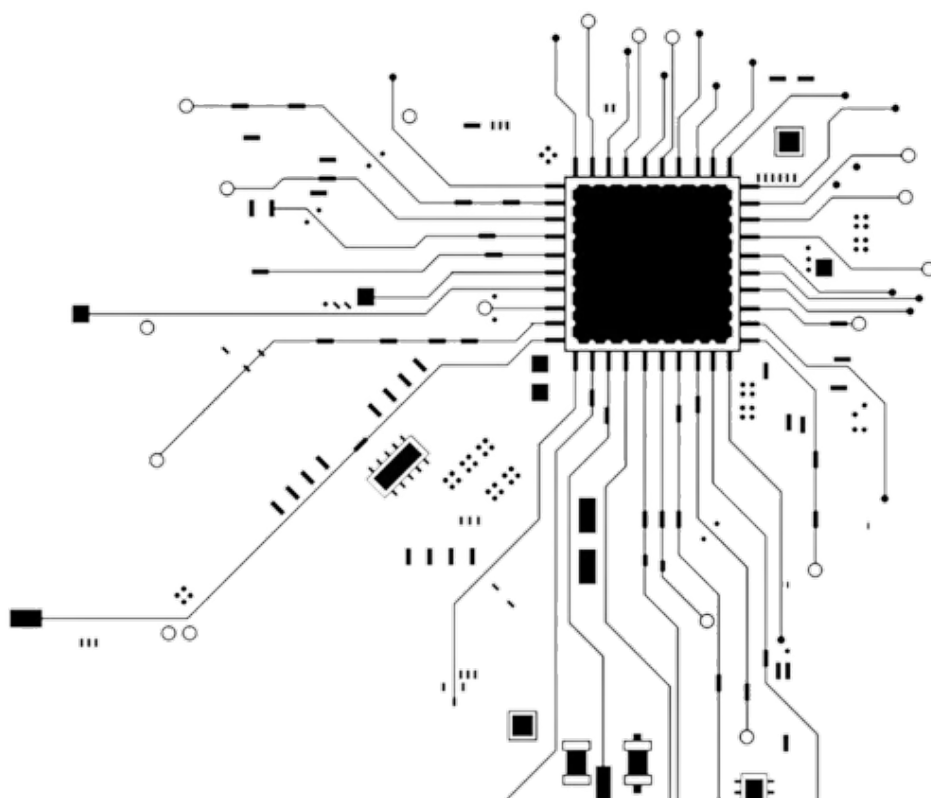


MBoom 项目报告

第七届龙芯杯决赛作品设计报告



华中科技大学一队

龚言龙 程家骏 刘思辰

2023 年 8 月 18 日

目录

1 概述	2
1.1 项目背景	2
1.2 项目概述	2
1.2.1 语言与指令	2
1.2.2 cpu 架构	2
1.2.3 系统软件	2
1.3 开发平台	2
1.3.1 硬件平台	2
1.3.2 软件平台	2
2 cpu 架构	3
2.1 前端	3
2.1.1 NLP	3
2.1.2 BPU	4
2.1.3 InstrCache	4
2.2 后端	4
2.2.1 解码阶段	5
2.2.2 重命名阶段	6
2.2.3 分发阶段	6
2.2.4 发射阶段	6
2.2.5 算术指令流水线 (ALU)	7
2.2.6 乘除法指令流水线 (MDU)	7
2.2.7 访存子系统 (LSU)	8
2.2.8 DataCache	9
2.3 支持指令集	10
2.4 cp0 协处理器	10
2.5 内存管理	11
2.6 异常处理	11
3 soc 设计	12
3.1 总体结构	12
3.2 外设支持	12
4 性能优化	14
4.1 推测唤醒	14
4.2 bypass 通路	14
4.3 微结构调整	14
5 系统软件	15
5.1 监控程序	15
5.2 引导程序	15
5.2.1 pmon	15

5.2.2	uboot	15
5.3	操作系统程序	16
5.3.1	ucore	16
5.3.2	linux	16
6	测试结果	18
6.1	功能测试	18
6.2	性能测试	18
6.3	系统测试	18
6.3.1	监控程序	18
6.3.2	记忆游戏	19
7	总结与展望	20

1 概述

1.1 项目背景

本项目为第七届龙芯杯初赛提交作品，在 FPGA 平台上运行 mips32 架构的 cpu 并启动监控程序，pmon, u-core 等，并最终成功地在我们的 soc 上通过 uboot 启动 linux，充分验证了 cpu 实现的标准性与稳定性。

1.2 项目概述

1.2.1 语言与指令

项目使用 chisel 这一基于 scala 的高级硬件描述语言进行编写。Chisel 是基于 Scala 高级语言的 RTL 框架，相对于传统 Verilog 或 VHDL 而言具有更强的抽象能力。但是与之对应的是 Chisel 这门语言翻译生成的 RTL 比较偏向软件思维，在处理关键模块时应该慎之又慎。

支持 92 条指令，其中逻辑运行指令 14 条，算术运算指令 21 条，分支指令 13 条，访存指令 12 条，移动操作指令 2 条，系统相关指令 30 条。

1.2.2 cpu 架构

cpu 整体架构为乱序双发射，乱序发射，顺序提交，最理想情况下一个周期可以提交两条指令，整体大致可以分为前端和后端两个部分，前端模块负责进行取址，后端模块负责进行指令的解码，分发，执行，写回等。

1.2.3 系统软件

我们运行的系统软件如下所示

- pmon: 龙芯提供的系统引导程序
- linux6.5-rc2
- u-boot 引导程序
- 系统监控程序

1.3 开发平台

1.3.1 硬件平台

本次使用的硬件平台是比赛方提供的龙芯体系结构教学实验箱 (Artix-7)，其核心是一块基于 FPGA 芯片的嵌入式系统开发板，型号为 XC7A200T-FBG676，此外，平台包含了 DDR3、SRAM、NAND、Flash 等丰富的外设资源。

1.3.2 软件平台

对于 chisel 代码的编写，使用的是 JetBrains IntelliJ IDEA，用于生成 verilog 代码，对于上板以及 soc 设计使用 vivado2019.2

2 cpu 架构

Mboom 采用的是乱序双发射的结构，理想情况下每个周期可以发射两条指令，提交两条指令，整体结构上可以分为前端和后端。cpu 的整体架构图如图所示

2.1 前端

前端主要负责取回指令、分支预测，将指令送给后端。其主要有以下模块

- pcGen: 综合分支预测、precode 模块重定向和后端重定向生成下一个取指 PC 值。
- ICache: 向内存发起取指请求并缓存指令。
- NLP: 根据 PC 值进行单周期分支预测，本质是 uBTB。
- BPU: 根据 PC 值进行分支预测，接收后端 commit 信息进行训练。
- PreDecode: 对指令进行预解码，修正 J 指令和 JAL 指令和非跳转指令的预测结果。
- IBuffer: 本质是特化的多端口队列，接收 PreDecode 得到的指令信息，传递给后端。

