韶关学院

毕 业 设 计

题　　目： 基于 Chisel 的精简指令集微处理器设计

学生姓名： 荀阳霖

学　　号： 19125042040

二级学院： 信息工程学院

专　　业： 物联网工程

班　　级： 20 物联网 02

指导教师： 王娜　　**职　　称** 讲师

起止时间 2024 年 03 月 12 日

基于 Chisel 的精简指令集微处理器设计

摘要： 写完再摘

高效抽象的处理器芯片 硬件设计语言与工具.快速迭代的芯片设计与 系统级原型验证方法(敏捷开发)让开发硬件像开发软件那 么容易 高质量芯片人才需要掌握计算机整体的软硬件理论知识与全程工程技术。既要在“学中做”，也要在“做中学”[1]。本文作者通过设计可运行基本操作系统的处理器芯片并完成投片。尝试贯穿芯片设计全链条，结合贯通计算机专业的诸多课程。 完成知识的有机融合，打通从数字电路、组成原理、体系结构、操作系统、嵌入式开发的全流程知识点。

关键词： 我；就是；测试用；关键词

Chisel implementation of a RISC microprocessor

ABSTRACT: English abstract

Key words: Dummy; Keywords; Here; It Is

目　　录

绪论

1.1 研究工作的背景与意义

世界正在加速转型进入数字经济时代。而做为数字化引擎的处理器芯片，则是一切的基石。虽然信息领域市场巨大，竞争强烈，但十几年来却已经形成了“赢家通吃”的场面。高精尖技术与人才长期被把握在少数西方大公司的壁垒之中。

随着中美贸易环境的变化和技术封锁措施，一些中国企业和高校被列入美国政府的‘实体清单’，这对中国芯片技术研发和人才培养带来了显著影响。目前，我国的计算机专业人才培养面临着较大的结构问题。顶层应用开发者过多，而底层软硬件研发人员缺乏。特别是芯片设计人才严重不足。2022 年间我国集成电路产业人才缺口达到 3.5 万[2]。这与当前芯片设计门槛过高，导致中国大学无法开展芯片相关教学与研究密切相关[3]。

晶体管性能高速提升的时代即将结束，半导体行业即将进入后摩尔时代。提升芯片性能与能效的压力逐渐转移到了架构设计师与电路设计师上[4]。对领域特定架构的需求日渐增加，产业界急需一种更加快速、灵活的研发方法和基础架构。

RISC-V 是一款开源且免费的精简指令集架构。自从该指令集架构发布以来，在国内外吸引了许多关注。

如今，已有来自 70 多个国家的 3900 多个实体加入了 RISC-V 联盟，其中高级成员有超过 40% 来自中国。国内外越来越多的公司、组织和开发人员正在支持和贡献 RISC-V。包括芯片设计商、软件开发商、工具提供商以及三星电子、西部数据和英伟达等行业巨头。开放的 RISC-V 架构鼓励了全球范围内的协作和技术创新，降低了进入壁垒，极大地促进学术界和创业公司在芯片设计领域的创新活动。其开源免费、模块化、容易定制等特点，为我国芯片设计的发展带来了新的机遇。有助于我国掌握核心关键技术的同时开放地对待国际交流与合作。

物联网时代的到来对处理器芯片提出了更多的需求。RISC-V 的模块化设计允许制造商根据特定应用场景进行裁剪或扩展，设计出高度优化的芯片解决方案，降低功耗、提升性能并缩小芯片面积，进而降低成本。从而满足物联网行业中多样化、碎片化的市

场需求。 同时， 鉴于物联网设备数量庞大且分布广泛，其安全问题愈发凸显。RISC-V 因其透明的架构设计，有利于定制深度的防御策略，确保从硬件层面强化安全特性。诸如“港华芯”[5] 这样的 RISC-V 物联网安全芯片的成功案例表明，RISC-V 能够有效应对物联网安全挑战，并在能源、智慧城市等多个领域实现关键突破。

1.2 国内外研究现状

注：这块没写完

RISC-V 具有简洁，现代的特点，受到了国内外的广泛关注。

在国际上，伯克利大学基于 RISC-V 指令集架构，使用 Chisel 语言构建来开源的 SoC 生成器，Rocket Chip[6] 。由伯克利大学的研究团队于 2012 年推出。该生成器支持生成有序执行内核（Rocket）与乱序执行内核（Boom），还支持指令集扩展，协处理器等功能。用户通过配置文件描述设计后，Rocket Chip 生成器可以自动挑选所需的模块组合到设计中。

在国内，睿思芯科仅用了 7 个月的时间[7]，就完成了一款基于 RISC-V 的 64 位可编程终端 AI 芯片。这相比传统芯片设计效率提高了大概三倍。

中科院香山团队在 2021 年成功设计并流片了第一版雁栖湖架构处理器。在 28nm 的工艺节点下达到 1.3GHz 的频率。而后在 2023 年成功设计并流片了第二版南湖架构处理器，在 14nm 工艺节点下频率达到 2GHz。 香山处理器的目标是成为面向世界的体系结构创新开源平台，基础能力、设施、流程的建立是香山处理器长期高质量发展的关键，香山团队设计了丰富的基础工具支撑起了这一复杂度量级的敏捷验证流程。

1.3 内容与结构安排

处理器架构与敏捷开发

2.1 指令集架构

2.1.1 复杂指令集与精简指令集

指令集发明于 1960 年代，当时 IBM 为了解决该公司不同计算机产品线的程序无法通用的情况。 推出了完全独立于硬件架构的指令集架构（ISA）这一概念。

在当时，程序员通常直接使用汇编语言进行开发。因此市场普遍认为指令集应当更加丰富，每条指令本身应该能实现更多功能。 因此，当前一些高级语言中常见的概念，如函数调用、循环、数组访问都有直接对应的机器指令。这使得早期的指令集均为复杂指令集（CISC）。

1980 年开始，随着存储器成本大幅下降、指令数目持续增加、高级语言的普及，CISC 的一些缺陷开始暴露。 指令利用率低、控制电路趋于复杂、研发周期过长、验证难度增加等[8] 催生了对于精简指令集（RISC）的研究。

