

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机体系结构 |
| 姓 名： |  |
| 学 院： | 计算机科学与技术学院 |
| 系： | 计算机科学与技术 |
| 专 业： | 计算机科学与技术 |
| 学 号： |  |
| 指导教师： |  |

2023年 10 月 31 日

**浙江大学实验报告**

课程名称： 计算机体系结构 实验类型： 综合

实验项目名称： 高速缓存的设计

学生姓名： 专业： 计算机科学与技术 学号：

同组学生姓名： 指导老师：

实验地点： 曹西301 实验日期： 2023 年 10 月 31 日

1. 实验目的和要求

实验目的：

1. Understand Cache Line.

2. Understand the principle of Cache Management Unit (CMU) and State Machine of CMU.

3. Master the design methods of CMU.

4. Master the design methods of Cache Line.

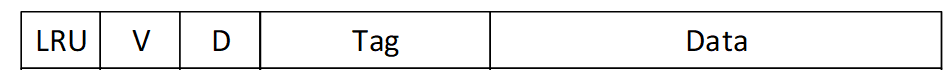
5. Master verification methods of Cache Line.

实验任务：

1. Design of Cache Line and CMU.
2. Verify the Cache Line and CMU.
3. Observe the Waveform of Simulation.
4. 实验内容和原理

2.1 cache line的基本表示

在一般的cache line表示中，我们需要一个LRU位用于记录最临近使用的cache line数据，便于后续miss时触发的置换操作；另外，我们需要V与D bit来分别表示cache line对映数据是否处于有效或已被修改（脏数据）状态；另外，我们也需要设置对映的Tag位用于比对数据地址来源；最后则是真正存放与当前cache line中的数据。具体的cache line样例如下：



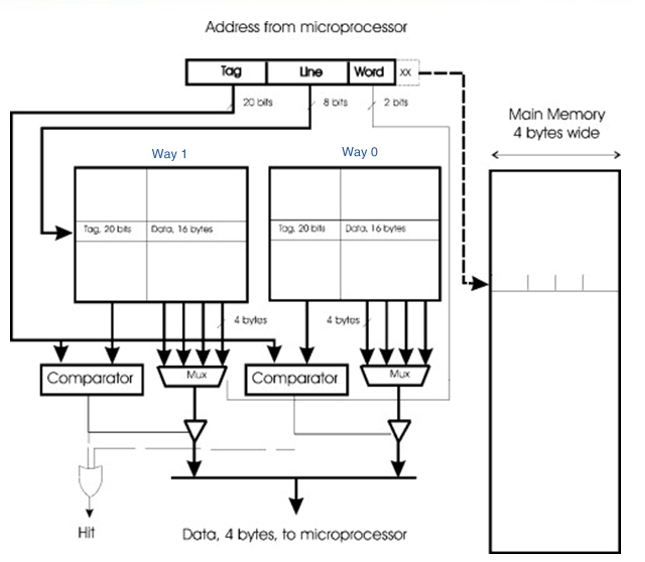
2.2 cache mode

首先，我们需要知晓地址在使用cache中所起的作用，一般我们对address的分解如下：



第一部分是Tag位，用于判断和校对cache中存放的数据是否为需要数据，这一部分的长度往往由总address bit长度减去后两部分的长度得到。第二部分是index索引，它规定了每一part中cache可以容纳的最大block数量；最后一部分为offset偏移量，它便于我们在block中寻找对映的word/byte。

Cache一般的操作规则如下（我们以2-way set association为例）：



首先，我们对address进行分解，接着我们取出index，并于两份cache中寻得对映的cache line；接着我们从cache line中取得tag与data数据。对于tag而言，我们将其与address中的tag位进行比对，只要两者中有一与address\_tag一致，我们则将hit置1；没有则反之。另外，对于从cache中取出的数据而言，我们可以通过address地址中带有的offset数值，来判断具体的数据需求。

2.3 write策略

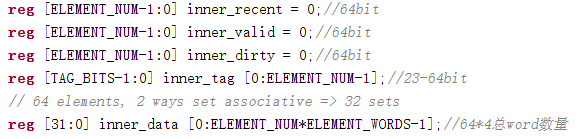
在cache的实现中，我们避不开对写策略的实现。当我们在查询cache并获得hit的结果后，我们也有两种write的策略。

第一种，write back，对于这种策略而言，我们仅仅将需要写的内容写入cache中即停止。而对于第二种，write through，我们不仅要将需要写的内容写入cache中，我们还需要将其写入对映的内存block中，保持cache与memory的一致性。

但对于cache miss而言，我们也有write allocate和no write allocate两种策略。对于第一种而言，我们需要将需要写入的block从memory放入到cache中，再进行修改；而对于后一种而言，我们则直接将需要修改的内容写入memory中对映的block里。