前端总体架构图如图所示。

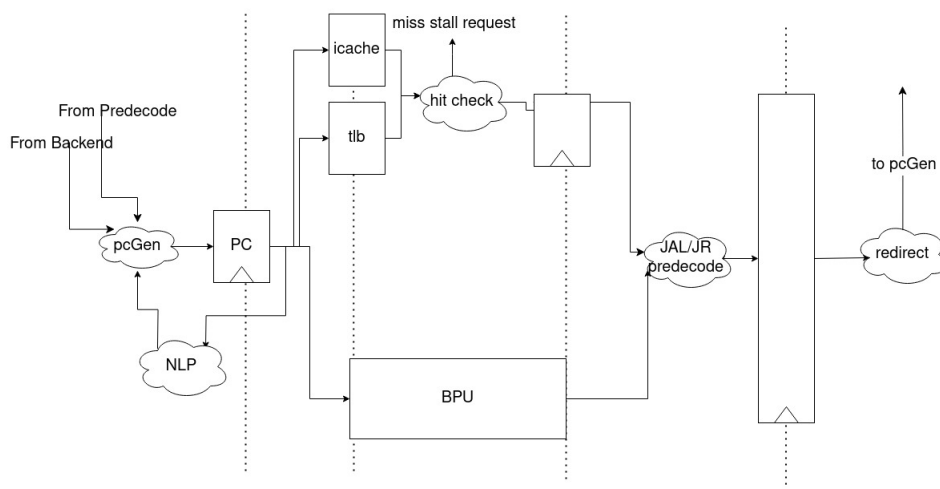


图 1: 前端总体架构

2.1.1 NLP

NLP 是一个单周期分支预测器，其主要目的是减少前端分支预测延迟带来的气泡。NLP 由 BTB 和 BIM 组成，BTB 是为了预测当前指令是否是分支指令，BIM 则是由简单二位饱和和分支预测器组成，用来预测分支是否应该跳转。

NLP 在我们的设计中实际是 BPU 的第一级，BPU 访问 BTB 获取的信息可以被 NLP 复用。BPU 第二级则需要缓存 NLP 的预测结果，在下一个周期将 NLP 结果传递给 Predecode 阶段，对 NLP 阶段的分支预测结果加以修正。

2.1.2 BPU

在 BPU 分支预测中, 使用了三级流水进行处理, 主要包含 BTB, 局部历史分支预测器和函数调用返回栈。

第一级中访问 BTB(Branch Target Buffer) 查找跳转的地址和跳转的类型, 并访问 BHT(Branch History Table) 获取到跳转的历史信息。

第二级中使用 BHT 中取出的跳转历史访问 PHT(Pattern History Table), 综合 BTB 的信息获取分支预测结果, 并输出预测结果。

第三级中根据上一级流水的结果向 RAS(Return Address Stack) 发起 Push 或 Pop 操作。

子分支预测器设计细节如下:

- 局部历史分支预测器: 我们使用的局部分支预测器采用位拼接法, 将 bht 中获取的四位历史和 pc 中的一部分拼接后对 pht 进行寻址获取到二位饱和计数器进行分支预测。由于局部历史数量众多, 不方便做推测更新, 因此为了保证训练的正确性, 我们在 commit 阶段进行历史的修改和训练。这种方案对大多数循环效果不错, 但对密集的短循环由于不能及时更新历史效果较差。
- 函数跳转返回栈: 我们支持 32 项 RAS, 每项带有一个计数器, 支持递归调用压缩存储。在 Commit 阶段分支预测训练时, 将 Decode 阶段获取的分支类型写入 BTB 中, 在分支预测时我们根据 BTB 中保存的分支类型进行 RAS 的 Push 或 Pop 操作。目前方案中 RAS 未实现分支预测失败时的状态恢复。考虑到实际执行时大多情况下工具函数的调用占主要部分, 当函数跳转时很快就会返回, 所以该设计也能达到不错的效果。

2.1.3 InstrCache

InstrCache 实现为 4-way VIPT Cache, 采用两级流水线实现。第一级流水线获取 tag 并和 MMU 得到的翻译结果进行比较, 第二级流水获取指令, 并处理指令缺失逻辑。这里如果遇到指令 TLB Miss, 则直接返回全 0, 即 NOP 指令, 等到其提交时再触发 TLB Miss 异常。

InstrCache 支持使用 Cache 指令进行管理。后端解码出 ICache Invalidation 操作后监听 InstrCache 的状态, 当 InstrCache 空闲时发起 Invalidate 请求, 在 InstrCache 的第二个周期处理。

2.2 后端

后端进行指令的解码, 寄存器重命名, 执行, 提交, 异常处理等 cpu 的主要功能, 整体架构图如图 2 所示

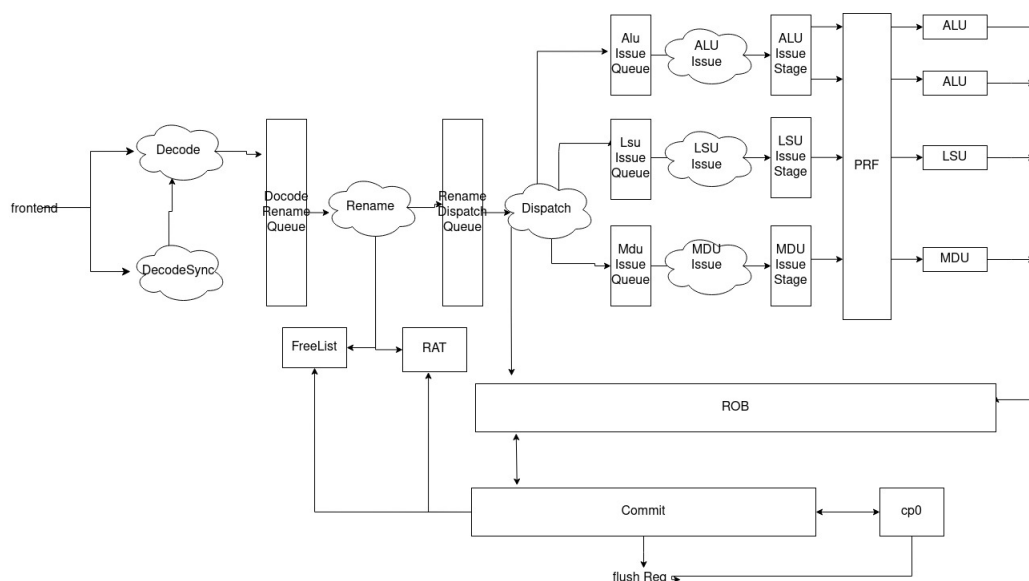


图 2: 后端总体架构

2.2.1 解码阶段

后端接受前端传递来的指令进行解码，通过 Controller 生成控制信号，生成的信号如表 1 所示

信号名	含义
regWriteEn	写寄存器信号
loadMode	load 指令类型
storeMode	store 指令类型
aluOp	运算指令类型
op1Recipe	寄存器操作字段 (rs)
op2Recipe	寄存器操作字段 (rt)
opExtRecipe	寄存器操作字段 (rd)
bjCond	指令跳转类型
regDst	目的寄存器
instrType	指令类型
mduOp	乘除法以及系统相关指令类型
immRecipe	立即数类型

