Revision History

Author	Description	Date
刘恒雨	Initialised and updated spec doc	2025/04/10
刘恒雨	Finialised automatic code referencing logic	2025/04/23
胡英瀚，杨钧铎，饶忠禹	提供模块功能框图，以及功能部件阐述	2025/04/23
刘恒雨	完善 spec 文档	2025/04/30

Contents

1 Terminology

PRF Physical Register File

RMT Register Map Table

RMT Architectural Map Table

STQ STore Queue

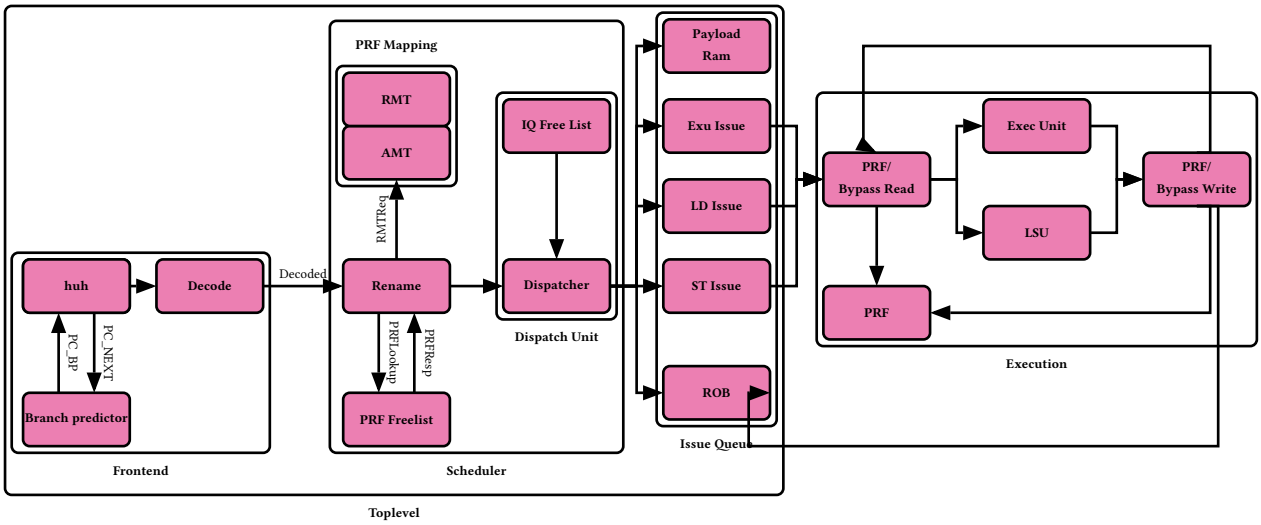
LDQ LoaD Queue

LSU Load Store Unit

EXU Execution Unit, 是所有 FU 的总称

FU Functional Unit, 指 ALU、MUL、DIV 等功能元件

2 Overview



})

3 Microarchitecture

本处理器为乱序执行双发射 RV32IM 架构，代码设计风格部分借鉴于 BOOM。

3.1 功能部件介绍

3.1.1 取指 (IF)

取指单元在单周期内读取 CORE_WIDTH 个指令，随后在检测指令组是否包含跳转指令并与分支预测结果（BP）进行比较，进而确定 PC_Next，同时将 CORE_WIDTH 个指令发送到译码单元。这 CORE_WIDTH 个指令将保持程序原有的执行次序，并且随带着 valid 位表示是否为有效指令。valid 位为主要由 PC 地址低有效位中非对齐部分和分支预测结果决定。

PC_Next 对于存储器来说永远是对齐的（把低有效位置零即可），而 PC_Next 本身不一定是对齐的，因此在取指后，需要将 PC(非对齐)之前的指令作废（vaild = 0)。

PC_Next 主要来自：ROB、分支预测(BP)、$\mathrm{PC}(\text{对齐}) + \mathrm{CORE\_WIDTH} \times 4$，其中优先级由高到低排列。当 ROB 发出转跳信号时（ROB 退休指令提示分支预测错误），PC_Next 取自 ROB，否则按照 BP 命中与否来决定 PC_Next 是 PC_BP 还是 $\mathrm{PC}(\text{对齐}) + \mathrm{CORE\_WIDTH} \times 4$。

此外，取指单元同样也会检测指令组是否包含 CSRRW，

为了简化分支预测逻辑，我们只将 PC_Next 发送给 BP。

参见： FetchUnit

3.1.2 分支预测 (BP)

