浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：郑乔尹

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：3210104169

指导教师：姜晓红

2023 年　11　月　　20　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 Cache 的流水线 CPU

学生姓名：郑乔尹　专业：计算机科学与技术　学号：3210104169

同组学生姓名：无　　指导老师：姜晓红

实验地点：曹西 301　　实验日期：2023 年 11 月 14 日

20% bonus: 4-way set associative

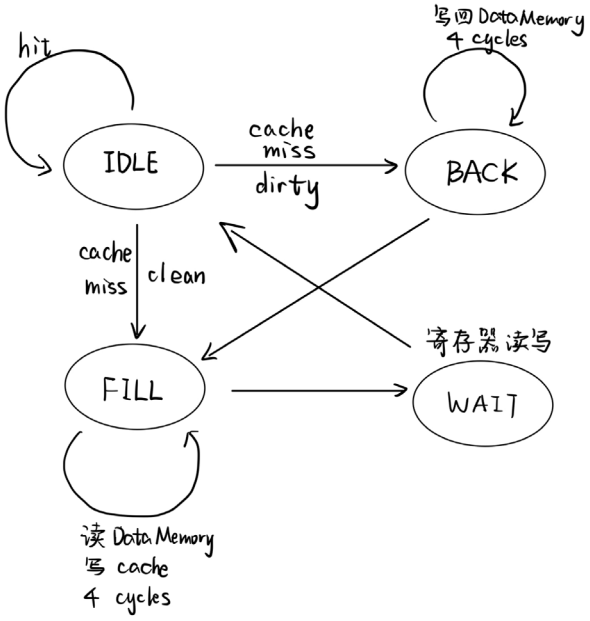
一、 实验目的和要求

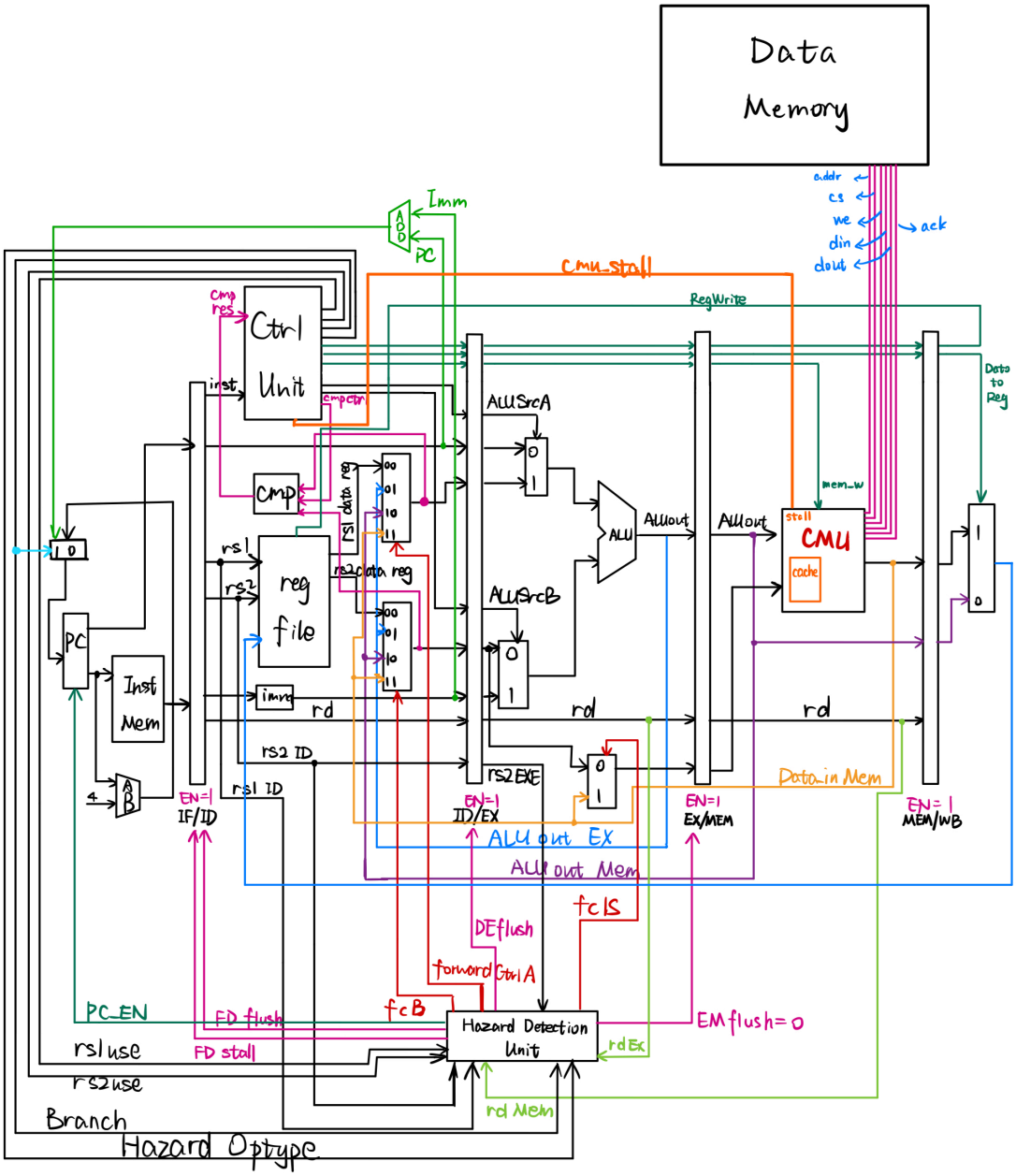
Task 1: 写出本次实验的目的与要求 (5 points)

1. 理解 Cache Management Unit 以及其中状态机的工作原理
2. 掌握 CMU 的设计方法，并将它集成到 CPU 中
3. 掌握 CMU 的验证方法并比较 CPU 在是否有 cache 情况下的性能

二、 实验内容和原理

Task 2: 简要画出本次实验实现的 core 的电路图和 CMU 的状态机。（可以使用 PPT 上的线路图进行修改，但是必须和自己的实现保持一致。如果不一致，本题将不给分） (10 points)





Task 3: 请给出 CMU 状态转移逻辑的代码并加以解释 (10 points)

```
always @ (*) 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_BACK;
                else
                    next_state = S_FILL;
            end
            else begin
```

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

CMU 共有 4 种状态，分别为 S_IDLE，S_BACK，S_FILL，S_WAIT，以下详细介绍每个状态的含义以及如何进行状态转移的逻辑：

1. S_IDLE：IDLE 状态，没有访问主存，cache hit，继续停留在当前状态；

假如 cache 有效但是 dirty，需要改写 cache，转移至 BACK 状态；否则转移至 FILL 状态。

2. S_BACK：写回状态，在当前 cache line dirty 但 miss 时，需要先将 cache 中的数据写回内存，当内存接收信号拉高(mem_ack_i=1)且 word_count 达到 2'b11 时(4 个 cycle 过去，整个 cache line 被替换)，状态转移至 FILL
3. S_FILL：读内存，同样需要 4 个 cycle，当 4 个 cycle 过去且内存接收信号拉高，cache line 的替换完成，状态转移至 WAIT
4. S_WAIT：真正读/写寄存器的状态，从 cache 中存取响应数据，然后回到 IDLE 状态。

Task 4: 请给出 CMU CPU 侧输出引脚的逻辑、代码以及解释 (10 points)

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

data_r 是 CPU 此时读取的数据，将其直接连接至 cache 的数据输出，当 cache 有效且命中时该值被取用。

stall 是 CPU 的 cmu_stall 信号，假如下一个状态不是 IDLE，则说明当前还在进行内存存取或者 cache 存取操作，cache 中的数据并未准备好，需要将 CPUstall，直至数据被准备好。

