Lab3实验报告

焦培淇 PB17151767

一、实验目的

1. 权衡 cache size 增大带来的命中率提升收益和存储资源电路面积的开销  
2. 权衡选择合适的组相连度（相连度增大 cache size 也会增大，但是冲突 miss 会减低）  
3. 体会使用复杂电路实现复杂替换策略带来的收益和简单替换策略的优势（有时候简单策  
略比复杂策略效果不差很多甚至可能更好）  
4. 理解写回法的优劣

二、实验环境

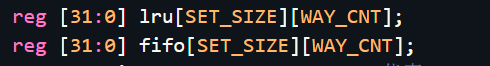
WIN10+vivado 2018.2+vscode

三、实验内容与过程

1. 第一阶段：cache的编写与验证

（1）编写思想

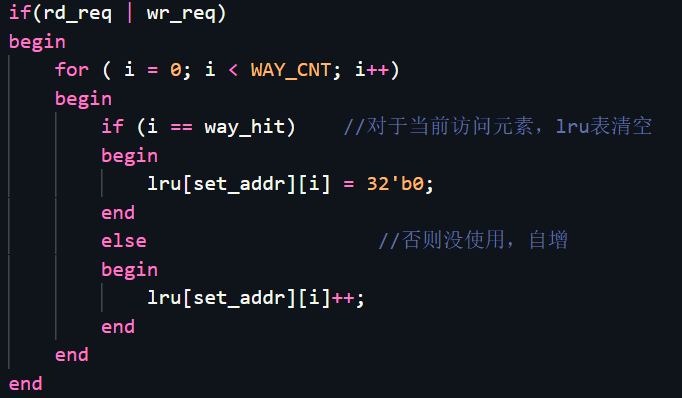
对于能实现FIFO和LRU的Cache，首先定义能够LRU和FIFO信息表如下：



