基于 MIPS 指令集的乱序四发射 CPU 设计

哈尔滨工业大学（深圳）1 队

胡博涵、黎庚祉、施杨、王世焜

2020 年 8 月 17 日

目录

设计变更说明

初赛完成后，我们为 CPU 增加了 TLB 支持并加入了精度更高的二级分支预测器。我们也优化了部分关键路径，使得 CPU 的频率可以达到 90MHz。

同时，我们修复了特权指令（MFC0，MTC0，ERET）等类型指令若位于延迟槽中，可能会被错误的清除的 bug。并增设了部分 CP0 寄存器，以达到支持 TLB 的目的，实现了 TLB 相关异常，但遗憾的是，由于添加 TLB 之后，片上布线资源不足，导致时序无法收敛（综合报告提示资源占用过高），因此，最终提交包中并没有包含 TLB。

为提升指令级层面的并行（ILP）程度，我们在原有分组的旁路网络基础上，将 Load/Store 的唤醒对其他指令的唤醒进行了提前，使得具有相关性的算数指令与访存指令能够尽量并行地执行。同时，为了保证频率，未对旁路网络进行进一步的扩充。经过后期对测试程序的分析，这或许是影响 IPC 得分的重要因素之一。

一、设计简介

在本次大赛中我们所提交的设计是一个动态 10 级乱序四发射的 CPU，实现了大赛要求的基本功能，初赛提交版本频率 75MHz，性能得分 38.8 分；决赛提交版本频率 90MHz，与初赛使用的 GS132 的基准的运行时间比值为 49，IPC 比值为 27.5；主要特色包含但不限于：

- 动态十级超标量乱序执行流水线（取指宽度为 2，发射数 4，提交数 2）
- 2 个整数执行单元，1 个存储加载单元和 1 个乘除法运算单元
- 3 个容量为 8 的压缩的指令队列（保留站）
- 4 路组 ICache（4KB）和 2 路组 DCache（16KB）
- 32 项全相联分支预测缓冲
- 基于统一物理寄存器的重命名机制
- 高度流水化乘除法单元
- 非阻塞数据访存模块

二、设计方案

（一）总体设计思路

1. 流水线结构

完整的流水线结构图如下：

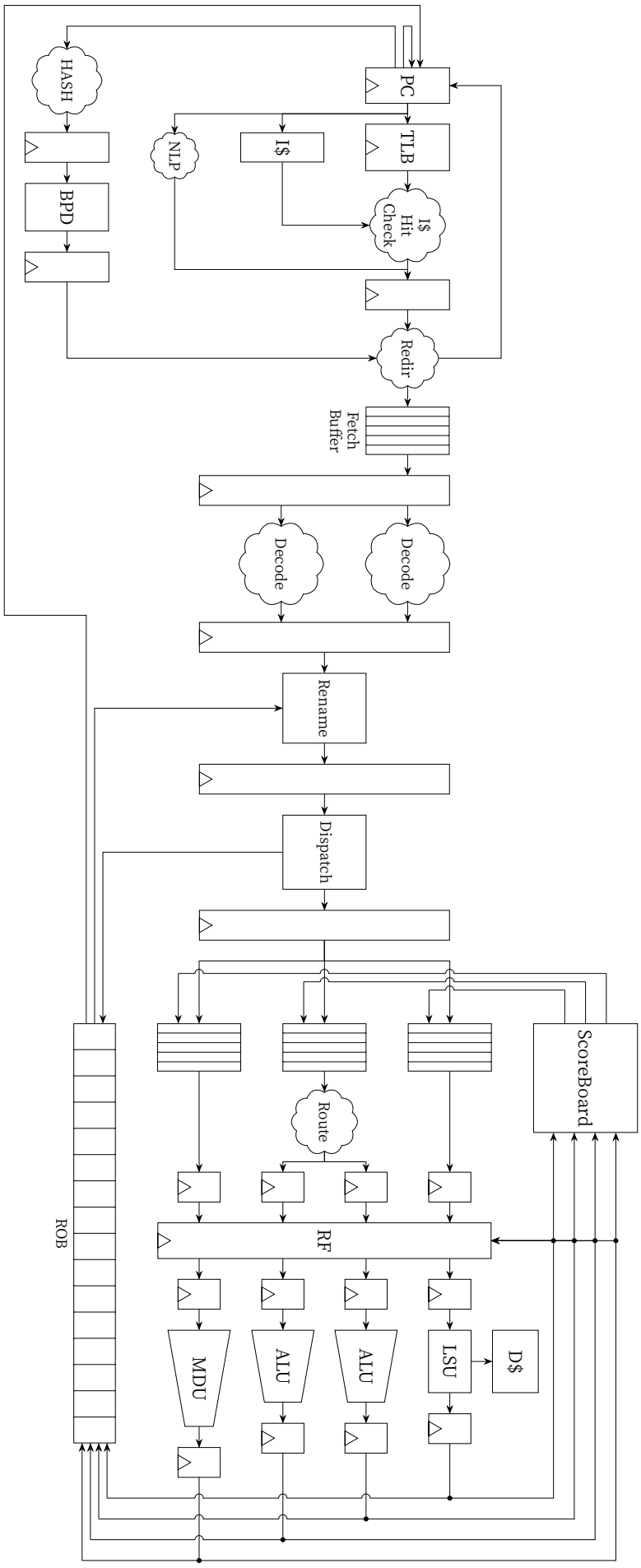
HASH
PC
NLP
I$
TLB
BPD
I$ Hit Check
Redir
Fetch Buffer
Decode
Decode
Rename
Dispatch
ScoreBoard
Route
ROB
RF
MDU
ALU
ALU
LSU
D$

图 1: 整体框架

设计的 CPU 包含 13 级流水，其中取指阶段分为 3 级，分别为 PCGen（新 PC 生成逻辑）、TagCheck（Cache 命中检查）和 Predecode（分支预解码）；取指阶段后接指令缓冲，用于缓解前端取指和后端执行效率带来的差异问题，同时一定程度上使前端和后端解耦，方便模块化的开发。

流水线前端 流水线前端包括 PC 生成逻辑、Cache、分支预测器、指令 Buffer 和译码逻辑部分。流水线前端的主要工作是保证指令的稳定供给。前端综合后端的反馈、分支预测的结果和各类异常信息，决定生成的下一个 PC，并访问 Cache，取出指令。若 Cache 未命中，则前端取指暂停，阻塞 Cache 访问，直到 Cache 从下级存储器中得到相应的猝发指令序列。若后端在执行过程中发生了暂停，则前端的取指并不暂停，而是继续根据预测结果和重定向结果进行取指，直到指令缓冲满。前端每个周期最多同时解码、给后端供给 2 条指令的译码结果（设计中称为”uOP”）。（由于时间有限，本处理器并未实现完整的分支预测机制，对流水线效率造成了一定的影响。）

重命名和分派 重命名阶段主要解决的是指令之间的“假相关”问题，通过对每一条具有目的寄存器的指令的**目的寄存器**进行重命名，使得具有 WAR 和 WAW 相关的指令可以乱序执行，进而提高指令之间的并行程度。分派阶段将指令路由到对应的发射队列中，这是处理器顺序执行的最后一个阶段。由于流水线的设计是乱序执行、顺序提交，所以在分派阶段，为了记录指令的顺序关系，需要在这个阶段写入重定序缓存（ROB）。

发射 发射阶段是处理器顺序执行到乱序执行的开端。此部分主要由发射队列、仲裁逻辑和计分板（Scoreboard）构成。指令顺序地进入发射队列，监视计分板中对应操作数的状态。如果计分板显示指令的操作数已经准备好，那么队列中该指令旧可以提出申请，由仲裁逻辑按照一定的原则，决定该指令是否可以发射到执行单元执行，而不需要等待前序指令执行完毕。在发射时，被发射的指令还会对其他有相关性的指令进行唤醒，通知这些指令其操作数即将准备就绪。

执行与写回 执行部分包括物理寄存器文件（Physical Register File, PRF）、整数执行单元、访存单元和乘除法单元和旁路网络。物理寄存器文件存储所有推测状态和非推测状态的执行结果。执行的指令使用物理寄存器号访问相应的物理寄存器，读出对应的值，或通过旁路网络获取相应的操作数。为了不影响频率，并未在不同的功能单元之间架设完整的旁路网络，而是仅在两个整数执行单元之间架设旁路网络，以达到整数指令的背靠背执行的目的。

提交 流水线的最后一个阶段，是指令提交阶段。这一阶段，是从乱序执行到顺序提交的重排序过程，关键的部件是重排序缓冲（ROB）。指令在分派阶段，按顺序进入重排序缓冲，在执行阶段，被执行单元标记为完成，直到指令达到 ROB 尾部的时候，才可以被提交，从推测态达到确定态，最终离开流水线。如果在 ROB 尾部的指令发生了异常或分支预测失败，则会冲刷 ROB 和流水线中剩余的内容，从正确的 PC 开始取指，并恢复重命名映射关系。

