

浙江大学

本科实验报告

课程名称: 计算机体系结构

姓 名: 姜雨童

学 院: 计算机科学与技术学院

专 业: 计算机科学与技术

邮 箱: 3220103450@zju.edu.cn

QQ 号: 1369218489

电 话: 18867766468

指导教师: 王小航

报告日期: 2024 年 11 月 17 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: Lab04 Pipelined CPU with Cache

学生姓名: 姜雨童 学号: 33220103450 同组学生姓名: /

实验地点: 玉泉曹西 301 实验日期: 2024 年 11 月 17 日

一、目标与原理、

1-1 实验目的

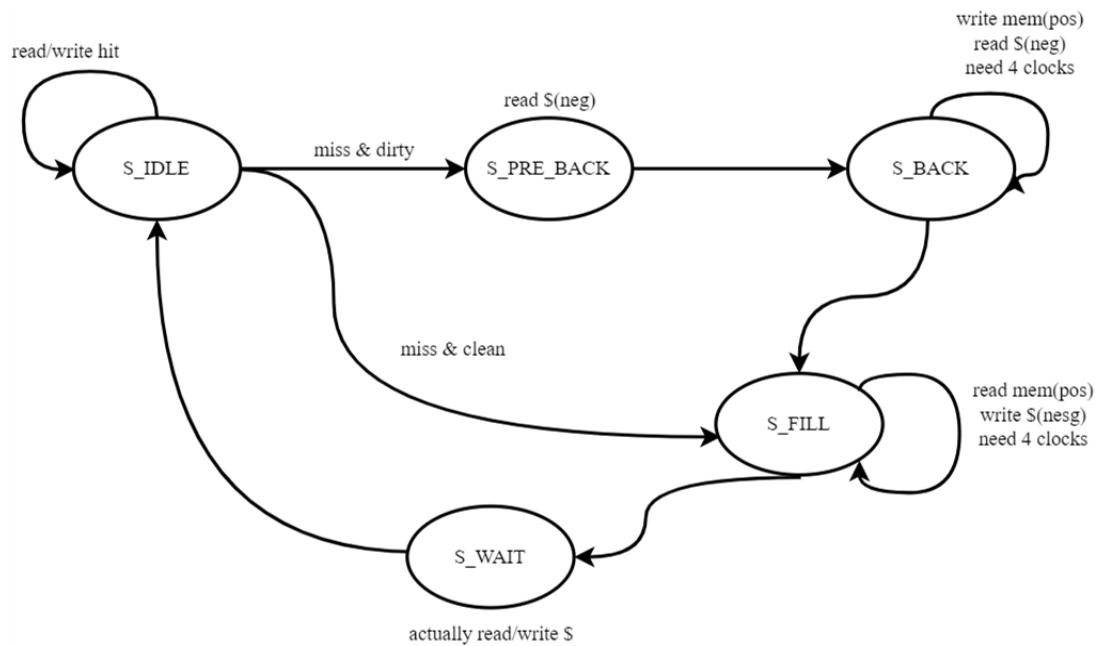
- Understand the principle of Cache Management Unit (CMU) and State Machine of CMU.
- Master the design methods of CMU and Integrate it to the CPU.
- Master verification methods of CMU and compare the performance of CPU when it has cache or not.

1-2 实验原理

本实验延续了实验三（实现 cache），需要实现 cache controller，即 cache 的管理单元。它的实现就是一个拥有五个状态的有限状态机（见下图）：

1. S_IDLE:初始状态，如果 hit，则继续维持在此状态；
2. S_PRE_BSCK:和 S_BACK 一起构成 miss 时需要写回内存（即 dirty）的状态转移；
3. S_BACK:miss 时需要写回内存时的状态转移，之后去到状态 S_FILL；
4. S_FILL:miss 且不需写回内存/已经写回内存后的状态，从内存中读数据到 cache；
5. S_WAIT:miss 处理后重新执行指令。

在实现 cache controller 的状态机后，还需要将其接入 pipelined CPU。因此当 cache 不能较快地返回数据时，需要对 pipeline 进行 stall 操作，因此在 cache controller 中还需要对 stall 进行判断。