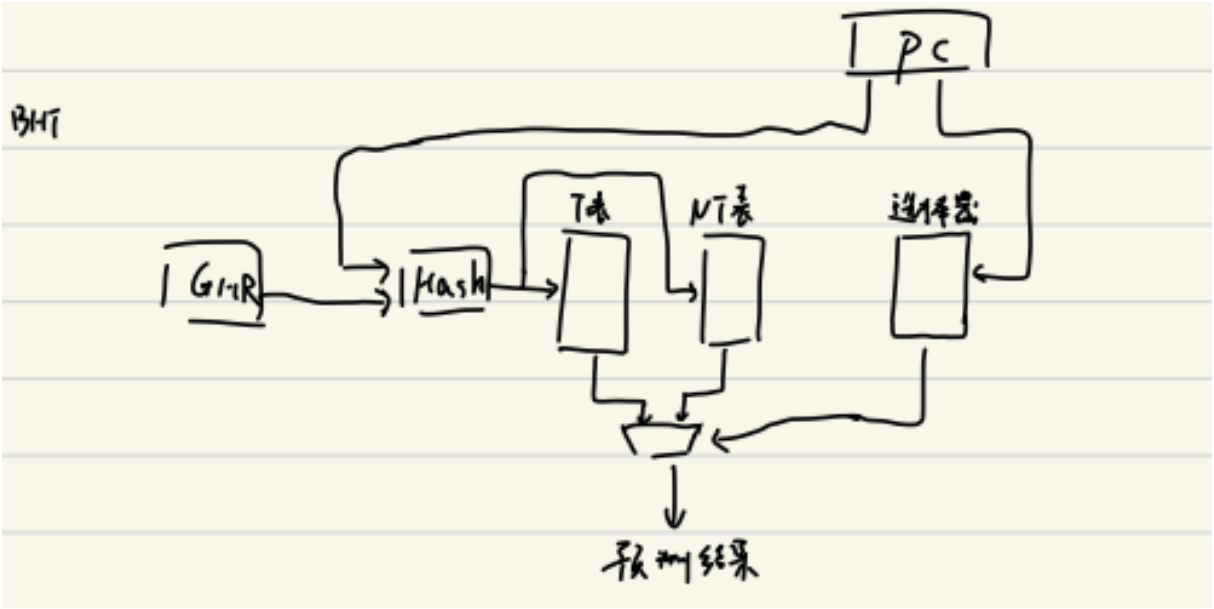


Figure 1: BHT 结构：采用分开的 T/NT 策略，以 PC 部分低位作为选择器索引

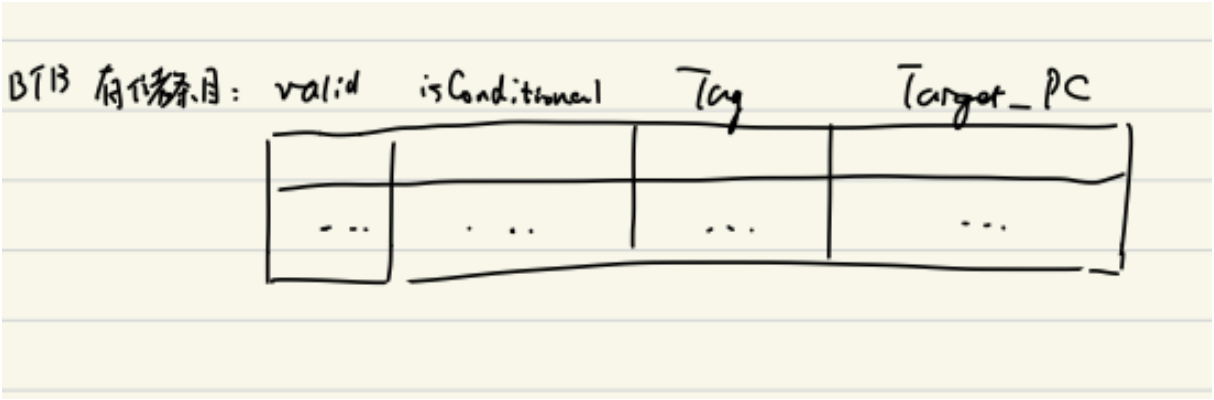


Figure 2: BTB 条目

我们省却了 RAS，并对跳转和分支指令使用共同的 BTB 表，表中以 isConditional 区分两者。若 BTB 命中且 isConditional 为假，则跳转结果为 T（覆盖 BHT 结果）。

分支跳转指令退休时，ROB 会发出信号更新 BTB、BHT 表以及 GHR。

为了避免组合逻辑循环，BHT 预测结果和 BTB 输出必须是寄存器输出。

参见： BranchPredictorUnit

3.1.3 译码 (ID)

从取指单元接受 CORE_WIDTH 个指令，并对有效的指令进行译码操作，最后发送到重命名单元。

译码单元不会造成流水线堵塞，因此只传递后级的阻塞信号。

参见： DecodeUnit

3.1.4 重命名 (RU)

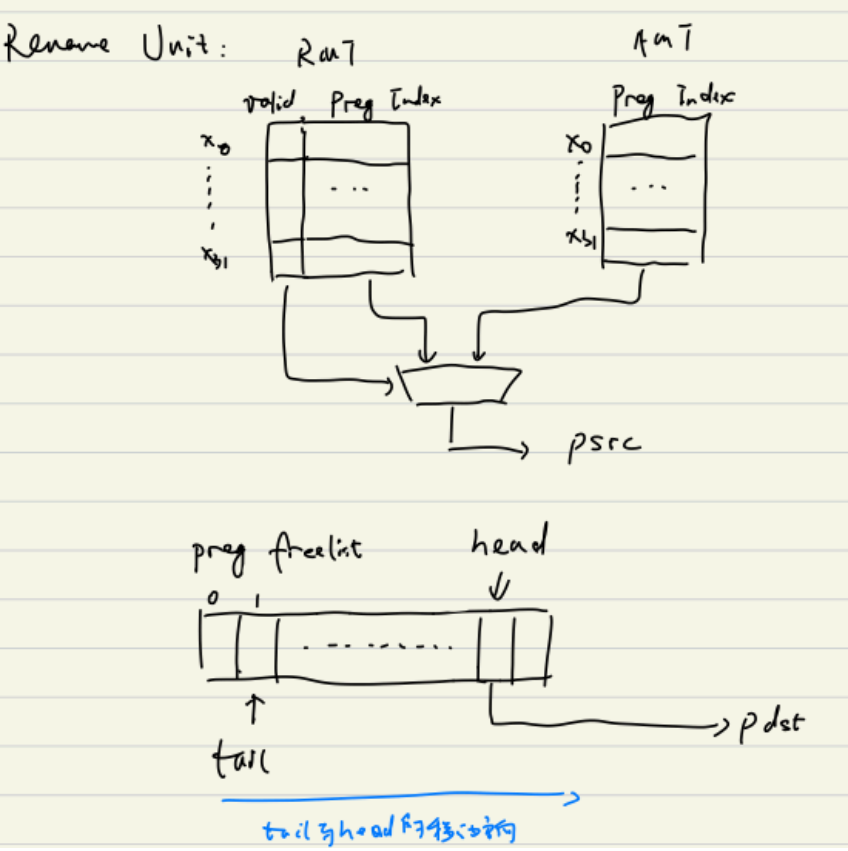


Figure 3: 重命名架构

从译码单元接受 CORE_WIDTH 个指令，执行时必须确保 CORE_WIDTH 个指令能够同时被重命名（不包括无效指令），否则阻塞流水线。

对指令源寄存器重命名时分别从 RMT 和 AMT 获得对应的映射关系。其中 RMT 优先级高于 AMT，亦即当 RMT 中的映射关系被记为有效时，应当选取 RMT，否则 AMT 取胜。RMT 中的有效位只有在回滚时被清零，目的是为了减少冗余的复制操作。

