浙江大学
ZHEJIANG UNIVERSITY

Computer Architecture Experiment

Topic 3. Cache Design

浙江大学计算机学院

2021年9月

Outline

- Experiment Purpose
- Experiment Task
- Basic Principle
- Operating Procedures
- Checkpoints

Experiment Purpose

- 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.

Experiment Task

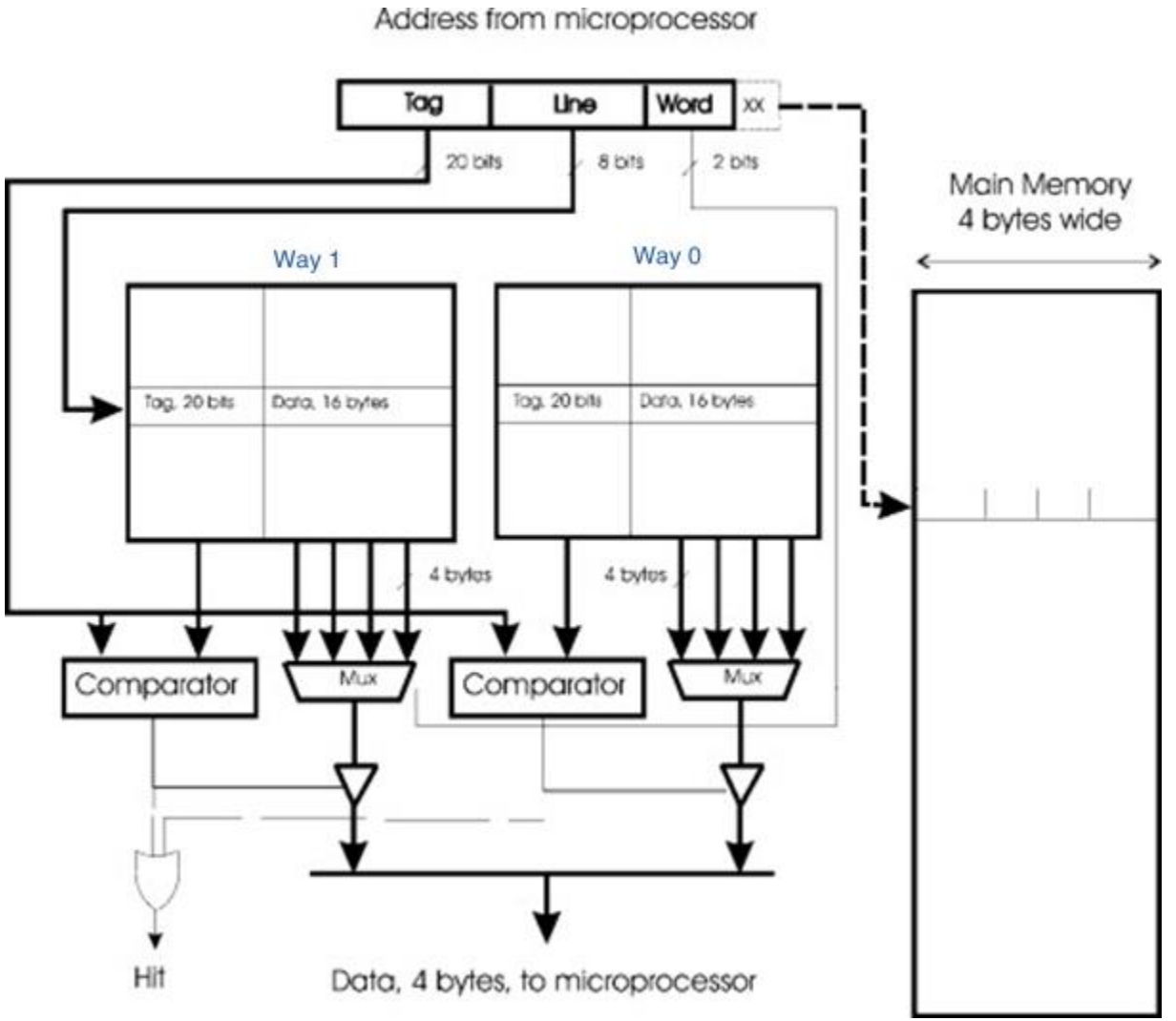
- **Design of Cache Line and CMU.**
- **Verify the Cache Line and CMU.**
- **Observe the Waveform of Simulation.**

