基于敏捷开发语言 Chisel 的 RISC-V 处理器设计与验证*

秦国锋** 冯煊

同济大学计算机科学与技术学院，嘉定 200092

摘 要 随着处理器技术的发展，指令集对处理器制造的影响日益突出。市场上 X86 和 ARM 指令集分别在 PC 端和移动端占据主导地位，开源指令集 RISC-V 在近年来逐渐与 X86 和 ARM 指令集形成三足鼎立的发展态势。本文剖析了三种指令集的优劣，选择 RISC-V 指令集以避免版权问题，通过一种新的敏捷设计语言 Chisel 进行开发，基于软件开发领域的敏捷开发流程，设计了一款 RISC-V 指令系统的 6 级流水线处理器，并通过 riscv-tests 测试套件进行测试，结果表明该处理器通过 RV64I 基础指令集测试。该处理器具有高代码密度和快速调试能力的优点，并可在模拟后正确执行，一定程度上可以为 Chisel 处理器设计开发流程提供参考。

关键词 RISC-V， Chisel， 处理器设计， 分支预测

Design and Verification of RISC-V Processor Based on Agile Development Language Chisel

GuoFeng Qin Xuan Feng

School of Computer Science and Technology, Tongji University
Shanghai 201800,China

Abstract—With the advancement of processor technology, the impact of instruction sets on processor manufacturing has become increasingly prominent. The X86 and ARM instruction sets dominate the PC and mobile markets, respectively. Recently, the open-source RISC-V instruction set has gradually formed a three-way competition with the X86 and ARM instruction sets. This article analyzes the advantages and disadvantages of these three instruction sets, choosing the RISC-V instruction set to avoid copyright issues. This article uses Chisel, a new agile design language, to develop a 6-stage pipelined processor based on the agile development process in software development. The processor was tested using the riscv-tests test suite, demonstrating that it passed the RV64I basic instruction set tests. The processor boasts high code density and fast debugging capabilities, and executes correctly after simulation. This article provides a reference for the Chisel processor design and development process.

Keywords— risc-v, chisel, processor design, branch prediction

1 引 言

指令集架构是计算机体系结构中与编程密切相关的组成部分，也是处理器设计需要考虑的核心之一。X86 和 ARM 两大体系在当代计算领域占据主导地位，分别归属于复杂指令集计算机（Complex Instruction Set Computing，CISC）设计体系和精简指令集计算机（Reduced Instruction Set Computing，RISC）设计体系。RISC-V 作为新兴开放架构[1]，凭借其开源免许可费的核心优势打破传统市场格局，为中小型创新企业提供了显著的成本优势与发展空间[2]，正逐步改变行业竞争态势。

相比于 X86 和 ARM 指令集架构，RISC-V 有着独有的优势[3]3。RISC-V 并非出于商用目的开发，因此不需要像 X86 与 ARM 为了维护版本迭代而出现的“向下兼容”问题。而 RISC-V 的开源特性也激发了创新，RISC-V 如今已经应用于各种行业，如嵌入式系统、物联网、高性能计算机，具有很高的拓展性和灵活性。

从 RISC-V 诞生的 2010 年到现在，已经超过 200 家公司参与了 RISC-V 的研究，甚至包括谷歌、英伟达、高通等科技巨头[3]5。在学术界，国外的伯克利[4]、国内的中科院[5]都基于 RISC-V 指令集研发了自己的处理器用于教学和科研活动。在计算机硬件教学界，东北大学王彤[6]等人积极探索软硬件协同创新的教学模式，对硬件学科具有重要的指导意义；哈尔滨工业大学的张彦航[7]等人则是分析了硬件教学的痛点，提出“学、练、用”三位一体的培养计划，成功实现了学生水平的提升；而西北工业大学王毅航[8]等人则是提出了“智能硬件”的概念，旨在开发更方便的教学仪器并且与物联网等新领域联系起来，比以往方式更加生动且富有实践意义。在商业界，阿里巴巴等公司也都非常看重 RISC-V 指令集，设计出了自己的商业 RISC-V 处理

