浙江大学

本科实验报告

课程名称: 计算机体系结构

姓　　名: 王俊

学　　院: 海洋学院

专　　业: 海洋工程与技术

学　　号: 3170100186

指导教师: 翁恺

2020 年　1　月　3　日

Lab7—Memory Hierarchy

课程名称： 计算机体系结构 实验类型： 综合

实验项目名称： 设计 cache

学生姓名： 王俊 专业： 海洋工程与技术 学号： 3170100186

同组学生姓名： None 指导老师： 翁恺

实验地点： 曹光彪西-301 实验日期： 2020 年 1 月 3 日

一、实验目的和要求

Experiment Purpose:

Part1:

•Understanding the principle of Memory Hierarchy

•Master the design methods of pipelined CPU accessing Mem. in multiple cycle.

•master methods of program verification of Pipelined CPU accessing Mem. in multiple cycle.

Part2:

•Understand Cache Line.

•Understand the principle of Cache Management Unit and State Machine of CMU.

•Master the design methods of CMU.

•Master the design methods of Cache Line.

•master verification methods of Cache Line.

•Implement CPU with cache and memory by two steps

–Implement a CPU accessing memory in multiple cycle

–Implement a CPU with cache – memory two level hierarchy

Experiment Apparatus:

- Computer (Intel Core i5 or higher, 4GB RAM or higher) system
- Sword-V4 development board
- Xilinx ISE 14.4 and above development tools

Experimental Materials:

No

二、实验内容和原理

2.1.Experimental task:

Part1:

•Design of Pipelined CPU accessing Mem. in multiple cycle

–Redesign Inst. ROM & Data RAM

–Modify CPU Controller

–Modify datapath

•Verify the Pipelined CPU with program and observe the execution of program

Part2:

•Design of Cache Line and CMU.

•Verify the Cache Line and CMU.

•Observe the Waveform of Simulation.

2.2.Basic principle

2.2.1.Inst. ROM's Job

•Initial State: S_IDLE

•Working State: S_READ

•Read for 8 cycles, then output the value, set the signal ACK

•STALL = CS & ~ACK

2.2.2.Data RAM's Job

•Initial State: S_IDLE

•Working State: S_READ/S_WRITE (according to signal we)

•Read/Write for 8 cycles, then output the value/write the value, set the signal ACK

•STALL = CS & ~ACK

2.2.3.When ROM&RAM STALL

•Inst. ROM Stalls, Stall should be in ID stage otherwise jump indicator would be lost

