LAB3 Report

何跃强 PB22111649

一、实验目的

1.学习不同策略（FIFO,LRU,LFU）的cache的编写

2.学习cache接入CPU的步骤，并对一些程序（如快速排序，矩阵相乘）进行模拟，测试其正确性

3.考虑cache的不同参数，体会cache size、组相连度、替换策略针对不同程序的优化效果，以及策略改变带来的电路面积的变化。

二、实验要求

1.编写FIFO,LRU,LFU策略的cache

2.把写好的cache接入到CPU中，运行不同规模的快速排序，矩阵相乘的程序

3.修改cache不同参数，分析不同参数对cache的性能与成本的影响

三、代码思路

3.1 不同策略的cache实现

3.1.1 直接映射cache转变为组相联cache

添加WAY_SIZE，代表组相联程度，即每一路的块的数目

```
localparam WAY_SIZE         = 1 << WAY_CNT ;                    // 计算组相连度，即每组有多少路 line


reg [            31:0] cache_mem    [SET_SIZE][WAY_SIZE][LINE_SIZE]; // SET_SIZE * WAY_SIZE 个line，每个line有LINE_SIZE个word
reg [TAG_ADDR_LEN-1:0] cache_tags   [SET_SIZE][WAY_SIZE];            // SET_SIZE * WAY_SIZE 个TAG
reg                    valid        [SET_SIZE][WAY_SIZE];            // SET_SIZE * WAY_SIZE 个valid(有效位)
reg                    dirty        [SET_SIZE][WAY_SIZE];            // SET_SIZE * WAY_SIZE 个dirty(脏位)
reg [            31:0] score        [SET_SIZE][WAY_SIZE];            // SET_SIZE * WAY_SIZE 个score，用于使用FIFO,LRU,LFU等算法时使用
```

命中判断需要用并行判断

```
reg cache_hit = 1'b0;
reg [WAY_CNT-1:0] cache_hit_way = 0; // 用于记录命中的line的索引
always @ (*) begin              // 判断 输入的address 是否在 cache 中命中
    // if(valid[set_addr] && cache_tags[set_addr] == tag_addr)   // 如果 cache line有效，并且tag与输入地址中的tag相等，则命中
    //     cache_hit = 1'b1;
    // else
    //     cache_hit = 1'b0;
    for( integer i = 0; i < WAY_SIZE; i++) begin
```

```
        if(valid[set_addr][i] && cache_tags[set_addr][i] == tag_addr) begin
            cache_hit = 1'b1;
            cache_hit_way = i; // 记录命中的line的索引
            break;
        end else begin
            cache_hit = 1'b0;
        end
    end
end
```

3.1.2 LRU策略的实现

每一路中对每个块设立一个新的变量：score，用于裁决该淘汰什么块。

LRU中，每次命中的块变为0，其余的块都加1。这样每次就可以淘汰score最大的块。我们添加一个状态用于比较得到score最大的块。

代码实现见附件。

3.1.3 LFU策略的实现

每一路中对每个块设立一个新的变量：score，用于裁决该淘汰什么块。

LFU中，每次命中的块加1。这样每次就可以淘汰score最小的块。我们添加一个状态用于比较得到score最小的块。

代码实现见附件。

3.1.4 FIFO策略的实现

每一路中对每个块设立一个新的变量：score，用于裁决该淘汰什么块。

FIFO中，每次命中的块变为0，其余的块都加1。这样每次就可以淘汰score最小的块。我们添加一个状态用于比较得到score最小的块。

代码实现见附件。

3.2 cache接入CPU的实现

3.2.1 WB模块的改写

WB阶段我们需要实现cache的接入以及rd_req，wr_req和miss的握手协议。

```
//rd_req需要完成握手协议
    reg rd_req = 1'b0;
    always@(*)begin
        if( !bubblew)begin
            if( load_type != 0 )begin
                rd_req = 1'b1;
            end
            else begin
                rd_req = 1'b0;
            end
        end
    end

    reg wr_req = 1'b0;
    always@(*)begin
        if( !bubblew)begin
```

```
            if( write_en != 0 )begin
                wr_req = 1'b1;
            end
            else begin
                wr_req = 1'b0;
            end
        end
    end

    cache_LRU #(
    .LINE_ADDR_LEN  ( 3             ),
    .SET_ADDR_LEN   ( 2             ),
    .TAG_ADDR_LEN   ( 12            ),
    .WAY_CNT        ( 3             )
) cache_test_instance (
    .clk            ( clk           ),
    .rst            ( rst           ),
    .miss           ( miss          ),
    .addr           ( addr          ),
    .rd_req         ( rd_req        ),
    .rd_data        ( data_WB_raw   ),
    .wr_req         ( wr_req      ),
    .wr_data        ( in_data       )
);
```