表 1: 控制信号

得到控制信号后将会将相关的信息存储到一个 MicroOp 的结构中，后续在流水线中进行传递时也会使用该结构进行传递。

目前我们对 cp0 指令、tlb 指令等敏感指令的处理方法保证其与上下文严格顺序执行，具体方案是将其和其后紧跟一条指令阻塞至 dispatch 阶段。为了整理阻塞信息需要上下文，DecodeSync 模块就是用来保存该信息的模块，它会记录上一个通过 Decode 阶段的指令是否是敏感指令。

2.2.2 重命名阶段

在寄存器重命名阶段使用统一物理寄存器的方式，对写寄存器进行重命名，共有 64 个物理寄存器；使用寄存器映射表 (RMT) 在存储逻辑寄存器到物理寄存器的映射关系，具体结构如图 3 所示。

物理寄存器使用 freelist 进行分配管理，并使用 busytable 来维持此时的寄存器占用情况。为了减少 rename 阶段的延迟，freelist 采用 fifo 实现，可以几乎零成本获取到空闲物理寄存器项。

在多发处理器中，rename 阶段需要检查当周期多条指令之中的 WAW 相关性和 RAW 相关性，WAW 相关性的检查是为了消除 commit 阶段释放 original register 的冲突，RAW 相关性的检查是因为重命名本身无法解决 RAW 冲突，故当周期需要额外加以检查。

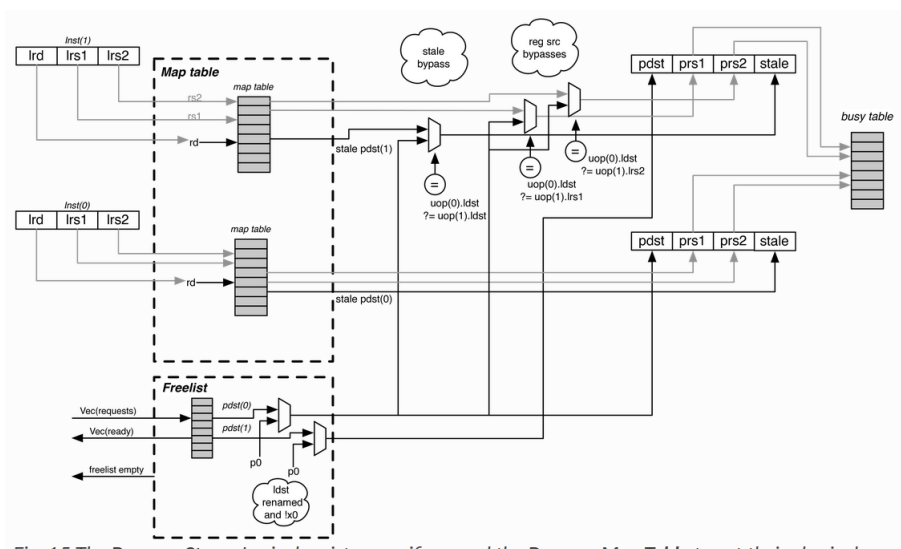


图 3: rename 阶段

2.2.3 分发阶段

在指令的分发阶段，由 dispatch 单元进行处理，向两个运算单元，一个访存单元，一个乘除法单元，一个分支处理单元，每个周期最多从 RenameDispatchQueue 中处理两条指令，当两条指令都是访存指令时或者乘除法指令时，此时需要分两次进行发射。

当可以分发时，也向 ROB 发起入队请求，将当前指令的相关信息写入 ROB 以待 commit 阶段使用。

2.2.4 发射阶段

在指令的发射阶段，使用分布式的压缩队列的方式进行指令的发射。对于两个运算单元使用集中式发射队列，乘除法单元，访问单元，分支处理单元分别使用单独的发射队列。发射队列均使用压缩队列实现。压缩队列的窗口大小为 8，每次取出一条到两条满足条件的进行发射，示意图如下

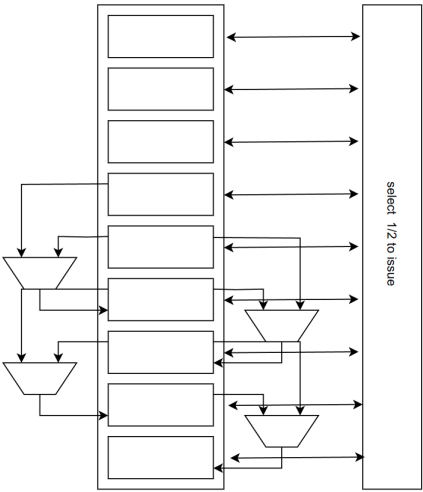


图 4: 指令发射阶段

2.2.5 算术指令流水线 (ALU)

在指令的执行阶段，进行运算指令，访存指令等的执行。
对于运算指令的执行，可以被分为三级流水，如图 5 所示

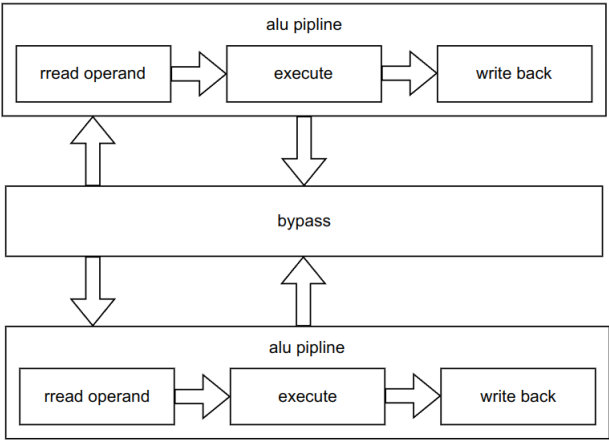


图 5: 算术指令执行阶段

在第一级流水中获取到操作数，操作数的来源有三个：立即数，bypass，prf
在第二级流水中进行正式的运算，得到结果同时进行 bypass
在第三级流水中进行写回，向 rob 以及 prf 写回结果

