略（Issue Control）

- inst1不发射指令的情况
  1. 对于例外相关指令（SYSCALL、ERET等）以及可能修改TLB的相关指令（MTC0、TLBWI等）都只会进行单发操作，在inst0中发射，inst1不发射任何指令，这样可以减少大量判断逻辑（如inst1发出清空流水线的请求，但inst0无异常，此时inst0需要保留，而保留一条指令刷去另一条指令的操作会增加许多不必要的逻辑判断而造成延迟）。
- inst1可能发射指令的情况
  1. 由于我们是顺序双发处理器，绝对不能进行乱序运行，所以当inst0不发射时，inst1肯定不发射。
  2. 对CP0、HILO或寄存器堆的同一个操作数inst0发出写请求，而inst1为读请求，由于没有同一级内部指令间的数据前递，此时会产生RAW冲突，这时inst1不予发射。
  3. 当执行级为访存指令，由于访存指令只有在访存级才会得到结果，这时我们也不予发射指令。因为最多只会停顿一拍，所以我们没实现访存相关的提前唤醒。
  4. 乘除单元以及访存单元只有一个，当两条指令都需同时占用这类单元时，inst1延迟到下一拍发射。
  5. 对于跳转指令，由于MIPS的延迟槽的存在：当inst0为跳转指令时，倘若inst1不能发射，将会被延迟到下一拍作为inst0发射；当inst1为跳转指令时，由于跳转指令会刷新流水线，我们也将该指令延迟到下一拍的inst0发射。

2.4 分支预测（Branch Predictor Unit）

我们实现了多种分支预测器，包括局部历史分支预测、全局分支预测器以及卷积神经网络分支预测器（因为过拟合严重而被放弃）。在除了卷积神经网络的分支预测方法中，局部分支预测器取得的预测效果最好，因此我们最终使用了局部分支预测器。

策略	几何平均准确度
卷积神经网络分支预测	96.58%
局部历史分支预测	94.48%
全局分支历史预测	75.86%

同时，我们对局部历史分支预测进行了设计空间探索。由于局部历史分支预测受到分支历史寄存器查找表（Branch History Register Table，BHT）和模式历史查找表（Pattern History Table，PHT）的深度的影响。因此我们对每一个分支历史寄存器表的深度和模式历史表的深度计算出分支预测的失败率如下图：

