浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：张天逸

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：3220106424

指导教师：姜晓红

2024 年　　10 月　　12　　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：张天逸 专业：计算机科学与技术 学号：3220106424

同组学生姓名： 指导老师：姜晓红

实验地点： 实验日期：2024 年 10 月 12 日

20% bonus: 4-way set associative

一、 实验目的和要求

理解 Cache 中 Tag, Valid, Dirty 的作用和 Least Recently Used 的概念

理解 Cache Management Unit 的作用及与 CPU 结合的方法

理解 CMU 的各种 State，以及处理 Hit，Miss 等状况的方法

掌握验证 Cache 和 CMU 设计正确性的方法

二、 实验内容和原理

cache line 的 tag，dirty，valid，lru 信号如何得到

由于 address 总是由 tag，index，word，byte 组成，所以根据 addr_define.vh 的定义，可以填补 addr_tag 的内容

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

LRU Cache 的替换原则就是将最近最少使用的内容替换掉。其实， LRU 译成最久未使用会更形象， 因为该算法每次替换掉的就是一段时间内最久没有使用过的内容。因而当使用 1 号块时传出 2 号块的 valid，dirty，tag，当使用 2 号块时传出 1 号块的 valid，dirty，tag

```
valid <= hit1 | hit2 | (recent1 ? valid2 : valid1);
dirty <= recent1 ? dirty2 : dirty1;
tag <= (hit1) ? tag1 : (hit2) ? tag2 : {TAG_BITS{1'b0}};
```

对于 load，store，replace 都是将 hit 块的 inner_recent 参数设置为 1，另一块设置为 0，以 load 为例子

```
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_element1] <= 1'b0;
                inner_recent[addr_element2] <= 1'b1;
            end
        end
```

对于 store，要记得更新 hit 块的 dirty 参数，以 hit1 为例子

```
inner_dirty[addr_element1] <= 1'b1;
```

对于 replace，需要更新 data，tag 以及自身的 valid 值，另外需要更新 dirty 为 0，recent 的更新和之前两种命令一样

```
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_element1] <= 1'b0;
                inner_recent[addr_element2] <= 1'b1;
            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
```

三、 实验过程和数据记录及结果分析

Cache_sim.v

```
`timescale 1ns / 1ps
```

```
module cache_sim;
```

```
    // Inputs
```

```
reg clk;
reg rst;
reg [31:0] addr;
reg load;
reg replace;
reg store;
reg invalid;
reg [2:0] u_b_h_w;
reg [31:0] din;
```

```
// Outputs
wire hit;
wire [31:0] dout;
wire valid;
wire dirty;
wire [22:0] tag;
```

```
// Instantiate the Unit Under Test (UUT)
cache uut (
    .clk(clk),
    .rst(rst),
    .addr(addr),
    .load(load),
    .replace(replace),
    .store(store),
    .invalid(invalid),
    .u_b_h_w(u_b_h_w),
    .din(din),
    .hit(hit),
    .dout(dout),
    .valid(valid),
    .dirty(dirty),
    .tag(tag)
);
```

```
initial begin
    clk = 1;
    forever #0.5 clk = ~clk;
end
```

```
reg [31:0]counter = -1;
```

```
always @(posedge clk) begin
    counter = counter + 32'b1;
```

```
        case (counter)
            // Initialize Inputs
            32'd0: begin
                rst <= 1;
                addr <= 0;
                load <= 0;
                replace <= 0;
                store <= 0;
                invalid <= 0;
                u_b_h_w <= 0;
                din <= 0;
            end

            32'd1: begin
                rst <= 0;
                load <= 0;
                store <= 0;
                replace <= 1;
                invalid <= 0;
                addr <= 32'h4;
                din <= 32'h12345678;
                u_b_h_w <= 2;
            end
```

```
            32'd2: begin
                load <= 0;
                store <= 0;
                replace <= 1;
                invalid <= 0;
                addr <= 32'hc;
                din <= 32'h23456789;
                u_b_h_w <= 2;
            end
```

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

```
            32'd4: begin
                load <= 0;
                store <= 0;
                replace <= 1;
                invalid <= 0;
                addr <= 32'h14;
                din <= 32'h45678901;
                u_b_h_w <= 2;
            end
```

```
            32'd5: begin
                load <= 1;
                store <= 0;
                replace <= 0;
                invalid <= 0;
                addr <= 32'h20;
                din <= 32'h0;
                u_b_h_w <= 2;
            end
```