Cache Line

LRU	V	D	Tag	Data

Cache Mode

- **2-way set associative**
- **Write Back**
- **Write Allocate**



Address

- Bytes of word = 4, WORD_BYTES_WIDTH = 2
- Words of line = 4, LINE_WORDS_WIDTH = 2
- Tag bits = 23
- Address bits = 32
- Ways = 2
- LINE_INDEX_WIDTH = ADDR_BITS - TAG_BITS - LINE_WORDS_WIDTH - WORD_BYTES_WIDTH = ?
- Line Number = ?

Cache Line Memory

- reg [LINE_NUM-1:0] inner_recent = 0;
- reg [LINE_NUM-1:0] inner_valid = 0;
- reg [LINE_NUM-1:0] inner_dirty = 0;
- reg [TAG_BITS-1:0] inner_tag [0:LINE_NUM-1];
- reg [WORD_BITS-1:0] inner_data [0:LINE_NUM*LINE_WORDS-1];

Input and output Signals of Cache Line

```
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 store,  // set valid to 1 and reset dirty to 0
        input wire edit,  // 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
        );
```

Simulation Example(1)

```
// init
32'd10: begin
            load <= 0;
            store <= 1;
            edit <= 0;

            din <= 32'h11111111;
            addr <= 32'h00000004;
end

32'd11: begin
            addr <= 32'h0000000C;
end

32'd12: begin
            addr <= 32'h00000010;
end

32'd13: begin
            addr <= 32'h00000014;
end
```

```
// read miss
32'd14: begin
            load <= 1;
            store <= 0;
            edit <= 0;

            u_b_h_w <= 3'b010;
            din <= 0;
            addr <= 32'h00000020;
end

// read hit
32'd15: begin
            u_b_h_w <= 3'b010;
            addr <= 32'h00000010;
end
```

Simulation Example(2)

```
// write miss
32'd16: begin
            load <= 0;
            store <= 0;
            edit <= 1;

            u_b_h_w <= 3'b010;
            din <= 32'h22222222;
            addr <= 32'h000000024;
end

// write hit
32'd17: begin
            u_b_h_w <= 3'b010;
            addr <= 32'h00000014;
end
```