（二）CPU 内部模块设计

1. 取指模块设计

取指模块主要负责取回指令、将其缓存下来并传递给后端，主要包含以下模块：

- 下址生成：综合 PC 的值、分支预测结果与后端重定向结果，生成下一个 PC

- ICache：缓存已取回的指令
- 分支预测与预解码：通过两个分支预测器以及与解码信息进行分支预测
- 指令缓冲队列：用于匹配前后端执行速度
- 指令解码模块：用于解码指令

其模块结构如下：

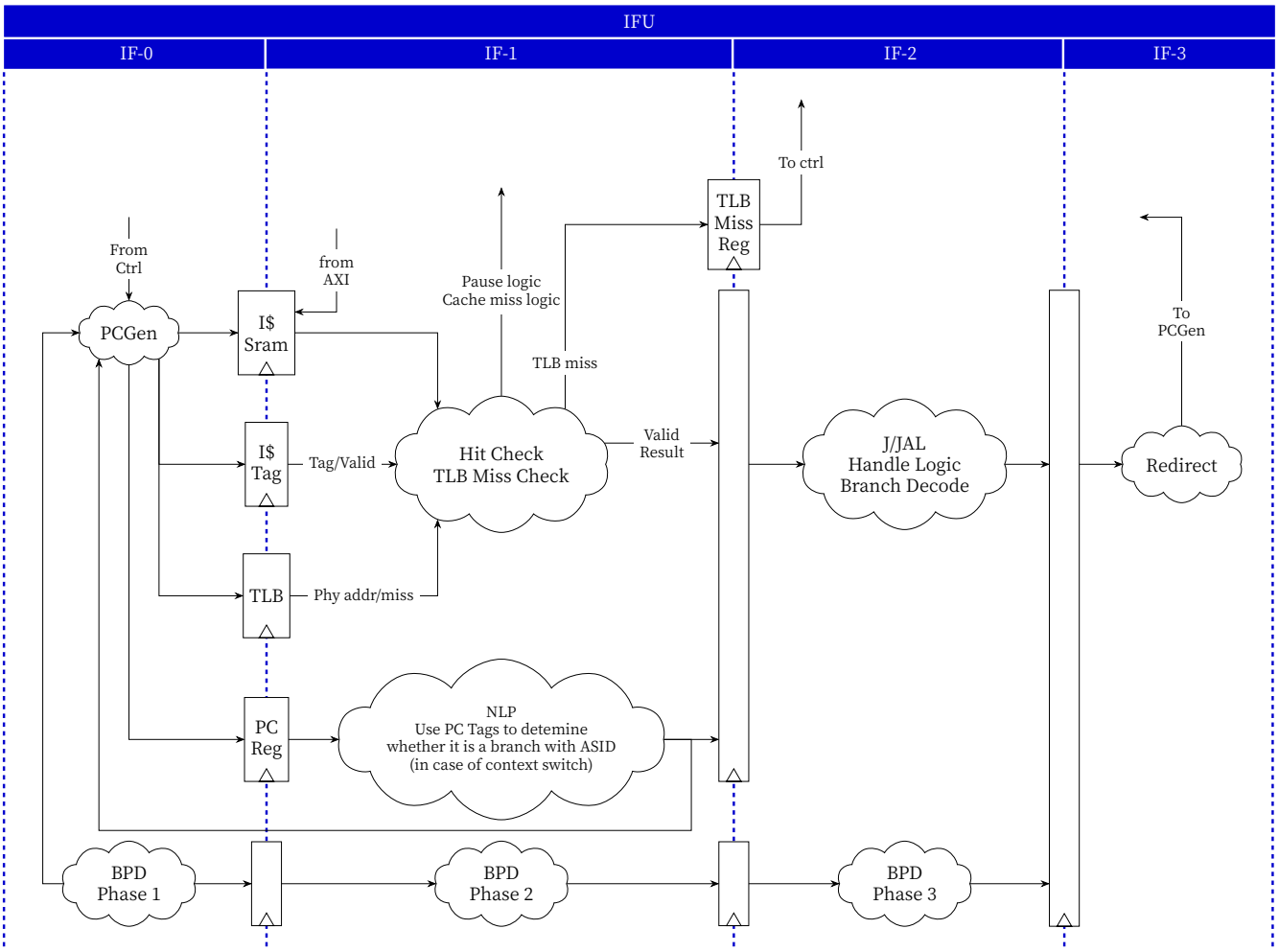

图 2: 取指模块框架

各模块详细内容如下：

通用数据接口 通用数据接口中包含了指令的大部分信息；通过通用数据接口，指令在流水线中向后传递，并在这个过程中由流水线各级逐渐完善各个数据域。接口内容如下：

信号名称	信号来源	信号意义
inst	ICache	指令内容
pc	下址生成模块	指令地址
isBr	预解码器	是否条件跳转指令
isJ	预解码器	是否无条件跳转指令
isDs	指令缓冲	是否延迟槽指令
valid	流水线各级	指令是否有效
nlpInfo	下址预测器	下址预测器做出的预测信息
target	下址预测器	预测目标地址
bimState	下址预测器	预测时对应饱和计数器内容
taken	下址预测器	是否跳转
target	下址预测器	该预测有效
bpdInfo	二级分支预测器	二级分支预测器做出的预测信息
RESERVED		
predTaken	指令预解码与前端重定向模块	前端最终有无决定跳转
predAddr	指令预解码与前端重定向模块	前端决定跳转的地址
jBadAddr	指令预解码与前端重定向模块	指令是否跳转到未对齐的地址

表 1: 通用数据接口内容（译码前）

信号名称	信号来源	信号意义
pc	下址生成模块	地址
id	重定序缓存	在重定序缓存中的编号
uOP	译码模块	译码出的微指令
aluType	译码模块	运算类型，用于决定 ALU 的操作
rs_type	译码模块	类型，用于决定指令发往哪个指令队列
op0LAddr	译码模块	源操作数 0 的逻辑寄存器
op0PAddr	寄存器重命名	源操作数 0 的物理寄存器
op0re	译码模块	源操作数 0 读使能
op1LAddr	译码模块	源操作数 1 的逻辑寄存器
op1PAddr	寄存器重命名	源操作数 1 的物理寄存器
op1re	译码模块	源操作数 1 读使能
dstLAddr	译码模块	目标操作数的逻辑寄存器
dstPAddr	寄存器重命名	目标操作数的物理寄存器
dstPStale	寄存器重命名	目标操作数可以释放的物理寄存器
dstwe	译码模块	目标操作数写使能
imm	译码模块	立即数
branchType	译码模块	跳转类型
branchTaken	算数运算单元	是否跳转
branchAddr	算数运算单元	跳转目标地址
predTaken	指令预解码与前端重定向模块	是否预测跳转
predAddr	指令预解码与前端重定向模块	预测跳转目标地址
nlpBimState	下址预测器	预测该指令时，下址预测器的饱和计数器值
causeExc	流水线各级	指令是否导致异常
exception	流水线各级	指令异常类型
BadVaddr	流水线各级	未对齐访存时的目标地址
isDS	指令缓冲	是否延迟槽指令
isPriv	译码模块	是否特权指令
cp0Addr	译码模块	需要访问的 CP0 寄存器地址
cp0Data	译码模块	需要存入 CP0 的值
cp0Sel	译码模块	需要访问的 CP0 sel 值
busy	流水线各级	指令忙
valid	流水线各级	指令有效
committed	重排序缓存	指令已经提交

表 2: 通用数据接口内容（译码后）

下址生成 下址生成模块综合 PC 的值、分支预测结果与后端重定向结果，生成下一个 PC，优先级为后端重定向 > 分支预测结果 >PC+8。

ICache ICache 被置于流水线中，接收 PC 的值并且返回取指结果；并在未命中时发起访存请求。

设计说明 本设计的 ICache 大小为 4KB，四路组相联，替换策略为 PLRU，每轮取两条指令。ICache 的设计目标是在命中时做到流水化的一周期连续返回，在未命中时从 AXI 接口处取回整个 CacheLine，并在取回成功当拍将指令传递到后级。同时，在设计时我们采用了 VIPT（Virtually Index, Physically Tag)。为此，我们采用了同时向 TLB 和 ICache 发送地址，再在下一拍进行 tag 的比对。同时，由于 AXI 事务不可终止，在清空流水线时也需要一定的特殊处理，为此，iCache 的控制由状态机实现，其状态转移图如下：