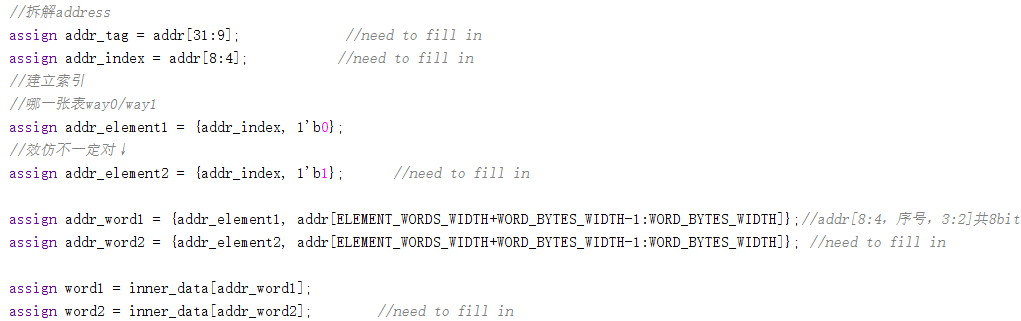
1. 实验过程和数据记录
   1. Cache的具体实现

在本实验中，大部分的实现代码均已给出，因而在此我也将这部分给出的代码作一个大致的解释分析。

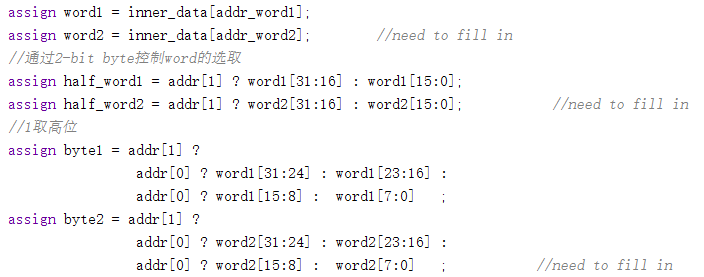
首先，我们可以建立一系列寄存器，用于保存两张cache中的word信息、valid位信息、dirty位信息以及最近使用信息。

接着，我们需要对这些寄存器进行初始化操作，其中主要初始化tag寄存器组与data寄存器组即可。

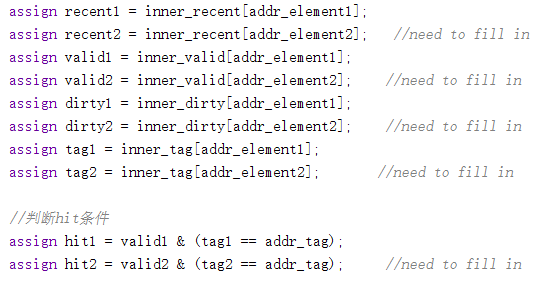
接下来，我们需要对address进行拆解。我们可以直接获得相应的tag与index信息，并组件新的element索引信息，用于区别cache表中的way1与way2数据，并根据新的索引，获得对映的word数据。



因为lb、lw、lf等指令对不同字节位数数据的要求，我们也需要进行相应的数据分割。



接着，我们需要对一些控制变量进行赋值，用于记录对映cache line的valid、dirty、是否最近被使用状态以及对应的tag寻址参数，并判断cache line是否被hit。



完成这些操作后，我们需要对输入操作指令不同时cache的操作进行定义。在此，我们仅以way1为例进行分析。

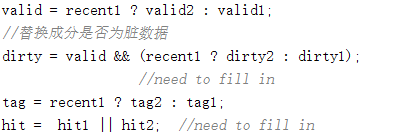
对于load指令而言，在确保cache hit的前提下，我们需要根据指令的需求，为数据赋予相应字节数的cache内容；并在赋值最后，改变way1与way2的最近使用状态。

而对于edit修改而言，我们首先也要确保hit的产生。接着同样根据指令的需求，为数据赋予相应字节数的cache内容；在此时，我们不仅要改变way1与way2最近的使用状态，我们还需要将对映way的dirty位进行置位。

对于store操作而言，我们要确定最近使用的way是哪个，对不最近使用的way进行替换，或直接填入空闲的way中。对于替换操作而言，我们需要将目标数据写入cache中，并把dirty为置0、valid位置1，并重置对映的tag数据。

而对于invalid情况，我们需要把所有valid、dirty、最临近使用数据清零。

最后，我们需要谨慎设置模块输出的控制信号。



对于hit而言，只要way1或way2产生了hit，我们便将其置1。对于valid而言，我们首先要确定哪一个cache数据被替换了，再将被替换的cache数据对映line valid信号进行输出。而dirty的置位与否，不仅要如valid位的筛选条件一般进行dirty位的选择；dirty位的置位还需要以valid位置位为前提，若valid无效，dirty将不影响数据的写回与否。最后是tag，仅需要根据被置换的cache数据进行输出即可。