器[9]。

而处理器设计中另一个很重要的内容则是开发模式。软件工程领域早已凭借敏捷开发模式适应了市场需求的高频变换，Scrum 等敏捷方法论已形成完整实践体系[10]。这种开发模式通过模块化迭代显著缩短开发周期，配合丰富的开源工具生态有效降低了人力与资源消耗。相比之下，硬件开发仍普遍沿用传统瀑布模型，其微架构设计、RTL 开发与验证环节处于割裂状态，且缺乏软件领域成熟的工具链支持[11]，这些特性共同制约着硬件开发效率的提升。

伯克利团队研发的 Chisel 语言为硬件领域的敏捷开发带来可能性[12]。Verilog 与 VHDL 长期主导硬件编程领域，作为硬件描述语言（HDLs）最初定位于仿真验证，后期才延伸至综合应用，但其综合过程易出错且代码复用依赖复制粘贴。Chisel 凭借抽象层级与代码密度优势，P. Lennon 和 R. Gahan 在生成等效设计时其代码规模仅为 Verilog 的 46.6%[13]，同时集成 PeekPokeTester 等测试组件加速 RTL 细化进程，相较于 Verilog 可实现更高效的验证流程。

本文采用 Chisel 语言编写，设计一个具有 6 级流水线的超标量处理器，实现了 RV64I 指令集，并将 GShare 算法结合 BTB 和 RAS 提高分支预测准确率、使用寄存器重命名来减少数据相关带来的延迟与风险。实验结果符合设计预期目标，该处理器 IP 能用于大学本科实验与教学。

2 相关工作

2.1 RISC-V 开源指令集架构

RISC-V 架构的核心基础被定义为“基本整数指令集（Base Integer Instruction Set）”，核心特征在于整数寄存器位宽与用户地址空间完全对应，具体划分为 32 位（RV32I）与 64 位（RV64I）两个版本[14]11。在基础指令集框架下，设计人员可从标准扩展模块中灵活选配，构建定制化指令系统。最具代表性的 IMAFD 扩展组由 I/M/A/F/D 五个核心模块构成：基础 I 模块涵盖整数运算核心功能，覆盖整数加载、存储及控制流指令，属于所有 RISC-V 实现的必备组件；M 模块对应整数乘除运算单元；A 模块提供原子操作指令集，通过自动化的内存读—改—写机制实现处理器同步控制；F 模块引入单精度浮点运算体系，扩展支持浮点寄存器及单精度计算指令；D 模块作为 F 模块的增强版本，将浮点运算精度提升至双精度级别，全面支持双精度操作数处理。

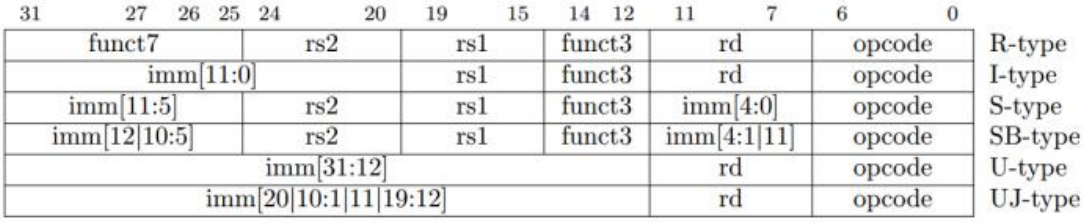

图 1 RV32I 不同类型指令格式[14]

RISC-V 基本整数指令集 I 采用 32 位定长指令格式[14]21，其 0-6 位对应操作码字段，12～14 位设置 funct3 功能码，其格式如图 1 所示。操作码用于标识指令类别，功能码则实现具体指令的二次细分。该架构将指令划分为 R/I/S/B/U/J 六种类型（详见表 1），其中 R 型指令结构具有特殊性：在保留 funct3 功能码基础上，额外使用 25～31 位定义 funct7 字段，通过双功能码组合实现 R 型指令的精细化分类。

表 1 不同指令类型的字段含义

指令类型	含义	典型指令
R	寄存器与寄存器算术指令	add sub xor or sll
I	寄存器与立即数算术指令或者加载指令	addi xori slli srli lb lh lw lbu
S	存储指令	sb sh sw
B	条件跳转指令	beq bne blt bge
U	长立即数操作指令	lui auipc
J	无条件跳转指令	jal