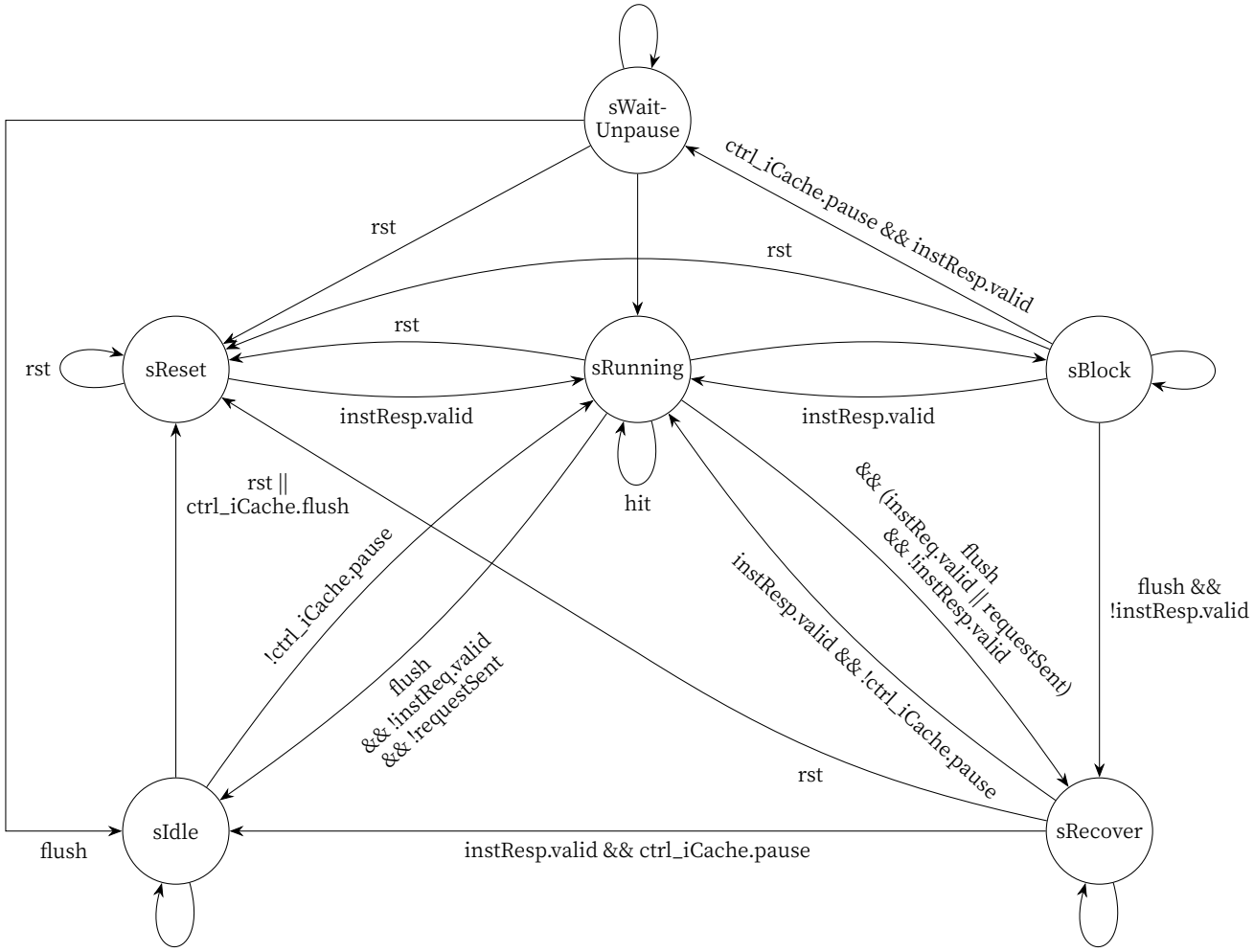

图 3: iCache 的状态转移图

其中，sRunning 代表正在正常运行；sBlock 代表此前指令未命中，正在访存以获得结果；sRecover 代表在访存时流水线清空，需要等待访存事务完成才能处理新的取指；sIdle 代表流水线清空后等待前端地址抵达 iCache；sWaitUnpause 代表在访存时流水线遭暂停，需要暂存访存结果等待暂停结束；sReset 则代表 ICache 整体重置，清空内部有效位等各个数据。

输入输出信号说明

信号名称	信号方向	信号来源/目标	信号意义
clk	输入	顶层模块	时钟
rst	输入	顶层模块	重置
ctrl_iCache			
pauseReq	输出	控制模块	暂停请求
pause	输入	控制模块	暂停流水线
flush	输入	控制模块	清空流水线
iCache_tlb			
virAddr	输出	TLB	指令虚拟地址
phyAddr	输入	TLB	指令物理地址
instReq			
pc	输出	AXI 接口	CacheLine 首地址
valid	输出	AXI 接口	握手信号
ready	输入	AXI 接口	握手信号
instResp			
cacheLine	输入	AXI 接口	四个指令字
valid	输入	AXI 接口	握手信号
ready	输出	AXI 接口	握手信号
regs_iCache			
PC	输入	上级流水线	请求的指令地址
onlyGetDS	输入	上级流水线	下次取指时，丢弃第二条指令
inst0	输入	上级流水线	指令信息（标准接口）
inst1	输入	上级流水线	指令信息（标准接口）
iCache_regs			
inst0	输出	下级流水线	指令信息（标准接口）
inst1	输出	下级流水线	指令信息（标准接口）

表 3: iCache 输入输出信号说明

下址预测器　下址预测器是一个很小的分支预测器，用于在一周期之内预测出当前 PC 对应的两条指令是否包含了跳转指令、其跳转方向与跳转地址。

设计说明　下址预测器为全相联结构，共有 16 项，每次替换最早进入的一项，预测基于两位饱和计数器。设置下址预测器的原因在于，由于较高精度的分支预测器需要至少一个周期，为了避免错误 PC 导致不必要的访存或者流水线中出现气泡，因此设置一个简单但快速的下址预测器，尽量使得 iCache 访存不会访问不正确的地址。

输入输出信号说明

信号名称	信号方向	信号来源/目标	信号意义
clk	输入	顶层模块	时钟
rst	输入	顶层模块	重置
regs_nlp			
PC	输入	IF-0	需要预测的 PC
nlp_if0			
nlpInfo0	输出	IF-0	对指令 0 的预测信息。
nlpInfo1	输出	IF-0	对指令 1 的预测信息。
target	输出	IF-0	跳转目标地址。
bimState	输出	IF-0	做出预测时的 BIM 状态。
taken	输出	IF-0	是否跳转。
valid	输出	IF-0	预测有效。
if3_nlp	输入	IF-3	来自 IF-3 的 NLP 更新信息
backend_nlp	输入	后端	来自后端的 NLP 更新信息
update	输入	IF-3/后端	NLP 的更新信息。
pc	输入	IF-3/后端	需要更新的 PC。
target	输入	IF-3/后端	PC 对应的目标。
bimState	输入	IF-3/后端	更新前的 BIM 状态。
shouldTake	输入	IF-3/后端	目标 PC 发生了一次跳转。
valid	输入	IF-3/后端	更新有效。

表 4: 下址预测器输入输出信号说明

二级分支预测器 已实现基于 GShare 的全局历史预测器和基于 2Bit 局部历史的预测器，同时尝试了 TAGE 分支预测，但时间所限，二级分支预测器未上板验证，详细仿真结果见（三）性能分析结果及反思。

指令预解码与前端重定向模块

设计说明 指令预解码与重定向需要综合各个信息源，并尝试将前端引导向正确的分支。具体地说，二级分支预测器的结果可以覆盖下址预测器的结果；而预解码出的结果则可以覆盖二级分支预测器的结果。

输入输出信号说明

信号名称	信号方向	信号来源/目标	信号意义
clk	输入	顶层模块	时钟
rst	输入	顶层模块	重置
regs_if3			
inst0	输入	上级流水线	指令信息（标准接口）
inst1	输入	上级流水线	指令信息（标准接口）
bpd_if3			
RESERVED			
if3_regs			
inst0	输出	下级流水线	指令信息（标准接口）
inst1	输出	下级流水线	指令信息（标准接口）
if3_0			
redirect	输出	下址生成模块	使前端重定向
redirectPC	输出	下址生成模块	前端重定向目标
if3_nlp			
update			
pc	输出	下址预测器	需要更新的 PC。
target	输出	下址预测器	PC 对应的目标。
bimState	输出	下址预测器	更新前的 BIM 状态。
shouldTake	输出	下址预测器	目标 PC 发生了一次跳转。
valid	输出	下址预测器	更新有效。
ctrl_if3			
flushReq	输出	控制单元	清空前级流水线请求
flush	输入	控制单元	清空流水线
pause	输入	控制单元	暂停流水线

表 5: 指令预解码与前端重定向模块输入输出信号说明

指令缓冲 指令缓冲用于缓冲指令，使前后端解耦。

设计说明 由于多发射处理器的天然特性，后端执行的速度不是固定的。为了避免前端被频繁地暂停，有必要令前后端尽量解耦。同时，由于译码后指令字段容量增加，我们将指令缓冲放在了指令译码之前。指令缓冲每周期最多进入两条指令、放出两条指令。通过适当地在未对齐跳转指令前插入气泡，我们能够在输出时将所有跳转指令放置在低位，而将其延迟槽指令放置在高位，以便后续处理。为了降低流水线深度，当指令缓冲为空的时候，前端的指令可以直接“透传”给后级，无需等待一周期。

