结题报告

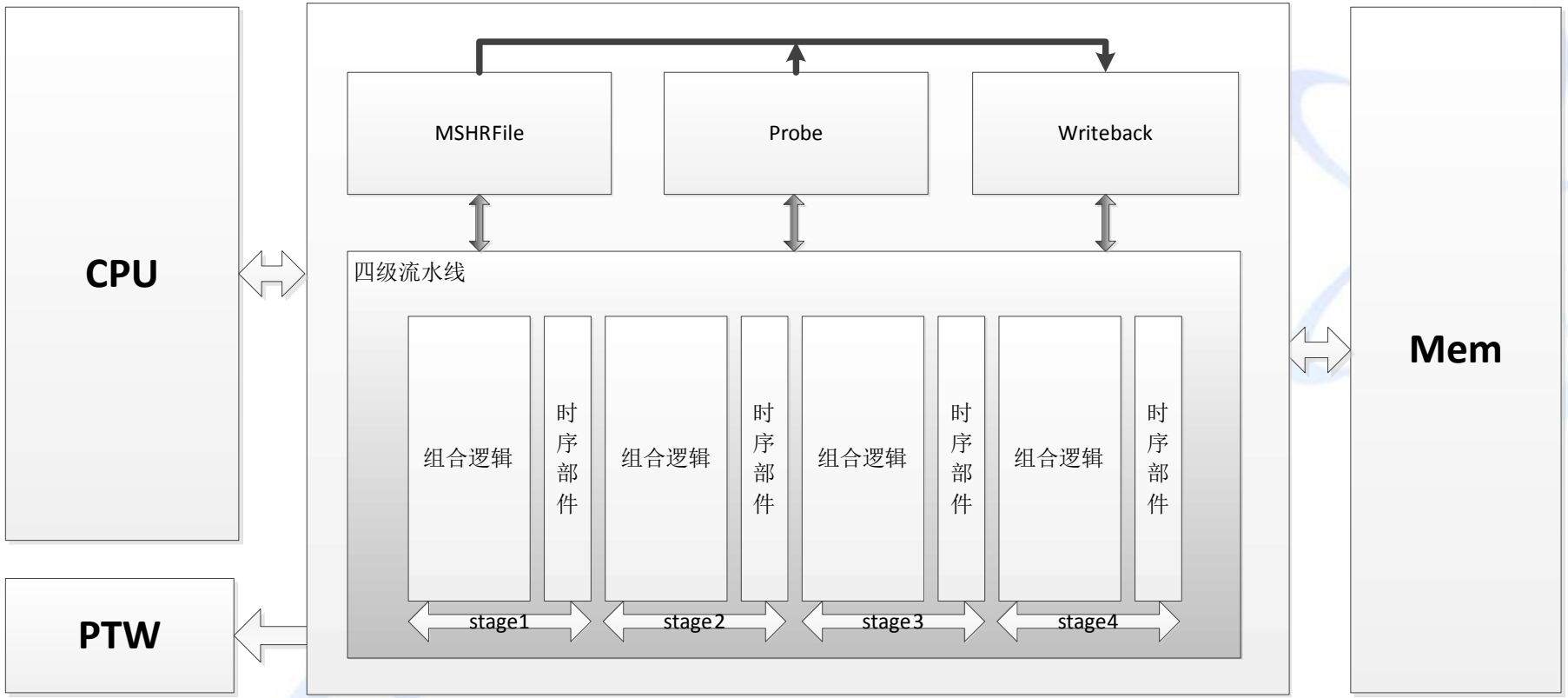
MPRC-NBDCACHE

刘文利

主要内容

- 整体结构
- 双端口SRAM
- 四级流水线
- 读写模式
- 原子操作
- 非阻塞设计

1 整体结构及主要功能


CPU
PTW
MSHRFile
Probe
Writeback
四级流水线
组合逻辑
时序部件
组合逻辑
时序部件
组合逻辑
时序部件
组合逻辑
时序部件
stage1
stage2
stage3
stage4
Mem

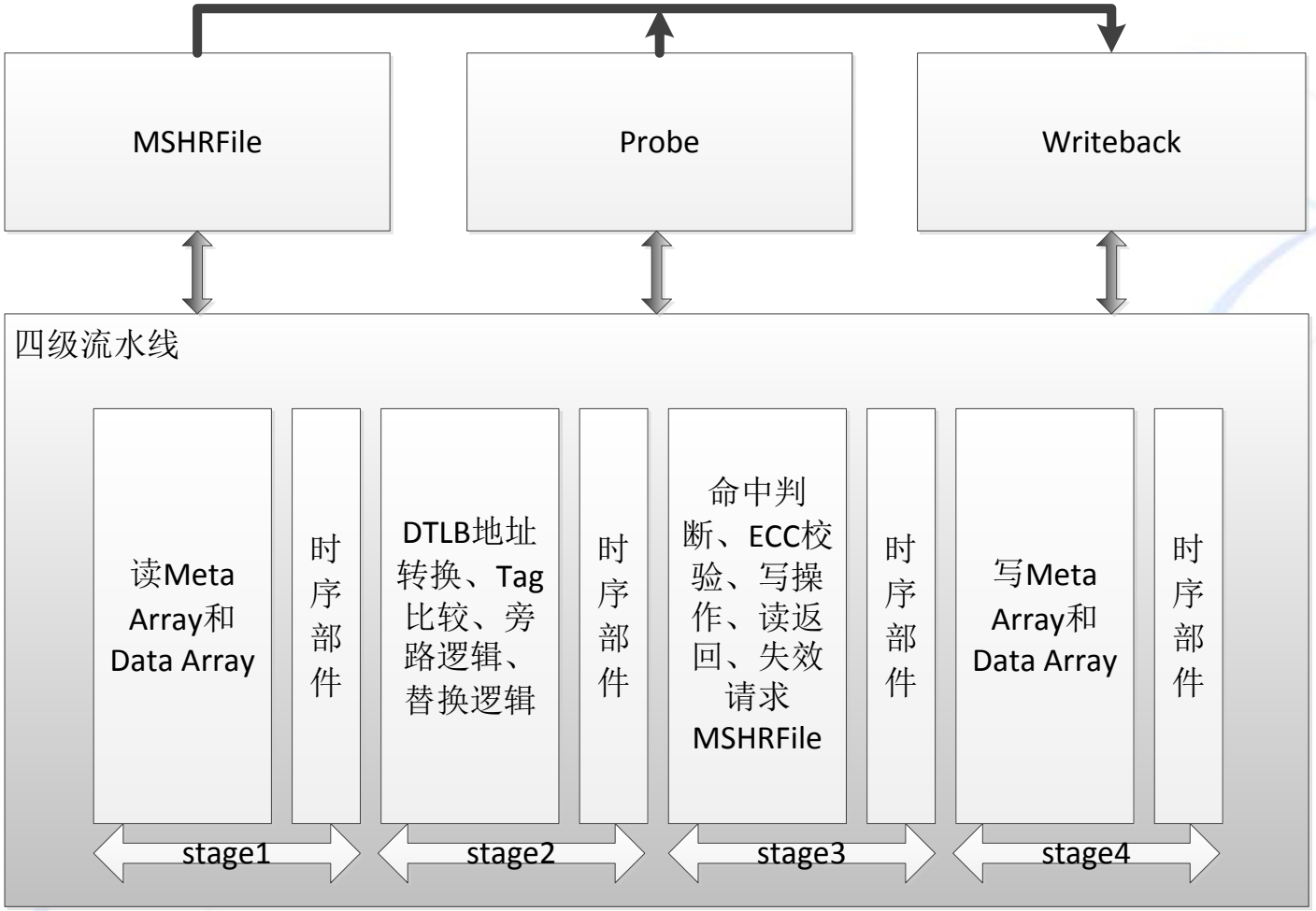
1 整体结构及主要功能

- 大小为16KB，组织结构为4路组相连，总共64组
- 每个Cache 行包括16个字(64 Bytes)、Tag(20 bits)、状态位(2 bits)
- 每个Cache 行又分为4个子块，每个子块大小为16 Bytes
- 采用虚拟索引、物理标签（Virtual-Index、Physical-Tag）寻址
- 多Bank设计
- 采用写返回（Write-Back）与按写分配（Write-Allocate）的写策略
- 和L1 DTLB并行查找
- 采用Pseudo-Least-Recently-Used（Pseudo-LRU）替换算法
- 支持Modified/Exclusive/Shared/Invalid（MESI）一致性协议
- 双端口SRAM
- 四级的Cache流水操作
- 支持多种读写模式
- 支持原子存储操作
- 支持Misses under Misses非阻塞访问

