浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：夏尤楷

学　　院：计算机科学与技术学院

　　系：计算机科学与技术

专　　业：计算机科学与技术

学　　号：3210104331

指导教师：陈文智

2023 年 11 月 21 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：带缓存的流水线 CPU

学生姓名：夏尤楷 专业：计算机科学与技术 学号：3210104331

同组学生姓名：来思锐 指导老师：陈文智

实验地点：曹西 301 实验日期：2023 年 10 月 31 日

一、 实验目的和要求

1. 理解缓存控制单元的原理和状态机。
2. 掌握缓存控制单元的设计方法，将其综合入 CPU。
3. 掌握缓存控制单元的验证方法，比较当有和无缓存时 CPU 的性能。

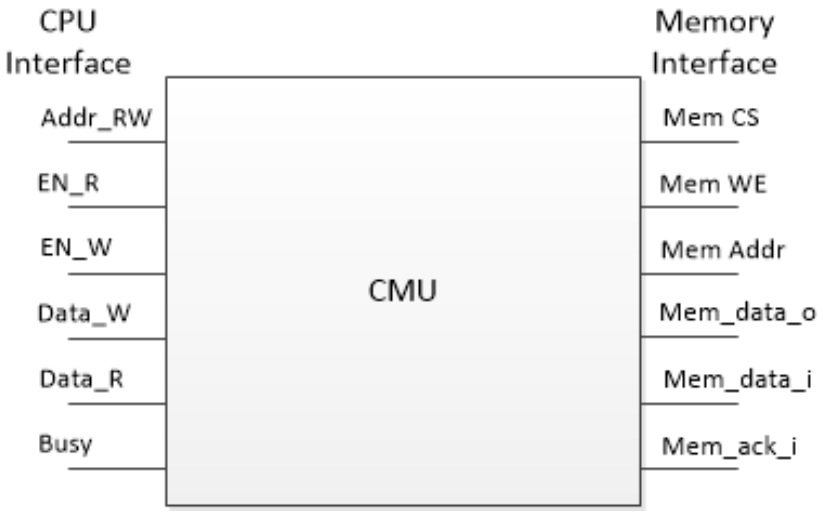
二、 实验内容和原理

内容：

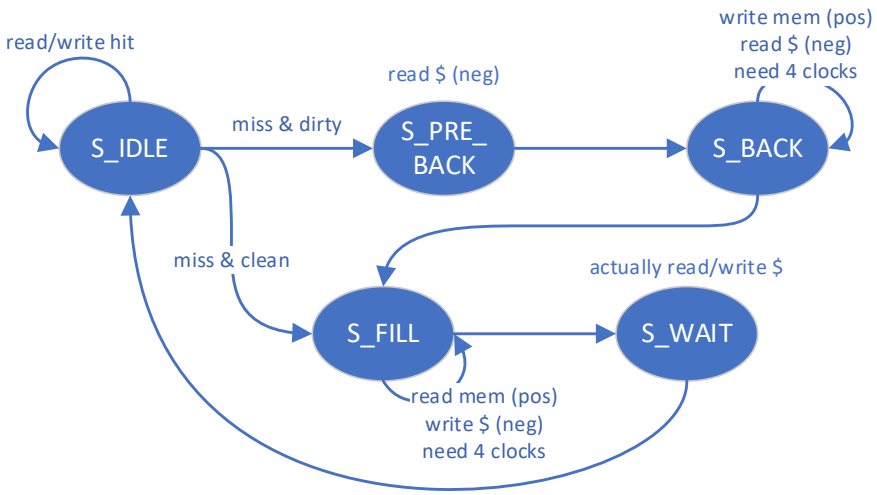
1. 设计缓存控制单元，并将其综合入 CPU。
2. 观察并分析仿真的波形。
3. 比较当有和无缓存时 CPU 的性能。

