

2023计算机组成原理综合实验——高速缓存

前言

大家好，在本年度的综合实验中，我们提供给大家多个处理器拓展的选择，但毫无疑问，高速缓存的设计是最为困难的，和其他几个拓展难度完全不对等。鉴于这种情况，我们将为大家提供一个代码框架，其中包括：

- 指令高速缓存（需要补全）
- 数据高速缓存（需要补全）
- AXI仲裁—转接桥（无需补全）

大家只要**补全了上述代码框架，将两个高速缓存接入流水线，并通过了所有的测试**，就可以完成综合实验的线下检查部分了。此外，我们只需要大家**梳理填充代码**，并阅读代码框架，**在报告中回答有关于高速缓存的几个问题**，就可以完成综合实验的实验报告了。

在这一部分，我们希望通过实验框架，能充分地让大家**对高速缓存的设计进行思考**，而并不是用代码量来“压榨”大家。代码框架中需要书写的内容和其他几个拓展相比并不多，但需要大家充分思考才能正确地设计出高速缓存，而在阅读本文档时，也请**多注意需要补全的模块**，不要被文档的篇幅吓到。框架中涉及到填写的部分大部分已经在实验拓展课中讲过，大家可以参考拓展课的PPT和讲义完成本次实验。

希望选择这个拓展的各位同学能够加油！如果对代码框架有疑惑，可以来询问马子睿助教。

马子睿

2023年5月21日

实验内容

- 补全指令高速缓存（ICache）和数据高速缓存（DCache）的代码框架，并能够通过随机地址访存的测试
- 将ICache和DCache接入自己的流水线，处理流水线停顿和冒险，并成功运行快速排序程序

代码框架概述

对于每一份代码框架，其整体都由以下几个部分构成：

- 输入输出接口声明
- 内部变量声明
- 用/* */型注释分割的若干组件

在补全过程中，助教已经将需要补全或修改的地方使用**TODO**标识出来了。只要正确补全了这些内容，就可以通过随机地址访存测试。测试的方法会在后文说明。

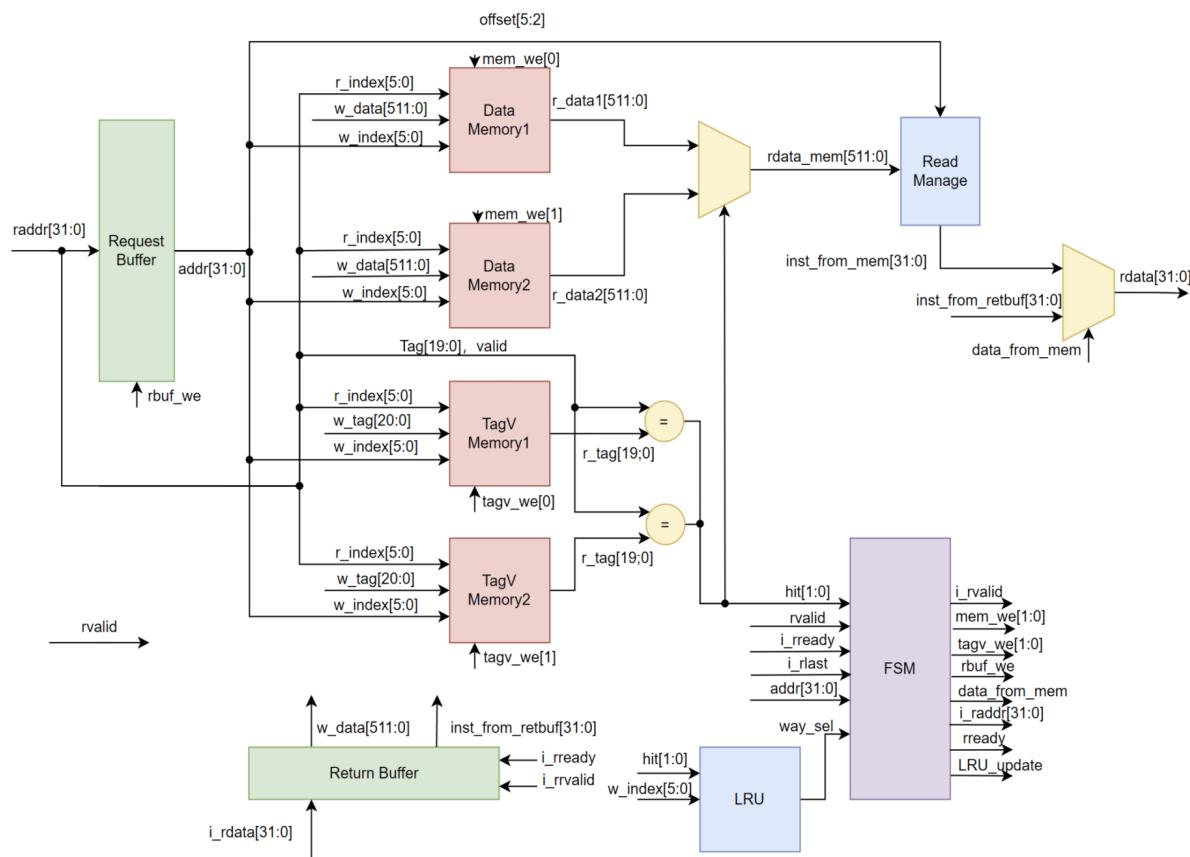
在代码框架中，我们不需要大家完全掌握**AXI总线协议**的相关内容，这一部分的绝大部分工作已经由助教完成了。