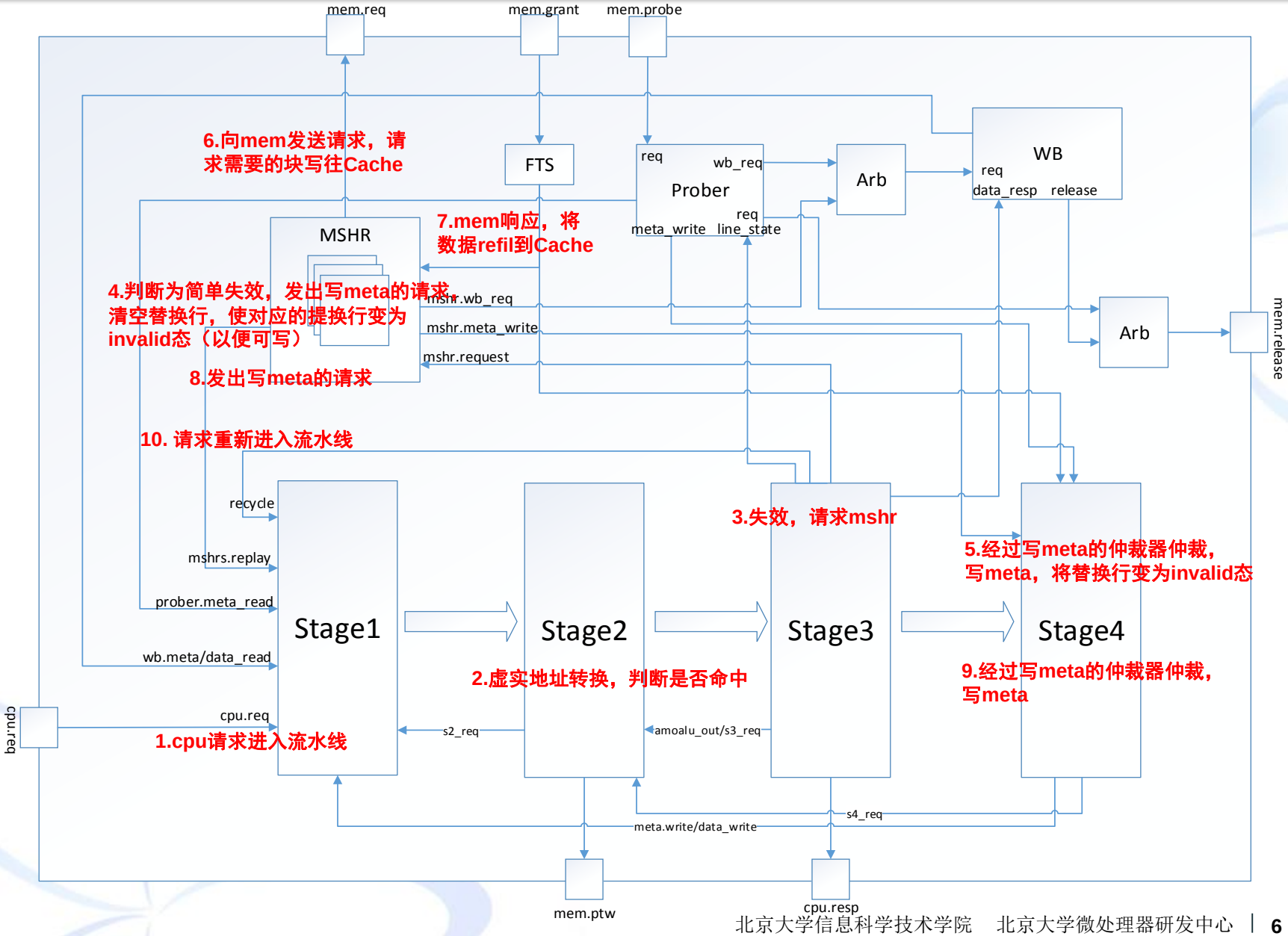
2 双端口SRAM

- 双端口SRAM
  - 一个读口一个写口，可同时对SRAM进行读写。
  - 配合四级流水线设计，可在一个时钟周期内对Cache进行读写，提高Cache性能。
  - 读写同一地址时，读出的数据不是最新的，会调用bypass逻辑进行处理。

3 四级流水线


MSHRFile
Probe
Writeback
四级流水线
读Meta Array和 Data Array
时序部件
DTLB地址转换、Tag比较、旁路逻辑、替换逻辑
时序部件
命中判断、ECC校验、写操作、读返回、失效请求 MSHRFile
时序部件
写Meta Array和 Data Array
时序部件
stage1
stage2
stage3
stage4

3 四级流水线——简单失效

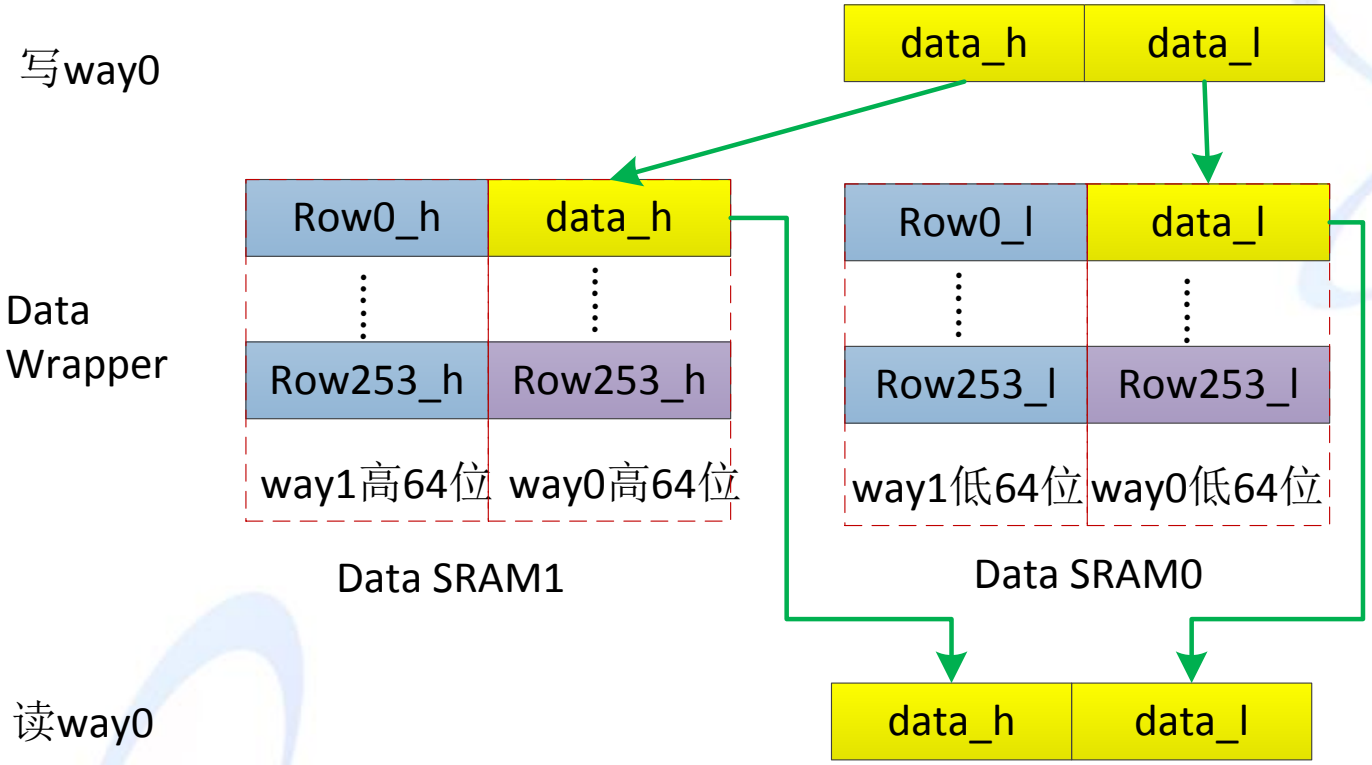

mem.req
mem.grant
mem.probe
6.向mem发送请求，请求需要的块写往Cache
FTS
req
wb_req
Prober
req
meta_write
line_state
Arb
WB
req
data_resp
release
MSHR
7.mem响应，将数据refil到Cache
4.判断为简单失效，发出写meta的请求，清空替换行，使对应的提换行变为invalid态（以便可写）
mshr.wb_req
mshr.meta_write
mshr.request
8.发出写meta的请求
Arb
mem.release
10. 请求重新进入流水线
recycle
mshrs.replay
prober.meta_read
wb.meta/data_read
cpu.req
1.cpu请求进入流水线
Stage1
Stage2
Stage3
Stage4
3.失效，请求mshr
5.经过写meta的仲裁器仲裁，写meta，将替换行变为invalid态
2.虚实地址转换，判断是否命中
9.经过写meta的仲裁器仲裁，写meta
s2_req
amoalu_out/s3_req
s4_req
meta.write/data_write
cpu.req
mem.ptw
cpu.resp

4 读写模式

- **按照读写模式分类：**
  - ✓ 写模式：写128位，写高64位，写低64位。
  - ✓ 读模式：读128位，读高64位。
- **按照请求类型分类：**
  - ✓ memory读写请求：读128位请求；写128位请求。
    - 读128位请求：由读128位读模式实现。
    - 写128位请求：由写128位写模式实现
  - ✓ cpu读写请求：读高64位请求、读低64位请求；写高64位请求、写低64位请求。
    - 读高64位请求：由读高64位读模式实现。
    - 读低64位请求：由读128位读模式实现（在流水线会有截取低64位处理）。
    - 写高64位请求：由写高64位写模式实现。
    - 写低64位请求：由写低64位写模式实现。