2.2.6 乘除法指令流水线 (MDU)

我们将系统指令和乘除法指令交由 MDU 执行，他们都是长度各异流水线指令，为了简单性考虑我们在执行阶段每次只放行一条指令，不允许执行阶段存在两条或两条以上指令。

这些指令的指令有三个执行阶段。第一个阶段读取操作数，第二个阶段指令执行，此时会将指令二次派发到四个不同的执行器，分别对应乘除法指令，cp0 相关指令，tlb 相关指令，cache 指令，第三个阶段综合第二个阶段获取的结果写回 ROB 和 PRF。

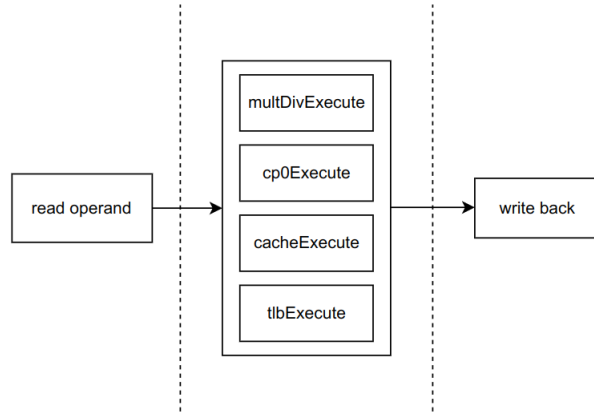


图 6: 访问以及系统相关指令执行

2.2.7 访存子系统 (LSU)

LSU 负责数据的读写。由于乱序唤醒机制的存在，所以总体 IPC 受 LSU 访存级数的影响很大，为了尽可能提高效率，我们将访存命中时的流水线压缩到了四级流水线，总体架构如图所示。

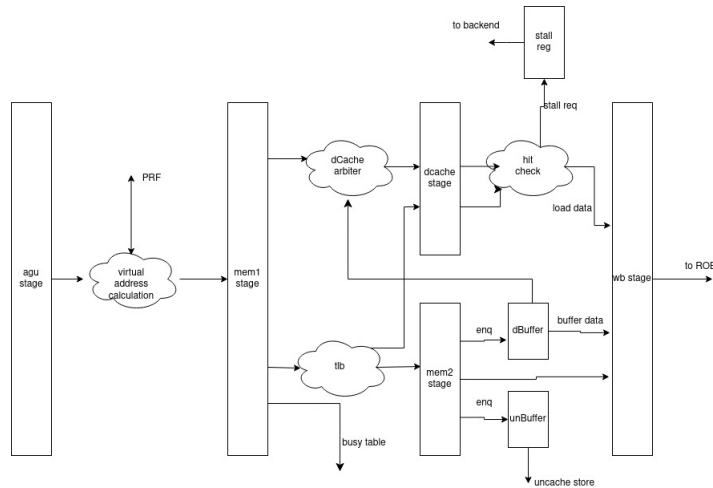


图 7: 访存子系统设计

- load 指令的执行流程如下：
 - a) (AGU Stage) 访问 prf 获取物理寄存器值并计算虚拟地址。
 - b) (Mem1 Stage) 访问 tlb 进行物理地址的计算,同时使用 virtual index 访问 dCache; 将 busyTable 中对应寄存器 busy 位清空。

- c) (Mem2 Stage) 使用 physical tag 在 dCache 中作命中检查 (若不命中则该周期需要阻塞数个周期), 获取 dCache 中的数据; 同时使用 physical address 访问 dBuffer 查询尚未提交和写回的写请求中的数据。
 - d) (WB Stage) 将访存数据写回 prf, 并通知 ROB 对应指令已完成。
- store 指令的执行流程如下:
 - a) (AGU Stage) 访问 prf 获取物理寄存器值并计算虚拟地址。
 - b) (Mem1 Stage) 访问 tlb 进行物理地址的计算。
 - c) (Mem2 Stage) 根据 tlb 翻译的 uncache 结果将写数据和写地址入队 dBuffer 或者 unBuffer。
 - d) (WB Stage) 通知 ROB 对应指令已完成。
 - store 指令的提交流程如下:
 - a) commit 阶段提交 store 指令后告知 dBuffer 和 unBuffer 数据可以写回。
 - b) dBuffer 在等到 Mem1 Stage 空闲或者为 store 指令时向 dCache 发起访存请求, 并占用 dCache 的 Mem2 Stage 进行写 cache 或缺失处理。
 - c) unBuffer 在 unCache store queue 未滿时发起写请求。

为了实现高效率访存, LSU 中设计了推测唤醒机制, 在 Mem1 S 阶段写 BusyTable, 这样就可以保证能在 load 指令写 prf 的下一个周期下一条指令就能取出对应数据。而当 unCache read 或者 dCache read 缺失时则需要阻塞所有执行单元的发射。这种方案虽然暴力, 但是考虑到 dCache 命中率很高, 所以能取得不错的效果。

2.2.8 DataCache

我们的“DataCache”模块实际上包含了 DataCache 缓存和 UnCache Unit 两个部分, 也就是说数据的读写最终都由“DataCache”完成。

这种设计是考虑了 VIPT Cache。为了实现 VIPT, LSU 会同时向 DataCache 和 TLB 发起访问请求, 而发起请求时 LSU 并不知道访存地址的属性是否是 Cache。因此我们在 LSU 发起请求时会把访存地址属性均当成 Cache 处理, 等到下一周期 TLB 结果返回后再分流为 Cache 或 UnCache 处理。这样 DataCache 和 UnCache Unit 在一起实现是很自然的事情。

数据缓存 (DataCache) 实现为 2-way VIPT Cache, 替换策略为 PLRU 算法。读写数据缓存均采用两级流水线实现。第一级流水获取 Cache 行的 tag 和 valid。对于读指令, 第二级流水取出缓存数据; 对于写指令, 第二级流水通过字节选信号写入数据。因此, 缓存命中时, 读写指令均有一个时钟周期延迟, 而若缓存未命中, 则有更长延迟。

UnCache Unit 的处理则是在数据缓存的第二级流水进行二次分发, 分发给相应子单元发起 UnCache 读写请求。

通过上面的探讨, 可以发现第二级流水的状态是复杂的, 因此需要引入状态机对各功能的状态进行描述。第二级流水状态转移图如下:

