《计算机系统结构》实验报告

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 年级、专业、班级 | |  | | | 姓名 |  |
| 实验题目 | **实验二 Cache设计与实现** | | | | | |
| 实验时间 | 2022年11月6日 | | 实验地点 | DS1421 | | |
| 实验成绩 |  | | 实验性质 | □验证性 🗹设计性 □综合性 | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  1. 加深对Cache原理的理解  2. 通过使用Verilog实现Cache，加深对状态机的理解 | | | | | | |
| 二、实验项目内容  1. 最低要求：参考指导书中直接映射写直达Cache的实现，实现写回策略的Cache  2. 替换实验环境中的Cache模块，并通过仿真测试  3. [完成][较高要求]性能优化，实现2路组相联的Cache  4. [完成][更高要求]性能优化，使用伪LRU等替换策略实现4路以上组相联的 Cache  5. [未完成][最高要求]实现其它Cache性能优化方法，如axi burst传输 | | | | | | |
| 三、实验过程或算法（源程序）  （一）总体框架  项目的总体框架如下图所示。    MIPS core通过两个类sram接口对外进行指令访问和数据访问。当MIPS core向Cache模块请求指令和数据时，Cache模块如果命中，则马上返回数据，否则需要访问内存。而访问内存时，Cache 模块只需产生类sram信号，传给下方模块处理即可实现访存。  我们需要实现的是Cache这一部分的功能，其中包括I-Cache和D-Cache，由于I-Cache只读，而D-Cache可读可写，因此实验中我们只需实现D-Cache即可。  首先需要了解模块的输入与输出，其定义如下表。   |  |  |  |  | | --- | --- | --- | --- | | 信号名 | 方向 | 位宽 | 功能描述 | | clk | input | 1 | 时钟 | | rst | input | 1 | 复位 | | MIPS core 上层接口 | | | | | cpu\_data\_req | input | 1 | CPU是否发出数据请求 | | cpu\_data\_wr | input | 1 | CPU是否要写数据 | | cpu\_data\_size | input | 2 | 结合地址最低两位，确定数据的有效字节  （用于sb、sh等指令） | | cpu\_data\_addr | input | 32 | 数据地址, 一个字 | | cpu\_data\_wdata | input | 32 | 要写入的数据 | | cpu\_data\_rdata | output | 32 | 读出的数据 | | cpu\_data\_addr\_ok | output | 1 | Cache已经收到地址 | | cpu\_data\_data\_ok | output | 1 | 可以读出数据了 | | axi interface 下层接口 | | | | | cache\_data\_req | output | 1 | 是否要发送访存请求 | | cache\_data\_wr | output | 1 | 是否要写数据 | | cache\_data\_size | output | 2 | 数据大小 | | cache\_data\_addr | output | 32 | 数据写入地址 | | cache\_data\_wdata | output | 32 | 要写入的数据 | | cache\_data\_rdata | input | 32 | 下方读出的数据 | | cache\_data\_addr\_ok | input | 1 | 下方已经收到地址 | | cache\_data\_data\_ok | input | 1 | 下方是否已经准备好了数据 |   其中将输入输出分为两部分，一部分是与上方（CPU）的交互，一部分是与下方（MEM）的交互。CPU进行访存过程中，信号量值如下：  CPU请求读数据时：cpu\_data\_req==1 && cpu\_data\_wr==0 && cpu\_data\_addr有效。  CPU请求写数据时：cpu\_data\_req==1 && cpu\_data\_wr==1 && cpu\_data\_addr有效。  CPU取出读数据时：cpu\_data\_rdata有效 && cpu\_data\_data\_ok == 1。  