Task 5: 请给出 CMU MEM 侧输出引脚的逻辑、代码以及解释 (5 points)

```
// mem ctrl
    always @ (*) begin
        case (state)
            S_IDLE, S_WAIT: begin
                mem_cs_o = 1'b0;
                mem_we_o = 1'b0;
                mem_addr_o = 32'b0;
            end

            S_BACK: begin
                mem_cs_o = 1'b1;
                mem_we_o = 1'b1;
                mem_addr_o = {cache_tag,
addr_rw[ADDR_BITS-TAG_BITS-1:BLOCK_WIDTH], word_count,
{ELEMENT_WORDS_WIDTH{1'b0}}}; // next word count -> word count
            end

            S_FILL: begin
```

```
                mem_cs_o = 1'b1;
                mem_we_o = 1'b0;
                mem_addr_o = {addr_rw[ADDR_BITS-1:BLOCK_WIDTH], word_count,
{ELEMENT_WORDS_WIDTH{1'b0}}}; // next word count -> word count
            end

            default: begin
                mem_cs_o = 1'b0;
                mem_we_o = 1'b0;
                mem_addr_o = 32'b0;
            end
        endcase
    end
    assign mem_data_o = cache_dout;
```

1. S_IDLE 和 S_WAIT：在这些状态下，禁用内存芯片选择（mem_cs_o）和写使能（mem_we_o）。内存地址（mem_addr_o）设置为零。
2. S_BACK：在 S_BACK 状态下，启用内存芯片选择和写使能，表示要执行写操作。内存地址设置为{cache_tag, addr_rw[ADDR_BITS-TAG_BITS-1:BLOCK_WIDTH],word_count, {ELEMENT_WORDS_WIDTH{1'b0}}}，通过当前的 word_count 读取对应的 word。
3. S_FILL：在 S_FILL 状态下，启用内存芯片选择，禁用写使能，表示执行读操作。内存地址设置为 {addr_rw[ADDR_BITS-1:BLOCK_WIDTH],word_count,{ELEMENT_WORDS_WIDTH{1'b0}}}，由于是读取，不需要 cache_tag，通过 word_count 读取对应的 word。
4. mem_data_o：mem_data_o 信号连接至 cache_dout，即 cache 的输出数据，表示在读操作中，从缓存中读取的数据传递给内存。

Task 6: 请给出 CMU stall 的逻辑、代码以及解释，包含 CMU 部分和 core 部分。目前需要改变流水线的操作有 LR stall、predict not taken、interrupt/exception、mret、cmu stall。他们的优先级如何？如何实现优先级？(15 points)

CMU 部分：

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

下一状态不是 IDLE，根据设计好的状态机，必定是处于 miss 的处理过程中或者即将进行 miss 处理，故将 stall 信号拉高。

core 部分：

```
assign reg_FD_EN = ~cmu_stall;
assign reg_DE_EN = ~cmu_stall;
assign reg_EM_EN = ~cmu_stall;
assign reg_MW_EN = ~cmu_stall;
assign reg_EM_flush = 1'b0;
assign reg_MW_flush = cmu_stall;
assign PC_EN_IF = ~stall && ~cmu_stall; // stall, no IF
```

一旦发生 cmu_stall，通过关闭各阶段间寄存器的使能信号，暂停流水线。

优先级：

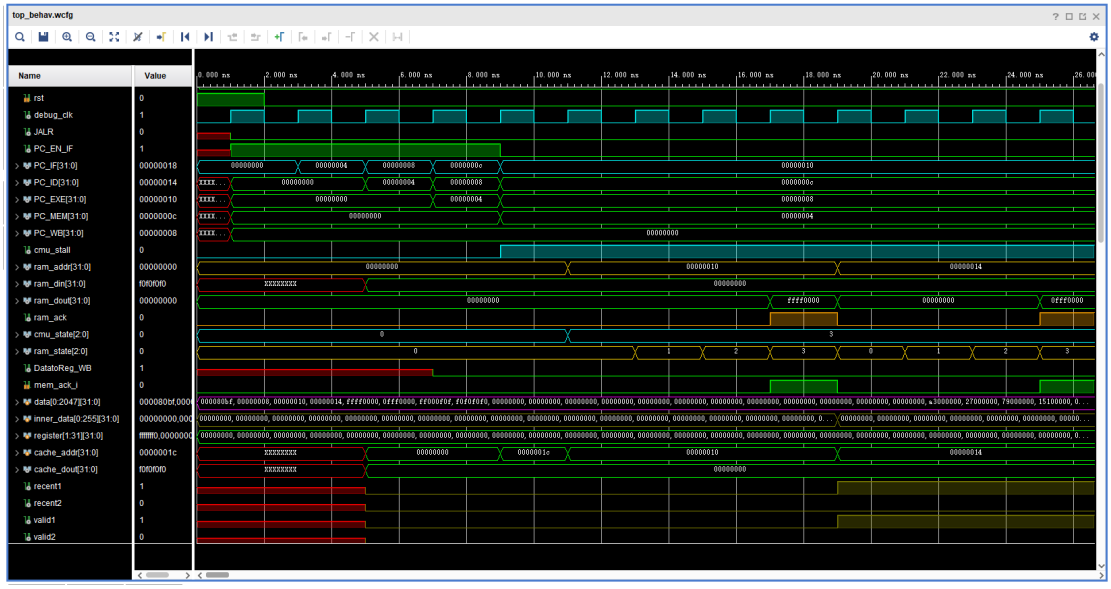
Predict not taken> LR stall> cmu_stall > mret > Interrupt/exception；根据发生的阶段，predict not taken 如果需要 flush FD，在 ID 阶段就可拉高 flush 信号，从而实现最高优先级，LR stall 实际上在前一条 load 指令进入 EXE 阶段就可以判断，故通过在 Load 指令进入 EXE，Store 指令进入 ID 阶段时给出 stall 信号，可以规避优先级冲突；cmu_stall 在 MEM 阶段检测，即需要在 cache 中 load/store 了才知道是否命中，进而确定是否需要 stall；mret 也在 MEM 阶段，但是 mret 在中断处理结束被调用故优先级低于 cmu_stall，因为模板整体将中断放在 WB 处理；进入中断 interrupt/exception 相关的指令在实验中都放在 WB 阶段处理，故有着最低的优先级。

