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

实验名称：Lab3:Cache Design

指导教师：何水兵 完成日期：2023 年 11 月 21 日

实验分工

姓名：蒋奕 3210103803

同组学生姓名：黄俊涛 3210101831

分工情况：

蒋奕：cache 模块 + 设计测试仿真

cache 采用两路组关联，write back 和 write allocate 策略。

黄俊涛：cache 模块 + 设计测试仿真

1 实验目的和要求

- Understand Cache Line.
- Understand the principle of Cache Management Unit (CMU) and State Machine of CMU.
- Master the design methods of CMU.
- Master the design methods of Cache Line.
- Master verification methods of Cache Line.

2 实验内容和原理

2.1 实验内容

- Design of Cache Line and CMU.
- Verify the Cache Line and CMU.
- Observe the Waveform of Simulation.

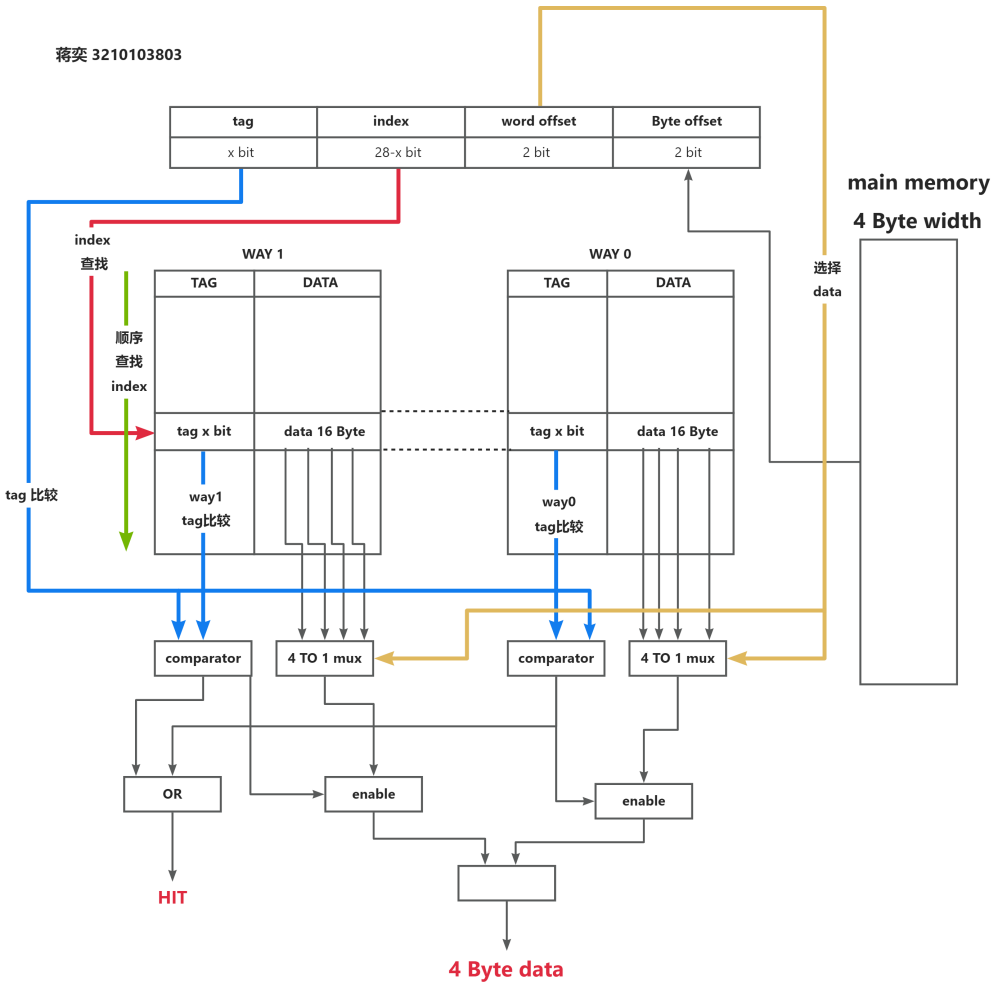
2.2 实验原理

- Structure of Cache The cache line is as follows:

LRU	Dirty	Valid	Tag	Data
block replacement				

In this experiment, we use 2-way set associative Write Back and Write Allocate to design the cache.

The schematic is as follows:



- Find index Just as the schematic given above, we can find how we search for proper index by order from 0 to $2^n - 1$ and find the proper index for next operation.
- Compare tag Just as the figure given above, we can find how we compare tag between the 2-way and find the proper tag and way for next operation.
- Select data Just like the line in the graph, we use offset to select the proper data from the data block of the cache.

  As for implementing the 2-way associative cache in this lab, we should do things as follows:

- Generate cache internal signal

  This lab uses 2-way association, so various signals are symmetrically divided into two channels, 1 and 2.

- Generate cache output signal

  The function of valid, dirty and tag signals is to determine page change whether it is necessary to write back, and because this lab requires the cache to use the LRU replacement strategy,

what needs to be sent out is the value of the block that is about to be swapped out. When using block No. 1, the numbers valid, dirty and tag of block No. 2 are sent out, and vice versa. The hit signal has nothing to do with which block was recently used. As long as any one of the two paths hits, it is a cache hit.

3 实验过程和数据记录及结果分析

3.1 实验过程

The details of cache can be seen in comments of the corresponding code.