指令高速缓存代码框架说明

在本次实验中，我们要求大家实现**2路组相连**的指令高速缓存，并能够通过参数来调节Cache的行宽度和组数。代码框架中已经能够对这两个指标进行参数化，故大家在补全代码的时候也一定要使用对应的参数进行代码编写。

设计说明

指令高速缓存的设计如图所示：



上图的高速缓存是一种可能的参数配置，其中的线网名称可能与框架中有所不同，请以框架为准。

其中组件及其功能如下：

- **Request Buffer**：请求缓存寄存器，用于锁定本次读请求的请求地址。
 - 由于所有的存储器都是BRAM，因此需要一个周期来读数据，恰好此时就将请求地址写入请求缓存。
 - 在缺失时，请求缓存将会发挥重要作用。这个缓存中记录了本次缺失访存的地址，因此被用作对主存发起访问的主体。其写使能由状态机控制，只有当一次访存全部完成后，才会让请求缓存写入新的请求。
- **Return Buffer**：返回数据缓存移位寄存器，用于接受并拼接缺失处理时逐个返回的数据。
 - 一旦高速缓存缺失，就需要向主存请求一整行（可能包括多个字）的数据，这时就需要用返回数据缓存来拼接按照地址从低到高顺序返回的多个字。
 - 当所有的字都返回后，状态机会控制返回数据缓存对Cache存储器发起写操作。同时，为了提高性能，Cache在缺失处理后会直接将这个缓存中的某个字送给流水线，而无需重新读取Cache存储器。这个字就是流水线向存储器请求的那个字。
- **Data Memory**：高速缓存的核心存储器，存储了主存中的一部分内容。