2.2 敏捷开发语言 Chisel

Verilog 与 VHDL 作为硬件开发领域的主流语言，均属于硬件描述语言（HDL）范畴。这两类语言最初以硬件仿真为目标开发，后期逐步扩展至硬件设计领域。由于实际能用于硬件实现的部分仅仅是语言中可综合的子集，而设计时所用到的部分很容易超出该子集覆盖范围，导致其应用复杂度显著提升。相较于软件领域的高级编程语言，HDL 在抽象层级（仅略高于门级电路）、调试手段（依赖波形分析）与模块复用（缺乏标准化包管理）等方面存在明显代差，这些特性客观上制约了硬件开发效率的提升[15]。

Chisel 的引入为上述问题提供了改进路径。该硬件构造语言基于 Scala 实现（全称为 Constructing Hardware In a Scala Embedded Language），通过继承宿主语言的高抽象特性，有效提升了代码复用效率。受益于 Scala 原生支持的参数化设计与元编程能力，Chisel 采用生成器范式实现硬件开发——在电路生成阶段通过参数注入动态调整结构配置，相较于传统 HDL 语言展现出更强的设计灵活性与可扩展性。这种机制使得硬件模块可像软件库那样进行多层次复用，显著优化了开发流程。

3 处理器微架构设计

3.1 流水线划分

流水线技术通过指令执行阶段的重叠机制，在硬件资源有限条件下实现效率提升[16]。MIPS 架构定义的经典 5 级流水线模型（取指、译码、执行、访存、写回）为本研究提供基础框架[17]。本文处理器架构在保留核心功能的基础上实施结构调整：将传统执行与访存阶段合并为统一操作单元，同时引入发射与提交两个新增环节，形成 6 级流水线结构（取指、译码、发射、执行、写回、提交）。该调整方案主要面向超标量处理器的实现需求，通过与 Tomasulo 算法动态调度机制的深度适配完成功能优化，图 2 展示了改进后的流水线阶段划分及数据流向。

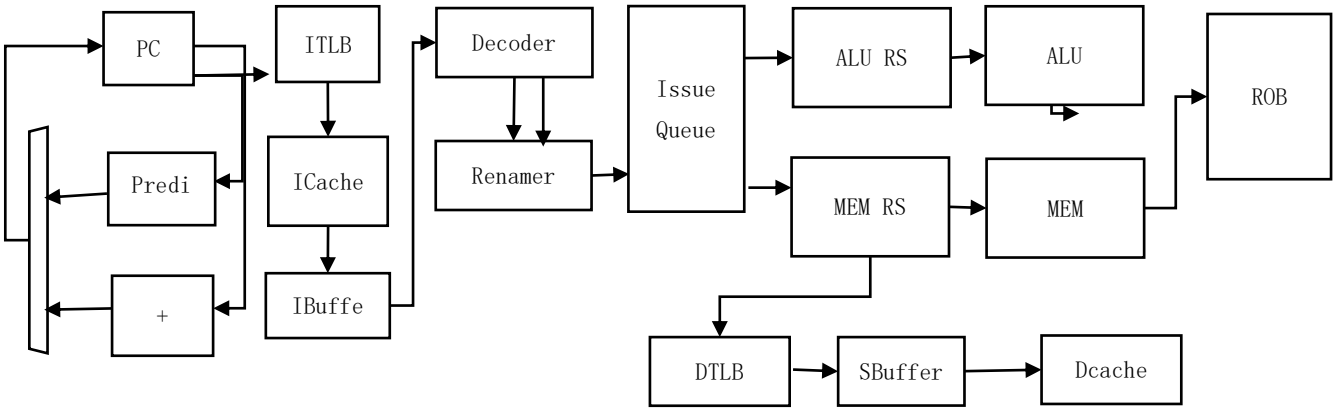

图 2 流水线结构

3.2 分支预测模块

分支预测作为流水线 CPU 的关键优化技术，通过预判分支指令的后续走向维持流水线运转连续性，有效缓解因条件判断产生的流水线阻塞现象。其核心功能涵盖跳转方向预测（是否发生分支跳转）与跳转地址预测（目标指令位置确定）两个维度。