3.2.2 顶层模块的改写

顶层模块（RV32ICore）添加miss变量即可，用于连接WB和Harzard两个模块。

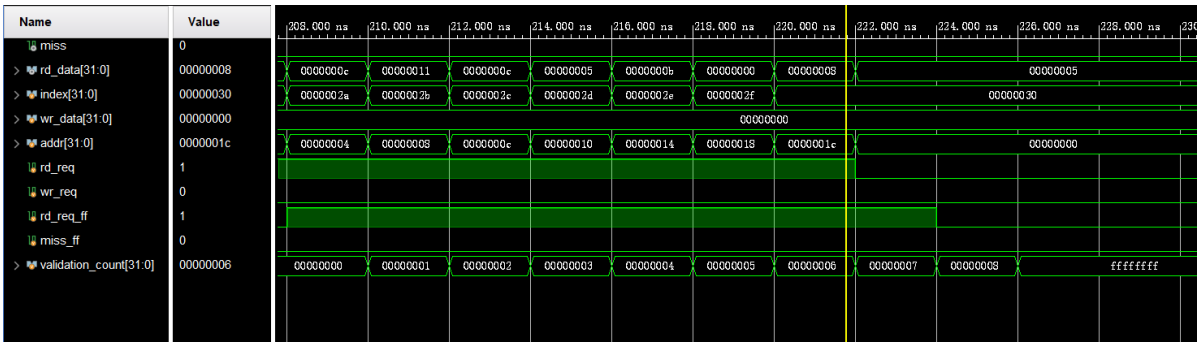
3.2.3 Harzard 模块的改写

对于miss的信号，流水线CPU的所有阶段需要暂停

```
if (miss)
            begin
                debug = 0 ;
                bubbleM = 1;
                flushM = 0;
                bubbleW = 1;
                flushW = 0;
                bubbleE = 1;
                flushE = 0;
                bubbleD = 1;
                flushD = 0;
                bubbleF = 1;
                flushF = 0;
            end
```

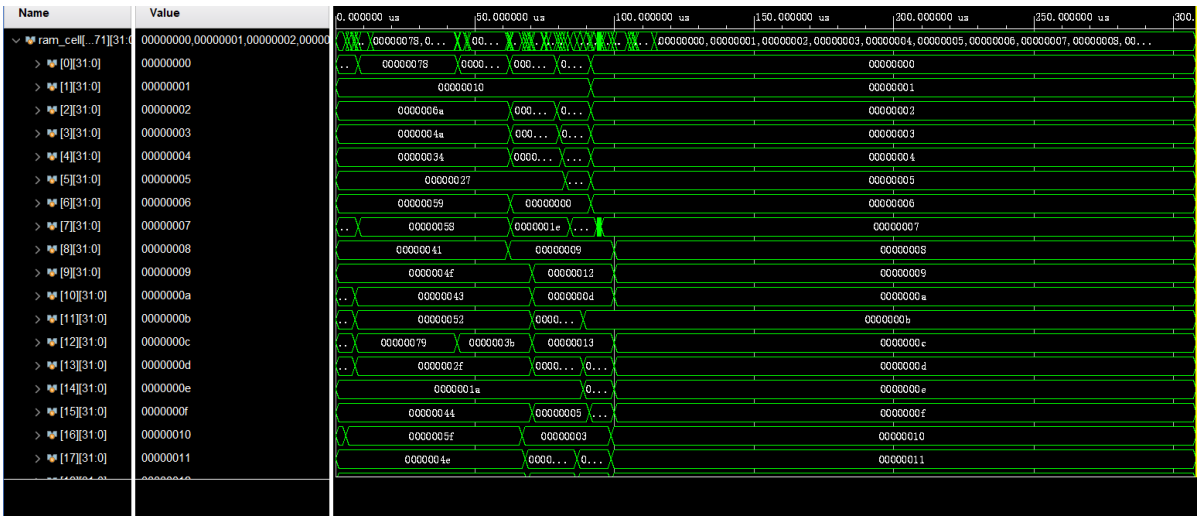
四、运行测试结果(以LRU为例子)