二、操作方法与实验步骤

2-1 状态机

本实验的整体框架和代码均已给出，只需要填几个空。而根据上述对于状态机的分析及状态转移图，可以轻松写出状态转移的这几个空，故这里不做过多分析。另外有几个空是针对计数器的，由于项目中不论是从 memory 读入 cache 还是从 cache 写回 memory 都是每次一个 word，因此当收到 ack 信号（成功读入/写回时），计数器加一，否则维持当前值。

附源代码：

```

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_PRE_BACK;
      else
        next_state = S_FILL;
    end
    next_word_count = 2'b00;
  end

  S_PRE_BACK: begin
    next_state = S_BACK;
    next_word_count = 2'b00;
  end
end

```

```

        S_BACK: begin
            if (mem_ack_i && word_count == {ELEMENT_WORDS_WIDTH{1'b1}}) //
2'b11 in default case
                next_state = S_FILL;
            else
                next_state = S_BACK;

            if (mem_ack_i)
                next_word_count = word_count + 2'b01;
            else
                next_word_count = word_count;
        end

        S_FILL: begin
            if (mem_ack_i && word_count == {ELEMENT_WORDS_WIDTH{1'b1}})
                next_state = S_WAIT;
            else
                next_state = S_FILL;

            if (mem_ack_i)
                next_word_count = word_count + 2'b01;
            else
                next_word_count = word_count;
        end

        S_WAIT: begin
            next_state = S_IDLE;
            next_word_count = 2'b00;
        end
    end
endcase

```

2-2 判断 stall

当需要状态转移到 S_IDLE 以外的状态时，说明需要写回 memory/读入 cache，因此要对流水线进行 stall 操作，代码如下：

```
assign stall = (next_state != S_IDLE);
```