- •if_en=0, id_en=0, rst_exe=1

•DATA RAM stalls

- •…, rst_wb=1

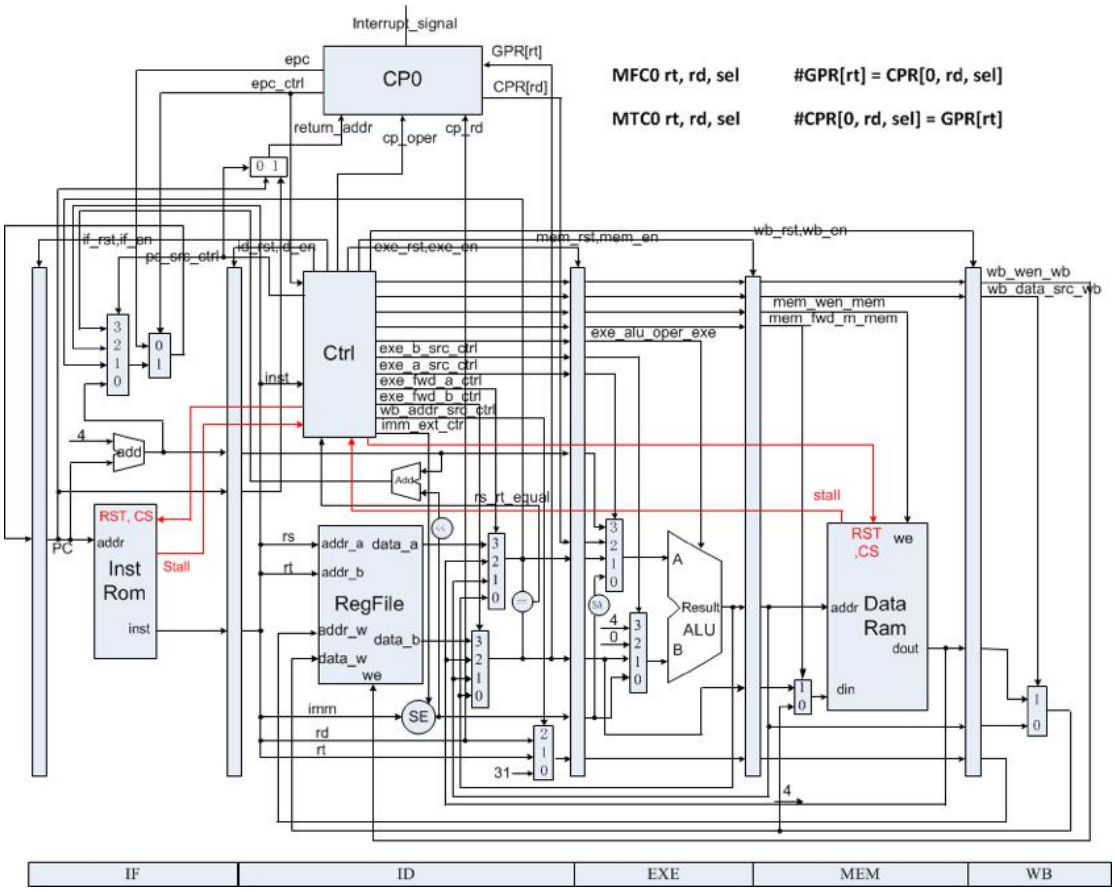
•Add TCR in CP0

- •TCR - Tick Counter Register
- •TCR ++ when System Clock

Cycle goes on:

```
// CP0 registers
localparam
        //CP0_SR   = 0,
        //CP0_EAR  = 1,
        CP0_EPCR  = 2,
        CP0_EHBR  = 3,
        //CP0_IER  = 4,
        //CP0_ICR  = 5,
        //CP0_PDBR = 6,
        //CP0_TIR  = 7,
        //CP0_WDR  = 8,
        CP0_TCR  = 9;
```

2.2.4.Datapath of CPU accessing Mem. in multiple cycle



2.2.5.Cache design

Cache Line:

V	D	tag	Data

Cache Mode:

- Direct Map
- Write Back

•Write Allocate

Address:

```
parameter
        TAG_BITS = 22,
        LINE_WORDS = 4,
        LINE_WORDS_WIDTH = 2;
    localparam
        ADDR_BITS = 32,
        WORD_BYTES = 4;
    localparam
        WORD_BITS = 8 * WORD_BYTES,    // 32
        WORD_BYTES_WIDTH = 2,
        LINE_INDEX_WIDTH = ADDR_BITS - TAG_BITS -
LINE_WORDS_WIDTH - WORD_BYTES_WIDTH,    // 6