- 在高速缓存中，我们使用地址的较低位来索引一个Cache行中的每个字节，用地址的中间几位来索引Cache行，而用地址的高位来作为标签（具体的划分和高速缓存配置有关）。数据存储器的行可能很长，每一行我们都称之为一个“Cache行”，由若干字构成。而我们正是用地址的中间几位作为这个存储器的地址，来读出一整行的数据。
 - 这个存储器使用BRAM实现，框架中并没有采用IP核，而是直接手写了一个BRAM。这个BRAM可以被Vivado综合为真正的BRAM。
 - 这个存储器的读和写地址来源并不相同：读操作的地址直接由流水线给出，而写操作是由Request Buffer给出。之所以如此，就是利用了BRAM的特性，提前一个周期给出地址开始读，后一个周期送出数据，从而不需要流水线停顿（具体的理解请参考实验拓展课的对应PPT）
- **TagV Memory**：标签—有效位表，存储了数据存储器对应行中数据对应的地址高位和有效位
 - 这个表也使用通过与访存地址进行比较，可以判断访存是否命中。这个表通过地址的中间几位来索引，其中存储了每一个Cache行对应的地址高位。
 - 在刚刚上电的时候，数据存储器中每一行都不是有效数据，因此我们还需要再标签表中多存储一位，一旦这个Cache行被更新，这一位就写1，用以表示该Cache行的数据是从主存取出来的，而不是上电后Cache存储器中的随机数据。
 - 举一个例子：假如一次访存的中间几位是2，用这个2作为TagV表的地址，就可以读出高速缓存存储器中第2行的数据对应的地址标签是多少，同时也可以读出和这个数据是不是有效的。
- **Hit**：一个逻辑电路，用以判断本次访存是否命中。（需要补全）
 - 访存的命中条件是：当前访存的地址高位与标签表读出的标签相同，并且有效位为1，那么就可以判断为高速缓存命中
 - 请大家思考：这个比较应该在什么时候进行，应该用流水线给出的地址还是用请求缓存中的地址与读出的标签比较呢？
- **Read Control**：读控制，用以控制Cache送到流水线的数据。（需要补全）
 - 不管从存储器中读，还是从返回数据缓存中读，读出来的数据一定是一整行，由很多字构成，但我们只需要其中的一个字，具体是哪一个字呢？我们需要使用本次访存的地址来选择。
 - 什么时候该送出从存储器中读出的数据，什么时候应该送出从返回数据缓存中读出的数据？这应该由状态机发信号来进行控制。
- **LRU**：最近最少使用替换模块，用以控制Cache进行替换。（需要补全）
 - 这个模块需要存储Cache的最近最少使用信息，从而在Cache缺失的时候提供被替换的路号，并替换掉这一路的对应行。
 - 这个模块也需要更新，同样需要状态机发出对应信号进行更新。
- **Main FSM**：主状态机，控制整个Cache的行为。（需要补全）
 - 这是一个两段式状态机，通过组合电路给出相应信号。具体对应的状态转换请大家阅读代码进行理解（这一部分需要大家在报告中回答一些问题）
 - 在代码补全阶段，大家无需修改状态转换，只需要在对应的状态发出控制上述几个模块的信号即能通过随机访存测试。但接入流水线时，大家可能需要对状态机的状态进行添加，从而解决一些流水线的冲突冒险。

除此之外，高速缓存中已经集成了两个计数器，用以计算命中率，大家无需对这两个计数器进行修改。

输入输出信号解读

在代码框架中，助教已经将输入输出信号做好了分类。这些信号可以分为两大类：

- **面向流水线**：高速缓存与普通的SRAM不同，由于其缺失处理的不确定性，故而需要进行握手。当流水线发起一次读请求时，**不仅需要给出地址，还需要给出一个rvalid**。当cache给出数据时，需要在给出数据的同时给出一个rready，从而能够让流水线清楚地知道哪个周期递送出的数据是有效的。
- **面向主存**：也就是面向仲裁—转接桥。**这些信号都由i开头**，也使用了valid-ready握手协议，只不过在面向流水线时，流水线为发起人，Cache为响应人；在面向主存时，Cache为发起人，主存为响应人。除此以外，Cache和主存的通信还需要几个信号：
 - **i_rsize**：由Cache发出。AXI总线可以分多个周期传递若干顺序的数据，这个信号就标识了每个数据的宽度。假如 $i_rsize=k$ ，那么每次传输就有 2^k 个字节，也就是 8×2^k 个位。在本次实验中，我们每次向主存访问的数据宽度都是32为，因此始终将这个值置为2。
 - **i_rlen**：由Cache发出，规定了本次访存连续访问的数据个数，**返回数据的个数=i_rlen+1**。这个信号应该随Cache行的大小变化而变化，因为**一次访存事务必须返回整行的数据**，而单次返回只有32位，所以需要计算出这个值的大小，以满足取数据的需要。
 - 举个例子：假如一行由512位构成，现在规定单次返回32位（ $i_rsize=2$ ），那么需要把 i_rlen 置为 $512/32-1=15$ ，才能保证一次访存事务填满一整行存储器。
 - **i_rlast**：由主存发出，通知Cache这是一次访存事务中返回的最后一个数据。这个信号必须与 i_rready 同时到达才算有效，否则必须忽视这个信号。