三、实验结果与分析

3-1 仿真

Simulation

inst.v

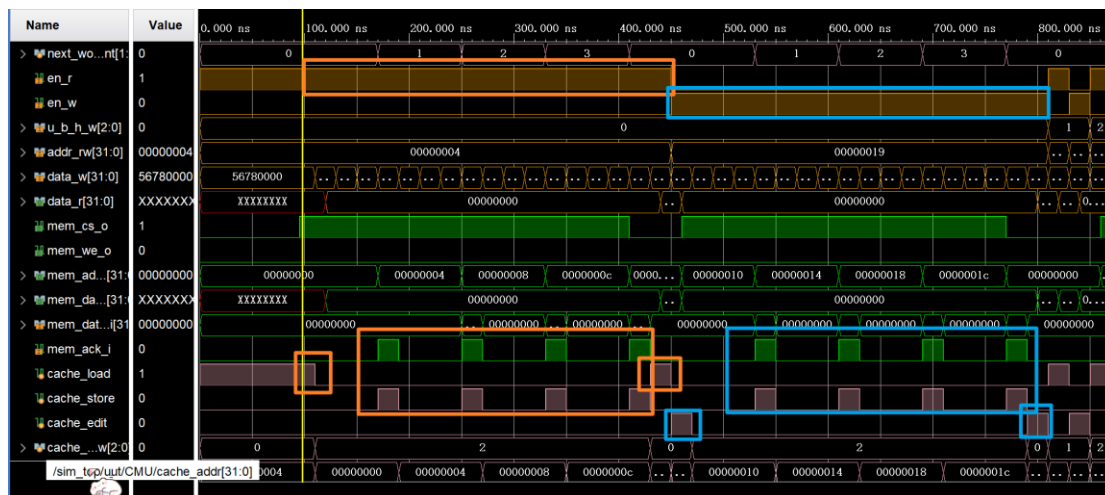
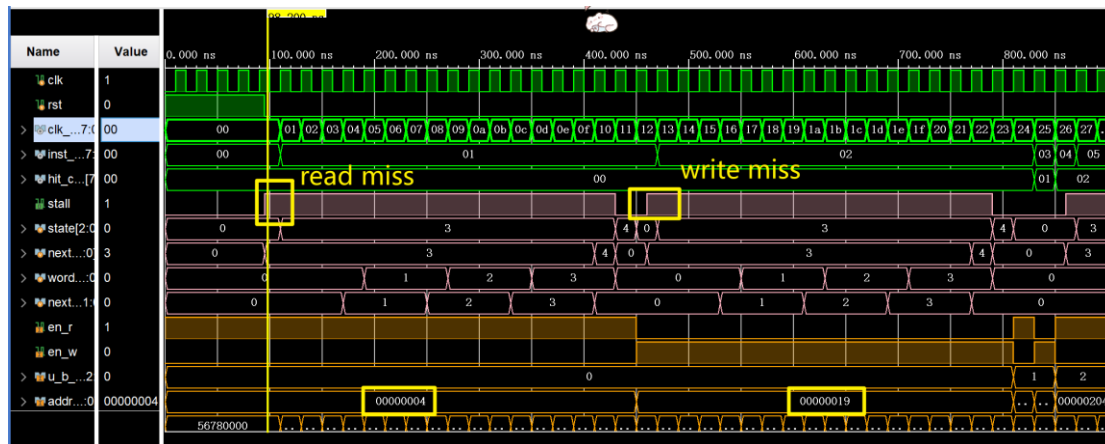
```

reg [39:0] data [0:9];
initial begin
    data[0] = 40'h0_2_00000004; // read miss          1+17
    data[1] = 40'h0_3_00000019; // write miss         1+17
    data[2] = 40'h1_2_00000008; // read hit           1
    data[3] = 40'h1_3_00000014; // write hit       1

    data[4] = 40'h2_2_00000204; // read miss          1+17
    data[5] = 40'h2_3_00000218; // write miss         1+17
    data[6] = 40'h0_3_00000208; // write hit           1
    data[7] = 40'h4_2_00000414; // read miss + dirty  1+17+17
    data[8] = 40'h1_3_00000404; // write miss + clean 1+17
    data[9] = 40'h0;           // end                total: 128
end
assign
    u_b_h_w = data[index][38:36],
    valid = data[index][33],
    write = data[index][32],
    addr = data[index][31:0];

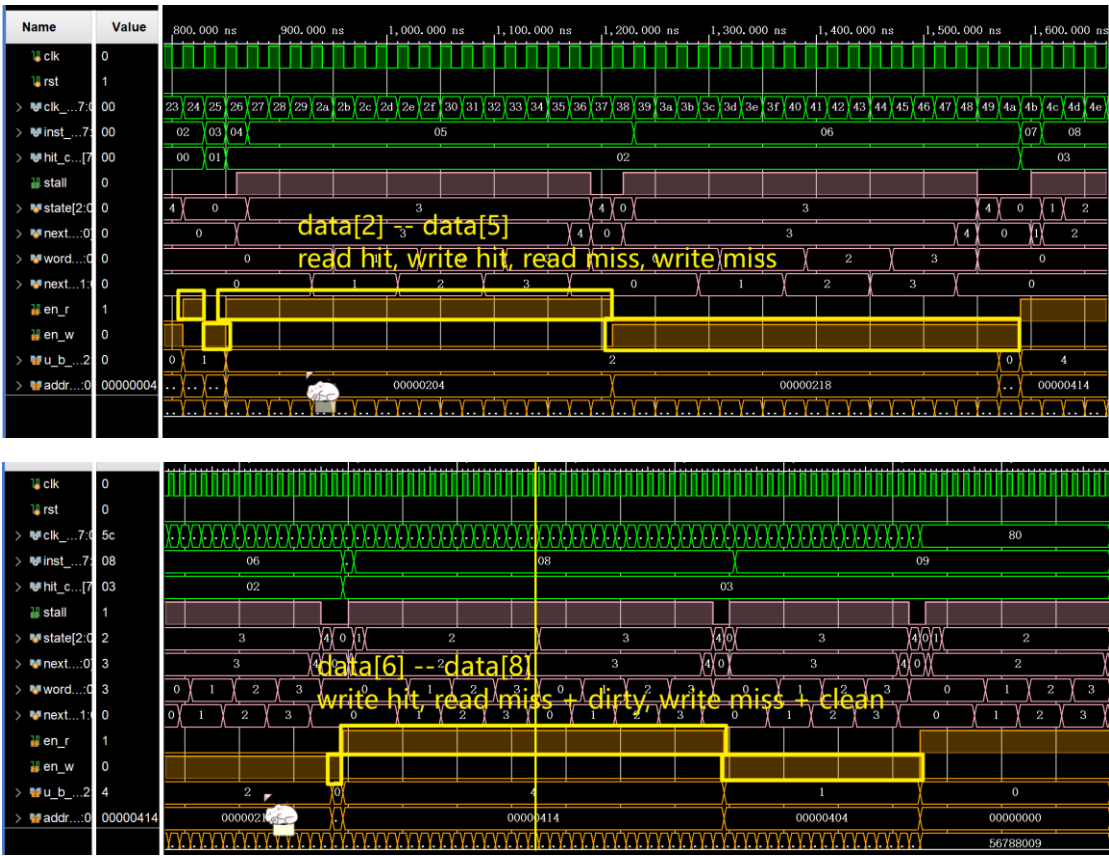
```

