浙江水学

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						

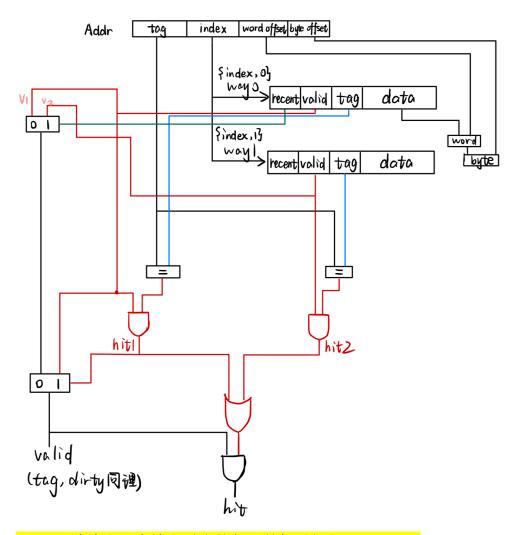
2023年 11 月 6 日

浙江大学实验报告

课程名称	尔:计	算机体	系结构	实验类型:综合							
实验项目	目名称: <u></u>			Cache Design							
学生姓名	名: <u>郑</u>	7乔尹	专业:_	计算机	科学与技术	_学号:		3210104	1169		
同组学生姓名:			无			· 导老师:					
实验地点:			曹西 30	1	_实验日期:	2023	年 _	<u>11</u> 月 <u>6</u> 日			
20% bon	nus: 4-way	y set asso	ciative								
— `,	实验目的	刀和要求									
Tas ²	k 1: 写出	本次实验	<mark>俭的目的</mark>	与要求(5 points)						
1.	理解 Cac	he line									
2.	掌握 Cache line 的设计方法										
3.	掌握 Cache line 的验证方法										
<u> </u>	实验内容	8和原理									
Tasi	k 2: 简要	画出本	欠实验实	现的 cacl	ne 内部组成-	与结构,	不是	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:

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。

```
{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]}
                               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;</pre>
                inner_recent[addr_element1] <= 1'b1;</pre>
                inner_recent[addr_element2] <= 1'b0;</pre>
           else if (hit2) begin
                inner_data[addr_word2] <=</pre>
                    u_b_h_w[1] ?
                        din
                       u_b_h_w[0] ? // half word?
                           addr[1] ?
                               {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]}
                               addr[0] ?
                                   {word2[31:16], din[7:0],
word2[7:0]}
                                   {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</pre>
```

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 recent 1 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;</pre>
                 inner valid[addr element1] <= 1'b1;</pre>
                 inner_dirty[addr_element1] <= 1'b0;</pre>
                 inner_tag[addr_element1] <= addr_tag;</pre>
                 inner_recent[addr_element1] <= 1'b1;</pre>
                 inner recent[addr element2] <= 1'b0;</pre>
             else if (hit2) begin
