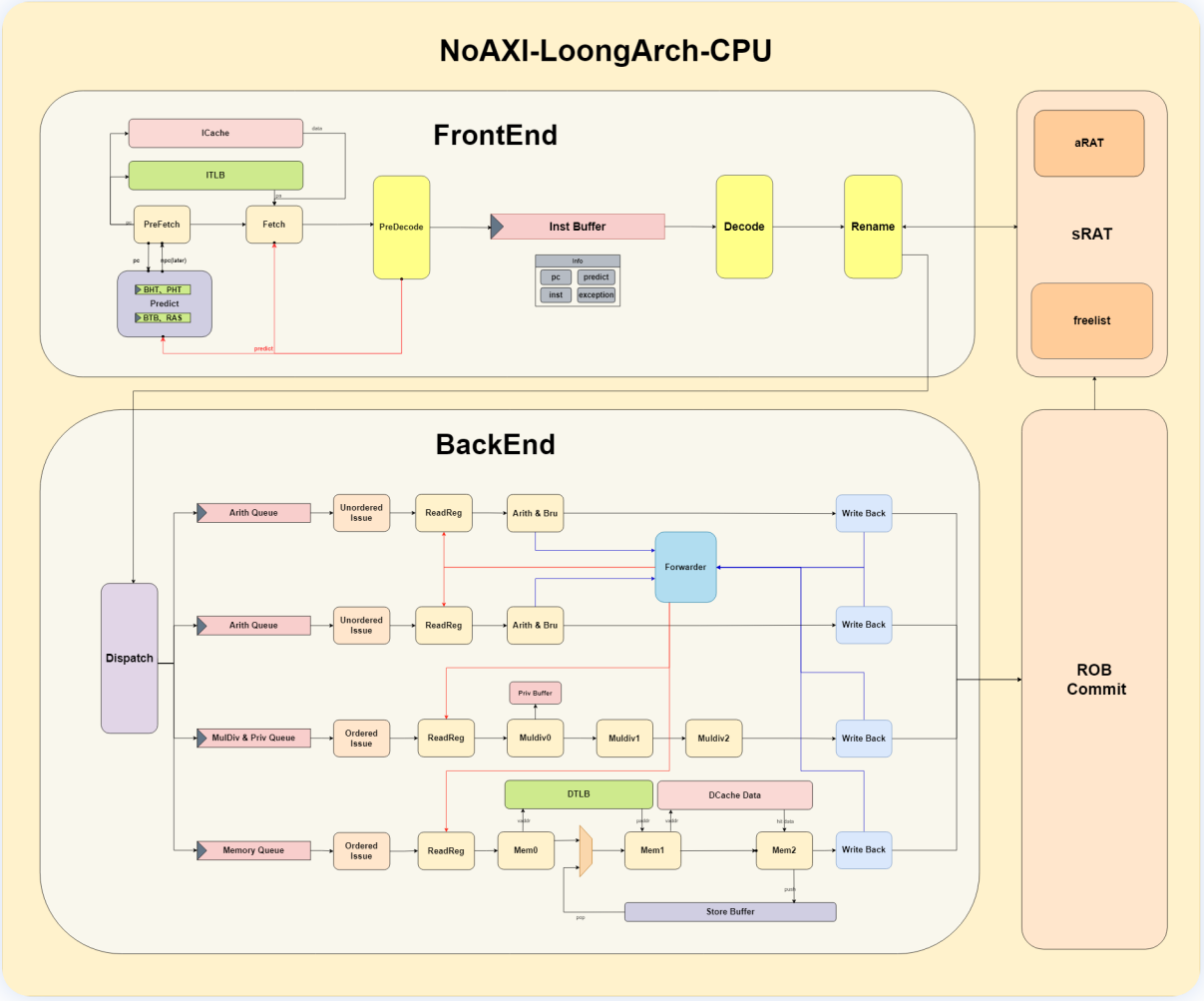
NSCSCC2024 初赛设计报告

杭州电子科技大学 NoAXI队

陈润澎 刘文涛



目录

1. 概述

1.1 项目简介

本项目是第八届"龙芯杯"全国大学生计算机系统能力培养大赛（NSCSCC 2024）的参赛作品。

本项目基于龙芯架构32位精简版LA32R，开发了一款乱序多发射的CPU。项目基于Tomasulo动态调度算法，使用ROB重排序缓存，实现了一个**乱序执行**、**后端四发射**的CPU。此外，我们还实现了分支预测、Cache缓存等诸多特性，使得CPU取得了较好的性能表现。