指令 0 和 1 分别是 read miss 和 write miss，如下两图所示，由于 miss 每条指令都需要 1+17（4*4+1，需要读/写 4 次 word，最后加一是状态转移后需要重新执行指令）个时钟周期的时间，中间有四次 ack 信号表示读取/写入 word 成功。（图 2 橙色框为 read miss，蓝色框为 write miss。）



其他指令的分析不在此赘述，仿真波形截图如下：

其中 read miss dirty 需要 1+17+17 的时间是因为 read miss 的 cache 为 dirty 时首先需要将其写回内存(4*4+1, 1 为状态转移后重新执行的时间)，然后从内存中读取需要的页面(4*4+1)。



3-2 上板

一共发生六次 miss，根据 32 位地址的划分，可以算出这些指令对应地址的 tag，index 和 offset:

No	Tag	Index	Offset
1	0	1	15
6	1	1	0
9	0	0	0
12	1	0	0
13	2	0	0
14	2	1	0

分析代码的实现可知，index 为奇数对应 set1，偶数对应 set0；tag 奇数对应 line1，偶数对应 line0，因此可以得到指令右侧标注的 set 和 line。

附指令如下：

ROM

NO.	Instruction	Addr.	Label	ASM	Comment
0	00000013	0	__start:	addi x0, x0, 0	
1	01c00083	4		lb x1, 0x01C(x0)	# F0F0F0F0 in 0x1C # FFFFFFF0 miss, read 0x010~0x01C to set 1 line 0
2	01c01103	8		lh x2, 0x01C(x0)	# FFFFFFF0 hit
3	01c02183	C		lw x3, 0x01C(x0)	# F0F0F0F0 hit
4	01c04203	10		lbu x4, 0x01C(x0)	# 000000F0 hit
5	01c05283	14		lhu x5, 0x01C(x0)	# 0000F0F0 hit
6	21002003	18		lw x0, 0x210(x0)	# miss, read 0x210~0x21C to cache set 1 line 1
7	abcde0b7	1C		lui x1 0xABCDE	
8	71c08093	20		addi x1, x1, 0x71C	# x1 = 0xABCDE71C
9	00100023	24		sb x1, 0x0(x0)	# miss, read 0x000~0x00C to cache set 0 line 0
10	00101223	28		sh x1, 0x4(x0)	# hit
11	00102423	2C		sw x1, 0x8(x0)	# hit

ROM

NO.	Instruction	Addr.	Label	ASM	Comment
12	20002303	30		lw x6, 0x200(x0)	# miss, read 0x200~0x20C to cache set 0 line 1
13	40002383	34		lw x7, 0x400(x0)	# miss, write 0x000~0x00C back to ram, then read 0x400~40C to cache set 0 line 0
14	41002403	38		lw x8, 0x410(x0)	# miss, no write back because of clean, read 0x410~41C to chache set 1 line 0
15	0ed06813	3c	loop:	ori x16, x0, 0xED	# end
16	ffdf06f	40		jal x0, loop	