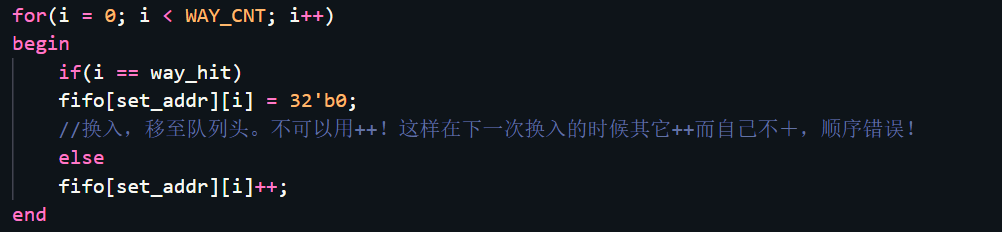
存储每个set内的每个位置的信息，以此在换出时确定需要进行换出的块。

然后确定信息表的更新模式

对于LRU表，每次在进行cache访问时，如果命中，则将对应位置的信息清零，而对于处于同一个set内的其它位置，信息值加一，代码如下：

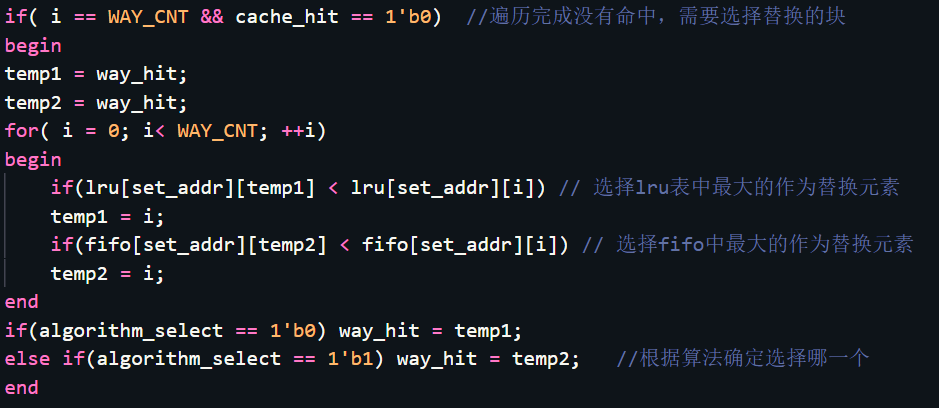


而对于FIFO表，则需要在换入时进行更新，对新换入的位置，其信息置0，其他位置信息加一，代码如下：



确定每次换出位置的选择

对于LRU方式，每次换出信息值最大的位置（代表访问时间最久远的位置），而对于FIFO方式，同样选择信息值最大的位置（代表最早换入的位置），代码实现如下：



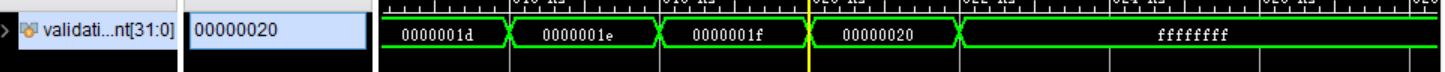
这样，每次在进行cache替换时，就根据上面way\_hit，选择要踢出的块。

（2）正确性验证

利用规模为32的样例验证LRU如下，可见编写正确



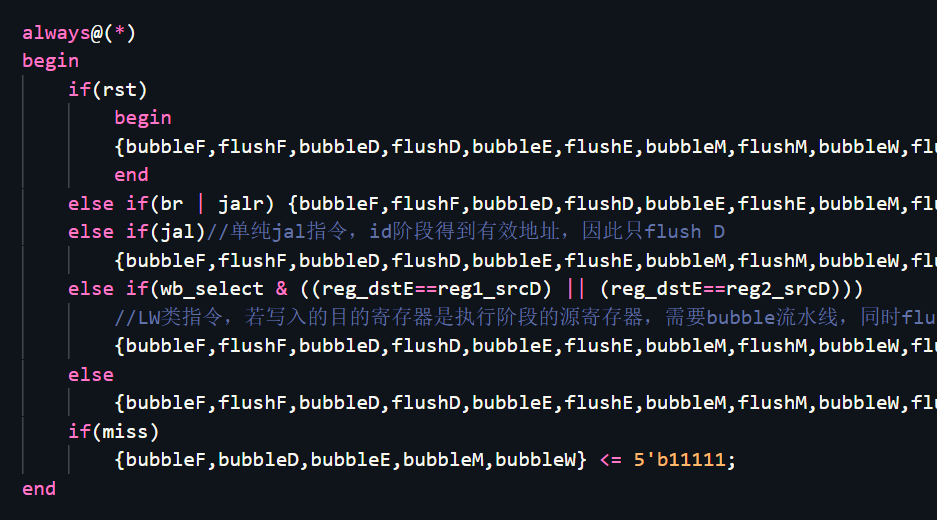
利用规模为32的样例验证FIFO如下，可见编写正确



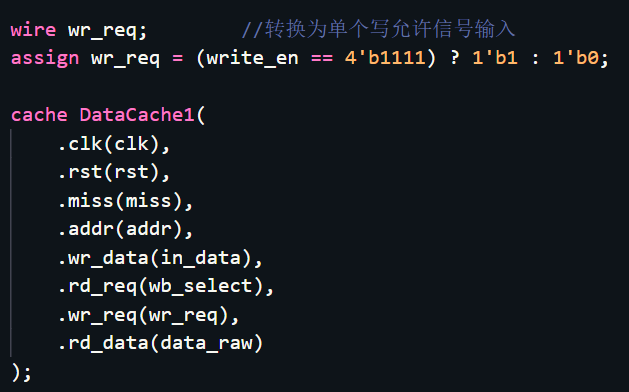
2. 第二阶段：添加cpu代码并成功运行矩阵乘法和快速排序算法

（1）额外数据通路的添加

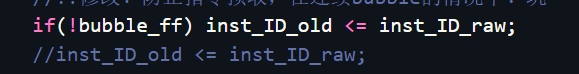
首先在Hazard模块中添加miss信号的输入，对于发生cache miss的情况，暂停整条流水线。代码如下：

在发生miss是，整个流水线进行bubble。

然后，在WbData中调用新增加的cache模块，当write\_en信号为4’b1111时，视为发出wr\_req信号，当wb\_select选择memory时，代表发出rd\_req信号，实现代码如下：