原理：

缓存控制单元（CMU）的接口分为 CPU 接口和内存接口，如下图：



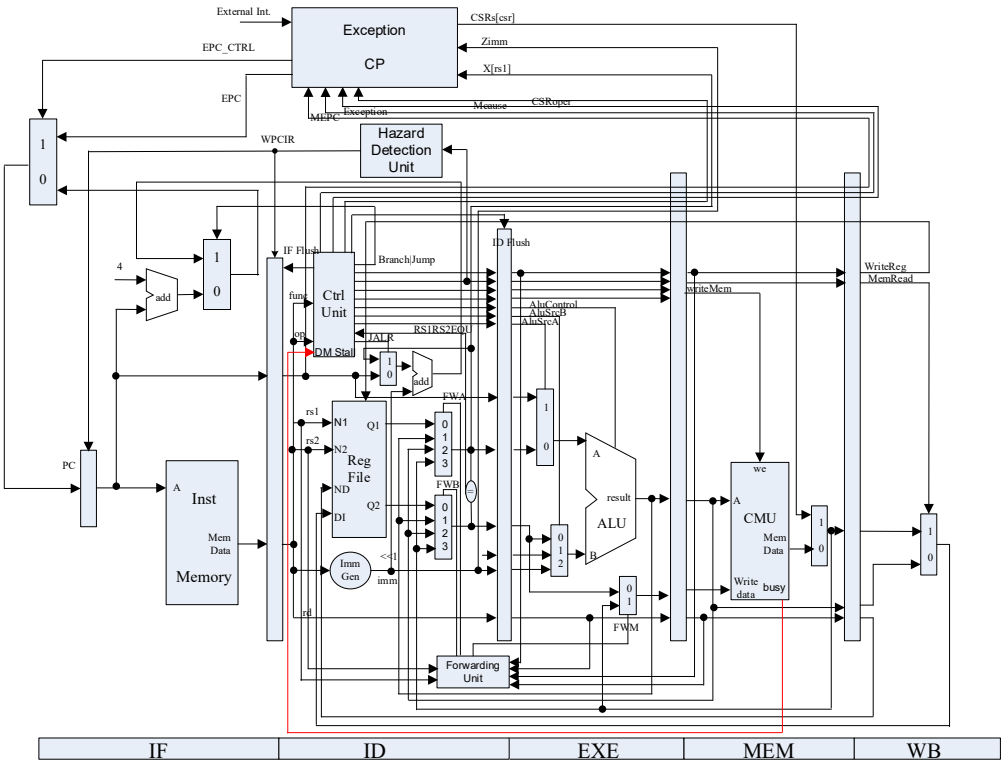
在实现缓存的读和写的功能时，我们采用状态机来实现，状态机的各状态及互相切换的关系如下图：



下面对各个状态及其如何切换到其他状态做解读：

1. S_IDLE：空闲状态，不进行内存操作，缓存操作命中的情况下一直处于这个状态。当缓存操作失配时，如果要替换掉的页为脏，则切换至 S_PRE_BACK 状态；如果要替换掉的页是干净的，则切换至 S_FILL 状态。
2. S_PRE_BACK：为了写回，先读一次缓存，然后进入 S_BACK 状态。
3. S_BACK：上升沿将上个状态的数据写回到内存，下降沿从缓存读下次需要写回的数据（因此最后一次读无意义），由计数器控制直到整个缓存行全部写回。由于内存设置为 4 个周期完成读写 1 个字操作，因此需要等待内存给出确认信号，才能进行状态的改变（变为 S_FILL）。
4. S_FILL：上升沿从内存读取数据，下降沿向缓存写入数据，由计数器控制直到整个缓存行全部写入。与 S_BACK 类似，内存同样需要四个周期来读写 1 个字，需要等待确认信号然后再进入 S_WAIT 状态。
5. S_WAIT：执行之前由于失配而不能进行的缓存操作。

加入缓存控制单元的 CPU 结构图如下图。



三、 实验过程和数据记录

1. 状态机状态切换

根据“二”中介绍的缓存控制单元状态机的切换条件和关系，不难完成以下控制状态机的代码：

```
// state ctrl
always @ (*) begin
    if (rst) begin
        next_state = S_IDLE;
        next_word_count = 2'b00;
    end
    else begin
        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

            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 + 1;
            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 + 1;
            else
                next_word_count = word_count;
        end

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

2. CMU 发出的 CPU 停顿控制信号

显然，在 CMU 状态机处于 S_PRE_BACK、S_BACK、S_FILL 状态，或是正处于 S_IDLE 状态但下一状态不是 S_IDLE 状态时，根据 CPU 发出的数据请求，缓存肯定要将相应的数据替换进来，这些时候 CPU 没有办法从缓存中读取数据，须由缓存发出一个信号来停顿 CPU。该信号赋值如下：

