存储系统与高速缓存设计

真正的存储系统

- SRAM：送入一个地址，
  - 异步读（Distruted RAM of Register File）：当周期组合地给出结果
  - 同步读（Block RAM）：下一个时钟上升沿到来后给出数据

  不管如何，送入一个地址，一定在可预测的时机给出数据
- 真正的存储器：DRAM（Dynamic RAM）的不确定性
  - 真正的 DRAM 由于需要等待充电时机和准备数据的延迟，当给出一个地址后，会在随机的时机给出数据，甚至地址都没有收到！
  - 这时必须要有一套握手协议，在读操作是确保两件事情：
    1. 确保读地址可以被存储器接收到
    2. 确保读数据可以被处理器接收到

    在写操作时要确保三件事情：
    1. 确保写地址可以被存储器接收到
    2. 确保写数据可以被存储器接收到
    3. 确保处理器可以得知存储器的写操作完毕

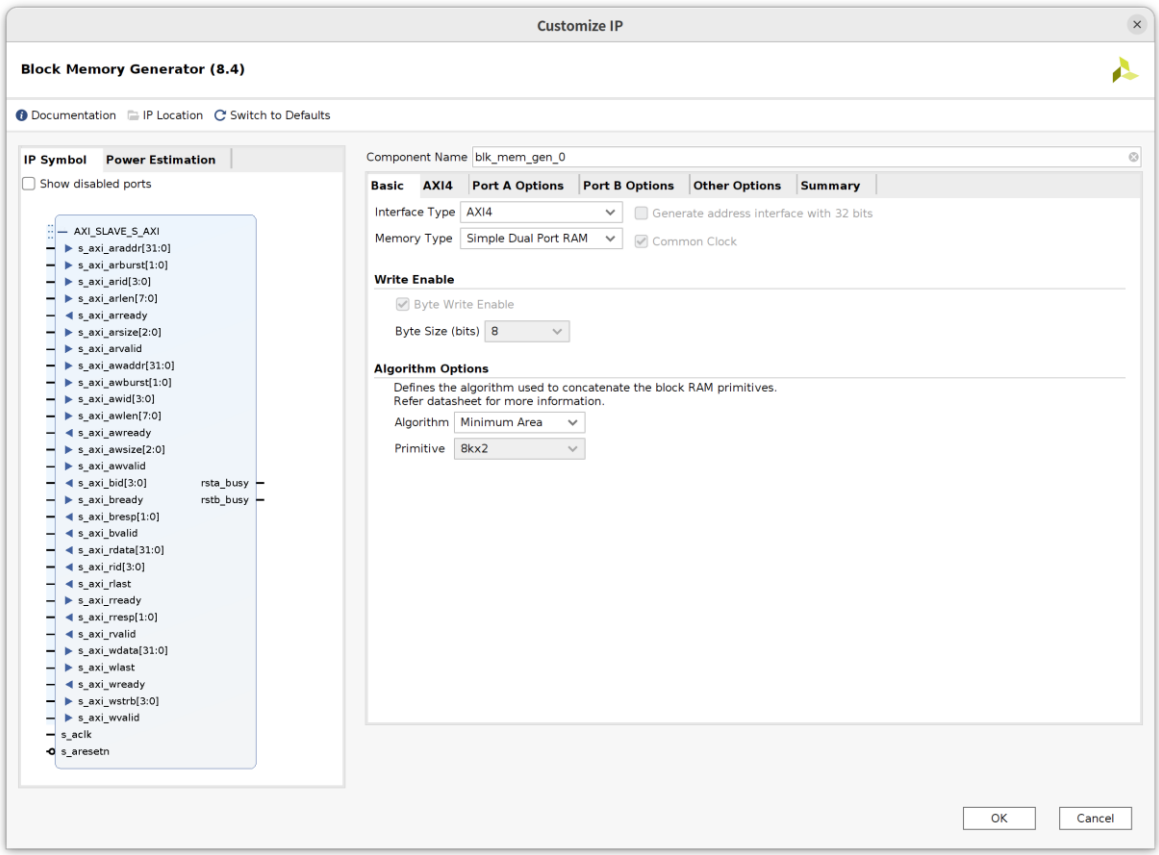
AXI 总线协议

- 同时**并行**地保证了以上五件事情
  - 发起读请求前，需要读地址握手
  - 读请求处理时，需要读数据握手
  - 发起写请求前，需要写地址握手
  - 写请求处理时，需要写地址握手
  - 写请求处理后，需要写完毕握手

  除此之外，AXI 总线协议充分利用了 DRAM 的访存特性，提供了更加丰富、高效的功能：
  - 突发读写：给出首地址、读数据宽度、读数据数量，内存将断续地返回连续的多个数据
  - rlast 信号：让读数据不需要维护计数器
  - wlast 信号：简化了 DRAM 内部的设计
  - 读写并行：增加带宽（注意：同时读写同一位置，返回原先数据和写入数据都是符合

AXI 总线协议约定的，但我们应避免这种情况的发生）

- 获得一个较为真实的存储器：Vivado AXI 风格 Block RAM



  - 注意：这个存储器在仿真时一定要给几个周期的 reset

- AXI 信号省流理解：
  - 类型：
    - a 开头：地址相关
    - r 开头：读数据相关
    - w 开头：写数据相关
    - b 开头：写确认相关
  - 信号名：
    - valid：发起的一方请求信号。地址一定是处理器，写数据是处理器，读数据和写回确认是存储器