        LINE_NUM = 1 << LINE_INDEX_WIDTH;    // 64
```

Cache Line Memory

•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];

Read and Write Cache:

dout<= inner_data[addr[ADDR_BITS-TAG_BITS-1:WORD_BYTES_WIDTH]];

if (store || edit)

inner_data[addr[ADDR_BITS-TAG_BITS-1:WORD_BYTES_WIDTH]] <= din;

Dirty, Valid, tag of Cache:

•invalid

inner_valid[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=0

inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <= 0

•load

inner_valid[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=1

inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=0

inner_tag[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=addr[ADDR_BITS-1:ADDR_BITS-TAG_BITS]

•edit

inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=1

inner_tag[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=addr[ADDR_BITS-1:ADDR_BITS-TAG_BITS]

valid <=

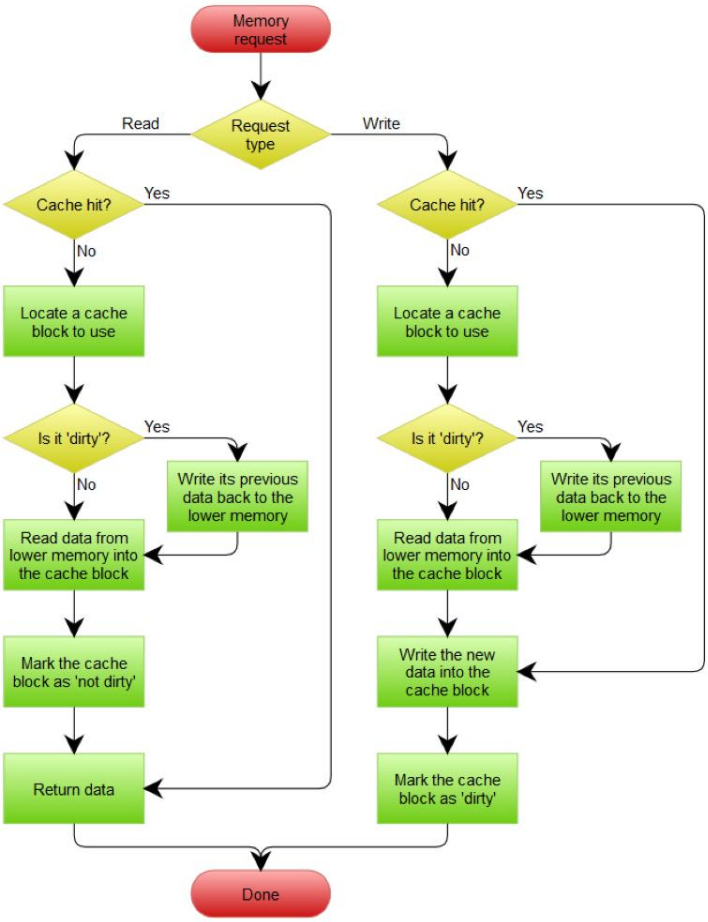
inner_valid[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]]

