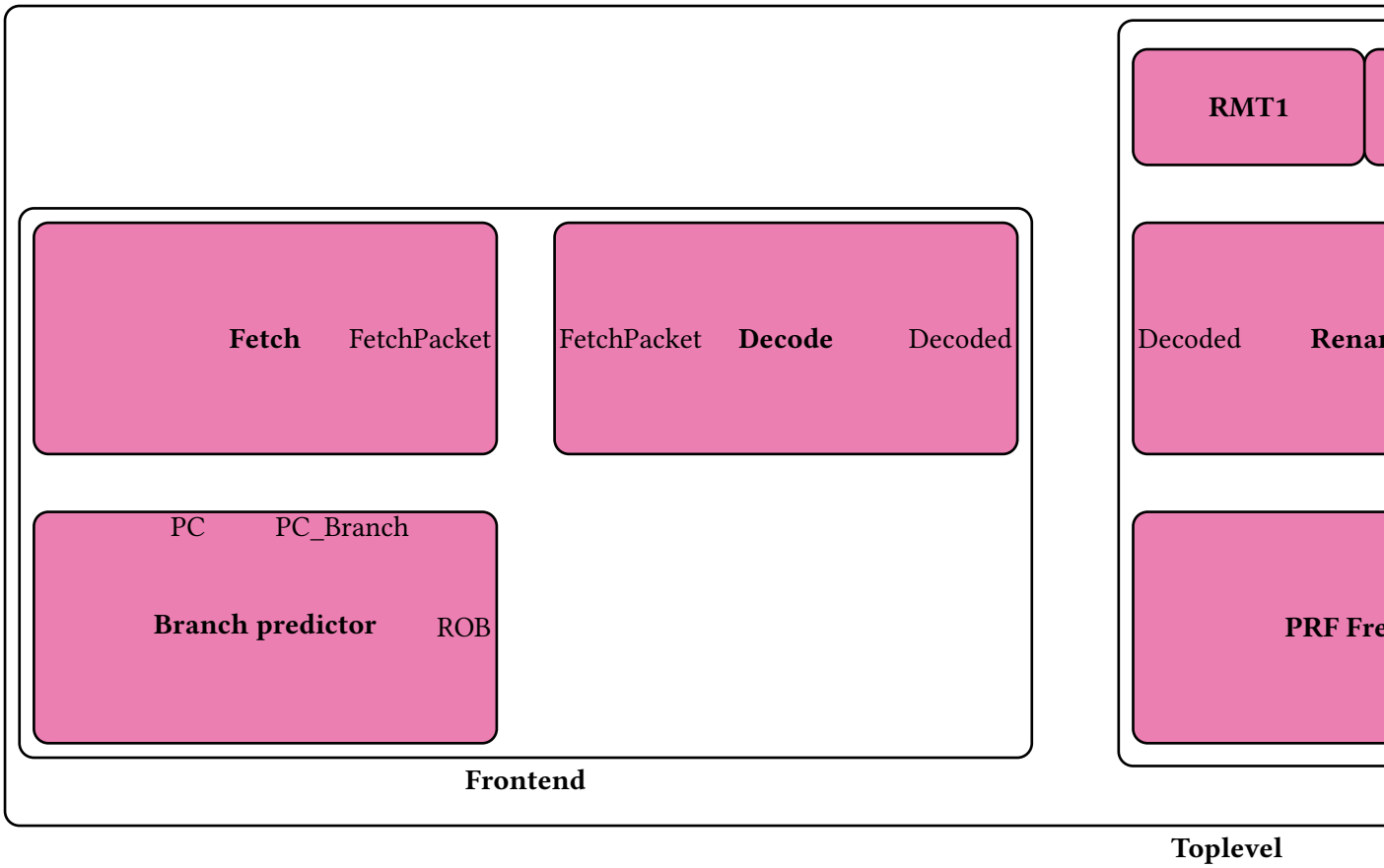
Revision History

Contents

1 Terminology

LE None

2 Overview



3 Parameters

4 Interface

5 Microarchitecture

5.1 FreeList

参数化的 Freelist

FreeList 的构造参数如下：

Name	Type	Description
read_ports	Int	Number of read ports
depth	Int	Depth of the freelist buffer

```scala
  val io = IO(
    val requests = Input(Vec(depth, Bool()))
    val data  = Output(Vec(depth, Valid(UInt(data_width.W))))

    val dealloc  = Input(Vec(depth, Valid(UInt(data_width.W))))
  )
```

5.2 PRFFreeList

获取 PRF 中可用的寄存器

PRFFreeList 的构造参数如下：

Name	Type	Description
read_ports	Int	with desc
depth	Int	Another desc
data_width	Int	

```scala
  //BLablabla

```

5.3 DCacheRequest

```scala
  val addr = UInt(p.CoreMaxAddrbits.W)
  val data = Bits(p.CoreDataBits.W)
```

5.4 DCacheResponse

```scala
  val addr = UInt(p.CoreMaxAddrbits.W)
  val data = Bits(p.CoreDataBits.W)
```

5.5 LSUMemIO

描述与数据内容交互的信号

```scala
val req           = new DecoupledIO(p.lsuWidth,Valid(new DCacheRequest))//LSU发出的数据缓存请求
val resp          = new Flipped(p.lsuWidth,Valid(new DCacheResponse))//LSU接收数据缓存响应

val req_kill      = Output(p.lsuWidth,Bool())//表示每个req是否被kill
val req_nack_adv  = Input(p.lsuWidth,Bool())//表示某个req是否被拒绝
val store_ack     = Flipped(Vec(p.lsuWidth,new ValidIO(new DCacheRequest)))//存储请求的确认
val nack          = Flipped(Vec(p.lsuWidth,new ValidIO(new DCacheRequest)))//作为接口接受neck
val load_rel_resp = Flipped(new DecoupledIO(new DCacheResponse))//作为接口接受缓存的load/release响应
val bradate       = Output(new bradateInfo)//报告分支更新信息
val exception     = Output(Bool())//输出异常

val rob_pnr_idx   = Output(UInt((p.robAddrSz).W))
val rob_head_idx  = Output(UInt((p.robAddrSz).W))//rob中的后备和头部索引

val release       = Flipped(new DecoupledIO(new TLBundle(edge.bundle)))//处理缓存协议的释放操作

val force_order   = Output(Bool())//强制顺序控制，在保证l/s顺序的时候激活
val order         = Input(Bool())//顺序控制信号，表示当前是否满足顺序要求

val perf          = Input(new Bundle {
  val acquire     = Bool()
  val release     = Bool()
})
//指示信号的获取和释放，进行性能的监控
```