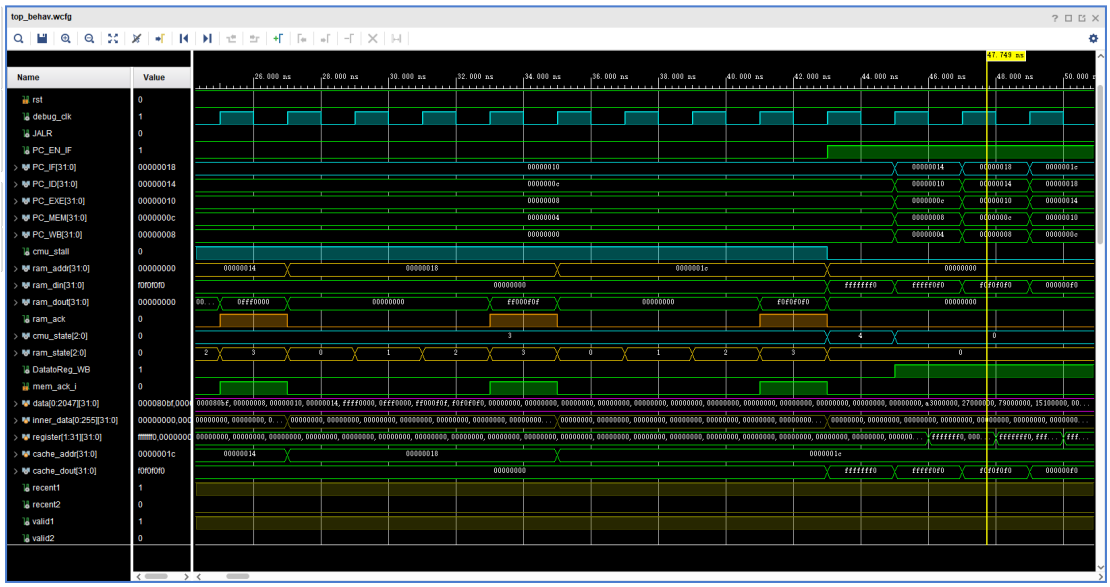
三、 实验过程和数据记录及结果分析

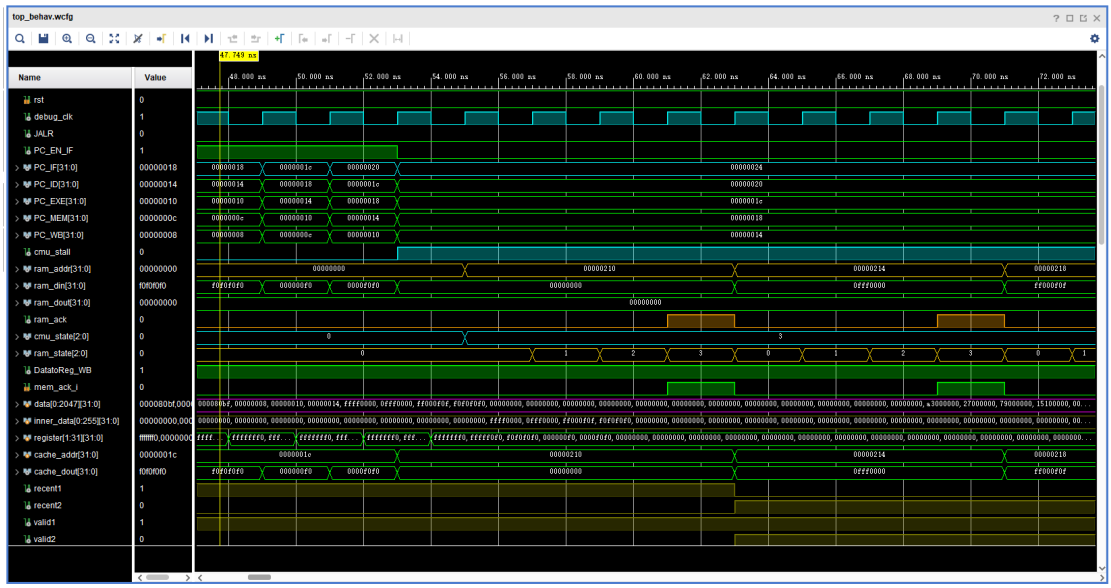
仿真图片应完整包含时间信息和信号名称。

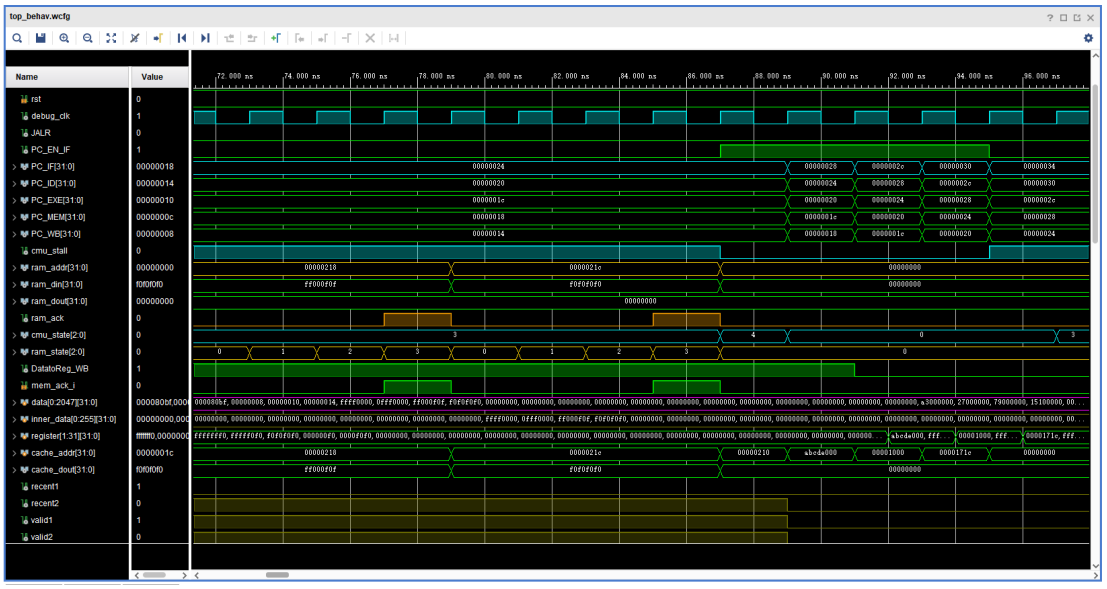
对仿真的解释示例：XXXns，X 信号变为 X，由于 XXX，导致 X 信号变为 XXX，……，我们发现 X 被 forward 到了 X。

Task 7: 请给出本次实验仿真的完整截图 (5 points)

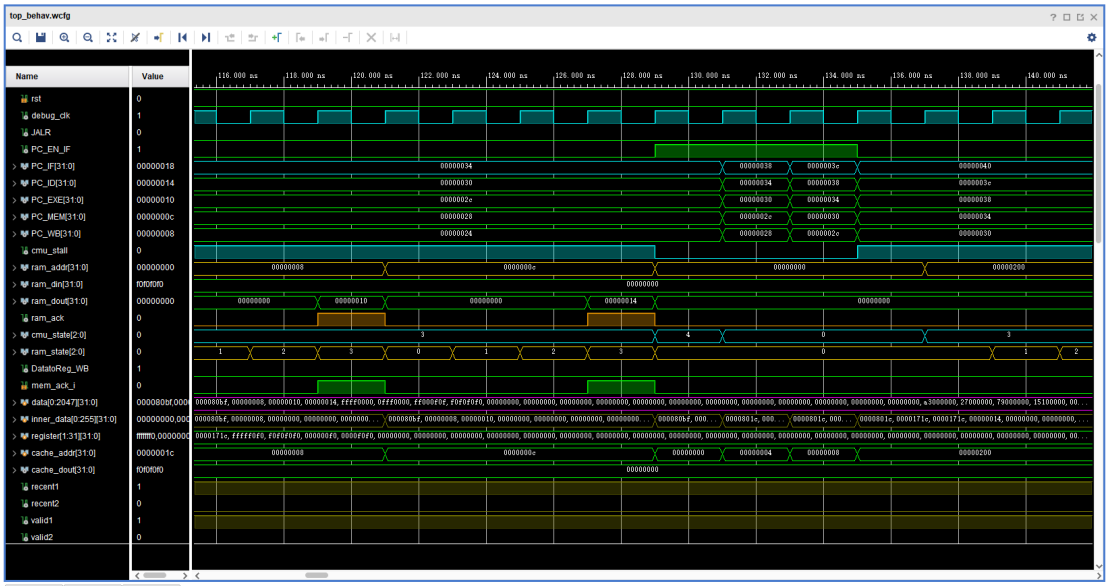

top_behav.wcfg
Name
Value
rst
debug_clk
JALR
PC_EN_IF
PC_IF[31:0]
PC_ID[31:0]
PC_EXE[31:0]
PC_MEM[31:0]
PC_WB[31:0]
cmu_stall
ram_addr[31:0]
ram_din[31:0]
ram_dout[31:0]
ram_ack
cmu_state[2:0]
ram_state[2:0]
DatatoReg_WB
mem_ack_i
data[0:2047][31:0]
inner_data[0:255][31:0]
register[1:31][31:0]
cache_addr[31:0]
cache_dout[31:0]
recent1
recent2
valid1
valid2