```
assign stall = (state==S_IDLE & next_state!=S_IDLE) | (state==S_BACK) |
   (state==S_FILL) | (state==S_PRE_BACK);
```

四、 实验结果分析

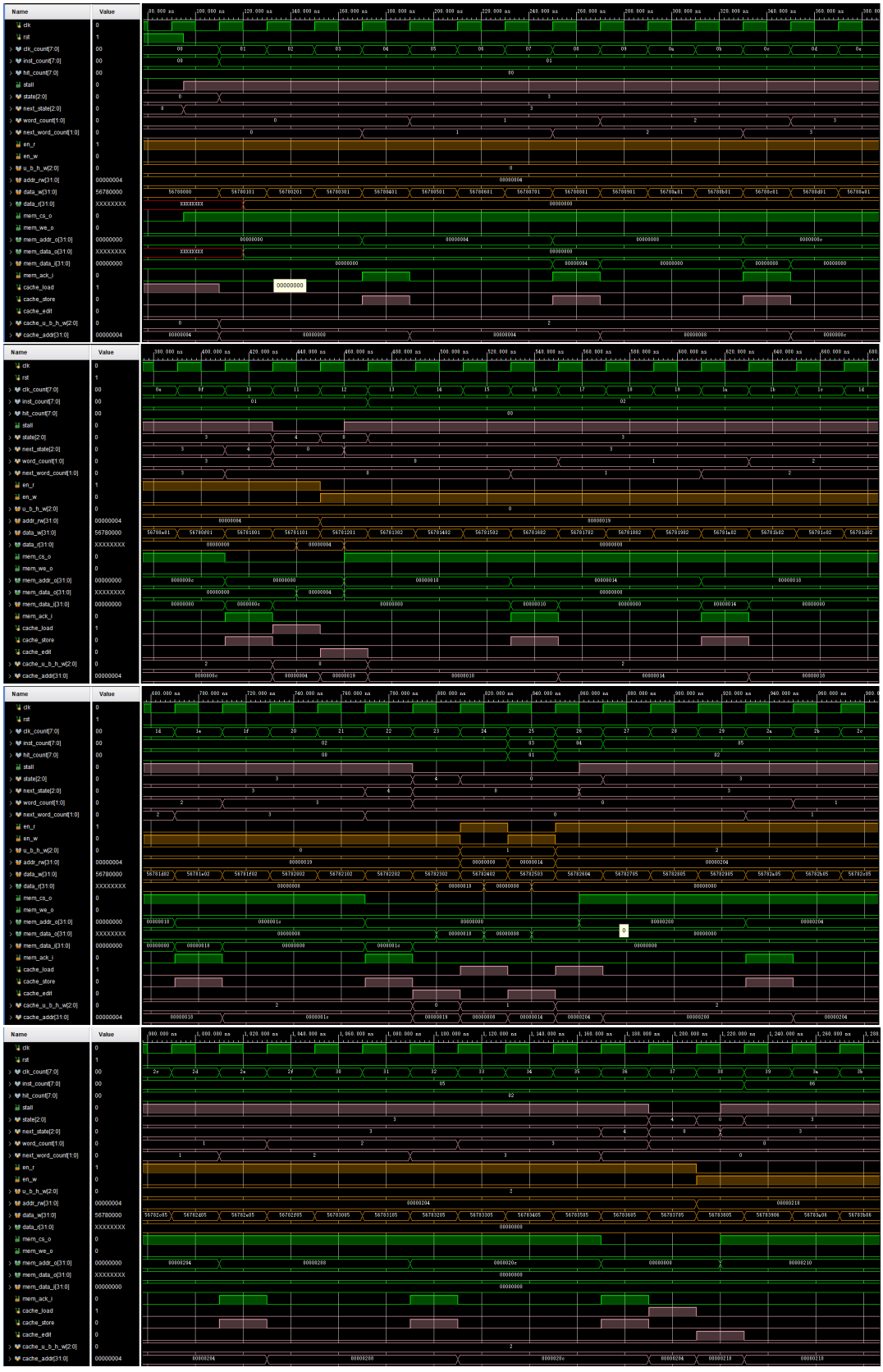
仿真：

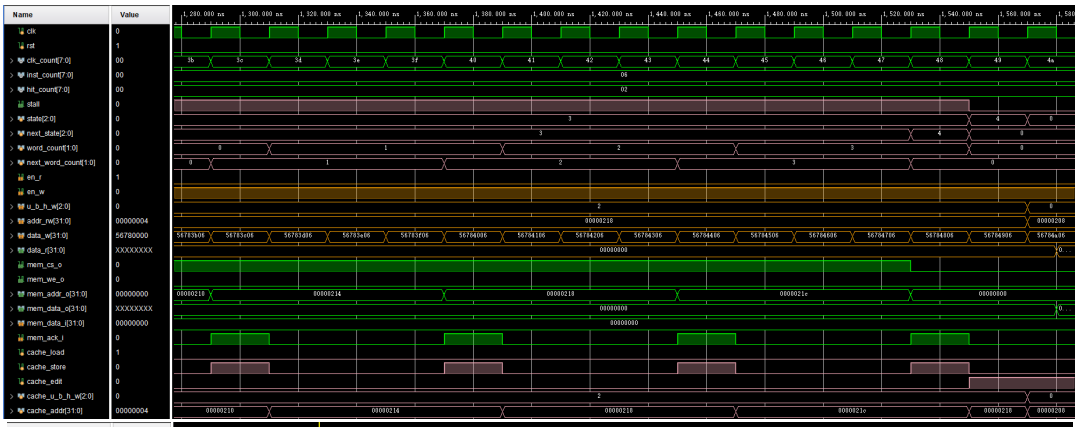
使用以下仿真代码来测试 CMU（其中相当于设置了 9 条指令）：

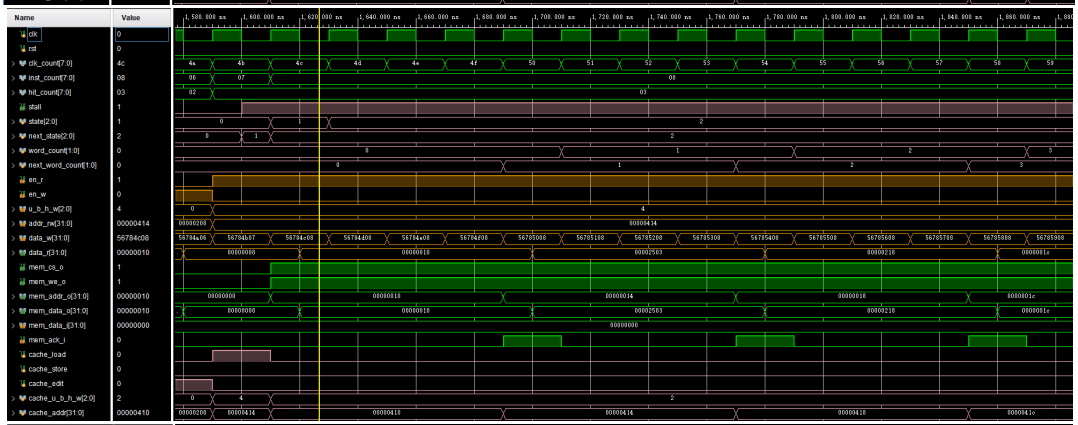
```
   reg [39:0] data [0:9];
   initial begin
       data[0] = 40'h0_2_00000004;  // read miss             1+17
       data[1] = 40'h0_3_00000019;  // write miss            1+17
       data[2] = 40'h1_2_00000008;  // read hit              1
       data[3] = 40'h1_3_00000014;  // write hit          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