CPU完成写数据时：cpu\_data\_data\_ok == 1。  （二）直接映射Cache + 写直达-写不分配策略  1. 直接映射  直接映射 Cache 结构如下图所示。    内存地址被拆成tag，index和offset三部分。通过index 访问一个Cache line，通过offset确定Cache line中数据块中对应的字。由于Cache的容量远小于内存的容量，因此无法做到一一对应。当地址的 index和offset都相同时，就会将两个地址映射到Cache中的同一个位置造成冲突，因此需要tag来表示两个不同的地址。Cache通过比较地址的tag和Cache line的tag看是否命中。  2. 写直达-写不分配  在CPU执行写内存操作时，可以将数据同时写入到Cache和内存。这样保证了Cache 和内存中的数据都是修改后最新的数据。这种策略就叫做写直达。写直达通常结合写不分配策略一起使用。当写缺失时，直接写入内存，而不写入 Cache。  这种方式的状态机如下图所示。其中IDLE表示空闲状态，RM表示正在读取内存，WM表示正在写内存。    （二）直接映射Cache + 写回-写分配策略  1. 写回-写分配  在CPU执行写内存操作时，只写入Cache，同时标记该Cache line为已修改，等到该 Cache line被替换时再写入内存。这种策略就称为写回。写回通常结合写分配策略一起使用，即写缺失时，只写入Cache，而不写入内存。下面分析各种情况下的操作。  （1）读命中：CPU直接读取Cache line的数据。  （2）读缺失：①如果索引到的Cache line是干净的，直接向内存发送读请求，将从内存读取的数据返回给CPU，同时将数据写入到索引到的Cache line中；②如果索引到的cache line是脏的，首先向内存发送写请求将脏数据写回内存，写请求处理完成后再发送读请求，将从内存读取的数据返回给CPU，同时将数据写入到索引到的Cache line中。（此时dirty位应该置为0，valid位应该置为1）。  （3）写命中：直接将数据写入到对应的Cache line中，将dirty位置为1。  （4）写缺失：①如果索引到的Cache line是干净的，直接向内存发送读请求，将从内存读取的数据写入到索引到的Cache line中，并将CPU传过来的数据写入到索引到的Cache line中，同时将dirty位置为1（为什么这里不直接写入Cache line？因为当出现sb等指令写入部分字节时，Cache line中的数据并非要写的地址中的数据，因此需要先将数据读出来再向其中写入部分字节）；②如果索引到的cache line是脏的，首先向内存发送写请求将脏数据写回内存，写请求处理完成后再向内存发送读请求，将从内存读取的数据写入到索引到的cache line中，并将CPU传过来的数据写入到索引到的cache line中，同时将dirty位置为1。  根据上面的分析，可以设计写回-写分配策略的状态机如下图所示。    对该状态图值得注意的是：  当CPU发出请求且命中时，直接对cache进行读写。  当CPU发出请求但不命中时，考虑两种情况：   1. 如果index索引到的块是脏数据，先将脏数据写回（WM），再把需要的数据读入cache（RM），再处理CPU的请求。 2. 如果index索引到的块不是脏数据，直接将需要的数据读入cache（RM），再处理CPU请求。   具体实现如下：    2. 代码修改  由于源代码实现的是写直达，因此我们需要在源代码的基础上修改部分逻辑以匹配写回策略。具体变化如下：  （1）增加dirty位及其相关逻辑。  （2）由于读写遇到脏位时应该要先把Cache line写回内存，因此需要改变向axi传递的访存地址以及数据。    （3）由于读写缺失且Cache line dirty时需要用两个进行处理，因此需要暂存一下本周期的各种数据（tag、index等），避免下一个周期找错数据。    （4）对于sb、sh等指令需要使用掩码来写入正确的数据，在采用写回后，写入的Cache line的数据可能和内存中的数据不一样，因此需要RM去读出正确的数据。所以特别要注意下面的掩码使用中的old\_data和write\_data。    （5）更新Cache line时要注意上一个周期是由IDLE（wr\_save=0）转过来的还是WM （wr\_save=1）转过来的。要根据这个选择写入的数据是什么。    （三）2路组相联Cache + 写回-写分配策略  1. 2路组相联  2路组相联的Cache结构如下图所示。    2路组相联Cache与直接映射Cache的最大区别就是多加了一路（可以理解为一个index对应了两个cache line，需要将tag进行比较才能确定命中否）。  