dirty <= inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]]

tag <= inner_tag[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]]

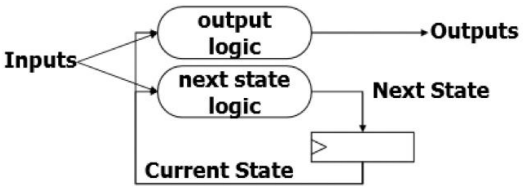
hit = valid & (tag == addr[ADDR_BITS-1:ADDR_BITS-TAG_BITS]);

- **Cache Operation Flow:**



- **Cache Management State Machine**

- **Next State Logic**
- State assignment
- Output



三、实验过程

3.1.cache design:

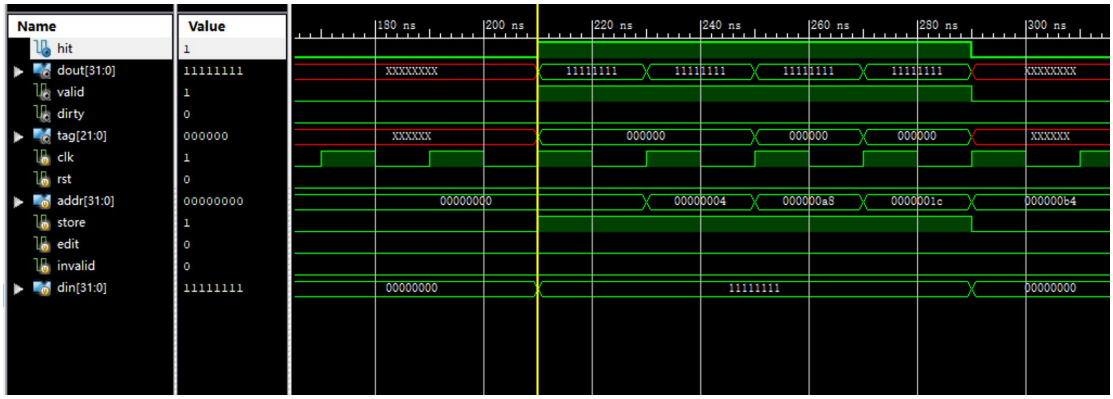
根据题目的输入输出：

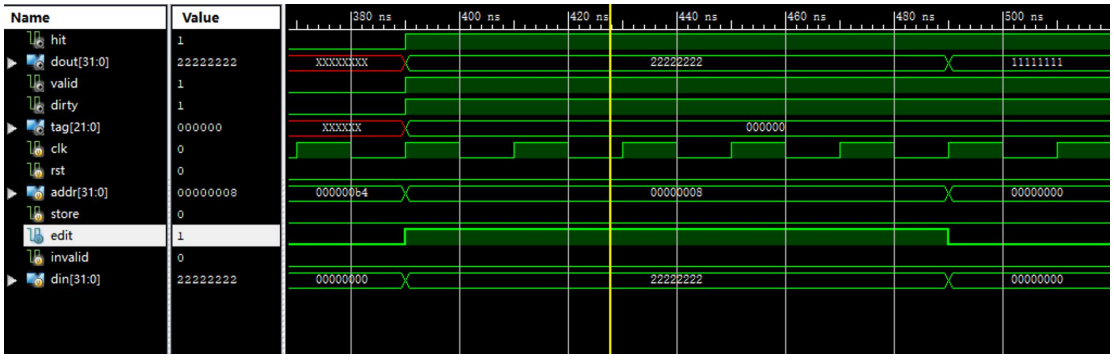
```
initial begin
    inner_valid = 0;
    inner_dirty = 0;
 end


assign hit = valid & (tag == addr[ADDR_BITS-1:ADDR_BITS-TAG_BITS]);
assign dout = inner_data[addr[ADDR_BITS-TAG_BITS-1:WORD_BYTES_WIDTH]];
assign valid = inner_valid[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]];
assign dirty = inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]];
assign tag = inner_tag[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]];

always @(posedge clk) begin

    if (rst) begin
        inner_valid <= 0;
    end else begin
        if (store || edit) begin
            inner_data[addr[ADDR_BITS-TAG_BITS-1:WORD_BYTES_WIDTH]] <= din;
        end
        if (invalid) begin
            inner_valid[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <= 0;
            inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <= 0;
        end else if(store) begin
            inner_valid[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <= 1;
            inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <= 0;
            inner_tag[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <= addr[ADDR_BITS-1:ADDR_BITS-TAG_BITS];
        end else if (edit) begin
            inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <= 1;
            inner_tag[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <= addr[ADDR_BITS-1:ADDR_BITS-TAG_BITS];
        end
    end

end
```