1.2 实现功能

支持指令集：龙芯架构32位精简版中的以下指令

- 算数类指令：`ADD.W`，`SUB.W`，`ADDI.W`，`LU12I.W`，`SLT[U]`，`SLT[U]I`，`PCADDU12I`，`AND`，`OR`，`NOR`，`XOR`，`ANDI`，`ORI`，`XORI`，`MUL.W`，`MULH.W[U]`，`DIV.W[U]`，`MOD.W[U]`，`SLL.W`，`SRL.W`，`SRA.W`，`SLLI.W`，`SRLI.W`，`SRAI.W`
- 跳转指令：`BEQ`，`BNE`，`BLT[U]`，`BGE[U]`，`B`，`BL`，`JIRL`
- 访存指令：`LD.B`，`LD.H`，`LD.W`，`LD.BU`，`LD.HU`，`ST.B`，`ST.H`，`ST.W`
- CSR相关指令：`CSRRD`，`CSRWR`，`CSRXCHG`
- Cache相关指令：`CACOP` (暂未实现)
- TLB相关指令：`TLBSRCH`，`TLBRD`，`TLBWR`，`TLBFILL`，`INVTLB`
- 其他杂项指令：`RDCNTVL.W`，`RDCNTVH.W`，`RDCNTID`，`SYSCALL`，`BREAK`，`ERTN`

2. 处理器微架构设计

2.1 整体介绍

NoAXI处理器采取前后端设计，前端共5个流水级，后端共6个流水级，共11个流水级，最深级数为13。

前端五个流水级，分别为 **预取指**、**取指**、**预译码**、**译码**、**重命名**。

后端分为六个流水级，分别为 **分发**、**发射**、**读寄存器**、**执行**、**写回**、**提交**。

前端采取顺序双发射设计，取指宽度为2，每个周期可以向后端最多传输2条指令。

后端采取乱序四发射设计，流水线条数为4，每个周期可以进行最多两条指令的提交。

2.2 前端设计

前端分为 **预取指**（PreFetch）、**取指**（Fetch）、**预译码**（PreDecode）、**译码**（Decode）、**重命名**（Rename） 共五级。

其中，预译码级与译码级之间，使用了一个指令缓冲（Inst Buffer）进行解耦。

2.2.1 预取指 PreFetch

1. v_pc发送至tlb，tlb进行翻译
   - 若是直接地址翻译或直接地址映射，寄存器锁存；(目前实现：p_pc = v_pc)
   - 若是映射地址翻译，给tlb表项发地址
2. v_pc_index发送至iCache，给iCache索引line
3. 进行分支预测
   - 分支预测包含btb, bht, pht
   - asid、pc送入btb判断是否命中，送入bht+pht判断是否预测进行跳转
   - btb使用异步sram维护，直接映射
4. 下一拍，pc置为pc_next
   - 默认为pc_next_line
   - 发生跳转预测为btb_target
   - 发生冲刷为bru_target

2.2.2 取指 Fetch

1. tlb返回paddr
2. paddr发送至iCache，判定是否命中
3. 取出n条指令，最早的一条跳转及以后的指令舍弃

2.2.2.1 分支预测器 BPU

在乱序处理器当中，分支预测失败带来的冲刷会极大的影响性能。因此，我们在PreFetch级和Fetch级使用分支预测器进行了分支预测。

1. 根据pc判断是否为跳转指令
   - 利用 `BHT` 表，添加 `valid` 项，表示pc对应的是否为跳转指令，在更新 `BHT` 表的时候更新 `valid`
   - 读取 `BHT` 表，看该pc是否为跳转. 利用 `BHT(Branch History Table)` 记录pc对应的跳转历史 `BHR(Branch History Register)`
   - 更新时机： `commit` 时