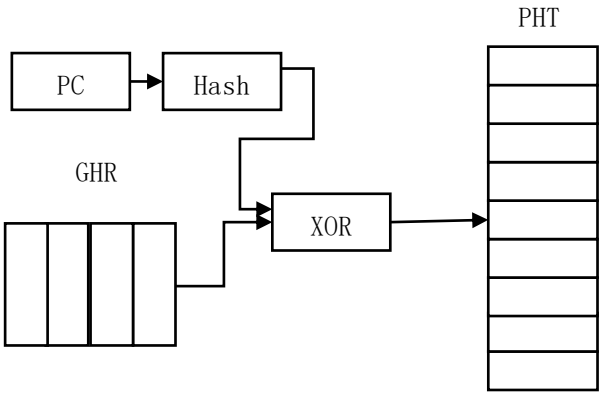

图 3 Gshare 算法的原理

当前技术方案中，静态预测与动态预测构成主要实现路径，前者依赖固定规则而后者通过运行时数据分析优化预测精度，但动态方案在提升准确率的同时需要更多存储资源与硬件逻辑支持。本研究采用gshare动态预测算法，该方案在硬件复杂度与预测效率间实现平衡优化。其运作机制依托全局历史寄存器（GHR）记录近期分支跳转状态（1/0表示跳转与否），结合模式历史表（PHT）建立预测模型。为降低PHT表项数量与指令地址（PC）的映射冲突，引入哈希运算将PC值压缩至与PHT容量匹配的索引范围，同时通过PC哈希值与GHR的异或操作进一步分散索引分布，图3展现了该算法的完整执行流程与交互逻辑。

针对分支目标地址预测的复杂性，本研究构建了双轨道预测体系：通过BTB（Branch Target Buffer）处理直接跳转指令地址预测，RAS（Return Address Stack）应对间接跳转场景。考虑到CALL/RETURN类子程序调用指令构成间接跳转主体，RAS采用栈式存储机制进行管理——当检测到CALL指令时将返回地址（PC+4）压入栈顶，遭遇RETURN指令时则弹出栈顶地址作为跳转目标。该预测系统的执行流程通过图4进行可视化呈现，其中方向预测模块（Direction Predicor）的结构逻辑已在图3实现完整解析。