首先指令 1 处，访问未访问过的 cache，是 miss read，需要从内存中读取：

指令 1 进入 mem 阶段，开始读 1/ (1+17)	<pre> PC 1F 00000010 INST 1F 01C04203 rs1Data 00000000 rs2Data 00000000 PC 1D 0000000C INST 1D 01C02183 rs1Addr 00000000 rs2Addr 0000001C PC 1E 0000000B INST 1E 01C01103 CPU-RdW 00000000 PCJump0 00000020 PC 1F 00000004 INST 1F 01C00083 B/PCE-S 00000000 B/C-Hzd 00000000 PC 1B 00000000 INST 1B 00000013 L/ABSe1 00010001 PCIFNxt 00000014 ALU 010 00000000 ALU-Out 0000001C CPUAddr 00000000 ALUCTr1 00000001 LJ 010 0000001C WB Data 00000000 CPU-Dat 00000000 WR-FID 00000001 wr 1F 0000001C WB Addr 00000000 CPU-Dno 00000000 RegW/DR 00010000 IDE 00 00000013 CODE-01 01C00083 CODE-02 01C01103 CODE-03 01C02183 IDE 04 01C04203 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000 wr 00 00000000 CODE-00 00000000 CODE-0A 00000000 CODE-0B 00000000 </pre>
读第一个 word 计数器第一拍 1/17	<pre> PC 1F 00000010 INST 1F 01C04203 rs1Data 00000000 rs2Data 00000000 PC 1D 0000000C INST 1D 01C02183 rs1Addr 00000000 rs2Addr 0000001C PC 1E 0000000B INST 1E 01C01103 CPU-RdW 00030001 PCJump0 00000020 PC 1F 00000004 INST 1F 01C00083 B/PCE-S 00000000 B/C-Hzd 00000000 PC 1B 00000000 INST 1B 00000013 L/ABSe1 00010001 PCIFNxt 00000014 ALU 010 00000000 ALU-Out 0000001C CPUAddr 00000000 ALUCTr1 00000001 LJ 010 0000001C WB Data 00000000 CPU-Dat 00000000 WR-FID 00000001 wr 1F 0000001C WB Addr 00000000 CPU-Dno 00000000 RegW/DR 00010000 IDE 00 00000013 CODE-01 01C00083 CODE-02 01C01103 CODE-03 01C02183 IDE 04 01C04203 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000 wr 00 00000000 CODE-00 00000000 CODE-0A 00000000 CODE-0B 00000000 </pre>
计数器第二拍 1/17	<pre> PC 1F 00000010 INST 1F 01C04203 rs1Data 00000000 rs2Data 00000000 PC 1D 0000000C INST 1D 01C02183 rs1Addr 00000000 rs2Addr 0000001C PC 1E 0000000B INST 1E 01C01103 CPU-RdW 00030002 PCJump0 00000020 PC 1F 00000004 INST 1F 01C00083 B/PCE-S 00000000 B/C-Hzd 00000000 PC 1B 00000000 INST 1B 00000013 L/ABSe1 00010001 PCIFNxt 00000014 ALU 010 00000000 ALU-Out 0000001C CPUAddr 00000000 ALUCTr1 00000001 LJ 010 0000001C WB Data 00000000 CPU-Dat 00000000 WR-FID 00000001 wr 1F 0000001C WB Addr 00000000 CPU-Dno 00000000 RegW/DR 00010000 IDE 00 00000013 CODE-01 01C00083 CODE-02 01C01103 CODE-03 01C02183 IDE 04 01C04203 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000 wr 00 00000000 CODE-00 00000000 CODE-0A 00000000 CODE-0B 00000000 </pre>

