

浙江大学

本科实验报告

课程名称：计算机体系结构

姓 名：郑乔尹

学 院：计算机科学与技术学院

系：计算机科学与技术系

专 业：计算机科学与技术

学 号：3210104169

指导教师：姜晓红

2023 年 11 月 6 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: Cache Design

学生姓名: 郑乔尹 专业: 计算机科学与技术 学号: 3210104169

同组学生姓名: 无 指导老师: 姜晓红

实验地点: 曹西 301 实验日期: 2023 年 11 月 6 日

20% bonus: 4-way set associative

一、实验目的和要求

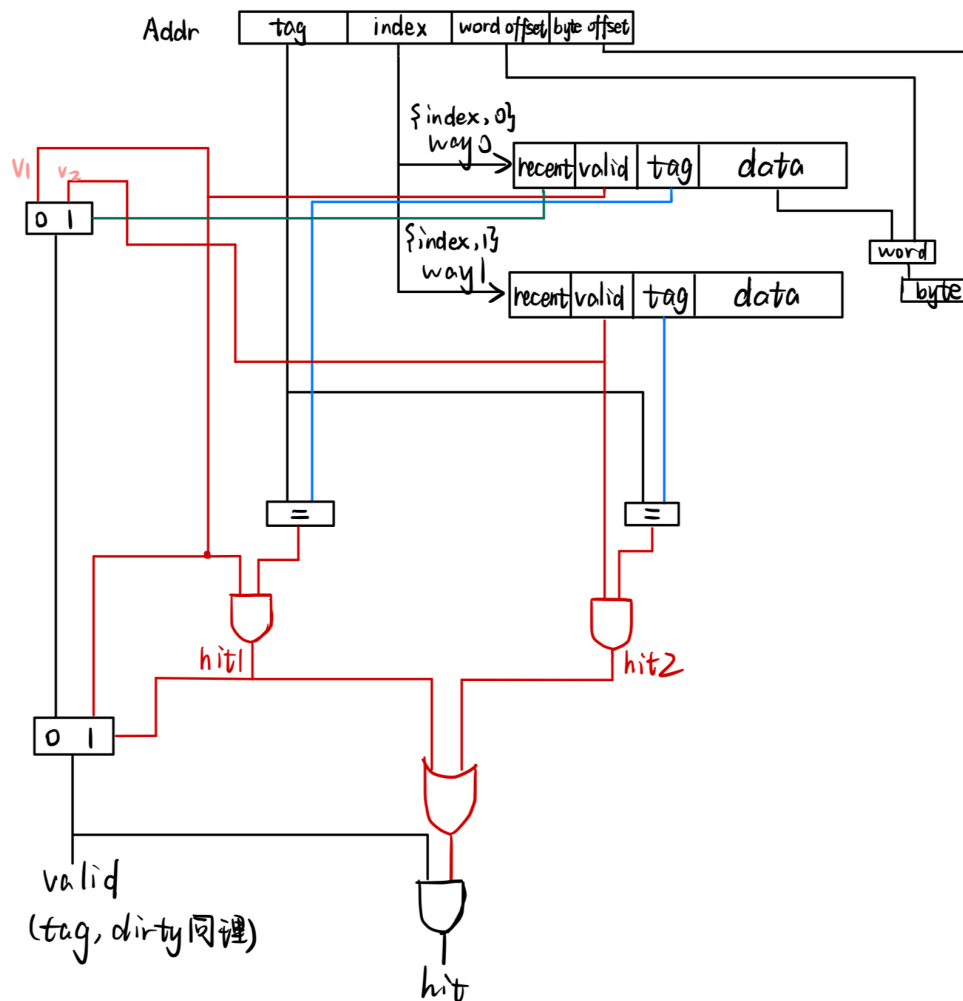
Task 1: 写出本次实验的目的与要求 (5 points)

1. 理解 Cache line
2. 掌握 Cache line 的设计方法
3. 掌握 Cache line 的验证方法

二、实验内容和原理

Task 2: 简要画出本次实验实现的 cache 内部组成与结构, 不需要画出连线。

(可以使用 PPT 上的线路图进行修改, 但是必须和自己的实现保持一致。如果不一致, 本题将不给分) (10 points)



Task 3: 请给出 5 个输出引脚的代码并加以解释 (20 points)

以下是 valid 信号的输出，当组内第一路命中时，输出 valid1 的值，当组内第二路命中时，输出 valid2 的值，当都未命中时，根据 LRU bit 确定输出哪一路的 valid 信号——假如 recent1 为 1，代表第二路将被替换，输出 valid2；否则输出 valid1。

```
assign valid = hit1 ? valid1 : hit2 ? valid2 : recent1 ? valid2 : recent2 ?
valid1 : 0;
```

以下是 hit 信号的输出，如果 valid 信号为 0，不可能 hit；如果 valid 信号为 1，则检查是否有一路 hit，如果有，则 hit 置 1：

```
assign hit = valid && (hit1 || hit2);
```

以下是 dirty 信号的输出，与 valid 信号类似，根据两路的 hit 信号以及 LRU bit 决定输出哪一路的 dirty 信息：

```
assign dirty = hit1 ? dirty1 : hit2 ? dirty2 : recent1 ? dirty2 : recent2 ?
dirty1 : 0;
```

以下是 tag 信号的输出, 与 valid 信号类似, 根据对应路的 hit 信号以及 LRU bit 决定输出的是哪一路的 tag 信息:

```
assign tag = hit1 ? tag1 : hit2 ? tag2 : recent1 ? tag2 : tag1;
```

以下是 dout 信号的输出, 根据 load 时的两路 cache 的命中状况以及 u_b_h_w 信号决定输出结果:

```
assign dout = (load && hit1) ?
    (u_b_h_w[1] ? word1 :
        (u_b_h_w[0] ?
            {u_b_h_w[2] ? 16'b0 : {16{half_word1[15]}}, half_word1} :
            {u_b_h_w[2] ? 24'b0 : {24{byte1[7]}}, byte1})) :
    (load && hit2) ?
    (u_b_h_w[1] ? word2 :
        (u_b_h_w[0] ?
            {u_b_h_w[2] ? 16'b0 : {16{half_word2[15]}}, half_word2} :
            {u_b_h_w[2] ? 24'b0 : {24{byte2[7]}}, byte2})) :
    inner_data[ recent1 ? addr_word2 : addr_word1 ];
```