对于目的寄存器，需从 PRF Freelist 获取 CORE_WIDTH 个空闲物理寄存器，获取成功后将 index 写入 RMT，否则 stall。

参见： RenameUnit

3.1.5 派遣 (DP)

从重命名单元接受 CORE_WIDTH 个指令，执行时必须确保 CORE_WIDTH 个指令能够同时被派遣，否则阻塞流水线。

此单元将根据 uop 的类型，将指令分别派遣到 exu_issue, ld_issue, st_issue 等指令发射队列。

参见： DispatchUnit

3.1.6 指令发射 (IQ)

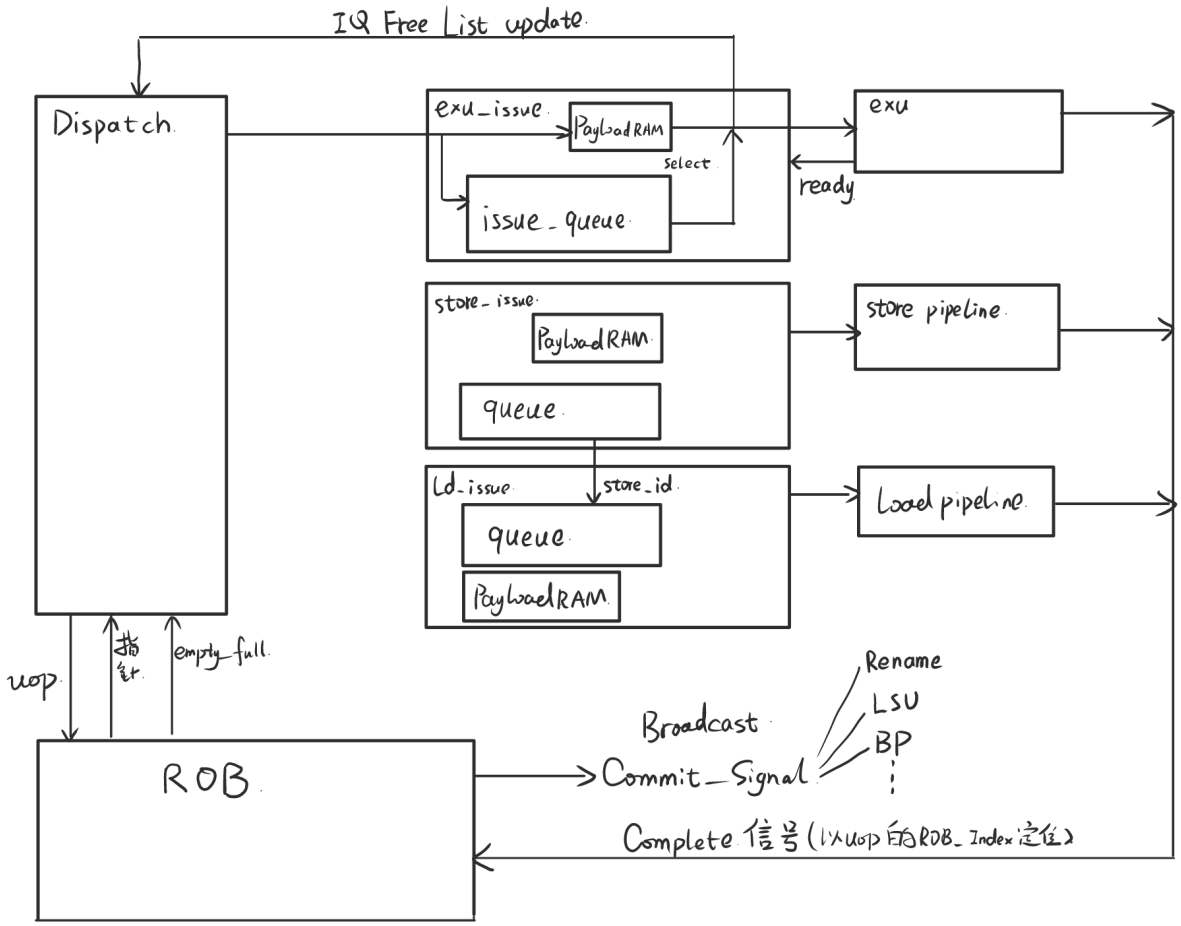


Figure 4: Issue Queue 工作原理图

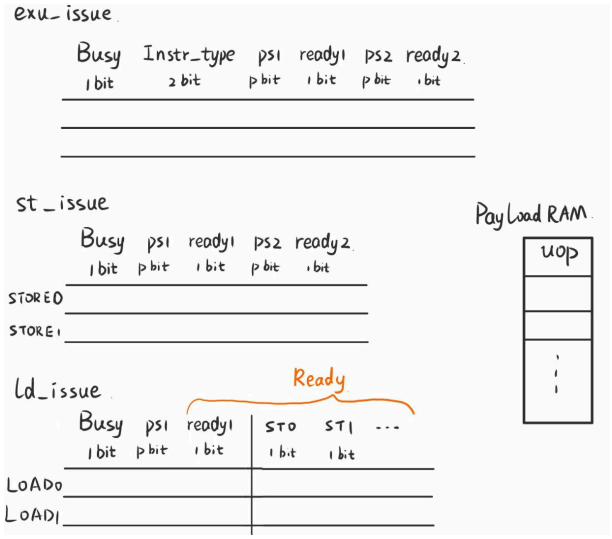


Figure 5: Issue Queue 条目

Issue queue 用于向各个 EXU 发射命令，需要监听后续 EXU 是否空闲以及各操作数的就绪状态。操作数设置为就绪有三种途径：

1. 监听 PRF 中各物理寄存器的 Valid 信号
2. 监听 EXU 处的 ready 信号

3. 监听 EXU 后的级间寄存器的 ready 信号

在本 CPU 中，IQ 分为三部分，分别是 **exu_issue**, **ld_issue**, **st_issue**，其中后面二者间有数据传输，可见后续说明。

