浙江大学

《计算机体系结构》
实验报告

实验题目	：	Lab3 Cache Design
姓　　名	：	王晓宇
学　　号	：	3220104364
电子邮箱	：	3220104364@zju.edu.cn
联系电话	：	19550222634
授课教师	：	姜晓红
助　　教	：	简英钊&黄鸿宇

2024 年 11 月 4 日

Lab3 Cache Design

Lab3 Cache Design

课程名称：　计算机组成与设计　　实验类型：　综合

实验项目名称：　Lab3：Cache Design

学生姓名：　王晓宇　学号：　3220104364　同组学生姓名：　无

实验地点：　玉泉曹西301室　　实验日期：　2024 年 11 月 4 日

1　实验目的和要求

> Tips：写出本次实验的目的与要求

1.1　实验目的

- 理解Cache中Tag，Valid，Dirty的作用和Least Recently Used的概念
- 理解处理Hit，Miss等状况的方法
- 掌握验证Cache设计正确性的方法

1.2　实验要求

- 64 cache lines
- 2-way set associative
- 4 words per cache line
- Write Back
- Write Allocate
- LRU Replacement

2　实验内容和原理

> Tips：结合代码实现，简单说明cache line的tag，dirty，valid，lru信号是如何得到的，并解释cache的存取过程。

2.1 Cache Line信号获取

2.1.1 Tag信号

Tag信号用于标识缓存行中的数据是否与请求的地址匹配。代码中，`inner_tag`数组存储了每个缓存行的标签。

```
reg [TAG_BITS-1:0] inner_tag [0:ELEMENT_NUM-1];
```

在地址解析过程中，`addr_tag`从地址中提取出来，并与缓存行的标签进行比较可以的到`hit`标志位：

```
assign addr_tag = addr[ADDR_BITS-1:ADDR_BITS-TAG_BITS];
assign tag1 = inner_tag[addr_element1];
assign tag2 = inner_tag[addr_element2];
assign hit1 = valid1 & (tag1 == addr_tag);
assign hit2 = valid2 & (tag2 == addr_tag);
```

2.1.2 Dirty信号

Dirty信号表示缓存行中的数据是否被修改。代码中，`inner_dirty`数组存储了每个缓存行的脏位。

```
reg [ELEMENT_NUM-1:0] inner_dirty = 0;
```

在存储操作中，如果命中缓存行，则设置相应的脏位：

```
if (store) begin
  if (hit1) begin
    inner_dirty[addr_element1] <= 1'b1;
  end else if (hit2) begin
    inner_dirty[addr_element2] <= 1'b1
  end
end
```

2.1.3 Valid信号

Valid信号表示缓存行中的数据是否有效。代码中，`inner_valid`数组存储了每个缓存行的有效位。

```
reg [ELEMENT_NUM-1:0] inner_valid = 0;
```

在替换操作中，如果命中缓存行，则设置相应的有效位：

```
if (replace) begin
  if (hit2 | ((~hit1) & recent1)) begin
    inner_valid[addr_element2] <= 1'b1;
  end else begin
    inner_valid[addr_element1] <= 1'b1;
  end
end
```

2.1.4 LRU信号

LRU（最近最少使用）信号用于实现缓存替换策略。代码中，`inner_recent`数组存储了每个缓存行的最近使用位。

```
reg [ELEMENT_NUM-1:0] inner_recent = 0;
```

在加载和存储操作中，如果命中缓存行，则更新相应的最近使用位：

```
if (load) begin
  if (hit1) begin
    inner_recent[addr_element1] <= 1'b1;
    inner_recent[addr_element2] <= 1'b0;
  end else if (hit2) begin
    inner_recent[addr_element2] <= 1'b1;
    inner_recent[addr_element1] <= 1'b0;
  end
end
```

2.2 Cache的存取过程

2.2.1 地址解析

首先，将输入地址分解为标签位、索引位和字位：

```
assign addr_tag = addr[ADDR_BITS-1:ADDR_BITS-TAG_BITS];
assign addr_index = addr[ADDR_BITS-TAG_BITS-1:ADDR_BITS -TAG_BITS -
SET_INDEX_WIDTH];
assign addr_word = addr[WORD_BYTES_WIDTH + ELEMENT_WORDS_WIDTH -
1:WORD_BYTES_WIDTH];
```

2.2.2 数据读取

根据地址计算标签、索引和字位，并比较标签和有效位，确定是否命中缓存：

```
assign hit1 = valid1 & (tag1 == addr_tag);
assign hit2 = valid2 & (tag2 == addr_tag);
```

如果命中缓存，根据 `u_b_h_w` 信号选择读取的数据宽度（字节、半字、字）：

```
if (load & hit1) begin
  dout <= 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};
end else if (load & hit2) begin
  dout <= 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};
end
```

2.2.3 数据写入

根据地址计算标签、索引和字位，如果命中缓存，则写入数据并设置脏位和最近使用位：

```
if (store) begin
  if (hit1) begin
    inner_data[addr_word1] <= u_b_h_w[1] ? din
      : u_b_h_w[0] ? addr[1] ? {din[15:0], word1[15:0]} :
{word1[31:16], din[15:0]}
        :addr[1] ? addr[0] ? {din[7:0], word1[23:0]} :
{word1[31:24], din[7:0], word1[15:0]}
          :addr[0] ? {word1[31:16], din[7:0], word1[7:0]} :
{word1[31:8], din[7:0]};
    inner_dirty[addr_element1] <= 1'b1;
    inner_recent[addr_element1] <= 1'b1;
    inner_recent[addr_element2] <= 1'b0;
  end else if (hit2) begin
```

```
    inner_data[addr_word2] <= u_b_h_w[1] ? din
      :u_b_h_w[0] ? addr[1] ? {din[15:0], word2[15:0]} :
{word2[31:16], din[15:0]}
        :addr[1] ? addr[0] ? {din[7:0], word2[23:0]} :
{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]};
    inner_dirty[addr_element2] <= 1'b1;
    inner_recent[addr_element2] <= 1'b1;
    inner_recent[addr_element1] <= 1'b0;
  end
end
```