需要 load 时, 检查是否命中, 如果第一路命中, 则根据 u_b_h_w 输出需要的数据 (u_b_h_w[1]为 1, 输出对应 word; u_b_h_w[1]为 0, 检查 u_b_h_w[0], 为 1 则根据 u_b_h_w[2]输出符号拓展(u_b_h_w[2]=0)或无符号(u_b_h_w[2]=1)的 half word, 否则输出符号拓展或无符号的 byte), 第二路命中同理; 若当前无命中, 则输出需要替换的 word 值。

Task 4: 请给出 store 修改缓存内部状态的逻辑、代码以及解释 (5 points)

当需要 store 时, 命中则写入, 否则取消写入。命中时根据 u_b_h_w 信息确定写入数据的位宽, 再根据 word offset 和 byte offset 决定写入数据在当前 cache line 中的具体位置。写入后, 更新当前 cache line 为 dirty, 更新 LRU bit——比如第一路命中, 写入第一路后, 第一路对应 cache line 的 inner_recent 设置为 1, 同时将第二路对应 cache line 的 inner_recent 设置为 0, 代表第一路最近被使用。

```
if (store) begin
    if (hit1) begin
        inner_data[addr_word1] <=
            u_b_h_w[1] ?          // word?
                din
            :
            u_b_h_w[0] ?          // half word?
                addr[1] ?          // upper / lower?
                    {din[15:0], word1[15:0]}
            :
    end
end
```

```

        {word1[31:16], din[15:0]}
    :    // byte
        addr[1] ?
            addr[0] ?
                {din[7:0], word1[23:0]}    // 11
            :
                {word1[31:24], din[7:0], word1[15:0]}
// 10
        :
            addr[0] ?
                {word1[31:16], din[7:0],
word1[7:0]}    // 01
        :
            {word1[31:8], din[7:0]} // 00
    ;
    inner_dirty[addr_element1] <= 1'b1;
    inner_recent[addr_element1] <= 1'b1;
    inner_recent[addr_element2] <= 1'b0;
end
else if (hit2) begin
    //need to fill in
    inner_data[addr_word2] <=
        u_b_h_w[1] ?            // word?
            din
        :
            u_b_h_w[0] ?        // half word?
                addr[1] ?        // upper / lower?
                    {din[15:0], word2[15:0]}
                :
                    {word2[31:16], din[15:0]}
        :    // byte
            addr[1] ?
                addr[0] ?
                    {din[7:0], word2[23:0]}    // 11
                :
                    {word2[31:24], din[7:0], word2[15:0]}
// 10
            :
                addr[0] ?
                    {word2[31:16], din[7:0],
word2[7:0]}    // 01
            :
                {word2[31:8], din[7:0]} // 00
    ;

```

```

        inner_dirty[addr_element2] <= 1'b1; // set dirty
        inner_recent[addr_element1] <= 1'b0;
        inner_recent[addr_element2] <= 1'b1; // set recent / LRU
    end
end

```

Task 5: 请给出 replace 的逻辑、代码以及解释，只需要说明与上一题内容的不同之处 (5 points)

replace 与 store 的不同之处在于，它替换的是整个 cache line（但是本实验中是通过 4 次[word_offset_width=2]word 替换实现的整个 cache line 的替换），当未命中时需要根据 LRU bit 进行替换 cache line 的选择——我代码中的逻辑是 recent 值如果为 1，对应着最近被使用，故倘若 recent1 为 1，则替换第二路，反之替换第一路，如果两路都没被使用过，也替换第一路。值得注意的是，整个 cache line 替换完，它是干净的，dirty 位需要置 1。

```

if (replace) begin
    if (hit1) begin
        inner_data[addr_word1] <= din;
        inner_valid[addr_element1] <= 1'b1;
        inner_dirty[addr_element1] <= 1'b0;
        inner_tag[addr_element1] <= addr_tag;
        inner_recent[addr_element1] <= 1'b1;
        inner_recent[addr_element2] <= 1'b0;
    end
    else if (hit2) begin
        inner_data[addr_word2] <= din;
        inner_valid[addr_element2] <= 1'b1;
        inner_dirty[addr_element2] <= 1'b0;
        inner_tag[addr_element2] <= addr_tag;
        inner_recent[addr_element1] <= 1'b0;
        inner_recent[addr_element2] <= 1'b1;
    end
    else if (recent1) begin // replace 2
        inner_data[addr_word2] <= din;
        inner_valid[addr_element2] <= 1'b1;
        inner_dirty[addr_element2] <= 1'b0;
        inner_tag[addr_element2] <= addr_tag;
        inner_recent[addr_element1] <= 1'b0;
        inner_recent[addr_element2] <= 1'b1;
    end else begin
        // recent2 == 1 => replace 1
        // recent2 == 0 => no data in this set, place to 1
    end
end

```

```

        inner_data[addr_word1] <= din;
        inner_valid[addr_element1] <= 1'b1;
        inner_dirty[addr_element1] <= 1'b0;
        inner_tag[addr_element1] <= addr_tag;
        inner_recent[addr_element1] <= 1'b1;
        inner_recent[addr_element2] <= 1'b0;
    end
end

```

Task 6: 请给出 LRU 的逻辑、代码以及解释(10+10 points)

LRU：通过一个 `inner_recent` 位来记录每一路的最近使用情况，如果一路 cache line 最近被使用了，那么它对应的 `inner_recent` 将被设置为 1，同时需要将另一路设置为 0，代表另一路的上次使用时间更早，作为接下来被替换的对象。由于这是两路组关联，故替换时只需检测对应 cache line 的 `inner_recent` 值即可，为 0 即为被替换。

以下是 `recent1`、`2` 的信号：

```

assign recent1 = inner_recent[addr_element1];
assign recent2 = inner_recent[addr_element2];

```

以 `replace` 中的替换为例：

```

if (replace) begin
    if (hit1) begin
        inner_data[addr_word1] <= din;
        inner_valid[addr_element1] <= 1'b1;
        inner_dirty[addr_element1] <= 1'b0;
        inner_tag[addr_element1] <= addr_tag;
        inner_recent[addr_element1] <= 1'b1;
        inner_recent[addr_element2] <= 1'b0;
    end
    else if (hit2) begin
        inner_data[addr_word2] <= din;
        inner_valid[addr_element2] <= 1'b1;
        inner_dirty[addr_element2] <= 1'b0;
        inner_tag[addr_element2] <= addr_tag;
        inner_recent[addr_element1] <= 1'b0;
        inner_recent[addr_element2] <= 1'b1;
    end
    else if (recent1) begin // replace 2
        inner_data[addr_word2] <= din;
        inner_valid[addr_element2] <= 1'b1;
        inner_dirty[addr_element2] <= 1'b0;
        inner_tag[addr_element2] <= addr_tag;
    end
end