与 CISC 不同，RISC 强调每条指令的功能单一化。通过指令的组合来完成复杂操作。由于指令复杂程度下降，编译器更容易进行代码生成。同时节约的电路面积可以用于缓存或流水线等高级结构，RISC 在成本与性能上均好于 CISC

2.2 RISC-V 指令集

2010 年，David Patterson 等人出于教学目的，在仔细评估了市面上各类指令集后，发现它们都存在设计上的不足[9] 。 决定重头开始，吸取指令集架构领域 25 年期间的经验教训。设计了一款新的开源指令集架构—— RISC-V。

2.2.1 RISC-V 指令集的优势

- 基础加扩展式的 ISA：RISC-V 架构以模块化的方式组合在一起，用户可以根据需求选择不同模块以满足不同场景的应用。
- 紧凑的代码体积：智能物联网产品对成本和存储空间通常较为敏感，RISC-V 提供统一的指令压缩扩展， 可以在同样的大小下存储更多指令，并且不对成本产生显著影响。

- 开放：传统 ISA 高额的授权费和严格的使用限制对于学术界和初创企业来说是巨大的负担。而 RISC-V 是开源架构,其规范可公开获取,无需支付专利费或许可费，有助于减少行业依赖于少数专有指令集架构的风险，促进公平竞争和市场活力。
- 简洁： RISC-V 在设计之初参考了大量现有 ISA 的经验与缺点，设计更加现代化，基础指令只有 40 余条。避免了对某一特定微架构的过度优化，允许研究者在不被各类复杂的兼容性需求约束的情况下，自由探索各类微架构设计。
- 灵活:RISC-V 允许自定义和扩展指令集架构,使芯片设计师能够根据业务需求量身定制架构。 这种灵活性使 RISC-V 在物联网设备到高性能计算系统等广泛应用领域都具有吸引力。
- 便于教育和研究:RISC-V 作为简洁且现代的开源架构，使其成为体系结构教学与研究的绝佳平台。许多大学和学术机构已采用 RISC-V 进行教学,并开发创新的硬件和软件解决方案。
- 完整的软件支持：RISC-V 提供完整的软件堆栈，编译器工具链以及继承开发环境和操作系统支持。

2.3 芯片敏捷开发方法及其应用

敏捷开发是一种现代软件开发方法论，它倡导迭代、增量开发，强调灵活性和响应变化的能力。这种方法论源于对传统瀑布模型的反思与改进，后者往往在一个阶段结束后才转入下一个阶段，各个阶段间耦合度较高，且不易适应需求变更。进而发展为项目延期，客户需求无法满足，甚至项目失败。

在敏捷开发中，项目被分解为一系列短周期（称为冲刺或迭代），每个周期专注于交付可用的产品增量。团队在整个开发过程中持续集成、测试和交付，保证软件质量。敏捷开发的核心价值观和原则体现在《敏捷宣言》[10] 中，包括重视个体和交互胜过流程和工具、可工作的软件高于详尽的文档、与客户协作并积极响应变化、以及持续交付并欢迎更改需求。

2.3.1 敏捷开发的优点

传统的芯片开发流程包含一个固定的‘代码冻结’阶段, 在此阶段之后, 代码修订受到严格限制。这就导致了后端物理设计团队的反馈不能及时有效地帮助前端架构团队改进设计。

而使用敏捷思想指导开发的硬件，在完成每一个功能点的编写后，前后端团队都会一起进行测试与验证。为快速改进设计提供了可能。

基于生成器的设计促进了模块在项目之间的重用。敏捷开发可大幅降低芯片的设计周期，并在市场需求变化时快速响应。通过敏捷开发，伯克利大学的研究团队成功在五年内开发并生产了 11 种不同的处理器芯片。[11]。

2.3.2 敏捷开发的意义

与传统互联网时代相比，智能物联网时代（AIoT）的设备成本功耗受限，专用性强。传统上追求性能和通用的计算芯片不再适用。需要根据细分领域的需求，软硬件件协同优化，深度定制领域专用处理器（XPU）。 以满足特定场景对于芯片的成本，性能与能效需求。[12]

这将使不同 AIoT 设备对芯片的需求差异化、专一化、碎片化。 如果继续沿用传统的瀑布式开发流程，过长的芯片研发和上市时间显然无法满足海量 AIoT 设备的快速定制需要。

2.3.3 芯片行业的敏捷开发

在芯片设计领域中, 敏捷开发意味着在设计初期就利用高级语言快速构建模拟器原型, 软件的灵活性允许工程师快速尝试各种设计方案，尽早获得反馈, 并迅速迭代设计。设计成熟后，敏捷开发流程允许团队在较短时间内将设计迁移到 FPGA 上进行验证和调试，FPGA 的高速和高准确性允许工程师在这一步运行完整的操作系统与测试程序。同时与客户紧密合作，确保芯片设计能满足客户需求。设计过程中，设计完成后，可以通过电子设计自动化（EDA）工具对芯片电路的面积、功耗和时序进行细致的分析。整个过程中，前端架构和后端物理团队同步协作，共同参与每次迭代的验证和优化，从而显著缩短了设计周期，提高了产品质量。

当芯片电路设计敲定后，就可以与晶圆厂对接，准备进行投片生产。

整个设计流程如 图 2.1 所示。其前三步都可以通过敏捷开发大幅提高效率与质量。

图 2.1 芯片开发流程

2.4 Chisel 语言在芯片设计中的优势及其实现原理

目前，最流行的两种硬件定义语言（HDL）是 Verilog 和 VHDL[13]。硬件工程师通过 HDL 描述硬件电路的具体行为与结构。Verilog 与 VHDL 最初都是为硬件仿真而创建的，后来才被用于综合。这些语言缺乏抽象能力，使得组件难以在项目之间重复使用。