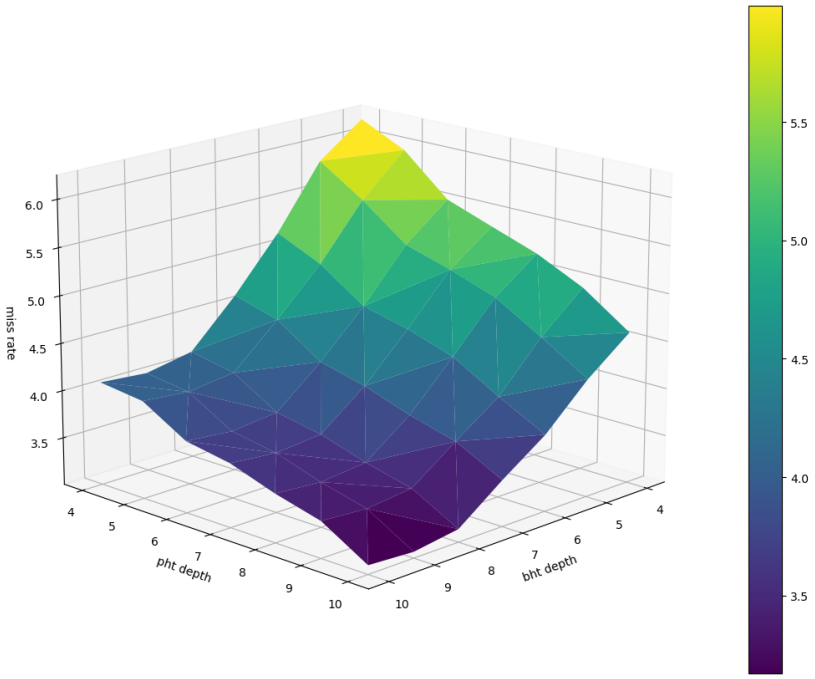


图 4: BPU miss rate

可以看到，在4-8的范围内优化BHT的深度更具性价比。最终出于频率的考虑，我们将BHT的深度设置为6，PHT的深度设置为4。

2.5 缓存（Cache）

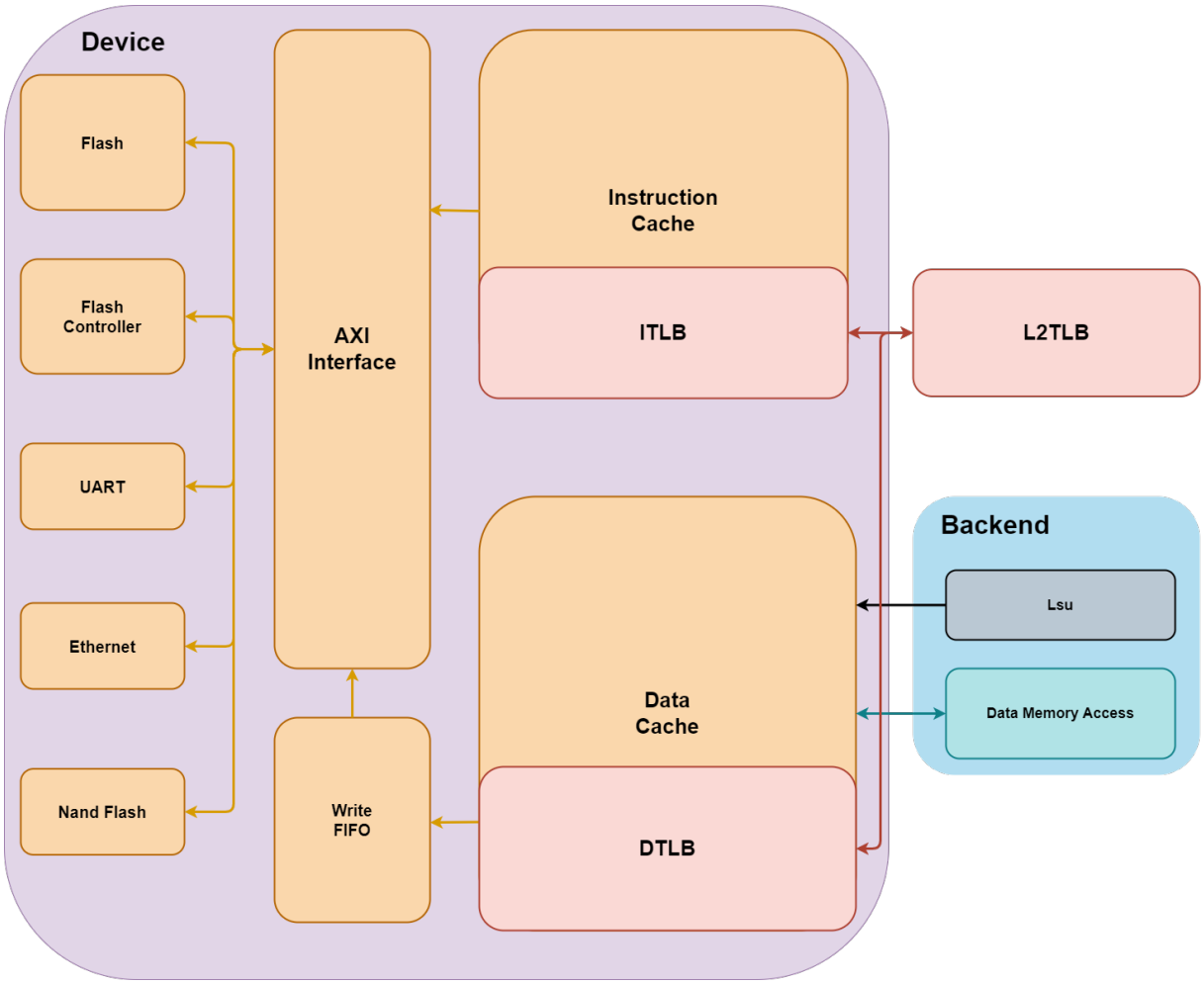


图 5: Cache Design

在PUA-MIPS中，设计实现了指令缓存和数据缓存。

- PLRU替换策略
- VIPT
- AXI突发传输
  - 十六字突发
- 两级TLB平衡延迟
- 两路8KB大缓存
- 高度自由可配置项
- 指令缓存
  - 二宽度取指（可通过参数快速配置）
- 数据缓存
  - 写队列
  - victim Cache