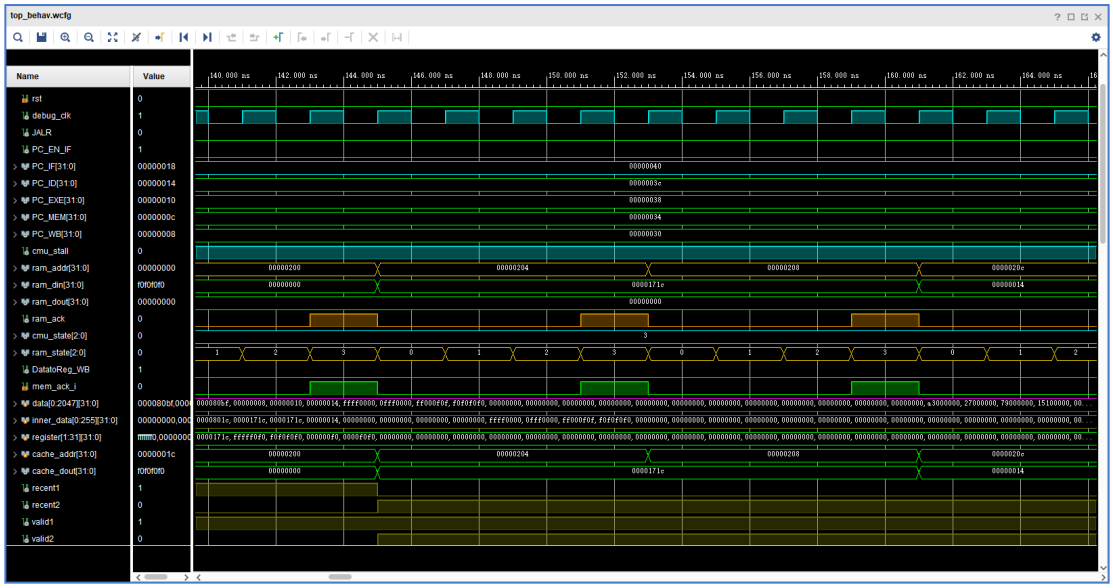

top_behav.wcfg
47.749 ns
Name
Value
rst
debug_clk
JALR
PC_EN_IF
PC_IF[31:0]
PC_ID[31:0]
PC_EXE[31:0]
PC_MEM[31:0]
PC_WB[31:0]
cmu_stall
ram_addr[31:0]
ram_din[31:0]
ram_dout[31:0]
ram_ack
cmu_state[2:0]
ram_state[2:0]
DatatoReg_WB
mem_ack_i
data[0:2047][31:0]
inner_data[0:255][31:0]
register[1:31][31:0]
cache_addr[31:0]
cache_dout[31:0]
recent1
recent2
valid1
valid2

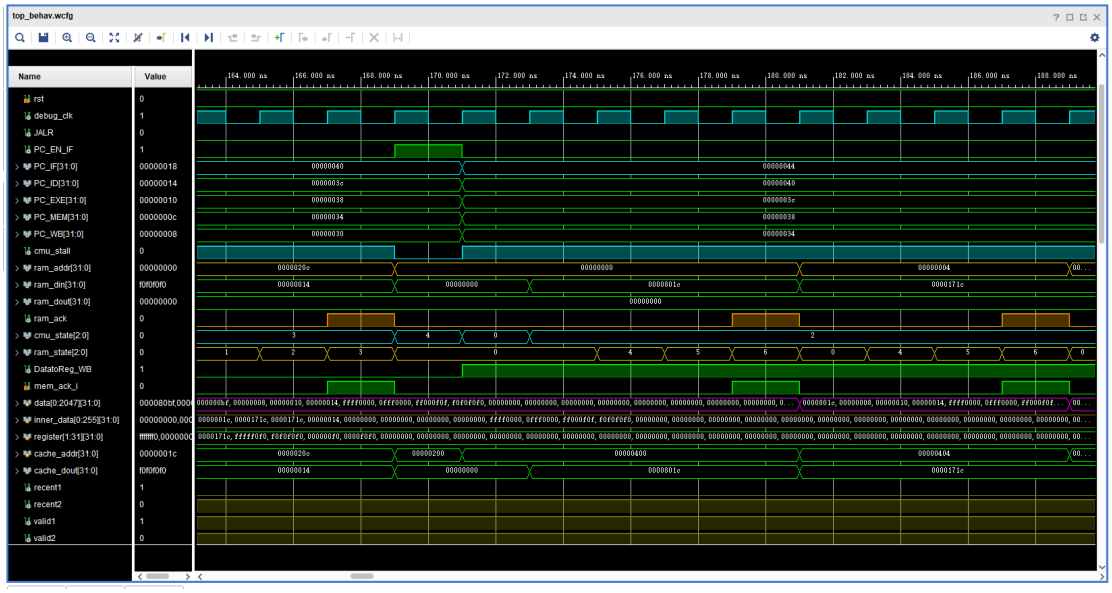

top_behav.wcfg
47.749 ns
Name
Value
rst
debug_clk
JALR
PC_EN_IF
PC_IF[31:0]
PC_ID[31:0]
PC_EXE[31:0]
PC_MEM[31:0]
PC_WB[31:0]
cmu_stall
ram_addr[31:0]
ram_din[31:0]
ram_dout[31:0]
ram_ack
cmu_state[2:0]
ram_state[2:0]
DatatoReg_WB
mem_ack_i
data[0:2047][31:0]
inner_data[0:255][31:0]
register[1:31][31:0]
cache_addr[31:0]
cache_dout[31:0]
recent1
recent2
valid1
valid2

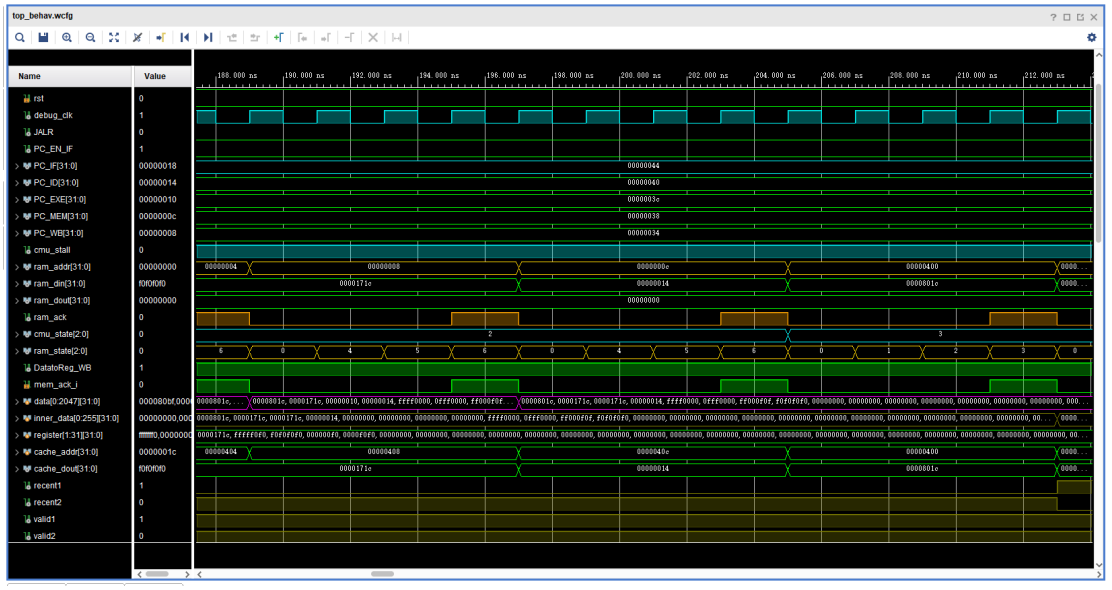

top_behav.wcfg
Name
Value
rst
debug_clk
JALR
PC_EN_IF
PC_IF[31:0]
PC_ID[31:0]
PC_EXE[31:0]
PC_MEM[31:0]
PC_WB[31:0]
cmu_stall
ram_addr[31:0]
ram_din[31:0]
ram_dout[31:0]
ram_ack
cmu_state[2:0]
ram_state[2:0]
DatatoReg_WB
mem_ack_i
data[0:2047][31:0]
inner_data[0:255][31:0]
register[1:31][31:0]
cache_addr[31:0]
cache_dout[31:0]
recent1
recent2
valid1
valid2