- TODO:可以使用哈希(如异或)映射pc

2. 利用 `PHT(Pattern History Table)` 记录 `BHR` 对应的跳转历史
   - 暂定所有分支指令共用一个 `PHT`
3. *基于全局历史 `GHR(Global History Register)` 的分支预测
   - 可以理解成 `BHT` 中只有一个表项，此时 `BHT` 表就变成了 `GHR` ，这一项记录所有跳转指令的历史
4. *竞争的分支预测
   - 对于一个分支指令，记录其使用 `BHR` 和使用 `GHR` 的成功失败信息，如果使用某一个历史失败两次，则转用另一个历史
5. 对于直接跳转类型和 `CALL` ，利用 `BTB(Branch Target Buffer)` 记录pc对应的跳转地址，同时 `BTB` 兼顾判断跳转类型(CALL,Return,其他)的职责
   - 对 `pc` 进行哈希运算
   - 如果缺失，顺序取指
   - 认为一个 `pc` 哈希映射后的结果仅存在一个跳转，若存在多个则会降低预测成功率(因此使用直接相连)

Valid	Tag	BTA	Br_type
1 b	1 b	1 b	2 b

6. 对于间接跳转类型(`Return`)，利用 `RAS(Return Address Stack)` 记录最近执行的 `CALL` 指令的下一条指令地址，并使用栈顶值作为预测跳转地址
   - `RAS` 满时，对 `RAS` 使用循环更新，即再从底向上开始更新
   - *TODO:对 `RAS` 的每一项添加一个计数属性，表示该地址出现的次数

2.2.3 预译码 PreDecode

分支预测btb的添加放在这一级

此时需要进一步进行分支预测和流水线冲刷，并对prefetch的btb进行更新。具体而言，分三种情况：

1. 未被记录的无条件跳转指令，但流水线顺序取指，所以一定为预测失败
2. 未被记录的条件跳转指令，暂且认定为需要跳转，则预测失败
3. 被标记为跳转成功的指令，但在此发现不是跳转，则预测失败

当检测到预测失败信号时，清空前两级流水线

同时，这一级也用于标记中断。

2.2.3.1 指令缓冲 Instruction Buffer

在预译码级与译码级之间，我们使用FIFO，实现了一个指令缓冲，用于解耦取指级与译码级。

当后端因访存等因素而导致流水线阻塞时，指令缓冲的存在能使得前端继续进行取指，从而尽量减少前端取指未命中导致的性能损耗。

需要注意的是，指令缓冲不会保存气泡信息，在入队 / 出队的时候，都会对于气泡信息进行屏蔽。

2.1.4 译码 Decode

本级用于译码，我们通过chisel语言的特性，生成了两个树形译码元件进行译码，并输出指令对应的微码。其中包含了指令所用的相关功能模块类型 `funcType` ，指令在模块内对应的操作符 `opType` ，指令对应的 `rj` `rk` `rd` 寄存器编号，以及指令对应的后端流水线编号等信息。

2.2.5 重命名 Rename

本级流水线用于进行寄存器重命名。本级与寄存器映射表RAT与后端的重排序缓存ROB连接，向二者发出请求，接收RAT返回的寄存器信息，以及ROB返回的对应编号。具体操作如下。