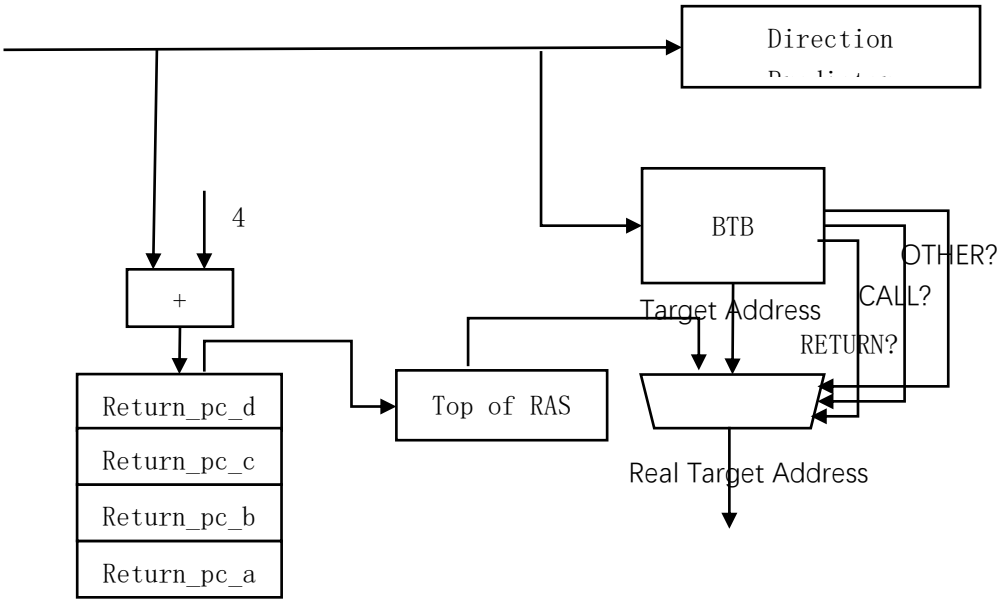

图 4　分支预测器的内部结构，结合Gshare和BTB以及RAS

3.3　虚实地址转换

虚拟内存作为处理器架构的关键功能模块，通过构建虚拟地址空间扩展应用的可用内存范围。RISC-V体系依据位宽差异定义多种地址转换模式（RV32支持Sv39，RV64兼容Sv39/Sv48），本文采用Sv39方案实现虚实地址映射。如图5所示，Sv39转换机制的核心流程包含：SATP寄存器存储一级页表基址的物理页号（PPN），通过左移12位后与虚地址的VPN[2]字段拼接，生成一级页表项（PTE）的查询地址。该过程通过三级页表逐级索引（每级对应VPN[2]/[1]/[0]），末级页表的PPN经12位移位后与虚地址偏移量(Offset)组合形成物理地址。最终实现基于实地址的数据访问操作。为优化虚实地址转换效率，本研究集成页表缓存（TLB）机制降低访问延迟。TLB作为高频访问页表项（PTE）的快速缓存区，存储了近期使用的虚拟地址与物理页映射关系。当处理器访问已缓存的虚拟地址时，可绕过传统三级页表查询流程，直接通过TLB记录的PTE获取物理地址。ITLB采用二路组相连结构，每组8块，若命中则可直接根据地址取数据，这种利用时空局部性原理的缓存策略，有效缩减了地址转换环节的时间开销。

存储架构设计为一级缓存分立结构，包含独立指令缓存（L1 ICache）与数据缓存（L1 DCache）。双路组相联架构下每个缓存行容量为128比特，恰好容纳四条32位定长指令。存储器地址可以划分为如图6所示的三个核心字段：标志位（Tag）、索引位（Index）与偏移量（Offset）。访存地址解析时，索引字段用于定位目标缓存组，标志位比对验证缓存行有效性，块偏移量（Block Offset）完成数据块内精确定位。缓存访问采用并行流水机制：索引解码与标志比对同步进行，当周期完成数据读取与命中状态判断。若标志位校验失败则触发缓存缺失流程，此时根据LRU策略置换最近最少使用的缓存行——该算法基于时间局部性原理，通过维护访问时间戳队列，优先淘汰历史访问间隔最长的陈旧数据。