```

```

        inner_recent[addr_element1] <= 1'b0;
        inner_recent[addr_element2] <= 1'b1;

    end else begin
        // recent2 == 1 => replace 1
        // recent2 == 0 => no data in this set, place to 1
        inner_data[addr_word1] <= din;
        inner_valid[addr_element1] <= 1'b1;
        inner_dirty[addr_element1] <= 1'b0;
        inner_tag[addr_element1] <= addr_tag;
        inner_recent[addr_element1] <= 1'b1;
        inner_recent[addr_element2] <= 1'b0;

    end
end

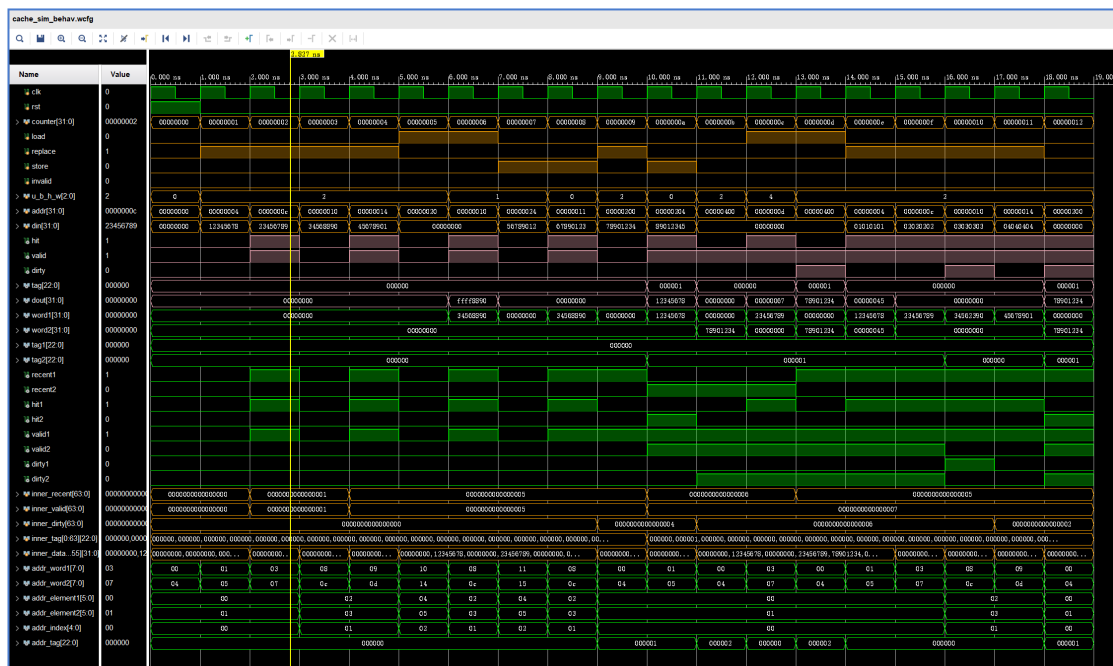
```