2.2.4 缓存替换

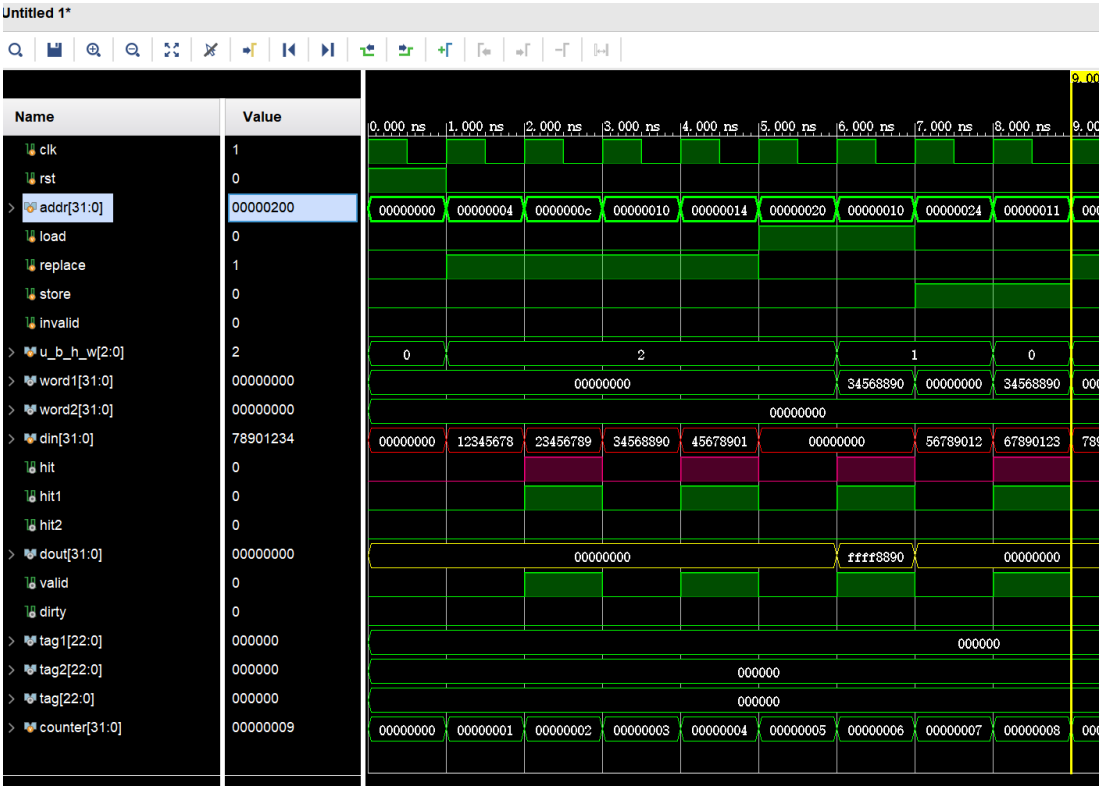
如果需要替换缓存行，根据 `replace` 信号选择要替换的缓存行，并更新数据、标签、有效位、脏位和最近使用位：