需要完成的工作

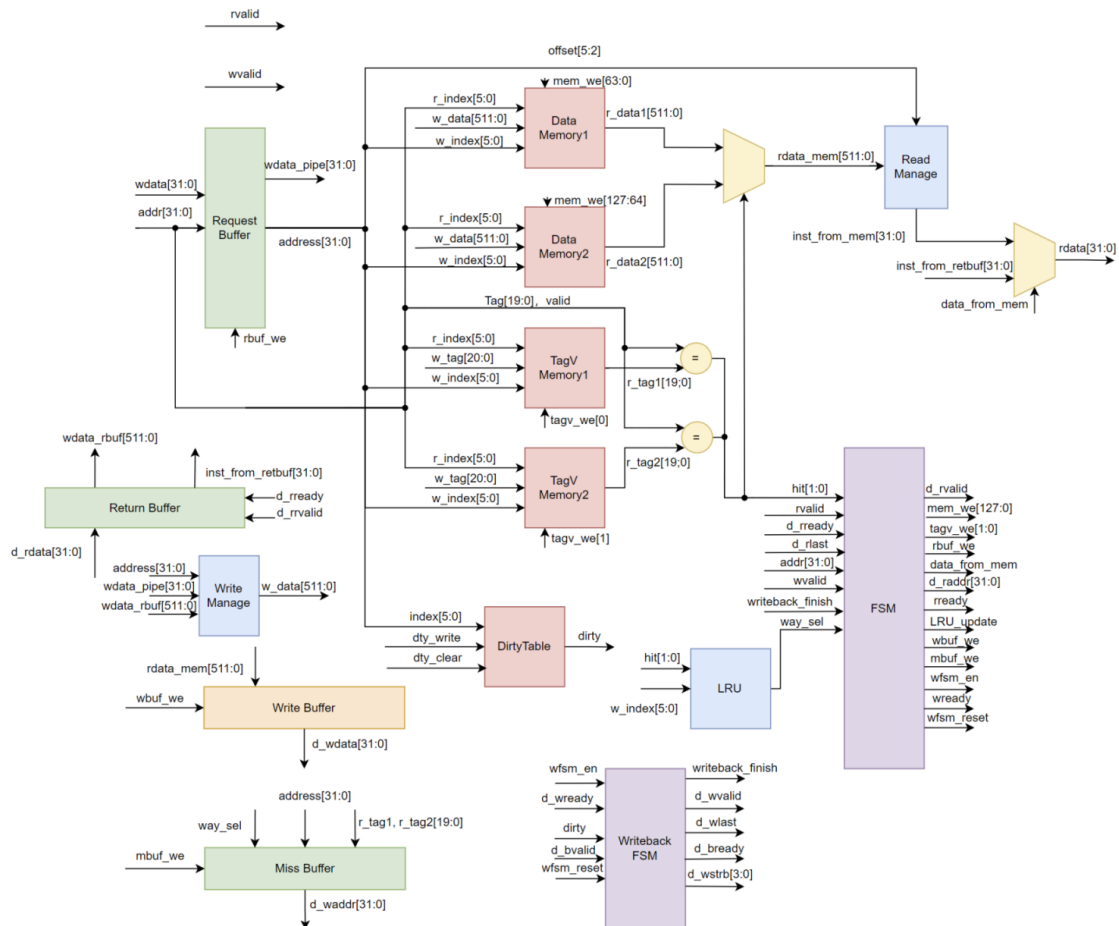
- 补全ICache的命中逻辑
- 设计选择逻辑，通过访存地址和状态机来选择出正确的读数据呈递给流水线
- 设计LRU替换算法
- 补全主状态机（这一部分主要是生成上面两个设计中需要用到的信号）