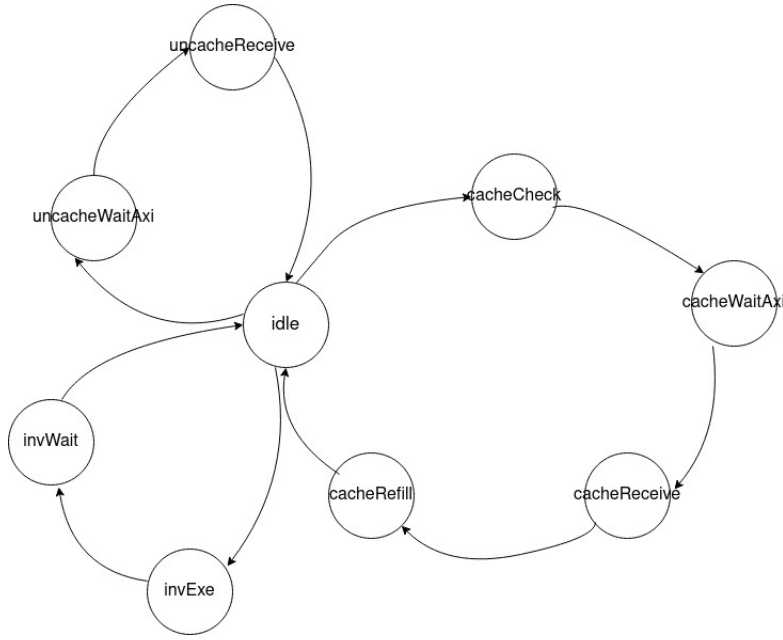


图 8: dcache 状态转移图

2.3 支持指令集

目前我们支持了 MIPS32 Release1 指令集中除了浮点指令, Branch Likely 指令和 LL/SC 指令外的其他指令。具体指令支持如下:

- 自陷指令: TGE, TEGU, TLT, TLTU, TEQ, TNE, TGEI, TGEIU, TLTI, TLTIU, TEQI, TNEI
- 分支指令: BLTZ, BGEZ, BLTZAL, BGEZAL, BEQ, BNE, BLEZ, BGTZ, JR, JALR, J, JAL
- 逻辑指令: AND, OR, XOR, ANDI, ORI, XORI, NOR, SLL, SRL, SRA, SLLV, SRLV, SRAV
- 算术指令: ADD, ADDU, SUB, SUBU, ADDI, ADDIU, MUL, MULT, MULTU, DIV, DIVU, MADD, MADDU, MSUB, MSUBU, CLO, CLZ
- 访存指令: SB, SH, SW, SWL, SWR, LB, LH, LWL, LWR, LW, LBU, LHU
- 特权指令: CACHE, SYSCALL, BREAK, TLBR, TLBWI, TLBWR, TLBP, ERET, MTC0, MFC0, WAIT
- 条件移动指令: SLT, SLTU, SLTI, SLTIU, MOVN, MOVZ
- 无条件移动指令: LUI, SLT, SLTU, MFHI, MFLO, MTHI, MTLO
- 其他指令: PREF, SYNC

2.4 cp0 协处理器

我们实现了 22 个 cp0 寄存器用于提供对操作系统的支持, 如表 2 所示

寄存器名	编号	寄存器名	编号
Index	0,0	Cause	13,0
Random	1,0	Epc	14,0
EntryLo0	2,0	Prid	15,0
EntryLo1	3,0	Ebase	15,1
Context	4,0	Config	16
PageMask	5,0	Config1	17
Wired	6,0	Config2	18
BadVAddr	7,0	TagLo	28
Count	8,0	TagHi	29
EntryHi	9,0	ErrorEPC	30
Compare	10,0		
Status	11,0		

表 2: 实现的 cp0 寄存器

段	虚拟地址	物理地址	uncache
kseg3/ksseg	0xC0000000-0xFFFFFFFF	由 TLB 转换	TLB 项
kseg1	0xA0000000-0xBFFFFFFF	0x00000000-0x1FFFFFFF	是
kseg0	0x80000000-0xFFFFFFFF	0x00000000-0x1FFFFFFF	否
useg	0x00000000-0x7FFFFFFF	由 TLB 转换	TLB 项

表 3: MIPS 地址映射

2.5 内存管理

我们依照 MIPS 要求实现了 TLB 虚拟地址转换过程, MIPS 标准对虚拟地址到物理地址的映射的要求如表 3 所示。目前我们的 CPU 支持 TLB 项数为 8 项, 取指或访存之前均需要对 MMU 进行访问, 当 TLB 缺失时则触发 TLB Miss 异常, 由操作系统对 TLB Miss 进行处理, 使用 TLBWI 指令或 TLBWR 指令对 TLB 项进行填充。

2.6 异常处理

由于 mips 中的异常为精确异常, 在其之前的指令都需要被执行, 所以我们选择在 commit 阶段进行处理, 当出现异常后需要清空流水, 保证其后的指令不被执行, 然后将异常信号传递到 cp0 中, 由 cp0 计算得到下一步的 PC 值。

当发生地址类型的错误时, 还需要将错误的地址存入到 BadVAddr 中, 当中断处理程序完成后, 将会执行异常返回指令 (eret), 我们将异常返回也作为一种异常进行处理, 此时将 EPC 寄存器中的值送入到 PC 寄存器中, 同时将 status 寄存器中的 exl 位标记为 0, 表示此时异常处理完成。

我们实现了大赛以及运行 linux 所需要的所有异常, 如表 4 所示。

异常类型	异常符号	编码	异常描述
中断	Int	0x00	有中断发生
TLB 修改异常	mod	0x01	修改一个不为脏的 tlb 条目
读 TLB 异常	tlbl	0x02	load 时 tlb 条目无效或不存在
写 TLB 异常	tlbs	0x03	store 时 tlb 条目无效或不存在
读地址异常	adEL	0x04	load 时地址不对齐
写地址异常	adES	0x05	store 时地址不对齐
系统调用	syscall	0x08	系统调用异常
断点	bp	0x09	执行断点指令
保留指令	ri	0x0a	执行了未实现的指令
算术溢出	ov	0x0c	定点数加减法溢出
陷阱	trap	0x0d	执行陷阱指令