4 读写模式

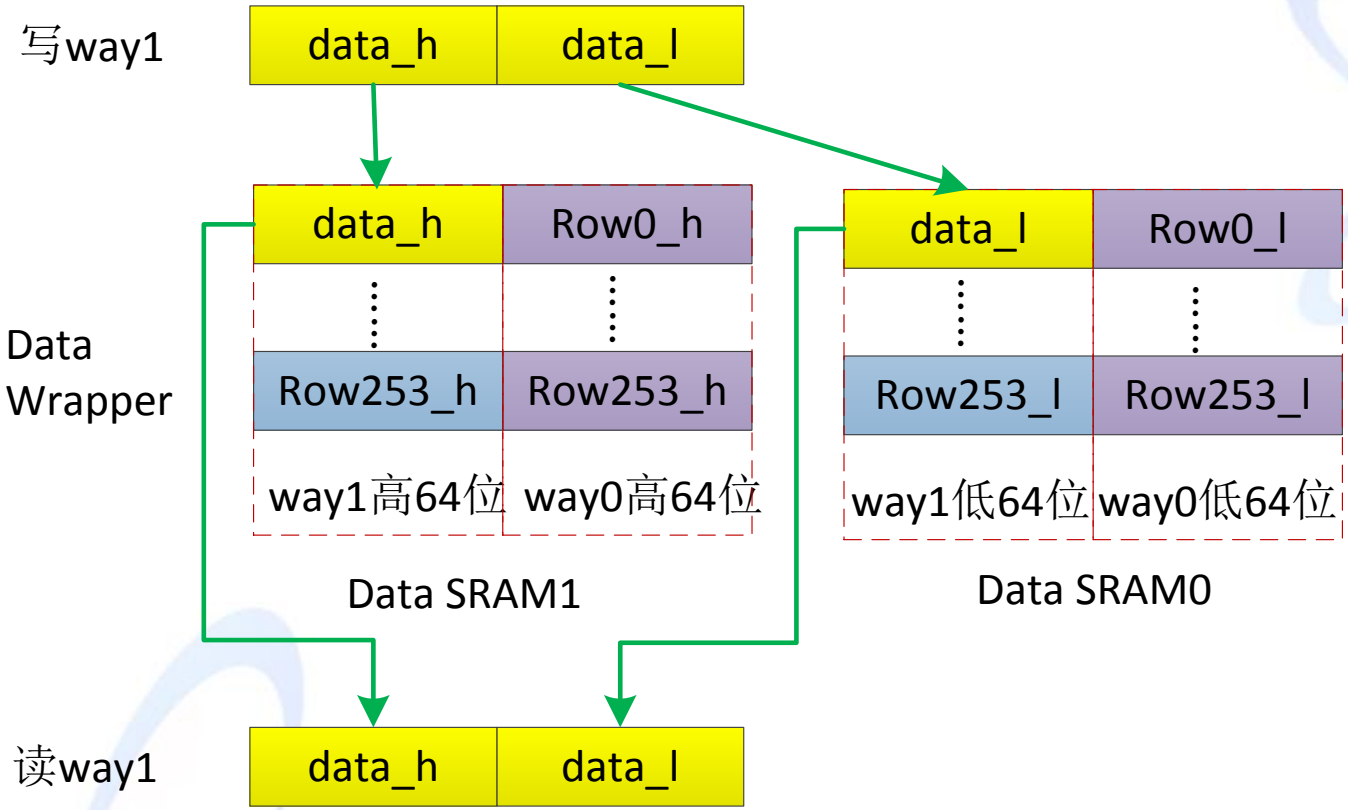
- 读写数据时会进行重新组合
- memory读写way0



memory读写way0

4 读写模式

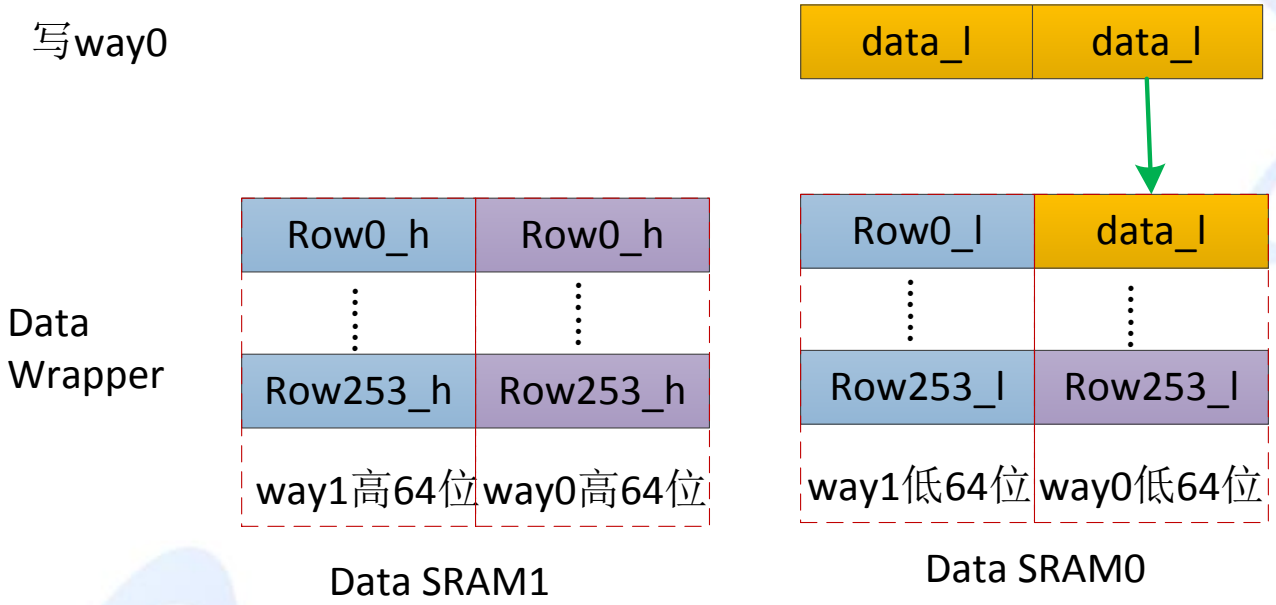
- memory读写way1



memory读写way1

4 读写模式

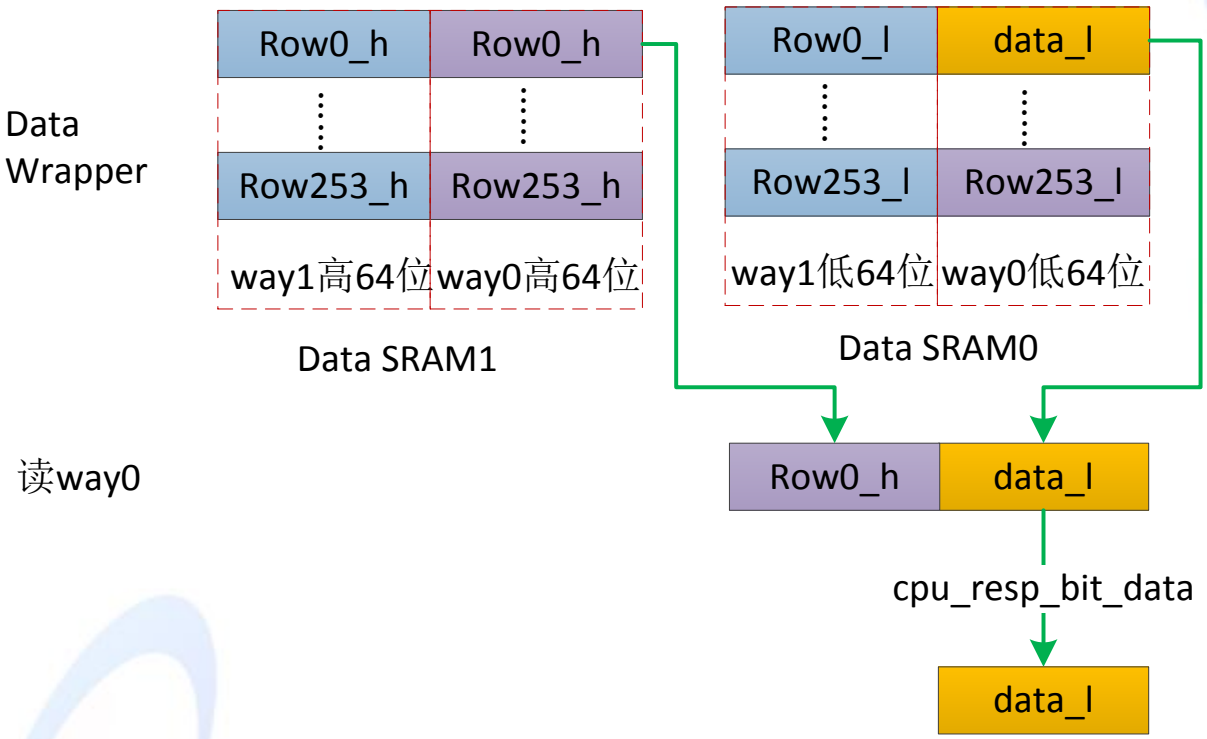
- CPU写way0低64位



CPU写way0低64位

4 读写模式

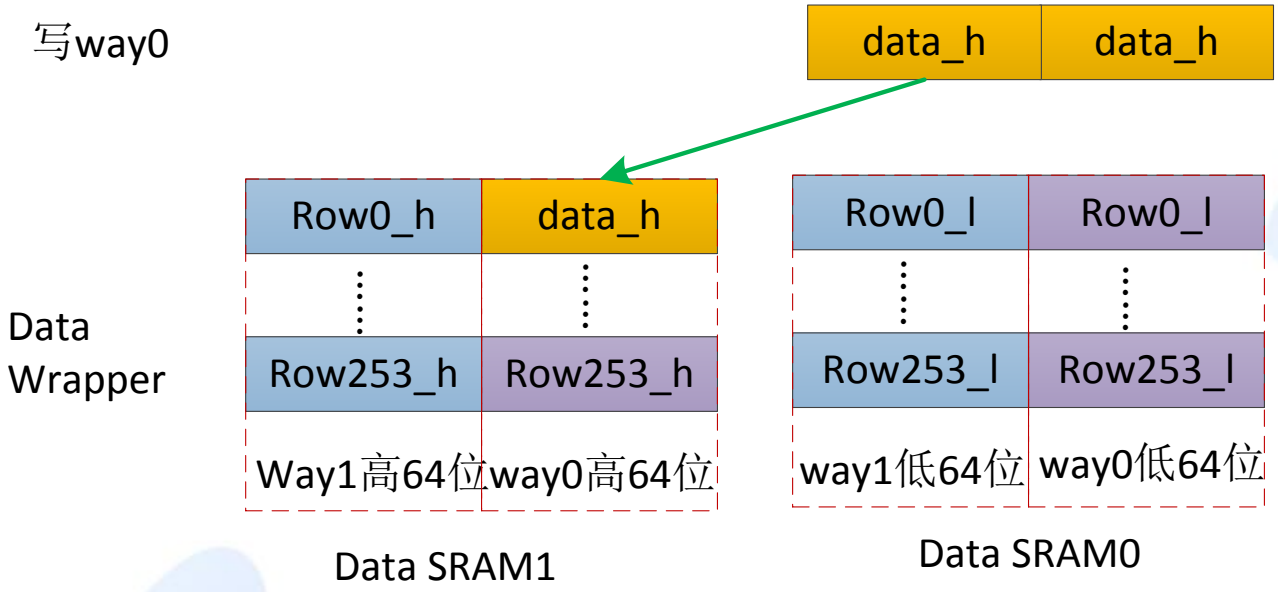
- CPU读way0低64位



CPU读way0低64位

4 读写模式

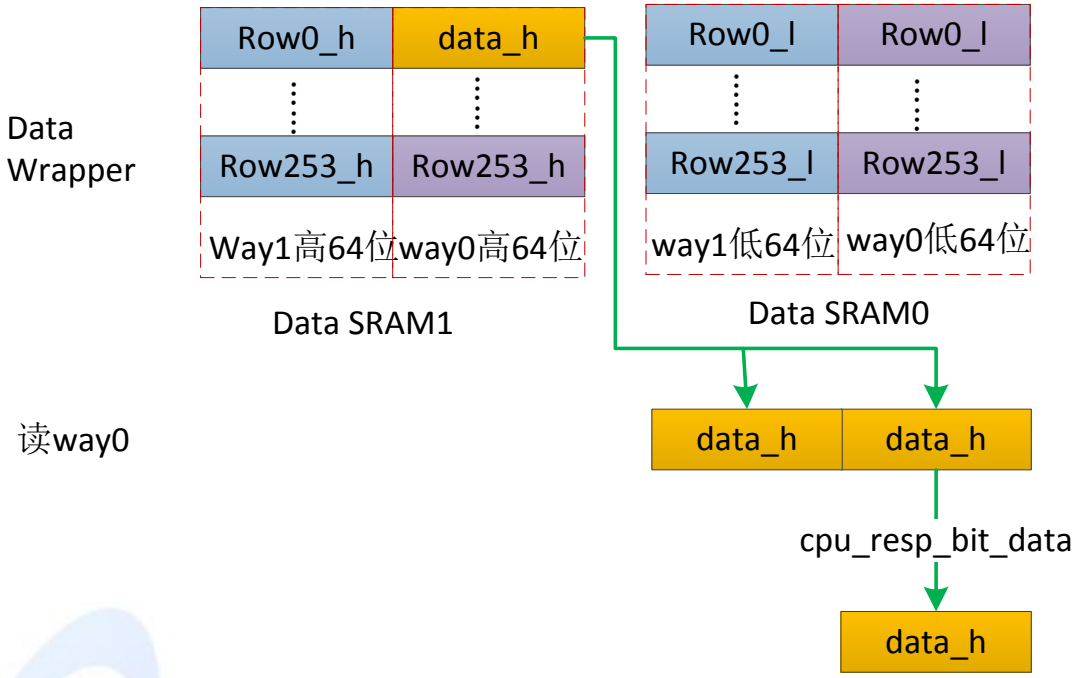
- CPU写way0高64位



CPU写way0高64位

4 读写模式

- CPU读way0高64位



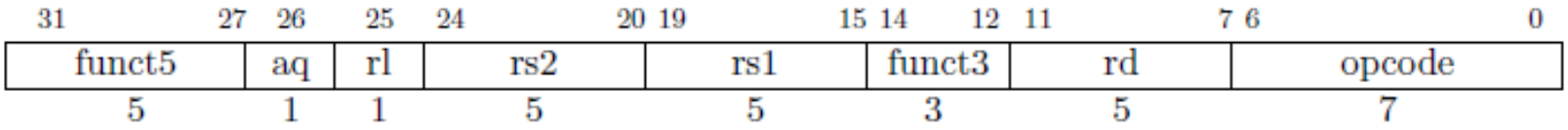
CPU读way0高64位

4 读写模式

- 优点：
  - 128位读写：留作float等double word类型的数据进行读写。
  - 128位读写：和mem直接进行128Byte交互，提高效率。
  - 将数据备份，数据冗余提高cache可靠性，实例化末级cache时可提高性能。
- 缺点：
  - 每次读写都要使能两路能耗会增大。
  - cache接口信号会比较复杂。

5 原子操作

- RISC-V有两类原子指令
  - load-reserved/store-conditional指令。
  - fetch-and-op原子存储指令——amoalu模块进行处理。
    - 指令类型：支持的操作包括交换、整数加、逻辑AND、逻辑OR、逻辑XOR和有符号、无符号整数最大值和最小值