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

```
            32'd7: begin
                load <= 0;
                store <= 1;
                replace <= 0;
                invalid <= 0;
                addr <= 32'h24;
                din <= 32'h56789012;
                u_b_h_w <= 1;
            end
```

```
            32'd8: begin
                load <= 0;
                store <= 1;
```

```
            replace <= 0;
            invalid <= 0;
            addr <= 32'h11;
            din <= 32'h67890123;
            u_b_h_w <= 0;
        end
```

```
        32'd9: begin
            load <= 0;
            store <= 0;
            replace <= 1;
            invalid <= 0;
            addr <= 32'h200;
            din <= 32'h78901234;
            u_b_h_w <= 2;
        end
```

```
        32'd10: begin
            load <= 0;
            store <= 1;
            replace <= 0;
            invalid <= 0;
            addr <= 32'h204;
            din <= 32'h89012345;
            u_b_h_w <= 0;
        end
```

```
        32'd11: begin
            load <= 0;
            store <= 0;
            replace <= 0;
            invalid <= 0;
            addr <= 32'h400;
            din <= 32'h0;
            u_b_h_w <= 2;
        end
```

```
        32'd12: begin
            load <= 1;
            store <= 0;
            replace <= 0;
            invalid <= 0;
            addr <= 32'hd;
            din <= 32'h0;
```

```
            u_b_h_w <= 4;
        end
```

```
        32'd13: begin
            load <= 0;
            store <= 0;
            replace <= 1;
            invalid <= 0;
            addr <= 32'h60;
            din <= 32'ha98ac7;
            u_b_h_w <= 2;
        end
```

```
        32'd14: begin
            load <= 0;
            store <= 0;
            replace <= 1;
            invalid <= 0;
            addr <= 32'h160;
            din <= 32'h153158e;
            u_b_h_w <= 2;
        end
```

```
        32'd15: begin
            load <= 0;
            store <= 0;
            replace <= 1;
            invalid <= 0;
            addr <= 32'h260;
            din <= 32'h1fca055;
            u_b_h_w <= 2;
        end
```

```
        32'd16: begin
            load <= 0;
            store <= 0;
            replace <= 1;
            invalid <= 0;
            addr <= 32'h360;
            din <= 32'h2a62b1c;
            u_b_h_w <= 2;
        end
```

```
        32'd17: begin
```

```
                load <= 1;
                store <= 0;
                replace <= 0;
                invalid <= 0;
                addr <= 32'h560;
                din <= 32'h0;
                u_b_h_w <= 2;
            end
```

```
            32'd18: begin
                load <= 1;
                store <= 0;
                replace <= 0;
                invalid <= 0;
                addr <= 32'h60;
                din <= 32'h0;
                u_b_h_w <= 2;
            end
```

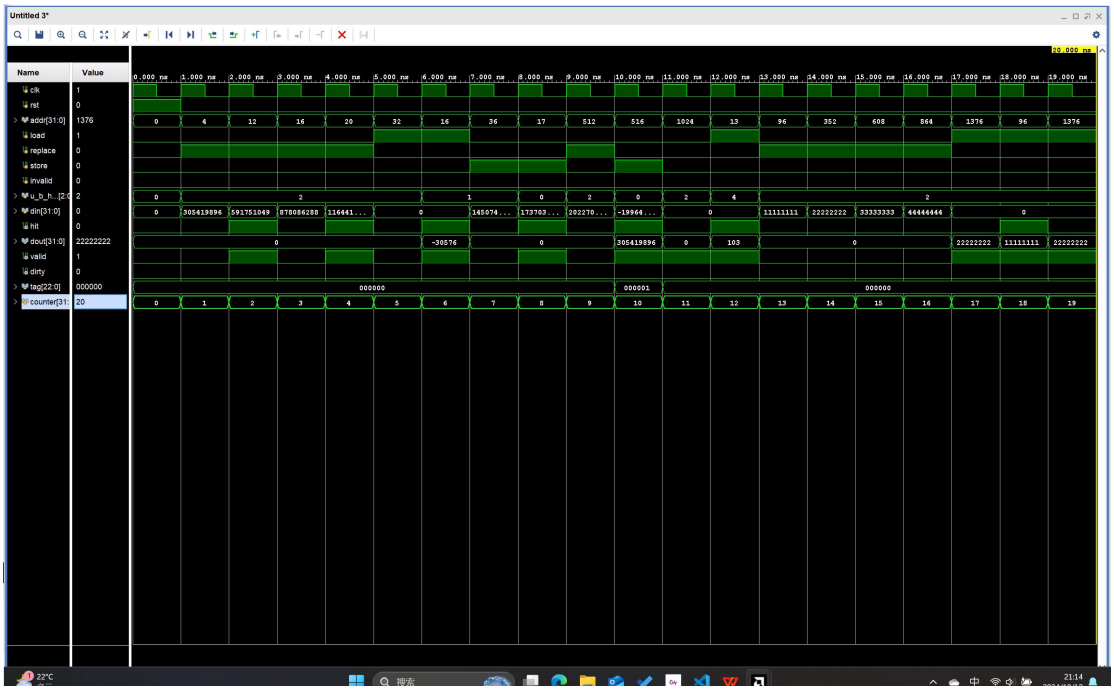
```
            32'd19: begin
                load <= 1;
                store <= 0;
                replace <= 0;
                invalid <= 0;
                addr <= 32'h560;
                din <= 32'h0;
                u_b_h_w <= 2;
            end
```