Chisel[14] 是一种基于高级编程语言 Scala[15] 的新型硬件构造语言（HCL）。由伯克利大学的研究团队于 2012 年推出。Chisel 在 Scala 中提供了各类硬件原语的抽象，使得工程师者可以用 Scala 类型描述电路接口，以 Scala 函数操作电路组件。这种元编程可实现表现力强、可靠、类型安全的电路生成器，从而提高逻辑设计的效率和稳健性。

需要说明的是，虽然 Chisel 具有许多传统硬件描述语言不具备的高级特性，但其还是一门硬件构造语言，而不是高层次综合。Chisel 代码在运行后会生成对应硬件的形式化描述，以寄存器传输语言灵活中间码（Flexible Intermediate Representation for RTL，FIRRTL）形式记录[16]。

FIRRTL 随后被输入硬件编译框架(Hardware Compiler Framework，HCF) CIRCT[17]。经过多次递降变换（lowering transformations）[18]，逐级去除高层次的抽象，最终输出优化后的底层 RTL 代码（如 Verilog）。使用 HCF 的好处之一是其内置了大量的优化器，可

进行常量传播、共用表达式合并、不可达代码消除或向专用集成电路（ASIC）与现场可编程门阵列（FPGA）提供针对性的优化[19]。这些特点使工程师能够更专注于逻辑功能的设计，从而显著提升设计效率与代码质量。

2.4.1 Chisel 的优点

开发高效

Chisel 语言的高效主要体现在四个方面：

- 信号整体连接： Chisel 丰富的类型系统和连接功能。允许工程师以组件化的方式声明和连接多个信号线，并进行整体连接。 从而简化了总线接口修改的工作流程。避免了在使用传统 HDL 开发时，需要手动更新多个模块的端口声明而产生潜在错误。SystemVerilog 虽提供类似的功能，但其端口定义不可嵌套，仍然无法达到良好的抽象程度。
- 元编程： Chisel 可以抽象出多份相似模块的共用部分,通过使用参数化的硬件模板，工程师可以创建出可复用的硬件库。有效减少了代码冗余。SystemVerilog 虽然提供类似的功能，但仅用于验证[20]，属于不可综合代码。
- 面向对象: 通过继承, 工程师可以把 Chisel 中多个电路模块的共性抽提成一个父类。在实现具体模块时，只需通过继承的方式，就可以让模块自动拥有父类的所有特征。从而减少代码冗余，便于分层次设计。
- 函数式: Chisel 的函数式特性将允许工程师使用 map 或 zip 算子实现电路级联的批处理操作，可以简洁地描述数据流。如 图 2.2 所示，级联的算子操作可以直接对应生成后的电路。与 Verilog 的数据流建模相比，工程师不再需要关心运算符优先级。

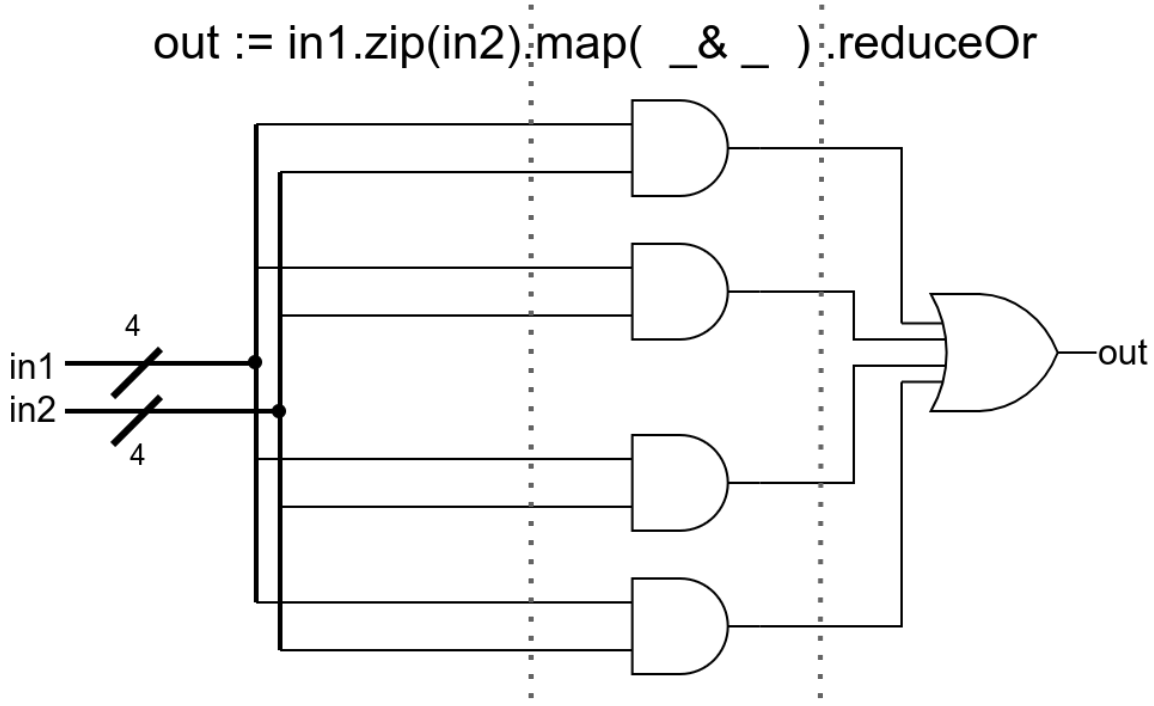

图 2.2 函数式操作与电路设计紧密对应

代码质量更优

余子濠等人（2019）的研究[21] 指出。 在 FPGA 上，与经验丰富的专职工程师使用 Verilog 相比, 经训练的本科生使用 Chisel 的高级特性开发同样功能一致的二级缓存模块。 无论是能效、频率还是资源占用，均优于 Verilog 实现，且代码可维护性优更优。

表 2.1 资源占用与性能对比

类型	Verilog	Chisel
最高频率/MHz	135	154
功耗/W	0.77	0.74
LUT 逻辑	5676	2594
LUT RAM	1796	1492
触发器	4266	747
代码长度	618	155