exu_issue 的运行过程如下，在非满时接收 dispatch 模块的 uop 存入 payloadRAM 中，将有用信息（instrType、psrc）存入 queue 中用于判断就绪状态，等待发射，每周期根据条目就绪状态至多选择 2 条已就绪的指令发射给 FU（内含两个 ALU、一个 BU、一个乘法器、一个除法器），同时向 dispatch 模块发送本周期发出的指令的 IQ 地址，用以更新 IQ Free List（位于 dispatch 单元）。

st_issue 基本与 **exu_issue** 相同，向 store 流水线（位于 LSU 中）发送 uop，不同的是要向 **ld_issue** 发送每周期仍未发射的条目信息。**ld_issue** 需要从 **st_issue** 接收信息的目的在于解决 load-store 违例。

在本CPU中，store指令走完store流水线的周期固定为2，而load指令在load流水线第二级才会从stq获取前馈数据(见Figure 6)，所以只需要保证在程序顺序上位于该load指令之前的store指令全部进入STU后再发射load，就可以避免load指令越过有数据依赖的store指令，错误地读取数据（store – load violation）。为此，需要一个存储依赖关系的矩阵，load指令在进入**ld_issue**时，就要接收当前周期 **st_issue** 中尚未发射的 store 条目，以二进制数(位宽为 ST_ISSUE_QUEUE_DEPTH)的形式存入矩阵的一行中。每当 **st_issue** 发射一条 store 指令，就要更新 **ld_issue** 矩阵中对应的列，将其全部置为 0 表示就绪。当一条 load 命令所在行全为 0 时，表示与其相关的（比该 load 指令更年轻的）store 指令已经全部发射，该 load 指令解除限制（又若 load 指令的 psrc 为 ready 状态，则该 load 指令具备发射条件）。

参见： exu_issue_queue | ld_issue_queue | st_issue_queue

3.1.7 重排序 (ROB)

Reorder Buffer(ROB)，是一个 FIFO 结构，用于按照程序序存储指令，可以实现顺序 retire，从而实现分支预测失误后的回滚与精确异常（本项目无需实现精确异常）

在实际工作时，ROB 从 dispatch 接收 uop 并储存，把自身头尾指针以及 empty/full 标志传递给 dispatch 单元。从各个 EXU 接收指令完成信号，更新 complete 状态。ROB 总是广播位于头部的两条指令，用额外的 valid 位（两条指令各有一个）来指示本周期是否 retire 该指令，用于 AMT 的更新、通知 STQ 可以写入内存、通知其他模块发生分支预测错误。

参见： ROB

3.1.8 寄存器访问和旁路 (RRDWB)

指令被发射后会先通过寄存器访问和旁路单元(RRDWB)获得操作数。此时，若旁路有数据传回，则在寄存器写入地址（pdest）与当前指令 psrc 相同的条件下载入传回数据，否则载入 PRF 数据。功能单元的每一个寄存器读端口均有一个 RRDWB 单元。

参见： BypassNetwork

3.1.9 功能单元 (FU)

FU 为 ALU, BU, MUL, DIV, CSR 的抽象类，主要提供基本的输出入输出端口。指定单元（BU, CSR 等）有额外读写端口专门与 ROB 交互。

3.1.9.1 ALU

主要实现加减法，逻辑和算数位移操作，可流水化实现，不阻塞流水线。

3.1.9.2 BU

内部实现减法器，检测分支跳转预测结果是否正确。通过额外的端口将跳转信息写入 ROB。

3.1.9.3 MULFU

多周期整数乘法流水线

3.1.9.4 DIVFU

多周期整数除法流水线

3.1.9.5 CSR

由于写入 CSR 寄存器时会更改处理器的状态，所以我们按照一般做法：ID 在检测到 CSRRW 指令后，阻塞流水线直到 ROB 队首执行 CSRRW 为止。与 LSU 类似，在 ROB 队首遇到了 CSRRW 才执行指令。

CSR 寄存器通过 Memory mapping 把地址映射到特定的寄存器（可以是外设寄存器，或是 io 端口）。本处理器只实现简单的 mcycle 寄存器，实现方法是额外设立一个计数器，在 csr 读写时，使能读写端口。