输入输出信号说明

信号名称	信号方向	信号来源/目标	信号意义
clk	输入	顶层模块	时钟
rst	输入	顶层模块	重置
ctrl_instBuffer			
pauseReq	输入	控制单元	队列满，请求暂停取指
ifu_instBuffer			
inst0	输入	上级流水线	指令信息（标准接口）
inst1	输入	上级流水线	指令信息（标准接口）
instBuffer_backend			
inst0	输出	下级流水线	指令信息（标准接口）
inst1	输出	下级流水线	指令信息（标准接口）
valid	输出	下级流水线	握手信号
ready	输入	下级流水线	握手信号
flush	输入	控制单元	清空缓冲

表 6: 指令缓冲输入输出信号说明

译码模块 译码模块用于译码。

设计说明 译码模块用于将指令码译为微指令码。在这个阶段，译码模块会根据译码的结果填充输出的各个数据段。需要注意的是，MULT 指令和 DIV 指令需要特殊处理。由于我们对 HI 和 LO 寄存器亦进行了重命名，因此我们需要将 MULT 和 DIV 拆成两个微指令，分别写 HI 和 LO 寄存器，以满足寄存器重命名的宽度要求。CPU 中实例化了两个并行的 decode 模块。

输入输出信号说明

信号名称	信号方向	信号来源/目标	信号意义
regs_decode			
inst	输入	上级流水线	指令信息（标准接口）
decode_regs			
uOP0	输出	下级流水线	指令信息（标准接口）
uOP1	输出	下级流水线	指令信息（标准接口）

表 7: 译码模块信号说明

2. 寄存器重命名

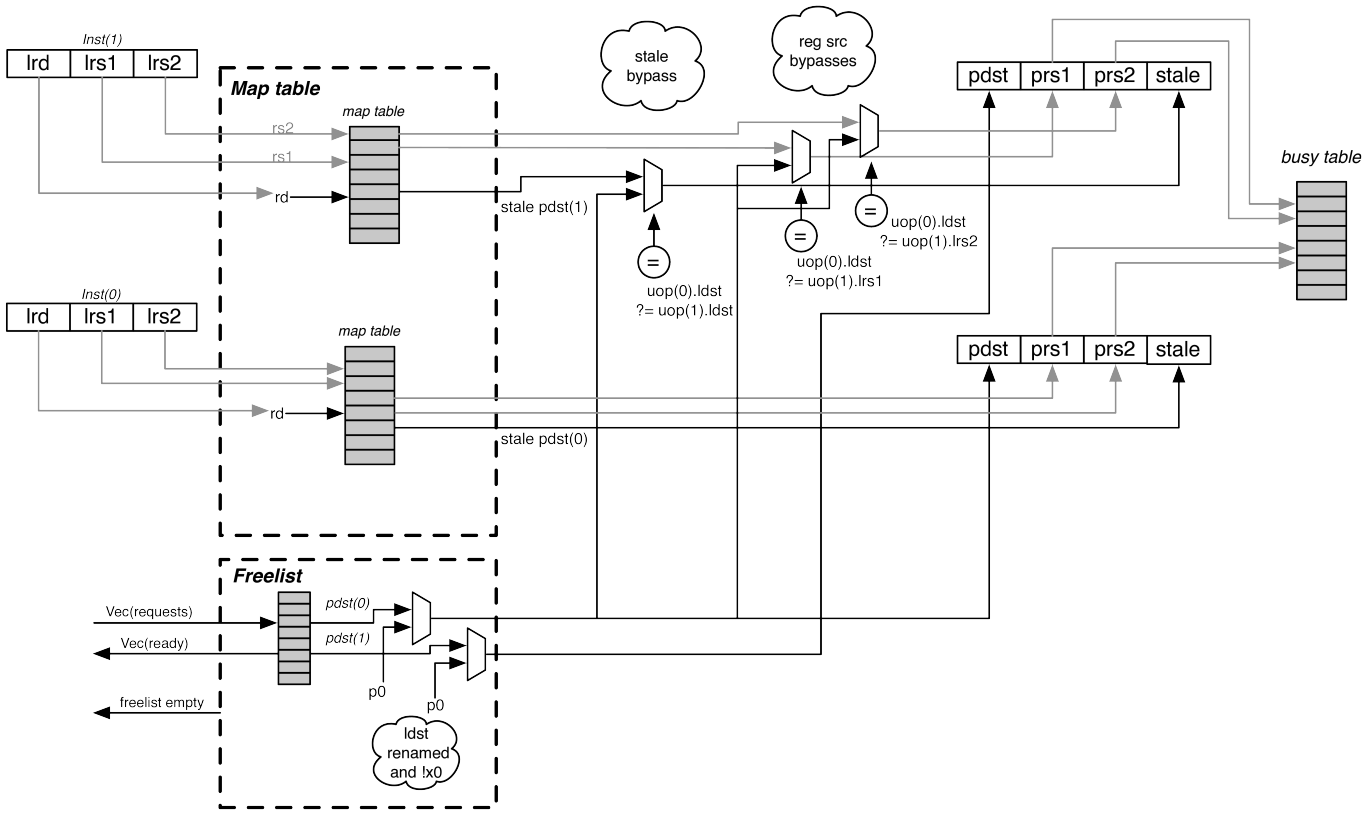

图 4: 重命名阶段，参见https://docs.boom-core.org/en/latest/sections/rename-stage.html

寄存器重命名阶段主要的任务是将指令对应的源寄存器映射到物理寄存器，并为有写回的指令分配新的物理寄存器。

在指令被分发到各个发射队列之前，需要对各个指令进行寄存器重命名操作。寄存器重命名操作的目的是消除指令之间的假相关性（WAW/WAR 相关），仅保留指令之间的真相关（RAW 相关）。寄存器重命名的思路是，对于存在目的寄存器的指令，一定会对其目的寄存器进行重命名。对于特殊的指令，例如存在两个目的寄存器的乘除法指令，将 Hi 寄存器规定为通用寄存器 33，Lo 寄存器规定为通用寄存器 34，则可以对 Hi,Lo 寄存器进行重命名。对于任何 CP0 寄存器，规定其状态必须属于确定态，不允许有猜测态的 CP0 寄存器，故 CP0 寄存器均不进行重命名操作。

重命名采用统一的物理寄存器重命名方式，使用一个映射表（Rename Map Table）来管理逻辑寄存器和物理寄存器之间的映射关系，同时再使用另一个“已提交”的映射表记录已经确定完成的指令的映射关系。当分支预测错误或异常发生时，清空流水线的同时会将已提交的映射关系恢复回去。

物理寄存器的分配和管理使用“空闲向量 (Free List)”的形式。向量共有 64 位，每一位都能表示每个物理寄存器被占用的情况。使用 2 个优先编码器，即可对空闲向量中的空闲寄存器号进行编码。其具体实现形式是：使用优先编码器，编码相应的物理寄存器之后，将对应的空闲位清除，再传递给第二个优先编码器。第二个优先编码器采取同样的操作进行编码，编码完成后将对应的寄存器号给出，同时将映射关系写入映射表中。

对于源操作数，查询映射表即可得到其对应的物理寄存器。

对于物理寄存器的释放，其原则是：当同一个逻辑寄存器被再次写入时，其原先对应的物理寄存器会被释放，因为这个逻辑寄存器的旧值已经不再被使用。因此，寄存器重命名的过程中，还会

读出旧的映射关系，在指令提交的时候，对应的旧的映射关系被覆盖，旧的物理寄存器被释放，可以分配给其他指令使用。

3. 指令分派

指令分派阶段的任务，是将重命名之后的指令发送到不同的发射队列，同时将指令顺序地写入重定序缓冲（ROB），这是 CPU 顺序执行的最后一个阶段。指令分派的内部实现了一个路由网络，根据译码阶段得到的指令类型，会定向地将指令写入相应的发射队列中，每个周期最多写入 2 条指令。

需要注意的是，在超标量处理器中，部分指令并不能在猜测的状态下执行，例如特权指令（如 CP0 指令和 ERET 指令等），在分派阶段，需要对其进行处理，具体的形式是，指令位于分派阶段时，如果存在特权指令，需要向前级发送暂停请求，等待 CPU 执行到非推测状态（ROB 为空）时，才可以将特权指令串行地送到发射队列中。即，分派阶段如果存在特权指令，其执行必须是串行的，一条指令必须等到另一条指令执行完之后，才可以发出。

4. 指令队列

指令队列是顺序执行和乱序执行的分界点。指令顺序地进入指令队列，经由仲裁逻辑，乱序地发射到执行单元进行执行。本处理器实现的 3 个指令队列，分别为整数队列、乘除法队列和访存队列。其中，整数队列一次可以接收 2 条指令，发射 2 条指令至 2 个整数执行单元；访存队列和乘除法队列一次可以接收 2 条指令，发射 1 条指令。

为方便实现 Oldest-First 仲裁原则，指令队列设计为压缩队列（Collasping Queue）。即：当有指令从队列中发出时，位于其后的指令会同时向前移动，队列中的所有指令永远占据队列头部的空间，不存在气泡。这种设计方便了队列满逻辑的实现和队列空间的分配，发射仲裁也显而易见地简单了，仅需要优先编码器就可以实现。