我们通过性能计数器对Cache的命中率做了统计：

测试集	ICache hit	DCache hit
bitcount	99.83%	96.91%
bubble sort	99.97%	99.79%
coremark	99.36%	95.59%
crc32	99.98%	97.85%
dhrystone	99.89%	81.88%
quick sort	99.96%	97.85%
select sort	99.98%	99.33%
sha	99.94%	98.89%
stream copy	99.65%	93.37%
stringsearch	99.96%	90.52%
几何平均值	99.72%	95.00%

2.5.1 指令缓存（Instruction Cache）

指令缓存提供了取指支持，大小和相连度可配置，我们使用状态机来控制缓存。

- IDLE：Cache空闲状态。地址L1TLB查询或直接翻译。若L1TLB发生缺失则暂停，等待L2TLB并处理有可能遇到的异常。
- UNCACHE：向AXI发送读请求或写请求（无突发传输），并等待AXI访存结束。
- REPLACE：cache缺失后，若没有脏位，需要重新向AXI发送读请求（突发传输），访存并替换对应路的缓存行。其中替换策略采用PLRU。
- SAVE：当Cache因为TLB缺失或Uncached导致Cache暂时无法正常访问数据时的保留状态。

2.5.2 数据缓存（Data Cache）

数据缓存相比指令缓存的状态机多了一级。

- WRITEBACK：当需要清除或替换缓存行时，若存在脏位，则需要将脏数据写回，即向AXI发送写请求（突发传输）。

2.5.2.1 Victim Cache

我们通过统计访存指令发现测试集中经常出现一个程序频繁地使用3个数据，而且恰好处在同一个缓存行中，这导致一个way中的数据经常被踢出数据缓存后又马上读入，发生了“抖动”。Victim Cache可以将被踢出数据缓存的数据暂时保存起来，这一操作极大的减少了数据缓存的缺失率。

2.5.2.2 写队列（Write Fifo）

在我们的设计中，对于Uncached的AXI写请求，我们设置了写请求队列，将AXI写请求数据缓存下来，再逐个写回，该期间如果没有Cached的访问AXI请求，则流水线不会因为写回数据而阻塞，提高了流水线的执行效率。

2.6 CP0与例外

我们实现了启动Linux所必需的所有CP0寄存器和例外。

CP0编号	CP0名称	CP0描述
0	Index	TLB 数组的索引
1	Random	随机数
2	EntryLo0	TLB 项的低位
3	EntryLo1	TLB 项的低位
4	Context	指向内存中页表入口的指针
5	PageMask	控制 TLB 的虚拟页大小
6	Wired	控制 TLB 中固定的页数
8	BadVAddr	记录最新地址相关例外的出错地址
9	Count	处理器内部计数器
10	EntryHi	TLB 项的高位
11	Compare	计时中断控制器
12	Status	处理器状态与控制寄存器
13	Cause	存放上一次例外原因
14	EPC	存放上一次发生例外指令的 PC
15	PRId	处理器版本和标识符
15	EBase	中断向量基地址寄存器
16	Config0	处理器配置
28	TagLo	缓存标签 Tag 的低位
29	TagHi	缓存标签 Tag 的高位
30	ErrorEPC	上一次发生例外的计数器数值

例外编号	例外名称	例外描述
00	INT	中断异常
01	MOD	TLB 条目修改异常
02	TLBL	TLB 非法取指令或访问异常
03	TLBS	TLB 非法存储访问异常
04	ADEL	地址未对齐异常（取指令或访问异常）
05	ADES	地址未对齐异常（存储访问异常）
08	SYS	系统调用异常
09	BP	断点异常
0a	RI	保留指令异常
0b	CPU	协处理器不可用异常
0c	OV	算术溢出异常
1f	NO	无异常