    - ready：接收的一方确认信号。和 valid 恰好相反
    - addr：地址
    - data：数据
    - last：发起一方发送的最后一个数据标识信号。读是存储器，写是处理器
    - len：本次突发读写的数据个数

        - size：本次突发读写的数据宽度（注意这个信号的值代表了真实宽度的对数）
        - wstrb：写掩码，标识本次写请求写一个字中哪些字节。
        - burst：突发传输模式（具体编码参考文档或参考书），01 为顺序递增传输
        - 其他信号：没啥用，看一下手册就行
- AXI 总线协议时序和注意事项
    - 读请求
        - 地址握手，同时处理器给出突发传输模式、传输长度和单次传输宽度。一旦存储器给出 ready 信号，证明本次传输的所有设置都已经被接受
        - 不停地进行读数据握手，每次 AXI 总线给出 valid，都需要对应给出一个 ready，如此存储器才会给出下一个数据。特别注意：**AXI 总线并不保证传输过程中，在倒数第二个数据被接收和最后一个数据到来之间 rlast 不被提前置位，因此只有当 ready 和 last 同时到来时，本次传输才算结束！**
        - rlast 对应的数据握手完毕后，本次访存结束
    - 写请求
        - 地址握手，同时处理器给出突发传输模式、传输长度和单次传输宽度。
        - 不停进行写数据握手，每次 AXI 总线给出 ready，都代表处理器已经接收到了这个写数据（注意，“接收到”和“正式写入”时两个截然不同的概念）。最后一个写数据时，处理器必须给出 wlast 信号
        - 全部正式写入完毕后，存储器会给出 bvalid 信号，这时处理器必须给出一个 bready 信号进行握手，本次访存才算结束（否则存储器会维持住当前状态，不接受任何新请求）
    - 特别注意：**存储器并不能保证在非处理事务时总线上信号全部置低。这些随机信号很有可能给处理器带来诡异的影响，必须要格外小心！**
    - 虽然 AXI 总线协议可以支持高达 128 位的单次传输，但这仅限于仿真，真实 FPGA 的单次传输以 32 位为宜。同时，单次传输宽度过大，存储器准备时间也会相应变长