- Complete 2-way basic things

```
assign word1 = inner_data[addr_word1];
assign word2 = inner_data[addr_word2];
assign half_word1 = addr[1] ? word1[31:16] : word1[15:0];
assign half_word2 = addr[1] ? word2[31:16] : word2[15:0];
assign byte1 = addr[1] ?
addr[0] ? word1[31:24] : word1[23:16] :
addr[0] ? word1[15:8] :  word1[7:0]   ;
assign byte2 = addr[1] ?
addr[0] ? word2[31:24] : word2[23:16] :
addr[0] ? word2[15:8] :  word2[7:0]   ;
assign recent1 = inner_recent[addr_element1];
assign recent2 = inner_recent[addr_element2];
assign valid1 = inner_valid[addr_element1];
assign valid2 = inner_valid[addr_element2];
assign dirty1 = inner_dirty[addr_element1];
assign dirty2 = inner_dirty[addr_element2];
assign tag1 = inner_tag[addr_element1];
assign tag2 = inner_tag[addr_element2];
assign hit1 = valid1 & (tag1 == addr_tag);
assign hit2 = valid2 & (tag2 == addr_tag);
```

- Generate output signal of cache

  The function of valid, dirty and tag signals is to determine the page change whether it is necessary to write back, and because this lab requires the cache to use the LRU replacement strategy, what needs to be sent out is not the value of the block where the currently accessed address is located, but the value of the block that is about to be swapped out, so when the recently used block is No.1, the valid, dirty and tag values of block No.2 are sent out, and vice versa.

  The hit signal has nothing to do with which block was recently used. As long as any one of the two paths hits, the cache hit.

```
always @ (posedge clk) begin
        write_miss <= 1'b0;
        valid <= recent1 ? valid2 : valid1;
        dirty <= recent1 ? dirty2 : dirty1;
        tag <= recent1 ? tag2 : tag1;
        hit <= hit1 | hit2;
```

- cache read hit

  When Cache read hit, the corresponding bit output in the result is taken out according to the requirements of u_b_h_w, and updated recent signal at the same time.

```
else if (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};
        inner_recent[addr_element1] <= 1'b0;
        inner_recent[addr_element2] <= 1'b1;
end
```

- cache write hit

  When Cache write hit, write the corresponding bit according to the requirements of u_b_h_w, and update recent and dirty value.

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

- Switch page

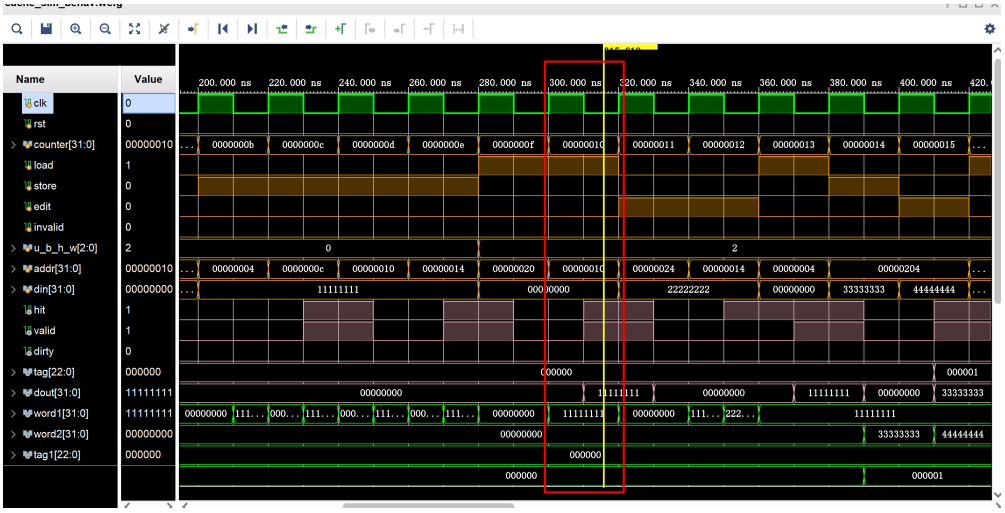
  When switching a page, the contents of the entire block are changed to the entered data, and the valid, dirty, and tag information are reset.

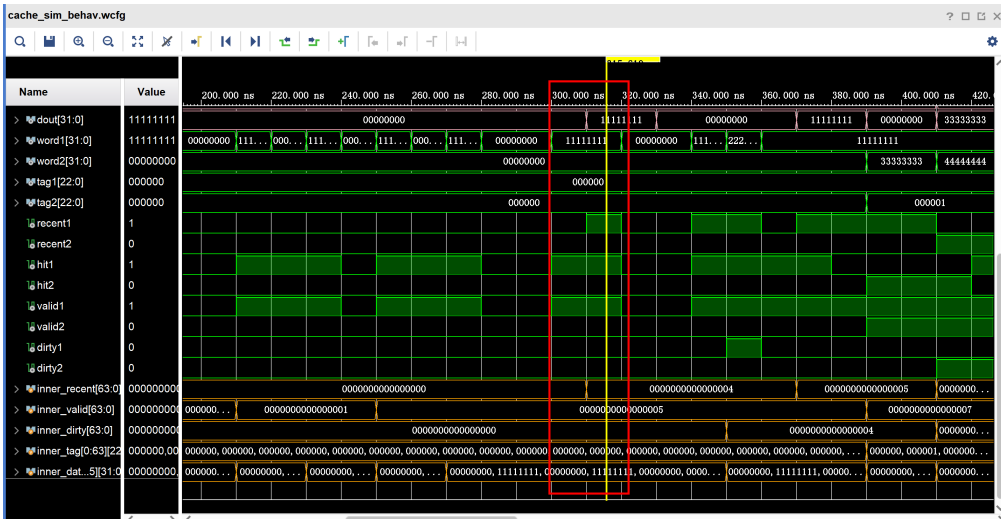
```
if (store) begin
        if (recent1) begin  // replace 2
                inner_data[addr_word2] <= din;
                inner_valid[addr_element2] <= 1'b1;
                inner_dirty[addr_element2] <= 1'b0;
                inner_tag[addr_element2] <= addr_tag;
        end else begin
        // 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;
        end
end
```

3.2 数据记录及结果分析

Since there are many cases in simulation wave, I only take some examples to show.For the convenience of explanation, all my explanations below are for the positions corresponding to the **yellow vertical lines** in the simulation diagram.