                 inner_data[addr_word2] <= din;</pre>
                 inner_valid[addr_element2] <= 1'b1;</pre>
                 inner dirty[addr element2] <= 1'b0;</pre>
                 inner_tag[addr_element2] <= addr_tag;</pre>
                 inner_recent[addr_element1] <= 1'b0;</pre>
                 inner_recent[addr_element2] <= 1'b1;</pre>
             else if (recent1) begin // replace 2
                 inner_data[addr_word2] <= din;</pre>
                 inner valid[addr element2] <= 1'b1;</pre>
                 inner_dirty[addr_element2] <= 1'b0;</pre>
                 inner_tag[addr_element2] <= addr_tag;</pre>
                 inner_recent[addr_element1] <= 1'b0;</pre>
                 inner_recent[addr_element2] <= 1'b1;</pre>
```

```
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</pre>
```

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_rencent 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;</pre>
                 inner_valid[addr_element1] <= 1'b1;</pre>
                 inner_dirty[addr_element1] <= 1'b0;</pre>
                 inner_tag[addr_element1] <= addr_tag;</pre>
                 inner_recent[addr_element1] <= 1'b1;</pre>
                 inner recent[addr element2] <= 1'b0;</pre>
            else if (hit2) begin
                 inner_data[addr_word2] <= din;</pre>
                 inner_valid[addr_element2] <= 1'b1;</pre>
                 inner_dirty[addr_element2] <= 1'b0;</pre>
                 inner_tag[addr_element2] <= addr_tag;</pre>
                 inner recent[addr element1] <= 1'b0;</pre>
                 inner_recent[addr_element2] <= 1'b1;</pre>
            else if (recent1) begin // replace 2
                 inner_data[addr_word2] <= din;</pre>
                 inner valid[addr element2] <= 1'b1;</pre>
                 inner_dirty[addr_element2] <= 1'b0;</pre>
                 inner_tag[addr_element2] <= addr_tag;</pre>
```

```
inner_recent[addr_element1] <= 1'b0;