数据高速缓存代码框架说明

在本次实验中，数据高速缓存的所有配置和指令高速缓存完全一致，只是加入了写模块，并支持**写回写分配**策略。需要大家注意的是，**数据高速缓存目前只支持整字读写**，虽然助教给出了支持非整字读写的方法，但这一部分不需要大家进行实现。

设计说明

数据高速缓存的设计如图所示：



数据高速缓存大部分内容与指令高速缓存一致，主要的不同还是在于写模块和写回模块。与指令高速缓存相同的部分这里就不再赘述了

- **Data Memory**：数据高速缓存的存储器和指令高速缓存略有不同，这个存储器支持“字节写使能”，也就是一次写操作不一定要写整数数据，而是可以精确到写某个字节。如果一行有512位，那么就有64个字节，那么这个存储器的写使能就有64位宽，每一位标识了这一行中的对应字节是否要写。对于写入的数据来说，只有写使能对应位为1的字节才会被写入存储器。
- **Write Manage**：处理流水线发来的写数据
 - 流水线一次只会发来32位数据，但cache的一行一定比32位宽，因此需要根据地址对这个数据进行适当移位，使其对到Cache行的相应为止
 - 同时，由于数据存储器只有一个写口，因此**流水线写**和**缺失返回写**就产生了端口复用的情况。这时就需要状态机发出信号来控制写入的数据。
- **Dirty Table**：脏位表，**每一路有一个**，用来标识该路某一行是否有脏数据（需要补全）
 - 对于数据高速缓存来说，在**某一行被替换回主存时**，需要判断这一行是否已经被写过。如果被写过，那么这一行的数据与主存就不一致了，需要把一整行写回到主存当中；如果没有被写过，那么直接替换掉这一行即可。脏位表正是对流水线的写情况进行记录。
 - 脏位表不仅记录了哪一行被写，还防止了刚刚上电后存储器中的随机数据被写回主存。这是因为初始时脏位表全都为0，也就是说在替换时不需要写回。
 - 脏位表有两个更新时间：
 - **写操作命中时**：这时应该把命中路的脏位表对应行写为1

- **写操作缺失处理时**：当写操作缺失处理时，需要把主存中某一行取回到Return Buffer中，并结合写数据写入到存储器。这时还是需要写入一个和主存中内容不同的数据，因此必须把替换路的脏位表对应行写1
- **读操作缺失处理时**：当读操作缺失处理的Return Buffer需要对存储器发起写整行操作时，这一行的数据和主存并没有任何区别，应该把替换路的脏位表对应行写0。虽然这里保持脏位表不变不会造成任何错误，但会造成比较大的性能损失
- **Write Buffer**：写回数据缓存，存储了需要写回的数据
 - 数据通路中这一部分存在问题：**不应该直接将命中的那一路数据放入写回数据缓存中，而是应该将被选中的那一路数据放入写回缓存**。但这一部分助教已经帮助大家实现好了，不需要大家额外进行修改。
 - 写回数据缓冲每次写回一个字后会自动进行右移，以确保每次写回的数据都在这个缓存的[31:0]位
- **Miss Buffer**：写回地址缓存，存储了需要写回的数据的起始地址（**需要补全**）
 - 高速缓存每次需要将一整行的数据写回主存，但不可能一次将整行都发回主存，因此需要逐个数据写回。这时就需要大家准确找到写回行的数据起始地址。
 - 这个起始地址并不是Request Buffer中存储的地址，而应该是由以下几个地址拼接而成：
 - 替换选中那一路的标签
 - Request Buffer中的“中间位”，也就是index
 - 低位全为0：这是因为需要“块地址对齐”，低位全为0恰好就是写回路的起始地址。
- **Writeback FSM**：写回状态机，控制cache的写回操作
 - 本质上写回状态机是一个辅助状态机，在主状态机只负责读操作的同时，利用了AXI总线读写并行的特点，同时写出一行数据
 - 写回状态机需要通过两个信号来与主状态机进行通信，至于他们是如何进行通讯的请自行阅读这一部分代码进行理解，我们将要求同学在实验报告中对这一部分进行阐述。