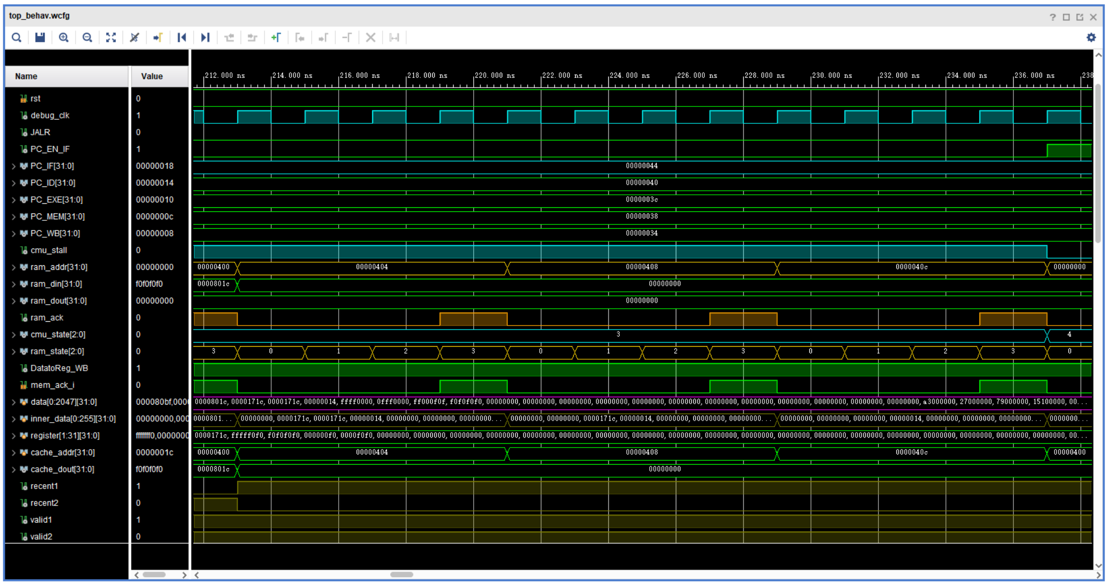

top_behav.wcfg
Name
Value
rst
debug_clk
JALR
PC_EN_IF
PC_IF[31:0]
PC_ID[31:0]
PC_EXE[31:0]
PC_MEM[31:0]
PC_WB[31:0]
cmu_stall
ram_addr[31:0]
ram_din[31:0]
ram_dout[31:0]
ram_ack
cmu_state[2:0]
ram_state[2:0]
DatatoReg_WB
mem_ack_i
data[0:2047][31:0]
inner_data[0:255][31:0]
register[1:31][31:0]
cache_addr[31:0]
cache_dout[31:0]
recent1
recent2
valid1
valid2

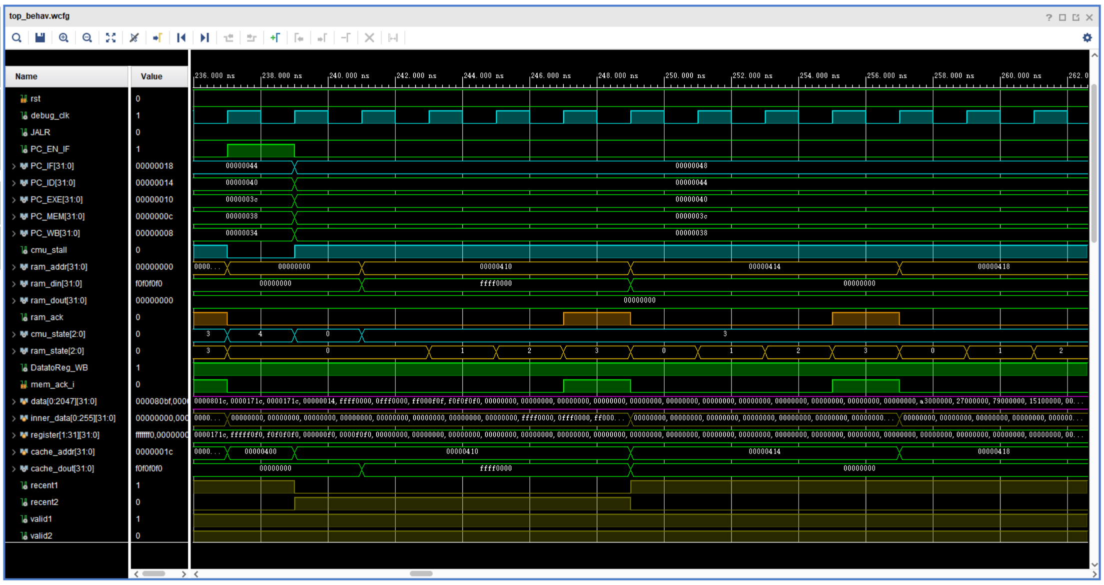

top_behav.wcfg
Name
Value
rst
debug_clk
JALR
PC_EN_IF
PC_IF[31:0]
PC_ID[31:0]
PC_EXE[31:0]
PC_MEM[31:0]
PC_WB[31:0]
cmu_stall
ram_addr[31:0]
ram_din[31:0]
ram_dout[31:0]
ram_ack
cmu_state[2:0]
ram_state[2:0]
DatatoReg_WB
mem_ack_i
data[0:2047][31:0]
inner_data[0:255][31:0]
register[1:31][31:0]
cache_addr[31:0]
cache_dout[31:0]
recent1
recent2
valid1
valid2

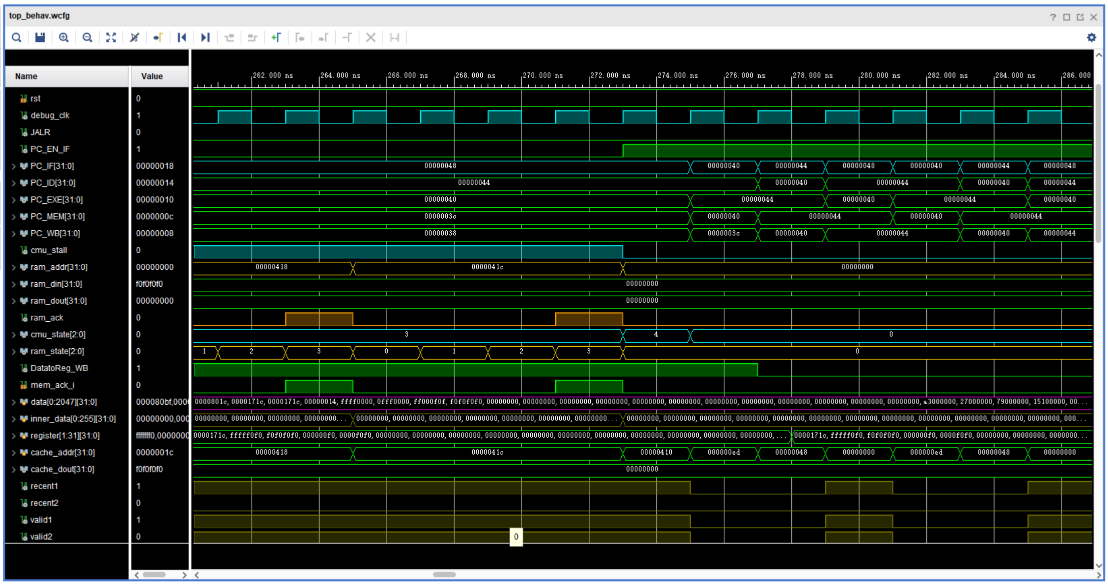

top_behav.wcfg
Name
Value
rst
debug_clk
JALR
PC_EN_IF
PC_IF[31:0]
PC_ID[31:0]
PC_EXE[31:0]
PC_MEM[31:0]
PC_WB[31:0]
cmu_stall
ram_addr[31:0]
ram_din[31:0]
ram_dout[31:0]
ram_ack
cmu_state[2:0]
ram_state[2:0]
DataToReg_WB
mem_ack_i
data[0:2047][31:0]
inner_data[0:255][31:0]
register[1:31][31:0]
cache_addr[31:0]
cache_dout[31:0]
recent1
recent2
valid1
valid2