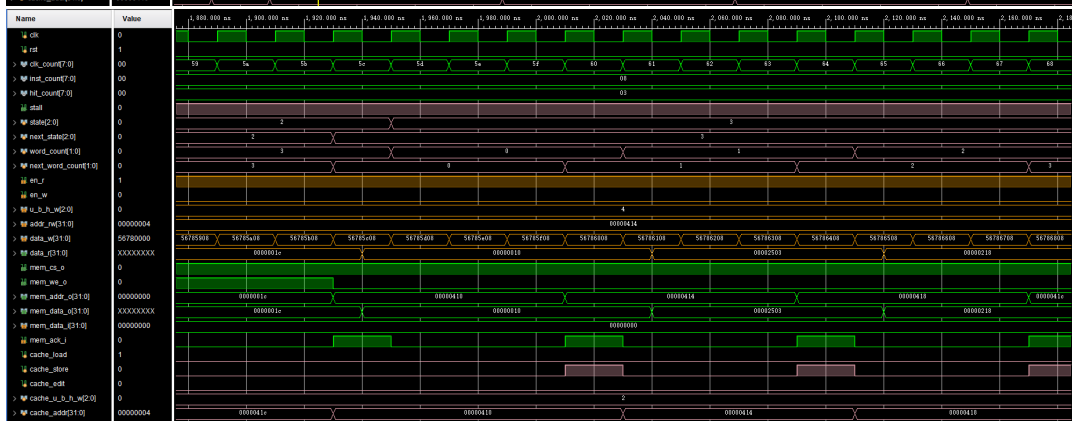
```
valid = data[index][33],
write = data[index][32],
addr = data[index][31:0];
```