    - 执行流程：从rs1地址读取数据值，将这个值写入寄存器rd，在这个值和寄存器rs2的原始值上施加一个二进制操作，然后把结果保存到rs1地址的存储器中。

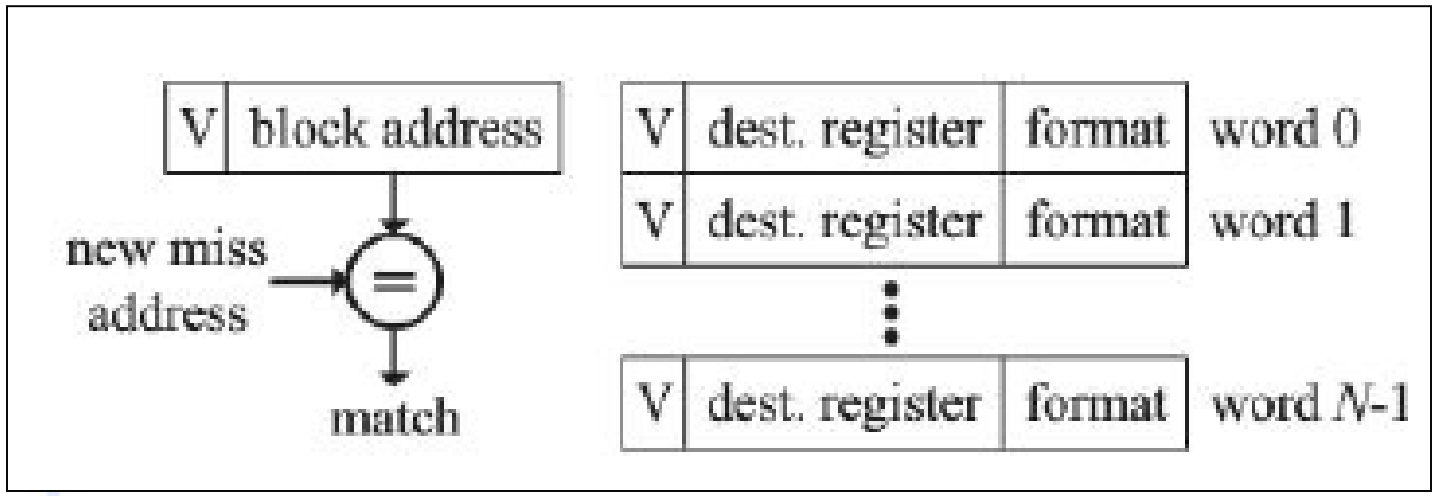


6 NBDCache实现

- 1. 缺失延迟：
  - ✓ 失效指令从开始访存到访存完成所用的时间。
- 2. 缺失代价：
  - ✓ 由失效指令引起的处理器停顿的时间。
- 3. 减少代价：
  - ✓ 对于store 指令缺失，可以增加 store buffer 来减少 store 缺失代价。Store指令提交到store buffer就相当于完成了，并且store buffer 提供bypass机制。
  - ✓ 对于load指令缺失，必须使用非阻塞的处理器和cache。非阻塞的cache可以保存失效的Load指令，并且在处理失效的同时还能够接受其它的访存指令。
- 4. Non-blocking cache
  - ✓ 通过重叠数据访问和数据计算的过程，来隐藏失效延迟， 但是，整个过程需要满足数据相关和一致性要求。
  - ✓ 实现Non-blocking cache的方法是：增加MSHR模块，用来保存、处理失效指令。这样一来，cache能继续接受其它的访存指令。

6.1 MSHR设计方法

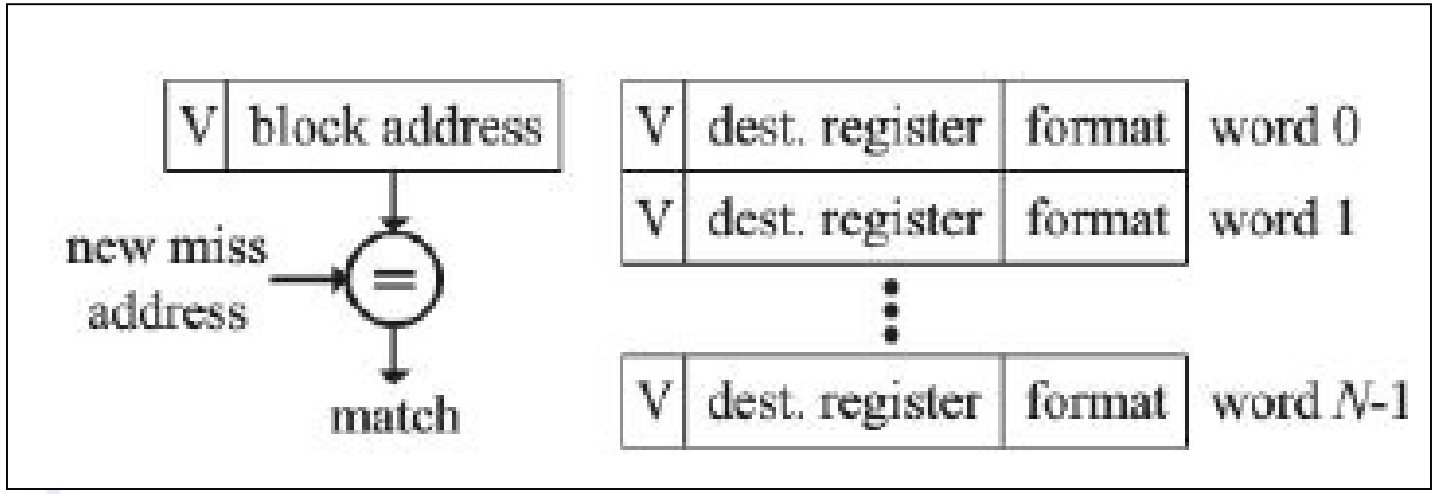
- 第一种设计：Implicitly Addressed MSHRs
  - block address域、valid域，这两个位域是在首次失效时被设置
  - 比较器，用来判断二次失效信息
  - Field表，有N个entry（N is the number of words in a block）
    - 保存失效指令详细信息：目的寄存器、格式信息（数据宽度、是否需要进行0扩展或者符号扩展、在一个word中的offset 等信息）