top_behav.wcfg
Name
Value
rst
debug_clk
JALR
PC_EN_IF
PC_IF[31:0]
PC_ID[31:0]
PC_EXE[31:0]
PC_MEM[31:0]
PC_WB[31:0]
cmu_stall
ram_addr[31:0]
ram_din[31:0]
ram_dout[31:0]
ram_ack
cmu_state[2:0]
ram_state[2:0]
DataToReg_WB
mem_ack_i
data[0:2047][31:0]
inner_data[0:255][31:0]
register[1:31][31:0]
cache_addr[31:0]
cache_dout[31:0]
recent1
recent2
valid1
valid2


top_behav.wcfg
Name
Value
rst
debug_clk
JALR
PC_EN_IF
PC_IF[31:0]
PC_ID[31:0]
PC_EXE[31:0]
PC_MEM[31:0]
PC_WB[31:0]
cmu_stall
ram_addr[31:0]
ram_din[31:0]
ram_dout[31:0]
ram_ack
cmu_state[2:0]
ram_state[2:0]
DataToReg_WB
mem_ack_i
data[0:2047][31:0]
inner_data[0:255][31:0]
register[1:31][31:0]
cache_addr[31:0]
cache_dout[31:0]
recent1
recent2
valid1
valid2

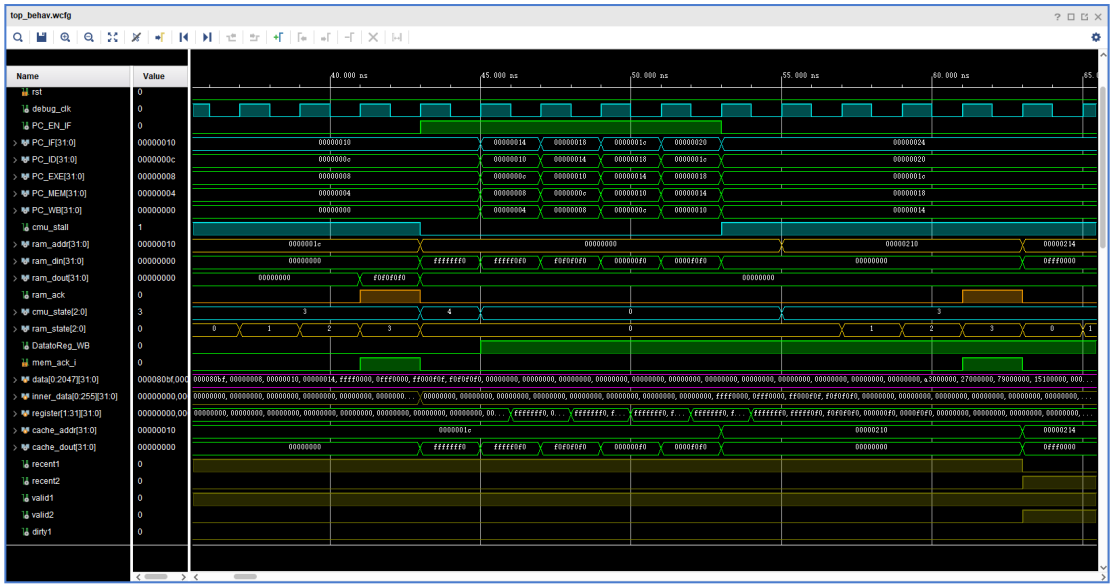






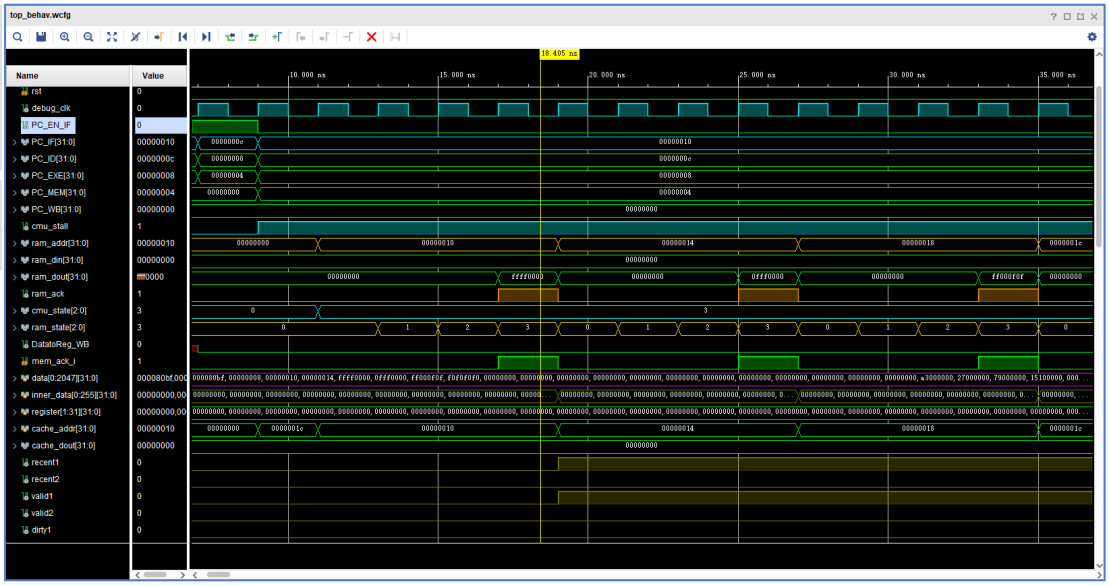
Task 8：请给出一个某条指令 load/store hit 部分仿真的高清图片，并对涉及

到的信号加以详细解释（5 points)

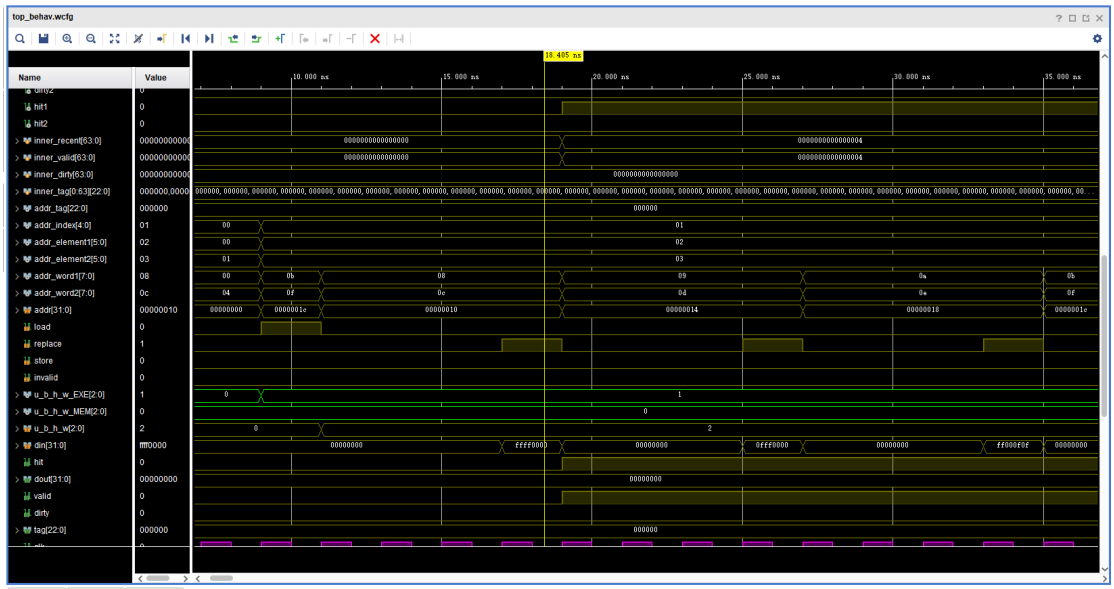


45000ns 时，PC_MEM=0x08，表示当前 lh x2, 0x01C(x0)指令进入访存阶段，由于上一条指令中，FFFFF0F0 已经被取到 cache 中，故此时发生 cache hit，可以看到当前指令没有发生 stall(cmu_stall=0).