表 4: 支持的异常

3 soc 设计

3.1 总体结构

整体架构图如图 9 所示, cpu 引出一个 AXI 接口, 通过 AXI interconnect 连接外设。

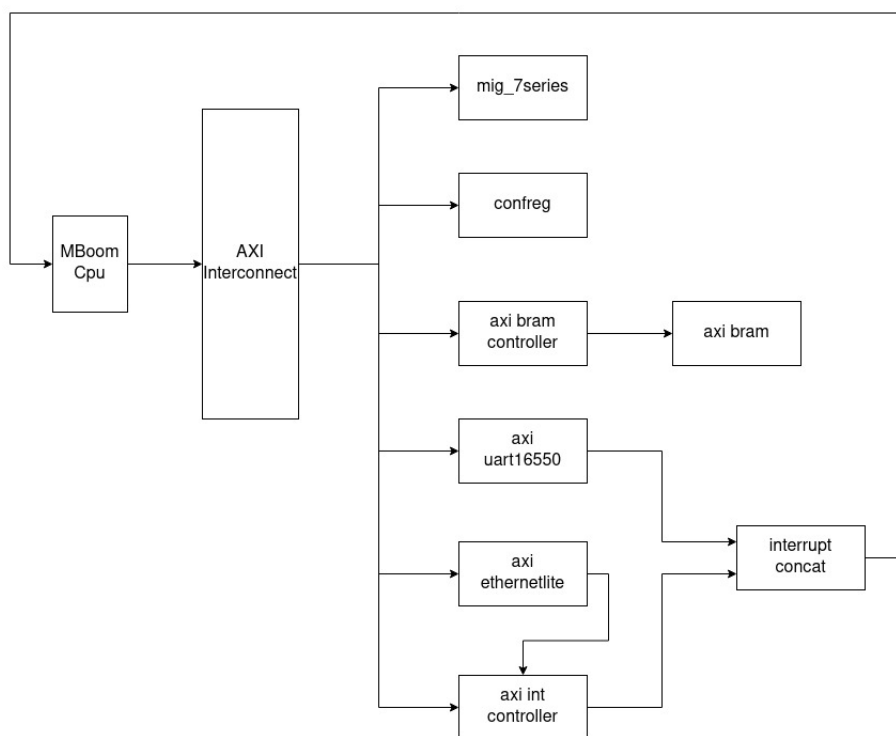


图 9: soc 总体架构

3.2 外设支持

目前 soc 支持如下的几种外设

- DRAM: 板载 128MiB DDR3 SDRAM 作为系统主存

- 片内 RAM: 板载 128KB BootROM 用于存放 uboot
- 以太网: 使用 Xilinx IP 构建的以太网控制器
- UART: 16550 兼容的串口控制器

对于板上外设的地址映射标如表 5 所示

设备	起始地址	结束地址	有效大小
axi bram	0x1FC00000	0x1FC3FFFF	256K
axi ethernetlite	0x1FF00000	0x1FFFFFFF	1M
axi intc	0x1FB00000	0x1FB0FFFF	64K
axi uart16550	0x1FE40000	0x1FE4FFFF	64K
confreg	0x1FAF0000	0x1FAFFFFFF	64K
mig_7series	0x00000000	0x07FFFFFF	12M

表 5: Soc 外设地址映射

4 性能优化

4.1 推测唤醒

Load 指令和其后的指令对其的依赖关系是程序中常见的依赖关系，同时也是 CPU 的常见瓶颈之一。尽可能优化 Load-Use 能够很大程度上优化 CPU 性能。

在我们的设计中，考虑到结构的复杂性，我们没有采用激进的 Load Forwarding 策略，因此理想情况下，Load to Use 为两个气泡，即 Load 指令的 write back 阶段和下一条指令的 read prf 阶段。

为了实现这个效果，我们参考了第六届参赛队伍 zencove 的思路，在 mem1 阶段提前发起唤醒请求，在 mem2 阶段如果出现 Cache 缺失或者 UnCache 时将流水线阻塞，等待数据最终返回再放行指令。

4.2 bypass 通路

由于整数指令流水线长度是一定的，因此比较适合做激进的 bypass 策略。我们实现了整数指令的完整 bypass 通路，有 RAW 依赖的整数指令之间能够完全背靠背执行，没有任何气泡。

4.3 微结构调整

为了减少复杂的结构对 CPU 频率的影响，我们对其中一些单元进行了结构优化。

以 Freelist 为例，riscv-boom 最初采用的是基于 bit array 的方案，因此在 rename 阶段需要多个优先编码器获取可以使用的物理寄存器号，这导致了项目早期 rename 阶段一直在我们的关键路径上。为了解决这个问题，我们使用空间换时间的策略，直接采用双端口先进先出队列来实现发热 freelist，这样就可以以很小的开销取出可用的寄存器。

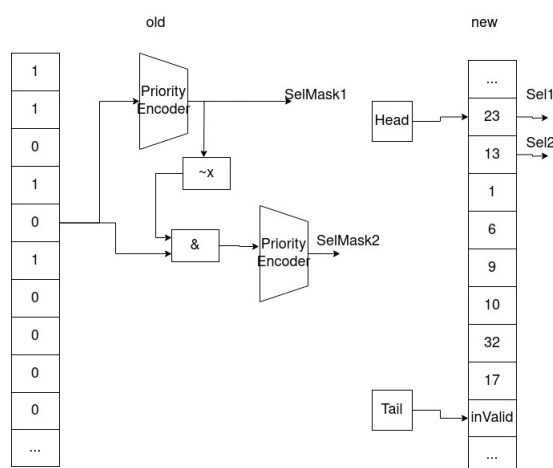


图 10: freelist 选取算法

5 系统软件

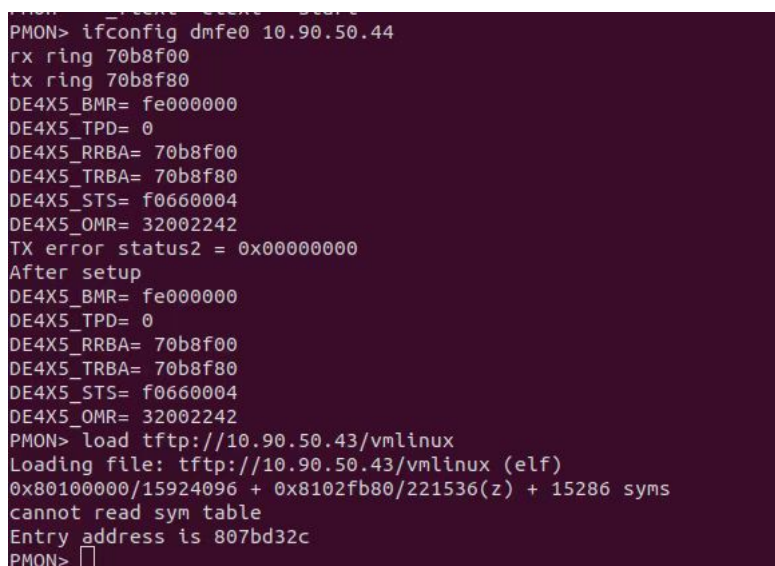
5.1 监控程序

监控程序是系统测试的内容，通过监控程序可以查看程序代码，运行程序，输入汇编代码数据等。同时监控程序附带 8 测试程序用于检验 cpu 的正确性。

5.2 引导程序

5.2.1 pmon

pmon 程序是由比赛官方提供的引导程序，相比与一般的引导程序，pmon 的功能更加强大，可以设置 ip，使用 tftp 服务，用于引导 linux 等操作系统，运行 pmon 结果如图 12 所示