MSHR内部结构图

6.1 MSHR设计方法

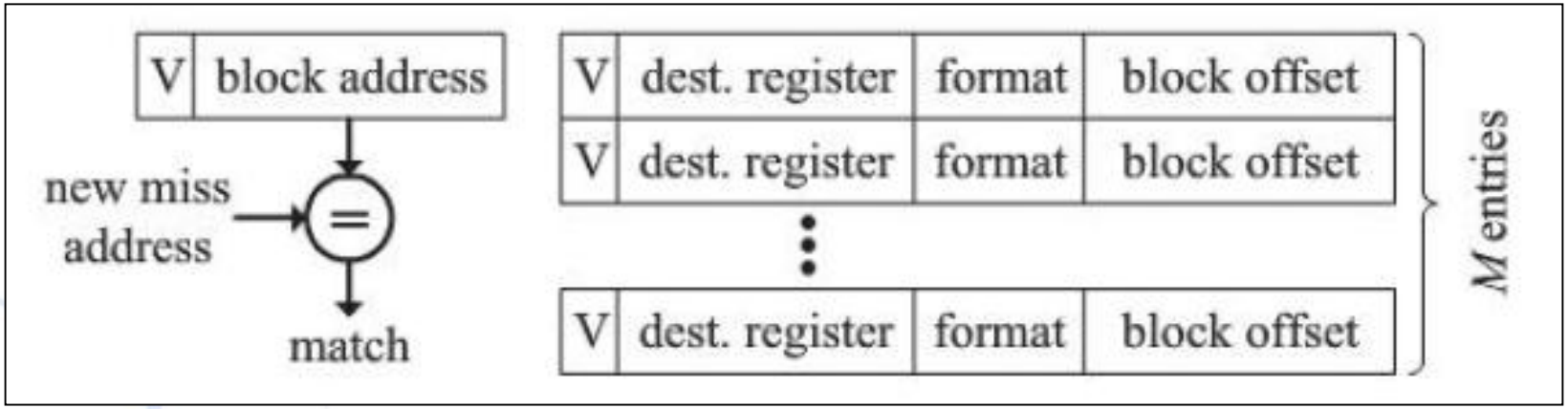
- 第一种设计：Implicitly Addressed MSHRs
  - 缺点：不接受一个word上的二次失效（miss on miss in the same word）
    - Block包含的words和MSHR中的entries（field表项）一一对应，对一个word的失效请求会保存到对应的表项，如果再来一个二次失效，那么，二次失效信息不能被保存，所以，MSHR不会接受这个二次失效



MSHR存储结构图

6.1 MSHR设计方法

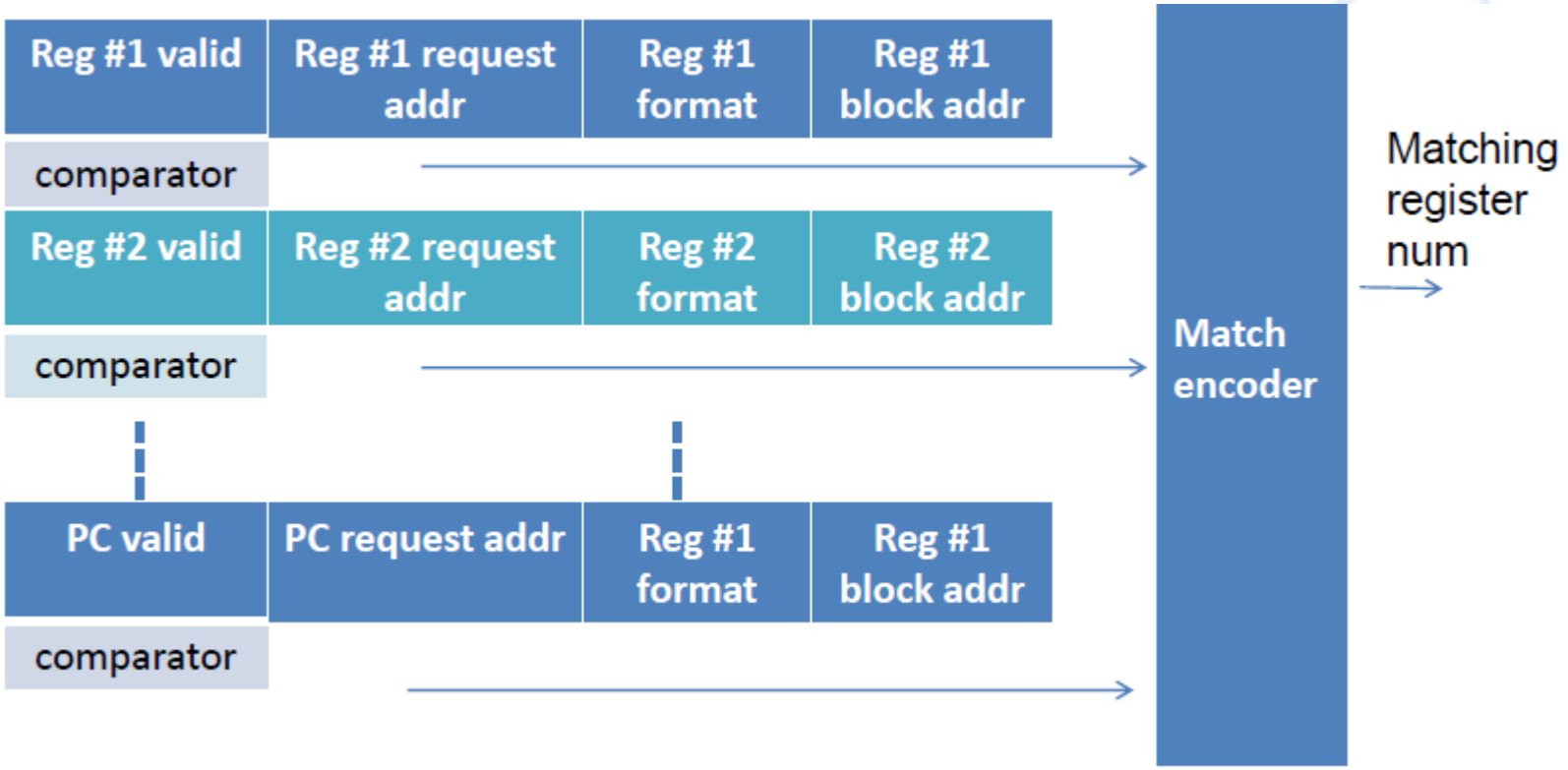
- 第二种设计：Explicitly Addressed MSHRs
  - 在 Implicitly Addressed MSHRs 方案基础上进行修改
    - 在 field 表中，添加 block offset 域
    - words 与 entries 没有一一对应的关系，同一个word的两次失效可以被保存到不同的表项
  - 这个结构能够接受更多的miss on miss失效请求



MSHR存储结构图

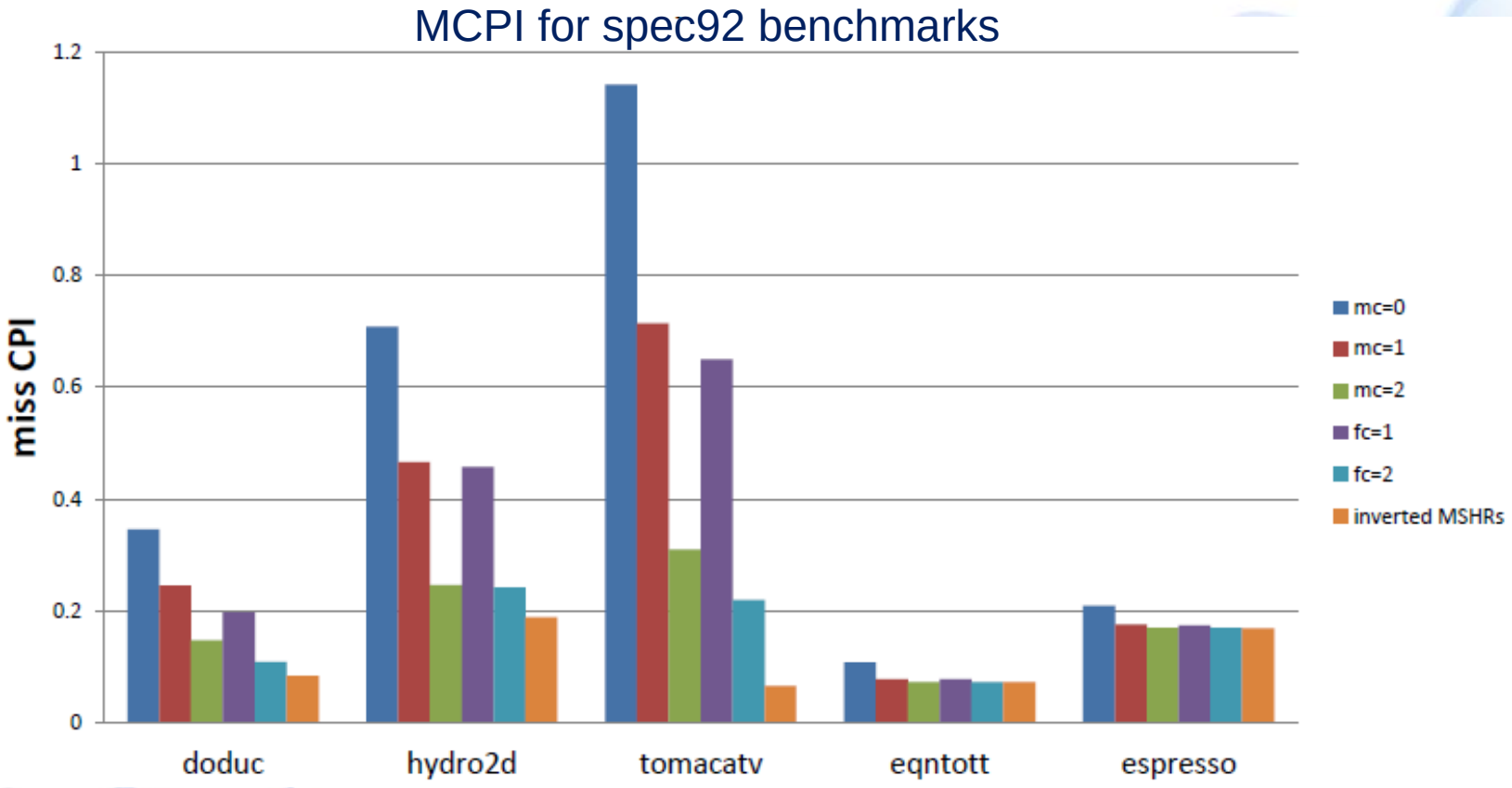
6.1 MSHR设计方法

- 第三种设计：Inverted MSHRs
  - 为每一个目的寄存器提供一个MSHR
    - 整点寄存器、浮点寄存器、PC、write buffer entries等，所以，一个典型的Inverted MSHRs可能包含65~75个MSHR