计数器第三拍 1/17	<pre> PC: 1F 00000010 INST: 1F 01C04203 rs1Data 00000000 rs2Data 00000000 PC: 1D 0000000C INST: 1D 01C02103 rs1Addr 00000000 rs2Addr 0000001C PC: 1E 00000000 INST: 1E 01C01103 CPU-Ram 00030003 PCJumpn 00000020 PC: 1F 00000004 INST: 1F 01C00003 B/PCE-S 00000000 B/C-Hzd 00000000 PC: 4B 00000000 INST: 4B 00000013 L/ABSel 00010001 PCIFNxt 00000014 ALU: r1n 00000000 ALU-Out 0000001C CPUAddr 00000000 ALUCtrl 00000001 ALU: R1n 0000001C WB-Data 00000000 CPU-Dai 00000000 WR-F10 00000001 Inst: L/1D 0000001C WB-addr 00000000 CPU-Dno 00000000 RegW/DR 00010000 CODE-00 00000013 CODE-01 01C00003 CODE-02 01C01103 CODE-03 01C02103 CODE-04 01C04203 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000 CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 00000000 CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 00000000 </pre>
计数器第四拍 (后续重复 3 遍)	<pre> PC: 1F 00000010 INST: 1F 01C04203 rs1Data 00000000 rs2Data 00000000 PC: 1D 0000000C INST: 1D 01C02103 rs1Addr 00000000 rs2Addr 0000001C PC: 1E 00000000 INST: 1E 01C01103 CPU-Ram 00030003 PCJumpn 00000020 PC: 1F 00000004 INST: 1F 01C00003 B/PCE-S 00000000 B/C-Hzd 00000000 PC: 4B 00000000 INST: 4B 00000013 L/ABSel 00010001 PCIFNxt 00000014 ALU: r1n 00000000 ALU-Out 0000001C CPUAddr 00000000 ALUCtrl 00000001 ALU: R1n 0000001C WB-Data 00000000 CPU-Dai 00000000 WR-F10 00000001 Inst: L/1D 0000001C WB-addr 00000000 CPU-Dno 00000000 RegW/DR 00010000 CODE-00 00000013 CODE-01 01C00003 CODE-02 01C01103 CODE-03 01C02103 CODE-04 01C04203 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000 CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 00000000 CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 00000000 </pre>
状态转移 重新执行 指令 1/17	<pre> x20: t3 00000000 x29: t4 00000000 x30: t5 00000000 x31: t6 00000000 PC: 1F 00000010 INST: 1F 01C04203 rs1Data 00000000 rs2Data 00000000 PC: 1D 0000000C INST: 1D 01C02103 rs1Addr 00000000 rs2Addr 0000001C PC: 1E 00000000 INST: 1E 01C01103 CPU-Ram 00030003 PCJumpn 00000020 PC: 1F 00000004 INST: 1F 01C00003 B/PCE-S 00000000 B/C-Hzd 00000000 PC: 4B 00000000 INST: 4B 00000013 L/ABSel 00010001 PCIFNxt 00000014 ALU: r1n 00000000 ALU-Out 0000001C CPUAddr 00000000 ALUCtrl 00000001 ALU: R1n 0000001C WB-Data 00000000 CPU-Dai 0000001C WR-F10 00000001 Inst: L/1D 0000001C WB-addr 00000000 CPU-Dno 00000000 RegW/DR 00010000 CODE-00 00000013 CODE-01 01C00003 CODE-02 01C01103 CODE-03 01C02103 CODE-04 01C04203 CODE-05 00000000 CODE-06 00000000 CODE-07 00000000 CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 00000000 CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 00000000 </pre>
进入下一条指令	<pre> PC: 1F 00000014 INST: 1F 01C05203 rs1Data 00000000 rs2Data 00000000 PC: 1D 00000010 INST: 1D 01C04203 rs1Addr 00000000 rs2Addr 0000001C PC: 1E 0000000C INST: 1E 01C02103 CPU-Ram 00030003 PCJumpn 0000002C PC: 1F 00000000 INST: 1F 01C01103 B/PCE-S 00000000 B/C-Hzd 00000000 PC: 4B 00000004 INST: 4B 01C00003 L/ABSel 00010001 PCIFNxt 00000010 ALU: r1n 00000000 ALU-Out 0000001C CPUAddr 00000000 ALUCtrl 00000001 ALU: R1n 0000001C WB-Data 0000001C CPU-Dai 0000001C WR-F10 00000001 Inst: L/1D 0000001C WB-addr 00000001 CPU-Dno 00000000 RegW/DR 00010001 CODE-00 00000013 CODE-01 01C00003 CODE-02 01C01103 CODE-03 01C02103 CODE-04 01C04203 CODE-05 01C05203 CODE-06 00000000 CODE-07 00000000 CODE-08 00000000 CODE-09 00000000 CODE-0A 00000000 CODE-0B 00000000 CODE-0C 00000000 CODE-0D 00000000 CODE-0E 00000000 CODE-0F 00000000 </pre>