三、实验过程和数据记录及结果分析

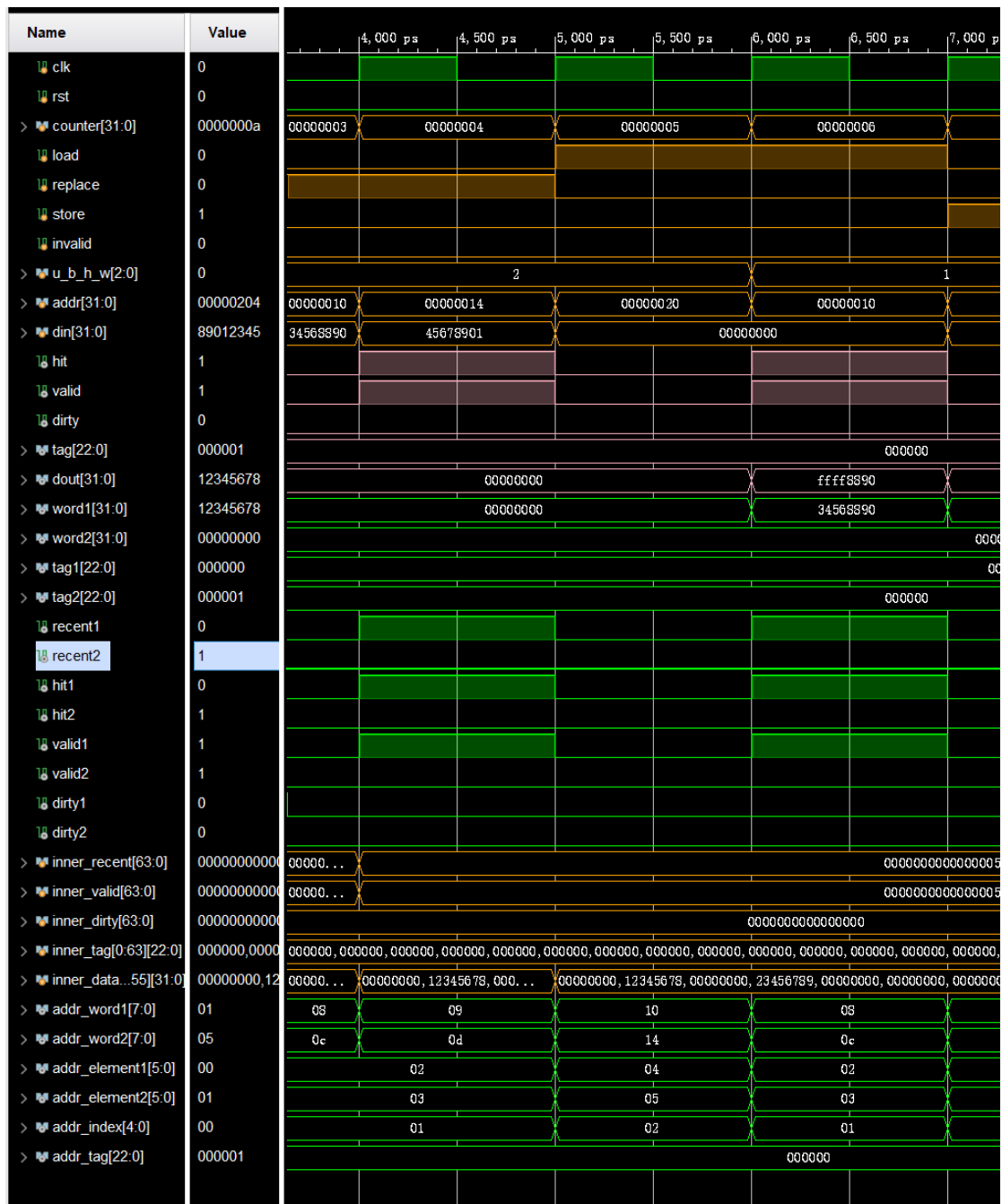
仿真图片应完整包含时间信息和信号名称。

对仿真的解释示例：XXXns，X 信号变为 X，由于 XXX，导致 X 信号变为 XXX，……，我们发现 X 被 forward 到了 X。

Task 7: 请给出本次实验仿真的完整截图 (5 points)



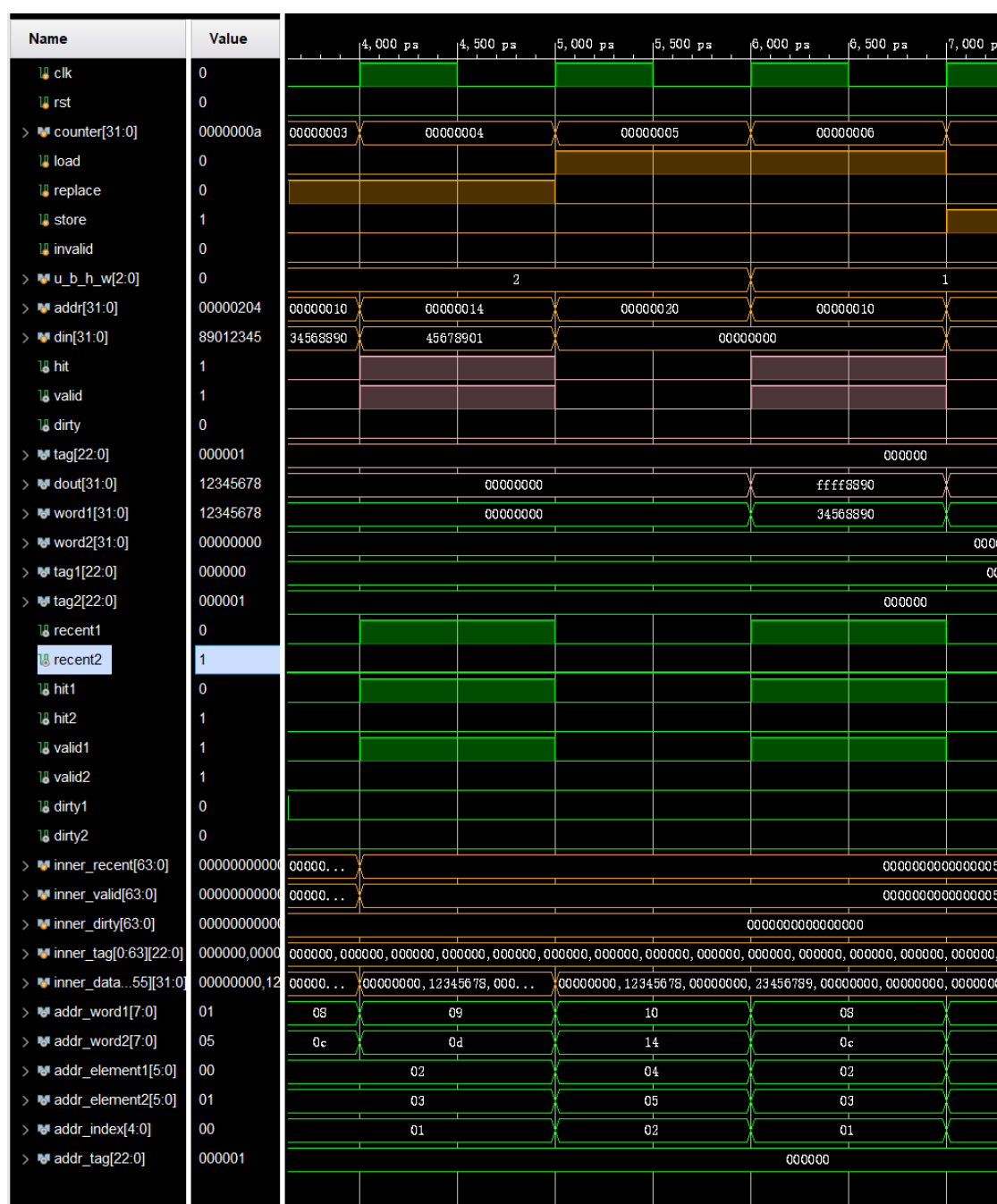
Task 8: 请给出一个 load hit 部分仿真的高清图片，并对涉及到的信号加以详细解释 (5 points)



图中 cycle6(counter=6, 6000ps 处), 请求 load 地址 0x10, 由于 word offset 与 byte offset 共四位, index 共 5 位, 可知 index=0x1(图中 addr_index 信号), addr_word1=8(根据 word_offset 计算得到, 用于在 cache line 中定位对应的 word), hit1=1(第一路命中), 这里需要检查一下前面写入第一路的数据 (查看 cycle1-4 中写入的 cache line 中的 inner_data[8]) 是否为此时 dout 中输出的数据, 可以看到都为 0x34568890, 可以说明对于 load hit 的实现是正确的。