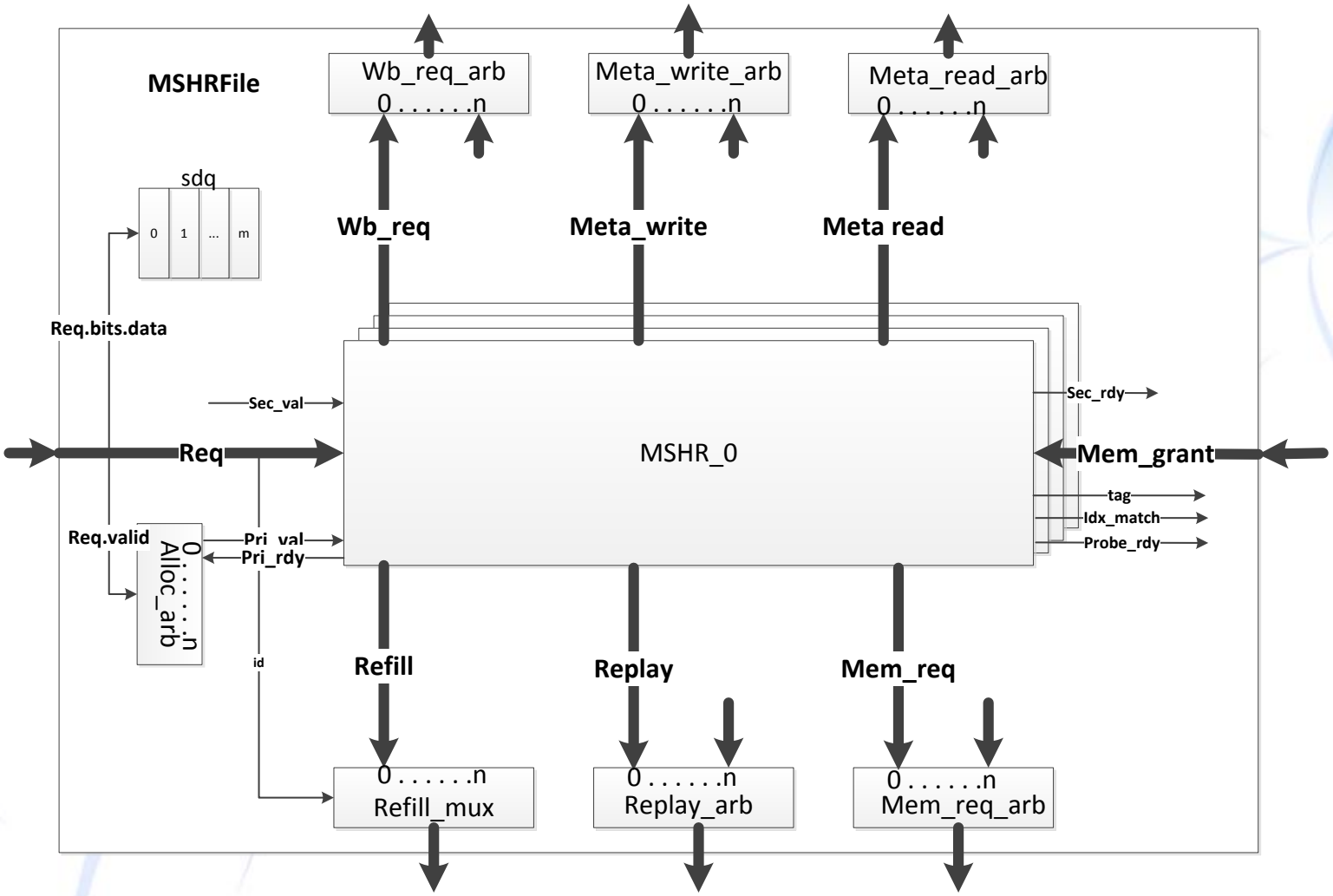
6.1 MSHR设计方法性能比较

- 性能评测

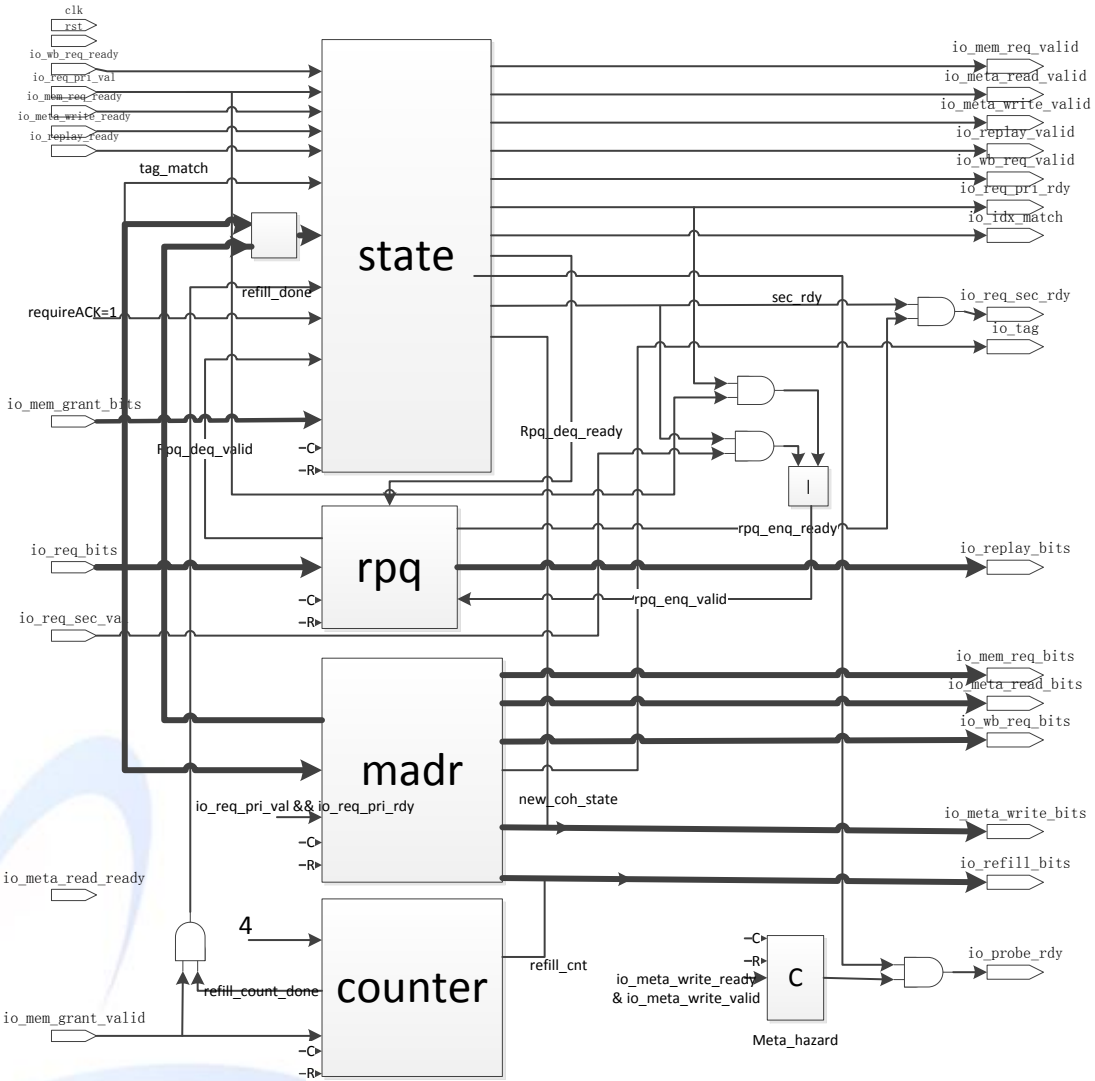


mc: the number of MSHR with one explicitly address
fc: the number of MSHR supporting unlimited number of secondary miss