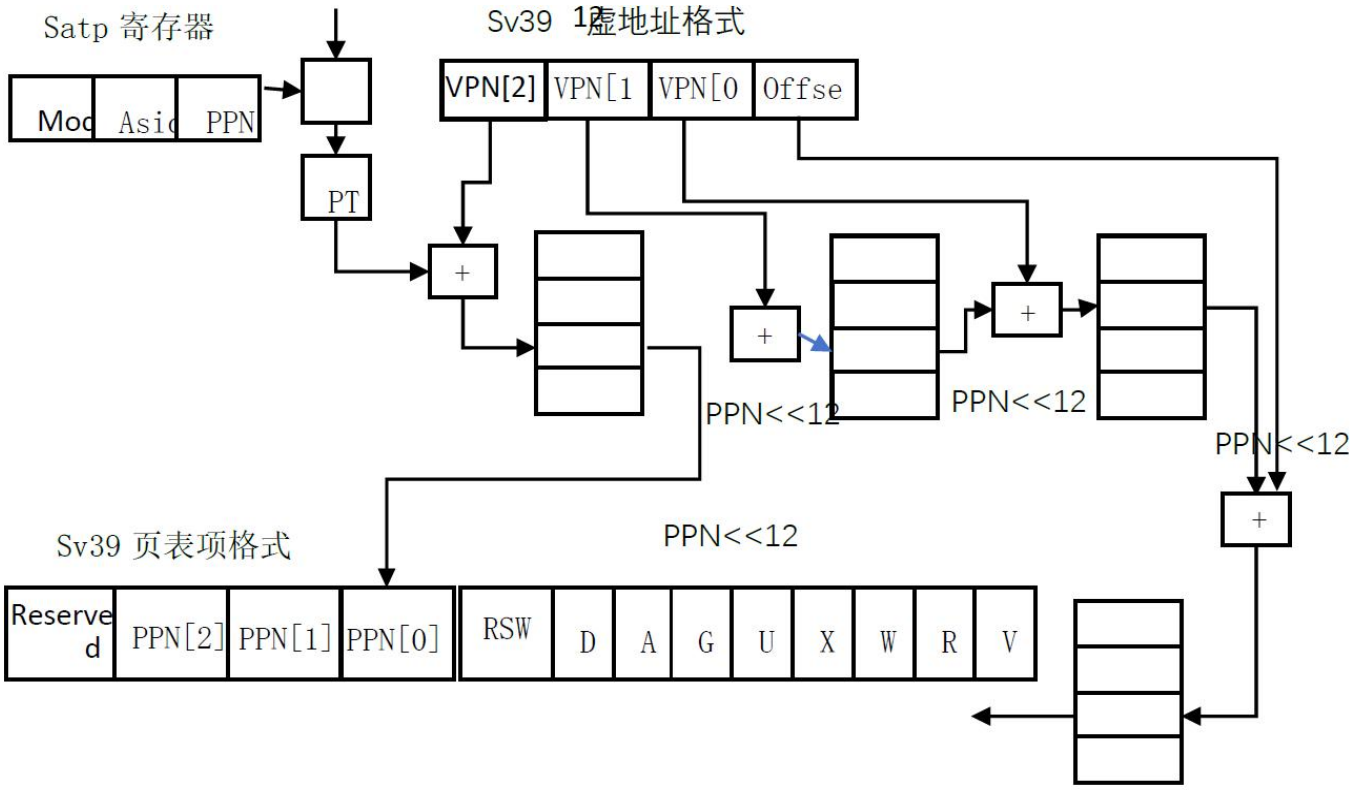

图 5 Sv39 虚实地址转换过程

3.4 超标量流水线

超标量架构技术通过指令级并行机制突破传统架构单周期单指令发射的限制，实现单周期多指令并发执行。相较于 CISC 架构，得益于精简指令集特性，RISC 在同等主频下展现更优的超标量执行潜力[18]。针对超标量流水线的数据相关性挑战，本研究选择 Tomasulo 算法作为核心调度框架，相较记分牌方案具有更广泛的数据相关性处理能力[19]。

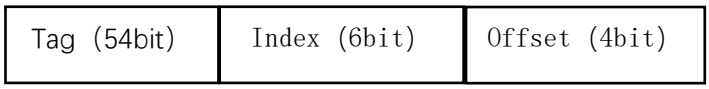

图 6 存储器地址的划分

而经典的 Tomasulo 算法是“乱序执行、乱序提交”，无法使得程序按原本的予以执行，这给调试、中断、异常处理等过程带来了不小的困难。因此本文的设计中引入了 ROB（重排序缓存）来使得指令得以顺序提交。整个算法的流程是这样的：在取指阶段取出 2 条指令送入 IBuffer（IBuff 一共可以存放 4 条指令，若空位小于 2 则阻塞），之后每个周期送入 2 条指令进行译码，译码后送入发射队列（Issue Queue），如果这条指令的两个（或一个）操作数准备就绪且对应执行单元空闲，则可以将操作数送入指令单元。执行完毕后将结果写入寄存器堆（PRF，Physical Register File）同时给 ROB 对应的项设置标志位。ROB 保存着所有未退休指令的状态，一旦某一条指令成为 ROB 中最“老”的指令它就可以退休，同时更新 PRF 中对应寄存器的值，这样就保证指令的顺序提交。

经典 Tomasulo 算法采用“乱序执行、乱序提交”机制，导致程序实际执行顺序可能偏离源码逻辑，这对调试定位、中断响应与异常恢复造成挑战。为此，本文设计引入重排序缓存（ROB）实现指令顺序提交机制。算法工作流程如图 7 所示：取指模块每周期从指令存储器提取两条指令存入 IBuffer（容量 4 条，剩余空间不足 2 条时暂停取指），译码模块每周期从 IBuffer 读取两条指令进行译码，解析完成的指令进入发射队列等待调度。当指令的源操作数就绪且对应功能单元空闲时，该指令被发射至执行单元。运算完成后，执行结果同时写入物理寄存器堆（PRF，Physical Register File）和 ROB 对应表项，并标记该结果有效状态。ROB 持续跟踪所有未退休指令的生命周期，当某条指令成为 ROB 中最旧的条目时，即可触发退休流程——将该指令 ROB 中移除，并同步更新 PRF 中的寄存器映射关系，从而确保全局状态始终按程序顺序演进。