在开发速度方面，专职工程师使用 Verilog，消耗至少 6 周完成开发。而另一位有 Chisel 经验的本科生则在 3 天内完成了所有工作， 而且代码长度显著短于前者。

而在芯片整体设计方面，吕治宽（2022）的研究[22] 表明，使用两种语言开发同一微结构的 RISC-V 处理器。Verilog 的时间消耗是 Chisel 的 3 倍。

处理器微架构设计与实现

本文将首先实现一个单发射，简单流水线，顺序执行的基本 RV32E 扩展处理器芯片微架构。

为方便拓展，本微架构流水级之间使用异步定时方式通信，通过握手信号控制数据是否流动。每个流水级的行为只取决于自身和下游模块的状态，流水级均可以独立工作。通过这种设计，避免了对全局控制器的需求，进而简化添加指令和流水级的难度。也为探索乱序执行做好了准备。

在逻辑上，为了方便优化，本微架构分为以下几个基础单元：取指单元，译码单元，执行单元，访存单元，写回单元。

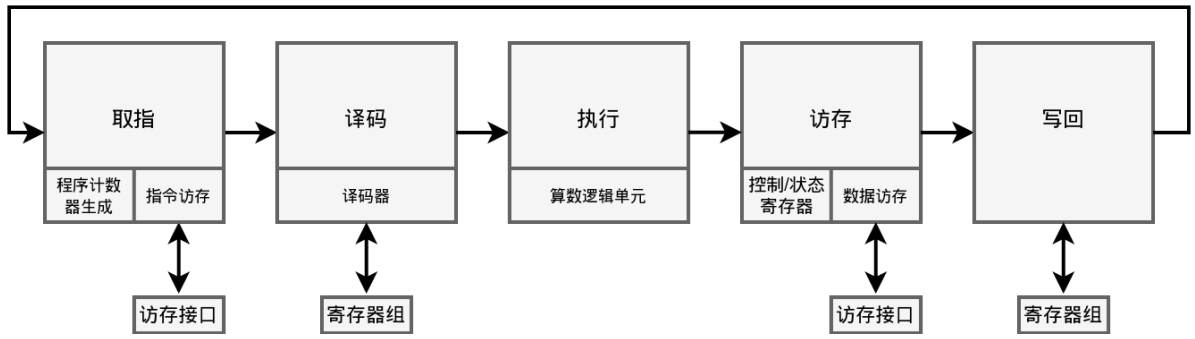

图 3.1 整体单元划分

3.1 信号握手

如 图 3.2 所示，每个基础单元均可以根据自身情况产生“准备/有效（Ready / Valid）”握手信号。Valid 标识模块当前的数据（Data）是否有效，Ready 标识本模块是否能接收 Data 输入。

模块间遵循 表 3.1 的语义进行传输握手。此外，为防止数据丢失，当模块表示当前数据有效后，除非数据成功传输，否则不能撤销有效信号。本级有效而下级忙的情况称为发生数据反压（Back pressure）

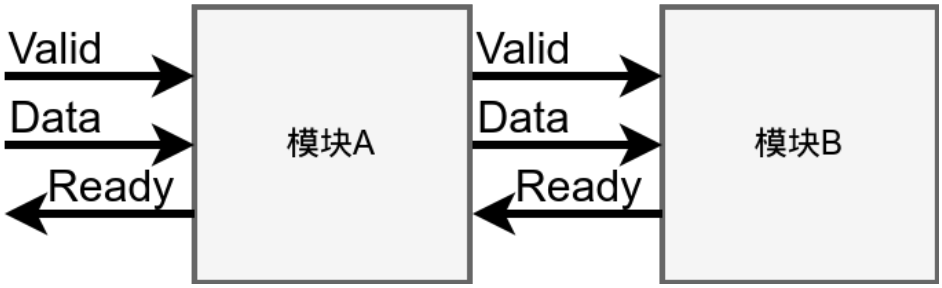

图 3.2 Ready / Valid 握手信号

表 3.1 握手信号行为说明

下级 Ready	本级 Valid	数据传输行为
有效	有效	数据成功传输至下级模块
有效	无效	当前数据无效 下级模块需等待
无效	有效	下级模块忙 本级需保持 Data 并等待
无效	无效	不发生数据传输

【接下来的这几个单元有必要写那么详细吗。我感觉我在说废话】

3.2 取指单元

取指单元的总体架构如图 3.3。

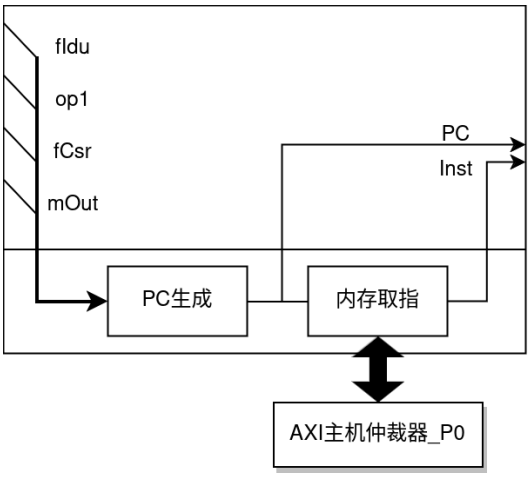

图 3.3 取指单元总体架构

取指单元根据上一条指令的译码、执行结果修改程序计数器（PC），并取出下一条指令。 整个取值单元分成 PC 生成器及指令访存器。

3.2.1 PC 生成器

PC 生成器将根据当前处理器的状态以及指令来确定下一执行周期的程序计数器的值。 目前，程序计数器生成器支持以下行为:

1. 顺序执行

2. 普通跳转（不切换特权级）

3. 暂停执行

4. 自陷指令——陷入

5. 自陷指令——返回

RISCV 架构支持比较两个通用寄存器的值并根据比较结果进行分支跳转（B 系列指令）。无条件跳转（J 系指令）， 也支持在异常发生时进入异常处理程序并返回。PC 寄存器的结构如图 3.4 所示,其行为如表 3.2 所示。