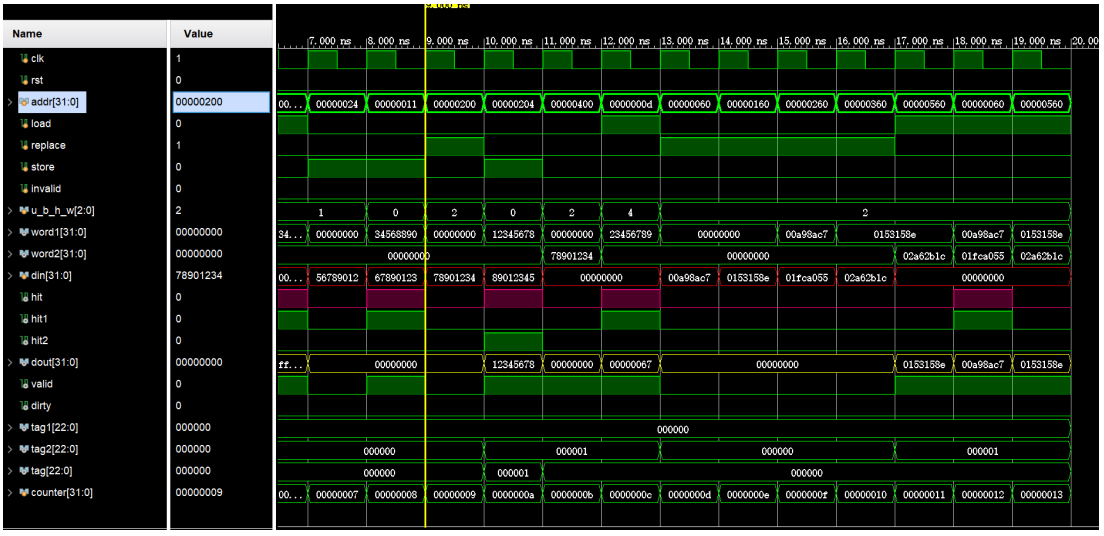
```
if (replace) begin
  if (hit2 | ((~hit1) & recent1)) begin
    inner_data[addr_word2] <= din;
    inner_valid[addr_element2] <= 1'b1;
    inner_dirty[addr_element2] <= 1'b0;
    inner_tag[addr_element2] <= addr_tag;
    inner_recent[addr_element2] <= 1'b1;
    inner_recent[addr_element1] <= 1'b0;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

3　实验过程和数据记录及结果分析

> Tips：请给出本次实验仿真关键信号截图，并结合波形简单解释cache的存取过程以及完成LRU Replacement的操作。

3.1　完整截图

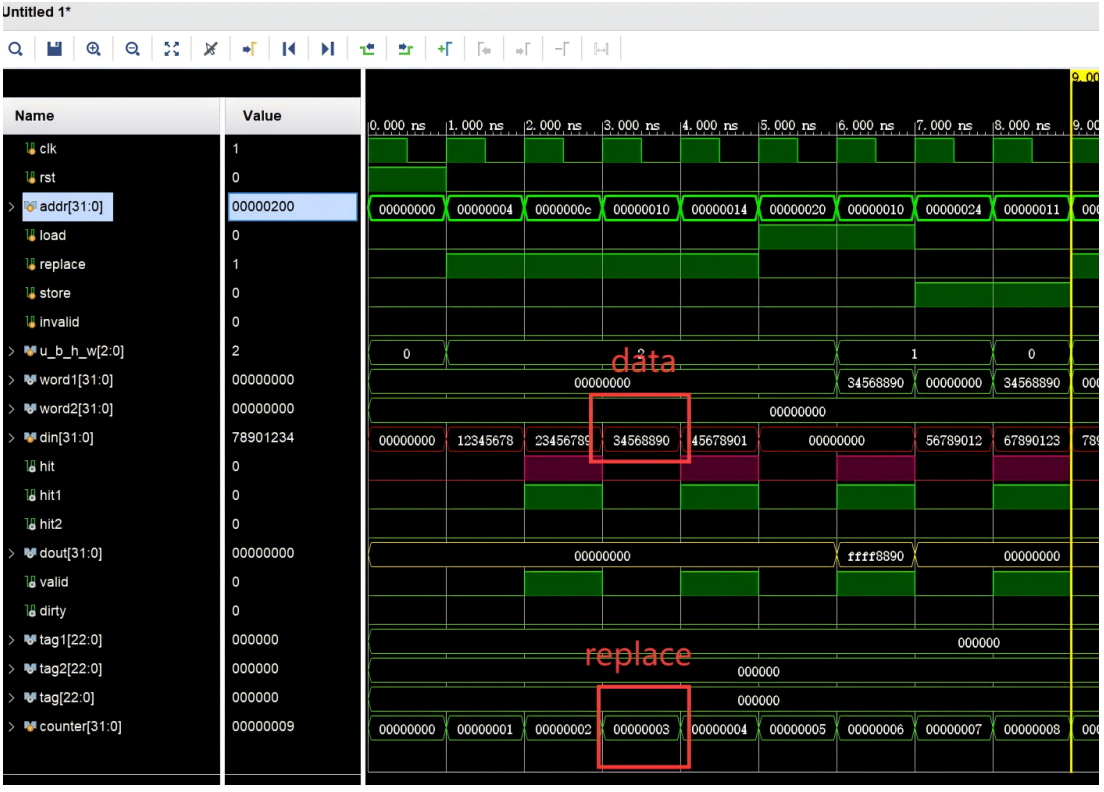




3.2 Cache存取

观察到计数器为3的仿真波形是replace，相当于对cache做了替换，对cache做了写操作

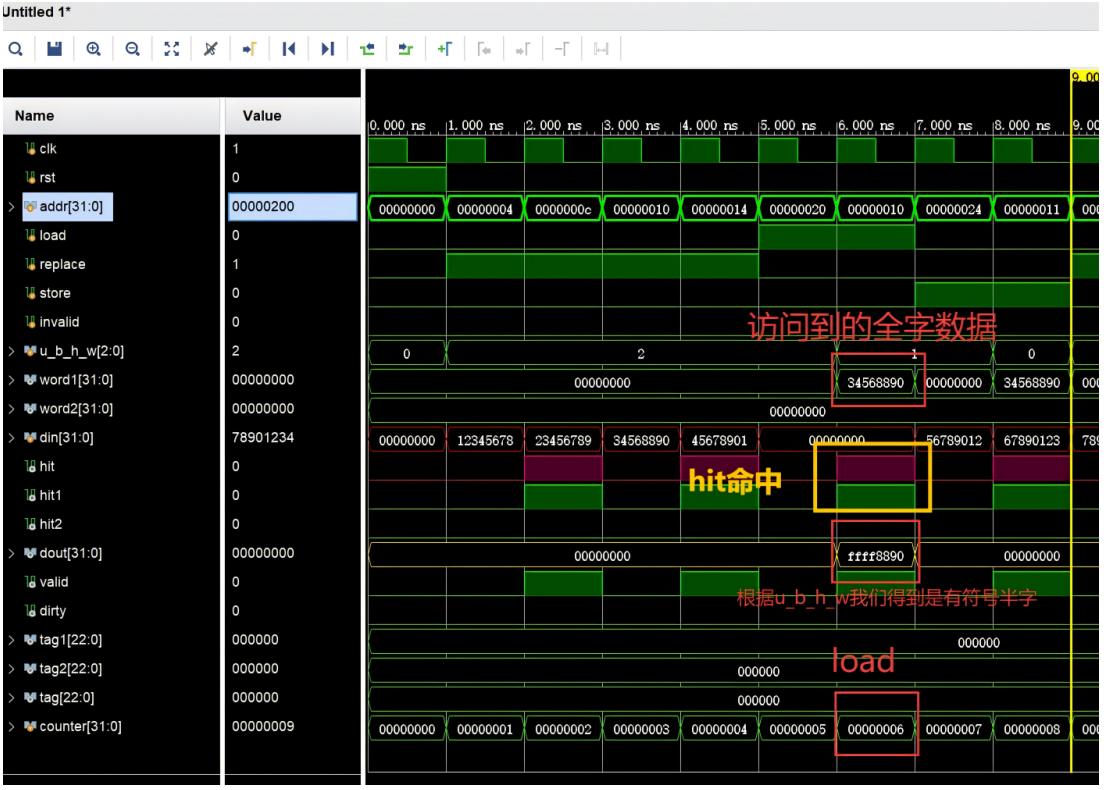
```
32'd3: begin
    load <= 0;
    store <= 0;
    replace <= 1;
    invalid <= 0;
    addr <= 32'h10;//10000
    din <= 32'h34568890;
    u_b_h_w <= 2;
end
```



从`32'h10`地址我们可以推断出index为0，此时index对应的cache块还是空闲的，所以直接在二路组相联的第一块进行了替换，`u_b_h_w <= 2;`我们可以得出替换长度为全字，也就意味着地址为`32'h10`的位置存放了0x34568890

现在我们聚焦于计数器为6时数据的取用：