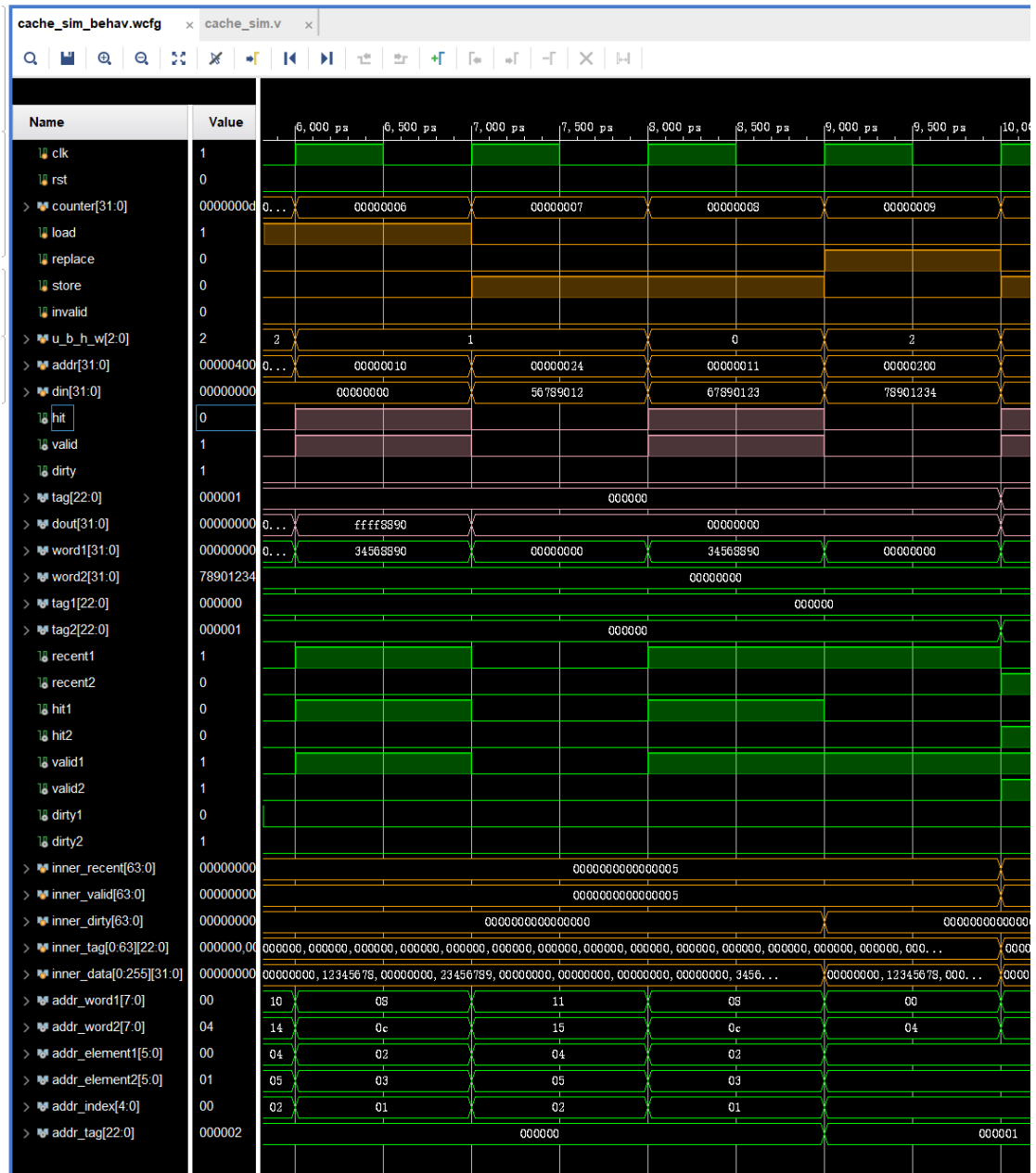
Task 9: 请给出一个 load miss 部分仿真的高清图片, 并对涉及到的信号加以

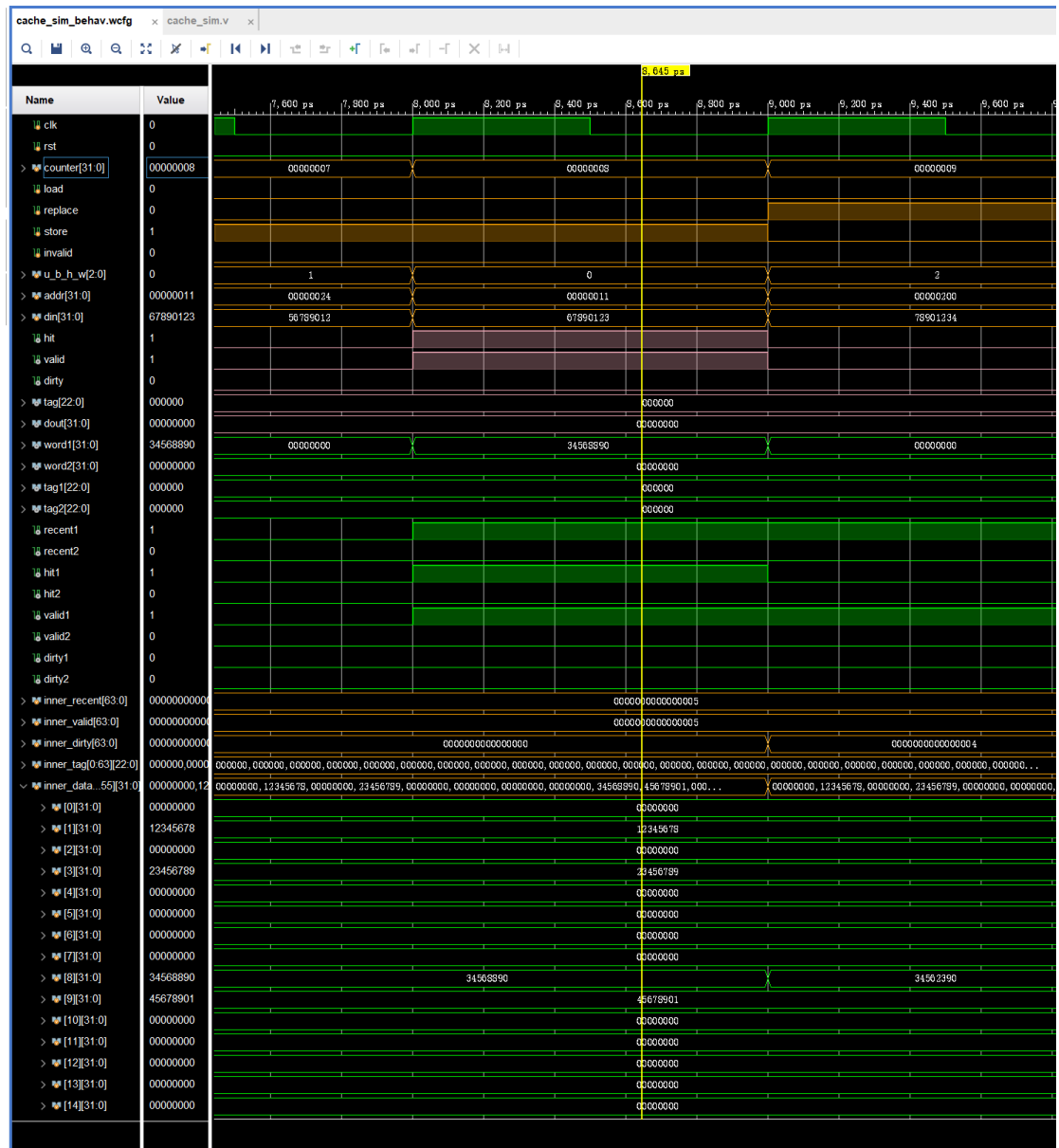
详细解释 (5 points)



