

# Revision History

Author	Description	Date
刘恒雨	Initialised and updated spec doc	2025/04/10

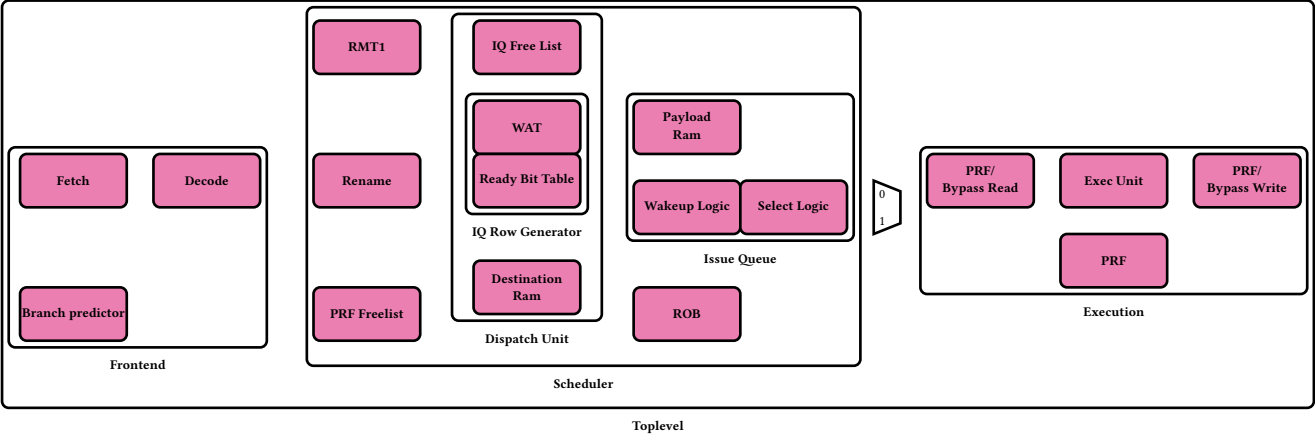
## Contents

Revision History .....	1
1 Terminology .....	1
2 Overview .....	2
3 Parameters .....	2
4 Interface .....	2
4.1 BP_IFU_Interface .....	2
4.2 BP_ROB_Interface .....	2
4.3 BP_IO .....	3
4.4 ALUIO .....	3
4.5 BypassNetworkIO .....	4
4.6 DCacheRequest .....	4
4.7 DCacheResponse .....	4
4.8 LSUMemIO .....	4
4.9 LSUCoreIO .....	5
4.10 LSUIO .....	6
5 Microarchitecture .....	7
5.1 Core .....	7
5.2 BranchPredictor .....	7
5.3 PayloadRAM .....	7
5.4 WakeupLogic .....	7
5.5 RenameUnit .....	8
5.6 ROB .....	9
5.7 DestinationRAM .....	10
5.8 ReadyBitTable .....	10
5.9 ReadyBitTable .....	10
5.10 WAT .....	10
5.11 ALU .....	11
5.12 BypassNetwork .....	11
5.13 LSU .....	11

## 1 Terminology

**PRF** Physical Register File  
**WAT** Wakeup Allocation Table  
**RMT** Register Map Table  
**SDA** Store Data Array  
**IRL** Instruction Replay Logic  
**STQ** STore Queue  
**LDQ** Load Queue

2 Overview



3 Parameters

4 Interface

4.1 BP\_IFU\_Interface

BP\_IFU\_Interface 的构造参数如下：

Name	Type	Description
fetch_width	Int	val GHR_width : Int

IO 端口定义如下：

Name	Type	Description
PC_curr	Input(UInt(32.W))	当前 IFU 的 PC 值
PC_target	Output(Vec(fetch_width, UInt(32.W)))	预测的目标地址
BTB_Hit	Output(Vec(fetch_width, Bool()))	是否命中 BTB
BHT_Taken	Output(Vec(fetch_width, Bool()))	是否预测为 taken
GHR	Output(UInt(GHR_width.W))	作出预测时的全局历史寄存器快照，使得更新 BHT 时能够生成正确的 index