5. 发射仲裁机构

发射仲裁是指令乱序执行的开始。在这个阶段，操作数都已经准备好的指令即可以被仲裁逻辑选中发射到其相应的功能单元中执行。发射仲裁机构和发射队列相连接，每个发射队列都有一个相应的仲裁机构。发射队列输出一个 `ready` 向量，对应的位表示队列中对应位置的指令是否已经准备好发射。仲裁逻辑非常简单，只需要 1 个（对于整数队列，发射数为 2，需要 2 个）优先编码器即可以完成仲裁。

对于整数指令，由于整数运算单元一个周期能够完成一条指令，所以无需关心整数运算单元是否繁忙，只要其操作数准备好了，就可以发射。

对于访存指令，其仲裁存在两个原则：一是为了避免内存的数据相关，规定所有的 `store` 指令必须在处于队列首部的时候（保证其之前的 `load` 指令都已经进入流水线）之后，才可以发射到访存单元进行执行。二是由于实现的是非阻塞 Cache，其内部队列满的时候，不能再向其发射指令。

对于乘除法运算指令，由于乘除法单元是高度流水化的，理论上可以一周期发射一条乘除法指令而不必关心乘除法单元是否有指令在执行。但乘除法指令由于有 2 个目的寄存器，并且目的寄存器已经被重命名至物理寄存器，而为乘除法指令单元分配的仅有 1 个写端口，所以乘除法指令需要 2 个周期，分别进行 Hi,Lo 寄存器的写回，在一定的情况下，需要阻止队列发射乘除法指令。

6. 物理寄存器堆

本处理器采用的是使用统一的物理寄存器进行寄存器重命名的方式，因此，整个处理器中仅存在一个物理寄存器堆。物理寄存器堆的大小为 64，具有 4 组 8 个读端口和 4 个写端口（分别对应 2 个整数单元、1 个乘除单元和 1 个访存单元）。其中 Hi，Lo 寄存器也会被重命名至物理寄存器堆。使用统一的物理寄存器堆进行重命名，是为了避免跟踪指令所需的操作数存在于哪个寄存器堆中，同时也为了恢复方便。在分支预测失败恢复或是进行异常处理时，完全不用对寄存器堆进行任何操作，仅需恢复原有的映射关系和空闲列表即可。

7. 乘除法执行单元设计

乘除法执行单元执行乘除法。

设计说明　我们观察到，部分性能测试程序中存在大量的乘除法指令。这样一来，就很有必要使乘除法单元流水化，为此，我们使用了 Xilinx 的 IP 核实现乘除法。

8. LSU 模块设计

LSU 模块负责数据的读写，其结构图如下：

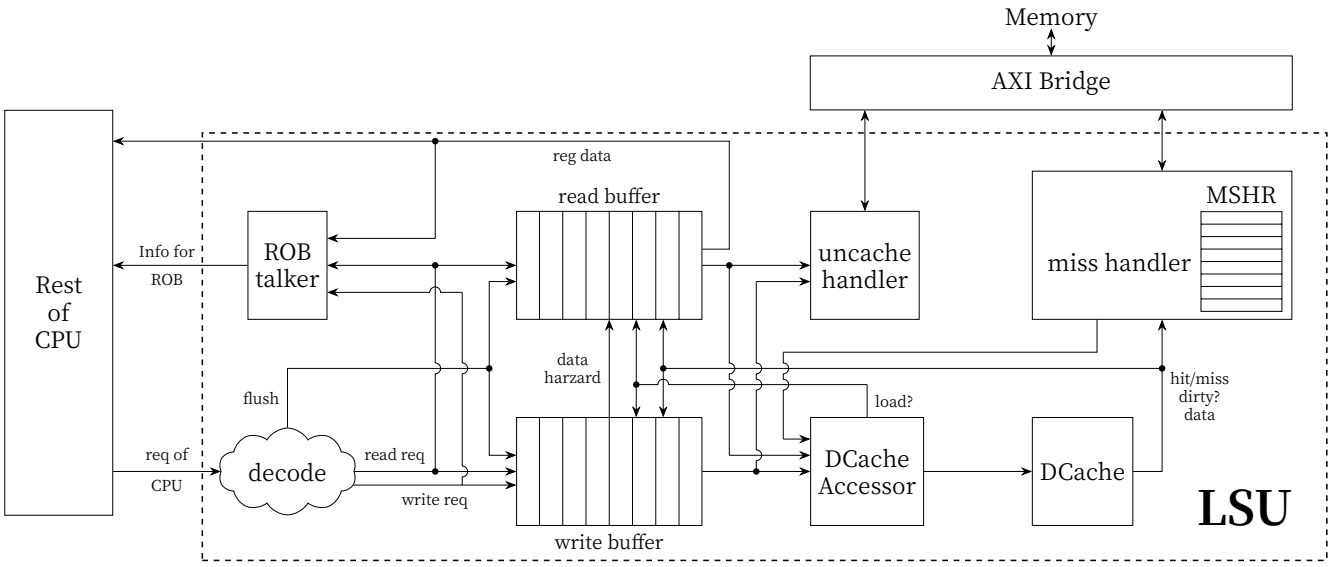

图 5: LSU 框架图

主要包含以下模块：LSU 中的大部分事件或者当拍完成，或者流水完成，仅存在少数状态机（繁忙/空闲两个状态）

- 写缓存：暂存未完成的写请求
  - 将写请求加入写缓存
  - 监听写提交信号，并对相关请求做可提交标记
  - 监听 flush 信号，flush 有效时将所有未标记为可提交的请求清除

  - 在最早进入的请求准备好后发送到 DCache 访问仲裁或 uncache 处理器
  - 在与 uncache 处理器握手成功后删除对应请求
  - 监听 DCache 反馈，如果请求命中，删除对应请求
  - 监听 DCache 反馈，对任意 dcache 请求的 miss 结果，将所有同 cache 行请求做 miss 标志
  - 监听 DCache 仲裁，如果发生载入 cache 行事件，消除相应请求的 miss 标志
  - 监听读请求，在收到读请求后向读缓存发送与该读请求存在数据相关写请求信息
  - 每个周期都向读缓存发送当前写缓存中请求的有效状态
- 读缓存：暂存未完成的读请求
  - 将读请求加入读缓存，并记录写缓存提供的依赖信息
  - 根据写缓存提供的请求有效状态，更新读请求依赖
  - 监听 flush 信号，flush 有效时清空所有请求
  - 监听 ROB 判断是否可以发送 uncache 请求
  - 将读缓存中准备好的请求发送到 DCache 访问仲裁或 uncache 处理器
  - 如果存在多条准备好的请求，则优先发送读缓存中地址较小者。
  - 在数据返回后（DCache 命中或 uncache 返回）打包数据传出，并清除对应请求
  - 监听 DCache 反馈，对任意请求的 miss 结果，将所有同 cache 行请求做 miss 标志
  - 监听 DCache 仲裁，如果发生载入 cache 行事件，消除相关请求的 miss 标志
- DCache 访问仲裁：调度对 DCache 的访问
  - 调度优先级排序（由高到低）
    1. cache 行载入
    2. cache 读请求
    3. cache 写请求
  - 行载入请求发生时告知写缓存和读缓存
- DCache：2 级流水 cache 访问
  - 第一级流水的工作包含
    * 读、写请求：读取 tag，判断是否命中
    * 出现结构相关时使用旁路获取 tag
  - 第二级流水的工作包含
    * 如果读写命中，获取相应 data
    * 如果读写缺失，获被替换块的 data
    * 如果写命中，更新 tag 中的 dirty 位

    * 更新替换策略模块信息
    * 对于载入请求，写入新的 tag 和 data

- 缺失处理器：负责缓存行的写回与载入
  - 监听 DCache 反馈，发生缺失时尝试将缺失保存到缺失信息队列
  - 如果要添加的缺失信息队列中已存在则直接丢弃
  - 如果待替换块需要写回则先执行写回，再执行载入
  - 监听内存返回数据，数据到达时向 DCache 仲裁发送载入请求
- Uncache 处理器：负责不经过 cache 的访存操作
  - 空闲状态下接受请求并转发给内存（总线仲裁）
  - 接受读请求后转为繁忙状态
  - 收到读数据后转为空闲状态，并向读缓存返回数据
- ROB 通信器：向 ROB 反馈 LSU 的工作状态

9. 重排序缓存设计

重排序缓存可以保存乱序执行的各个指令的位置信息，并保证他们按序退休。

设计说明 指令从发射队列中发射出来的时候，失去了它们原本的先后信息，这使得分支预测失败恢复、精确中断等变得难以实现。为此，我们将各个指令及其信息按序保持下来，并且给予其一个保证在生命周期内不会重复的 id。同时，各个执行单元在执行完某指令后通知 ROB，使得 ROB 中的指令可以退休。