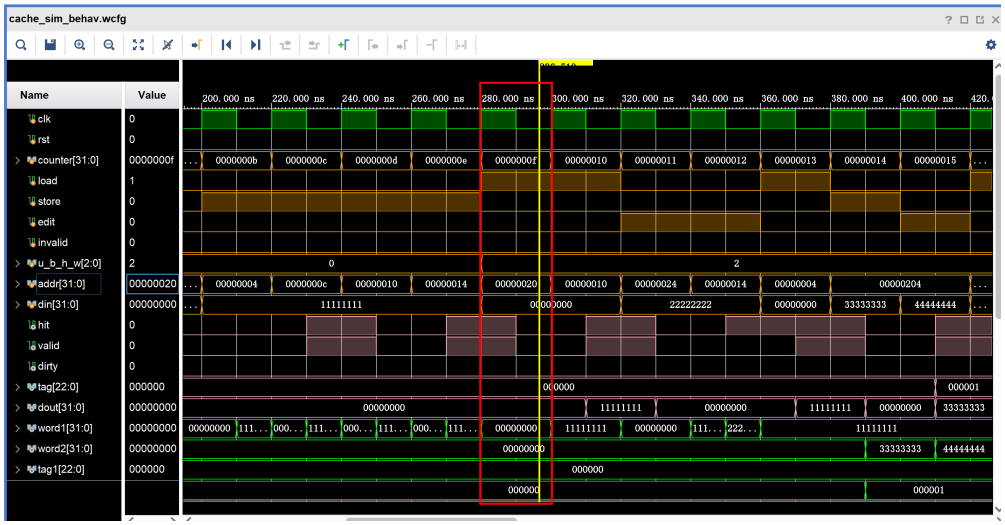
- read hit

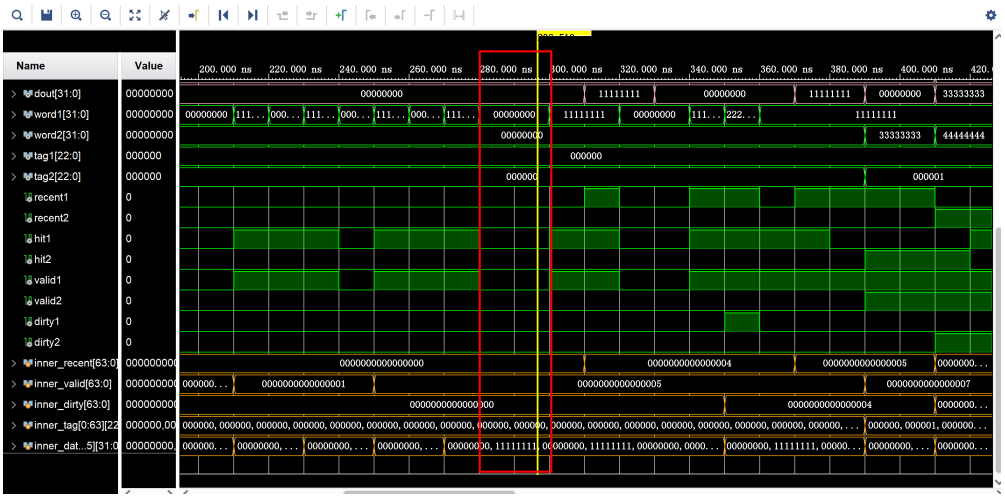




Because from addr = 0x10, we get offset = 0, index = 0x1, tag = 0. Then we have load=1, tag1=tag=0, valid1=valid=1, u_b_h_h = 010_2, we hit and load word from 0x10 to block 1 and set recent1 to 1.

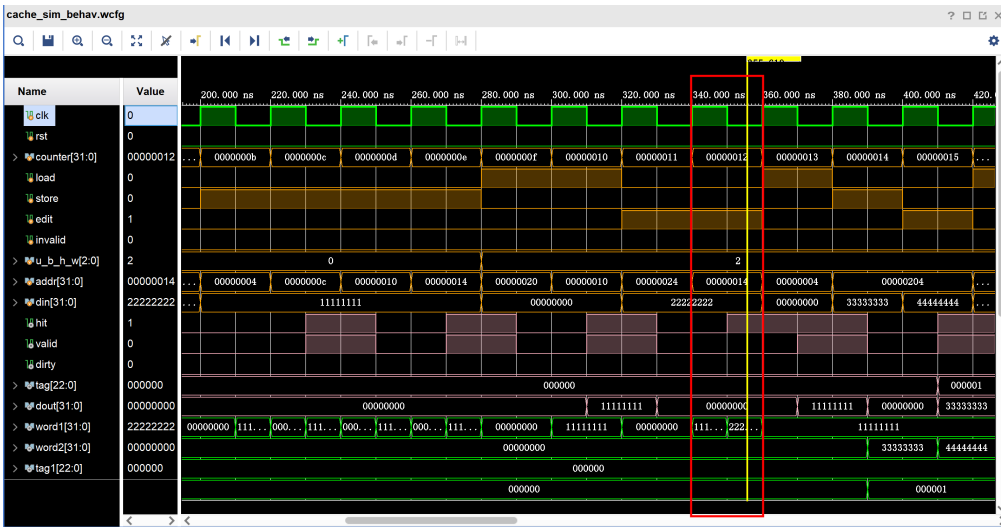
- read miss

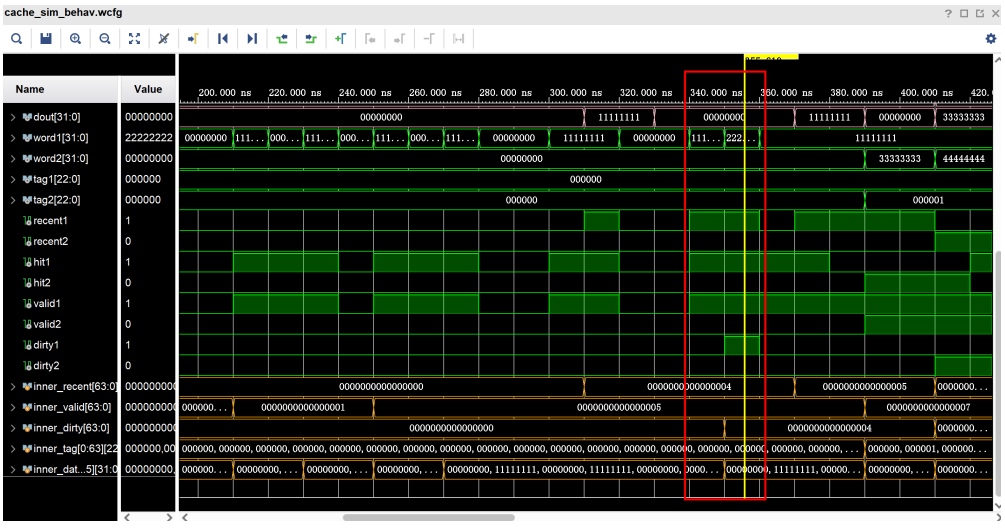




Because from addr = 0x20, we get offset = 0, index = 0x2, tag = 0. Then we have load=1, tag2=tag=0, valid2=valid=0, u_b_h_h = 010_2, we miss.