参见： FunctionalUnit | ALUFU | BranchFU | MULFU | DIVFU | CSRFU

3.1.9.6 FU 组合

见 Section 3.1.9

3.1.9.7 LSU

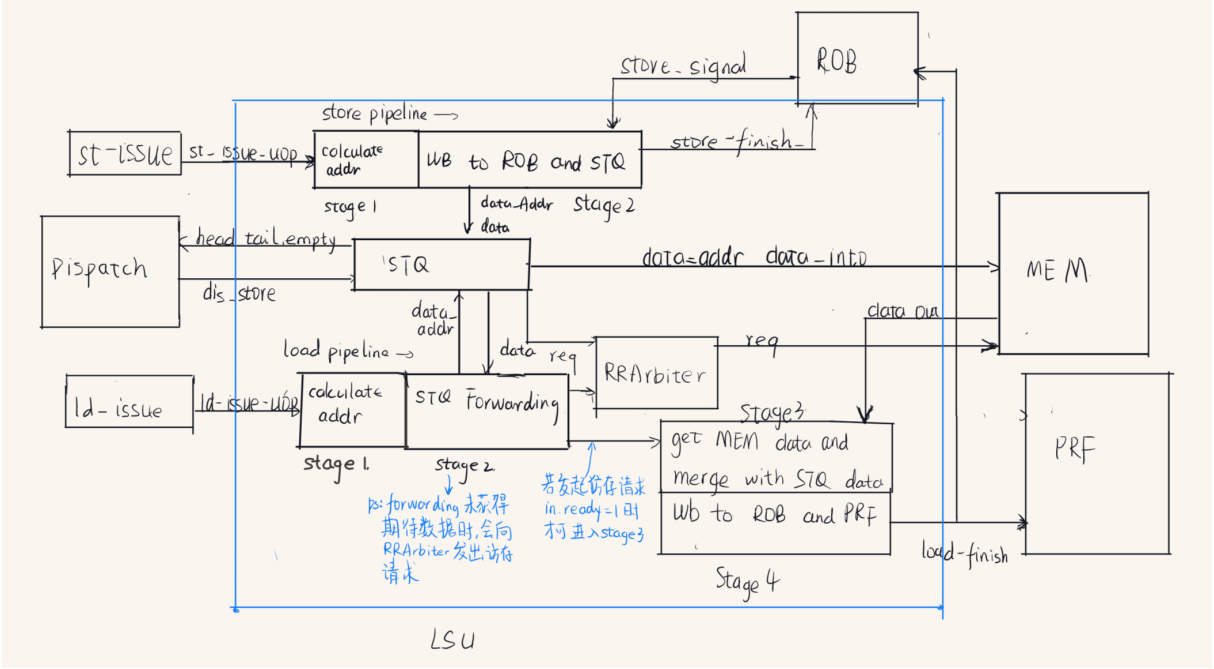


Figure 6: LSU 工作原理图

参见： LSU

3.1.10 物理寄存器堆 (PRF)

物理寄存器堆存有 PRF_DEPTH 个寄存器，寄存器宽度为 XLEN。由于存在多个 FU，PRF 必须有多个读写端口。

参见： PRF

Summary:

功能	实现方法
分支预测	BTB + BHT(T/NT)
重命名表	AMT+RMT, RMT 存有有效位，无效时以 AMT 为准
发射队列	一个执行单元对应一个发射队列
执行单元	两个执行单元，即 FU（ALU, BU, MUL, DIV, CSR）的组合和 LSU
Cache	无

3.2 特别案例处理方式

3.2.1 Load Store 违例 (Load Store Violation)

Load Store 违例是当发射队列中操作地址相同且较年轻的 load 指令比老的 store 指令更优先执行。其中主要原因可能是：

- Load 指令的源寄存器较 Store 指令更早就绪

为简便起见，我们采用分开的 load 和 store 发射 FIFO 队列，同时维护一个依赖矩阵，用以保存 load-store 之间的年龄信息。只有 load 之前的所有 store 执行完毕，load 指令才能执行。该矩阵的每一行对应一个 load 指令，每一列对应 store 指令。每当 load 载入 issue queue 时，对应行将载入 store queue 的快照；store 指令在完成之后，将对应列清零。由此，当且仅当 load 的一行全为零才符合被执行的条件。

3.2.2 分支误判回滚

分支预测在判定为错误后，由 BU 写入 ROB。直到当该分支指令退休时，才对系统进行回滚操作。回滚时，RMT 每一行的 vaild bit 清零，所有 freelist 以及 rob 头部指针与尾部指针重合，同时将跳转地址发送至 IF 并重置所有其它模块。

3.2.3 CSR 写入

见 Section 3.1.9.5

4 Units

4.1 FetchUnit class Module

IO 定义

io `new Fetch_IO()`

4.2 BranchPredictorUnit class Module

IO 定义

io `new BP_IO()`

4.3 DecodeUnit class Module

IO 定义

io `new Decode_IO()`

4.4 RenameUnit class Module

IO 定义

io `new RenameUnit_IO()`

4.5 DispatchUnit class Module

IO 定义

```
io new Dispatcher_IO
```

4.6 exu_issue_queue class Module

IO 定义

```
io new exu_issue_IO()
```

4.7 ld_issue_queue class Module

IO 定义

```
io new ld_issue_IO()
```

4.8 st_issue_queue class Module

IO 定义

```
io new st_issue_IO()
```

4.9 ROB class Module

IO 定义

```
io new ROBIO()
```

4.10 BypassNetwork class Module

BypassNetwork 的构造参数如下：

Name	Type	Description
bypassCount	Int	旁路输入宽度

IO 定义

io `new BypassNetworkIO`

bypass_signals `Input(Vec(bypassCount, Valid(new BypassInfo)))`

4.11 FunctionalUnit abstract class Module

功能单元的抽象类，定义了底层模块端口 FunctionalUnit 的构造参数如下：

Name	Type	Description
needInformBranch	`Boolean = false`	通知前端信息，比如 BU 需要提供转调信息
needROBSignals	`Boolean = false`	需要从 ROB 获得信息

IO 定义

req `new FUReq()`

out `new DecoupledIO(new ExuDataOut())`

rob_signal `if (needROBSignals)`

仅有 FU 为 CSR 时才有用

branch_info `if (needInformBranch)`

仅有 FU 为 BU 时才有用

4.12 ALUFU class FunctionalUnit

未完成的 ALU FU 实例

4.13 BranchFU class FunctionalUnit

4.14 MULFU class FunctionalUnit

4.15 DIVFU class FunctionalUnit

4.16 CSRFU class FunctionalUnit

4.17 LSU class Module

LSU 的模块定义，目前只完成了 IO 接口的定义，内部逻辑还未完成

IO 定义

io `new LSUIO()`

4.18 PRF class Module

PRF 的构造参数如下：

Name	Type	Description
dType	T	
regDepth	Int	
numReadPorts	Int	
numWritePorts	Int	

IO 定义

read_requests
```
Vec(numReadPorts, Flipped(Decoupled(
    UInt(log2Ceil(regDepth).W)
)))
```
读请求端口（Decoupled 接口实现流控）