表 3.2 PC 生成表

类型	add1	add2	next_pc	PC 寄存器
顺序执行	pc	4	sum_next_pc	允许写入
无条件跳转	pc	立即数	sum_next_pc	允许写入
条件跳转	pc	据结果选择	sum_next_pc	允许写入
异常陷入	无关	无关	mtVec	允许写入
异常返回	无关	无关	mePC	允许写入
反压暂停	无关	无关	无关	禁止写入

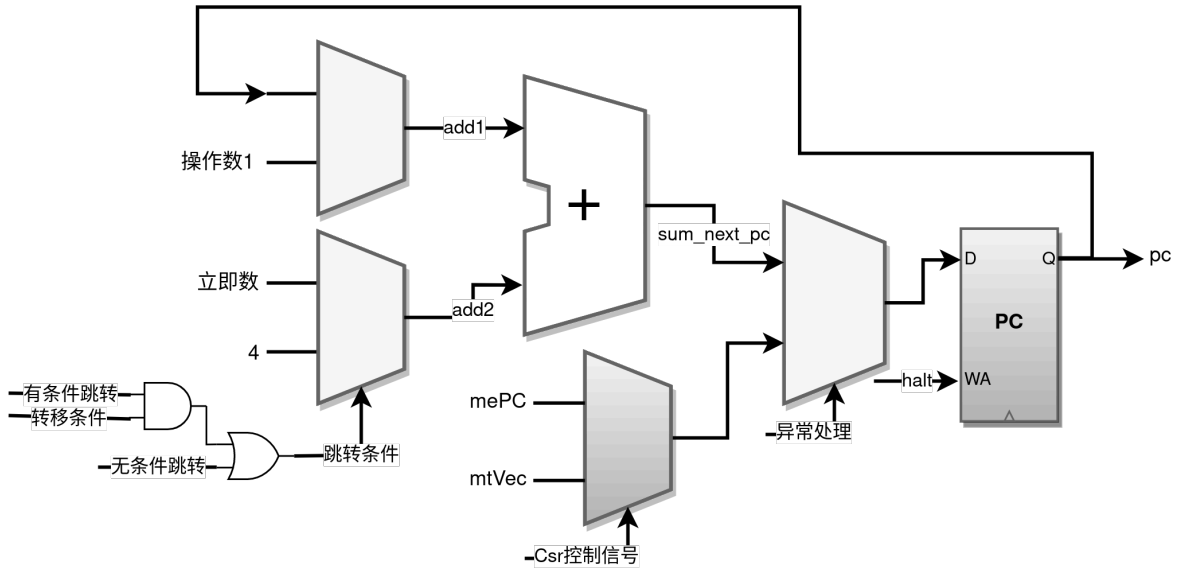

图 3.4 PC 生成器结构

3.2.2 指令访存器

指令访存器的任务是根据当前 PC，向处理器核内的 AXI[23] 主机仲裁单元发起访存请求。其结构如图 3.5 所示。在复位后，指令访存控制器在 `idle` 状态等待有效的访存事务发生。收到访存请求后，按照表 3.3 的行为对取值器的输出端口进行操作。需要注意的是，该状态机的输出均来自寄存器。

由于 AXI 协议规定在读事务完成时不再保持数据有效信号。在读取完成后，指令执行中的若干周期，指令有效信号 将由状态机提供【是这样的吗。我要确认一下】

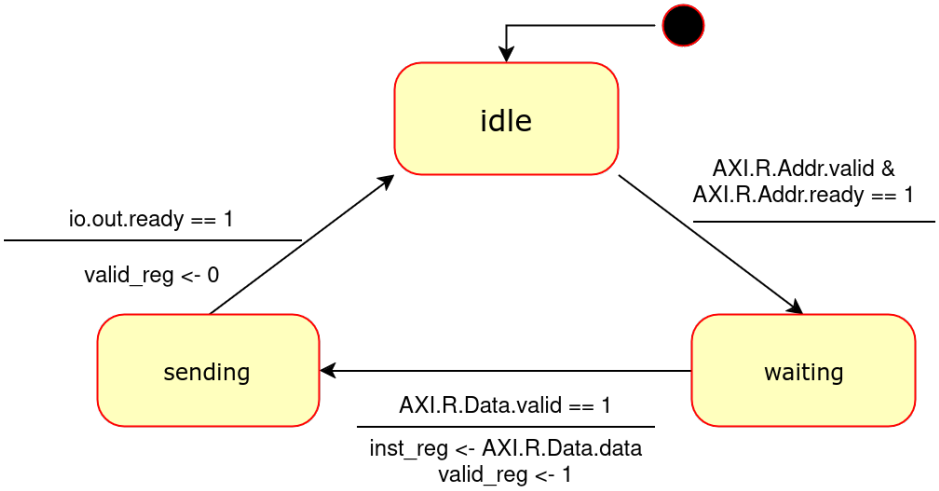

图 3.5 取指状态机

表 3.3 取指单元状态机转移说明

状态名	说明	离开条件
idle	当前未在访问指令存储器	存储器 AXI 读握手成功
waiting	读请求已发出。等待指令存储器回复	存储器回复数据
sending	向 CPU 下游部件提供指令码	下游部件汇报准备完成