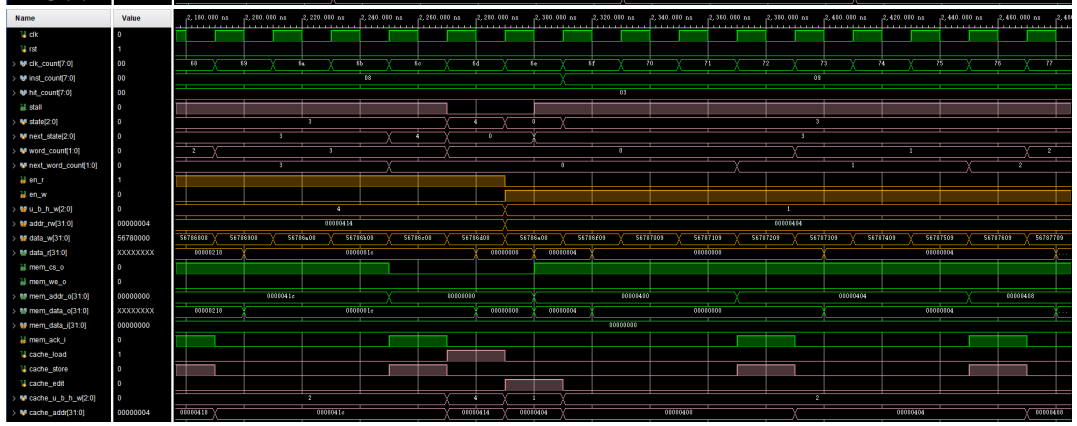
仿真结果如下面几张图：

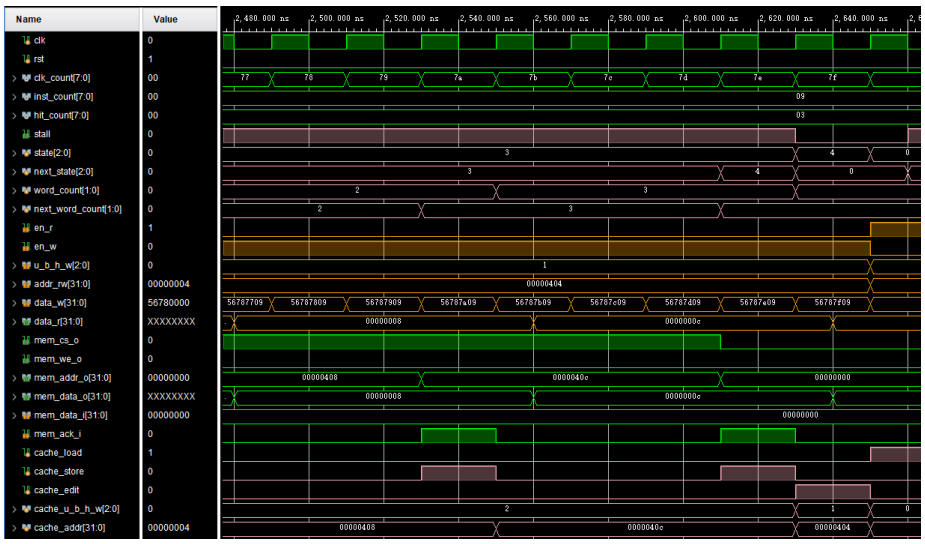



Name
Value
clk
rst
clk_count[7:0]
inst_count[7:0]
hit_count[7:0]
stall
state[2:0]
next_state[2:0]
word_count[1:0]
next_word_count[1:0]
en_r
en_w
u_b_h_w[2:0]
addr_rw[31:0]
data_w[31:0]
data_r[31:0]
mem_cs_o
mem_we_o
mem_addr_o[31:0]
mem_data_o[31:0]
mem_data_i[31:0]
mem_ack_i
cache_load
cache_store
cache_edit
cache_u_b_h_w[2:0]
cache_addr[31:0]


Name
Value
clk
rst
clk_count[7:0]
inst_count[7:0]
hit_count[7:0]
stall
state[2:0]
next_state[2:0]
word_count[1:0]
next_word_count[1:0]
en_r
en_w
u_b_h_w[2:0]
addr_rw[31:0]
data_w[31:0]
data_r[31:0]
mem_cs_o
mem_we_o
mem_addr_o[31:0]
mem_data_o[31:0]
mem_data_i[31:0]
mem_ack_i
cache_load
cache_store
cache_edit
cache_u_b_h_w[2:0]
cache_addr[31:0]


Name
Value
clk
rst
clk_count[7:0]
inst_count[7:0]
hit_count[7:0]
stall
state[2:0]
next_state[2:0]
word_count[1:0]
next_word_count[1:0]
en_r
en_w
u_b_h_w[2:0]
addr_rw[31:0]
data_w[31:0]
data_r[31:0]
mem_cs_o
mem_we_o
mem_addr_o[31:0]
mem_data_o[31:0]
mem_data_i[31:0]
mem_ack_i
cache_load
cache_store
cache_edit
cache_u_b_h_w[2:0]
cache_addr[31:0]


Name
Value
clk
rst
clk_count[7:0]
inst_count[7:0]
hit_count[7:0]
stall
state[2:0]
next_state[2:0]
word_count[1:0]
next_word_count[1:0]
en_r
en_w
u_b_h_w[2:0]
addr_rw[31:0]
data_w[31:0]
data_r[31:0]
mem_cs_o
mem_we_o
mem_addr_o[31:0]
mem_data_o[31:0]
mem_data_i[31:0]
mem_ack_i
cache_load
cache_store
cache_edit
cache_u_b_h_w[2:0]
cache_addr[31:0]



当在缓存和内存之间读写数据时，往往要读写 4 个字，而内存中每个字的读写要耗费四个时钟周期，其结束由内存发出的 ack 信号（值为 1，在图中为 mem_ack_i）来标志。

下面对仿真中 CMU 的行为作一些解释：

（1）100ns~450ns：在第 0 个指令期间，en_r == 1，u_b_h_w == 3'b0，addr_rw == 32'h00000004，表明要从缓存中读取内存中地址为 32'h00000004 的 1 字节有符号数据，但是 cache 中相应空间（第 0 行）全空（是干净的），于是读失配，110ns 时 CMU 状态从 S_IDLE（0）跳转到 S_FILL（3），并从内存中逐个读出从 32'h00000000 开始的 4 个字并写入缓存中（每个字占用 4 个时钟周期）。于是，在 16 个时钟周期后（430ns），CMU 状态从 S_FILL（3）跳转到 S_WAIT（4），持续一个时钟周期，进行从缓存读取数据到 CPU 的操作。之后 CMU 状态回到 S_IDLE（0）。

（2）450ns~810ns：在第 1 个指令期间，en_w == 1，u_b_h_w == 3'b0，addr_rw == 32'h00000019，表明要往内存地址 32'h00000019 对应的缓存空间中写入 1 字节有符号数据，但是 cache 中相应空间（第 1 行）全空（是干净的），于是写失配，在 S_IDLE 一个时钟周期后（470ns）,CMU 状态跳转到 S_FILL（3），并从内存中逐个读出从 32'h00000010 开始的 4 个字并写入缓存中（每个字占用 4 个时钟周期）。于是，在 16 个时钟周期后（430ns），CMU 状态从 S_FILL（3）跳转到 S_WAIT（4），持续一个时钟周期，进行 CPU 写数据到缓存的操作。之后 CMU 状态回到 S_IDLE（0）。