SIMULATION:

```
initial begin
    // Initialize Inputs
    clk = 0;
    rst = 0;
    addr = 0;
    store = 0;
    edit = 0;
    invalid = 0;
    din = 0;

    // Add stimulus here
    #210;
    store = 1;
    din = 32'h11111111;
    addr = 32'h00000000;
    #20   addr = 32'h00000004;
    #20   addr = 32'h000000A8;
    #20   addr = 32'h0000001C;
    #20   store = 0;  addr = 32'h000000B4;    din = 0;
    #100  edit = 1;   din = 32'h22222222;     addr = 32'h00000008;
    #100  edit = 0;   din = 0;                addr = 0;

end

initial forever #10 clk = ~clk;
```

Result:



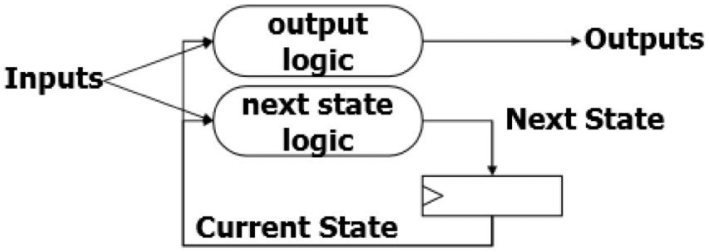


3.2.CMU-cache management unit:

- **State-machine:**

根据状态图以及 ppt 的填空得到：

- **Next State Logic**
- State assignment
- Output




```
// next state logic
always @(*) begin
    next_word_count = 0;
    next_state = S_IDLE;
    if (rst) begin
        next_word_count = 0;
        next_state = S_IDLE;
    end
    else case (state)
        S_IDLE: begin
            if (en_r || en_w) begin
                if (cache_hit)
                    next_state = S_IDLE;
                else if (cache_valid && cache_dirty)
                    next_state = S_BACK;
                else
                    next_state = S_FILL;
            end
        end
        S_BACK: begin
            if (mem_ack_i)
                next_word_count = word_count + 1'h1;
            else
                next_word_count = word_count;
            if (mem_ack_i && word_count == {LINE_WORDS_WIDTH{1'b1}})
                next_state = S_BACK_WAIT;
            else
                next_state = S_BACK;
        end
        S_BACK_WAIT: begin
            next_word_count = 0;
            next_state = S_FILL;
        end
        S_FILL: begin
            if (mem_ack_i)
                next_word_count = word_count + 1'h1;
            else
                next_word_count = word_count;
            if (mem_ack_i && word_count == {LINE_WORDS_WIDTH{1'b1}})
                next_state = S_FILL_WAIT;
            else
                next_state = S_FILL;
        end
        S_FILL_WAIT: begin
            next_word_count = 0;
            next_state = S_IDLE;
        end
    endcase
end
```

```
always @(posedge clk) begin
        if (rst) begin
                state <= 0;
                word_count <= 0;
        end
        else begin
                state <= next_state;
                word_count <= next_word_count;
        end
end
```

