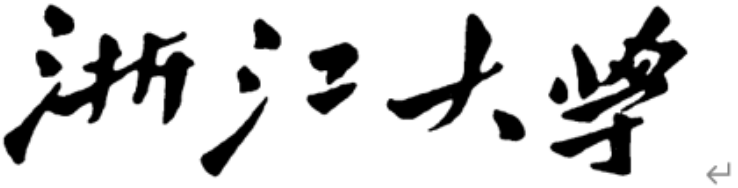


课程名称:	**计算机组成与设计**
姓 名:	杨吉祥
学 院:	计算机科学与技术学院
系:	竺可桢学院图灵班
专 业:	计算机科学与技术
学 号:	3230106222
指导教师:	刘海风

浙江大学实验报告

课程名称：计算机组成与设计　　实验类型： 综合

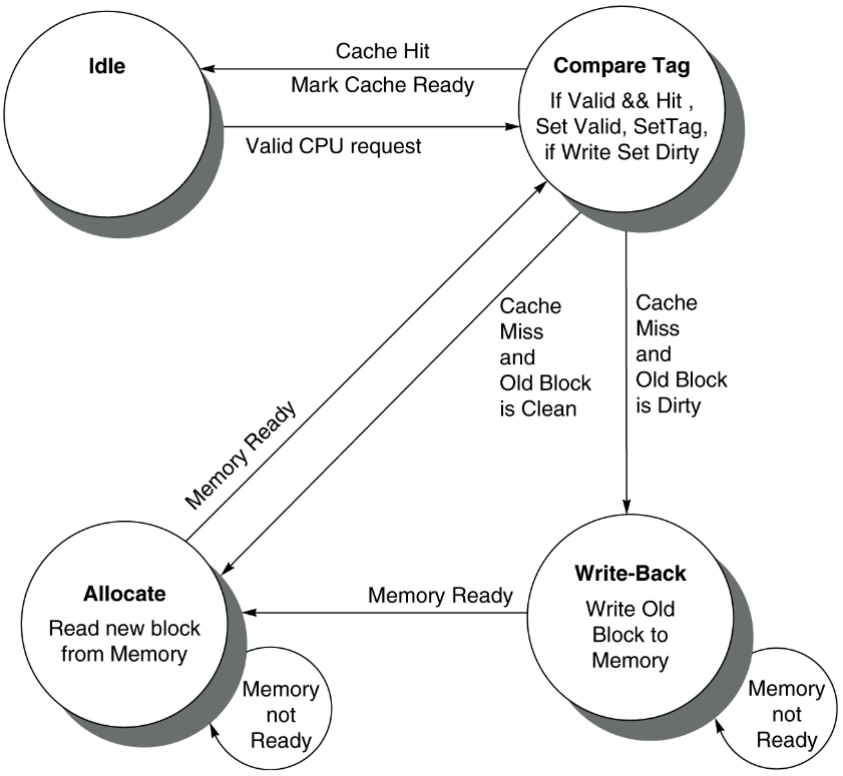
实验项目名称：cache设计

学生姓名： 杨吉祥 专业： 计算机科学与技术 学号： 3230106222

报告日期：12.27

一、操作方法与实验步骤

- **Cache控制采用FSM实现**



我们这里实现的是两路相联的cache,因为block size为16 bytes,cache size为4KB,所以一共有256个blocks,由于是两路相联,所以有128组,那index为7位,16bytes=4words,那offset为2位,32-2-7=23,tag为23位。cache还需要额外的三位分别表示valid,dirty,lru,valid bit表示该块是否有效,dirty bit表示该块是否被修改,lru bit表示该块最近是否被使用,所以cache一个块的大小为128+23+3=154位。Cache控制采用FSM实现,定义了idle,compare_tag,allocate,write_back四个状态。

- idle

  该状态下判断是否有读写内存请求,如果有就进入compare_tag状态,否则还在idle状态。rw为0表示无读写请求,1表示读请求,2表示写请求。

- compare_tag

  该状态判断地址对应的组的tag是否命中,命中则进入idle状态,否则根据地址对应的组中的两个块的lru位来判断要替换哪个块,如果要被替换的块的dirty位为1,则进入write_back状态,否则进入allocate状态。