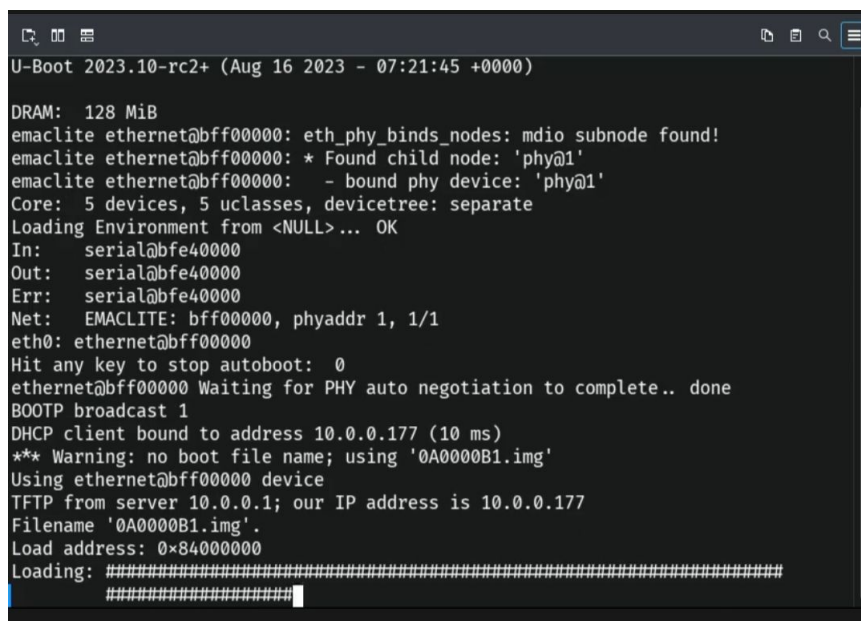
```
PMON> ifconfig dnfe0 10.90.50.44
rx ring 70b8f00
tx ring 70b8f80
DE4X5_BMR= fe000000
DE4X5_TPD= 0
DE4X5_RRBA= 70b8f00
DE4X5_TRBA= 70b8f80
DE4X5_STS= f0660004
DE4X5_OMR= 32002242
TX error status2 = 0x00000000
After setup
DE4X5_BMR= fe000000
DE4X5_TPD= 0
DE4X5_RRBA= 70b8f00
DE4X5_TRBA= 70b8f80
DE4X5_STS= f0660004
DE4X5_OMR= 32002242
PMON> load tftp://10.90.50.43/vmlinux
Loading file: tftp://10.90.50.43/vmlinux (elf)
0x80100000/15924096 + 0x8102fb80/221536(z) + 15286 syms
cannot read sym table
Entry address is 807bd32c
PMON> 
```

图 11: 运行 pmon

5.2.2 uboot

uboot 程序是出名的操作系统引导程序，通过 uboot 可以引导 linux 等操作系统，在我们的 soc 中使用 uboot 进行 linux 的引导。

我们通过给 uboot 编码参数的方式，让 soc 上电后自动向主机上的 tftp 服务器请求 Linux 内核。在主机 tftp 服务器可以正常工作时，可以模拟开机自动引导进入操作系统的效果。



```
U-Boot 2023.10-rc2+ (Aug 16 2023 - 07:21:45 +0000)

DRAM: 128 MiB
emac_lite ethernet@bfff0000: eth_phy_binds_nodes: mdio subnode found!
emac_lite ethernet@bfff0000: * Found child node: 'phy@1'
emac_lite ethernet@bfff0000: - bound phy device: 'phy@1'
Core: 5 devices, 5 uclasses, devicetree: separate
Loading Environment from <NULL>... OK
In: serial@bfe40000
Out: serial@bfe40000
Err: serial@bfe40000
Net: EMAC_LITE: bfff0000, phyaddr 1, 1/1
eth0: ethernet@bfff0000
Hit any key to stop autoboot: 0
ethernet@bfff0000 Waiting for PHY auto negotiation to complete.. done
BOOTP broadcast 1
DHCP client bound to address 10.0.0.177 (10 ms)
*** Warning: no boot file name; using '0A0000B1.img'
Using ethernet@bfff0000 device
TFTP from server 10.0.0.1; our IP address is 10.0.0.177
Filename '0A0000B1.img'.
Load address: 0x84000000
Loading: #####
```

图 12: uboot 自动引导 Linux

5.3 操作系统程序

5.3.1 ucore

ucore 是清华大学提供的教学操作系统，并由清华的同学为其增加了对 mips 架构的支持。

5.3.2 linux

Linux 是最为著名的开源操作系统内核，有丰富的软硬件支持。我们选择了 Linux 最新的主线版本 (v6.3) 进行移植，使其能够在我们的 CPU 上启动并正常运行。

在 Linux 内核上我们启用了 NFS，因此可以和 host 主机互相传递文件。我们也内置了很多比较复杂的应用，如 lynx 文本浏览器等，因此可以在 Linux 进行网上冲浪。

```

Mem: 27804K used, 93064K free, 21612K shrd, 0K buff, 21612K cached
CPU: 19% usr 52% sys 0% nic 27% idle 0% io 0% irq 0% sirq
Load average: 0.37 0.29 0.12 1/31 71

```

PID	PPID	USER	STAT	VSZ	%VSZ	%CPU	COMMAND
71	68	root	R	1972	2%	67%	top
10	2	root	SW	0	0%	3%	[ksoftirqd/0]
15	2	root	SW	0	0%	3%	[kcompactd0]
68	1	root	S	1972	2%	0%	-sh
1	0	root	S	1960	2%	0%	init
45	1	root	S	1960	2%	0%	/sbin/syslogd -n
49	1	root	S	1956	2%	0%	/sbin/klogd -n
7	2	root	IW	0	0%	0%	[kworker/u2:0-ev]
11	2	root	SW	0	0%	0%	[kdevtmpfs]
18	2	root	IW	0	0%	0%	[kworker/0:1-eve]
2	0	root	SW	0	0%	0%	[kthreadd]
24	2	root	IW	0	0%	0%	[kworker/u2:2-ev]
13	2	root	SW	0	0%	0%	[oom_reaper]
12	2	root	IW<	0	0%	0%	[inet_frag_wq]
14	2	root	IW<	0	0%	0%	[writeback]
6	2	root	IW<	0	0%	0%	[kworker/0:0H]
8	2	root	IW<	0	0%	0%	[mm_percpu_wq]
17	2	root	IW<	0	0%	0%	[blkcg_punt_bio]
19	2	root	IW<	0	0%	0%	[rpciod]
16	2	root	IW<	0	0%	0%	[kblockd]