- **Cache-control:**

根据 ppt 里的 output1：

Cache—control

```
    // cache control
    reg [LINE_WORDS_WIDTH-1:0] word_count_buf;
    always @(posedge clk) begin
        if (rst)
            word_count_buf <= 0;
        else
            word_count_buf <= word_count;
    end

    always @(*) begin
        cache_store = 0;
        cache_edit = 0;
        cache_addr = 0;
        cache_din = 0;
        case (next_state)
            S_IDLE: begin
                cache_addr = addr_rw;
                cache_edit = en_w;
                cache_din = data_w;
            end
            S_BACK, S_BACK_WAIT: begin
                cache_addr = {addr_rw[31:LINE_WORDS_WIDTH+2], next_word_count, 2'b00};
            end
            S_FILL, S_FILL_WAIT: begin
                cache_addr = {addr_rw[31:LINE_WORDS_WIDTH+2], word_count_buf, 2'b00};
                cache_din = mem_data_i;
                cache_store = mem_ack_i;
            end
        endcase
    end
```

- **Memory-control:**

根据 ppt 里的 output2：

```
// memory control
    always @(posedge clk) begin
        mem_cs_o <= 0;
        if (rst) begin
            mem_we_o <= 0;
            mem_addr_o <= 0;
        end
        else case (next_state)
            S_IDLE, S_BACK_WAIT, S_FILL_WAIT: begin
                mem_cs_o <= 0;
                mem_we_o <= 0;
                mem_addr_o <= 0;
            end
            S_BACK: begin
                mem_cs_o <= 1;
                mem_we_o <= 1;
                mem_addr_o <= {cache_tag, addr_rw[31-TAG_BITS:LINE_WORDS_WIDTH+2], next_word_count, 2'b00};
            end
            S_FILL: begin
                mem_cs_o <= 1;
                mem_we_o <= 0;
                mem_addr_o <= {addr_rw[31:LINE_WORDS_WIDTH+2], next_word_count, 2'b00};
            end
        endcase
    end
```

3.3.Simulate the cache:

Simulation (1)

```
module inst (
            input wire clk,
            input wire rst,
            input wire [3:0] index,  // instruction index
            output wire valid,  // stop running if valid is 0
            output wire write,  // write enable signal for cache
            output wire [31:0] addr  // address for cache
            );
            reg [33:0] data [0:7];
            initial begin  // clock cycles are only for reference
                        data[0] = 34'h200000004;  // read miss              1+17
                        data[1] = 34'h300000018;  // write miss             1+17
                        data[2] = 34'h200000008;  // read hit               1
                        data[3] = 34'h300000014;  // write hit              1
                        data[4] = 34'h210000004;  // read & clean replace    1+17
                        data[5] = 34'h310000018;  // write & dirty replace   1+17*2
                        data[6] = 34'h310000008;  // write hit              1
                        data[7] = 34'h0;          // end            total: 92
            end
            assign
                        valid = data[index][33],
                        write = data[index][32],
                        addr = data[index][31:0];
endmodule
```

Simulation (2)

ZheJiang University

```
`timescale 1ns / 1ps

module top (
    input wire clk,
    input wire rst,
    output reg [7:0] clk_count = 0,
    output reg [7:0] inst_count = 0,
    output reg [7:0] hit_count = 0
    );

    // instruction
    reg [3:0] index = 0;
    wire valid;
    wire write;
    wire [31:0] addr;
    wire stall;
    inst INST (
        .clk(clk),
        .rst(rst),
        .index(index),
        .valid(valid),
        .write(write),
        .addr(addr)
    );
```

```
always @(posedge clk) begin
    if (rst)
        index <= 0;
    else if (valid && ~stall)
        index <= index + 1'h1;
end
// ram
wire mem_cs;
wire mem_we;
wire [31:0] mem_addr;
wire [31:0] mem_din;
wire [31:0] mem_dout;
wire mem_ack;
data_ram #(
    .ADDR_WIDTH(5),
    .CLK_DELAY(3)
    ) RAM (
    .clk(clk),
    .rst(rst),
    .addr({26'b0, mem_addr[5:0]}),
    .cs(mem_cs),
    .we(mem_we),
    .din(mem_din),
    .dout(mem_dout),
    .stall(),
    .ack(mem_ack)
    );
```

Simulation (3)

ZheJiang University

```
// cache
cmu CMU (
    .clk(clk),
    .rst(rst),
    .addr_rw(addr),
    .en_r(~write),
    .data_r(),
    .en_w(write),
    .data_w({16'h5678, clk_count, inst_count}),
    .stall(stall),
    .mem_cs_o(mem_cs),
    .mem_we_o(mem_we),
    .mem_addr_o(mem_addr),
    .mem_data_i(mem_dout),
    .mem_data_o(mem_din),
    .mem_ack_i(mem_ack)
);
```