高速缓存的基本原理

高速缓存的意义

- 实际的 AXI 总线一次请求需要将近 50 个周期处理
- 程序局部性使得时空概念上的最近数据还会被继续使用
- 高速缓存的访问速度很快，使用 BRAM 命中时为两个周期
- 通过在高速缓存中放置高频使用的数据，极大提高了性能
- 有高速缓存和没有高速缓存的性能差距至少为 10-20 倍

高速缓存的实现策略

- 高速缓存可以保存一部分内存中的内容，但必然无法保存内存中的全部内容
- 我们在高速缓存中可以用块来组织数据，一个块包含了多个字节和字，构成了 Cache 存储器的宽度（也就是说 Cache 存储器的宽度不再是 32 位了，为什么要用块呢？）
- 块地址就是数据地址的较高位，如果一个字节的地址为 addr，若一个块有 n 个字节，那么这个字节所处的块地址就是 addr / n，对应的块内偏移就是 addr % n
- 使用地址分割法来实现一个多对少的映射：用较低的几位来定义高速缓存的块内偏移，用中间的几位来定义高速缓存的块地址，最高几位地址没有索引的作用，用作标签
- 高速缓存使用块地址索引出数据，并索引出对应的标签。
  - 如果标签和本次访存的地址高位相同，那么高速缓存命中，使用较低几位选择出块内具体数据，送出即可。
  - 如果标签和本次访存的地址高位不同，那么高速缓存缺失，开始从主存中取出本次访存的地址数据，存入高速缓存后更新标签，并送出数据。

- LRU（最近最少使用）算法：替换掉近期最少使用的路的对应组
  - 实现方法：为每一个 Cache 行都维护一个计数器，一旦某次访存没有使用这个路，就让计数器自增，每次缺失时，替换掉计数器值最大的那一路。——浪费大量空间
  - 高效实现方法：用寄存器记录路编号，并记录相对的使用情况