（3）810ns~830ns：在第 2 个指令期间，en_r == 1，u_b_h_w == 3'b1，addr_rw

== 32’h00000008，表明要从缓存中读取内存中地址为 32’h00000008 的 1 个半字有符号数据。cache 中相应空间（第 0 行）在经过第 0 个指令后已经储存有从 32’h00000000 开始的 4 个字的数据，这 1 个半字也在其中，所以读命中，在一个时钟周期内从缓存读取数据到 CPU，之后 CMU 状态仍保持为 S_IDLE（0）。

（4）830ns~850ns：在第 3 个指令期间，en_w == 1，u_b_h_w == 3’b1，addr_rw == 32’h00000014，表明要往内存地址 32’h00000014 对应的缓存空间中写入 1 个半字有符号数据。cache 中相应空间（第 1 行）在经过第 1 个指令后已经储存有从 32’h00000010 开始的 4 个字的数据，这 1 个半字也在其中，所以写命中，在一个时钟周期内从 CPU 写数据到缓存，之后 CMU 状态仍保持为 S_IDLE（0）。

（5）850ns~1210ns：在第 4 个指令期间，en_r == 1，u_b_h_w == 3’b10，addr_rw == 32’h00000204，表明要从缓存中读取内存中地址为 32’h000000204 的 1 个字有符号数据，但是 cache 中相应空间（第 0 行）的两个空间中，一个已经放置了 32’h00000000 开始的 4 个字的数据，32’h000000204 不被包含在其中；另一个空间为空（干净）。于是读失配，在 S_IDLE（0）一个周期后（870ns），CMU 状态跳转到 S_FILL（3），并从内存中逐个读出从 32’h00000200 开始的 4 个字并写入缓存中（每个字占用 4 个时钟周期），于是，在 16 个时钟周期后（1190ns），CMU 状态从 S_FILL（3）跳转到 S_WAIT（4），持续一个时钟周期，进行从缓存读取数据到 CPU 的操作。之后 CMU 状态回到 S_IDLE（0）。

（6）1210ns~1570ns：在第 5 个指令期间，en_w == 1，u_b_h_w == 3’b10，addr_rw == 32’h00000218，表明要往内存地址 32’h000000218 对应的缓存空间中写入 1 个字有符号数据，但是 cache 中相应空间（第 1 行）的两个空间中，一个已经放置了 32’h00000010 开始的 4 个字的数据，32’h000000218 不被包含在其中；另一个是空的（干净），于是写失配，在 S_IDLE（0）一个周期后（1230ns），CMU 状态跳转到 S_FILL（3），并从内存中逐个读出从 32’h00000210 开始的 4 个字并写入缓存中（每个字占用 4 个时钟周期），于是，在 16 个时钟周期后（1550ns），CMU 状态从 S_FILL（3）跳转到 S_WAIT（4），持续一个时钟周期，进行 CPU 写数据到缓存的操作。之后 CMU 状态回到 S_IDLE（0）。

（7）1570ns~1590ns：在第 6 个指令期间，en_w == 1，u_b_h_w == 3’b0，addr_rw == 32’h00000208，表明要往内存地址 32’h000000208 对应的缓存空间中

写入 1 字节有符号数据，cache 中相应空间（第 0 行）在经过第 4 个指令后已经储存有从 32’h00000200 开始的 4 个字的数据，这 1 字节也在其中，所以写命中，在一个时钟周期内从 CPU 写数据到缓存，之后 CMU 状态仍保持为 S_IDLE（0）。