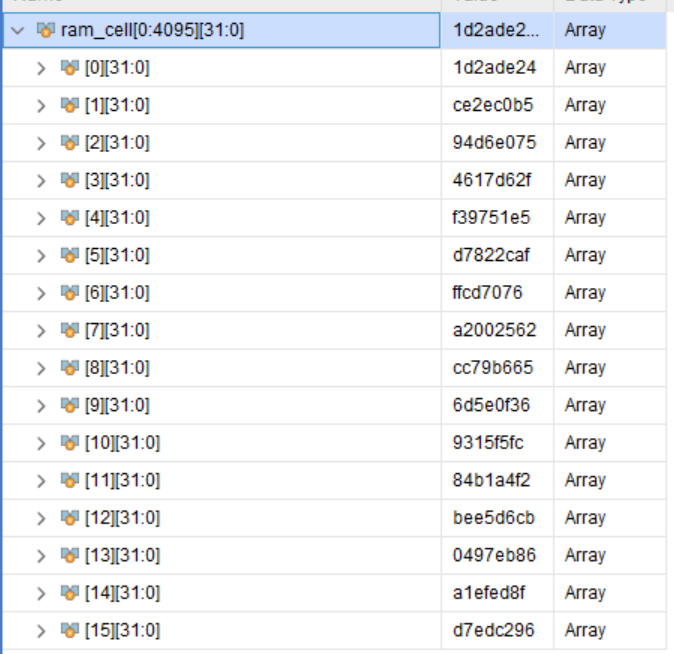
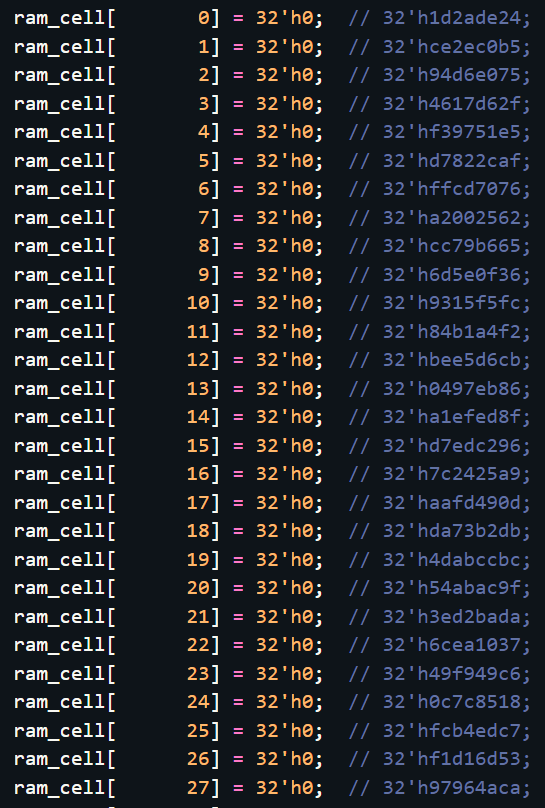