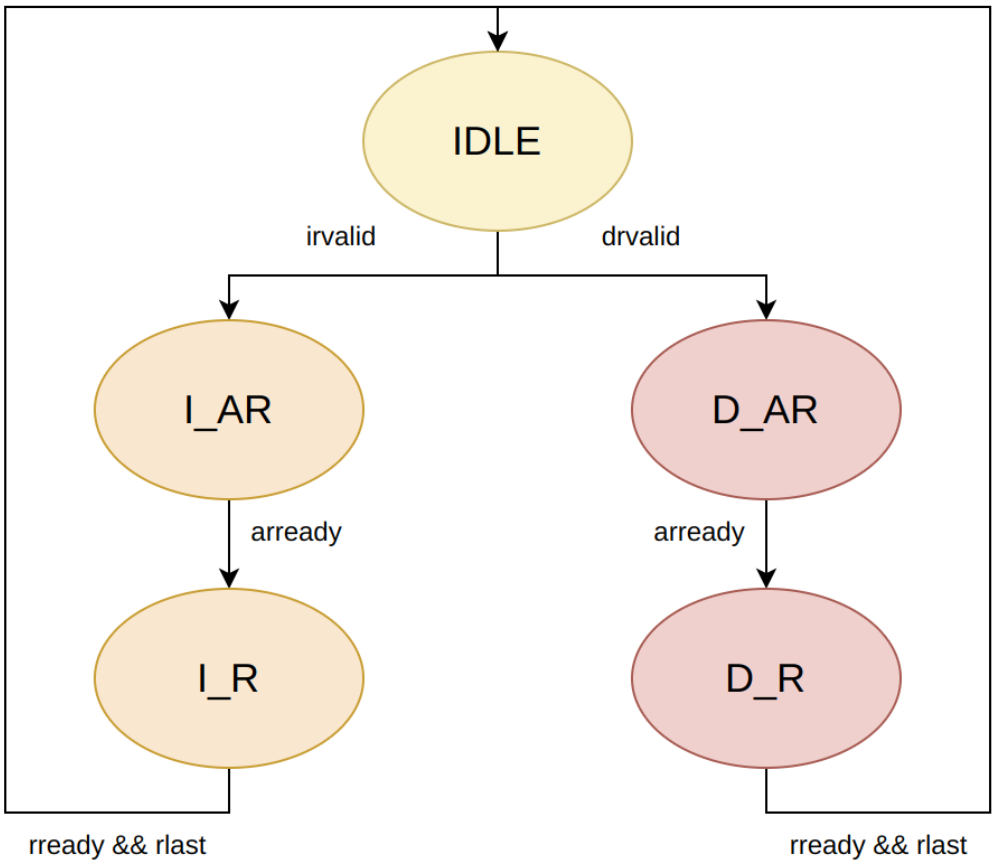
处理器存储系统设计任务

- 指令高速缓存 ICache：难度适中
- 数据高速缓存 DCache：难度较高
- 转接桥 CrossBar（难度较低）或二级高速缓存 MixedCache（难度高）
- 栅障指令、cache 操作指令的实现

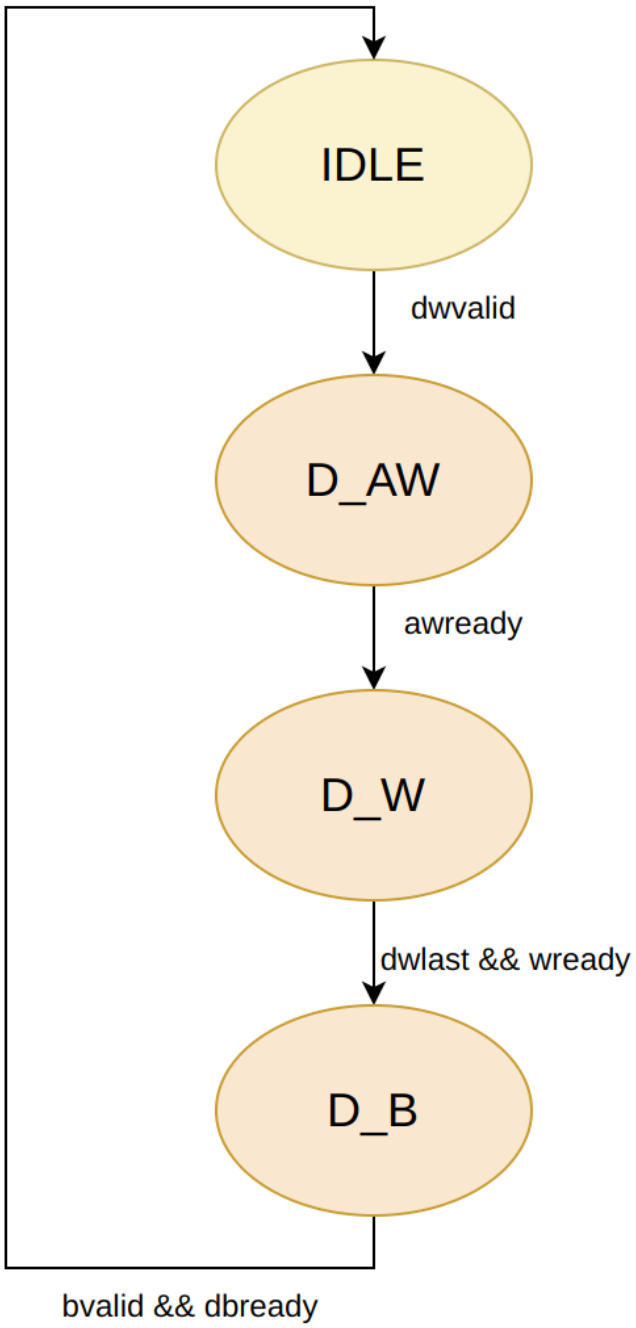
转接桥设计思路

- 主要任务：
  - 仲裁两个高速缓存的读请求，同时要保证读写并行
  - 用状态机将 SRAM+valid-ready 协议转化为 AXI 总线协议与主存通信，确保 AXI 总线协议对高速缓存不可见
- 时序和状态机设计：