输入输出信号解读

数据高速缓存的输入输出信号中，读部分与指令高速缓存完全一致，这里就不再赘述了。这里只来介绍一下写操作的端口

- 面向流水线：这里的握手机制与读操作无异，但必须注意wstrb这个信号，这个信号由4为组成，第0位标识了当前是否要写第0个字节，第1位标识了当前是否要写第1个字节，以此类推。**如果是写整字，那么在写操作时直接将这个值置为4'b1111即可。但在读操作时，必须将这个值置为0，否则cache会出错！**
- 面向主存：其中d_wlen和d_wsize和读操作无异
 - **d_wlast**：这个信号是有流水线发出的，用以通知主存，这是最后一个写数据了。
 - **d_bvalid**：主存发给cache的写回成功请求信号，标识了主存已经成功写入了cache的写数据
 - **d_bready**：cache向主存发出的写回成功响应信号，标识了cache已经知道主存成功写入

需要完成的工作

- 补全DCache的命中逻辑
- 设计选择逻辑，通过访存地址和状态机来选择出正确的读数据呈递给流水线
- 设计LRU替换算法
- 补全主状态机（这一部分主要是生成上面两个设计中需要用到的信号）
- 设计脏位表
- 补全miss buffer在替换时写入的替换行地址

AXI仲裁—转接桥介绍

注：这一部分代码已经由助教实现完毕，如果Cache实现得当是完全可以完成所有访存的，无需阅读这一部分。但不排除有同学需要修改转接桥才能适配自己的设计的情况，故在这里简单进行介绍。

仲裁—转接桥是由读、写两个状态机构成的，每一个状态机都将简单的valid-ready握手协议转换为了复杂的AXI总线协议。状态机的设计完全契合拓展课PPT中给出的内容，具体设计可以参考对应PPT。

AXI总线协议信号解读

- 以a开头的信号：地址握手时涉及到的相关信号，其中最为关键的就是地址握手arvalid-arready信号对和awvalid-awready信号对。
- 以r开头的信号：读数据握手时相关信号，其中读数据握手核心信号是rvalid-rready信号对
- 以w开头的信号：写数据握手时相关信号，其中写数据握手核心信号是wvalid-wready信号对
- 以b开头的信号：写回握手（也就是确认主存已经将数据写入）时的相关信号，其中写回数据握手核心信号是bvalid-bready信号对

AXI总线协议访问时序

- 读请求：
 1. 进行读地址arvalid-arready的握手，同时给出本次读请求的各项配置（size、len）
 2. 进行读数据rvalid-rready逐个握手，每返回一个数据就要握手一次，这样存储器才会给出下个数据。直至最后一个数据和rlast同时到来，本次读请求访问结束。
- 写请求：
 1. 进行写地址awvalid-awready的握手，同时给出本次写请求的各项配置（size、len）
 2. 进行写数据wvalid-wready逐个握手，每送出一个写数据就要握手一次，这样存储器才能够准备好接受下一个数据
 3. 最后一个数据需要和wlast同时给出，完成最后一次写握手后还需要进行一次写回握手，否则存储器会锁死。

接入流水线

接入流水线的步骤已经在拓展课中讲过，请大家参考录播视频或PPT，这里我们来介绍几种可能出现的问题：

1. 停顿冒险：由于cache并不知道CPU是否真的用段间寄存器接收到了这个数据，所以极有可能需要cache能看到流水线的stall信号，从而确保读数据不会丢失
2. 取出错误指令：cache在缺失处理时，很有可能流水线传来了flush信号，这时icache处理的请求是错误的，需要设计一个方法来解决这个问题
3. 段间寄存器停顿问题：一个段间寄存器可能有多个stall和flush信号，请仔细思考他们的优先级问题。