图中 cycle5(counter=5, 5000ps 处), 请求 load 地址 0x20, index=0x1(图中 addr_index 信号), addr_word1=10(根据 word_offset 计算得到, 用于在 cache line 中定位对应的 word), hit=0(没有任何一路 hit, load miss), dout=0, 没有读出有效数据, 可以说明对于 load miss 的实现是正确的。

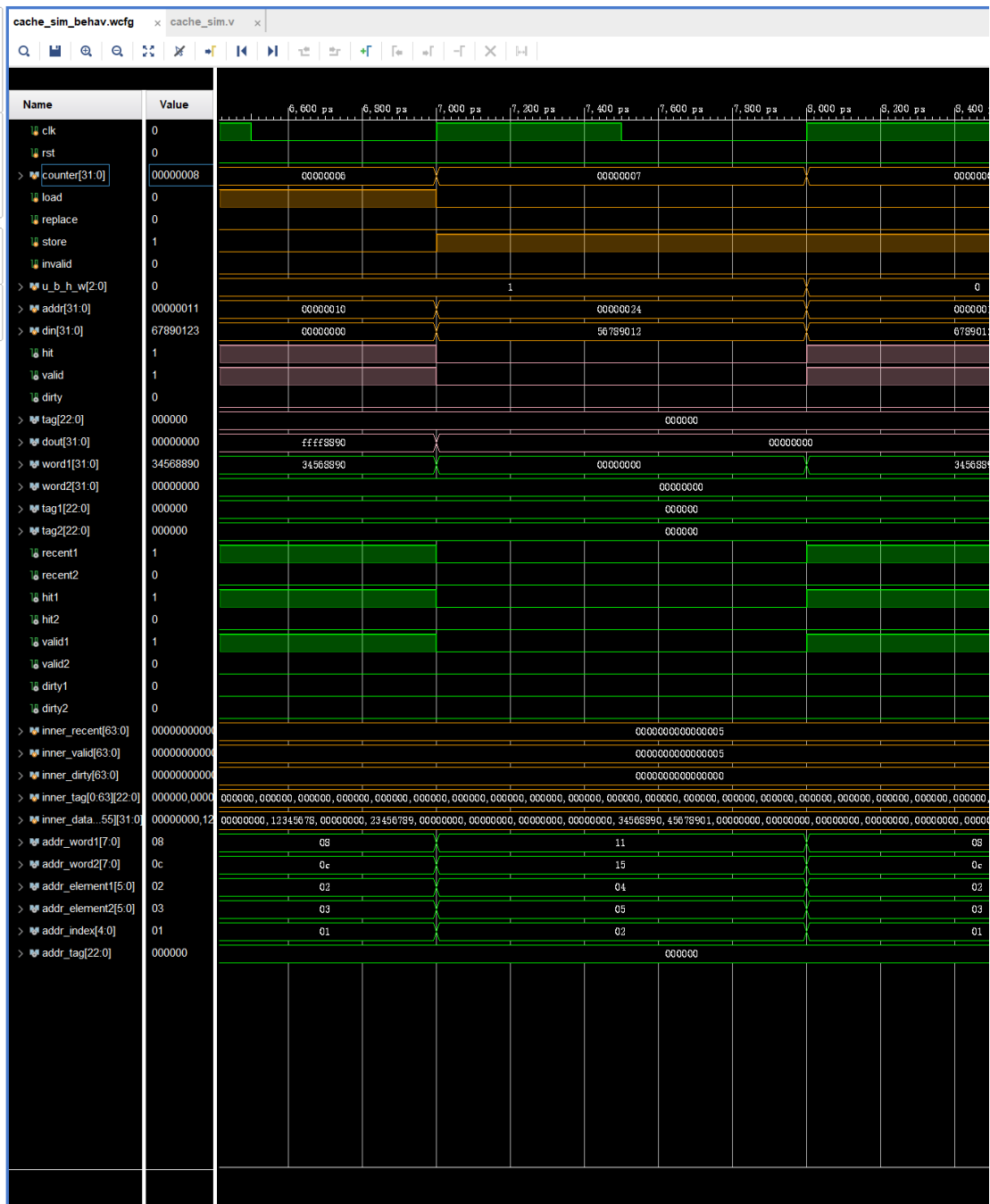
Task 10: 请给出一个 write hit 部分仿真的高清图片, 并对涉及到的信号加以详细解释 (5 points)