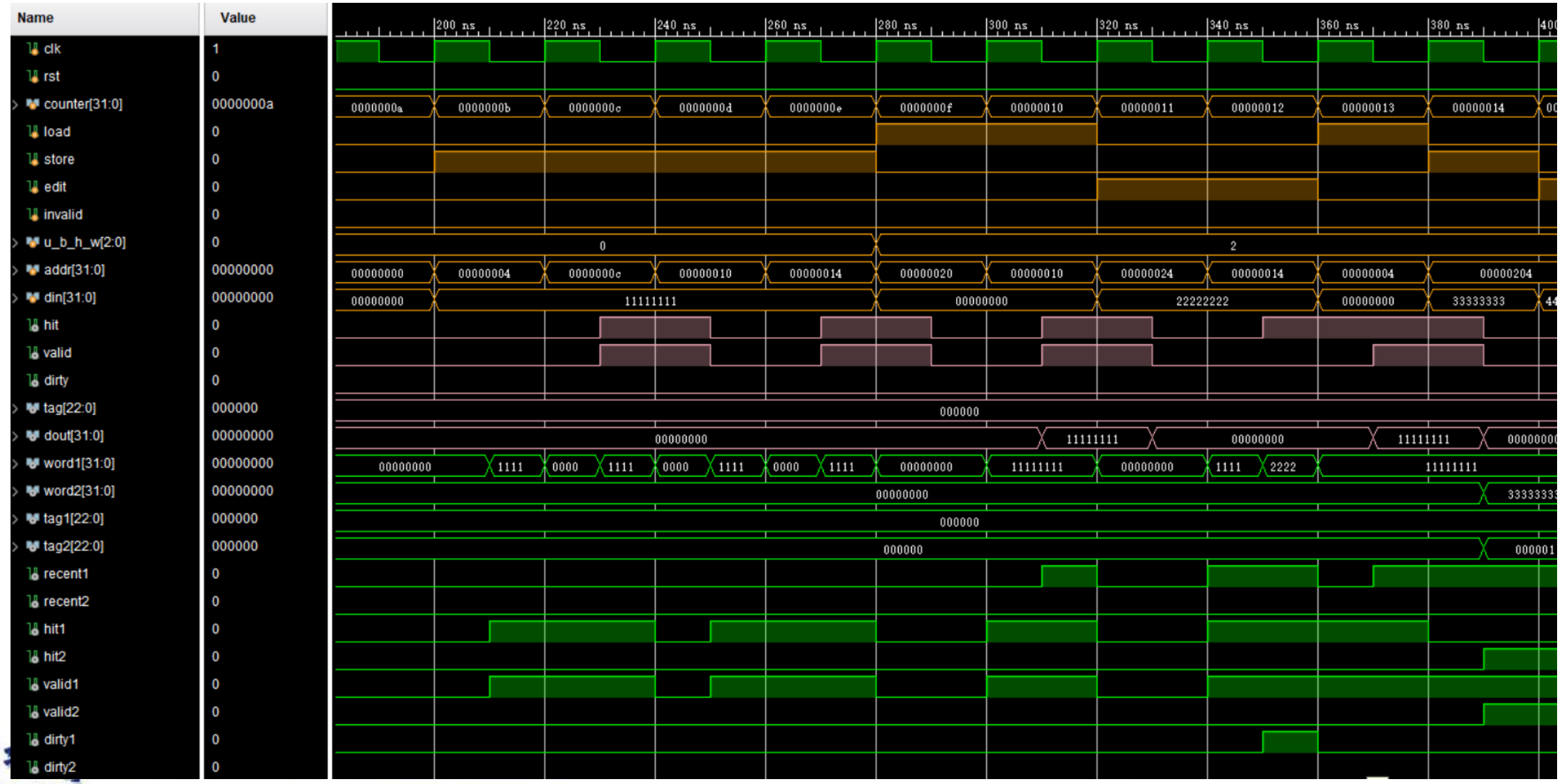
```
// read line 0 of set 0, set recent bit
            32'd18: begin
                        load <= 1;
                        store <= 0;
                        edit <= 0;

                        u_b_h_w <= 3'b010;
                        din <= 0;
                        addr <= 32'h00000004;
            end

// store to line 1 of set 0 due to line 0 recent
            32'd19: begin
                        load <= 0;
                        store <= 1;
                        edit <= 0;

                        u_b_h_w <= 3'b010;
                        din <= 32'h33333333;
                        addr <= 32'h00000204;
            end
```

Simulation Example(3)