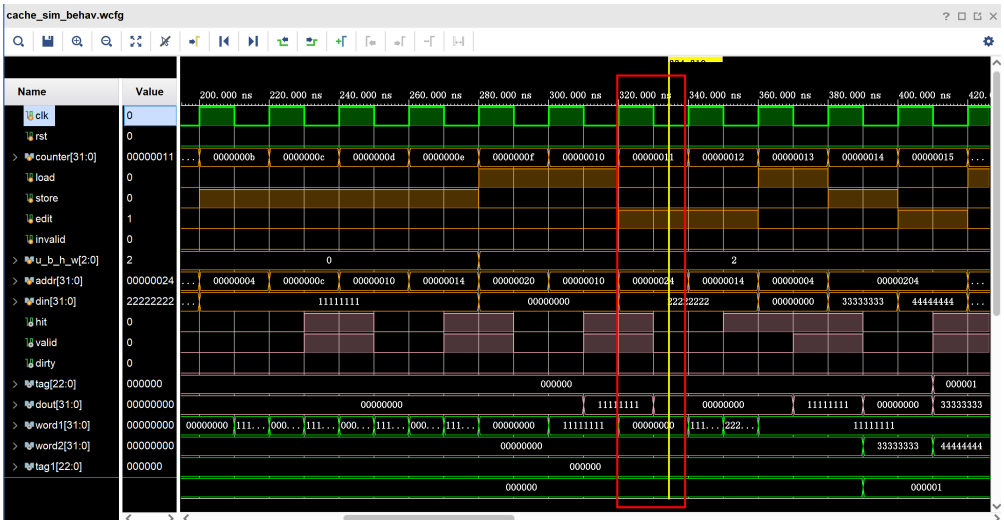
- write hit

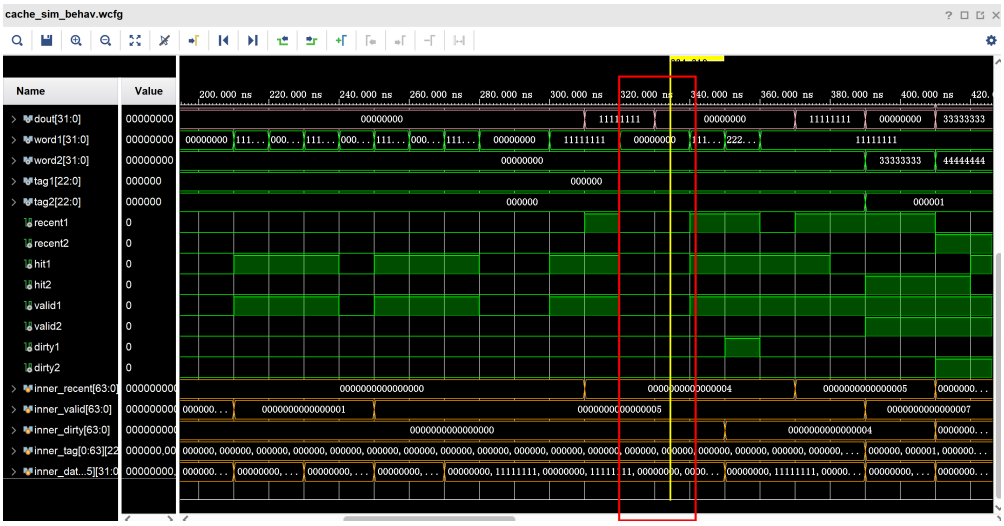




Because from addr = 0x14, we get offset = 0x4, index = 0x1, tag = 0. Then we have edit=1, tag1=tag=0, valid1=valid=1, u_b_h_h = 010_2, we hit and write word to block 1 and set recent1 to 1.