2.7 内存管理

CPU使用内存管理单元(MMU) 以及相应的地址映射关系来进行虚地址到实地址的转换。针对TLB的设计，我们参考了CDIM和GenshinCPU的二级TLB设计。这一设计使得我们在使用8项二级TLB的情况下仍能保持较高频率。

3 系统外设

3.1 外设

外设方面，我们修改了官方提供的Soc Up框架，构建了我们的Soc。这套SOC框架通过AXI总线与处理器核进行交互。外设包括：

- DRAM支持：使用板载 DDR3 SDRAM作为主存，DDR3分配的虚拟地址段为外设剩余的虚拟地址。大小为128MB。
- UART串口：实现串口控制器以调整波特率
- Ethernet以太网：使用IP核构建以太网控制器
- GPIO：使用confreg组件控制LED，数码管等组件
- 图像输出：实现静态图片传输

地址映射如下：

设备	起始地址	终止地址	大小
Flash	0×BFC00000	0×BFC0FFFF	1MB
GPIO	0×BFD00000	0×BFD0FFFF	64KB
UART	0×BFE40000	0×BFE43FFF	16KB
Flash Controller	0×BFE40000	0×BFE4FFFF	64KB
Nand Flash	0×BFE78000	0×BFE7BFFF	16KB
MAC	0×BFF00000	0×BFF0FFFF	64KB

3.2 系统

当前PUA MIPS可以启动uboot,ucore和最新Linux 6.5.0-rc3。Linux中去除了Branch-likely指令。我们通过串口与Linux进行交互。