4.2 BP\_ROB\_Interface

BP\_ROB\_Interface 的构造参数如下：

Name	Type	Description
GHR_width	Int	

IO 端口定义如下：

Name	Type	Description
PC	<code>Input(UInt(32.W))</code>	当前 ROB 的 PC 值
instrType	<code>Input(UInt(2.W))</code>	当前指令类型,该模块需要区分条件分支和无条件分支
BTB_Hit	<code>Input(Bool())</code>	是否命中 BTB。根据条件分支最初是否命中 BTB，给出不同的更新 BHT 策略
predict_Taken	<code>Input(Bool())</code>	是否预测为 taken
actual_Taken	<code>Input(Bool())</code>	实际是否 taken
GHR	<code>Input(UInt(GHR_width.W))</code>	作出预测时的全局历史寄存器快照，使得更新 BHT 时能够生成正确的 index
actualTargetPC	<code>Input(UInt(32.W))</code>	实际跳转的目标地址

### 4.3 BP\_IO

BP\_IO 的构造参数如下：

Name	Type	Description
fetch_width	<code>Int</code>	<code>val GHR_width : Int</code>

IO 端口定义如下：

Name	Type	Description
ifu	<code>new BP_IFU_Interface(fetch_width, GHR_width)</code>	
rob	<code>new BP_ROB_Interface(GHR_width)</code>	

### 4.4 ALUIO

IO 端口定义如下：

Name	Type	Description
in1	<code>Input(UInt(p(XLen).W))</code>	
in2	<code>Input(UInt(p(XLen).W))</code>	
fn	<code>Input(UInt(4.W))</code>	
out	<code>Output(UInt(p(XLen).W))</code>	
cmp_out	<code>Output(Bool())</code>	

## 4.5 BypassNetworkIO

IO 端口定义如下：

Name	Type	Description
exec_units	<code>Input(Vec(2, Valid(new BypassInfo)))</code>	
preg_rd	<code>Input(UInt(p(PhysRegIdxSz).W))</code>	
data_out	<code>Output(UInt(p(XLen).W))</code>	

## 4.6 DCacheRequest

IO 端口定义如下：

Name	Type	Description
addr	<code>UInt(p.CoreMaxAddrbits.W)</code>	
data	<code>Bits(p.CoreDataBits.W)</code>	

## 4.7 DCacheResponse

IO 端口定义如下：

Name	Type	Description
addr	<code>UInt(p.CoreMaxAddrbits.W)</code>	
data	<code>Bits(p.CoreDataBits.W)</code>	

## 4.8 LSUMemIO

描述与数据内容交互的各种信号，用于管理 L/S 请求和缓存访问，包含一部分异常处理、顺序控制的功能

IO 端口定义如下：

Name	Type	Description
req	<code>new DecoupledIO(p.lsuWidth, Valid(new DCacheRequest))</code>	LSU 发出的数据缓存请求
resp	<code>new Flipped(p.lsuWidth, Valid(new DCacheResponse))</code>	LSU 接收数据缓存响应
req_kill	<code>Output(p.lsuWidth, Bool())</code>	表示每个 req 是否被 kill
req_nack_adv	<code>Input(p.lsuWidth, Bool())</code>	表示某个 req 是否被拒绝
store_ack	<code>Flipped(Vec(p.lsuWidth, new ValidIO(new DCacheRequest)))</code>	存储请求的确认

nack	Flipped(Vec(p.lsuWidth, new ValidIO(new DCacheRequest)))	作为接口接受 neck
load_rel_resp	Flipped(new DecoupledIO(new DCacheResponse))	作为接口接受缓存的 load/release 响应
bradate	Output(new bradateInfo)	报告分支更新信息
exception	Output(Bool())	输出异常
rob_pnr_idx	Output(UInt((p.robAddrSz).W))	
rob_head_idx	Output(UInt((p.robAddrSz).W))	rob 中的后备和头部索引
release	Flipped(new DecoupledIO(new TLBundle(edge.bundle)))	处理缓存协议的释放操作
force_order	Output(Bool())	强制顺序控制，在保证 l/s 顺序的时候激活
order	Input(Bool())	顺序控制信号，表示当前是否满足顺序要求
perf	Input(new Bundle {	
acquire	Bool()	
release	Bool()	