输入输出信号说明

信号名称	信号方向	信号来源/目标	信号意义
clk	输入	顶层模块	时钟
rst	输入	顶层模块	重置
ctrl_rob			
pauseReq	输出	控制单元	暂停请求
dispatch_rob			
uOP0	输入	上级流水线	指令信息（标准接口）
uOP1	输入	上级流水线	指令信息（标准接口）
rob_commit			
uOP0	输出	下级流水线	指令信息（标准接口）
uOP1	输出	下级流水线	指令信息（标准接口）
ready	输入	下级流水线	握手信号
valid	输出	下级流水线	握手信号
alu0_rob	输入	ALU0	指令退休信息
alu1_rob	输入	ALU1	指令退休信息
lsu_rob	输入	LSU	指令退休信息
mdu_rob	输入	MDU	指令退休信息
setFinish			指令执行完成
id			执行完成的指令的 id
setBranchStatus			设置分支指令状况
branchTaken			分支指令是否跳转
branchAddr			分支指令跳转目标
setException			指令触发异常
exceptionType			触发异常类型
BadVAddr			未对齐的虚拟地址

表 8: 重排序缓存输入输出信号说明

10.AXI 接口控制模块设计

AXI 接口负责将 CPU 中各个模块发出的访存请求统合、转换为标准的 AXI 事务。

设计说明 为满足 CPU 的各种访存需求，AXI 接口控制模块提供了三条信道，分别是 ICache 取指信号、DCache 取指信道和 Uncached 访存信道。ICache 和 DCache 信道每次读/写 128bit，Uncached 信道每次读/写 32bit。为实现效率最大化，接口控制模块尽量使用猝发传输，并可以同时处理读写请求。为此，AXI 接口控制模块的读写状态分别由一个状态机管理，其状态转移图分别如下所示:

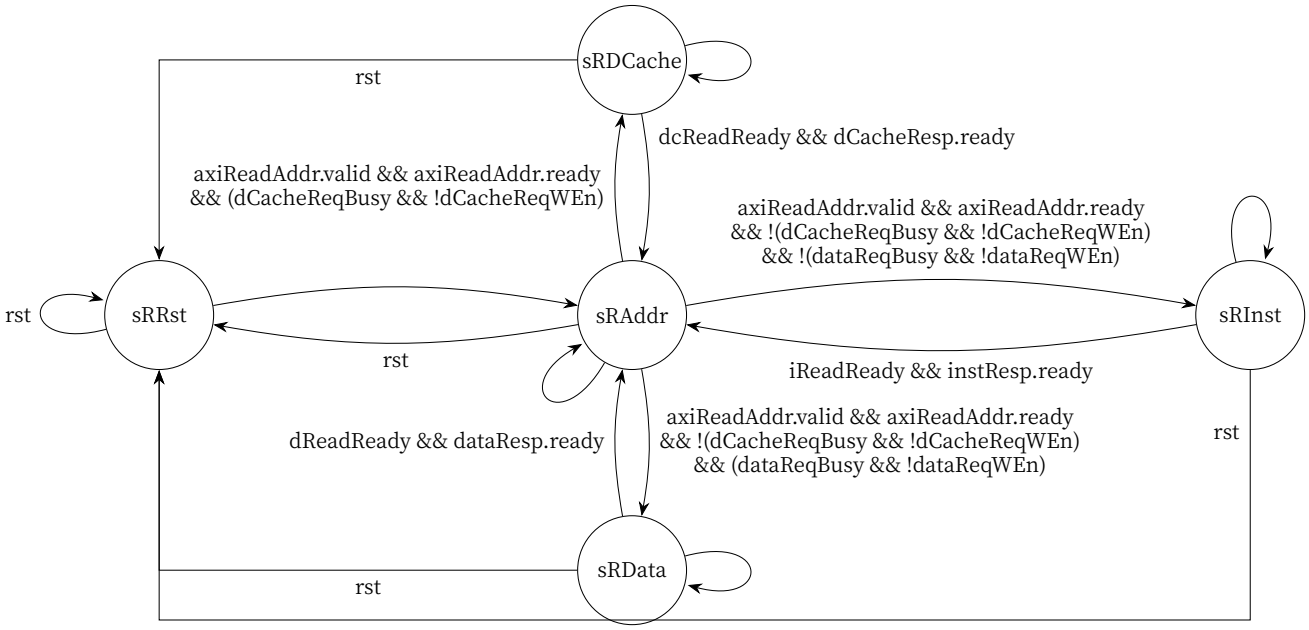

图 6: AXI 接口控制模块读状态的状态转移图

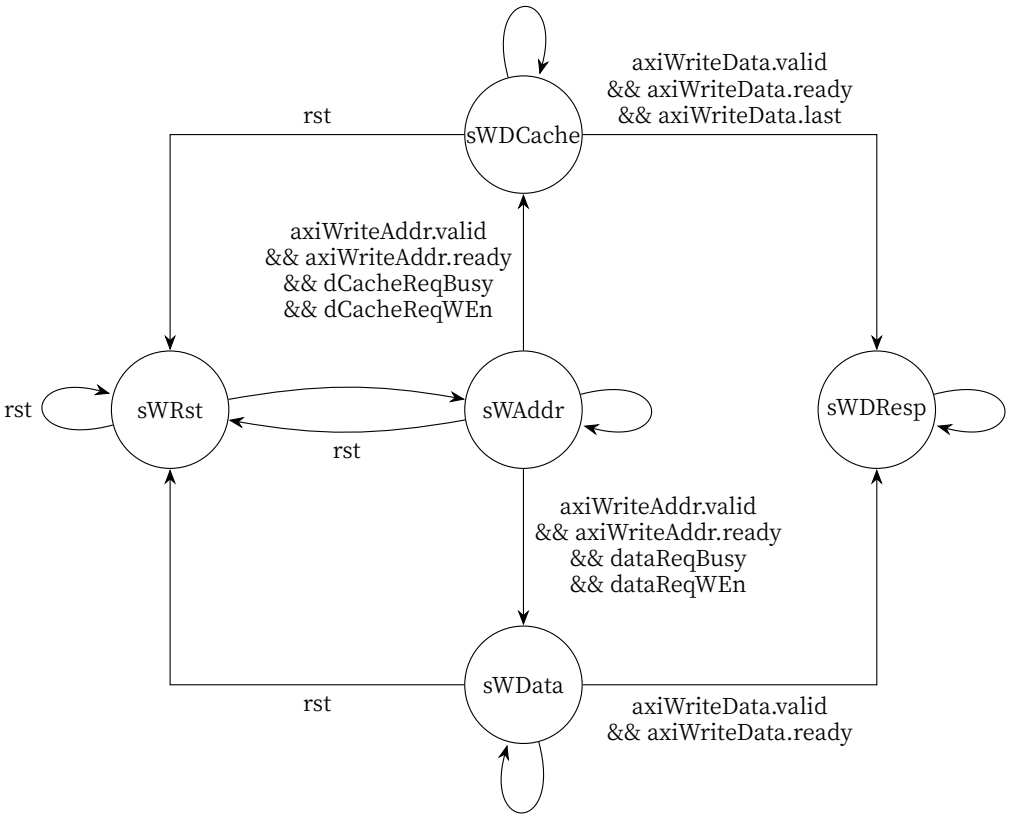

图 7: AXI 接口控制模块写状态的状态转移图

其中，对于读状态机，sRADDR 是等待与发送读地址；sRInst 是在发送 ICache 的读请求后，等待 AXI 读请求返回；SRData 是在发送 uncached 的读请求后，等待 AXI 读请求返回；sRDCache 是在发送 ICache 的读请求后，等待 AXI 读请求返回；sRPendingResp 是等待将读

取到的结果返回给各模块。对于写状态机，sWAddr 是等待与发送写地址，sWData 是等待与发送 uncached 写内容，sWData 是等待与发送 DCache 的写内容，sWDResp 和 sWDCache 都是等待 AXI 返回写入情况。

输入输出信号说明

表 9: AXI 接口控制模块输入输出信号说明

信号名称	信号方向	信号来源/目标	信号意义
clk	输入	顶层模块	时钟
rst	输入	顶层模块	重置
axiReadAddr		CPU 外部	AXI AR 信道
id	输出		ARID
address	输出		ARADDR
length	输出		ARLEN
size	输出		ARSIZE
burst	输出		ARBURST
lock	输出		ARLOCK
cache	输出		ARCACHE
protect	输出		ARPROT
valid	输出		ARVALID
ready	输入		ARREADY
axiReadData		CPU 外部	AXI R 信道
id	输入		RID
data	输入		RDATA
respond	输入		RRESP
last	输入		RLAST
valid	输入		RVALID
ready	输出		RREADY
axiWriteAddr		CPU 外部	AXI AW 信道
id	输出		AWID
address	输出		AWADDR
length	输出		AWLEN
size	输出		AWSIZE
burst	输出		AWBURST
lock	输出		AWLOCK
cache	输出		AWCACHE
protect	输出		AWPROT
valid	输出		AWVALID
ready	输入		AWREADY