- allocate

该状态下将地址对应的内存中的块搬进cache中,之后进入compare_tag状态。

- write_back

  该状态下将cache中要被替换的块的数据写回内存中,同时要清空dirty位,之后进入allocate状态。

- **cache.v**

```
`define idle 2'b00
`define compare_tag 2'b01
`define allocate 2'b10
`define write_back 2'b11
`define lru 151
`define dirty 152
`define valid 153

module cache(
    input clk,
    input rst,
    input [31:0]addr,
    input [31:0]write_data,
    input [127:0]mem_data,
    input mem_ready,
    input [1:0]rw,//0:non 1:read 2:write
    output reg cache_ready,
    output reg mem_rw,
    output reg [127:0]mem_write_data,
    output reg [31:0]read_data,
    output [153:0]cache00,
    output [153:0]cache01
);

reg [153:0]cache_data[127:0][1:0];
reg [1:0]state;
reg write_back;
wire valid1,valid2;
wire dirty1,dirty2;
wire lru1,lru2;
wire [1:0]offset;
wire [6:0]index;
wire [22:0]tag;

assign valid1 = cache_data[index][0][`valid];
assign valid2 = cache_data[index][1][`valid];
assign dirty1 = cache_data[index][0][`dirty];
assign dirty2 = cache_data[index][1][`dirty];
assign lru1 = cache_data[index][0][`lru];
assign lru2 = cache_data[index][1][`lru];
assign offset = addr[1:0];
assign index = addr[8:2];
assign tag = addr[31:9];

always @(posedge clk or posedge rst)begin
    if(rst)begin
        state <= `idle;
    end
    else begin
```

```
        case(state)
            `idle:begin
                cache_ready <= 1'b0;
                mem_rw <= 1'b0;
                mem_write_data <= 128'b0;
                read_data <= 32'b0;
                if(rw==2'b1 || rw==2'b10)begin
                    state <= `compare_tag;
                end
                else begin
                    state <= `idle;
                end
            end
            `compare_tag:begin
                if(cache_data[index][0][150:128]==tag && valid1)begin
                    if(rw==2'b1)begin
                        read_data <= cache_data[index][0][(32*offset)+:32];
                        cache_data[index][0][`lru] <= 1'b1;
                        cache_data[index][1][`lru] <= 1'b0;
                        cache_ready <= 1'b1;
                        state <= `idle;
                    end
                    else begin
                        cache_data[index][0][(32*offset)+:32] <= write_data;
                        cache_data[index][0][`dirty] <= 1'b1;
                        cache_data[index][0][`lru] <= 1'b1;
                        cache_data[index][1][`lru] <= 1'b0;
                        cache_ready <= 1'b1;
                        state <= `idle;
                    end
                end
                else if(cache_data[index][1][150:128]==tag && valid2)begin
                    if(rw==2'b1)begin
                        read_data <= cache_data[index][1][(32*offset)+:32];
                        cache_data[index][1][`lru] <= 1'b1;
                        cache_data[index][0][`lru] <= 1'b0;
                        cache_ready <= 1'b1;
                        state <= `idle;
                    end
                    else begin
                        cache_data[index][1][(32*offset)+:32] <= write_data;
                        cache_data[index][1][`dirty] <= 1'b1;
                        cache_data[index][1][`lru] <= 1'b1;
                        cache_data[index][0][`lru] <= 1'b0;
                        cache_ready <= 1'b1;
                        state <= `idle;
                    end
                end
                else begin
                    if(lru1)begin
                        if(dirty2)begin
                            mem_rw  <=  1'b1;
                            write_back  <=  1'b1;
                            mem_write_data  <=  cache_data[index][1][127:0];
                            state <= `write_back;
                        end
```

```
                        else begin
                            state <= `allocate;
                        end
                    end
                    else begin
                        if(dirty1)begin
                            mem_rw  <=  1'b1;
                            write_back  <=  1'b0;
                            mem_write_data  <=  cache_data[index][0][127:0];
                            state <= `write_back;
                        end
                        else begin
                            state <= `allocate;
                        end
                    end
                end
            end
            `allocate:begin
                if(mem_ready)begin
                    if(lru1)begin
                        cache_data[index][1][127:0] <= mem_data;
                        cache_data[index][1][151:128] <= tag;
                        cache_data[index][1][`valid] <= 1'b1;
                        cache_data[index][1][`dirty] <= 1'b0;
                        state <= `compare_tag;
                    end
                    else begin
                        cache_data[index][0][127:0] <= mem_data;
                        cache_data[index][0][151:128] <= tag;
                        cache_data[index][0][`valid] <= 1'b1;
                        cache_data[index][0][`dirty] <= 1'b0;
                        state <= `compare_tag;
                    end
                end
                else begin
                    state <= `allocate;
                end
            end
            `write_back:begin
                if(mem_ready)begin
                    cache_data[index][write_back][`dirty] <= 1'b0;
                    state <= `allocate;
                end
                else begin
                    state <= `write_back;
                end
            end
        endcase
    end
end

endmodule
```