指令 6 访问同一 set 不同 line，指令 9 访问不同 set，指令 12 访问也与前面都不相同，因此也都是 miss read，需要从内存中读取数据到 cache（过程类似，不放上板截图做赘述）。

指令 13 访问 set0，line0，与指令 9 是同一个块，但是二者 tag 不同，因此 miss。且指令 9 中将内容写入 cache，因此该块是 dirty 的，需要将数据写回内容：

指令 13 进入 mem 阶段，开始写回 1/ (1+17+17))	<pre> x20: t3 00000000 x29: t4 00000000 x26: t10 00000000 x27: t11 00000000 PC: 1F 00000010 INST: 1F FFDF006F rs1Data 00000000 rs31: t31 00000000 PC: 1D 0000000C INST: 1D 0ED00013 rs1Addr 00000000 rs2Addr 0000001C PC: 1E 00000000 INST: 1E 41002403 CPU-Ram 00000000 PCJumpn 0000002C PC: 1F 00000004 INST: 1F 40002303 B/PCE-S 00000000 B/C-Hzd 00000000 PC: 4B 00000000 INST: 4B 20002303 L/ABSel 00010001 PCIFNxt 00000010 ALU: r1n 00000000 ALU-Out 00000400 CPUAddr 00000000 ALUCtrl 00000001 ALU: R1n 00000410 WB-Data 00000000 CPU-Dai 00000000 WR-F10 00000001 Inst: L/1D 0000001C WB-addr 00000006 CPU-Dno 00000000 RegW/DR 00010001 CODE-00 00000013 CODE-01 01C00003 CODE-02 01C01103 CODE-03 01C02103 CODE-04 01C04203 CODE-05 01C05203 CODE-06 21002003 CODE-07 00000000 CODE-08 01C00093 CODE-09 00100023 CODE-0A 00101223 CODE-0B 00000000 CODE-0C 20002303 CODE-0D 40002303 CODE-0E 00000000 CODE-0F 00000000 </pre>
--	--