Task 9: 请给出一个某条指令 load/store miss 导致 replace 到 invalid line 部分仿真的高清图片，并对涉及到的信号加以详细解释（10 points)

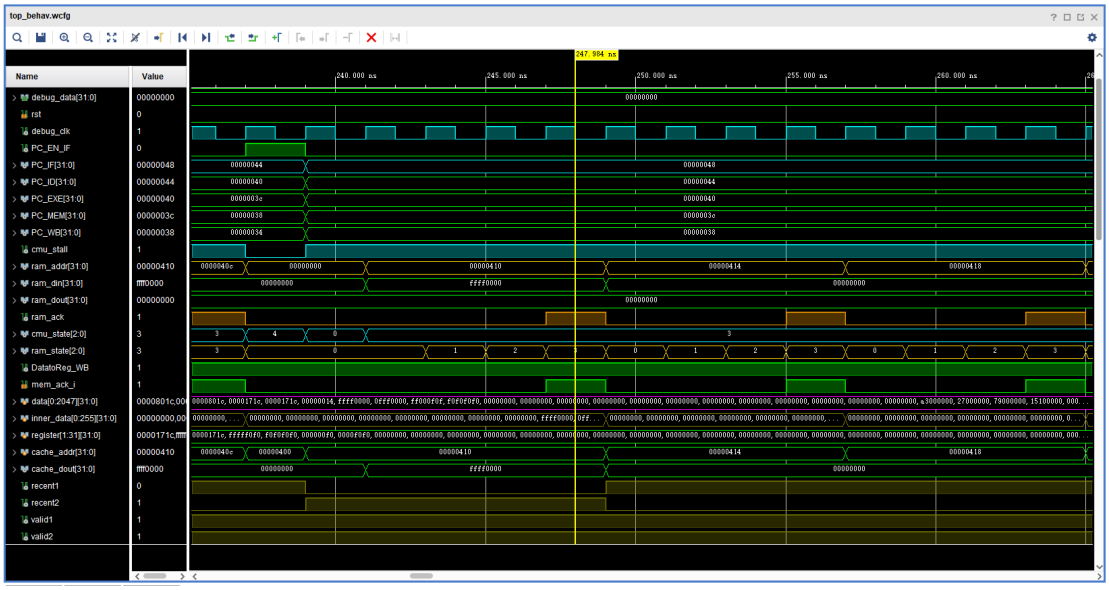


Cache 部分信号：

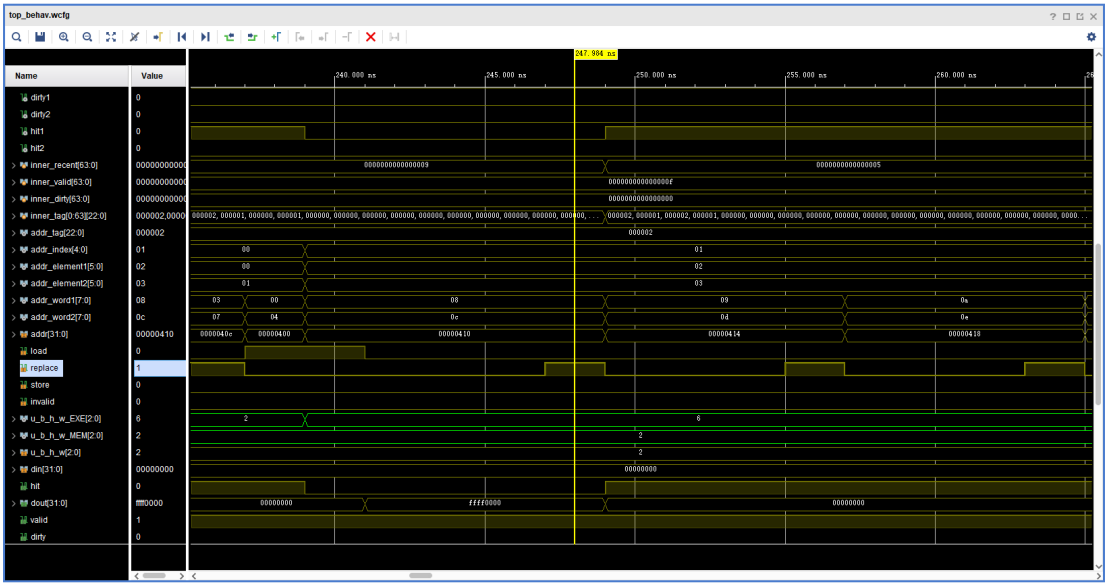


9000ns 时，PC=0x4 指令(lb x1,0x01c(x0))进入访存阶段，由于当前 cache 中没有任何数据，发生 cache miss(hit=0)，故将当前流水线 stall，cmu_stall 信号拉高，且当前并没有 valid 的 cache line(valid=0)，故需要从内存中取数据到 cache，ram 状态机运行 123 状态，进行 4 个 word 的读取，进行 cache replace。

Task 10: 请给出一个某条指令 load/store miss 导致 replace 到 valid and clean line 部分仿真的高清图片，并对涉及到的信号加以详细解释 (10 points)