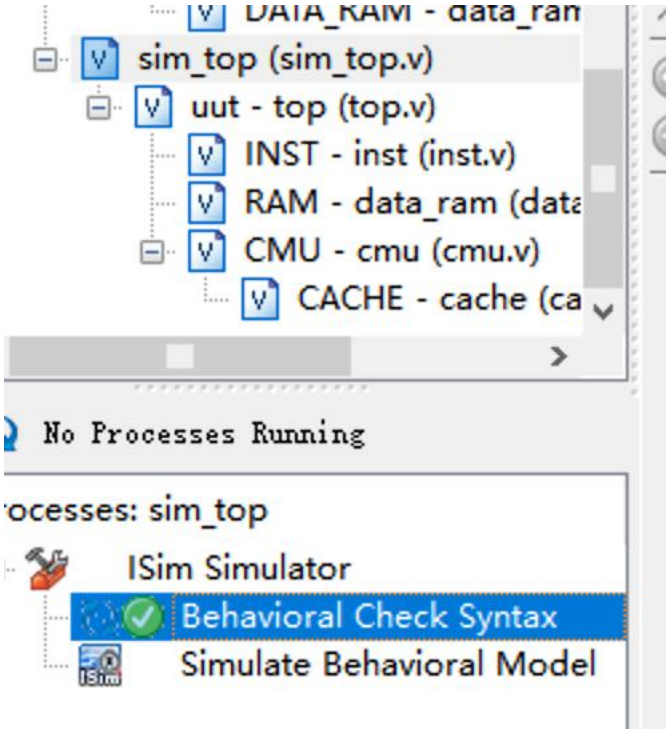
```
// counter
reg stall_prev;

always @(posedge clk) begin
    if (rst)
        stall_prev <= 0;
    else
        stall_prev <= stall;
end

always @(posedge clk) begin
    if (rst) begin
        clk_count <= 0;   // 时钟计数
        inst_count <= 0;  // 指令计数
        hit_count <= 0;   // 命中计数
    end
    else if (valid) begin
        clk_count <= clk_count + 1'h1;
        inst_count <= index + 1'h1;
        if (~stall_prev && ~stall)
            hit_count <= hit_count + 1'h1;
    end
end

endmodule
```

Simulation (4)

```
module sim_top;

    // Inputs
    reg clk;
    reg rst;

    // Outputs
    wire [7:0] clk_count;
    wire [7:0] inst_count;
    wire [7:0] hit_count;

    // Instantiate the Unit Under Test (UUT)
    top uut (
        .clk(clk),
        .rst(rst),
        .clk_count(clk_count),
        .inst_count(inst_count),
        .hit_count(hit_count)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 1;

        // Wait 100 ns for global reset to finish
        #95 rst = 0;

        // Add stimulus here


    end

    initial forever #10 clk = ~clk;

endmodule
```

四、实验结果分析

编译 simulation：



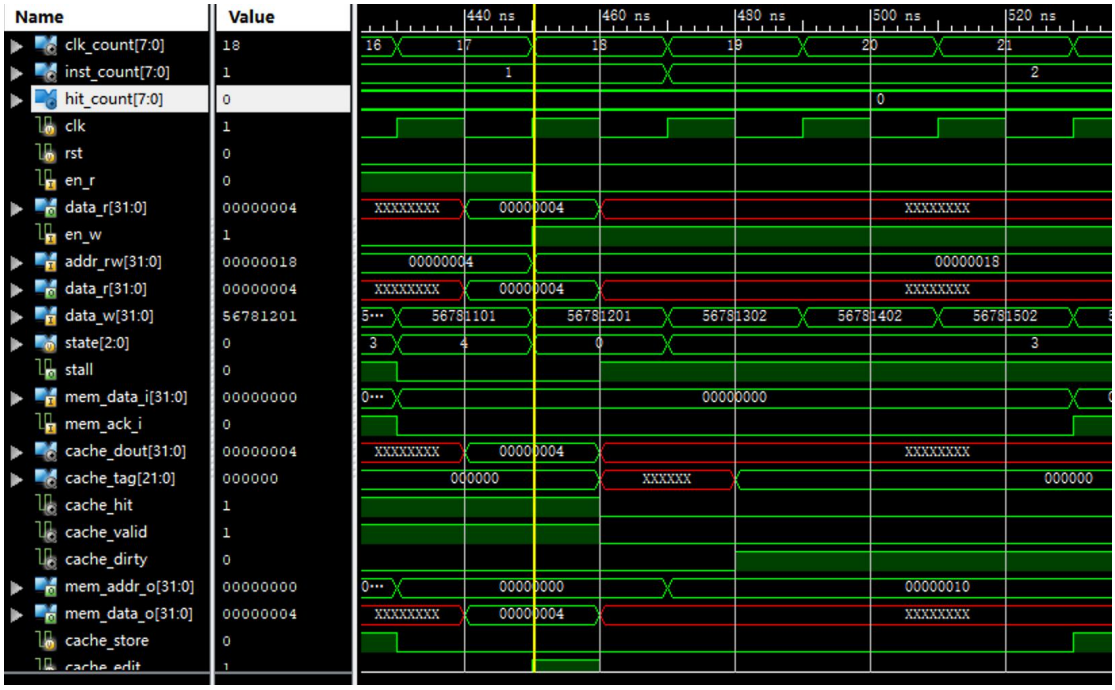
Sim_top:

```
initial begin
   // Initialize Inputs
   clk = 0;
   rst = 1;

   // Wait 100 ns for global reset to finish
   #95 rst = 0;

   // Add stimulus here


end

initial forever #10 clk = ~clk;



always @(posedge clk) begin
   if (rst)
      stall_prev <= 0;
   else
      stall_prev <= stall;
end

always @(posedge clk) begin
   if (rst) begin
      clk_count <= 0;
      inst_count <= 0;
      hit_count <= 0;
   end
   else if (valid) begin
      clk_count <= clk_count + 1'h1;
      inst_count <= index + 1'h1;
      if (~stall_prev && ~stall)
         hit_count <= hit_count + 1'h1;
   end
end
```