    inner_recent[addr_element2] <= 1'b1;
end else begin

    // recent2 == 1 => replace 1

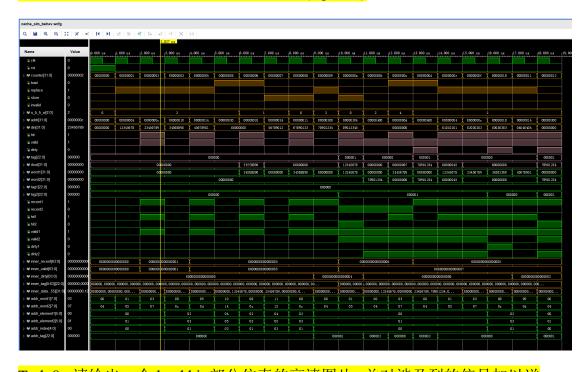
    // recent2 == 0 => no data in this set, place to 1
    inner_data[addr_word1] <= din;
    inner_valid[addr_element1] <= 1'b1;
    inner_dirty[addr_element1] <= 1'b0;
    inner_tag[addr_element1] <= addr_tag;
    inner_recent[addr_element1] <= 1'b1;
    inner_recent[addr_element2] <= 1'b0;
end
end</pre>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

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

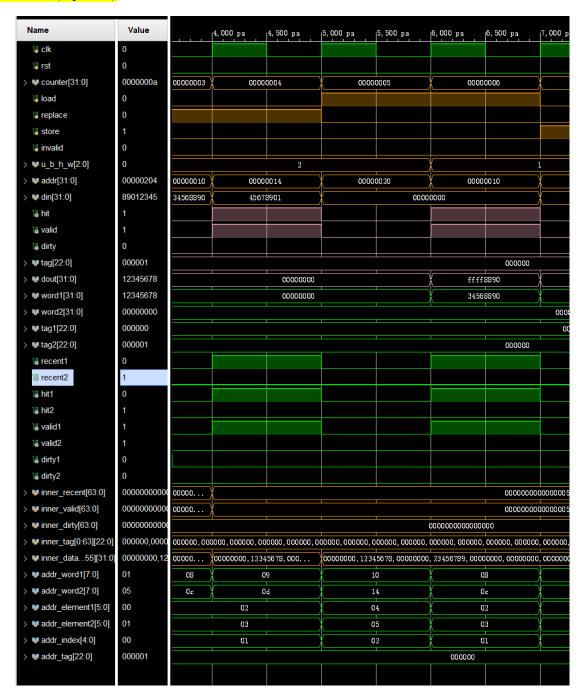
细解释 (5 points)

Name	Value		4,000 ps	4, 500 ps	5,000 ps	5,500 ps	6,000 ps	6,500 ps	7, 000
₩ clk	0								
₩ rst	0								
> W counter[31:0]	0000000a	00000003	000	00004	0000	00005	0000	00006	
₩ load	0								
↓ replace	0								
↓ store	1								
₩ invalid	0								
> ₩ u_b_h_w[2:0]	0			2			<u> </u>		1
> W addr[31:0]	00000204	00000010	000	00014	0000	00020	0000	$\overline{\chi}$	
> 😽 din[31:0]	89012345	34568890	456	78901	∤	0000	10000		7
18 hit	1				1				
₩ valid	1								
18 dirty	0								
> ₩ tag[22:0]	000001							000000	
> * dout[31:0]	12345678			00000000			fff	8890	\neg
> W word1[31:0]	12345678			00000000				38890	∜─
> W word2[31:0]	00000000						Λ		
> w tag1[22:0]	000000								
> w tag2[22:0]	000001							000000	
¼ recent1	0								
₩ recent2	1								
18 hit1	0								
18 hit2	1								
¹∄ valid1	1								
18 valid2	1								
18 dirty1	0								
la dirty2	0								
> w inner_recent[63:0]	000000000000	00000	V					0000000	กกกกกก
> w inner_valid[63:0]	00000000000	00000	↓ ——					0000000	
> • inner_dirty[63:0]	0000000000	00000	<u> </u>				00000000000000		1000000
> w inner_tag[0:63][22:0]	000000,0000	000000 00	nnnn nnnnnn r	, 000000, 000000, 000000, 000000, 0000					
> w inner_data55][31:0]	000000,0000	000000			J				
	01	08	00000000,12345678,000		10		08		, 000k
> • addr_word1[7:0]	05	0a 0c	09		()————————————————————————————————————		08 0e		\Rightarrow
> W addr_word2[7:0]	00	Ue	Od		04		02		\Rightarrow