## 4.9 LSUCoreIO

LSU 的核心 IO 接口，与 execution、dcache、rob 等进行交互

IO 端口定义如下：

Name	Type	Description
data	UInt(xLen.W)	
agen	Flipped(Vec(p.lsuWidth, Valid(new MemGen)))	地址生成器的输入
dgen	Flipped(Vec(p.memWidth + 1, Valid(new MemGen)))	数据生成器的输入, memWidth 为内存宽度
bypassable	Bool()	
speculative_mask	UInt(aluWidth.W)	
rebusy	Bool()	
iwakeups	Vec(p.lsuWidth, Valid(new Wakeup))	LSU 的唤醒信号
iresp	Vec(p.lsuWidth, Valid(new ExeUnitResp(xLen)))	
fresp	Vec(p.lsuWidth, Valid(new ExeUnitResp(xLen)))	

<b>sfence</b>	<code>Flipped(Valid(new rocket.SFenceReq))</code>	传递 sfence 的请求，用于内容屏蔽
<b>dis_uops</b>	<code>Vec(p.coreWidth, Valid(new MicroOp))</code>	解码后的微操作
<b>dis_ldq_idx</b>	<code>Output(Vec(p.coreWidth, UInt((1+ldpAddrSz).W)))</code>	
<b>dis_stq_idx</b>	<code>Output(Vec(p.coreWidth, UInt((1+stqAddrSz).W)))</code>	
<b>ldq_full</b>	<code>Output(Vec(p.coreWidth, Bool()))</code>	
<b>stq_full</b>	<code>Output(Vec(p.coreWidth, Bool()))</code>	
<b>commit</b>	<code>Input(new CommitInfo)</code>	提交信号 CommitInfo 是一个 Bundle，包含了提交的指令信息
<b>commitLoadAtRobHead</b>	<code>Input(Bool())</code>	当前是否在 rob 的头部进行加载
<b>clr_busy_bit</b>	<code>Output(Bool())</code>	清除 busybit 的信号
<b>clr_unsafe</b>	<code>Output(Bool())</code>	清除不安全的加载状态
<b>fence_dmem</b>	<code>Input(Bool())</code>	控制内存屏障操作
<b>bradate</b>	<code>Input(new bradateInfo)</code>	分支更新信息
<b>rob_head_idx</b>	<code>Input(UInt((p.robAddrSz).W))</code>	rob 的头部索引
<b>rob_pnr_idx</b>	<code>Input(UInt((p.robAddrSz).W))</code>	rob 的后备索引
<b>exception</b>	<code>Input(Bool())</code>	异常信号
<b>Fencei_ready</b>	<code>Output(Bool())</code>	表示 FENCEI 是否准备好
<b>load_excep</b>	<code>Output(Valid(new Exception))</code>	表示加载异常的信号
<b>tsc_reg</b>	<code>Input(UInt())</code>	时间戳寄存器(计数器)的值
<b>status</b>	<code>Input(new rocket.MStatus)</code>	接收机器模式状态寄存器的值
<b>bp</b>	<code>Input(Vec(breakpoint_num, new rocket.BP))</code>	接收断点设置
<b>mach_context</b>	<code>Input(UInt)</code>	接收机器模式上下文的值
<b>s_context</b>	<code>Input(UInt)</code>	接收超线程模式上下文的值
<b>perf</b>	<code>Output(new Bundle {</code>	
<b>acquire</b>	<code>Bool()</code>	
<b>release</b>	<code>Bool()</code>	
<b>tlbMiss</b>	<code>Bool()</code>	tlb 可能不需要，此处待定

## 4.10 LSUIO

提供 LSU 的核心、内存 (TLB) 的 IO 接口

IO 端口定义如下：

Name	Type	Description
ptw	<code>new rocket.TLBPTWIO</code>	TLB 的输入输出接口,待定
core	<code>new LSUCoreIO</code>	LSU 的核心接口
dcache_mem	<code>new LSUMemIO</code>	LSU 的内存接口