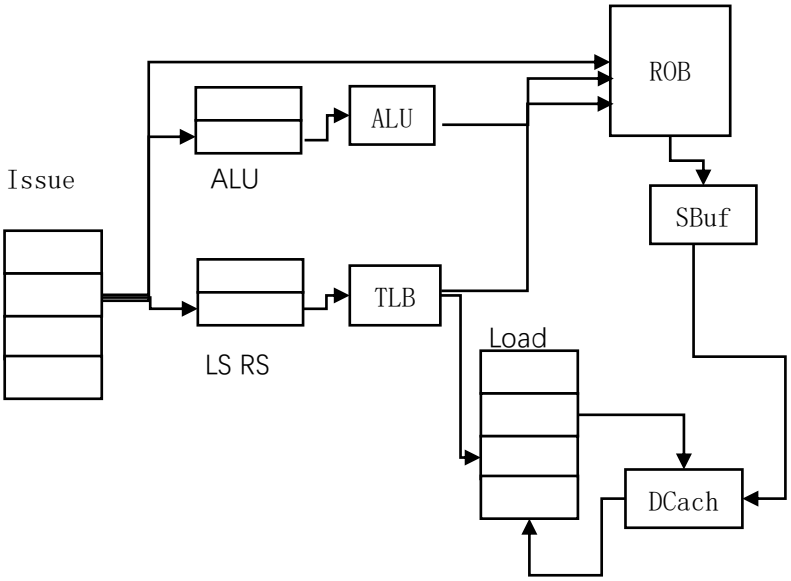

图 7 Tomasulo 算法示意图

4 功能验证

本研究采用的测试基准源于 riscv-tests 项目，该测试集由 Krste Asanovic 与 David Johnson 等学者主导开发[19]。作为面向 RISC-V 架构的标准化验证工具，其核心功能在于检测处理器实现是否符合指令集规范要求。鉴于 RISC-V 开放指令集的特性，确保硬件实现与标准的高度一致性成为设计验证的关键环节。

该测试套件包含 isa 与 benchmark 等核心模块，其中 isa 目录提供指令级验证工具，具体涵盖用户态测试（如 rv32ui-p-*系列验证 ADD/SUB 等基础运算）与特权态测试（如 rv32mi-p-*系列验证 CSR 操作与系统调用）。测试文件采用 TVM（测试向量标记）+目标环境的复合命名体系，具体编码规范详见表 2 与表 3。benchmark 目录集成性能评估组件，包含驱动模块与测试桩程序，支持开发者开展处理器验证工作。

表 2 TVM 名称含义

TVM名称	含义
rv32ui	RV32 用户级，仅限整数
rv32si	RV32 监督级，仅限整数
rv64ui	RV64 用户级，仅限整数
rv64uf	RV64 用户级，整数级和浮点级
rv64uv	RV64 用户级，整数、浮点和向量
rv64si	RV64 监督级，仅限整数
rv64sv	RV64 监督级，整数和向量

表 3 目标环境的名称含义

目标环境名	含义
P	虚拟内存已禁用，只有核心 0 启动
Pm	禁用虚拟内存，所有内核启动
Pt	禁用虚拟内存，定时器每 100 个周期中断一次
V	虚拟内存已启用

测试代码采用 C 和汇编语言实现，Riscv-tests 测试套件提供编译命令生成 ELF 及 DUMP 中间文件。ELF 作为标准的可执行文件格式，通常依赖操作系统进行加载执行。鉴于目标处理器未搭载操作系统支撑，需将 ELF 转换为二进制镜像文件（BIN 格式）。通过设定存储器的初始加载地址，将 BIN 文件载入指令存储器后即可启动功能验证。经实测验证，本设计已完整通过 RV64I 基础指令集测试，其行为符合 RISC-V 指令集架构规范。图 8 为指令通过测试的运行结果。

```
[info] Run completed in 4 seconds, 522 milliseconds.
[info] Total number of tests run: 1
[info] Suites: completed 1, aborted 0
[info] Tests: succeeded 1, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 12 s, completed Feb 11, 2025, 5:36:06 AM
```