最后，对于IR阶段进行修改，防止在进行bubble是，指令内容变化



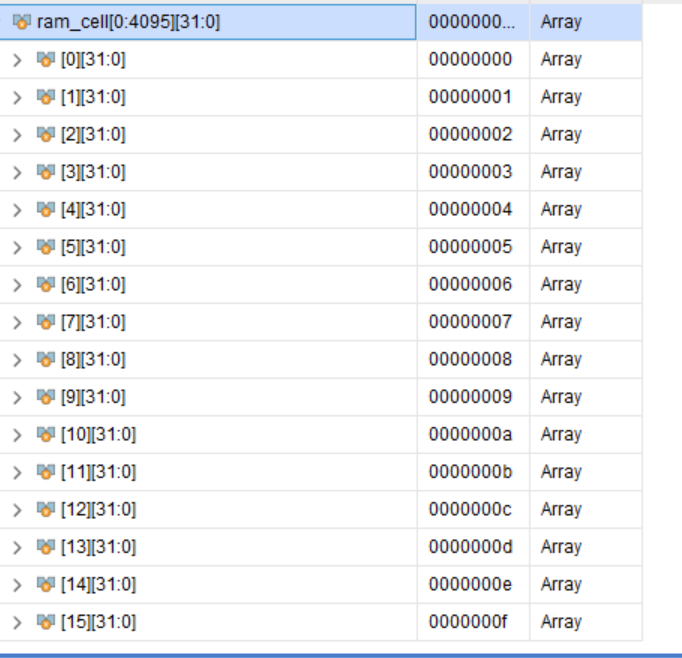
（2）正确性验证

运行规模为16的矩阵乘法结果如下：

对比两侧数据可见，算法运行正确。

运行规模为512的快速排序算法结果如下：



可见主存中的数据基本有序，算法运行正确。

3. cache分析

在保持主存（4096字节）和cache大小不变的情况下，对不同相连度的cache性能进行探究如下：

矩阵乘法（规模16\*16）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 策略 | 组数 | 相连度 | LUT | FF | Miss率 | 运行时间(ns） |
| FIFO | 4 | 8 | 8334 | 10445 | 0.6618 | 920138 |
| FIFO | 8 | 4 | 5711 | 10413 | 0.36067 | 515130 |
| FIFO | 32 | 1 | 7553 | 17645 | 0.74216 | 687906 |
| LRU | 4 | 8 | 8334 | 10445 | 0.59485 | 822950 |
| LRU | 8 | 4 | 5711 | 10413 | 0.35256 | 527202 |
| LRU | 32 | 1 | 7553 | 17645 | 0.7212 | 675796 |

快速排序算法（规模256）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 策略 | 组数 | 相连度 | LUT | FF | Miss率 | 运行时间(ns） |
| FIFO | 4 | 8 | 8334 | 10445 | 0.2524 | 151866 |
| FIFO | 8 | 4 | 5711 | 10413 | 0.07348 | 150806 |
| FIFO | 32 | 1 | 7553 | 17645 | 0.03433 | 168204 |
| LRU | 4 | 8 | 8334 | 10445 | 0.20334 | 150244 |
| LRU | 8 | 4 | 5711 | 10413 | 0.07245 | 149724 |
| LRU | 32 | 1 | 7553 | 17645 | 0.03433 | 168208 |

分析：

（1）. cache花费的电路面积随着相连度的减小而逐渐增大，可见对于全相联的cache复杂程度最高，而直接映射的cache较为简单。

（2）. 对于FIFO策略和LRU策略，对比可见，LRU换出策略要由于FIFO策略。

（3）. 快速排序算法的miss率要远远优于矩阵乘法，原因在于快速排序算法相比矩阵乘法能更好的利用空间局部性原理。

（4）. 对于cache来讲，相连度也并不是越大越好，存在一个较为适中的相连度能够提供最好的表现。对于快速排序算法，全相联的cache可以提供最好的效果，而对于矩阵乘法而言，4路组相联的表现要更好。

保持相连度为4不变，改变cache大小，进行探究如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 策略 | 样例 | 组数 | LUT | FF | Miss率 | 运行时间(ns） |
| FIFO | 排序 | 4 | 5621 | 5693 | 0.33902 | 188758 |
| FIFO | 排序 | 8 | 5711 | 10413 | 0.07348 | 150806 |
| FIFO | 排序 | 16 | 10277 | 19821 | 0.02668 | 135348 |
| FIFO | 矩阵 | 4 | 5621 | 5693 | 0.72838 | 1349290 |
| FIFO | 矩阵 | 8 | 5711 | 10413 | 0.36067 | 515130 |

分析：

（1）. 首先，很显然随着cache size的增大，消耗的资源也在变多。

（2）. 随着cache size的增大，miss率也在不断的下降，同时对应的运行时间也在不断降低，可见增加cache的大小可以在一定程度上优化cache的表现，但是也会带来更多硬件的代价，因此也不能无限制的增加cache的大小。

从电路面积，miss率，运行时间等项进行权衡，可见8组4路使用LRU策略的cache较优。

4. 写回法的劣势

可以看出，写回法通过减少不必要的主存访问提高了运行的速度，但是对于本次实验这种情况，就会产生因cache中数据没有完全写回，观察到的mem数据可能是不正确的情况。

四、实验小结

在本次实验中，实现了可以选择LRU和FIFO策略的cache，并进行了一系列的性能测试。通过实验中真实的调整cache参数，观察生成的cache的表现以及生成的cache复杂度等，是自己更加清楚了课上所讲的cache设计策略。同时，手工完成组相联的cache实现，加深了对于不同cache构造的理解。