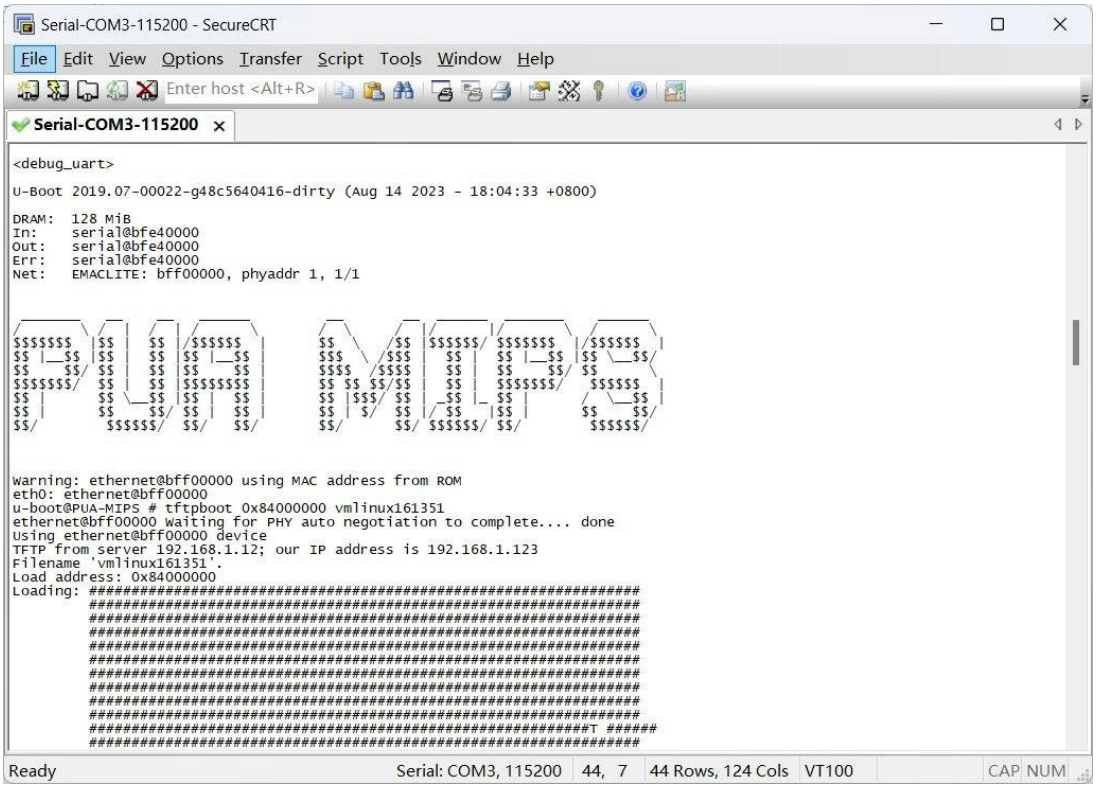


图 6： U-BOOT

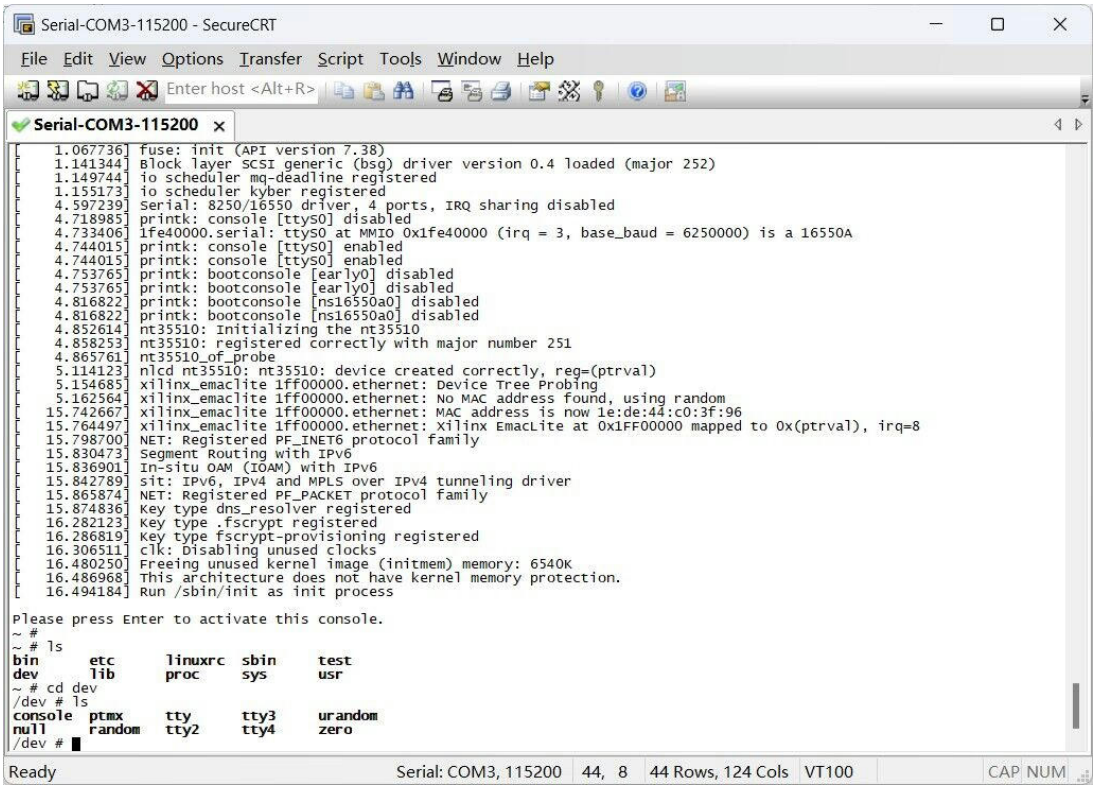


图 7： LINUX

4 参考资料

- NSCSCC往届处理器设计
  - nontrivial-mips 给了我们如何启动Linux操作系统的良好示范。
  - TrivialMIPS 中有许多顺序处理器设计的技巧。
  - amadeus-mips 帮助我们更好的使用Chisel语言。
  - UltraMIPS 给了我们Cache设计的良好示范。
  - CDIM 为我们提供了高效的测试框架。
  - GenshinCPU 有许多频率优化的设计方法。
  - zenCove 在高级语言的使用上启发了我们。
- 开源处理器
  - 香山处理器 给了我们很多架构设计上的启发。
  - 果壳处理器 一个优秀处理器的示范。
  - dinocpu 一个以Chisel语言为基础开设的处理器设计课程，使用基于Scala的交互式调试框架给了我们很多帮助。
- 书籍
  - 计算机组成与设计：硬件/软件接口
  - 计算机体系结构：量化研究方法(第6版)
  - CPU设计实战
  - 超标量处理器设计
  - 自己动手写CPU