缓存的单独测试：



在实验要求中，validation_count递增并最后到达ffffffff就是成功运行，通过波形图可见cache是正确的。

运行快排程序：



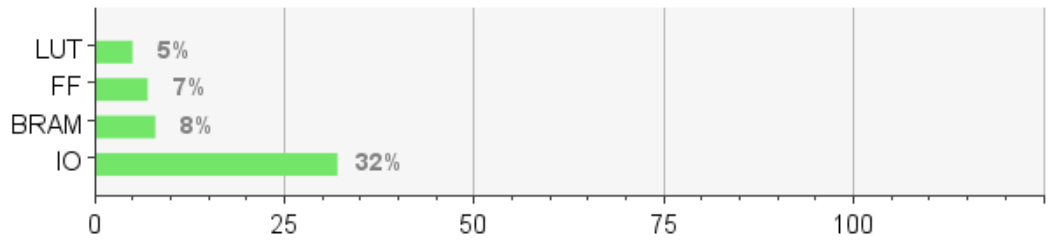
我们可以看到，ram_cell代表的是数据内存，经过一段时间之后，内存的数据按照顺序排序，可见cache与接入CPU是正确的。

五、对cache不同参数的修改与分析

5.1 对不同参数的cache进行资源消耗评估

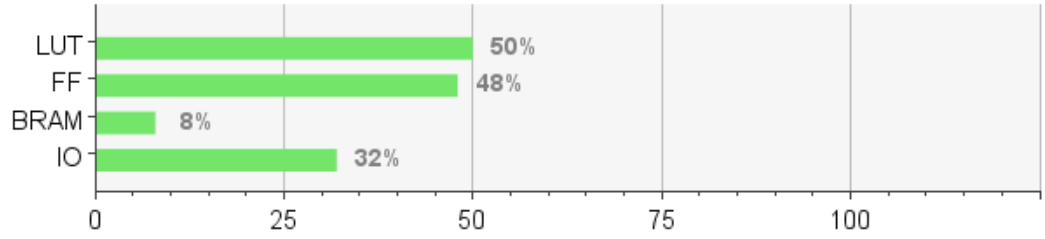
直接相连cache资源占用报告

Resource	Utilization	Available	Utilization %
LUT	1128	20800	5.42
FF	3008	41600	7.23
BRAM	4	50	8.00
IO	81	250	32.40



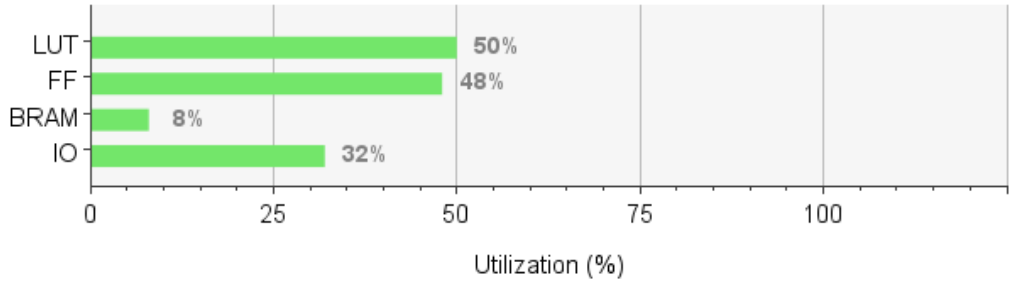
FIFO策略cache资源占用报告

Resource	Utilization	Available	Utilization %
LUT	10388	20800	49.94
FF	19921	41600	47.89
BRAM	4	50	8.00
IO	81	250	32.40



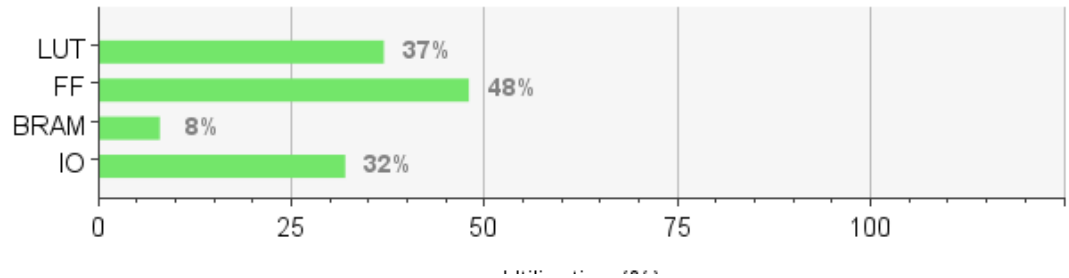
LRU策略cache资源占用报告

Resource	Utilization	Available	Utilization %
LUT	10388	20800	49.94
FF	19921	41600	47.89
BRAM	4	50	8.00
IO	81	250	32.40