当cache发生缺失时，需要替换Cache line时，我们需要从对应的Cache set中选择一个Cache line进行替换。在2路组相联中，我们可以用一位标记位记录下最近使用过的Cache line，则可以通过标志位选择另一个cache line进行替换。  2. 代码修改  基于上面的直接映射写回Cache多添加一维即可。具体修改如下：  （1）为Cache中的各个板块都添加一维，并添加一个cache\_LRU来记录哪一路最近被使用过（0：way\_0最近使用过；1：way\_1最近使用过），方便判断之后缺失时替换那个块。    （2）判断是否命中需要比较两路。    （3）选择要替换的路。如果还有invalid的Cache line，就先用这一块作为替换的块。如果没有invalid的Cache line了，那么就需要使用LRU算法选出最近没有被使用过的Cache line。因为cache\_LRU中存储了哪一路最近被访问过，因此替换的就是cache\_LRU相对的那一路。    （4）更新cache\_LRU，如果命中则将最近访问设为hit的路，缺失则将最近访问设为replace的那一路。    （5）其他需要改变其实不多，主要是将之前的index改为replace\_num即可。同时注意一下采用这种方式Cache存储方式一定要对每一块都赋初值，不能直接用一个索引就赋值了（这样会报错的）。  （四）4路组相联Cache + 伪LRU替换策略 +写回-写分配策略  1. 4路组相联  4路组相联的Cache结构如下图所示：    4路组相联主要是将每一行的大小和标志做扩展，命中的判断、无效块的选择和更新cache的选择做了相应的拓展，状态转移逻辑和其他信号保持不变，与2路组相联类似。  2. 伪LRU替换策略  4路组相联使用了伪LRU算法，为此我们为cache的每一行准备了3个比特。通过这3个比特，我们可以实现一种类似二叉树的伪LRU算法。  具体过程如下: 初始时可以全部赋0或1，假设是赋0，则在位0处向左子树查找，位1也是0，则选择左边的way0。所以每一位为0时代表左边比右边更久未访问。那更新这3个比特的方法也就呼之欲出了，每当我们访问了某一路，我们可以将路径上的比特置为方向的反(假设左走是0，右走是1)，比如我们访问了way1，那位0就应置为0的反即1，位1就应置为0。    3. 代码修改  （1）修改路数，添加用于伪LRU的三位存储。    （2）修改取出命中路数、无效块数等类似2路组相联，详见源代码。  （3）选择替换的Cache line所在的路数。由上面的分析可以模拟取出替换块的位置。具体实现如下。    （4）注意对于多维数组要用下面的方式进行初始化（不能直接赋值为0）.    （5）更新pLRU，命中时则使用hit\_num更新，未命中时则使用replace\_num更新。根据上面提到的方法进行更新。 | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  （一）直接映射Cache + 写直达-写不分配策略    （二）直接映射Cache + 写回-写分配策略    （三）2路组相联Cache + 写回-写分配策略    （四）4路组相联Cache + 伪LRU替换策略 +写回-写分配策略    从实验结果可以看到，写回策略要比写直达效率更高些；随着相联度的提升，性能逐渐变高。  （五）实验中遇到的问题及解决方法  **（1）仿真失败，如下图所示：**    定位到PC=0x9fc00c90：    发现是load指令出错，说明之前数据可能没写对。查看波形图：    发现cache\_data\_wdata一直都是x，这也进一步说明在写数据的时候出了错。  排查发现是如下代码有问题：    第二条代码不应该使用c\_block作为原数据，而应该是cache\_data\_rdata，修改后的代码如下：    **（2）仿真成功，但二路组相联总和四路组相联结果相同。**    经过排查发现，原来是因为所有num都没有变成两位，导致结果只取了一位！所以二路和四路结果相同。错误类似下图。    修改位数之后还是不能得到仿真的正确结果。    再排查发现，原来在更新LRU时还存在着一个问题，更新时使用了index而未使用index\_save从而使得LRU更新错误，没有起到该有的作用。错误如下图所示    将index修改为index\_save即可。    修改后便能正确跑出结果。 | | | | | | |
| 1. 小组分工   李卢奕：实现写回策略的直接映射Cache  贺爱兰：实现2路组相联的Cache  鲁茹芸：使用伪LRU实现4路组相联的Cache  王雪柔：解决仿真出现的问题，运行截图，完成实验报告 | | | | | | |