图 13: linux 上运行 top

```

# MIPS32 ISA - Search (p5 of 19)
Coprocessor (an essential part of the processor that is
implementation-defined in MIPS I-V), CP1 is an optional
floating-point unit (FPU) ... See more
Application-specific extensions
Registers See more
Calling conventions
MIPS has had several calling conventions, especially on the 32-bit
platform.
The O32 ABI is the most commonly-used ABI, owing to its status as
the original System V ABI for MIPS. It is strictly stack-based,
with only four registers $a0-$a3 available to pass ... See more
Simulators
Open Virtual Platforms (OVP) includes the freely available for
non-commercial use simulator OVPSim, a library of models of
processors, peripherals and platforms, and APIs which enable users
to develop their own models. The models in the library are open
source, ... See more
Design
MIPS is a modular architecture supporting up to four coprocessors
(CP0/1/2/3). In MIPS terminology, CP0 is the System Control
-- press space for next page --
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
H)elp O)ptions P)rint G)o M)ain screen Q)uit /search [delete]=history list

```

图 14: linux 上运行 lynx

6 测试结果

6.1 功能测试

89 个测试点均通过，短延时情况下测试通过如图 15 所示

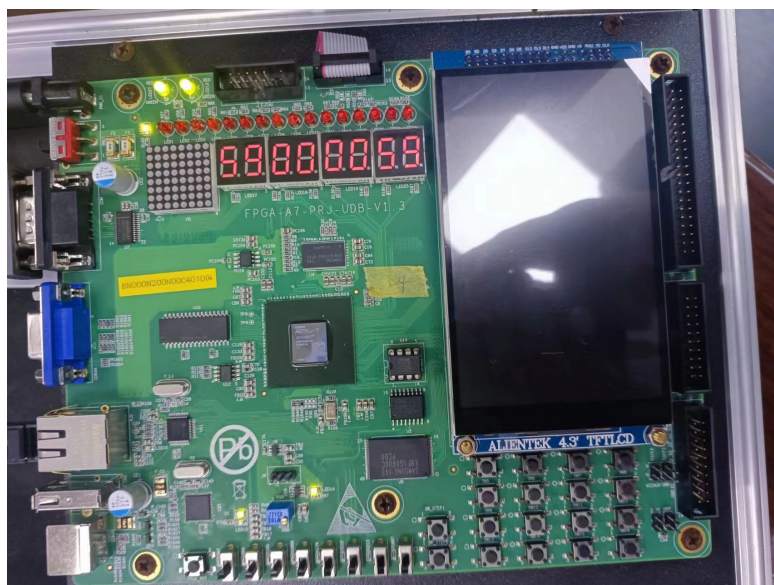


图 15: functest 通过

6.2 性能测试

性能测试各测试点得分情况如表 6 所示

序号	测试名	得分
1	bitcount	93.20198313
2	bubble_sort	77.54454172
3	coremark	70.45092463
4	crc32	102.4128701
5	dhrystone	82.13962439
6	quick_sort	59.52060916
7	select_sort	91.91479931
8	sha	99.59818399
9	stream_copy	82.68990075
10	stringsearch	84.22546711

表 6: 性能测试各测试点得分

最终性能得分为 83.387 分

6.3 系统测试

6.3.1 监控程序

运行系统所带的 8 个程序测试程序均通过，如图 16 所示

```
1 sudo python3 term.py -s /dev/ttyUSB0 -b 57600
[sudo] password for bot:
MONITOR for MIPS32 - initialized.
>> g
>>addr: 0x8000300c

elapsed time: 0.397s
>> g
>>addr: 0x8000303c

elapsed time: 0.513s
>> g
>>addr: 0x800030c4

elapsed time: 4.348s
>> g
>>addr: 0x8000315c
OK
elapsed time: 0.001s
>> g
>>addr: 0x80003180

elapsed time: 7.255s
>> g
>>addr: 0x800031b4

elapsed time: 6.801s
>> g
>>addr: 0x800031fc

elapsed time: 5.441s
>> g
>>addr: 0x80003228

elapsed time: 11.790s
>>
```

图 16: 通过系统测试

6.3.2 记忆游戏

通过记忆游戏，两次的输入完全相同 LED 显示 8 并亮绿灯。



图 17: 通过记忆游戏

7 总结与展望

项目的设计上收到了诸多的往年开源项目的启发，LSU 部分推测唤醒机制受到了 zen-cove(第六届清华大学参赛作品) 的启发，工程架构上参考了 amadeus-mips(第三届华东师范大学参赛作品，soc 的设计上参考了 cdim(第六届重庆大学参赛作品)。

半年来经历无数次调试一步一个脚步，从通过功能测试到通过性能测试，再到第一次启动 pmon 和 linux，终于实现了我们最初的目标：在自己的 cpu 上运行 linux。

当然我们仍有很多不足之处。由于早期设计时考虑不周，导致我们的 CPU 结构上有很多遗留下的弊病。这些弊病严重影响了我们进一步的 CPU 优化进程。

以下是我们 CPU 上能够继续改进的几个方面。

- bypass 通路: 由于早期执行流水并没有充分考虑 bypass 通路，导致 ALU 外的单元的设计添加 bypass 通路非常困难。在最终的版本中，我们甚至没有通过 prf 进行 bypass 的策略。没有充分的 bypass 严重影响了我们最终的性能。
- 分支预测: 我们采用的分支预测算法中的局部历史在使用时很大概率是错误的历史，这也比较严重地影响了我们的 CPU 在某几个测试上的性能。
- checkpoint 机制: 我们虽然实现了完整的 checkpoint 机制，但考虑了复杂性，最终版本我们并没有在最终版本中加入 checkpoint。

我们会在今后的学习中对自己的不足之处进行改进，希望能在性能优化这条路上更进一步。

最后感谢周健老师的指导以及计算机学院的大力支持，同时感谢徐梓晨，张承元，李涵，刘安珉等学长给予的指导和帮助。

参考文献

- [1] 谭志虎 秦磊华 吴非 肖亮. 计算机组成原理 北京: 清华大学出版社北京. 2021 年
- [2] 姚永斌 超标量处理器设计. 北京: 清华大学出版社. 2014 年