测试说明

本次实验需要大家通过两个测试：随机地址访存测试和快速排序测试

随机地址访存测试

- 测试方法说明：

请大家自行创建一个vivado工程，并将目录中除了cache_tb.sv以外的全部verilog代码加入工程。同时，需要使用block memory生成器生成一个AXI风格的BRAM（生成方法详见实验拓展课PPT）。之后，将文件中的cache_tb加入仿真文件，点击仿真即可运行测试

- 测试生成脚本说明：

我们提供一个python脚本，可以生成一个testbench。在执行脚本时，需要输入以下命令：

```
python cache_test_generator.py m n k > cache_tb.sv
```

其中m是高速缓存index（也就是地址中间部分，或者说是高速缓存存储器地址的对应部分）的宽度，n是索引一行中的字的地址宽度，k是总测试数。例如：一种高速缓存配置是每个存储器有8行，每一行有128位，那么m=3，n=128/32=4位。

- 注意：m和n必须是正整数（不能是0），k不可以大于4096

- 仿真说明：

生成testbench后，可以直接进行仿真。注意testbench中维护了几个重要的寄存器：

- i_test_index：标识了当前进行到icache的哪个测试
- d_test_index：标识了昂钱进行到dcache的哪个测试
- i_error_reg：一旦变为1，那么说明icache读出的数据与正确数据不符，此时可以通过波形定位出错位置的访存信息，帮助大家锁定错误位置
- d_error_reg：一旦变为1，那么说明dcache读出的数据与正确数据不符，此时可以通过波形定位出错位置的访存信息，帮助大家锁定错误位置
- i_pass_reg：一旦变为1，那么说明通过了全部icache测试
- d_pass_reg：一旦变为1，那么说明通过了全部dcache测试

- 注意：两个cache的测试是同时进行的，如果你希望只测试一个cache，那么只需要将testbench中向另一个cache发起的valid信号始终置为0即可。

- Debug建议：

建议大家在debug时先锁定error_reg的上升沿，这样对于icache的很多问题都可以迎刃而解。但对于dcache，如果发现问题是在写回主存时产生，那么就需要额外锁定被替换的行对应的存储器的变化时刻，找出对应的问题。

快速排序测试

- 在接入流水线后，大家需要成功运行**256个数字的升序快速排序**并得到正确的结果。我们提供一个快速排序的汇编代码和coe文件，**大家只需要把coe文件写入AXI主存即可。**
- 为了成功运行快速排序，需要实现以下指令：
 - 基础整数指令：addi, lui, auipc, add, sll, **slt**（有符号小于）, xor
 - 分支指令：beq, bge, bne, blt
 - 跳转指令：jalr, jal
 - 访存指令：lw, sw
 - 特殊指令（可以实现为nop）：ebreak
- 快速排序程序说明：
 - **快速排序程序的起始PC是0x0000！** 这里和大家之前实现的不同，需要对流水线做简单的修改
 - 程序的数据和指令是放在一起的，注意如果取到了非法指令，则很有可能是存储在最后的数据，**这时应该让流水线不产生不可预测的行为**
 - 快速排序结束后，程序会自动对结果进行校验，如果发现某个数比和他相邻且下标大于它的数大，那么程序会直接进入一个halt函数，同时将a0（10号寄存器）写为0xeeeeeeee，执行ebreak指令。如果没有实现ebreak指令，那么会陷入死循环
 - 如果排序结果正确，那么程序也会进入halt函数，但会将a0寄存器写为0xacacacac，执行ebreak指令。如果没有实现ebreak指令，那么会陷入死循环。
- Debug建议：
 - 可以先用原来的存储器，把这个coe文件分别加载到指令和数据存储器中，先运行一下，看一下流水线能否成功运行快速排序
 - 如果上述步骤完全正确，那么错误只会出现在流水线的适配上。这时大概率程序会陷入halt函数。**大家特别要注意“丢指令”问题，仔细检查每一个cache缺失处理结束后送出的指令是否被流水线接收到了**