LFU策略cache资源占用报告

Resource	Utilization	Available	Utilization %
LUT	7719	20800	37.11
FF	19887	41600	47.81
BRAM	4	50	8.00
IO	81	250	32.40



不同参数与策略cache的电路资源消耗：

策略	LINE_SIZE	SET_SIZE	WAY_SIZE	LUT	FF
LRU	8	8	8	10388	19921
LRU	4	8	8	6834	11322
LRU	8	4	8	5801	10423
LRU	8	8	4	5498	10403
LFU	8	8	8	7719	19887
LFU	4	8	8	4538	11310
LFU	8	4	8	7170	10417
LFU	8	8	4	4285	10410

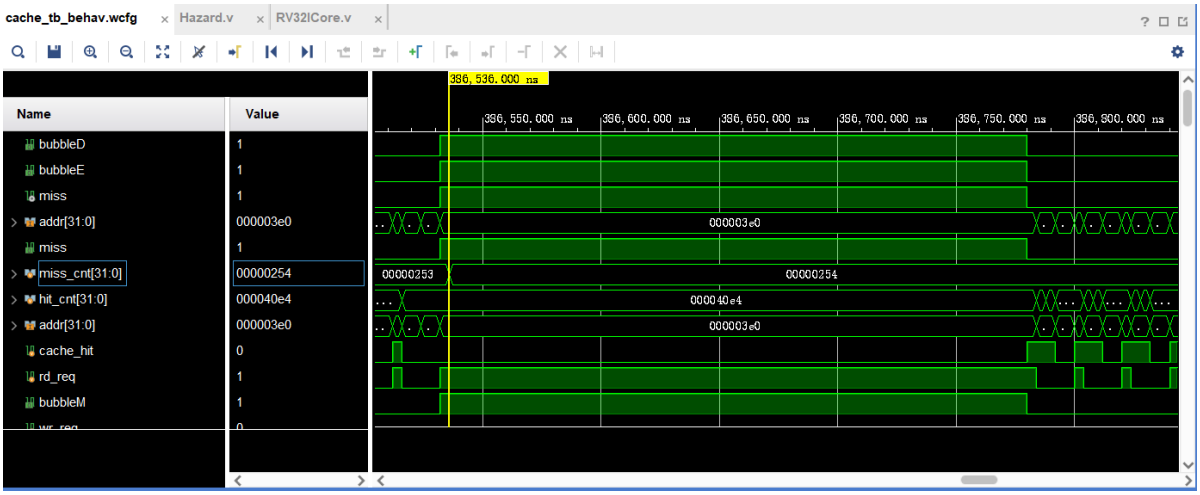
结论：各种size的增加大队电路资源消耗都有较大影响，且不同参数对资源消耗的影响占有的比重类似。

5.2 统计不同的cache的命中率

统计思路：

添加miss_cnt和hit_cnt两个变量，在IDLE状态中通过是否命中对两个变量进行自增操作。

运行结果：



可以看到miss_cnt和hit_cnt在miss和hit分别自增。可说明运行正确。

不同参数与策略cache的命中率统计：

快排：（256）

策略	LINE_SIZE	SET_SIZE	WAY_SIZE	Hit Count	Miss Count	命中率
LRU	8	8	8	152	4933	0.982
LRU	4	8	8	2b3	3ea0	0.959
LRU	8	4	8	22c	4129	0.967
LRU	8	8	4	152	4933	0.982

策略	LINE_SIZE	SET_SIZE	WAY_SIZE	Hit Count	Miss Count	命中率
LFU	8	8	8	152	4933	0.982
LFU	4	8	8	2b3	3ea0	0.959
LFU	8	4	8	22c	4129	0.967
LFU	8	8	4	152	4933	0.982
LFU	8	8	2	152	4933	0.982
LFU	8	8	1	152	4933	0.982

矩阵相乘：（16*16)

策略	LINE_SIZE	SET_SIZE	WAY_SIZE	Hit Count	Miss Count	命中率
LRU	8	8	8	1400	3349	0.719
LRU	4	8	8	1420	3235	0.714
LRU	8	4	8	15c0	313d	0.694
LRU	8	8	4	1400	3349	0.719
LFU	8	8	8	1400	3349	0.719
LFU	4	8	8	1420	3235	0.714
LFU	8	4	8	15c0	313d	0.694
LFU	8	8	4	1400	3349	0.719

结论：

LRU策略和LFU策略命中率几乎一样，因为逻辑十分相似。

组相联程度在较小规模的快排和矩阵相乘对命中率影响几乎没有，其他参数影响较小。