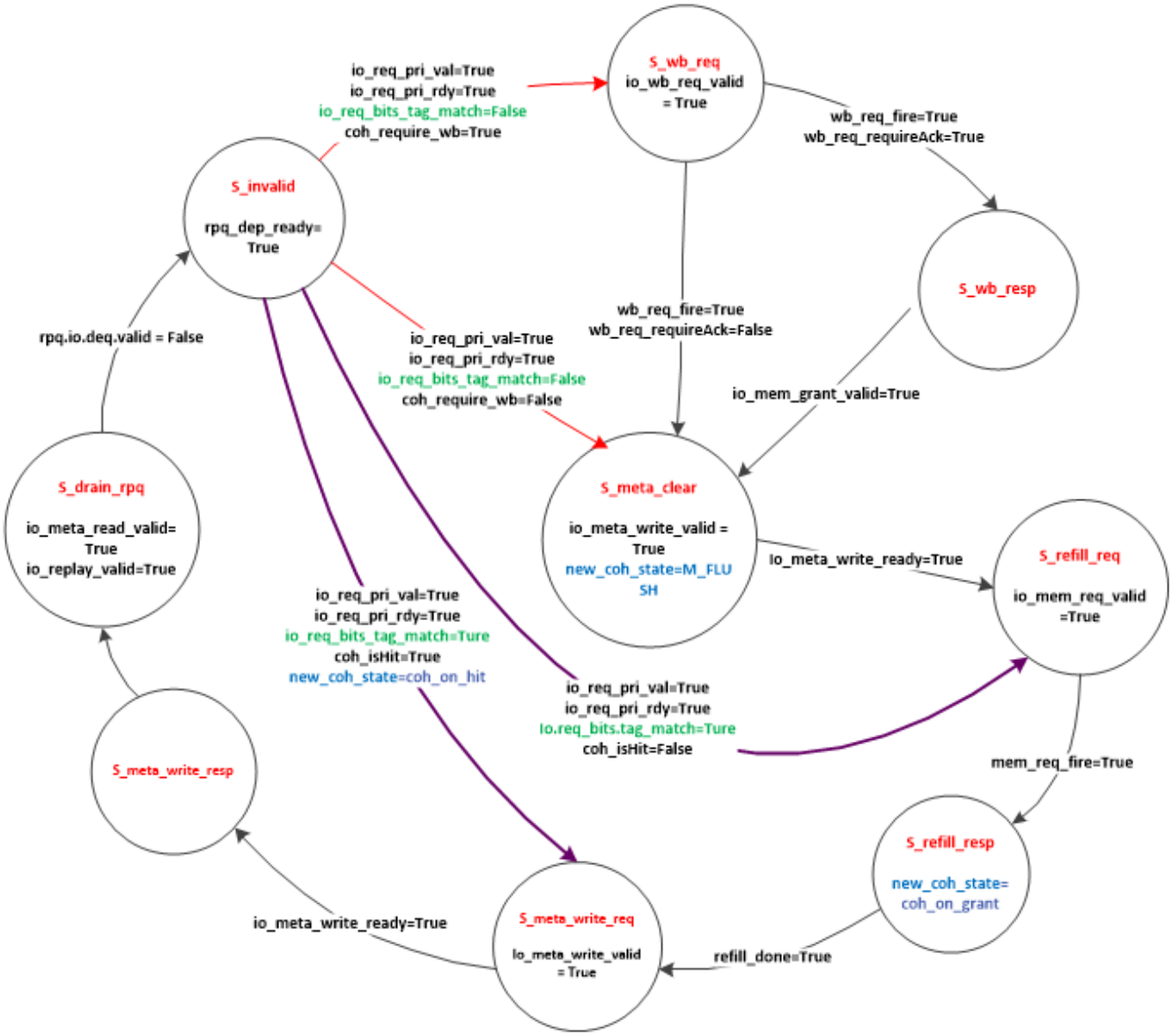
6.2 RISC-V MSHR实现——MSHRFIle



6.2 RISC-V MSHR实现——MSHR



6.2 RISC-V MSHR实现——状态机



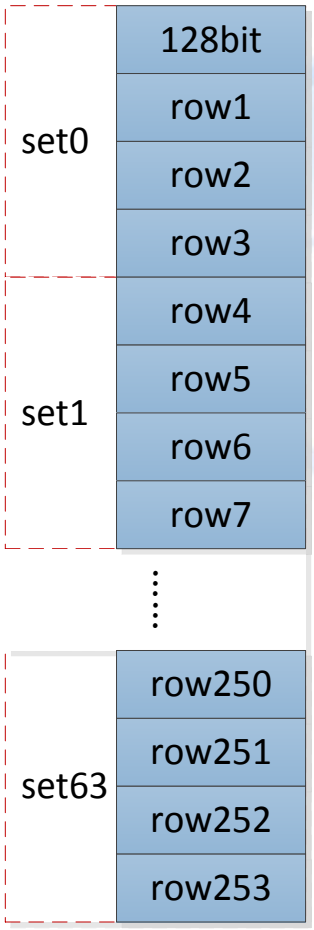
7.1 dataarray

- **模块描述**

包含存储cache数据主体的物理实体，其作用是对该物理实体进行相应的读写操作。

- **物理实体Data SRAM**

功能是保存L1 DCache各路的数据，预定义的L1 Dcache中路数为4，所以包含4个Data SRAM。每一个Data SRAM 结构如图



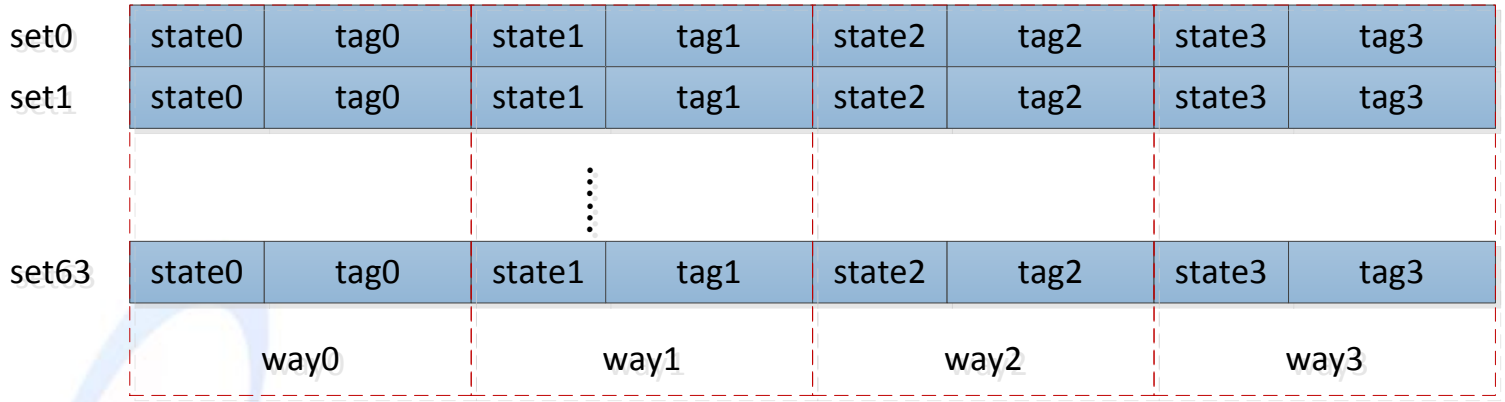
7.2 metaarray

- **模块描述**

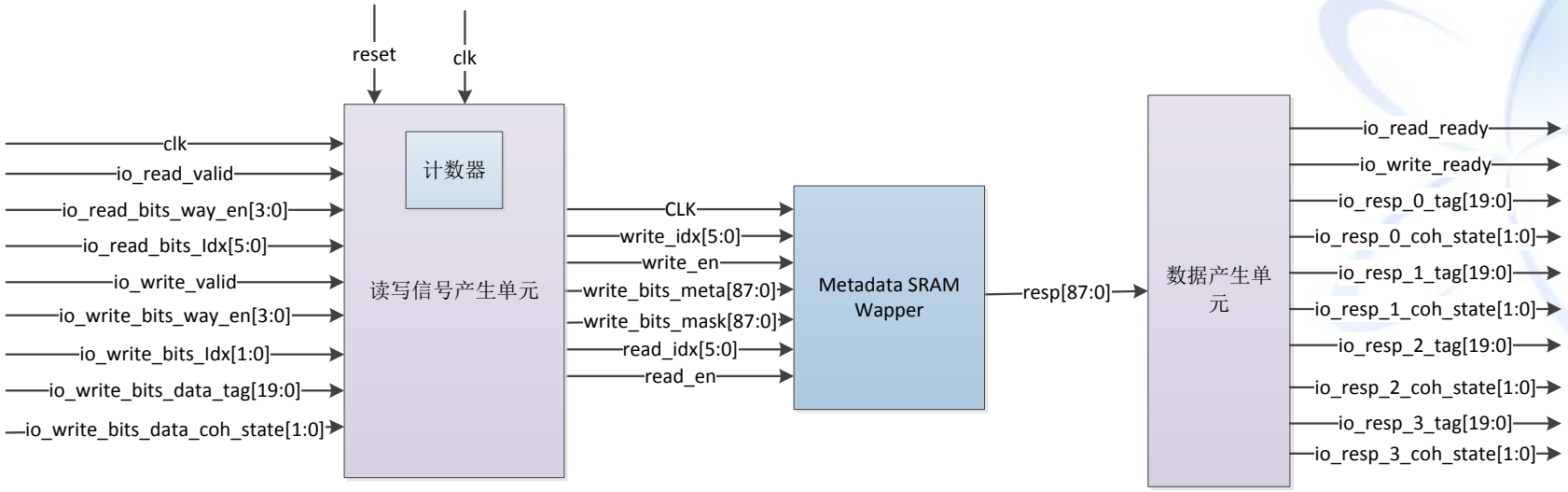
该模块包含存储cache tag域和state域的物理实体，其作用是对该物理实体进行相应的读写操作。

- **物理实体Meta SRAM**

功能是保存L1 Dcache 4路数据所对应的Tag域和state域，Tag域对应的是数据所在物理地址的高20位（即物理页号），state域为2位对应cache块的4种一致性状态。L1 DCache包含1个Metadata SRAM，结构如下：



7.2 metaarray

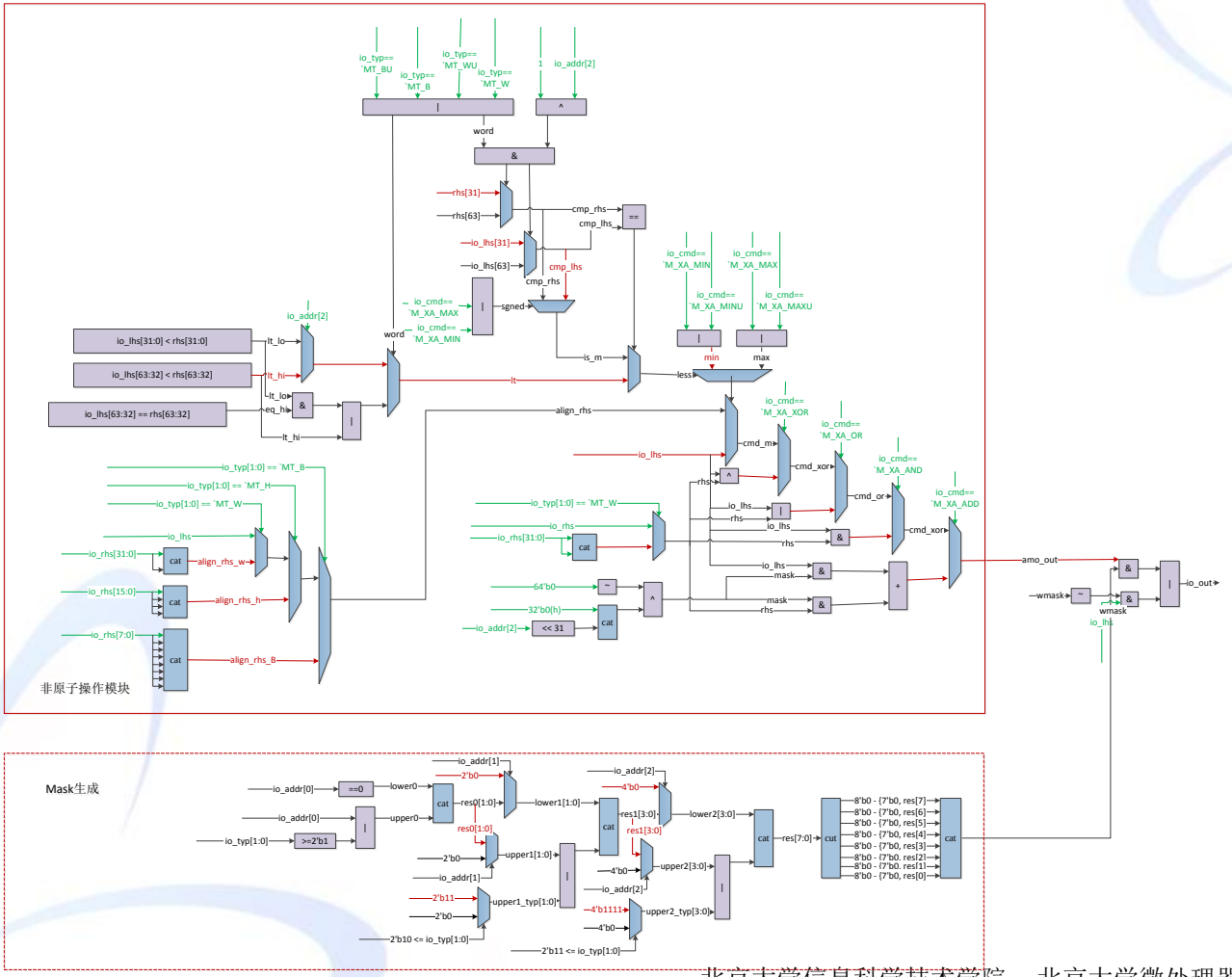


7.3 AMOALU

- **模块描述**
  - AMOALU有两个功能分别是处理原子存储操作指令和普通存储指令。
  - RISC-V提供了fetch-and-op风格的原子存储操作（AMO，Atomic Memory Operation），即将内存中的数据、与寄存器的数据进相应的原子操作，将结果写入到原内存中的地址，该操作执行期间不能被打断。AMOALU模块支持RISC-V提供的所有的AMO指令，包括整数加、逻辑AND、逻辑OR、逻辑XOR和有符号、无符号整数最大值和最小值。
  - 另一方面AMOALU模块根据数据操作类型对写cache体的数据进行相应的合并操作。