待续

表 9（续）

信号名称	信号方向	信号来源/目标	信号意义
axiWriteData		CPU 外部	AXI W 信道
id	输出		WID
data	输出		WDATA
strobe	输出		WRESP
last	输出		WLAST
valid	输出		WVALID
ready	输入		WREADY
axiWriteResp		CPU 外部	AXI B 信道
id	输入		BID
respond	输入		BRESP
valid	输入		BVALID
ready	输出		BREADY
instReq		ICache	ICache 读请求
pc	输入		读请求的地址
valid	输入		握手信号
ready	输出		握手信号
instResp		ICache	ICache 读返回
cacheLine	输出		读请求的结果
valid	输出		握手信号
ready	输入		握手信号
dataReq		LSU	Uncached 读/写请求
addr	输入		读/写请求的地址
write_en	输入		写使能
data	输入		写数据
strobe	输入		读写掩码
size	输入		读写字节数
valid	输入		握手信号
ready	输出		握手信号
dataResp		LSU	Uncached 读返回
data	输出		读请求的结果
valid	输出		握手信号
ready	输入		握手信号
dCacheReq		DCache	DCache 读/写请求
addr	输入		读/写请求的地址
write_en	输入		写使能
data	输入		写数据
valid	输入		握手信号
ready	输出		握手信号

待续

表 9（续）

信号名称	信号方向	信号来源/目标	信号意义
dCacheResp		DCache	DCache 读返回
data	输出		读请求的结果
valid	输出		握手信号
ready	输入		握手信号

三、设计结果

（一）设计交付物说明

SHIT-Core: 顶层目录

- source: cpu 代码目录
  - ip：cpu 所用 ip 核
    - data_blk：block memory，用做 dcache 数据存储块
      - data_blk.xci
    - data_ram_0：block memory，用作 icache 数据存储块
      - data_ram_0.xci
    - Divider_IP: 除法器
      - Divider_IP.xci
    - Multiplier：乘法器
      - Multiplier.xci
    - tag_blk：block memory，用作 dcache tag 存储块
      - tag_blk.xci
    - tag_ram：block memory，用作 icache tag 存储块
      - tag_ram.xci
  - new：自己编写的 Verilog/SystemVerilog 文件
    - defines：包含部分头文件
      - defines.sv：包含部分数据宏定义
    - exu：包含 ALU 和 MDU 相关模块文件
      - ALU.sv：ALU 逻辑
      - FU_Output_regs.sv：功能模块统一出口流水寄存器模块
      - Issue_RF_regs.sv：发射出口流水寄存器模块
      - MDU：乘除法逻辑
      - MDUIQ_RF_regs.sv：乘除法发射出口流水寄存器模块

    - prf：通用寄存器堆访问模块
    - RF_FU_regs.sv：寄存器堆到 ALU 的流水寄存器
    - RF_MDU_regs.sv：寄存器堆到 MDU 的流水寄存器
* ifu：取指模块，包含 icache 与分支预测
    - BPD.sv
    - CtrlUnit.sv
    - ICache.sv：ICache 访问控制模块
    - IF_0.sv
    - IF_3.sv
    - IF0_1_reg.sv
    - IF2_3_reg.sv
    - IF3_Output_reg.sv
    - IFU.sv
    - InstBuffer.sv
    - NLP.sv
    - Predecoder.sv
* interface：AXI 转接桥
    - AXIInterface.sv
    - AXIWarp.sv
* issue：发射相关模块
    - busy_table.sv
    - issue_arbiter_4.sv
    - issue_arbiter_8.sv
    - issue_arbiter_8_sel1.sv
    - issue_unit_ALU.sv
    - issue_unit_LSU.sv
    - issue_unit_MDU.sv
    - scoreboard.sv
    - store_bitmask_gen.sv
    - wake_unit.sv
* LSU：访存模块
    - dcache.sv：naive dcache 功能实现
    - dcache_accessor.sv：dcache 访问仲裁
    - dcache_wraper.sv：dcache 输入输出封装层
    - load_handler.sv：cache 行载入处理器
    - LSU.sv：LSU 顶层封装
    - LSU_defines.svh：LSU 接口定义

        - mem_listener.sv：内存监听模块
        - MHandler_defines.svh：缺失处理器接口定义
        - miss_handler.sv：缺失处理器
        - read_buffer.sv：读请求缓存
        - rob_talker.sv：ROB 通信器
        - uncache_handler.sv：uncache 访存器
        - write_buffer.sv：写请求缓存
        - writeback_handler.sv：写回处理器
    * renaming：重命名模块
        - busy_table.sv
        - free_list.sv
        - freelist_enc_64r.sv
        - freelist_enc8.sv
        - freelist_enc64.sv
        - map_table.sv
        - register_rename.sv
        - rob.sv
    * Commit.sv
    * CP0.sv
    * CtrlUnitBackend.sv
    * decode.sv：译码模块
    * decode_rename_regs.sv：译码模块到重命名模块的流水寄存器
    * defs.sv
    * dispatch.sv
    * dispatch_iq_regs.sv
    * exception.sv
    * instBuffer_decode_regs.sv：指令队列到译码单元的流水寄存器
    * interface.sv
    * MyCPU.sv
    * pipeline.sv
    * rename_dispatch_reg.sv
    * TLB.sv
- design.pdf：本文档

（二）设计演示结果

测试全部通过，参见下图。

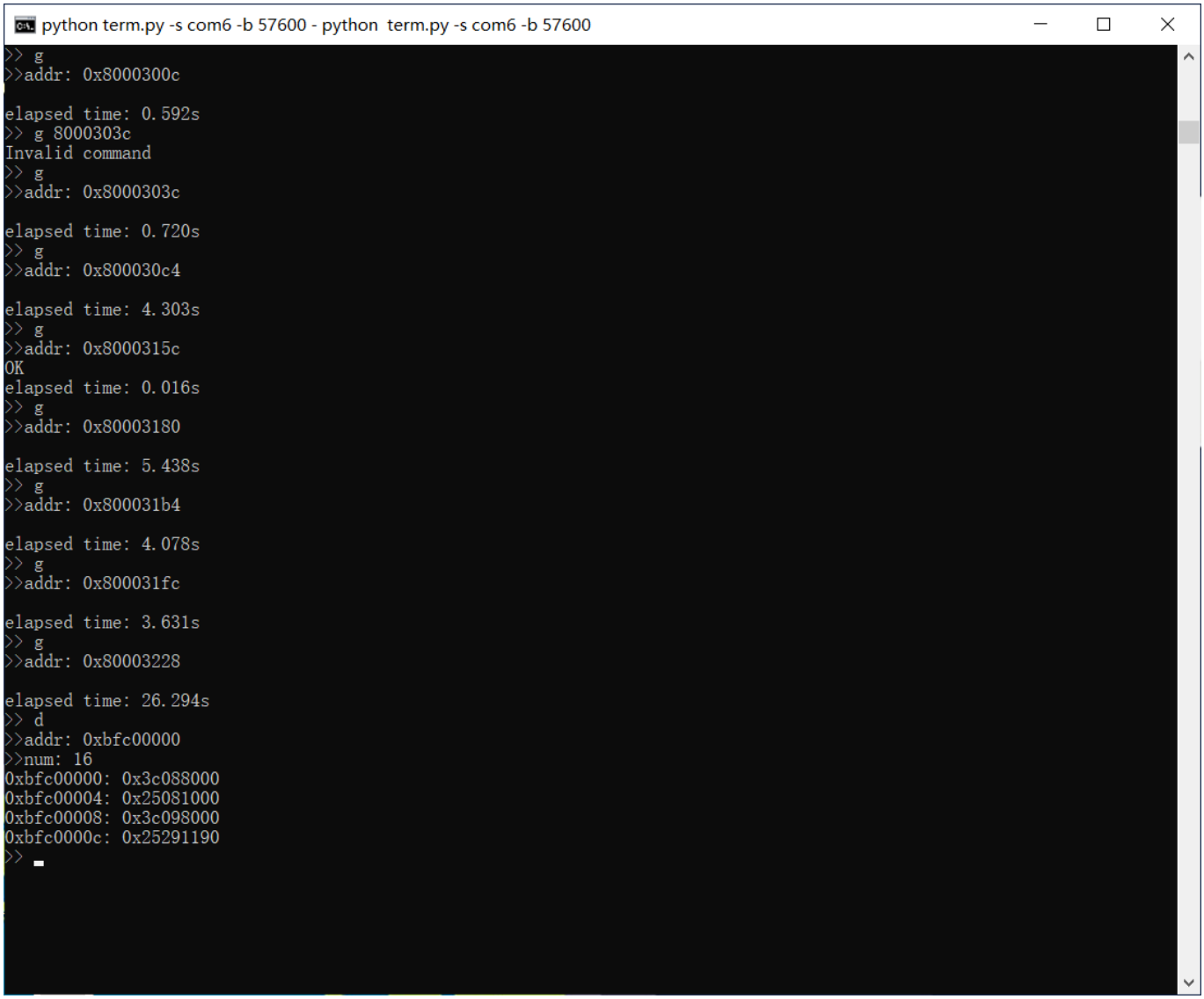
python term.py -s com6 -b 57600 - python term.py -s com6 -b 57600
>> g
>>addr: 0x8000300c
elapsed time: 0.592s
>> g 8000303c
Invalid command
>> g
>>addr: 0x8000303c
elapsed time: 0.720s
>> g
>>addr: 0x800030c4
elapsed time: 4.303s
>> g
>>addr: 0x8000315c
OK
elapsed time: 0.016s
>> g
>>addr: 0x80003180
elapsed time: 5.438s
>> g
>>addr: 0x800031b4
elapsed time: 4.078s
>> g
>>addr: 0x800031fc
elapsed time: 3.631s
>> g
>>addr: 0x80003228
elapsed time: 26.294s
>> d
>>addr: 0xbfc00000
>>num: 16
0xbfc00000: 0x3c088000
0xbfc00004: 0x25081000
0xbfc00008: 0x3c098000
0xbfc0000c: 0x25291190
>>