（8）1590ns~2290ns：在第 7 个指令期间，en_r == 1，u_b_h_w == 3’b100，addr_rw == 32’h00000414，表明要从内存地址 32’h000000414 对应的缓存空间中读取 1 字节无符号数据。cache 中相应空间（第 1 行）在经过第 1 和 5 个指令后已经分别储存有从 32’h00000010 和从 32’h00000210 开始的 4 个字的数据，这 1 字节不在两个之中的任何一个，所以读失配。32’h00000010 开始的 4 个字最近被第 3 条指令写过，脏；32’h00000210 开始的 4 个字最近被第 5 条指令写过，脏。根据 LRU 原则，缓存中被替换下来的是 32’h00000010 开始的脏的 4 个字。故在 S_IDLE（0）一个周期后（1610ns），CMU 先变为 S_PRE_BACK 状态（1），从缓存中读取数据，再在 1 个周期后进入 S_BACK 状态（2），花费 16 个时钟周期（因为每个字的读写要 4 个时钟周期），按照先写数据进内存后读数据出缓存的顺序，完成 4 个字的读写，CMU 再在 1950ns 时切换到 S_FILL 状态（3），花费 16 个时钟周期将从地址 32’h00000410 开始的 4 个字（每个字的读写花费 4 个时钟周期）从内存写到缓存中，CMU 再在 2270ns 时由 S_FILL 状态（3）转变为 S_WAIT 状态（4），在接下去的一个周期内从缓存中读取数据，然后回到 S_IDLE（0）状态。

（9）2290ns~2650ns：在第 8 个指令期间，en_w == 1，u_b_h_w == 3’b1，addr_rw == 32’h00000404，表明要往内存地址 32’h000000404 对应的缓存空间中写入 1 个半字无符号数据，cache 中相应空间（第 0 行）在经过第 0 和 4 个指令后已经分别储存有从 32’h00000000 和从 32’h00000200 开始的 4 个字的数据，这 1 个半字不在两个之中的任何一个，所以写失配。32’h00000000 开始的 4 个字最近被第 2 条指令读过，未被写过，干净；32’h00000200 开始的 4 个字最近被第 6 条指令写过，脏。根据 LRU 原则，缓存中被替换下来的是 32’h00000000 开始的干净的 4 个字。故在 S_IDLE（0）一个周期后（2310ns），CMU 变为 S_FILL 状态（3），花费 16 个时钟周期将从地址 32’h00000400 开始的 4 个字（每个字的读写花费 4 个时钟周期）从内存写到缓存中，CMU 再在 2630ns 时由 S_FILL 状态（3）转变为 S_WAIT 状态（4），在接下去的一个周期内往缓存中写数据，然后

回到 S_IDLE（0）状态。

物理验证：

使用以下指令进行物理验证：

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)	# FFFFF0F0 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		lw x7, 20(x0)	
8	402200b3	20		lui x1 0xABCDE	
9	71c08093	24		addi x1, x1, 0x71C	# x1 = 0xABCDE71C
10	00100023	28		sb x1, 0x0(x0)	# miss, read 0x000~0x00C to cache set 0 line 0
11	00101223	2C		sh x1, 0x4(x0)	# hit
12	00102423	30		sw x1, 0x8(x0)	# hit
13	20002303	34		lw x6, 0x200(x0)	# miss, read 0x200~0x20C to cache set 0 line 1
14	40002383	38		lw x7, 0x400(x0)	# miss, write 0x000~0x00C back to ram, then read 0x400~40C to cache set 0 line 0
15	41002403	3C		lw x8, 0x410(x0)	# miss, no write back because of clean, read 0x410~41C to chache set 1 line 0
16	0ed06813	40	loop:	lori x16, x0, 0xED	# end
17	ffdff06f	44		jal x0, loop	

使用以下内存数据进行物理验证：

NO.	Data	Addr.	NO.	Data	Addr.
0	000080BF	0	16	00000000	40
1	00000008	4	17	00000000	44
2	00000010	8	18	00000000	48
3	00000014	C	19	00000000	4C
4	FFFF0000	10	20	A3000000	50
5	0FFF0000	14	21	27000000	54
6	FF000F0F	18	22	79000000	58
7	F0F0F0F0	1C	23	15100000	5C
8	00000000	20	24	00000000	60

9	00000000	24	25	00000000	64
10	00000000	28	26	00000000	68
11	00000000	2C	27	00000000	6C
12	00000000	30	28	00000000	70
13	00000000	34	29	00000000	74
14	00000000	38	30	00000000	78
15	00000000	3C	31	00000000	7C

物理验证已通过线下验收。

五、 讨论与心得

这次实验让我对缓存的具体运行有了细致的理解。在理论课上，我们只是了解到缓存的作用，以及在何时用内存中的数据替换缓存中的数据，哪些替换是要把缓存里的数据写回内存之类的，但这些行为如何配合计算机 CPU 的时钟周期而发生，理论课上没有讲过；而这次实验，让我们直观地学习了这些东西。