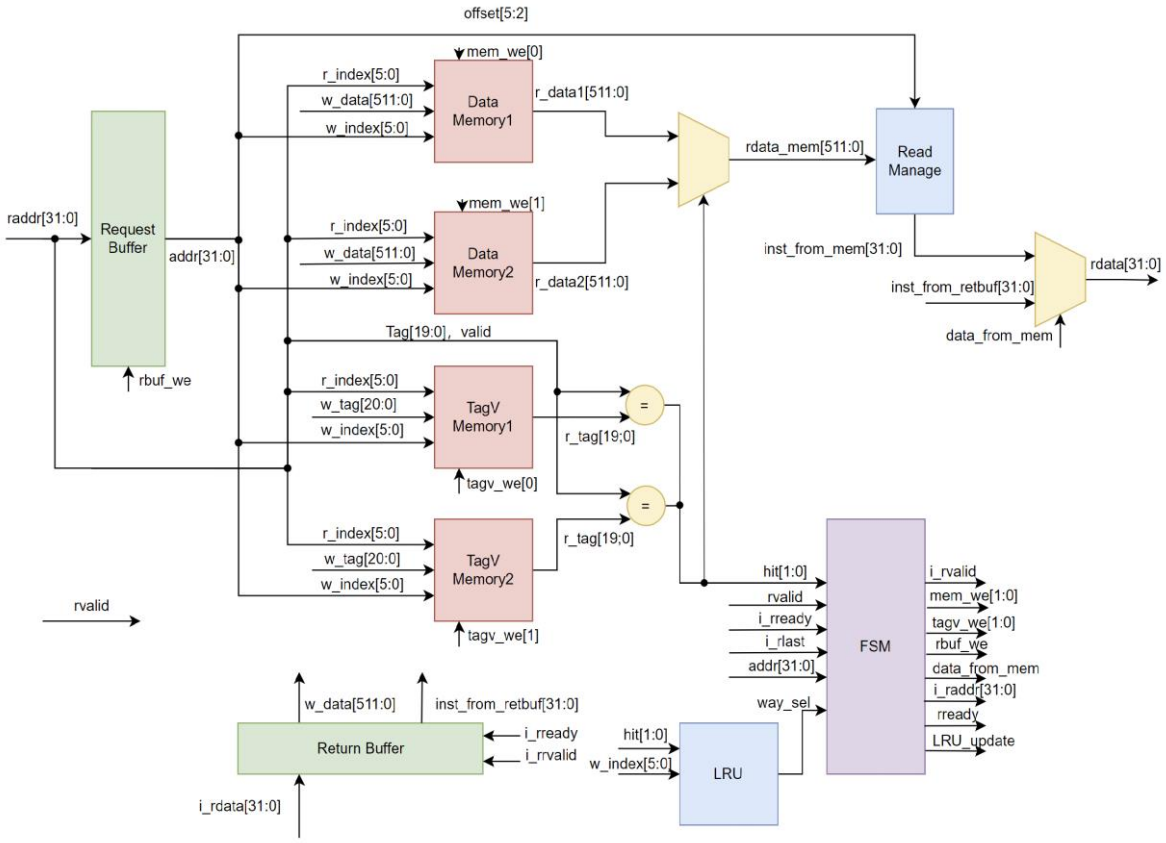
- 读状态机：


IDLE
irvalid
drvalid
I_AR
D_AR
arready
arready
I_R
D_R
rready && rlast
rready && rlast