## 5 Microarchitecture

本处理器为乱序执行多发射 RV32IM 架构，其设计主要借鉴于 RSD 以及 BOOM。

### 5.1 Core

主要核心

IO 端口定义如下：

Name	Type	Description
imem	<code>new InstructionMemoryInterface</code>	
dmem	<code>new DataMemoryInterface</code>	

### 5.2 BranchPredictor

BranchPredictor 的构造参数如下：

Name	Type	Description
fetch_width	<code>Int</code>	<code>val GHR_width : Int</code>

### 5.3 PayloadRAM

PayloadRAM 是一个 RAM 模块，用于存储指令的有效载荷。

IO 端口定义如下：

Name	Type	Description
we	<code>Input(Vec(p.DISPATCH_WIDTH, Bool()))</code>	写使能信号
waddr	<code>Input(Vec(p.DISPATCH_WIDTH, UInt(log2Ceil(NUM_PayloadRAM).W)))</code>	写地址
wdata	<code>Input(Vec(p.DISPATCH_WIDTH, UInt(Insr_WIDTH.W)))</code>	写数据
raddr	<code>Input(Vec(p.ISSUE_WIDTH, UInt(log2Ceil(NUM_PayloadRAM).W)))</code>	读地址
rdata	<code>Output(Vec(p.ISSUE_WIDTH, UInt(Insr_WIDTH.W)))</code>	读数据

### 5.4 WakeupLogic

WakeupLogic 是一个唤醒逻辑模块，用于处理指令的唤醒信号。它根据指令的依赖关系和状态，决定哪些指令可以被唤醒。

IO 端口定义如下：

Name	Type	Description
stall	<code>Input(Bool())</code>	停顿信号
write	<code>Input(Vec(p.DISPATCH_WIDTH, Bool()))</code>	写使能
writePtr	<code>Input(Vec(p.DISPATCH_WIDTH, UInt(p.ISSUE_QUEUE_INDEX_WIDTH.W)))</code>	写指针
writeSrcTag	<code>Input(Vec(p.DISPATCH_WIDTH, new SrcTag()))</code>	写源标签,SrcTag 可能包含寄存器标签与寄存器地址
writeDstTag	<code>Input(Vec(p.DISPATCH_WIDTH, new DstTag()))</code>	写目标标签,DstTag 可能包含寄存器标签
wakeup	<code>Input(Vec(p.WAKEUP_WIDTH, Bool()))</code>	唤醒信号
wakeupDstTag	<code>Input(Vec(p.WAKEUP_WIDTH, new DstTag()))</code>	唤醒目标标签
wakeupVector	<code>Input(Vec(p.WAKEUP_WIDTH + p.ISSUE_STORE_WIDTH, Vec(p.ISSUE_QUEUE_ENTRY_NUM, Bool())))</code>	唤醒向量
notIssued	<code>Input(Vec(p.ISSUE_QUEUE_ENTRY_NUM, Bool()))</code>	未发射标志
dispatchStore	<code>Input(Vec(p.DISPATCH_WIDTH, Bool()))</code>	是否是 Store
dispatchLoad	<code>Input(Vec(p.DISPATCH_WIDTH, Bool()))</code>	是否是 Load
memDependencyPred	<code>Input(Vec(p.DISPATCH_WIDTH, Bool()))</code>	内存依赖预测
opReady	<code>Output(Vec(p.ISSUE_QUEUE_ENTRY_NUM, Bool()))</code>	操作就绪标志

## 5.5 RenameUnit

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。

IO 端口定义如下：

Name	Type	Description
in	<code>Flipped(Vec(p.RENAME_WIDTH, DecodedInstr))</code>	输入
out	<code>Vec(p.RENAME_WIDTH, RenamedInsr)</code>	soaeuaoeu



## 5.6 ROB

重命名缓冲区，主要用于存储指令的执行结果。它是一个 FIFO 结构，先进先出。指令在执行完成后，将结果写入 ROB 中。ROB 中的数据可以被其他指令读取，从而实现数据的共享和重用。