read_data `Vec(numReadPorts, Output(dType))`

读响应数据输出（对齐流水线级延迟）

write_ports
```
Vec(numWritePorts, Flipped(Valid(new Bundle {
    val addr = Output(UInt(log2Ceil(numRegisters).W))
    val data = Output(dType)
})))
```
写端口数组（Valid 接口保证写时序）

addr `Output(UInt(log2Ceil(numRegisters).W))`

最大物理寄存器地址位宽

data `Output(dType)`

写入数据

5 Parameters

6 Interface

6.1 Fetch_IO class Bundle

成员定义

instr_addr `Output(UInt(p.XLEN.W))`

当前 IFU 的 PC 值

instr `Input(Vec(p.CORE_WIDTH, UInt(32.W)))`

id_uop `Vec(p.CORE_WIDTH, Valid(new IF_ID_uop()))`

id_ready `Input(Bool())`

ID 是否准备好接收指令

rob_commitsignal `Vec(p.CORE_WIDTH, Flipped(Valid(new ROBContent())))`

ROB 提交时的广播信号，发生误预测时对本模块进行冲刷

target_PC `Input(UInt(p.XLEN.W))`

预测的下个 cycle 取指的目标地址

btb_hit `Input(Vec(p.CORE_WIDTH, Bool()))`

1 代表 hit，0 相反；将最年轻的命中 BTB 的置为 1，其余为 0

branch_pred `Input(Bool())`

branch 指令的 BHT 的预测结果；1 代表跳转，0 相反

GHR `Input(UInt(p.GHR_WIDTH.W))`

作出预测时的全局历史寄存器快照

6.2 BP_IO class Bundle

成员定义

instr_addr `Input(UInt(p.XLEN.W))`

当前 PC 值，用于访问 BTB 获得跳转目标地址，以及访问 BHT 获得跳转预测结果

target_PC `Output(UInt(p.XLEN.W))`

预测的下个 cycle 取指的目标地址

btb_hit `Output(Vec(p.CORE_WIDTH, Bool()))`

1 代表 hit，0 相反；将指令包中命中 BTB 的最年轻的置为 1，其余为 0

branch_pred `Output(Bool())`

条件分支指令的 BHT 的预测结果；1 代表跳转，0 相反；非条件分支置 1

GHR `Output(UInt(p.GHR_WIDTH.W))`

作出预测时的全局历史寄存器快照，随流水级传递，在 ROB 退休分支指令时更新 BHT

rob_commitsignal `Vec(p.CORE_WIDTH, Flipped(Valid(new ROBContent())))`

ROB 提交时的广播信号，从中识别出分支指令更新 BHT 和 BTB

6.3 Decode_IO class Bundle

成员定义

if_uop `Vec(p.CORE_WIDTH, Flipped(Valid(new IF_ID_uop())))`

id_ready `Output(Bool())`

ID 是否准备好接收指令

rename_uop `Vec(p.CORE_WIDTH, Valid(new ID_RENAME_uop()))`

rename_ready `Input(Bool())`

Rename 是否准备好接收指令

rob_commitsignal `Vec(p.CORE_WIDTH, Flipped(Valid(new ROBContent())))`

ROB 提交时的广播信号，发生误预测时对本模块进行冲刷

6.4 RenameUnit_IO class Bundle

重命名单元将逻辑寄存器地址映射成实际寄存器。逻辑寄存器指的是 ISA 定义的 x0-x31，而实际寄存器数量多于 32 个，一般可达 128 个。主要解决 WAW，WAR 等问题。

成员定义

id_uop `Vec(p.CORE_WIDTH, Flipped(Valid(new ID_RENAME_uop())))`

来自 ID 单元的 uop

rename_ready `Output(Bool())`

反馈给 IDU，显示 Rename 单元是否准备好接收指令

rob_commitsignal `Vec(p.CORE_WIDTH, Flipped(Valid(new ROBContent())))`

ROB 提交时的广播信号，rob 正常提交指令时更新 amt 与 rmt，发生误预测时对本模块进行恢复

dis_uop `Vec(p.CORE_WIDTH, Valid(new RENAME_dis_uop()))`

发往 Dispatch 单元的 uop

dis_ready `Input(Bool())`

来自 Dispatch 单元的反馈，显示 dispatch 单元是否准备好接收指令

6.5 Dispatcher_IO class Bundle

成员定义

rename_uop `Vec(p.CORE_WIDTH, Flipped(Valid(new RENAME_DISPATCH_uop())))`

来自 rename 单元的 uop

dis_ready `Output(Bool())`

反馈给 rename 单元，显示 Dispatch 单元是否准备好接收指令

rob_uop `Vec(p.CORE_WIDTH, Valid(new DISPATCH_ROB_uop()))`

发往 ROB 的 uop

rob_full `Input(Bool())`

ROB 空标志(0 表示非满，1 表示满)

rob_head `Input(UInt(log2Ceil(p.ROB_DEPTH).W))`

ROB 头部指针

rob_tail `Input(UInt(log2Ceil(p.ROB_DEPTH).W))`

ROB 尾部指针，指向入队处

rob_commitsignal `Vec(p.CORE_WIDTH, Flipped(Valid(new ROBContent())))`

ROB 提交时的广播信号，发生误预测时对本模块进行冲刷

exu_issue_uop `Vec(p.CORE_WIDTH, Valid(new DISPATCH_EXUISSUE_uop()))`

发往 EXU 的 uop

exu_issued_id `Vec(p.CORE_WIDTH, Flipped(Valid(UInt(log2Ceil(p.EXUISSUE_DEPTH).W))))`