图 8 riscv-test运行结果，表明指令通过测试

5 结束语

本文基于新型的敏捷开发语言 Chisel 设计了一款基于 RISCV 的处理器，具有分支预测、寄存器重命名等技术，并且通过了 riscv-tests 的验证，保证了其正确性与合规性。本文设计的处理器还有进一步优化的空间，可以结合 RISC-V 手册实现拓展指令集如 V、M，以及可以在双核技术上做进一步研究来提升效率。

参 考 文 献

[1] Ali W. Exploring Instruction Set Architectural Variations: x86, ARM, and RISC-V in Compute-Intensive Applications[J]. Authorea Preprints, 2023 ,1(3).

[2] 郭倩、中国工程院院士倪光南.RISC-V 正构建全球主流 CPU 新格局[N].经济参考报,2024 ,21(7).

[3] Saidova J. RISC-V architecture and its role in the near future[J]. Journal of Advanced Scientific Research (ISSN: 0976-9595), 2024, 5(9).

[4] Asanovic K, Avizienis R, Bachrach J, et al. The rocket chip generator[J]. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, 2016, 4(6).

[5] 包云岗. “香山”处理器开源开发实践[J].软件和集成电路,2024,6(9):33-34.

[6] 王彤、陈景柱、邓庆绪等. 面向新工科的计算机类专业软硬件协同创新实践教学模式探索[C].计算机技术与教育学报.2022 ,10(5):103-110.

[7] 张彦航、战德臣、原峁等. 基于多校克隆班的“场景构建式、图谱理解式”硬件创新能力迭代递进式培养[C]. 计算机技术与教育学报 ,2025 ,12(3):172-176.

[8] 王毅航,毛强,薛菲菲.智能硬件课程建设介绍及智能硬件教学仪的研制[C]. 计算机技术与教育学报,2022 ,10(5):126-131.

[9] 钱玉娟. 玄铁探路 RISC-V[N].经济观察报,2024 ,1163(17).

[10] 郭靖,孙艺通,黎中有,等. 敏捷开发方法的实践与思考[J]. 数字技术与应用,2024,42(9):132-135.

[11] Y. Lee et al. An Agile Approach to Building RISC-V Microprocessors[J]. IEEE Micro, 2016,36(2).

[12] Im J, Kang S. Comparative analysis between verilog and chisel in risc-v core design and verification[C].2021 18th International SoC Design Conference (ISOCC). IEEE, 2021 ,36(2): 59-60.

[13] P. Lennon and R. Gahan, A Comparative Study of Chisel for FPGA Design[C]. 2018 29th Irish Signals and Systems Conference (ISSC), Belfast, UK, 2018 ,29(1): 1-6,

[14] Waterman A, Lee Y, Patterson D A, et al. The RISC-V instruction set manual, volume I: User-level ISA, version 2.0[J]. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54, 2014 ,1(1): 4.

[15] Bachrach J, Vo H, Richards B, et al. Chisel: constructing hardware in a scala embedded language[C].Proceedings of the 49th Annual Design Automation Conference. 2012 ,49(52): 1216-1225.

[16] Shen J P, Lipasti M H. Modern processor design: fundamentals of superscalar processors[M]. Prospect Heights:Waveland Press, 2013.

[17] I. Pantazi-Mytarelli, The history and use of pipelining computer architecture: MIPS pipelining implementation[C]. 2013 IEEE Long Island Systems, Applications and Technology Conference (LISAT), Farmingdale, NY, USA, 2013,9(1).

[18] Patel, Mr. A Review on Superscalar Technology with instruction level parallelism (ILP) for Faster Microprocessor[C]. International Journal for Research in Applied Science and Engineering Technology. 2018 ,6(3).

[19] 王磊. Tomasulo 算法与记分牌调度算法研究[J]. 自动化技术与应用,2013,32(06):23-26.

[20] RISC-V Software contributors. riscv-tests[EB/OL].[2025] https://github.com/riscvsoftware-src/riscv-tests.