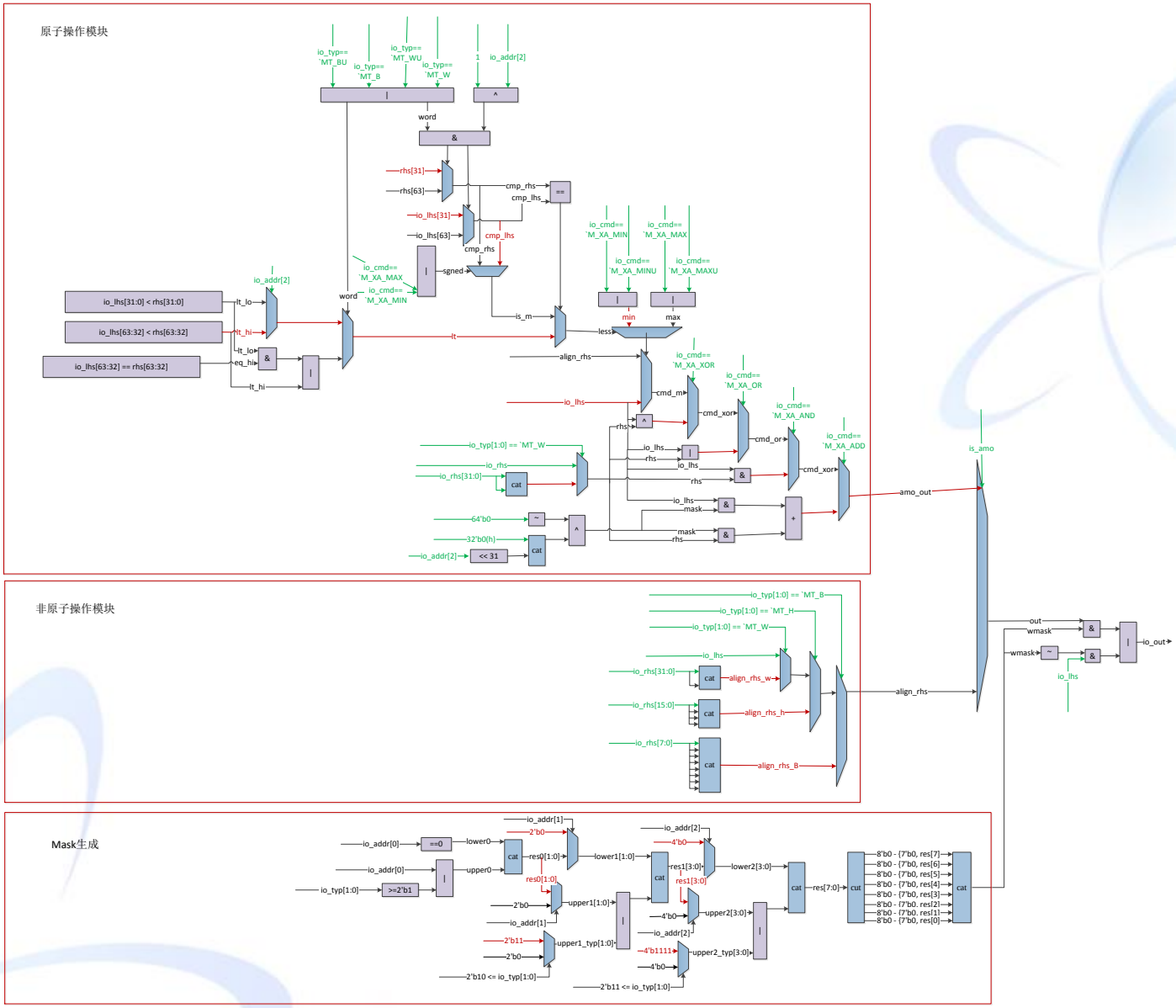
7.3 AMOALU模块实现

生成的Verilog中，未考虑路径长短：大部分指令为非原子操作指令，但非原子操作指令操作数要在最后选出，延长了处理时间。



7.3 AMOALU模块实现

优化后：



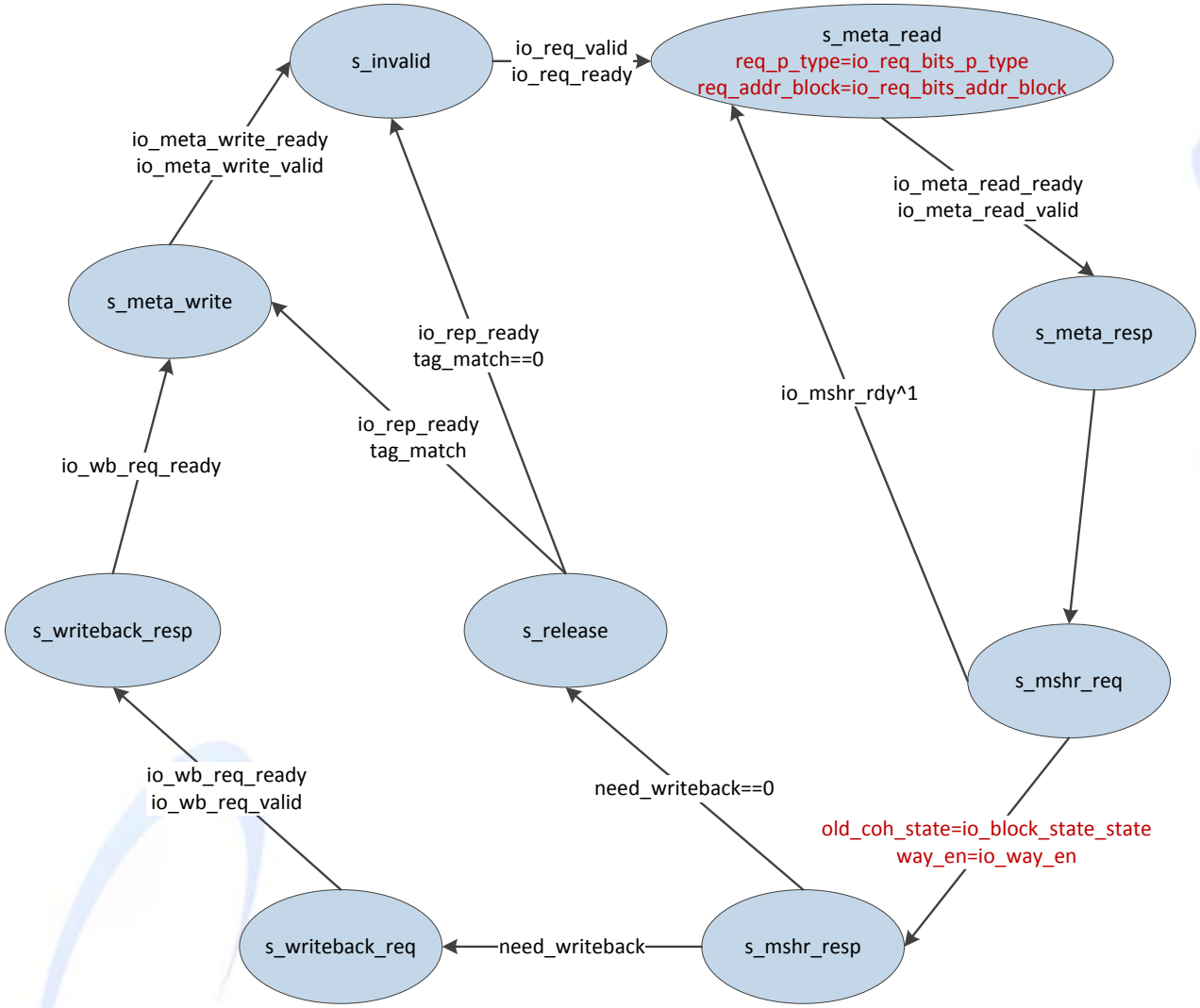
7.4 prober

➢ 模块描述

- ◆ 控制多核Cache一致性，处理memory发来的prober请求，即处理其他cache块修改本cache块一致性信息。比如：
  - ✓ 其它核的Cache对某个数据块进行了写操作，它通过给本地Probe模块发送invalidate信号来清除本地Cache对那个数据块的缓存
  - ✓ 其它核的Cache访问某个数据块发生Cache miss，若它访问的数据块只在本地Cache中有缓存，它会请求本地Probe模块将该数据块写回memory，memory再将数据块发送给请求的那个处理器核。

31

7.4 prober——状态机



prober处理状态转换图

8 总结

- 在很多方面降低了命中时间、失效损失，提高了失效率，提高了Cache性能，比如：
  - 虚拟索引、物理标签
  - PLRU
  - 流水化操作
  - 非阻塞
- 实现的还不完整，比如：
  - 还未添加ECC校验
  - 还未实现预取
- 还可增加其他优化技术，比如：
  - Victim Cache
  - 伪相联
- 其他优化，比如：
  - Cache一致性可以实现Migratory策略

谢谢！