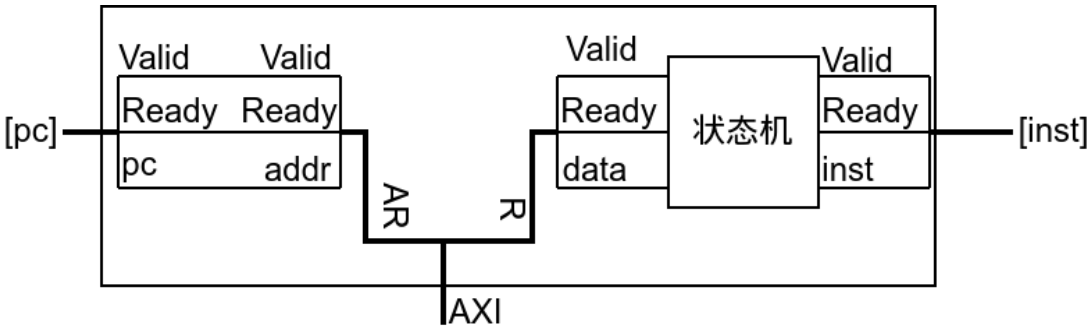

图 3.6 指令访存器结构

3.3 译码单元

译码单元的任务是任务是根据当前指令，通过译码器产生各类控制信号；从寄存器组中读出数据。

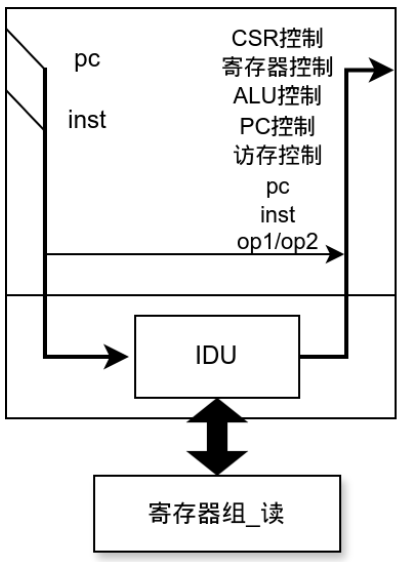

图 3.7 取指单元总体架构

3.3.1 译码器

译码器主要功能如下:

- 访问寄存器: 由【RISCV 指令格式图】所示, RISC-V 指令集中，对于需要访问寄存器的指令，其寄存器编号均在指令的固定位置。为简化电路，译码器被设计为不对当前指令类型进行判断，固定将 24～20 与 19～15 段的内容视为寄存器编号，进行寄存器访问。
- 立即数产生与符号扩展: 由【RISCV 指令格式图】所示, RISC-V 指令集中，对于具有立即数的指令, 无论立即数长度，其符号位固定位 31 位。 为简化电路，符号扩展被设计为固定使用 31 位做为符号进行扩展，各下级模块由解码结果决定是否使用。在后续设计中，将根据时序报告决定是否将本部件移动到执行单元。
- 控制信号生成: 根据指令类型和具体指令操作。为下级模块提供控制信号。【我觉得还是把这里优化成 chisel decoder 吧】

【要解释所有控制信号的用途吗?】

3.4 执行单元

执行阶段是微架构数据流的核心，如 图 3.8 所示。其任务是根据译码单元产生的控制信号，对操作数进行逻辑运算。

【todo：下面图的输出没有 pc 和 inst】

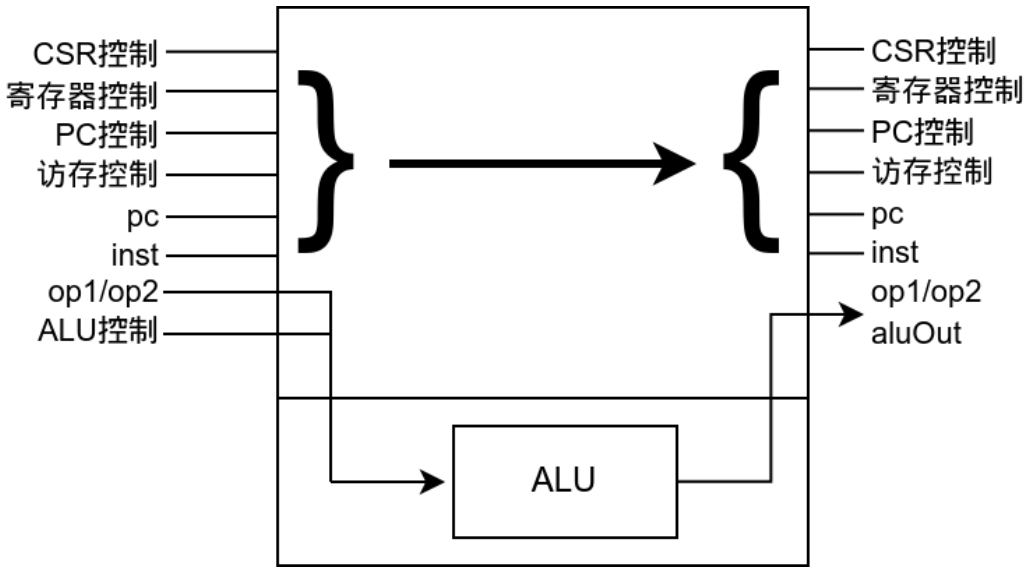

图 3.8 执行单元总体架构

算数逻辑模块（ALU）结构较为简单，其组成如 代码 3.1 所示。

```
io.out :=
 MuxLookup(io.fIdu.aluOp, 0.U(XLEN.W)) {
  Seq(
    AluOpTypes.sub -> (op1 - op2),
    AluOpTypes.add -> (op1 + op2),
    AluOpTypes.eq -> (op1 === op2),
    AluOpTypes.op2 -> (op2),
    AluOpTypes.neq -> (op1 =/= op2),
    AluOpTypes.less -> (op1 < op2),
    AluOpTypes.s_less -> (sop1 < sop2),
    AluOpTypes.ge -> (op1 >= op2),
    AluOpTypes.s_ge -> (sop1 >= sop2),
    AluOpTypes.or -> (op1 | op2),
    AluOpTypes.and -> (op1 & op2),
    AluOpTypes.xor -> (op1 ^ op2),
    AluOpTypes.sl -> (op1 << shamt),
    AluOpTypes.sr -> (op1 >> shamt),
    AluOpTypes.s_sr -> (sop1 >> shamt).asUInt,
    AluOpTypes.op1p4 -> (op1 + 0x4.U)
  )
}
```

代码 3.1 ALU 模块主要 Chisel 代码

3.5 访存单元

访存单元是为了装载（LOAD）与储存（STORE）指令设计的。其设计如 图 3.9 所示。 为了提高流水线后各阶段的效率，设计时分析了 LOAD/STORE 指令的资源需求，决定将 CSR 的读写合并于本模块内。