本周期 EXU ISSUE queue 发出指令对应的 issue queue ID，用于更新 iq freelist

st_issue_uop `Vec(p.CORE_WIDTH, Valid(new DISPATCH_STISSUE_uop()))`

发往 Store Issue 的 uop

st_issued_index `Valid(UInt(log2Ceil(p.STISSUE_DEPTH).W))`

本周期 Store Issue queue 发出指令对应的 issue queue ID，用于更新 issue queue freelist

ld_issue_uop `Vec(p.CORE_WIDTH, Valid(new DISPATCH_LDISSUE_uop()))`

发往 Load Issue 的 uop

ld_issued_index `Valid(UInt(log2Ceil(p.LDISSUE_DEPTH).W))`

本周期 Load Issue queue 发出指令对应的 issue queue ID，用于更新 issue queue freelist

stq_full `Input(Bool())`

Store Queue 空标志(0 表示非满，1 表示满)

stq_head `Input(UInt(log2Ceil(p.STQ_DEPTH).W))`

store queue 头部指针

stq_tail `Input(UInt(log2Ceil(p.STQ_DEPTH).W))`

store queue 尾部指针，指向入队处

st_dis `Output(Vec(p.CORE_WIDTH, Bool()))`

本 cycle 派遣 store 指令的情况(00 表示没有，01 表示派遣一条，11 表示派遣两条)，用于更新 store queue（在 lsu 中）的 tail（full 标志位）

6.6 exu_issue_IO class Bundle

成员定义

iq_id `Input(Vec(p.DISPATCH_WIDTH, UInt(log2Ceil(p.IQ_DEPTH).W)))`

IQ ID,

dis_uop `Flipped(Valid(Vec(p.CORE_WIDTH, new DISPATCH_EXUISSUE_uop())))`

来自 Dispatch Unit 的输入

exu_issue_uop `Vec(p.CORE_WIDTH, Valid(new EXUISSUE_EXU_uop()))`

发往 EXU 的 uop

mul_ready `Input(Vec(p.MUL_NUM, Bool()))`

乘法器的 ready 信号

div_ready `Input(Vec(p.DIV_NUM, Bool()))`

除法器的 ready 信号

dst_FU `Output(Vec(p.CORE_WIDTH, UInt(log2Ceil(p.ALU_NUM).W)))`

发射的指令的目标功能单元

issue_uop `Valid(Vec(p.CORE_WIDTH, new EXUISSUE_EXU_uop()))`

发射的指令(包含操作数的值)

value_o1 `Output(Vec(p.CORE_WIDTH, UInt(p.XLEN.W)))`

发射的指令的操作数 1

value_o2 `Output(Vec(p.CORE_WIDTH, UInt(p.XLEN.W)))`

发射的指令的操作数 2

exu_issue_raddr1 `Output(Vec(p.CORE_WIDTH, UInt(log2Ceil(p.PRF_DEPTH).W)))`

PRF 读地址 1

exu_issue_raddr2 `Output(Vec(p.CORE_WIDTH, UInt(log2Ceil(p.PRF_DEPTH).W)))`

PRF 读地址 2

exu_issue_value1 `Input(Vec(p.CORE_WIDTH, UInt(p.XLEN.W)))`

操作数 1

exu_issue_value2 `Input(Vec(p.CORE_WIDTH, UInt(p.XLEN.W)))`

操作数 2

prf_valid `Input(Vec(p.PRF_DEPTH, Bool()))`

PRF 的 valid 信号

wb_uop2 `Flipped(Valid((Vec(p.FU_NUM - p.BU_NUM - p.STU_NUM, new ALU_WB_uop()))))`

来自 alu、mul、div、load pipeline 的 uop

wb_uop1 `Flipped(Valid((Vec(p.FU_NUM - p.BU_NUM - p.STU_NUM, new ALU_WB_uop()))))`

来自 alu、mul、div、load pipeline 的 uop

ldu_wb_uop1 `Flipped(Valid(new LDPIPE_WB_uop()))`

来自 ldu 的 uop

exu_issued_index `Output(Vec(p.CORE_WIDTH, UInt(log2Ceil(p.EXUISSUE_DEPTH).W)))`

更新 IQ Freelist

rob_commitsignal `Vec(p.CORE_WIDTH, Flipped(Valid(new ROBContent())))`

ROB 提交时的广播信号，发生误预测时对本模块进行冲刷

6.7 ld_issue_IO class Bundle

成员定义

dis_uop `Vec(p.CORE_WIDTH, Flipped(Valid(new DISPATCH_LDISSUE_uop())))`

来自 Dispatch Unit 的输入

ld_issue_uop `Decoupled(new LDISSUE_LDPIPE_uop())`

发射的指令

value_o1 `Output(UInt(p.XLEN.W))`

发射的指令的操作数 1

value_o2 `Output(UInt(p.XLEN.W))`

发射的指令的操作数 2

ld_issue_raddr1 `Output(UInt(log2Ceil(p.PRF_DEPTH).W))`

PRF 读地址 1

raddr2 `Output(UInt(log2Ceil(p.PRF_DEPTH).W))`

PRF 读地址 2

ld_issue_value1 `Input(UInt(p.XLEN.W))`

操作数 1

value_i2 `Input(UInt(p.XLEN.W))`

操作数 2

prf_valid `Input(Vec(p.PRF_DEPTH, Bool()))`

PRF 的 valid 信号

wb_uop2 `Flipped(Valid((Vec(p.FU_NUM - p.BU_NUM - p.STU_NUM, new ALU_WB_uop()))))`

来自 alu、mul、div、load pipeline 的 uop

ldu_wb_uop2 `Flipped(Valid(new LDPIPE_WB_uop()))`

来自 ldu 的 uop

wb_uop1 `Flipped(Valid((Vec(p.FU_NUM - p.BU_NUM - p.STU_NUM, new ALU_WB_uop()))))`