```
32'd6: begin
    load <= 1;
    store <= 0;
    replace <= 0;
    invalid <= 0;
    addr <= 32'h10;//10000
    din <= 32'h0;
    u_b_h_w <= 1;
end
```



由`addr`我们可以推断出index与上文提到的一致，均为0，且`tag`相同，此时命中。数据直接取到了它的全字`0x34568890`，根据`u_b_h_w`为1我们可以得到想要取得的数据是有符号半字类型，即`0xffff8890`并将结果同步到`dout`

3.3 LRU替换策略

由于原测试仿真文件没有关于Cache替换，我们在测试文件中再补充测试

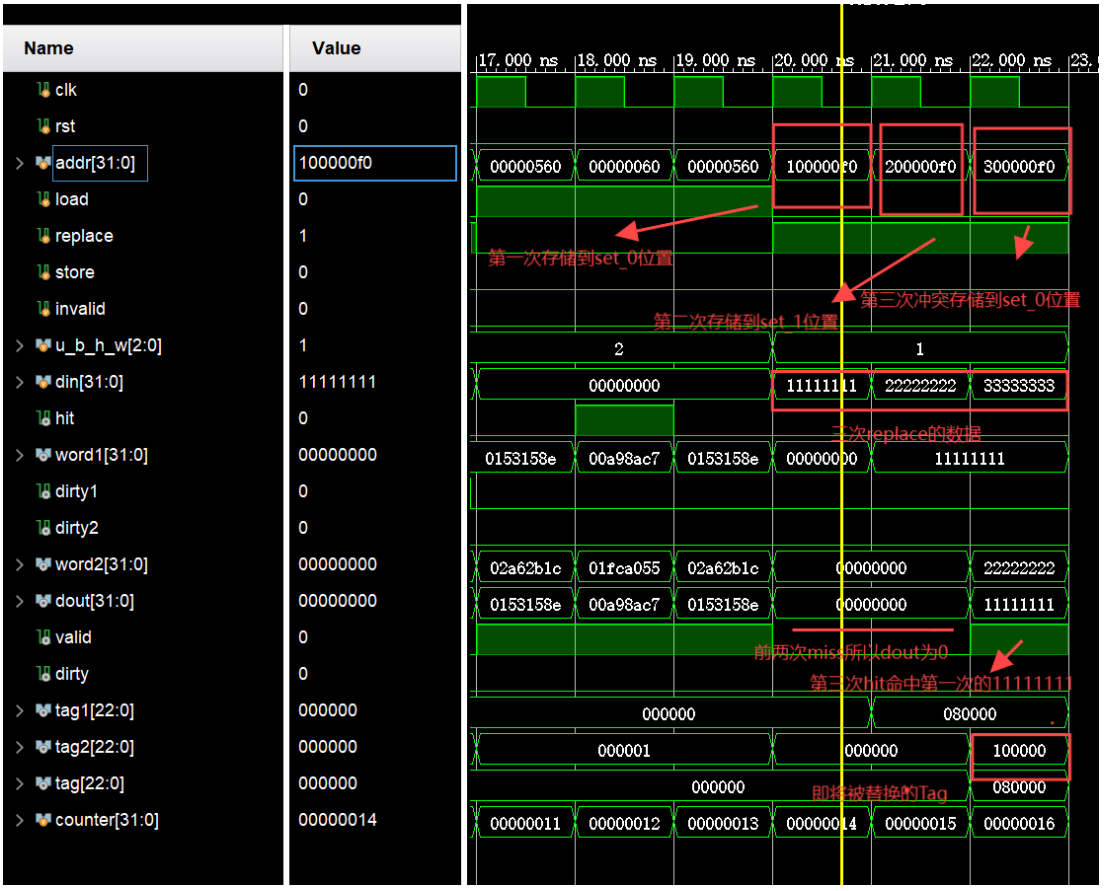
```
32'd20: begin
    load <= 0;
    store <= 0;
    replace <= 1;
```

```
    invalid <= 0;
    //00010000000000000000000011110000
    addr <= 32'h100000f0;
    din <= 32'h0;
    u_b_h_w <= 1;
end

32'd21: begin
    load <= 0;
    store <= 0;
    replace <= 1;
    invalid <= 0;
    //00100000000000000000000011110000
    addr <= 32'h200000f0;
    din <= 32'h0;
    u_b_h_w <= 1;
end