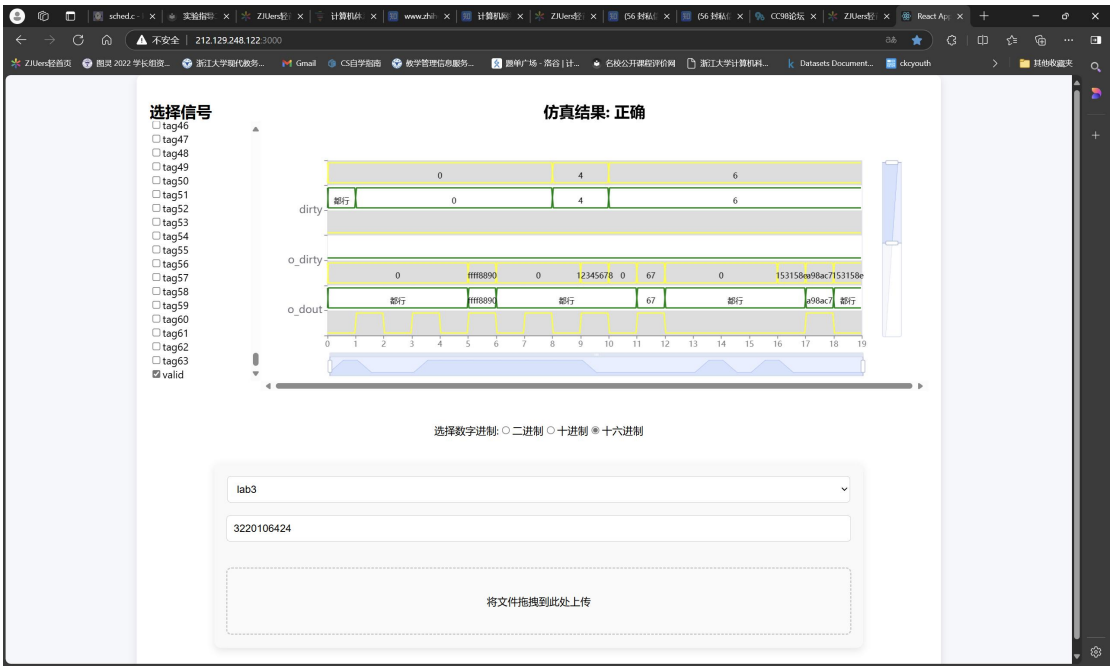
```
            default: begin
                $finish;
            end
        endcase
    end
```

```
endmodule
```

在 counter = 0 时，Cache 进行复位（rst = 1），所有寄存器和存储器内容被清零，初始化为无效状态。在 Counter 为 1、2、3、4、9、13、14、15、16 时，replace 信号为 1，表明发生了替换操作。这时，Cache 将根据 LRU 策略选择最近最少使用的 Cache 块来替换。替换时，valid 位会被设置为 1，dirty 位被重置为 0，表明新的数据块已经加载到 Cache 中。Replace: 在第 1 个时刻，地址 addr = 4，数据 din = 0x12345678 被加载到 Cache 中。在第 9 个时刻，地址 addr = 0x200，数据 din = 0x78901234 被加载。Load:当 load 信号为 1 时，表示读操作。如果 Cache 命中（即 hit 为 1），则从 Cache 中读取数据。否则，如果没有命中，则需要从外部内存中获取数据。

在第 5 个时刻，load 信号为 1，地址 addr = 0x20，对应的数据为 dout = 0（因为此时没有命中）。在第 17 个时刻，load 信号为 1，地址 addr = 0x560，同样没有命中，数据输出为 dout = 0。Store:当 store 信号为 1 时，表示写操作。Cache 会将数据写入对应的地址，并将 dirty 位设置为 1，表示该数据已经被修改，但尚未写回到主存。在第 7 个时刻，store 信号为 1，地址 addr = 0x24，数据 din = 0x56789012 被写入，dirty 位被设置为 1。LRU:在每次 load 命中时（如 counter = 6 处），Cache 更新 LRU 信息，将最近使用的块的 LRU 位设置为 1，未使用的块的 LRU 位设置为 0，以确保下次替换时可以替换最近最少使用的块。当 replace 信号激活且发生替换时，Cache 根据 LRU 位决定替换哪个块。在仿真过程中可以看到，例如在 counter = 9 时，Cache 根据 LRU 替换了地址 0x200 的数据。





四、　讨论与心得

本次实验作为 lab4 的前置实验可以说是相当简单的，需要填补的部分也不多，总体来说是前三次实验中最简单的一个啦。