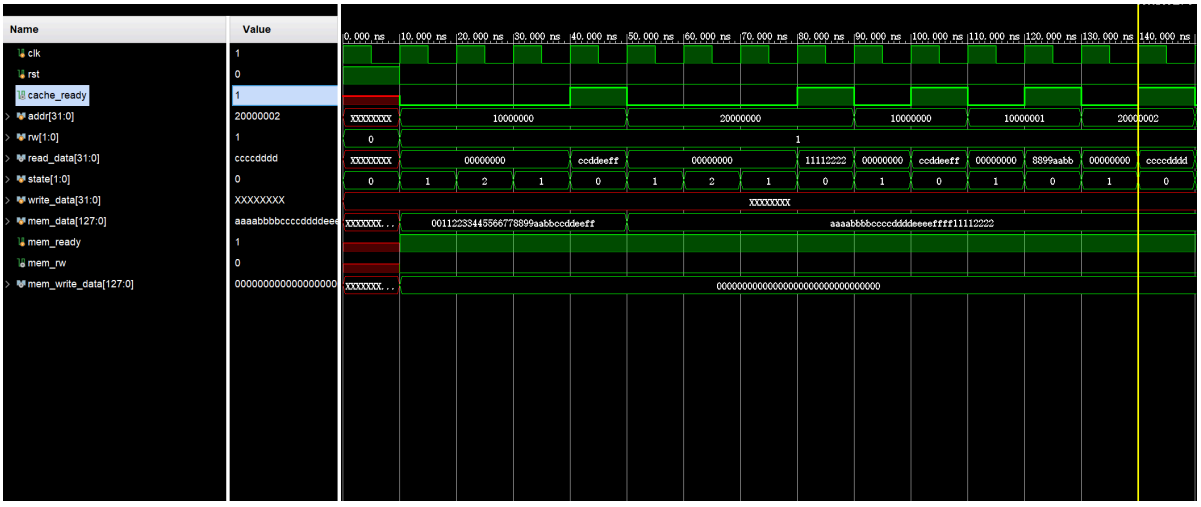
二、实验结果与分析

仿真代码

```
`timescale 1ns / 1ps
module cache_tb();
    reg clk;
    reg rst;
    reg [31:0]addr;
    reg [31:0]write_data;
    reg [127:0]mem_data;
    reg mem_ready;
    reg [1:0]rw;

    wire cache_ready;
    wire mem_rw;
    wire [127:0]mem_write_data;
    wire [31:0]read_data;

    cache m0(.clk(clk), .rst(rst), .addr(addr), .write_data(write_data),
.mem_data(mem_data), .mem_ready(mem_ready), .rw(rw), .cache_ready(cache_ready),
.mem_rw(mem_rw), .mem_write_data(mem_write_data), .read_data(read_data));

    initial begin
        clk=1;
        rst=1;
        rw=2'b0;
        #10;//read miss
        rst=0;
        rw=2'b1;
        addr=32'h10000000;
        mem_data=128'h00112233445566778899aabbccddeeff;
        mem_ready=1;
        #40;//read miss
        addr=32'h20000000;
        mem_data=128'haaaabbbbccccddddeeeeffff11112222;
        #40;//read hit
        addr=32'h10000000;
        #20;//read hit
        addr=32'h10000001;
        #20;//read hit
        addr=32'h20000002;
        #20;//read hit
        addr=32'h20000003;
        #20;//write hit
        rw=2'b10;
        addr=32'h10000001;
        write_data=32'hdeadbeef;
        #20;//write hit
        addr=32'h20000003;
        write_data=32'hcafebabe;
        #20;//read hit
        rw=2'b1;
        addr=32'h10000001;
```

```
        #20;//read hit
        addr=32'h20000003;
        #20;//write miss
        rw=2'b10;
        addr=32'h30000000;
        write_data=32'h11112222;
        mem_data=128'h33334444555566667777888889999aaaa;
        #50;//read hit
        rw=2'b1;
        addr=32'h30000000;
        #20;//read hit
        addr=32'h30000001;
        #20;//read hit
        addr=32'h30000002;
        #20;//read hit
        addr=32'h30000003;
        #20;
    end

    always #5 clk = ~clk;

endmodule
```