根据当前执行的指令不同, 模块顶层的 Ready/Valid 生成器会选择旁路或使用来自数据访存器的 Ready/Valid 信号。

要提一下根据 rv 的 load store 指令，aluOut 上是目标内存地址

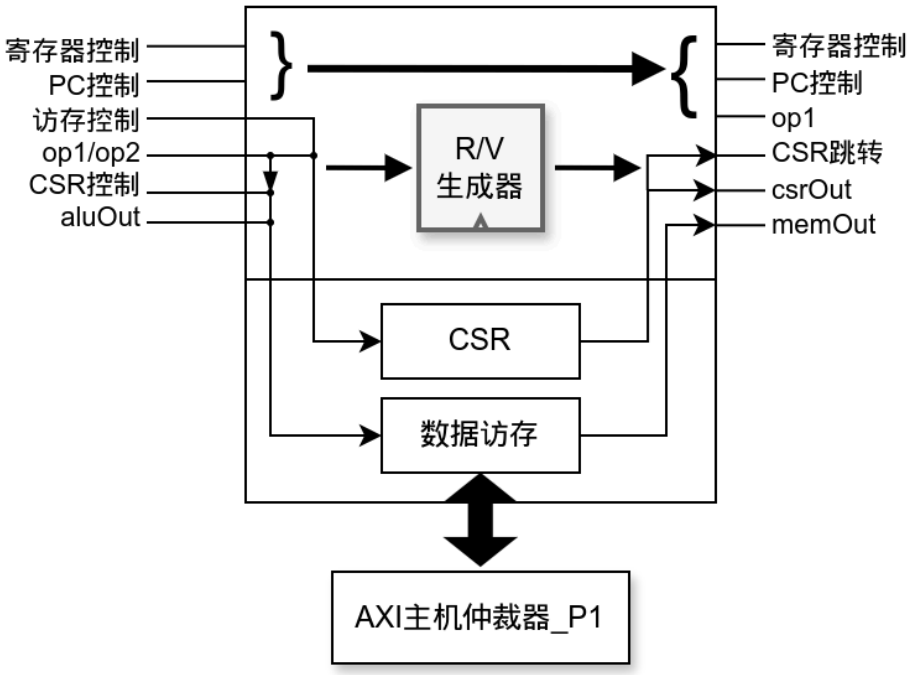

图 3.9 访存单元总体架构

3.5.1 数据访存器

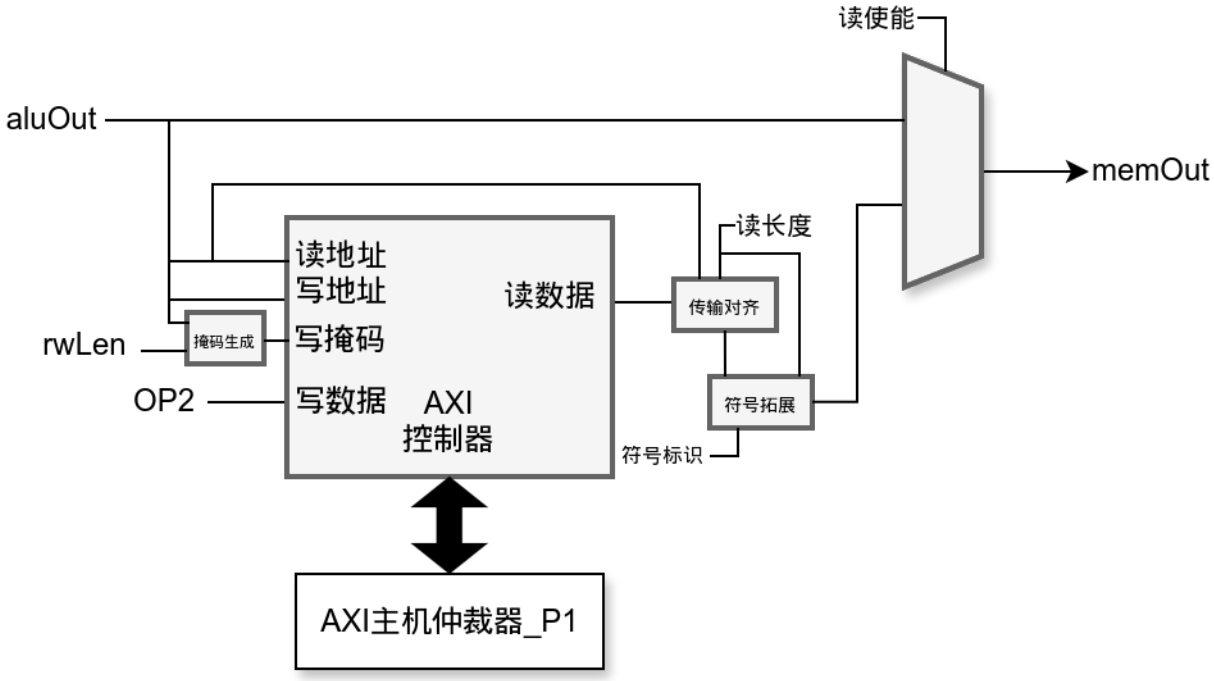

图 3.10 数据访存器架构

3.6 写回单元

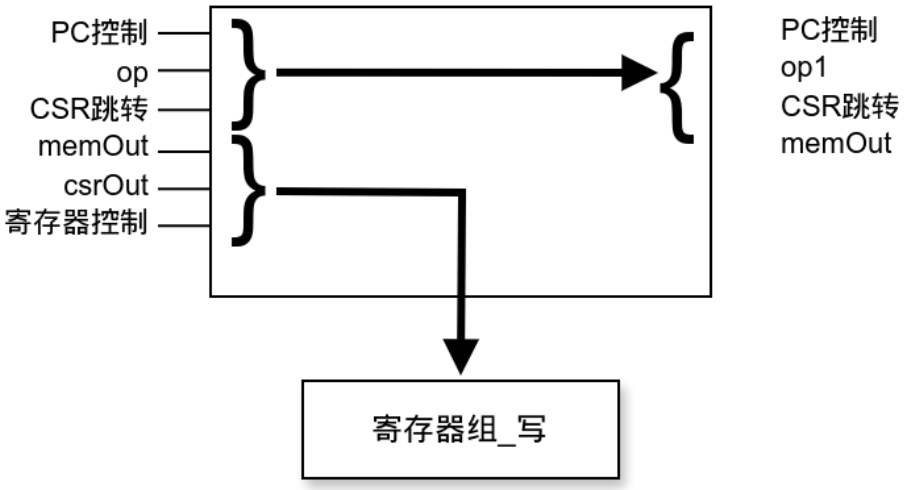

图 3.11 写回单元总体架构

3.7 仿照吕 加入 Chisel 段落

要强调 Chisel 具体好在哪里。给功能点举出例子

验证

4.1 验证平台

4.2 验证环境

介绍整体，引出 NPC，NEMU，Spike。使用 Difftest 将他们串在一起

4.3 Difftest

4.4 NPC

这里对 verilator 进行说明

sdb component

4.5 NEMU Spike

4.6 仿真分析

4.6.1 基础测试

如果这里想写长一点，可以参考吕的文章，按照流水段来写

4.6.2 特权指令集测试

4.6.3 启动 RT-thread

NEXT

参考文献

[1] 徐艳茹, 刘继安, 解壁伟, 等. 科教融合培养关键核心技术人才的理路与机制——OOICCI 芯片人才培养方案解析[J]. 高等工程教育研究, 2023(1): 20–26, 43.

[2] 雨前顾问, 安谋科技. 2023 年中国大陆集成电路产业人才供需报告[R/OL]. http://www.by-consulting.cn/home/projects/detail/id/10.

[3] 包云岗, 孙凝晖, 张科. 处理器芯片开源设计与敏捷开发方法思考与实践[J]. 中国计算机学会通讯, 2019, 15(10): 42-48.

[4] HENNESSY J L, PATTERSON D A. A New Golden Age for Computer Architecture[J/OL]. Communications of the ACM, 2019, 62(2): 48-60[2024-03-07]. https://dl.acm.org/doi/10.1145/3282307. DOI:10.1145/3282307.

[5] 微五科技. “芯征程”, 芯辉煌！“港华芯 ”销量突破 100 万片[EB/OL]. (2023)[2024-03-11]. http://www.chinafive.com.cn/site/news_details/143.

[6] ASANOVIĆ K, AVIZIENIS R, BACHRACH J, 等. The Rocket Chip Generator[R/OL]. (2016-04). http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html.

[7] 石亚琼. 睿思芯科推出基于 RISC-V 的 64 位可编程终端 AI 芯片 Pygmy[EB/OL]. (2018)[2024-03-11]. https://www.36kr.com/p/1722975698945.

[8] PATTERSON D A, DITZEL D R. The case for the reduced instruction set computer[J]. ACM SIGARCH Computer Architecture News, 1980, 8(6): 25-33.

[9] WATERMAN A S. Design of the RISC-V instruction set architecture[M]. University of California, Berkeley, 2016.

[10] FOWLER M, HIGHSMITH J, OTHERS. The agile manifesto[J]. Software development, 2001, 9(8): 28-35.

[11] LEE Y, WATERMAN A, COOK H, 等. An Agile Approach to Building RISC-V Microprocessors[J/OL]. IEEE Micro, 2016, 36(2): 8-20. DOI:10.1109/MM.2016.11.