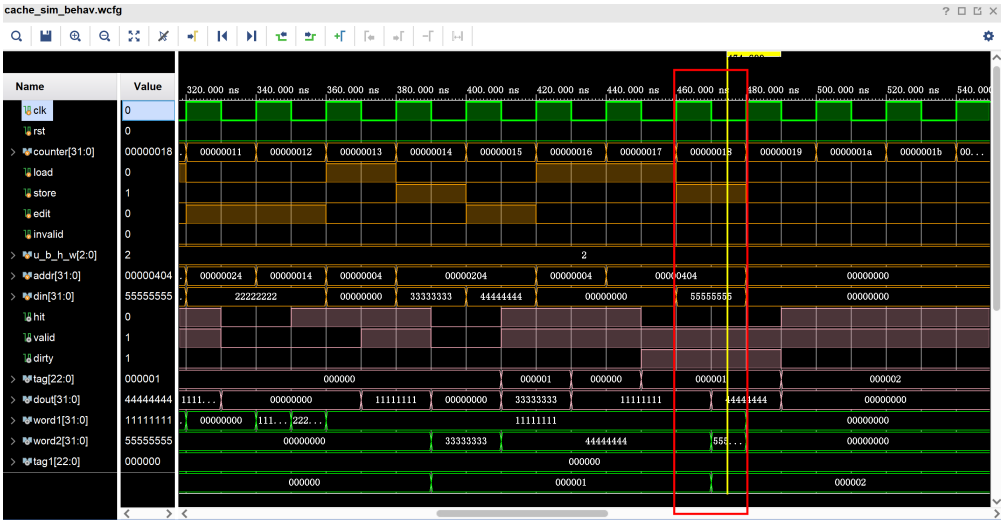
- write miss

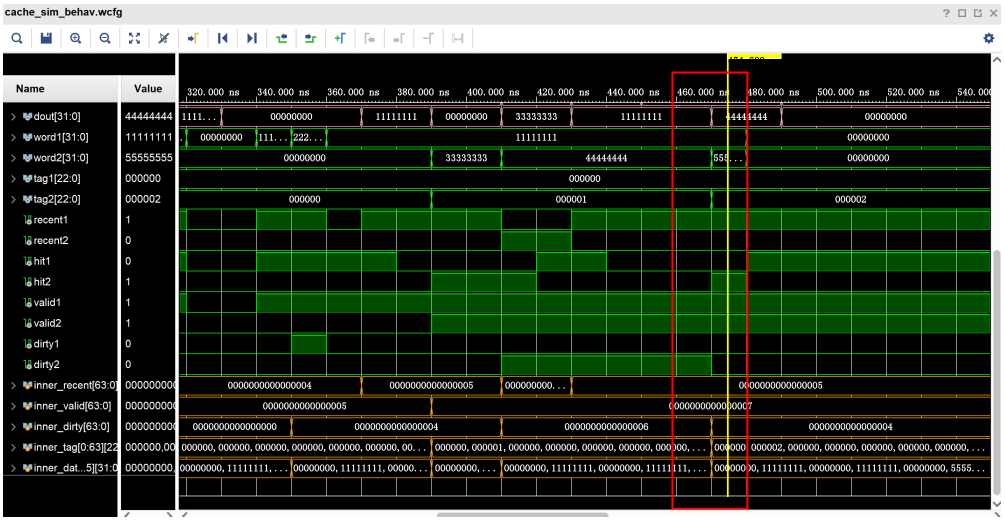




Because from addr = 0x24, we get offset = 0x4, index = 0x2, tag = 0. Then we have edit=1, tag2=tag=0, valid2=valid=0, u_b_h_h = 010_2, we miss.

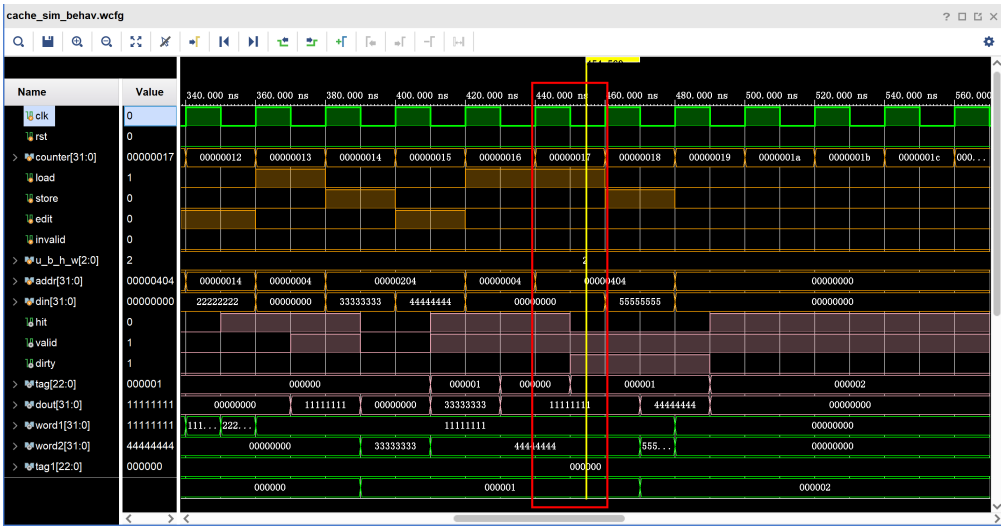
- autoplace

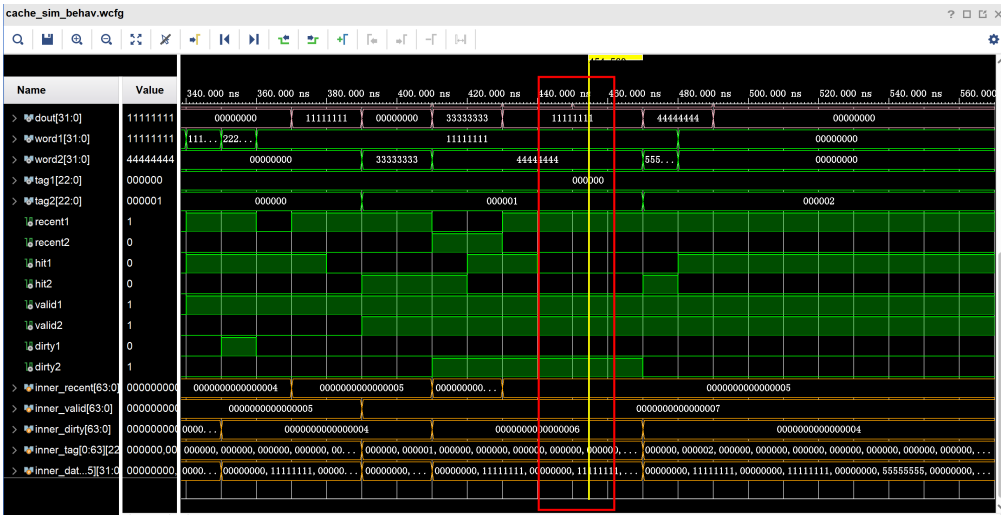




Because from addr = 0x404, we get offset = 0x4, index = 0x0, tag = 0x2. Then we have tag2=tag=2, valid2=valid=1, u_b_h_h = 010_2, we autoplace.