ROB 的构造参数如下：

Name	Type	Description
num_WakeupPorts	Int	

IO 端口定义如下：

Name	Type	Description
dispatch_in	Flipped(Vec(p.DISPATCH_WIDTH, DispatchedInsr))	来自 Dispatch Unit 的输入
branch_in	Flipped(Vec(p.BRANCH_WIDTH, BranchInfo))	来自 Branch Predictor 的输入
exception_fetch_pc	Input(UInt(p.ADDR_WIDTH.W))	异常发生时的 PC 值
ROB_head_idx	Output(UInt(p.ROB_ADDR_WIDTH.W))	ROB 头指针
ROB_tail_idx	Output(UInt(p.ROB_ADDR_WIDTH.W))	ROB 尾指针
ROB_pnr_idx	Output(UInt(p.ROB_ADDR_WIDTH.W))	ROB 安全上限指针
ROB_empty	Output(Bool())	ROB 空标志
ROB_ready	Output(Bool())	ROB 准备好标志(不仅仅是非满)
WB_resps	Flipped(Vec(num_WakeupPorts, Valid(?待确认?)))	来自执行单元的写回响应
LSU_clr_busy	Input(Vec(p.DISPATCH_WIDTH, Valid(UInt(p.ROB_ADDR_WIDTH.W))))	来自 LSU 的清除忙标志
LSU_clr_unsafe	Input(Vec(p.LSU_WIDTH, Valid(UInt(p.ROB_ADDR_WIDTH.W))))	来自 LSU 的清除不安全标志
LSU_exception	Input(Valid(new Exception()))	来自 LSU 的异常标志(Exception 类待定义)
CSR_replay	Input(Valid(new Exception()))	来自 CSR 的重放标志(Exception 类待定义)
CSR_stall	Input(Bool())	来自 CSR 的停顿标志
commit	Output(new Commit_Info())	提交信息(Commit_Info 类待定义)
rollback	Output(Bool())	回滚标志
commit_exception	Output(Valid(new Commit_exception_Info()))	提交异常信息 (To CSR) (Commit_exception_Info 类待定义)

<b>flush</b>	<code>Output(Valid(new Commit_exception_Info()))</code>	刷新标志(Commit_exception_Info 类待定义)
<b>flush_frontend</b>	<code>Output(Bool())</code>	前端刷新标志

## 5.7 DestinationRAM

IO 端口定义如下：

Name	Type	Description
<b>wrEn</b>	<code>Input(Bool())</code>	
<b>wrReg</b>	<code>Input(UInt(params.phyRegBits.W))</code>	
<b>wrIQIdx</b>	<code>Input(UInt(params.iqIdxBits.W))</code>	
<b>rdReg1</b>	<code>Input(UInt(params.phyRegBits.W))</code>	
<b>rdIQIdx1</b>	<code>Output(UInt(params.iqIdxBits.W))</code>	

## 5.8 ReadyBitTable

IO 端口定义如下：

Name	Type	Description
<b>init</b>	<code>Flipped(Valid(new Bundle { ... })))</code>	
<b>setReady</b>	<code>Flipped(Valid(new Bundle { ... })))</code>	
<b>readyBits</b>	<code>Output(Vec(params.iqEntries, Vec(2, Bool())))</code>	

## 5.9 ReadyBitTable

IO 端口定义如下：

Name	Type	Description
<b>init</b>	<code>Flipped(Valid(new Bundle { ... })))</code>	
<b>setReady</b>	<code>Flipped(Valid(new Bundle { ... })))</code>	
<b>readyBits</b>	<code>Output(Vec(params.iqEntries, Vec(2, Bool())))</code>	

## 5.10 WAT

IO 端口定义如下：

Name	Type	Description
<b>addEntry</b>	<code>Flipped(Valid(new Bundle { ... })))</code>	
<b>wakeup</b>	<code>Flipped(Valid(UInt(params.phyRegBits.W)))</code>	

```
setReady Valid(new Bundle { ... })
```

## 5.11 ALU

## 5.12 BypassNetwork

## 5.13 LSU

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