> W addr_element1[5:0]	01		02		1		()		\Rightarrow
> w addr_element2[5:0]	00		03		1	05		03	➾
> • addr_index[4:0]			01		<u> </u>	02		01	_\
> 😽 addr_tag[22:0]	000001						000000		

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。

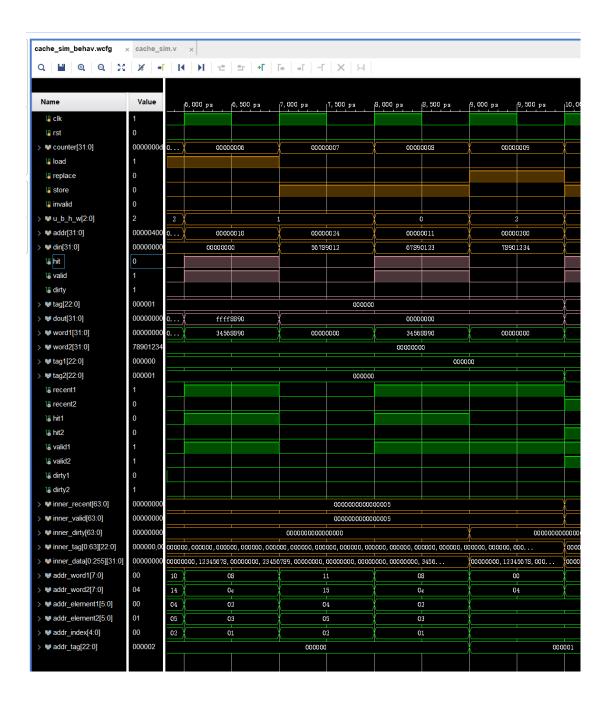
Task 9: 请给出一个 load miss 部分仿真的高清图片,并对涉及到的信号加以

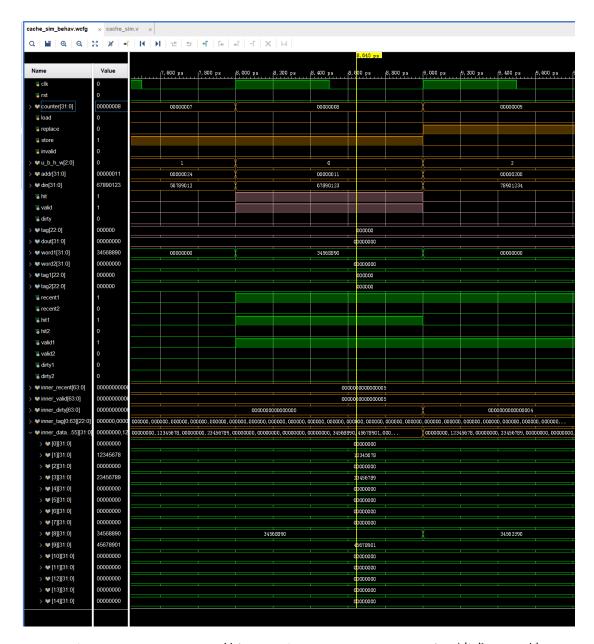
详细解释 (5 points)



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。

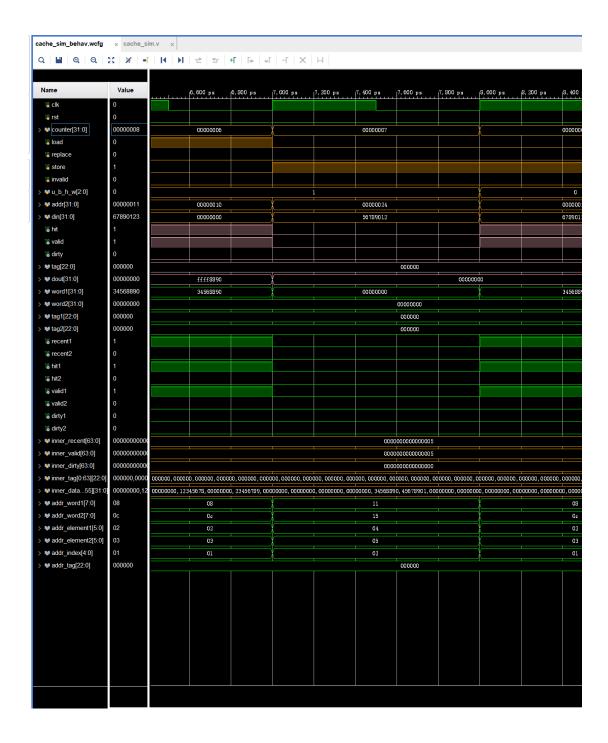
Task 10: 请给出一个 write hit 部分仿真的高清图片,并对涉及到的信号加以详细解释 (5 points)





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 部分仿真的高清图片,并对涉及到的信号加以详细解释 (5 points)

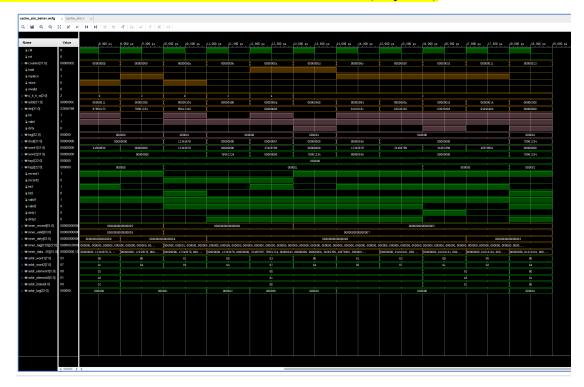


Q <u> </u> Q Q	20 × +		12 2r	+F Fe +F	-F X	[+-]					
4 = 4 4				. 10 0	1 1 1 7						
Name	Value		6,600 ps	6,800 ps	7,000 ps	7, 200 ps	7, 400 ps	7,600 ps	7,800 ps	8,000 ps	8, 200
18 dirty	0		*, ***, **	*,***,**	77777	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	7.77.77	777777	7,744	7, 7, 7, 7,	
₩ tag[22:0]	000000							000000			
→ dout[31:0]	00000000		ffff8890		<u> </u>				000000	00	
₩ word1[31:0]	34568890		34568890		\		00000000			<u> </u>	
₩ word2[31:0]	00000000				1			00000000		1	
₩ tag1[22:0]	000000							000000			
₩ tag2[22:0]	000000							000000			
18 recent1	1										
18 recent2	0										
18 hit1	1										
18 hit2	0										
18 valid1	1										
¹⊌ valid2	0										
18 dirty1	0										
18 dirty2	0										
₩ inner_recent[63:0]	00000000000						0000	0000000000005			
inner_valid[63:0] inner_valid[63:0]	00000000000							00000000000005			
▶ ™ inner_dirty[63:0]	00000000000							00000000000000			
inner_tag[0:63][22:0] inner_tag[0:63][22:0]			nn nnnnn nnn	000,000000,0000	ากก กกกกกก กกก						000000
/cache_sim/uut/inn											
> № [0][31:0]	00000000	71.0] 0000,12.	1430 75, 0000000	0, 20 400 10 2, 000		0, 00000000, 000	100000, 3430883	00000000	1	10,0000000,0	1
> W [1][31:0]	12345678							12345678			
> ₩ [2][31:0]	00000000							00000000			
> W [3][31:0]	23456789							23456789			