- 写状态机：

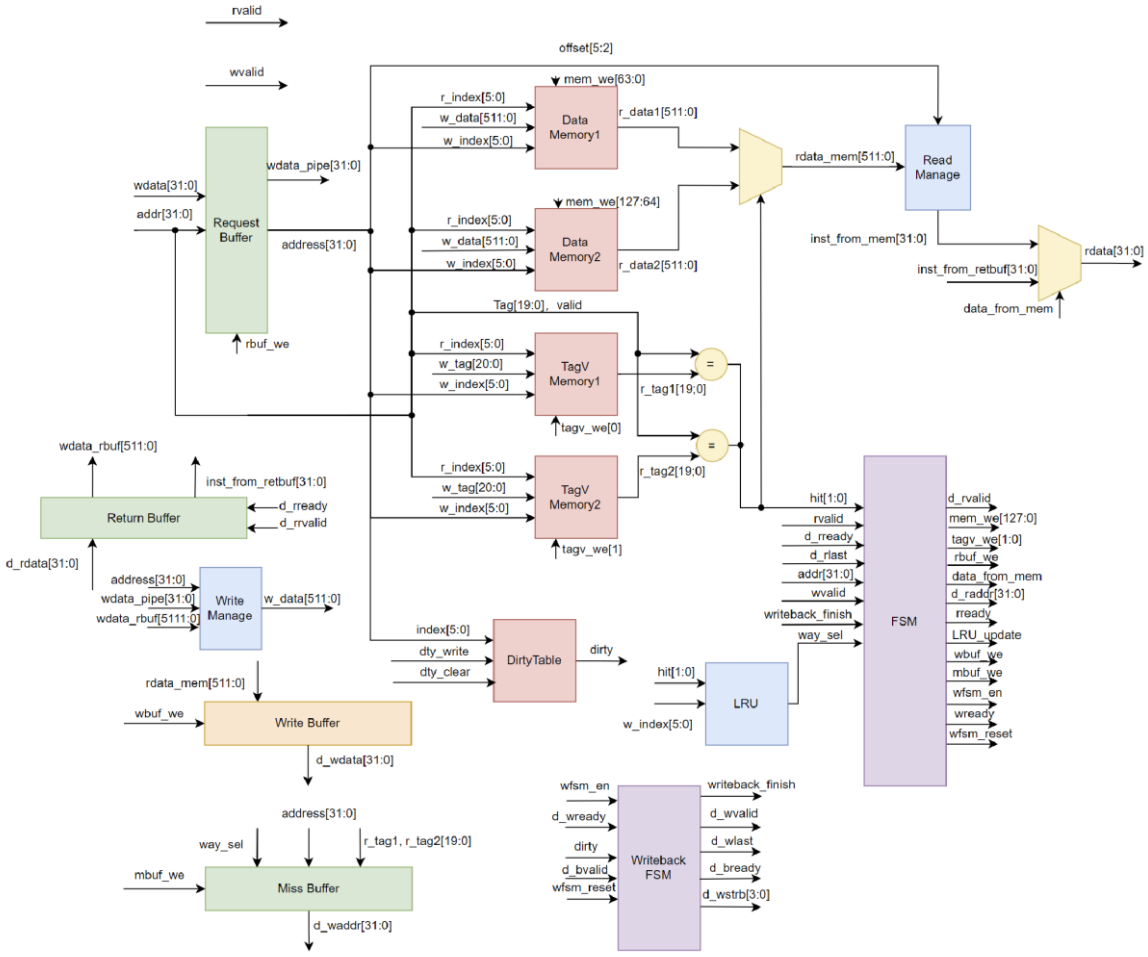

IDLE
dwvalid
D_AW
awready
D_W
dwlast && wready
D_B
bvalid && dbready

ICache 的设计思路



- 使用状态机控制，借助 BRAM 两周期流水访存（不要使用 DRAM：时序太差）
- 2 路组相连为宜：经过不完全统计，2 路组相连的命中率已经很高，替换策略实现较为简单，时序也较好
- 需要支持非可缓存读
- 实测：单行 64 字节命中率更高，行数不宜过多——过多行会导致 tag 更多
- 如果希望在决赛启动 linux，那么高 20 位必须作为标签（虚 index 实 tag）
- 通过 SRAM+单 valid-ready 握手与转接桥沟通（看不到 AXI 总线协议）
- 栅障指令的实现：无效掉 TagV 表中的所有行；
- Cache 操作指令的实现：根据指令要求对应在状态机中实现即可