写回第一个 word 计时器第一拍 1/17	<pre> PC 1F 00000040 INST-IF FFDFF06F x30: t5 00000000 x31: PC 1D 0000003C INST-ID 0ED06813 rs1Data 00000000 rs2D PC 12E 00000030 INST-EX 41002403 rs1Addr 00000000 rs2A PC 112 00000034 INST-M 40002303 CPU-Ram 00020000 PCJu PC 4B 00000030 INST-WB 20002303 B/PCE-S 00000000 D/C- ALU a1n 00000000 ALU-Out 00000400 I/ABSel 00010001 PCIF ALU B1n 00000410 WB-Data 00000000 CPUAddr 00000000 ALUC PC 121D 000000ED WB-Addr 00000006 CPU-Da i 0000000C WR- DE-00 00000013 CODE-01 01C00083 CPU-Da0 00000000 RegA DE-04 01C04203 CODE-05 01C05283 CODE-02 01C01103 CODE DE-08 71C08093 CODE-09 00100023 CODE-06 21002003 CODE DE-0C 20002303 CODE-0D 40002303 CODE-0A 00101223 CODE DE-10 FFDFF06F CODE-0E 00000000 CODE-0B 00000000 CODE </pre>
写回第一个 word 计时器第二拍	<pre> PC 1F 00000040 INST-IF FFDFF06F x30: t5 00000000 x31: PC 1D 0000003C INST-ID 0ED06813 rs1Data 00000000 rs2D PC 12E 00000030 INST-EX 41002403 rs1Addr 00000000 rs2A PC 112 00000034 INST-M 40002303 CPU-Ram 00020004 PCJu PC 4B 00000030 INST-WB 20002303 B/PCE-S 00000000 D/C- ALU a1n 00000000 ALU-Out 00000400 I/ABSel 00010001 PCIF ALU B1n 00000410 WB-Data 00000000 CPUAddr 00000000 ALUC PC 121D 000000ED WB-Addr 00000006 CPU-Da i 0000000C WR- DE-00 00000013 CODE-01 01C00083 CPU-Da0 00000000 RegA DE-04 01C04203 CODE-05 01C05283 CODE-02 01C01103 CODE DE-08 71C08093 CODE-09 00100023 CODE-06 21002003 CODE DE-0C 20002303 CODE-0D 40002303 CODE-0A 00101223 CODE DE-10 FFDFF06F CODE-0E 00000000 CODE-0B 00000000 CODE </pre>
写回第一个 word 计时器第三拍	<pre> PC 1F 00000040 INST-IF FFDFF06F x30: t5 00000000 x27: PC 1D 0000003C INST-ID 0ED06813 rs1Data 00000000 x31: PC 12E 00000030 INST-EX 41002403 rs1Addr 00000000 rs2D PC 112 00000034 INST-M 40002303 CPU-Ram 00020005 rs2A PC 4B 00000030 INST-WB 20002303 B/PCE-S 00000000 PCJu ALU a1n 00000000 ALU-Out 00000400 I/ABSel 00010001 D/C- ALU B1n 00000410 WB-Data 00000000 CPUAddr 00000000 PCIF PC 121D 000000ED WB-Addr 00000006 CPU-Da i 0000000C ALUC DE-00 00000013 CODE-01 01C00083 CPU-Da0 00000000 WR- DE-04 01C04203 CODE-05 01C05283 CODE-02 01C01103 RegA DE-08 71C08093 CODE-09 00100023 CODE-06 21002003 CODE DE-0C 20002303 CODE-0D 40002303 CODE-0A 00101223 CODE DE-10 FFDFF06F CODE-0E 00000000 CODE-0B 00000000 CODE </pre>
写回第一个 word 计时器第四拍	<pre> PC 1F 00000040 INST-IF FFDFF06F x30: t5 00000000 x31: PC 1D 0000003C INST-ID 0ED06813 rs1Data 00000000 rs2D PC 12E 00000030 INST-EX 41002403 rs1Addr 00000000 rs2A PC 112 00000034 INST-M 40002303 CPU-Ram 00020006 PCJu PC 4B 00000030 INST-WB 20002303 B/PCE-S 00000000 D/C- ALU a1n 00000000 ALU-Out 00000400 I/ABSel 00010001 PCIF ALU B1n 00000410 WB-Data 00000000 CPUAddr 00000000 ALUC PC 121D 000000ED WB-Addr 00000006 CPU-Da i 0000000C WR- DE-00 00000013 CODE-01 01C00083 CPU-Da0 00000000 RegA DE-04 01C04203 CODE-05 01C05283 CODE-02 01C01103 CODE DE-08 71C08093 CODE-09 00100023 CODE-06 21002003 CODE DE-0C 20002303 CODE-0D 40002303 CODE-0A 00101223 CODE DE-10 FFDFF06F CODE-0E 00000000 CODE-0B 00000000 CODE </pre>

后续重复三次写回 word 后，状态转移至读 word，后面情况和上述分析的 miss read 类似，这里不做赘述。

指令 14 访问 set1, line0，同指令一一样，但是指令一是读取指令，没有写，因此 cache 是非 dirty 的，不需要写回。

四、讨论、心得

本实验只需要填几个空完整有限状态机的状态转移过程，因此整体实验比较简单，过程中也没有遇到什么困难。

另外，跑仿真时发现只有前面几条指令的波形，在设置中修改 simulation 的时间后解决。