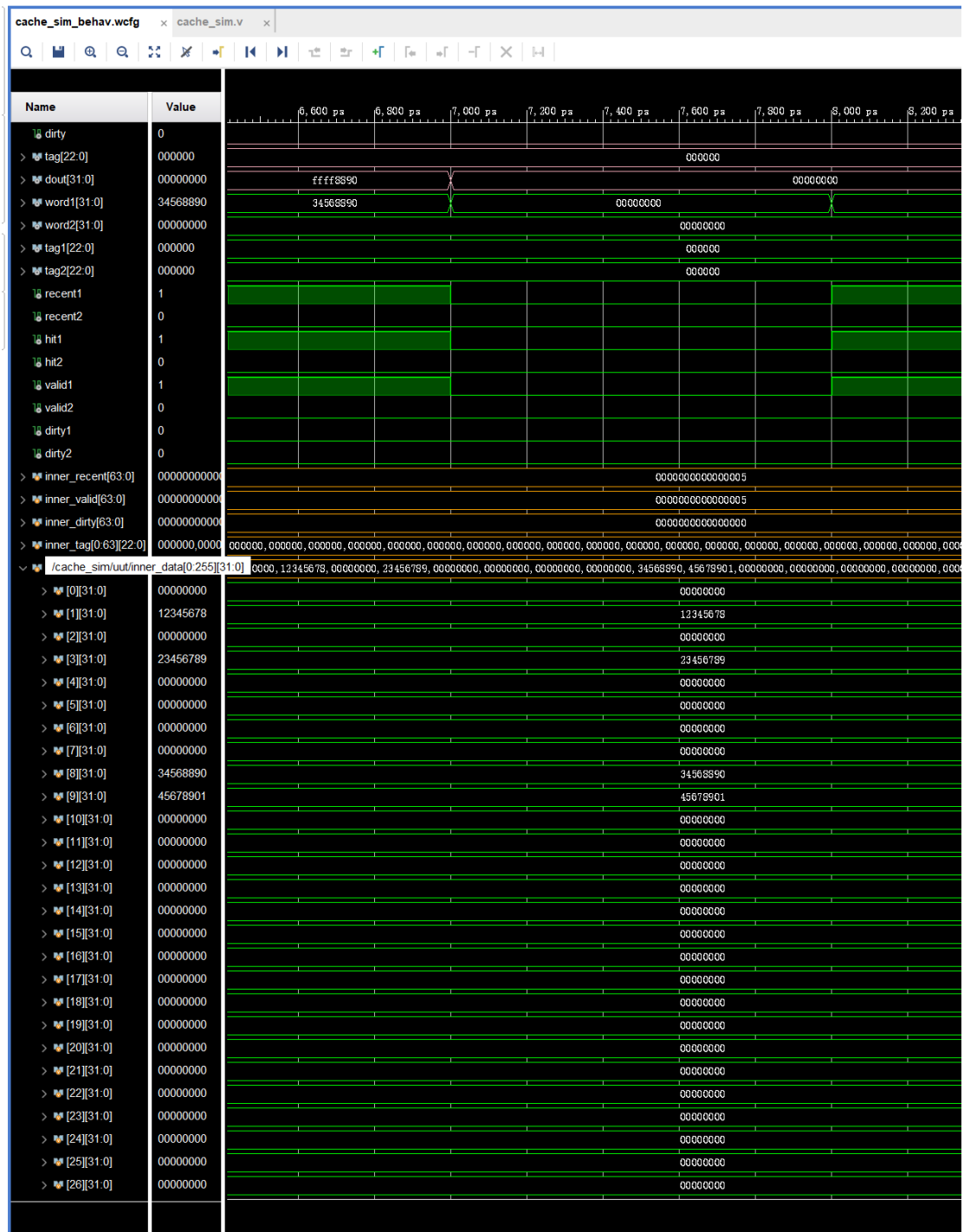




Cycle8 (counter=0x8, 8000ps 处) hit (hit=1,hit1=1,valid1=1), 请求 store 地址 0x11, 写入一个 byte(u_b_h_w=0), index=1 (addr_index=1), byte_offset=1,word_offset=0, 写入第二个字节, 可以看到 inner_data[8] (addr word1=8) 从 34568890 变为了 34562390, 正确实现。

Task 11: 请给出一个 write miss 部分仿真的高清图，并对涉及到的信号加以详细解释 (5 points)

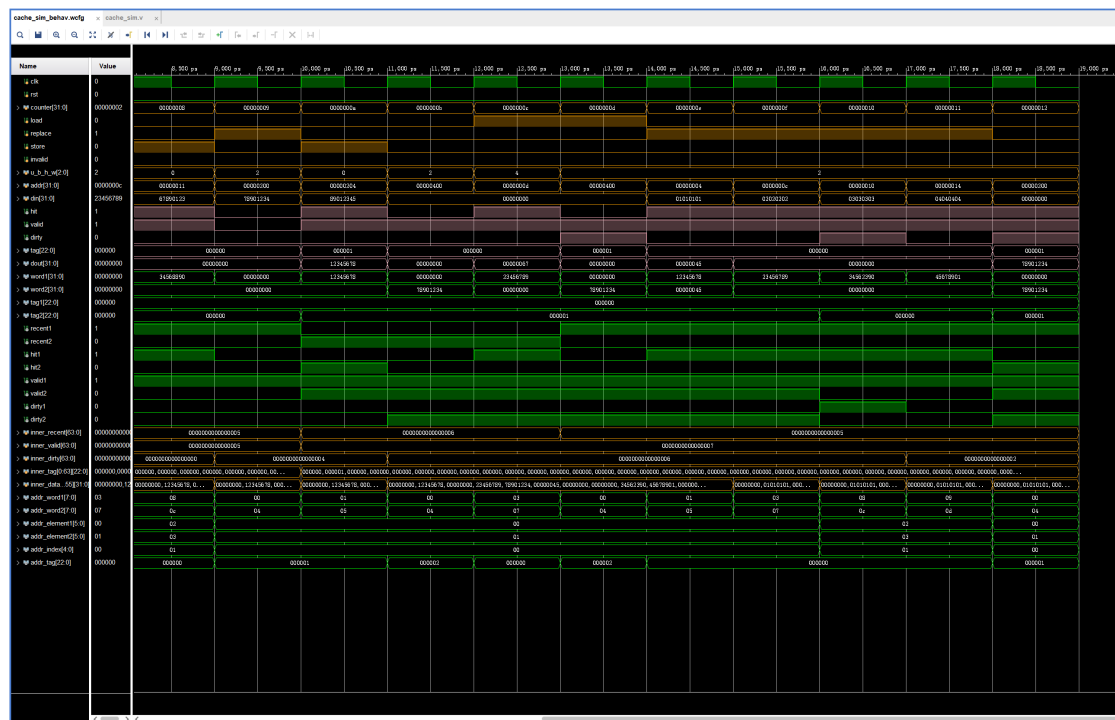




Cycle7(counter=0x7, 7000ps 处), store miss (hit=0), 应该停止写入, 通过观察该周期与下一周期的上升沿 (8000ps 处) 的 inner_data[addr_word1/2]值是否发生变化可以判断是否正确实现, 由于 addr_index 为 2, word_offset=1, 故 addr_word1={10, 0, 01}=0x11, addr_word2={10, 1, 01}=0x15, 而对应的 inner_data[11/15]并没有发生变化, 说明正确处理了 store miss 的情况。

Task 12: 请给出一个 replace 一个 valid 的 cache line 部分仿真的高清图片(给

的代码中没有这个), 并对涉及到的信号加以详细解释 (10 points)