1. 实验结果分析

对于上述设计，我们进行了仿真，仿真结果见后。

在此，我们对仿真结果作一定的分析（仅取部分），以表现我们模块设计的正确性。

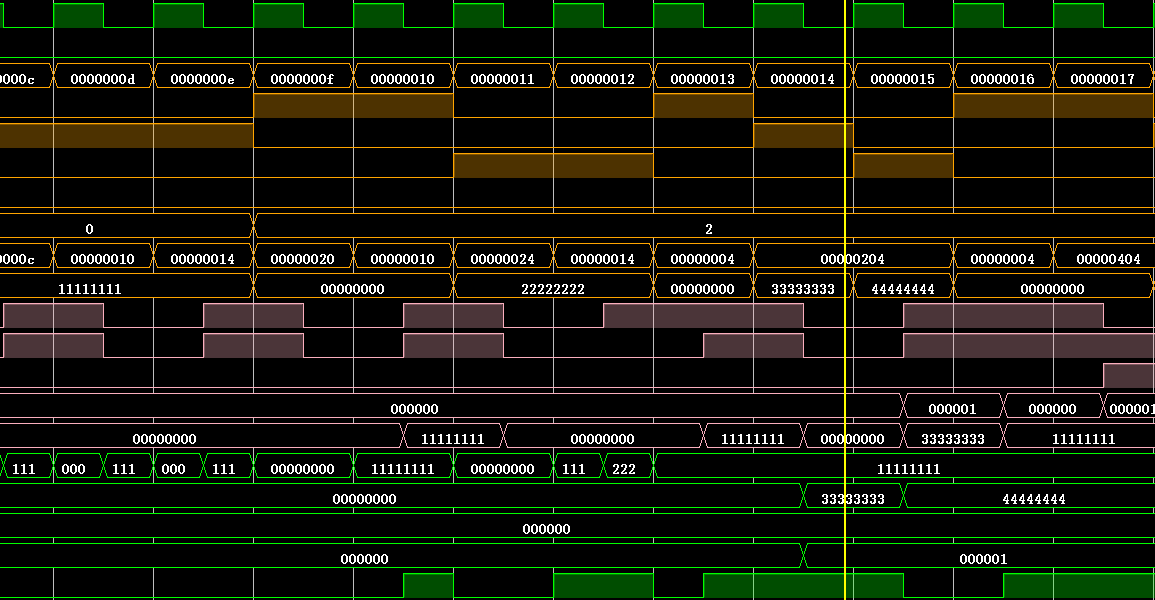
我们首先对一些地址进行了数据的填充，此时可以看到，store控制信号一直处于置位状态。我们使用这种方式，来模拟cache中已经存在了部分数据。

接下来，我们尝试使用load指令进行数据的读取。因为读取区域我们并未进行对映的写入操作，因而发生了read miss现象；从结果我们可以看见，hit为0，表示cache miss情况的产生。但在相邻的一个时钟周期后，我们还是进行load指令，而对应的地址已经被我们提前写入，因而此时我们完成了cache hit，进而hit已经被置位为1。

在后续的操作中，如第0x13步中，虽然产生了hit，但我们的valid位同样被置1；因为我们的hit信号，置位所取的是或操作，因而在仿真进行中，只要有一个way产生了cache hit，hit信号就会被置位；对于valid位而言，这只是说明了当前way2处于valid状态，且way1被最临近访问过，意味着如果发生替换，需要关注dirty信号的有无；另外，valid信号并没有给出更多的信息。

在接下来的仿真操作中，我们还测试了因tag miss而导致的read miss、edit hit line1 of set 0、read line 0 of set 0等情况，均得到了正确的控制信号输出结果。

可以注意到，在本次仿真图形中，我们设置的信号量变化均在时钟的下降沿产生；这并不是设计的错误，而是贴合了cache在时钟下降沿进行读数据或写数据的设计需求，具体分析与效用将在lab4进行展开。



1. 讨论与心得

整体而言，还是一个较为简单的实验。虽然这次实验的具体实现，或者说代码填空还是相对直截了当，但是cache的设计感觉复杂之中带有些许巧妙。

对于这次的实验设计而言，大体在于理解cache的设计原理，对于具体的实现，只需要将一些word进行数字替换即可。本实验最大的难点在于模块信号输出的控制，在没有课堂讲解以及lab4的解释下，我对此还是有一些理解偏颇的。在了解输出信号的原理与作用后，这些问题与具体实现也就迎刃而解了。

另外，多位寄存器组的较为奇特的定义方法，也算是本实验的一大收获了；localparam常数项的定义以及.h文件的规范应用，也使我学到了许多verilog新知识，受益匪浅。