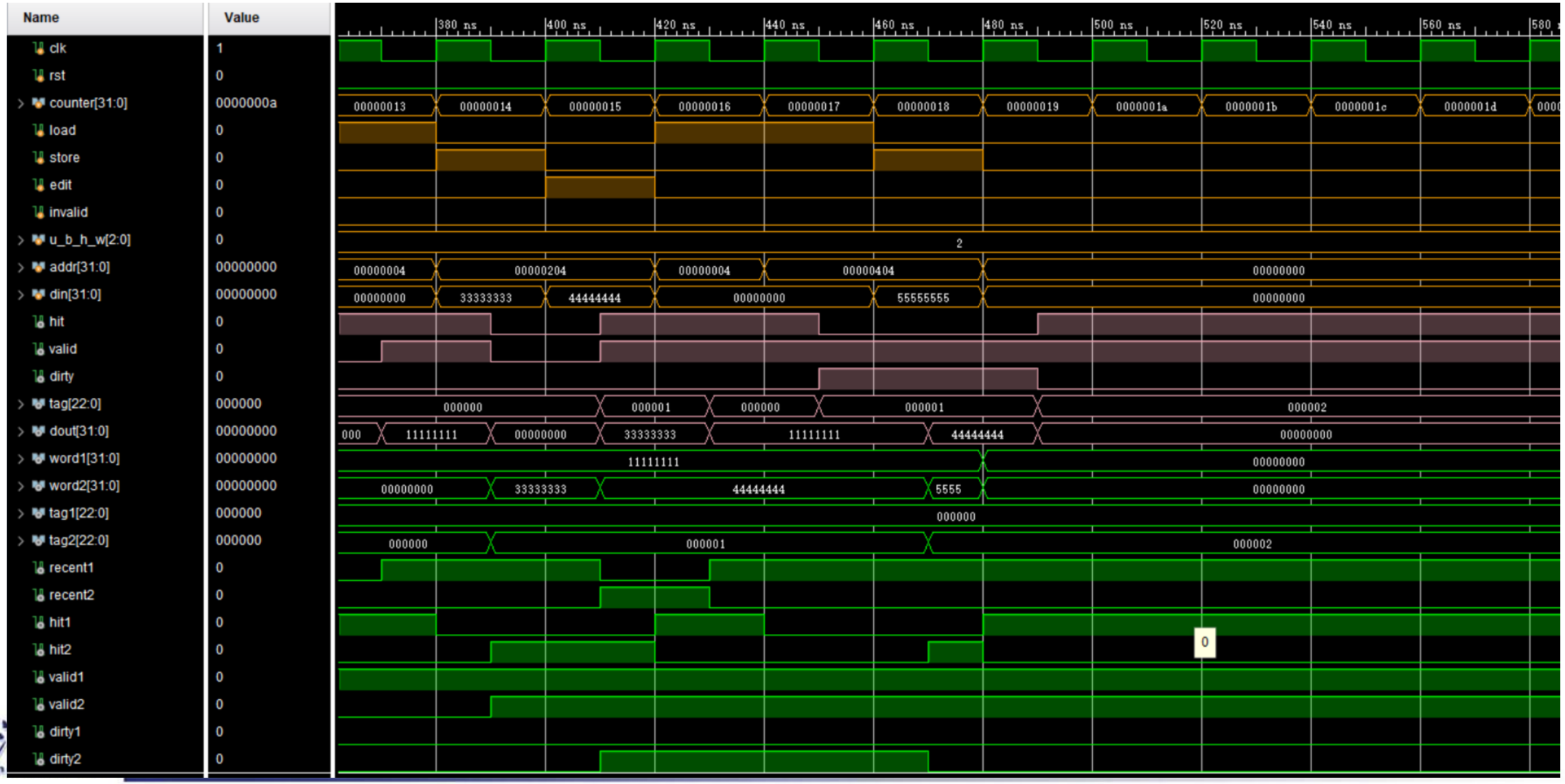
```
// edit line 1 of set 0, set dirty & recent
        32'd20: begin
                load <= 0;
                store <= 0;
                edit <= 1;

                u_b_h_w <= 3'b010;
                din <= 32'h44444444;
                addr <= 32'h00000204;
        end

// read line 0 of set 0, set recent bit
        32'd21: begin
                load <= 1;
                store <= 0;
                edit <= 0;

                u_b_h_w <= 3'b010;
                din <= 0;
                addr <= 32'h00000004;
        end
```

```
// read miss, tag mismatch. output tag (of line 1), valid and dirty == 1
        32'd22: begin
                load <= 1;
                store <= 0;
                edit <= 0;

                u_b_h_w <= 3'b010;
                din <= 32'h0;
                addr <= 32'h00000404;
        end

// auto replace line 1 of set 0
        32'd23: begin
                load <= 0;
                store <= 1;
                edit <= 0;

                u_b_h_w <= 3'b010;
                din <= 32'h55555555;
                addr <= 32'h00000404;
        end
```

Simulation Example(1)



Simulation Example(2)



Simulation

- **Write Simulation Code yourself**
  - cache initialization
  - read
    - **miss**
    - **hit**
  - write
    - **miss**
    - **hit**

Checkpoints

- **CP1:**

  Waveform Simulation of Cache Line.

Thanks!