32'd22: begin
    load <= 0;
    store <= 0;
    replace <= 1;
    invalid <= 0;
    //00110000000000000000000011110000
    addr <= 32'h200000f0;
    din <= 32'h0;
    u_b_h_w <= 1;
end

```



1. 访问地址 `32'h100000f0`：
    - 标签位：`00010000000000000000`
    - 索引位：`01111`
    - 命中缓存行1，更新 `inner_recent[addr_element1]` 为 `1`，`inner_recent[addr_element2]` 为 `0`。
2. 访问地址 `32'h200000f0`：
    - 标签位：`00100000000000000000`
    - 索引位：`01111`
    - 命中缓存行2，更新 `inner_recent[addr_element2]` 为 `1`，`inner_recent[addr_element1]` 为 `0`。
3. 访问地址 `32'h300000f0`：
    - 标签位：`00110000000000000000`
    - 索引位：`01111`
    - 发生缓存冲突，因为两个缓存行的标签位都不匹配。

- 应用LRU策略，选择最近最少使用的缓存行`addr_element1`即进行替换。

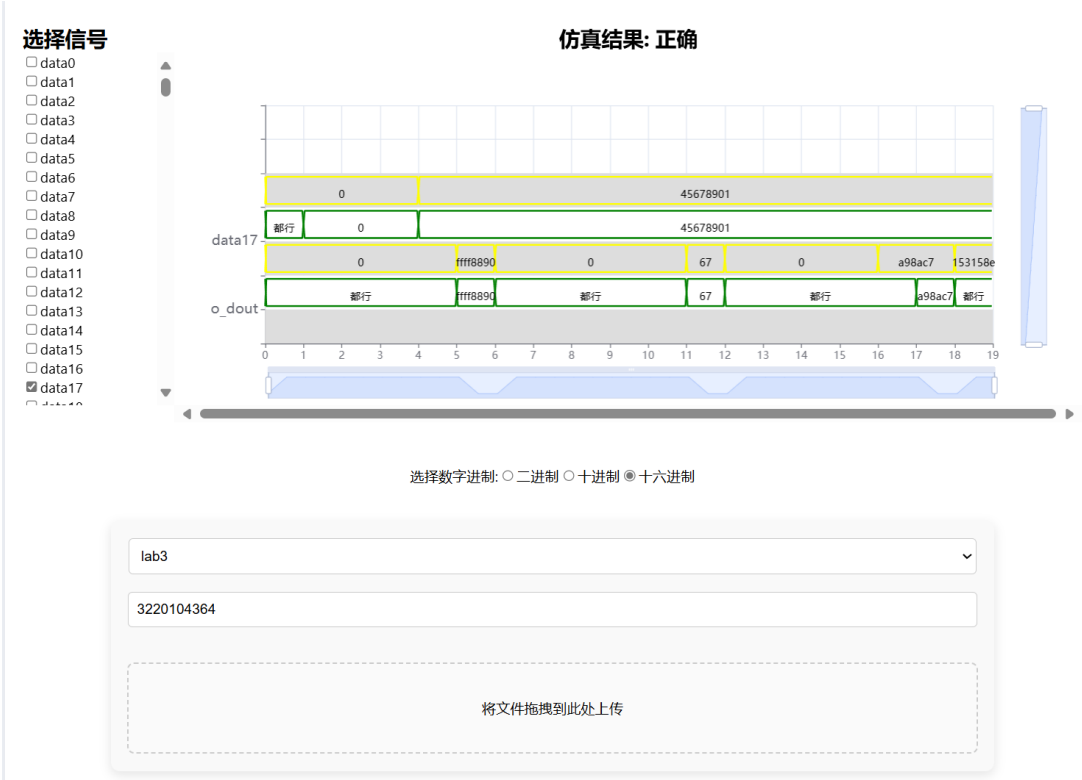
4 关于Bouns

4.1 验收json文件

验收将`config`部分的`associativity`参数改为4即可

```
"associativity": 2,  ---->"associativity": 4,
```

之后放到线上平台测试



测试成功

4.2 设计思路

除了LRU更换策略与二路组相联的略不同外，其余的部分基本一致，在此不在赘述

- **LRU替换策略的实现**

在所选的代码片段中，LRU（最近最少使用）替换策略通过 `inner_recent` 数组来实现。`inner_recent` 数组用于记录每个缓存行的最近使用情况，2位宽度的寄存器表示最近使用的程度。以下是详细的实现过程：

1. 定义和初始化

首先，定义了一个2位宽的寄存器数组 `inner_recent`，用于存储每个缓存行的最近使用情况：

```
reg [1:0] inner_recent [0:ELEMENT_NUM-1];
```

在初始化过程中，将 `inner_recent` 数组的所有元素设置为 `2'b0`，表示所有缓存行最初都未被使用：

```
initial begin
  for (i = 0; i < ELEMENT_NUM; i = i + 1)
    inner_recent[i] = 2'b0;
end
```

2. 更新最近使用位

在每次加载（load）或存储（store）操作时，更新 `inner_recent` 数组，以反映缓存行的最近使用情况。具体来说：

- 加载操作：
  - 假如命中了组内第x个缓存，记当前缓存的`recent`为m。
  - 如果其他缓存行的`recent`值n大于m，将其他缓存行`inner_recent` 值减1，表示它们的使用优先级降低。
  - 如果其他缓存行的`recent`值n小于等于m，代表这个的优先级不用更改。
  - 此时将缓存行x的 `inner_recent` 设置为最高值 `2'b11`，表示最近使用。

通过上述步骤，LRU替换策略得以实现，确保每次替换操作都选择最近最少使用的缓存行，从而提高缓存的命中率和整体性能。

4.3 源代码

```
// | ----------- address 32 ----------- |
// | 31   8 | 7     4 | 3    2 | 1    0 |
// | tag 24 | index 4 | word 2 | byte 2 |

module cache (
    input wire clk,  // clock
    input wire rst,  // reset
    input wire [ADDR_BITS-1:0] addr,  // address
    input wire load,    //  read refreshes recent bit
    input wire replace,  // set valid to 1 and reset dirty to 0
    input wire store,  // set dirty to 1
    input wire invalid,  // reset valid to 0
    input wire [2:0] u_b_h_w, // select signed or not & data width
                              // please refer to definition of LB,
LH, LW, LBU, LHU in RV32I Instruction Set
```

```
    input wire [31:0] din,  // data write in
    output reg hit = 0,  // hit or not
    output reg [31:0] dout = 0,  // data read out
    output reg valid = 0,  // valid bit
    output reg dirty = 0,  // dirty bit
    output reg [TAG_BITS-1:0] tag = 0// tag bits
    );

    `include "addr_define.vh"
    reg TO_BE_FILLED = 0;
    wire [31:0] word1, word2,word3,word4;
    wire [15:0] half_word1, half_word2, half_word3, half_word4;
    wire [7:0]  byte1, byte2, byte3, byte4;
    wire [1:0] recent1, recent2, recent3,recent4;//2 bits for recent bit,the higher the recent
    wire valid1, valid2,valid3, valid4, dirty1, dirty2, dirty3, dirty4;
    wire [TAG_BITS-1:0] tag1, tag2, tag3, tag4;
    wire hit1, hit2, hit3, hit4;
    reg [1:0] inner_recent [0:ELEMENT_NUM-1] ;
    reg [ELEMENT_NUM-1:0] inner_valid = 0;
    reg [ELEMENT_NUM-1:0] inner_dirty = 0;
    reg [TAG_BITS-1:0] inner_tag [0:ELEMENT_NUM-1];
    // 64 elements, 4 ways set associative => 16 sets
    reg [31:0] inner_data [0:ELEMENT_NUM*ELEMENT_WORDS-1];