[12] 包云岗, 常轶松, 韩银和, 等. 处理器芯片敏捷设计方法：问题与挑战[J]. 计算机研究与发展, 2021, 58(6): 1131-1145.

[13] FERDJALLAH M. Introduction to digital systems: modeling, synthesis, and simulation using VHDL[M]. John Wiley & Sons, 2011.

[14] BACHRACH J, VO H, RICHARDS B, 等. Chisel: Constructing Hardware in a Scala Embedded Language[C/OL]//Proceedings of the 49th Annual Design Automation Conference. Association for Computing Machinery, 2012: 1216-1225. DOI: 10.1145/2228360.2228584.

[15] ODERSKY M, ALTHERR P, CREMET V, 等. An overview of the Scala programming language[J/OL]. 2004. http://infoscience.epfl.ch/record/52656.

[16] LI P S, IZRAELEVITZ A M, BACHRACH J. Specification for the FIRRTL Language[R/OL]. (2016-02)[2024-03-01]. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-9.html.

[17] ELDRIDGE S, BARUA P, CHAPYZHENKA A, 等. MLIR as hardware compiler infrastructure[C]//Workshop on Open-Source EDA Technology (WOSET). 2021.

[18] LATTNER C, AMINI M, BONDHUGULA U, 等. MLIR: Scaling compiler infrastructure for domain specific computation[C]//2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). 2021: 2-14.

[19] IZRAELEVITZ A, KOENIG J, LI P, 等. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations[C/OL]//2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD): 卷 0. 2017: 209-216. DOI:10.1109/ICCAD.2017.8203780.

[20] What SystemVerilog features should be avoided in synthesis?[EB/OL]. https://stackoverflow.com/questions/20312810/what-systemverilog-features-should-be-avoided-in-synthesis.

[21] 余子濠, 刘志刚, 李一苇, 等. 芯片敏捷开发实践:标签化 RISC-V[J]. 计算机研究与发展, 2019, 56(1): 35-48.

[22] 吕治宽. 基于 verilog 和 chisel 的乱序超标量 RISC-V 处理器设计比较[D]. 2022.

[23] ARM LIMITED. AMBA AXI and ACE Protocol Specification AXI3, AXI4, AXI5, ACE and ACE5[EB/OL]. (2023). https://www.arm.com/architecture/system-architectures/amba/amba-specifications.

附录

A.1 附录子标题

A.1.1 附录子子标题

附录内容，这里也可以加入图片，例如图 A.1。

图 A.1 图片测试

致　　谢

感谢 NJU-LUG，提供 NJUThesis Typst 模板。