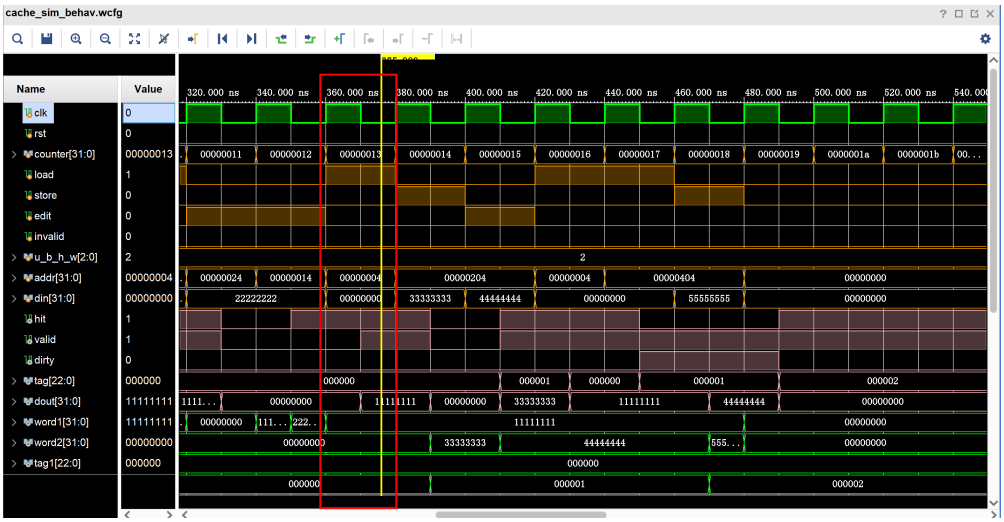
- tag miss

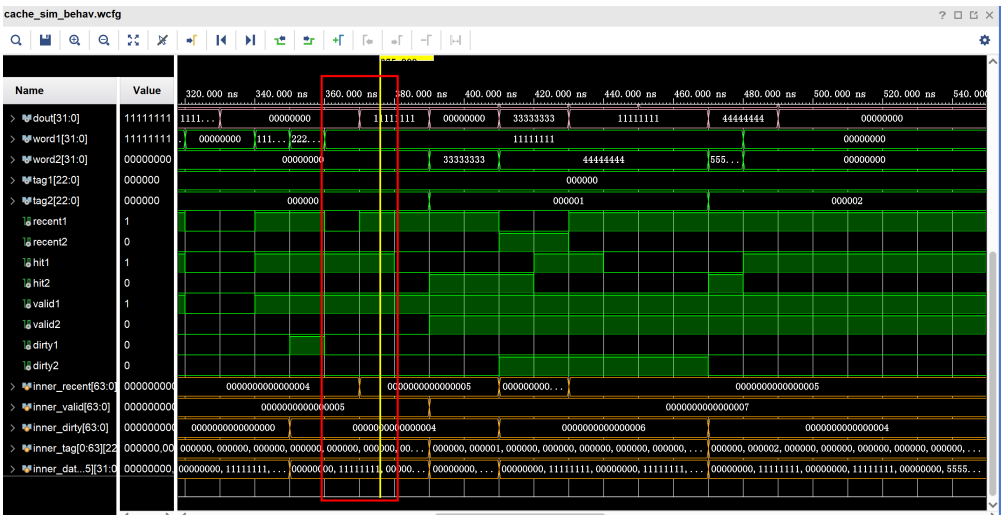




Because from addr = 0x404, we get offset = 0x4, index = 0x0, tag = 0x2. Then we have tag2≠tag, valid2=valid=1, u_b_h_h = 010_2, we tag miss.