    // initialize tag and data with 0
    integer i;
    initial begin
        for (i = 0; i < ELEMENT_NUM; i = i + 1)
            inner_tag[i] = 23'b0;

        for (i = 0; i < ELEMENT_NUM*ELEMENT_WORDS; i = i + 1)
            inner_data[i] = 32'b0;

        for (i = 0; i < ELEMENT_NUM; i = i + 1)
```

```
            inner_recent[i] = 2'b0;
    end

    // the bits in an input address:
    wire [TAG_BITS-1:0] addr_tag;
    wire [SET_INDEX_WIDTH-1:0] addr_index;     // idx of set
    wire [ELEMENT_WORDS_WIDTH+WORD_BYTES_WIDTH-WORD_BYTES_WIDTH-
1:0] addr_word;
    wire [ELEMENT_INDEX_WIDTH-1:0] addr_element1;
    wire [ELEMENT_INDEX_WIDTH-1:0] addr_element2;     // idx of
element
    wire [ELEMENT_INDEX_WIDTH-1:0] addr_element3;
    wire [ELEMENT_INDEX_WIDTH-1:0] addr_element4;
    wire [ELEMENT_INDEX_WIDTH+ELEMENT_WORDS_WIDTH-1:0] addr_word1;
    wire [ELEMENT_INDEX_WIDTH+ELEMENT_WORDS_WIDTH-1:0] addr_word2;
// element index + word index
    wire [ELEMENT_INDEX_WIDTH+ELEMENT_WORDS_WIDTH-1:0] addr_word3;
    wire [ELEMENT_INDEX_WIDTH+ELEMENT_WORDS_WIDTH-1:0] addr_word4;

    assign addr_tag = addr[ADDR_BITS-1:ADDR_BITS-TAG_BITS];
    assign addr_index = addr[ADDR_BITS-TAG_BITS-1:ADDR_BITS -
TAG_BITS - SET_INDEX_WIDTH];//TO_BE_FILLED;
    assign addr_word = addr[WORD_BYTES_WIDTH + ELEMENT_WORDS_WIDTH
- 1:WORD_BYTES_WIDTH];//TO_BE_FILLED;

    assign addr_element1 = {addr_index, 2'b00};
    assign addr_element2 = {addr_index, 2'b01};
    assign addr_element3 = {addr_index, 2'b10};
    assign addr_element4 = {addr_index, 2'b11};
    assign addr_word1 = {addr_element1, addr_word};
    assign addr_word2 = {addr_element2, addr_word};
    assign addr_word3 = {addr_element3, addr_word};
    assign addr_word4 = {addr_element4, addr_word};

    assign word1 = inner_data[addr_word1];
    assign word2 = inner_data[addr_word2];
```

```
80      assign word3 = inner_data[addr_word3];
81      assign word4 = inner_data[addr_word4];
82      assign half_word1 = addr[1] ? word1[31:16] : word1[15:0];
83      assign half_word2 = addr[1] ? word2[31:16] : word2[15:0];
84      assign half_word3 = addr[1] ? word3[31:16] : word3[15:0];
85      assign half_word4 = addr[1] ? word4[31:16] : word4[15:0];

86      assign byte1 = addr[1] ?
87                     addr[0] ? word1[31:24] : word1[23:16] :
88                     addr[0] ? word1[15:8] :  word1[7:0]   ;
89      assign byte2 =  addr[1] ?
90                     addr[0] ? word2[31:24] : word2[23:16] :
91                     addr[0] ? word2[15:8] :  word2[7:0]   ;
92      assign byte3 =  addr[1] ?
93                     addr[0] ? word3[31:24] : word3[23:16] :
94                     addr[0] ? word3[15:8] :  word3[7:0]   ;
95      assign byte4 =  addr[1] ?
96                     addr[0] ? word4[31:24] : word4[23:16] :
97                     addr[0] ? word4[15:8] :  word4[7:0]   ;
98
99      assign recent1 = inner_recent[addr_element1];
100     assign recent2 = inner_recent[addr_element2];
101     assign recent3 = inner_recent[addr_element3];
102     assign recent4 = inner_recent[addr_element4];
103     assign valid1 = inner_valid[addr_element1];
104     assign valid2 = inner_valid[addr_element2];
105     assign valid3 = inner_valid[addr_element3];
106     assign valid4 = inner_valid[addr_element4];
107     assign dirty1 = inner_dirty[addr_element1];
108     assign dirty2 = inner_dirty[addr_element2];
109     assign dirty3 = inner_dirty[addr_element3];
110     assign dirty4 = inner_dirty[addr_element4];
111     assign tag1 = inner_tag[addr_element1];
112     assign tag2 = inner_tag[addr_element2];
113     assign tag3 = inner_tag[addr_element3];
114     assign tag4 = inner_tag[addr_element4];
```