> W [4][31:0]	00000000							00000000			
	0000000										
> • [5][31:0]	00000000							00000000			
> ₩ [6][31:0]								00000000			
> 😻 [7][31:0]	00000000							00000000			
> • [8][31:0]	34568890							34568890			
> ₩ [9][31:0]	45678901							45678901			
> 🕨 [10][31:0]	00000000							00000000			
> 🕨 [11][31:0]	00000000							00000000			
> ₩ [12][31:0]	00000000							00000000			
> 🕨 [13][31:0]	00000000							00000000			
> 🕨 [14][31:0]	00000000							00000000			
> 🕨 [15][31:0]	00000000							00000000			
> 🕨 [16][31:0]	00000000							00000000			
> 😻 [17][31:0]	00000000							00000000			
> 😻 [18][31:0]	00000000							00000000			
> 😻 [19][31:0]	00000000							00000000			
> 😻 [20][31:0]	00000000							00000000			
> 😻 [21][31:0]	00000000							00000000			
> 🕨 [22][31:0]	00000000							00000000			
> 😻 [23][31:0]	00000000							00000000			
> 😻 [24][31:0]	00000000							00000000			
> 😻 [25][31:0]	00000000							00000000			
> 🕨 [26][31:0]	00000000							00000000			

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:

```
rst <= 0;
load <= 0;
store <= 0;
replace <= 1;</pre>
invalid <= 0;</pre>
addr <= 32'h4;
din <= 32'h01010101;</pre>
u_b_h_w <= 2;
load <= 0;
store <= 0;
replace <= 1;</pre>
invalid <= 0;</pre>
addr <= 32'hc;
din <= 32'h02020202;</pre>
u_b_h_w <= 2;
load <= 0;
```

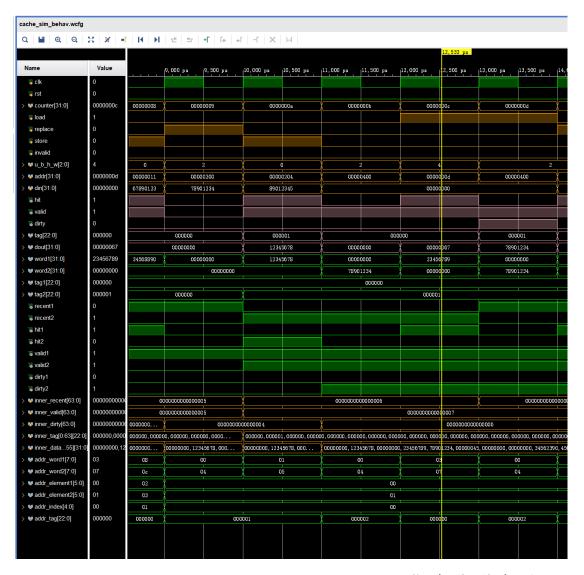
```
store <= 0;
    replace <= 1;</pre>
    invalid <= 0;
    addr <= 32'h10;
    din <= 32'h03030303;</pre>
    u_b_h_w <= 2;
32'd17: begin
    load <= 0;
    store <= 0;
    replace <= 1;</pre>
    invalid <= 0;
    addr <= 32'h14;
    din <= 32'h04040404;
    u_b_h_w <= 2;
    load <= 0;
    store <= 0;
    replace <= 0;
    invalid <= 0;</pre>
    addr <= 32'h200;
    din <= 32'h0;
    u_b_h_w <= 2;
```



cycle14-cycle17(counter=0x0e-0x11, 14000-18000ps 处), 此时对一开始写入的 cache line 再次进行替换,可以看到 valid 信号为 1(且 valid1=1)且第一路 hit

(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,写明白真实情况即可。 (+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。