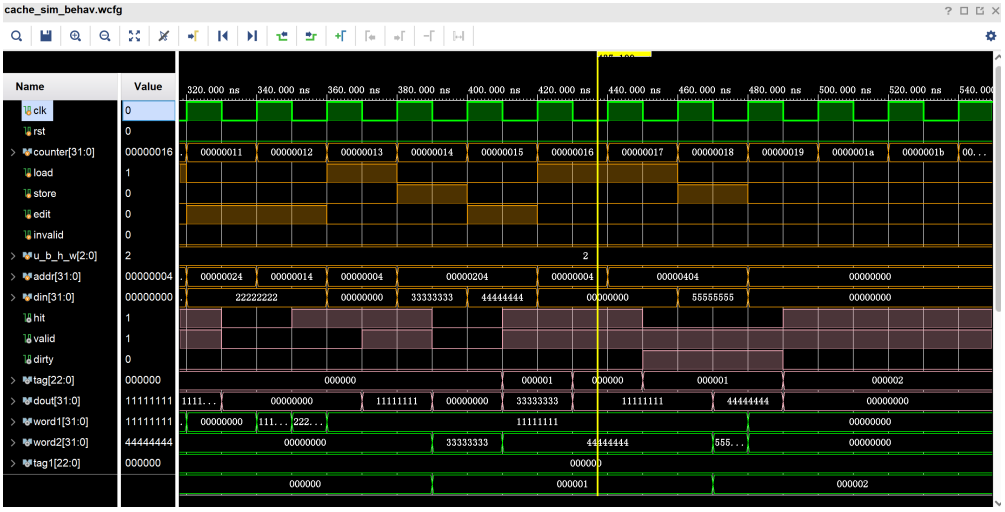
- set recent

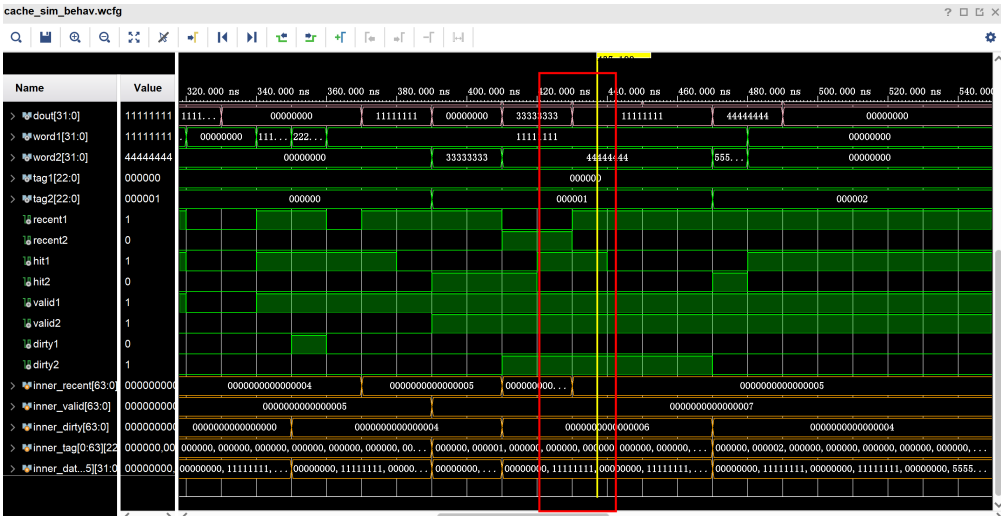




Because from addr = 0x4, we get offset = 0x4, index = 0x0, tag = 0x0. Then we have tag1=tag, valid1=valid=1, u_b_h_h = 010_2, we set recent1 to 1.

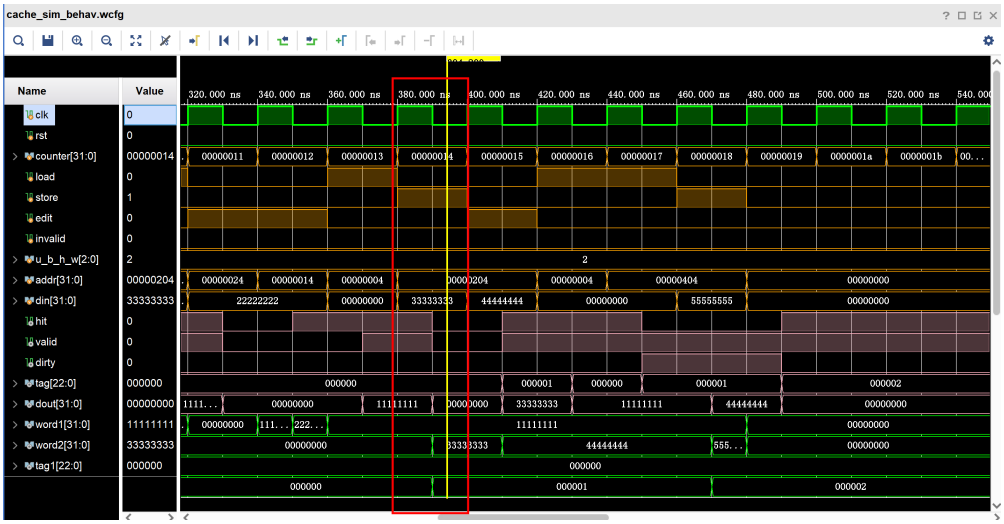
- reset recent

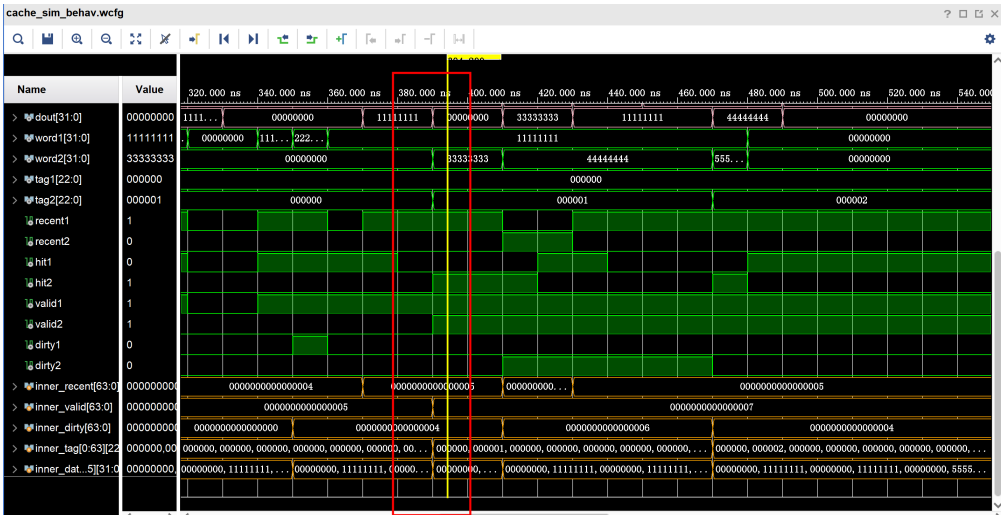




Because from addr = 0x4, we get offset = 0x4, index = 0x0, tag = 0x0. Then we have tag1=tag, valid1=valid=1, u_b_h_h = 010_2, we set recent1 to 1 and recent2 to 0.