```
115     assign hit1 = valid1 & (tag1 == addr_tag);
116     assign hit2 = valid2 & (tag2 == addr_tag);
117     assign hit3 = valid3 & (tag3 == addr_tag);
118     assign hit4 = valid4 & (tag4 == addr_tag);
119 
120     always @ (*) begin
121         valid <= hit1 ? valid1 : hit2 ? valid2 : hit3 ? valid3 :
    hit4 ? valid4 :
122             (recent1<=recent2&&recent1<=recent3&&recent1<=recent4)
    ? valid1 :
123             (recent2<=recent1&&recent2<=recent3&&recent2<=recent4)
    ? valid2 :
124             (recent3<=recent2&&recent3<=recent1&&recent3<=recent4)
    ? valid3 : valid4;
125         dirty <= hit1 ? dirty1 : hit2 ? dirty2 : hit3 ? dirty3 :
    hit4 ? dirty4 :
126             (recent1<=recent2&&recent1<=recent3&&recent1<=recent4)
    ? dirty1 :
127             (recent2<=recent1&&recent2<=recent3&&recent2<=recent4)
    ? dirty2 :
128             (recent3<=recent2&&recent3<=recent1&&recent3<=recent4)
    ? dirty3 : dirty4;
129         tag <= hit1 ? tag1 : hit2 ? tag2 : hit3 ? tag3 : hit4 ?
    tag4 :
130             (recent1<=recent2&&recent1<=recent3&&recent1<=recent4)
    ? tag1 :
131             (recent2<=recent1&&recent2<=recent3&&recent2<=recent4)
    ? tag2 :
132             (recent3<=recent2&&recent3<=recent1&&recent3<=recent4)
    ? tag3 : tag4;
133         hit <= hit1 | hit2 | hit3 | hit4;
134 
135         if (load & hit1) begin
136             dout <= u_b_h_w[1] ? word1 :
137             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};
        end
        else if (load & hit2) begin
            dout <= 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};
        end
        else if (load & hit3) begin
            dout <= u_b_h_w[1] ? word3 :
            u_b_h_w[0] ? {u_b_h_w[2] ? 16'b0 :
{16{half_word3[15]}}, half_word3} :
            {u_b_h_w[2] ? 24'b0 : {24{byte3[7]}}, byte3};
        end
        else if (load & hit4) begin
            dout <= u_b_h_w[1] ? word4 :
            u_b_h_w[0] ? {u_b_h_w[2] ? 16'b0 :
{16{half_word4[15]}}, half_word4} :
            {u_b_h_w[2] ? 24'b0 : {24{byte4[7]}}, byte4};
        end
        else begin
            // dout <= recent1 ? word2 : word1;
            dout <=
(recent1<=recent2&&recent1<=recent3&&recent1<=recent4) ? word1 :

 (recent2<=recent1&&recent2<=recent3&&recent2<=recent4) ? word2 :

 (recent3<=recent2&&recent3<=recent1&&recent3<=recent4) ? word3 :
word4;
        end
    end

    always @ (posedge clk) begin
        if (load) begin
            if (hit1) begin
```

```
166                 inner_recent[addr_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element1]) ?
167                     inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
168                 inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element1]) ?
169                     inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
170                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element1]) ?
171                     inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
172                 inner_recent[addr_element1] <= 2'b11;
173             end
174             else if (hit2) begin
175                 inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element2]) ?
176                     inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
177                 inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element2]) ?
178                     inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
179                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element2]) ?
180                     inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
181                 inner_recent[addr_element2] <= 2'b11;
182             end
183             else if (hit3) begin
184                 inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element3]) ?
185                     inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
186                 inner_recent[addr_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element3]) ?
```

```
187                     inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
188                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element3]) ?
189                     inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
190                 inner_recent[addr_element3] <= 2'b11;
191             end
192             else if (hit4) begin
193                 inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element4]) ?
194                     inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
195                 inner_recent[addr_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element4]) ?
196                     inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
197                 inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element4]) ?
198                     inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
199                 inner_recent[addr_element4] <= 2'b11;
200             end
201         end
202
203         if (store) begin
204             if (hit1) begin
205                 inner_data[addr_word1] <= u_b_h_w[1] ? din
206                     : u_b_h_w[0] ? addr[1] ? {din[15:0],
word1[15:0]} : {word1[31:16], din[15:0]}
207                         :addr[1] ? addr[0] ? {din[7:0],
word1[23:0]} : {word1[31:24], din[7:0], word1[15:0]}
208                             :addr[0] ? {word1[31:16], din[7:0],
word1[7:0]} : {word1[31:8], din[7:0]};
209                 inner_dirty[addr_element1] <= 1'b1;
```

```
210                 inner_recent[addr_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element1]) ?
211                     inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
212                 inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element1]) ?
213                     inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
214                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element1]) ?
215                     inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
216                 inner_recent[addr_element1] <= 2'b11;
217             end
218             else if (hit2) begin
219                 inner_data[addr_word2] <= u_b_h_w[1] ? din
220                     :u_b_h_w[0] ? addr[1] ? {din[15:0],
word2[15:0]} : {word2[31:16], din[15:0]}
221                         :addr[1] ? addr[0] ? {din[7:0],
word2[23:0]} : {word2[31:24], din[7:0], word2[15:0]}
222                             :addr[0] ? {word2[31:16], din[7:0],
word2[7:0]} : {word2[31:8], din[7:0]};
223                 inner_dirty[addr_element2] <= 1'b1;
224                 inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element2]) ?
225                     inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
226                 inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element2]) ?
227                     inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
228                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element2]) ?
229                     inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
230                 inner_recent[addr_element2] <= 2'b11;
```

```
231             end
232             else if (hit3) begin
233                 inner_data[addr_word3] <= u_b_h_w[1] ? din
234                     :u_b_h_w[0] ? addr[1] ? {din[15:0],
word3[15:0]} : {word3[31:16], din[15:0]}
235                         :addr[1] ? addr[0] ? {din[7:0],
word3[23:0]} : {word3[31:24], din[7:0], word3[15:0]}
236                             :addr[0] ? {word3[31:16], din[7:0],
word3[7:0]} : {word3[31:8], din[7:0]};
237                 inner_dirty[addr_element3] <= 1'b1;
238                 inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element3]) ?
239                     inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
240                 inner_recent[addr_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element3]) ?
241                     inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
242                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element3]) ?
243                     inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
244                 inner_recent[addr_element3] <= 2'b11;
245             end
246             else if (hit4) begin
247                 inner_data[addr_word4] <= u_b_h_w[1] ? din
248                     :u_b_h_w[0] ? addr[1] ? {din[15:0],
word4[15:0]} : {word4[31:16], din[15:0]}
249                         :addr[1] ? addr[0] ? {din[7:0],
word4[23:0]} : {word4[31:24], din[7:0], word4[15:0]}
250                             :addr[0] ? {word4[31:16], din[7:0],
word4[7:0]} : {word4[31:8], din[7:0]};
251                 inner_dirty[addr_element4] <= 1'b1;
252                 inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element4]) ?
```