仿真波形分析

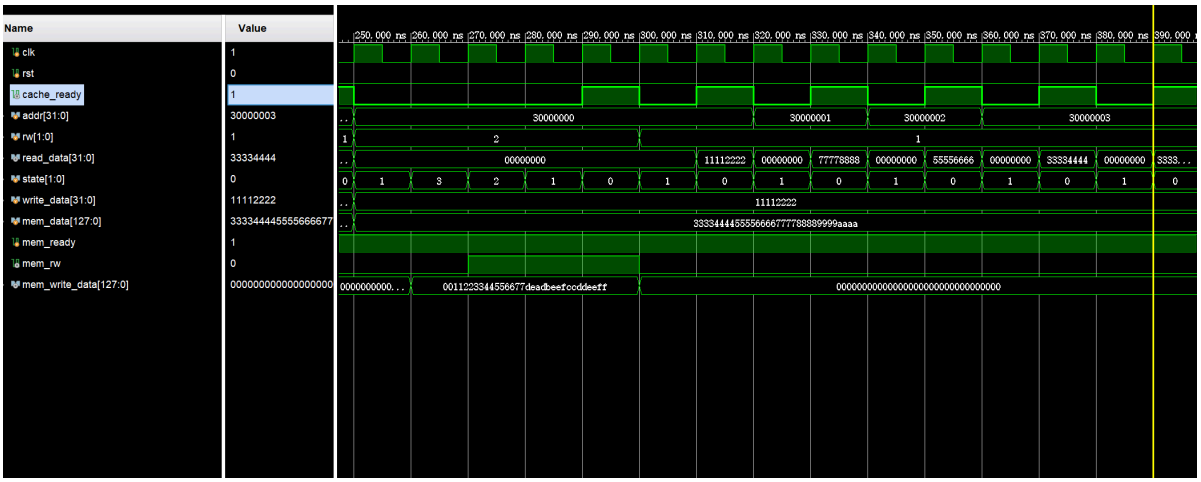


刚开始对地址32'h10000000进行读操作,但是此时cache中没有数据,要先从内存中读取数据到cache中,接着对地址32'h20000000进行读操作,此时还是read miss,也要从内存中读取数据到cache中,可以看到cache_ready为1时表示完成读写操作,读取到的正是从内存中读取的数据,状态变化为idle->compare_tag->allocate->compare_tag->idle。

接下来对地址32'h10000000,10000001,20000002,20000003进行读操作,由于之前已经从内存中读取了这些地址的数据到cache中,所以此时是read hit,状态变化为idle->compare_tag->idle,读取的数据与之前内存中读取的数据相符。

接着对地址32'h10000001,20000003进行写操作,此时是write hit,所以状态变化为idle->compare_tag->idle。

接下来再对地址32'h10000001,20000003进行读操作,来验证之前的写操作是否成功,可以看到读取到的值与之前写入的值相同,说明之前写操作成功,此时也是read hit,所以状态变化也是idle->compare_tag->idle。



对地址32'h30000000进行写操作,此时是write miss,但对应cache中的组中的两个块都有数据,根据二者的lru位去替换其中一个块,此时二者dirty位都是1,要先将数据写回内存中,之后再去内存中读取数据到块中,所以状态变化为idle->compare_tag->write_back->allocate->compare_tag->idle。

接下来对地址32'h30000000,30000001,30000002,30000003进行读操作,可以看到地址32'30000001,30000002,30000003中读取的数据与内存中读取的数据相同,而32'h30000000地址中读取的数据与之前写入该地址中的数据相同,此时是read hit,状态变化为idle->compare_tag->idle。

三、讨论、心得

心得

这学期计组实验的学习到这里基本就告一段落了,这学期还是学到很多新东西的,感觉cpu之类的也非常有意思,助教和老师人也很好,希望以后也能遇到这样的好老师和助教,抓紧时间复习期末考去了。