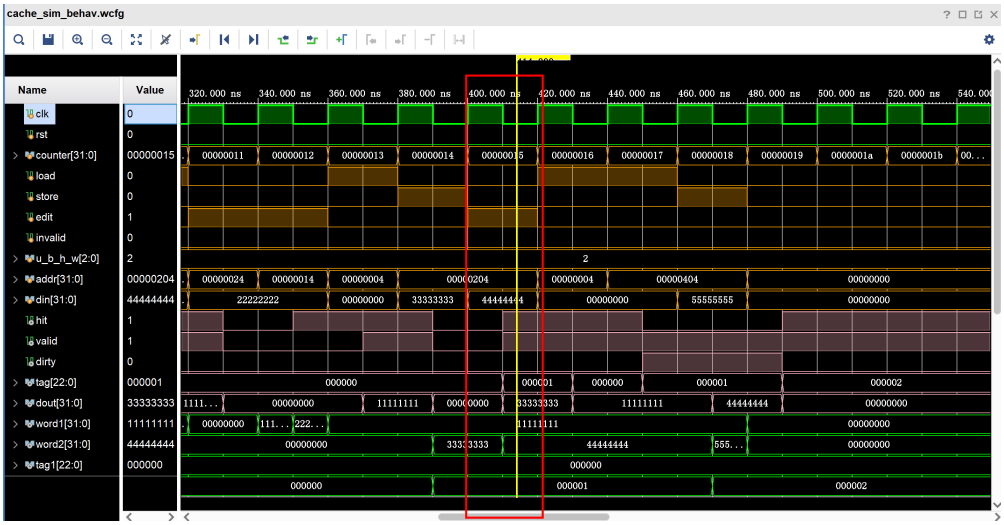
- due recent

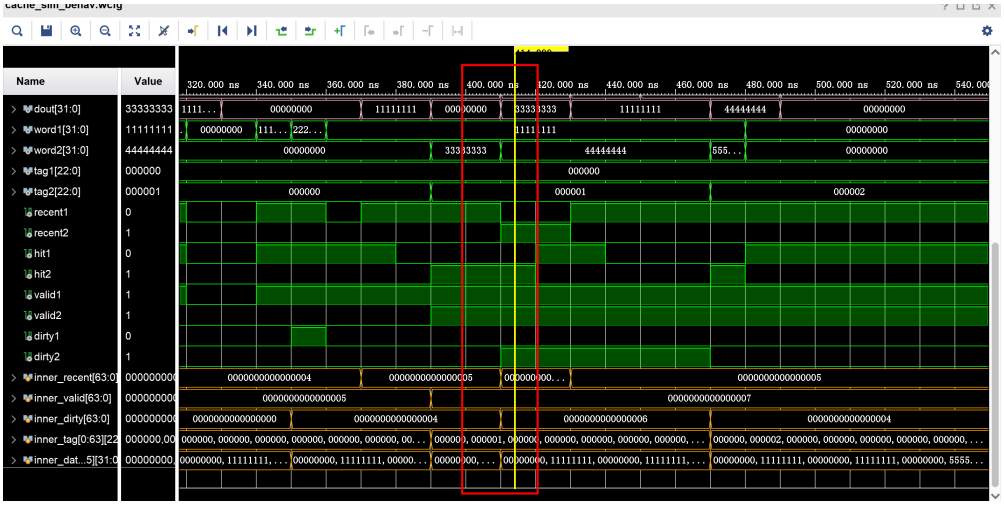




Because from addr = 0x204, we get offset = 0x4, index = 0x0, tag = 0x1. Then we have store=1,tag2=tag, valid2=valid=1.

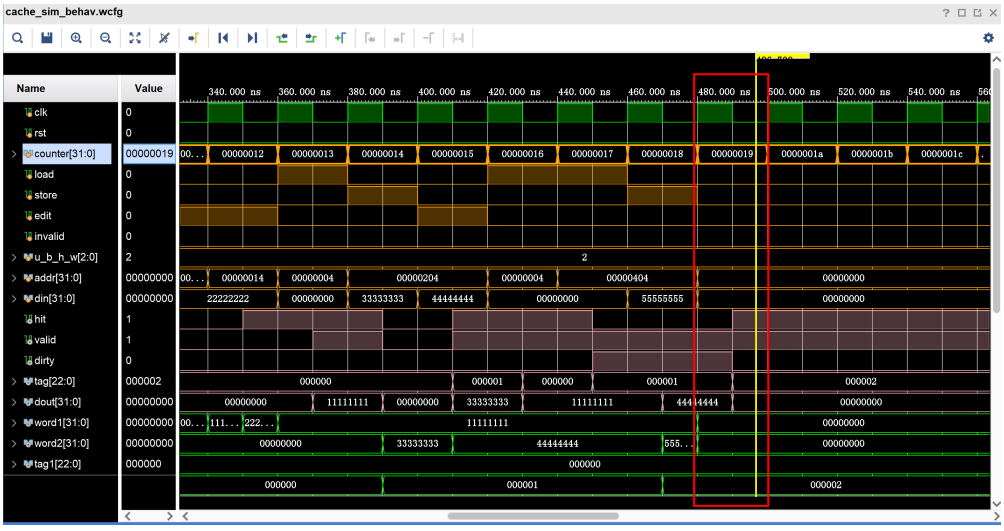
- edit

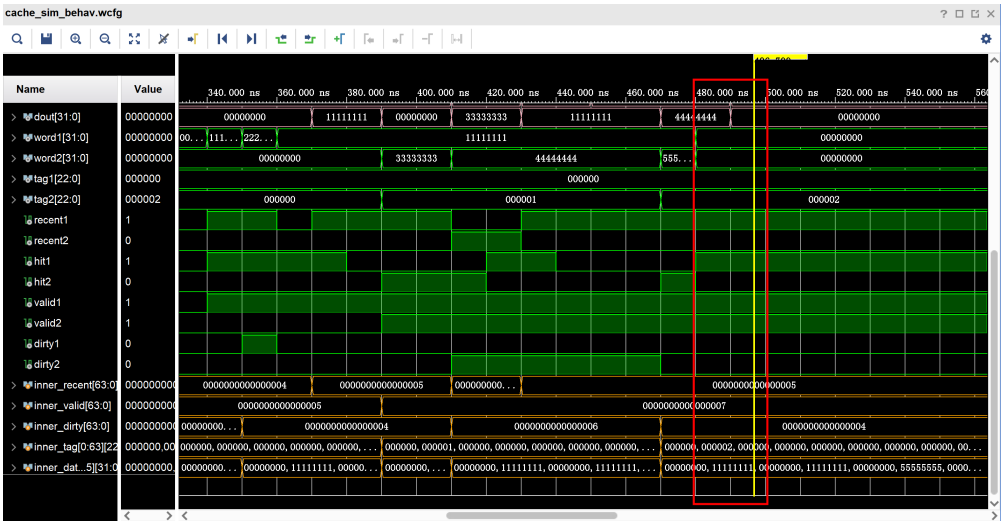




Because from addr = 0x204, we get offset = 0x4, index = 0x0, tag = 0x1. Then we have tag2=tag, valid2=valid=1, u_b_h_h = 010_2, we set recent1 to 0, rececnt2 to 1 and dirty2 to 1.

- clear





4 讨论与心得

This lab requires a certain understanding of cache to do well. Although the code can be completed according to the context, it still took me a lot of time to re-understand the cache.