来自 alu、mul、div、load pipeline 的 uop

ldu_wb_uop1 `Flipped(Valid(new LDPIPE_WB_uop()))`

来自 ldu 的 uop

ld_issued_index `Output(UInt(log2Ceil(p.LDISSUE_DEPTH).W))`

更新 IQ Freelist

rob_commitsignal `Vec(p.CORE_WIDTH, Flipped(Valid(new ROBContent())))`

ROB 提交时的广播信号，发生误预测时对本模块进行冲刷

6.8 st_issue_IO class Bundle

成员定义

iq_id `Input(Vec(p.CORE_WIDTH, UInt(log2Ceil(p.IQ_DEPTH).W)))`

IQ ID

dis_uop `Flipped(Valid(Vec(p.CORE_WIDTH, new DISPATCH_STISSUE_uop())))`

来自 Dispatch Unit 的输入

st_issue_uop `Decoupled(new STISSUE_STPIPE_uop())`

发射的指令

value_o1 `Output(UInt(p.XLEN.W))`

发射的指令的操作数 1

value_o2 `Output(UInt(p.XLEN.W))`

发射的指令的操作数 2

st_issue_raddr1 `Output(UInt(log2Ceil(p.PRF_DEPTH).W))`

PRF 读地址 1

st_issue_raddr2 `Output(UInt(log2Ceil(p.PRF_DEPTH).W))`

PRF 读地址 2

st_issue_value1 `Input(UInt(p.XLEN.W))`

操作数 1

st_issue_value2 `Input(UInt(p.XLEN.W))`

操作数 2

prf_valid `Input(Vec(p.PRF_DEPTH, Bool()))`

PRF 的 valid 信号

wb_uop2 `Flipped(Valid((Vec(p.FU_NUM - p.BU_NUM - p.STU_NUM, new ALU_WB_uop()))))`

来自 alu、mul、div、load pipeline 的 uop

LDU_complete_uop2 `Flipped(Valid(new LDPIPE_WB_uop()))`

来自 ldu 的 uop

wb_uop1 `Flipped(Valid((Vec(p.FU_NUM - p.BU_NUM - p.STU_NUM, new ALU_WB_uop()))))`

来自 alu、mul、div、load pipeline 的 uop

LDU_complete_uop1 `Flipped(Valid(new LDPIPE_WB_uop()))`

来自 ldu 的 uop

st_issued_index `Output(UInt(log2Ceil(p.STISSUE_DEPTH).W))`

更新 IQ Freelist

rob_commitsignal `Vec(p.CORE_WIDTH, Flipped(Valid(new ROBContent())))`

ROB 提交时的广播信号，发生误预测时对本模块进行冲刷

6.9 ROBIO class Bundle

成员定义

dis_uop `Vec(p.CORE_WIDTH, (new DISPATCH_ROB_uop()))`

Dispatch Unit 的 uop,存入条目中

empty_full `Output(Bool())`

ROB 空标志(0 表示非满，1 表示满)

rob_head `Output(UInt(log2Ceil(p.ROB_DEPTH)))`

ROB 头指针

rob_tail `Output(UInt(log2Ceil(p.ROB_DEPTH).W))`

ROB 尾指针

alu_wb_uop `Flipped(Valid(Vec(p.FU_NUM - p.BU_NUM - p.STU_NUM, new ALU_WB_uop())))`

来自 alu、mul、div、load pipeline 的 uop

bu_wb_uop `Flipped(Valid(Vec(p.BU_NUM, new BU_WB_uop())))`

来自 bu 的 uop,更新就绪状态

stu_wb_uop `Flipped(Valid(new STPIPE_WB_uop()))`

来自 stu 的 uop,更新就绪状态

LDU_complete_uop `Flipped(Valid(new LDPIPE_WB_uop()))`

来自 ldu 的 uop,更新就绪状态

mispred `Input(Bool())`

分支误预测信号

if_jump `Input(Bool())`

分支指令跳转信号

rob_commitsignal `Vec(p.CORE_WIDTH, Valid(new ROBContent()))`

广播 ROB 条目

6.10 BypassNetworkIO class Bundle HasUOP

成员定义

preg_rd `Input(UInt(log2Ceil(p.PRF_DEPTH).W))`

data_out `Output(UInt(p.XLEN.W))`

6.11 FUReq class Bundle HasUOP

每个 FunctionalUnit 都能通过 uop 的原指令生成立即数，并且判定操作数的类型

成员定义

kill `Input(Bool())`

Killed upon misprediction/exception

rs1 `Input(UInt(p(XLen).W))`

通过 RRDWB 获得的 rs1 数据

rs2 `Input(UInt(p(XLen).W))`

通过 RRDWB 获得的 rs1 数据

6.12 ExuDataOut class Bundle HasUOP

成员定义

data `UInt(p.XLEN.W)`

6.13 LSUIO class Bundle

成员定义

data_addr `Output(UInt(64.W))`

访存指令的目标地址

data_into_mem `Output(UInt(64.W))`

需要写入储存器的数据

write_en `Output(Bool())`

写使能信号

func3 `Output(UInt(3.W))`

访存指令的 fun3 字段

data_out_mem `Input(UInt(64.W))`

从储存器中读取的数据

st_issue_uop `Flipped(Valid(new STISSUE_STPIPE_uop()))`

存储指令的 uop

ld_issue_uop `Flipped(Decoupled(new LDISSUE_LDPIPE_uop()))`

加载指令的 uop

stq_tail `Output(log2Ceil(p.STQ_Depth).W)`

stq 的尾部索引

stq_head `Output(log2Ceil(p.STQ_Depth).W)`

stq 的头部索引

stq_full `Output(Bool())`

stq 是否为满,1 表示满

st_dis `Input(Vec(p.CORE_WIDTH, Bool()))`

存储指令被派遣的情况(00 表示没有，01 表示派遣一条，11 表示派遣两条)，用于更新 store queue（在 lsu 中）的 tail（full 标志位）

rob_commitsignal `Vec(p.CORE_WIDTH, Flipped(Valid(new ROBContent())))`

ROB 的 CommitSignal 信号

stu_wb_uop `Valid((new STPIPE_WB_uop()))`

存储完成的信号,wb to ROB

ldu_wb_uop `Valid((new ALU_WB_uop()))`

加载完成的信号,wb to ROB and PRF