1. RAT分配物理寄存器时，从空闲寄存器表freelist当中取最多两个空闲寄存器，作为原逻辑寄存器areg对应的物理寄存器preg。
2. ROB通过其fifo指针分配两条指令对应的ROB编号。

至此指令开始占用寄存器映射表信息、ROB表项、物理寄存器资源。

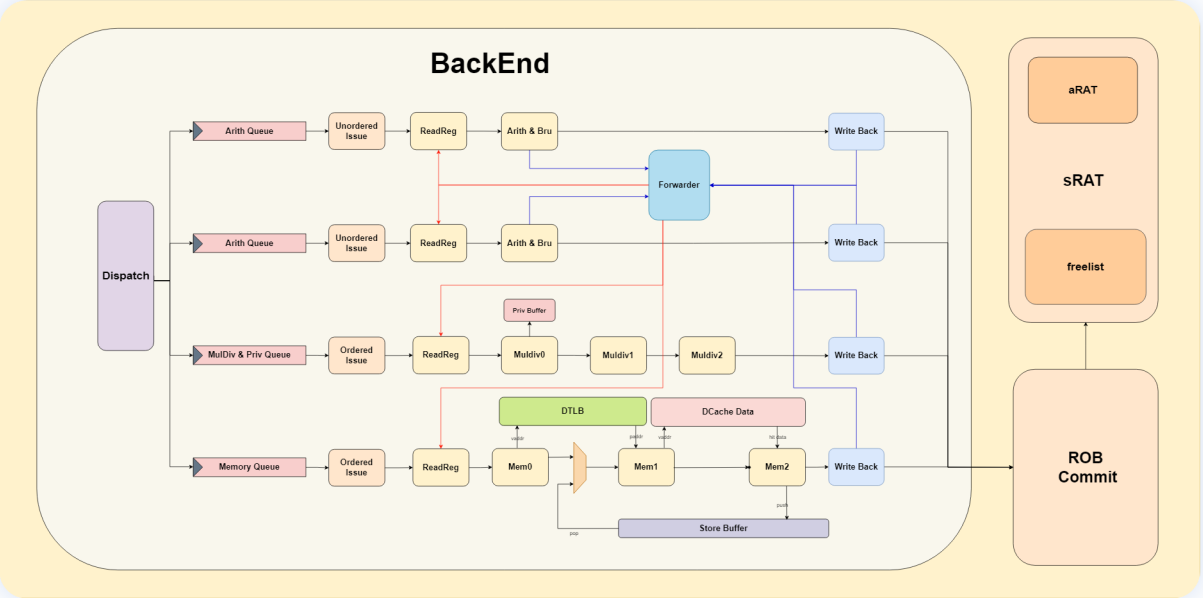
同时，这一级会向发射级发送占用寄存器的请求，以此阻塞发射队列当中的相关指令。

2.3 后端设计

后端分为六个流水级，分别为**分发**（Dispatch）、**发射**（Issue）、**读寄存器**（ReadReg）、**执行**（Execute）、**写回**（WriteBack）、**提交**（Commit）。其中，执行级又根据流水线功能的不同，细分为1~3个流水级。算术流水线只有一级执行级，而乘除和访存有三级执行级。

从发射级开始，后端分为四条独立的流水线，分别是2条**算术**执行流水线，1条**乘除**法执行流水线，1条**访存**执行流水线。

需要注意的是，特权指令放在乘除法。为了保证csr写指令的正确性，在重排序缓存提交指令的时候，才写csr寄存器并冲刷整条流水线。



2.3.1 分发 Dispatch

本级会根据指令的类型，分发指令到各个流水线发射缓存当中。需要注意的是，假如指令涉及的流水线重合，或者需要发射的目标流水线已满，则只会发射满足条件的指令，并会在本级进行阻塞，直至全部指令发射完成。

2.3.2 发射 Issue

这里定义了四个发射队列，对于单个发射队列，每个周期最多接收、发射一条指令。

不同流水线的发射情况如下：

1. 运算流水线，我们已经通过寄存器重命名保证了乱序写回的正确性，因此乱序发射即可。
2. 乘除流水线，由于我们将特权指令分配在了本流水线，因此本流水线需要保证顺序发射。
3. 访存流水线，由于数据缓存只有一个，且需要维护访存数据的一致性，所以必须保证顺序发射。

对于乱序发射的队列，我们使用了压缩队列实现。发射时将检查队列内所有指令的寄存器占用状态，若发现相关寄存器均已被唤醒则可以发射。关于寄存器占用状态的维护，我们会在发出唤醒信号的下一拍更新状态，并额外维护从写缓存中回传的uncached load的对应占用状态，防止冲刷导致相关寄存器的占用状态被解除。