```
253                     inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
254                 inner_recent[addr_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element4]) ?
255                     inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
256                 inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element4]) ?
257                     inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
258                 inner_recent[addr_element4] <= 2'b11;
259 
260             end
261         end
262 
263         if (replace) begin
264             if (hit1)begin
265                 inner_data[addr_word1] <= din;
266                 inner_valid[addr_element1] <= 1'b1;
267                 inner_dirty[addr_element1] <= 1'b0;
268                 inner_tag[addr_element1] <= addr_tag;
269                 inner_recent[addr_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element1]) ?
270                     inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
271                 inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element1]) ?
272                     inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
273                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element1]) ?
274                     inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
275                 inner_recent[addr_element1] <= 2'b11;
276             end
277             else if (hit2)begin
```

```
278                 inner_data[addr_word2] <= din;
279                 inner_valid[addr_element2] <= 1'b1;
280                 inner_dirty[addr_element2] <= 1'b0;
281                 inner_tag[addr_element2] <=
addr_tag;//[TO_BE_FILLED] <= addr_tag;
282                 inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element2]) ?
283                     inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
284                 inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element2]) ?
285                     inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
286                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element2]) ?
287                     inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
288                 inner_recent[addr_element2] <= 2'b11;
289             end
290             else if (hit3)begin
291                 inner_data[addr_word3] <= din;
292                 inner_valid[addr_element3] <= 1'b1;
293                 inner_dirty[addr_element3] <= 1'b0;
294                 inner_tag[addr_element3] <=
addr_tag;//[TO_BE_FILLED] <= addr_tag;
295                 inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element3]) ?
296                     inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
297                 inner_recent[addr_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element3]) ?
298                     inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
299                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element3]) ?
```

```
                    inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
                inner_recent[addr_element3] <= 2'b11;
            end
            else if (hit4)begin
                inner_data[addr_word4] <= din;
                inner_valid[addr_element4] <= 1'b1;
                inner_dirty[addr_element4] <= 1'b0;
                inner_tag[addr_element4] <=
addr_tag;//[TO_BE_FILLED] <= addr_tag;
                inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element4]) ?
                    inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
                inner_recent[addr_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element4]) ?
                    inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
                inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element4]) ?
                    inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
                inner_recent[addr_element4] <= 2'b11;
            end
            else if
(recent1<=recent2&&recent1<=recent3&&recent1<=recent4) 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_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element1]) ?
                    inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
                inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element1]) ?
```

```
324                     inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
325                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element1]) ?
326                     inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
327                 inner_recent[addr_element1] <= 2'b11;
328             end
329             else if
(recent2<=recent1&&recent2<=recent3&&recent2<=recent4) begin
330                 inner_data[addr_word2] <= din;
331                 inner_valid[addr_element2] <= 1'b1;
332                 inner_dirty[addr_element2] <= 1'b0;
333                 inner_tag[addr_element2] <= addr_tag;
334                 inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element2]) ?
335                     inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
336                 inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element2]) ?
337                     inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
338                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element2]) ?
339                     inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
340                 inner_recent[addr_element2] <= 2'b11;
341             end
342             else if
(recent3<=recent2&&recent3<=recent1&&recent3<=recent4) begin
343                 inner_data[addr_word3] <= din;
344                 inner_valid[addr_element3] <= 1'b1;
345                 inner_dirty[addr_element3] <= 1'b0;
346                 inner_tag[addr_element3] <= addr_tag;
347                 inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element3]) ?
```

```
348                     inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
349                 inner_recent[addr_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element3]) ?
350                     inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
351                 inner_recent[addr_element4] <=
(inner_recent[addr_element4] <= inner_recent[addr_element3]) ?
352                     inner_recent[addr_element4] :
inner_recent[addr_element4]-2'b01;
353                 inner_recent[addr_element3] <= 2'b11;
354             end
355             else begin
356                 inner_data[addr_word4] <= din;
357                 inner_valid[addr_element4] <= 1'b1;
358                 inner_dirty[addr_element4] <= 1'b0;
359                 inner_tag[addr_element4] <= addr_tag;
360                 inner_recent[addr_element1] <=
(inner_recent[addr_element1] <= inner_recent[addr_element4]) ?
361                     inner_recent[addr_element1] :
inner_recent[addr_element1]-2'b01;
362                 inner_recent[addr_element2] <=
(inner_recent[addr_element2] <= inner_recent[addr_element4]) ?
363                     inner_recent[addr_element2] :
inner_recent[addr_element2]-2'b01;
364                 inner_recent[addr_element3] <=
(inner_recent[addr_element3] <= inner_recent[addr_element4]) ?
365                     inner_recent[addr_element3] :
inner_recent[addr_element3]-2'b01;
366                 inner_recent[addr_element4] <= 2'b11;
367             end
368
369         end
370         // not used currently, can be used to reset the cache.
371         if (invalid) begin
372             inner_recent[addr_element1] <= 2'b0;
```

```
            inner_recent[addr_element2] <= 2'b0;
            inner_recent[addr_element3] <= 2'b0;
            inner_recent[addr_element4] <= 2'b0;
            inner_valid[addr_element1] <= 1'b0;
            inner_valid[addr_element2] <= 1'b0;
            inner_valid[addr_element3] <= 1'b0;
            inner_valid[addr_element4] <= 1'b0;
            inner_dirty[addr_element1] <= 1'b0;
            inner_dirty[addr_element2] <= 1'b0;
            inner_dirty[addr_element3] <= 1'b0;
            inner_dirty[addr_element4] <= 1'b0;
        end
    end

endmodule
```

5 讨论与心得

> Tips: 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。

暂无，大三秋冬课有点小多，以前计逻、计组还会在这发发牢骚，现在没时间了T_T（