在仿真代码末端添加以下代码:

```
32'd14: begin
    rst <= 0;
    load <= 0;
    store <= 0;
    replace <= 1;
    invalid <= 0;
    addr <= 32'h4;
    din <= 32'h01010101;
    u_b_h_w <= 2;
end
```

```
32'd15: begin
    load <= 0;
    store <= 0;
    replace <= 1;
    invalid <= 0;
    addr <= 32'hc;
    din <= 32'h02020202;
    u_b_h_w <= 2;
end
```

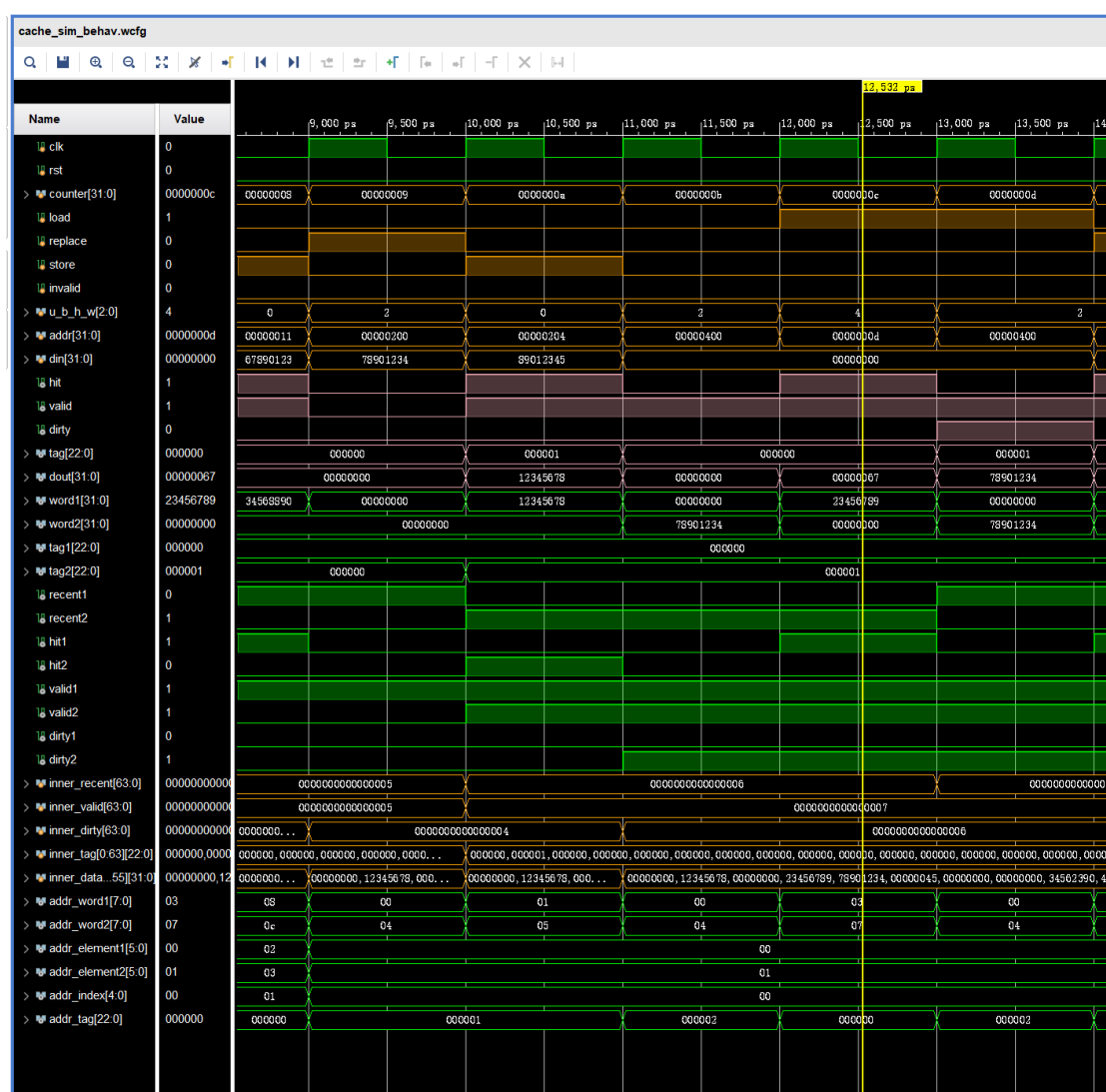
```
32'd16: begin
    load <= 0;
```

```
32'd17: begin
    load <= 0;
    store <= 0;
    replace <= 1;
    invalid <= 0;
    addr <= 32'h14;
    din <= 32'h04040404;
    u_b_h_w <= 2;
end
```

[illegible]

(hit1=1)，从结果上看，inner_data[addr_word1]中的数据被替换，也就是第一路被替换，但实际上第一路是最近使用的(recent1=1)，说明在 cache hit 的情况下，正确实现了直接替换 hit 的 cache line 而不管 LRU 的结果。

Task 13: 请给出一个 LRU 改变后 输出的待替换的 cache line 变化 部分仿真的高清图片，并对涉及到的信号加以详细解释 (10+10 points)



Cycle11-cycle13(counter=0xb-0xd , 11000ps-13000ps) , 指令分别为无/load/load, 同时 cycle11 与 cycle13 的要求地址相同并且都 miss (hit=0), 然而 cycle11 的 dout 为 word1, 说明访问的是第一路, 即, 但是看 cycle13 可知, 由于 cycle12 的 load hit, 现在第一路为最近访问, 根据 LRU 算法, 此次访问了第二路, 可以看到 dout 为 word2 的值, 代表这次访问分配到的是第二路, 与 cycle11 的 load miss 结果不同, 说明 LRU 算法是正确实现的。

四、 讨论与心得

Task 14: 请写出对本次实验内容的深入讨论，或者本次实验的心得体会。例如遇到的难题等等。请认真填写本模块，若不填写或胡乱填写将酌情扣分，写明真实情况即可。 (+10 points)

1. 一开始将输出改为组合逻辑的时候，遇到了一点小问题，dout 在使用三元表达式时总是在最后一项报错，最后发现是忘记写最后一个“:”，给最后一个 else 中的 dout 赋 0 解决此问题。
2. 在编写 replace 逻辑时，错误的将整个 cache line 的替换也当作造成 cache line dirty 的情况，回顾 cache 的设计后发现问题所在。