图 8: 系统测试结果截图

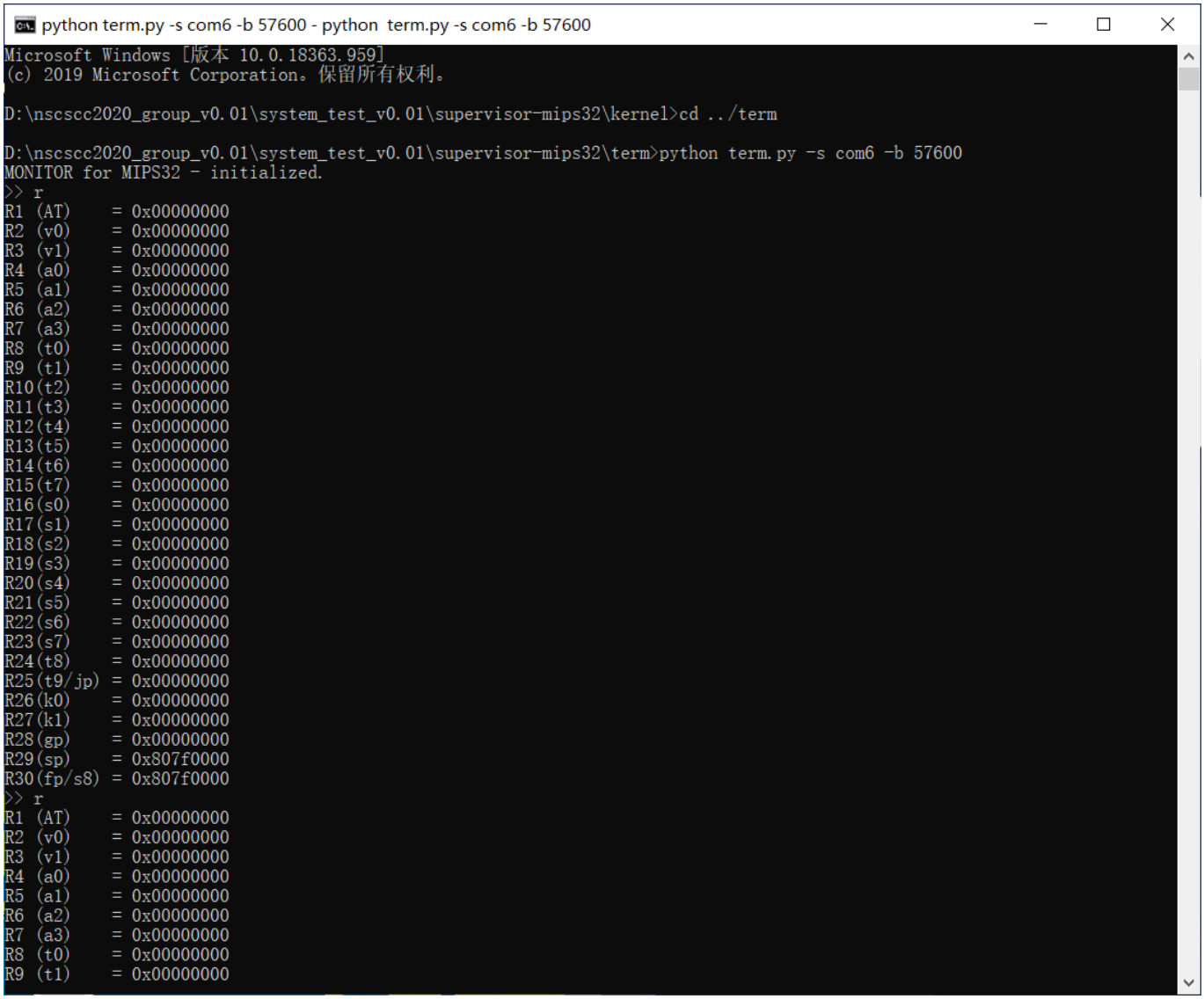

图 9: 系统测试结果截图

（三）性能分析结果及反思

本 CPU 虽然实现了现代超标量处理器的大部分特性，如乱序执行、寄存器重命名、多发射、非阻塞 Cache，但其在部分测试上的表现仍然不尽如人意，经过分析，其原因可能如下：

1. 分支预测失败惩罚

由于流水线级数较深，而本处理器并未采用分支预测失败时当拍 Flush 对应 Tag 指令的机制，而是选择在指令退休的时候对分支预测的正确性进行检查，若预测失败则刷新整个流水线，而刷新流水线的代价是非常大的，新的指令充满流水线至少需要十几个周期，这就导致了我们的处理器分支预测失败惩罚极大。由于时间所限，我们没有将实现的性能更高的 GShare 分支预测器进行上板验证，仅进行了仿真，仿真结果如下：

表 10: 添加 GShare 分支预测器后的仿真结果

测试程序	原 SoC Count	新 SoC Count	性能提升
Bitcount	c503	b913	6%
Bubble Sort	6ddce	5e57e	14%
Coremark	d11af	bd3aa	10%
Crc32	4c09d	3becb	21%
Dhrystone	16ca2	14f0a	8%
Quicksort	6a3ef	5f7b4	10%
Selectsort	38e9a	29725	27%
SHA	36493	3556b	2%
Stream Copy	90de	932c	-2%
Stringsearch	3c39d	33b27	14%

可以看出，添加了 GShare 分支预测器之后，带来的性能提升非常显著。

2. 流水线暂停惩罚

在程序运行的过程中，我们对部分数据进行了统计，发现 LSU 请求暂停数量较多。在算数指令较为密集的 SHA、CRC 等测试中，我们得到了 50~60 分的成绩；而一些访存密集的测试的结果则不尽如人意。经分析，原因为未实现 ALU 与 LSU 之间的旁路路径，导致各功能单元之间相互依赖的指令无法背靠背执行，而 LOAD 指令又常常处于相关性的顶端，且 Store 指令位于队列首部时才可参与仲裁，Store 指令得到来源于这就造成了整体 IPC 的下降。此外，前端 ICache 仅有 4KB，也一定程度上影响了取指的效率，增加了后端流水线空置的时间。

3. 过大的电路面积

虽然本处理器实现了十几级的流水线，但其频率却不尽如人意，通过分析 Vivado 给出的报告，我们发现，部分模块占用了过大的芯片面积，例如 ROB、寄存器重命名表、发射队列和乘除法单元。同时，由于我们将译码信息在级间连续传递，而没有及时丢弃，导致占用了过多的寄存器，使得 FPGA 片上资源紧张，布线困难，自然频率无法提高。

4. 不完整的旁路网络

在实现处理器的过程中，考虑到运行频率和布线复杂度，并未实现完整的旁路网络。事实证明，在数据相关性较高的 Coremark/Streamcopy 等测试程序中，观察到流水线中气泡较多，处理器并不能找到不相关的指令填充这些气泡，导致 IPC 的降低。

四、参考设计说明

我们在设计时，参考了多个开源项目的内容，并使用了一些 Xilinx 提供的 IP 核，列举如下：

- 我们参考了 The Berkley Out of Order Machine[1] 和《超标量处理器设计》[2] 的结构设计

- 在乘除法执行单元中，我们使用了 Xilinx 的 Multiplier[3] 和 Divider IP 核[4]
- 在 ICache 和 DCache 中，我们使用了 Xilinx 的 Block ram IP 核[5]

五、参考文献

[1] *UCB-BAR: Berkeley Out-of-Order Machine*. URL: https://bar.eecs.berkeley.edu/projects/boom.html (visited on 08/05/2020).

[2] *超标量处理器设计*. 清华大学出版社, 2014. ISBN: 9787302347071.

[3] *Multiplier*. Xilinx. Library Catalog: www.xilinx.com. URL: https://www.xilinx.com/products/intellectual-property/multiplier.html (visited on 08/05/2020).

[4] *Divider*. Xilinx. Library Catalog: www.xilinx.com. URL: https://www.xilinx.com/products/intellectual-property/divider.html (visited on 08/05/2020).

[5] *Block Memory Generator*. Xilinx. Library Catalog: www.xilinx.com. URL: https://www.xilinx.com/products/intellectual-property/block_memory_generator.html (visited on 08/05/2020).