Sim_inst:

```
initial begin  // clock cycles are only for reference
   data[0] = 34'h200000004;  // read miss              1+17
   data[1] = 34'h300000018;  // write miss             1+17
   data[2] = 34'h200000008;  // read hit               1
   data[3] = 34'h300000014;  // write hit              1
   data[4] = 34'h210000004;  // read clean replace     1+17
   data[5] = 34'h310000018;  // write dirty replace    1+17*2
   data[6] = 34'h310000008;  // write hit              1
   data[7] = 34'h0;
end
```

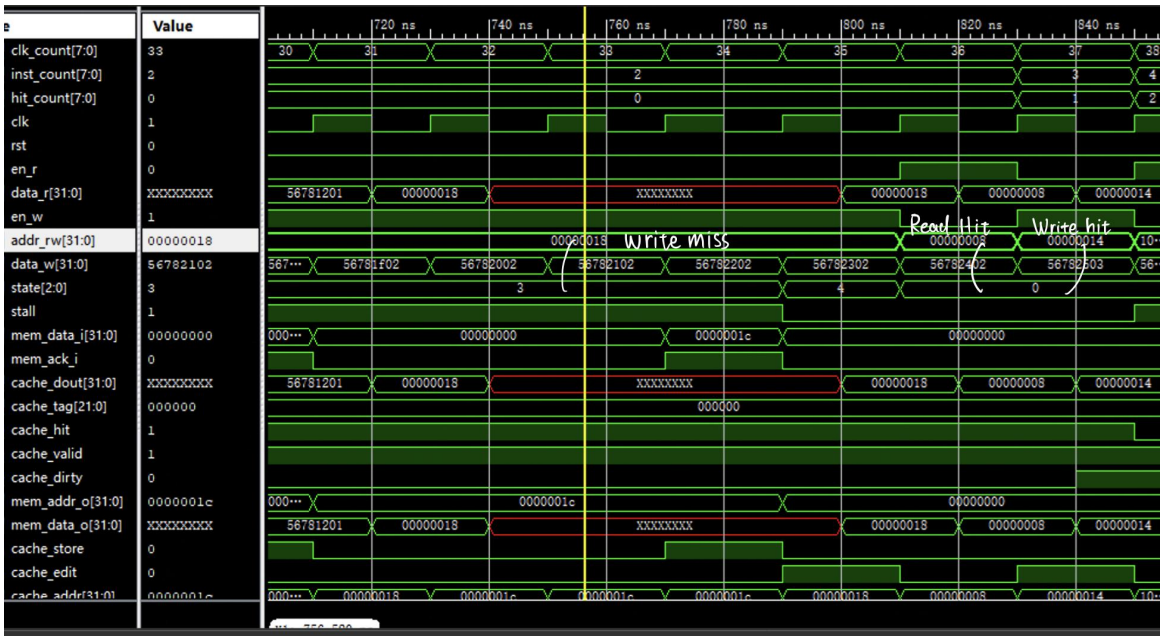
•Read miss at 0x00000004



•Write miss at 0x00000018

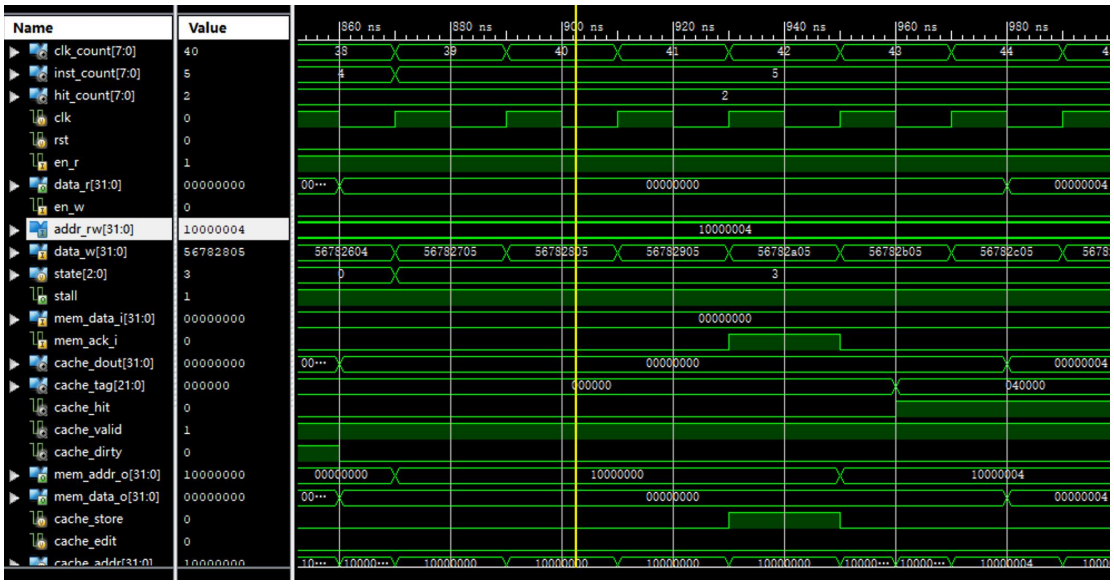
•Read hit at 0x00000008

•Write hit at 0x00000014



•Read & clean replace at 0x10000004

State : 0->3->4->0

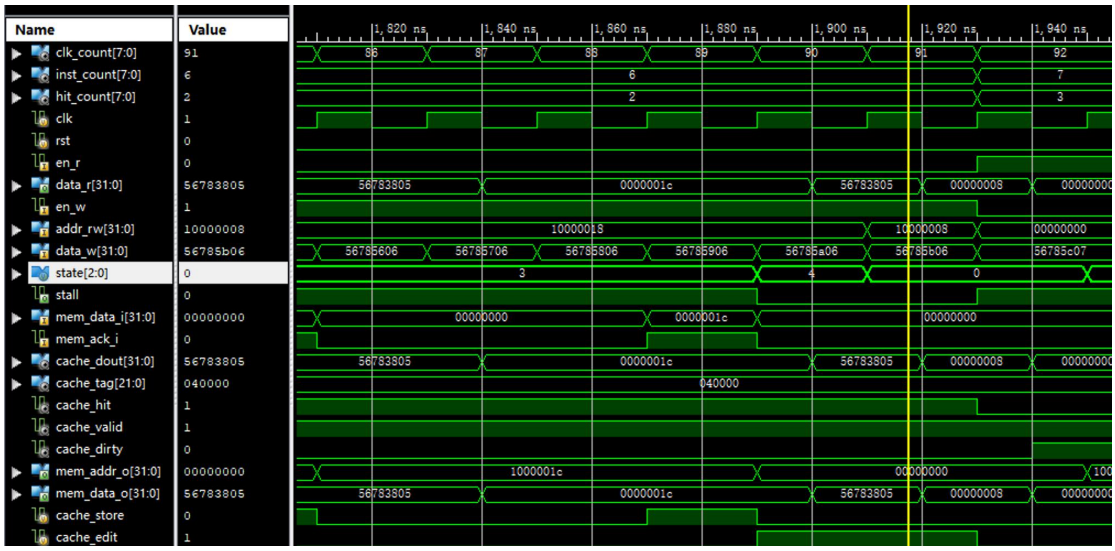


•Write & dirty replace at 0x10000018

State : 0->1->2->3->4->0

•Write hit at 0x10000008

State : 0->0



Performance Analysis:

•Result

–CLK_COUNT = 92

–INST_COUNT = 7

–HIT_COUNT = 3