DCache 的设计思路



- 在读和配置方面与 ICache 是相同的
- 推荐使用写回写分配，有效提高性能
- 需要支持非可缓存读写和原子访存（和 CSR 配合）
- 栅障指令的实现：无效掉 TagV 表中的所有行，写回脏行；
- Cache 操作指令的实现：根据指令要求对应在状态机中实现即可

高速缓存和流水线的适配

Step 1 加入指令高速缓存

- 切割 IF 流水级，在 IF1 阶段对 ICache 发起访存，在 IF2 阶段获得数据，并判断是否缺失，生成流水线控制信号
- 两种方案：
  - 停顿全流水线：只有 ICache 时是神，但是加入 DCache 就可能出现互锁
  - 停顿 ID 之前的流水线，同时令 IF2-ID 段间寄存器一直 flush，这种情况下一定要注意

这个 flush 的使用**（当 IF2-ID 寄存器因为其他原因停顿时，这个 flush 不应该生效）**

- 关注段间寄存器：
  - 上述的 IF2-ID 寄存器问题
  - IF1-IF2 寄存器应该优先后续的 flush（例如跳转 flush）
  - PC 应该优先 flush
- 选择第二种方案时，用一个 flush_reg 来保存缺失处理时到来的 flush 信号。如果缺失处理完毕后发现发生过 flush，那么直接送出一条 nop 或者不送出握手响应信号
  - 为什么不能让访问主存直接“中道崩殂”？因为 AXI 总线协议必须保证完整的进行一次访存，主存才可以接受下一次访存

Step 2 加入数据高速缓存

- 在 EX 阶段，直接将算出的地址接入数据高速缓存的 BRAM，抢出一个周期，就可以不必再切割流水线（数据高速缓存的所有入口均为寄存器）
- 数据高速缓存必须要停顿整个流水线（包括其后面的 MEM-WB 段间寄存器）
- 同样，如果在 wb 段传来了 flush 信号（例如：异常、ecall 等），那么直接不送出握手响应信号（相当于流水线没有发起这次访存）
- 不可以 flush MEM-WB 寄存器！因为可能这个寄存器中有后续 EX 段指令所需的前递数据！

Step 3 加入转接桥

- 能有写请求的只能是 dcache，读请求可能是两个 cache
- 这一部分要经过大量调试，因为 AXI 总线的信号很不稳定，需要能够应对各种随机信号情况

存储系统设计过程中的常见问题

- **load 出 0 了！出错了！**

  这种问题在测试中不好追踪，特别是对于写回的 DCache，很有可能是写回的地址、写掩码不对导致写回了 0。我们去年出现的问题是在 uncache 访问时写掩码出错了。

- **流水线莫名其妙卡死了！**

  特别注意握手暂停的原则：valid=1 且 ready=0。如果 valid=0，那么不可以暂停。这种问题很有可能发生在程序刚开始执行时流水线空的时候，或者刚刚被 flush 后

- **丢指令了！**

  这个问题一般是由不正确的 flush 和 bubble 关系造成的。我们一般 flush 优先级会高，但是在上文中提到的案例中，bubble 应该优先级更高，所以这个时候就要手动实现 bubble 和 flush 的关系了。

- **寄存器堆写回了莫名其妙的数据**

  很有可能是 wb 段 flush 时恰好 dcache 缺失，但是 dcache 中没有处理 flush 的方法，导致送出了一条错误的数据，恰好和一个新的读请求握手成功了。