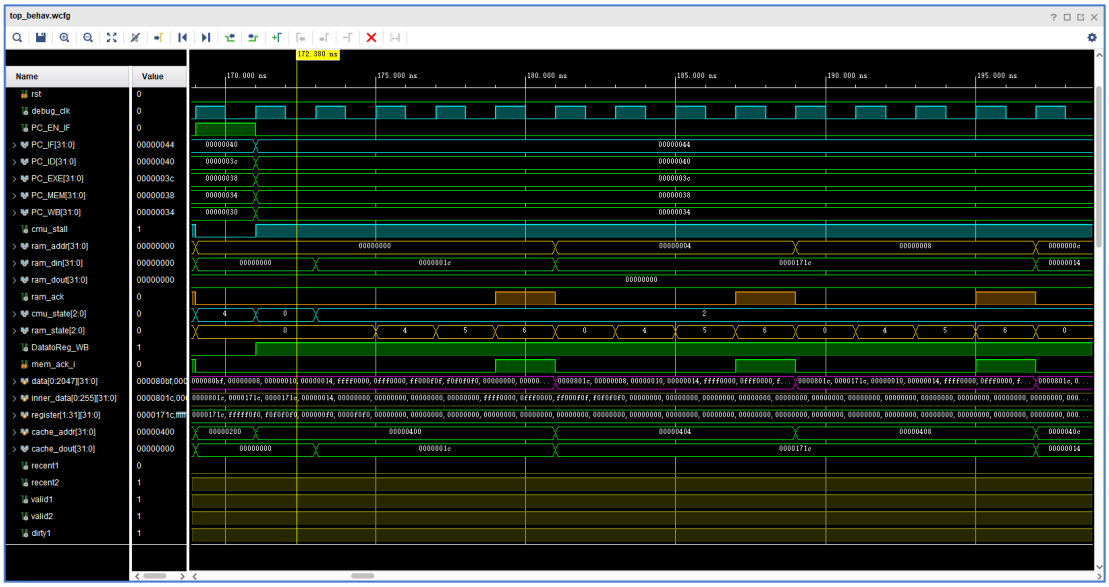
Cache 部分信号：

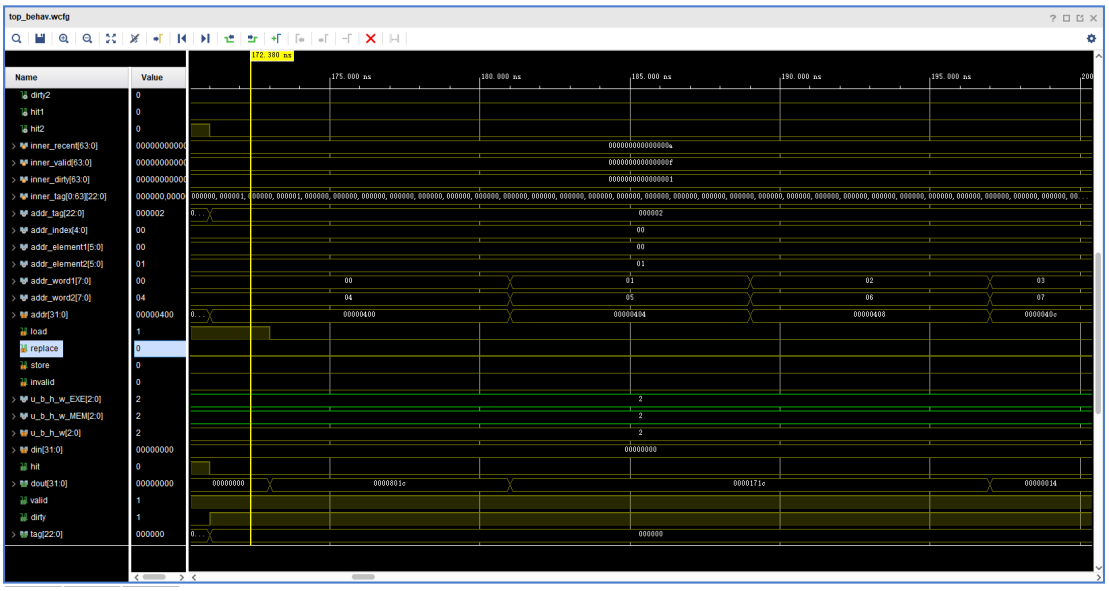


23900ns 时，PC=0x3c 指令进入访存阶段，发生 cache miss(hit=0)，从 cache 信号中可以看到此时 cache valid(valid=1)且 clean(dirty=0),无需向内存 write back，直接进行 cache replace 即可，ram 状态机运行 123 状态，在 24700ns 时，ram_state=3，从 ram 中取数据到 cache 中。

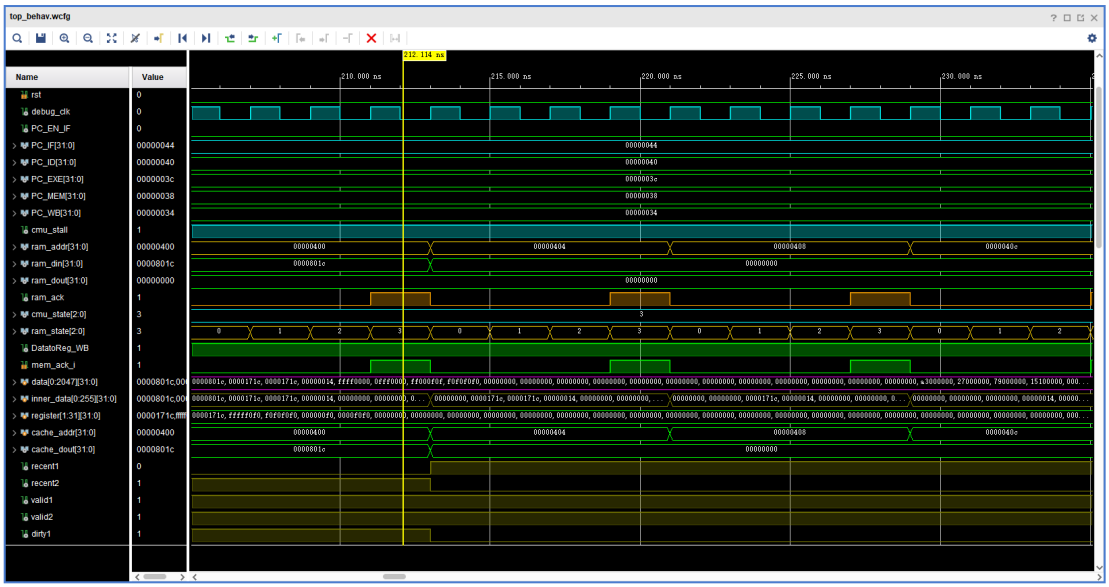
Task 11: 请给出一个某条指令 load/store miss 导致 replace 到 valid and dirty line 部分仿真的高清图片，并对涉及到的信号加以详细解释 (15 points)

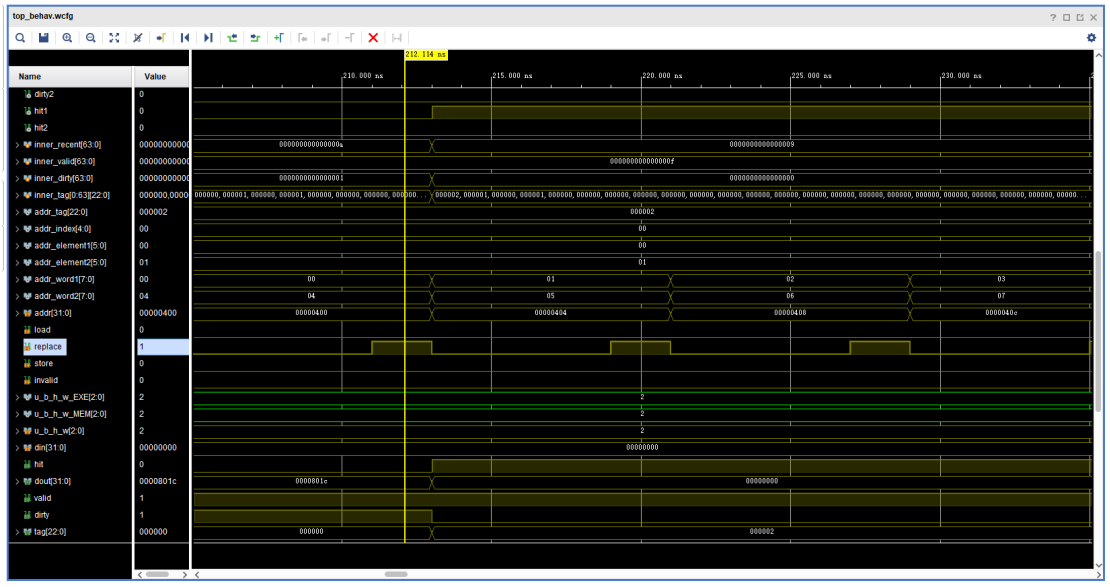
Load miss，dirty，向内存写回：





Cache replace：





17100ns 时，PC=0x38 指令进入访存阶段，此时发生 load miss，可以从对应的 cache 信号中看到，cache 中 load=1 时，hit=0，表明 load 未命中，而此时 cache 中 valid=1，dirty=1（cache line 里已经有数据），那么要先将 cache 中的数据写回内存后再将新的数据加载到 cache 中，故可以看到，17100ns 开始，ram 模块的状态机运行 456 状态，将整个 cache line 写回。在 20500ns 时，写回结束(ram_state=0)，开始 cache line 的 replace。

四、 讨论与心得

Task 12: 请写出对本次实验内容的深入讨论，或者本次实验的心得体会。例如遇到的难题等等。请认真填写本模块，若不填写或胡乱填写将酌情扣分，写明白真实情况即可。 (+10 points)

在解决 inferring latch 的问题之后，上板仍然出错，且 implementation 十分缓慢，在通过询问助教、与同学讨论后仍然无果，最后发现是自己没有关增量综合，猜测是 vivado 没有进行完整的重新综合导致上述问题的出现，关掉增量综合后，implementation 的速度大大加快，进入十分钟以内，且上板结果与仿真预期结果一致。