对于乱序发射的队列，会按照指令从旧到新的优先级进行发射，这是因为较旧的指令更有可能拥有较多的相关指令，一次发射能够唤醒更多的指令。对于顺序发射的队列，我们每个周期只会对队列头部的指令进行寄存器占用情况的检查，只会发射队列中的最旧指令。

2.3.3 读寄存器 ReadReg

这一级流水线在每条流水线当中都独立存在，负责从物理寄存器当中读出源操作数对应的数值。寄存器堆对于每一条流水线都分配了一个读接口。同时这一级也会接收执行级和写回级前递的信息，用于实现指令的背靠背执行。

如果是算术流水线的读寄存器阶段，还会进行相关指令的唤醒。

2.3.4 算术流水线 Arithmetic

算术流水线共有2条

只有一级，用于执行算术指令运算

2.3.5 乘除流水线 Muldiv

用于乘除指令与特权指令的执行，分为三级。拆分成多个流水级的主要原因是需要实现乘法运算的流水化。

对于乘法运算，我们使用xilinx ip搭载了一个三级流水的乘法器，实现了乘法运算的流水化。

对于除法运算，我们使用xilinx ip实现了除法器，并在乘除流水线的第一级中进行阻塞执行。

对于特权指令，我们会将写指令放入特权缓存PrivBuffer当中，并阻塞后续所有csr指令，直到后端ROB进行提交的时候，才进行csr寄存器的写入。

2.3.6 访存流水线 Memory

访存流水线是后端流水线当中最重要的一条。

2.3.6.1 Mem0

计算vaddr，并发送vaddr至tlb-hit和tag-sram；

2.3.6.2 Mem1

tlb-hit送入tlb-read，tlb-read返回paddr，判断是否产生tlb例外

tag-sram返回tag，判断命中，若未命中进行replace cache line

发送读信号到存储器

2.3.6.3 Mem2

对于load指令，获得sram返回的数据，并从写缓存当中合并读数据。

对于首次进入流水线的store指令，将在本级插入写缓存。

对于写缓存重新进入流水线的指令，将在本级进行读/写操作。

2.3.6.4 写缓存 Store Buffer

在乱序处理器当中，为了保证处于推测态的访存指令不会对外部状态进行更改，我们使用了写缓存Store Buffer来维护。

对于store指令和uncached load指令，会在Mem2级将其插入写缓存Store Buffer内，等待ROB的提交信号。当接收到ROB的提交后，会将写缓存内的信息通过Mem1级重新回流到访存流水线当中。

由于Mem1需要同时接受来自Mem0的数据和来自写缓存的数据，当Mem0向Mem1发送数据时，还会与写缓存发送的数据进行竞争，优先让写缓存内数据进行访存。

2.3.7 写回 WriteBack

这一级用于写物理寄存器，更新rob当中对应的提交信息，并更新发射队列当中的寄存器占用状态。

在我们的初版设计当中，为了时序考虑，直到这一级才会对访存流水线相关的指令进行串行的唤醒。

2.3.8 提交 Commit

这一级用于进行最终的提交，解除对于CPU资源的占用，并解除寄存器的推测态。

1. 解除对于上一个物理寄存器的占用（opreg），并将opreg对应的寄存器编号插入到空闲寄存器列表freelist
2. 更新aRAT的映射关系，aRAT当中存储的是由已经提交的指令构成的、不涉及推测态指令的寄存器映射表
3. 对于包括分支、访存、特权在内的部分非算术指令，由于分支预测器更新限制、写缓存入队限制等原因，我们在检测到这些指令的时候，不会进行双指令的提交。假如检测到分支预测失败或例外，还会向冲刷控制器发出冲刷请求。
4. 当发生流水线清空的时候，首先向冲刷控制器发出请求。冲刷控制器会延迟一拍后，对于各个流水线及功能部件发出冲刷信号，并令重命名相关部件进行状态恢复。具体而言，状态恢复时，会将aRAT赋值给cRAT，清空rob当中的所有表项，并将freelist的尾指针置为头指针位置。