Arch Lab4

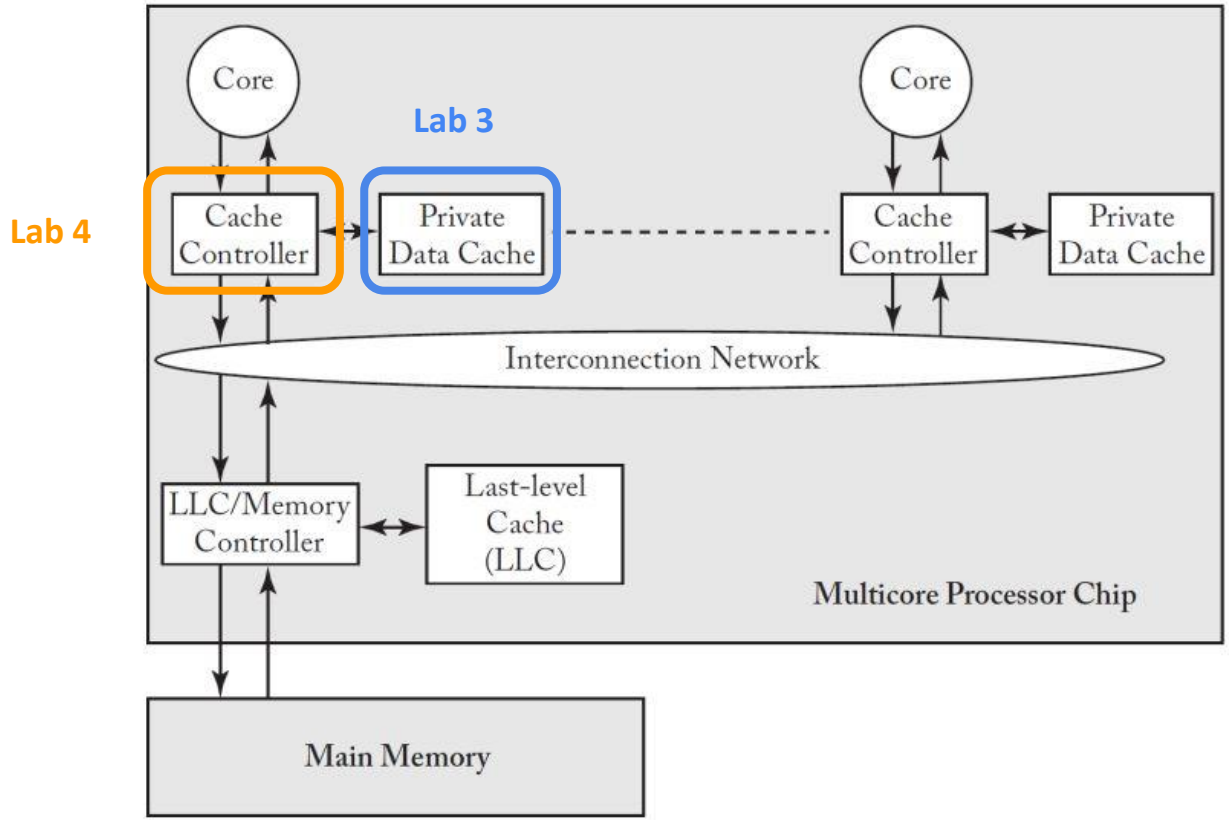
Pipelined CPU with Cache

Chenlu Miao 11/2022

Overview

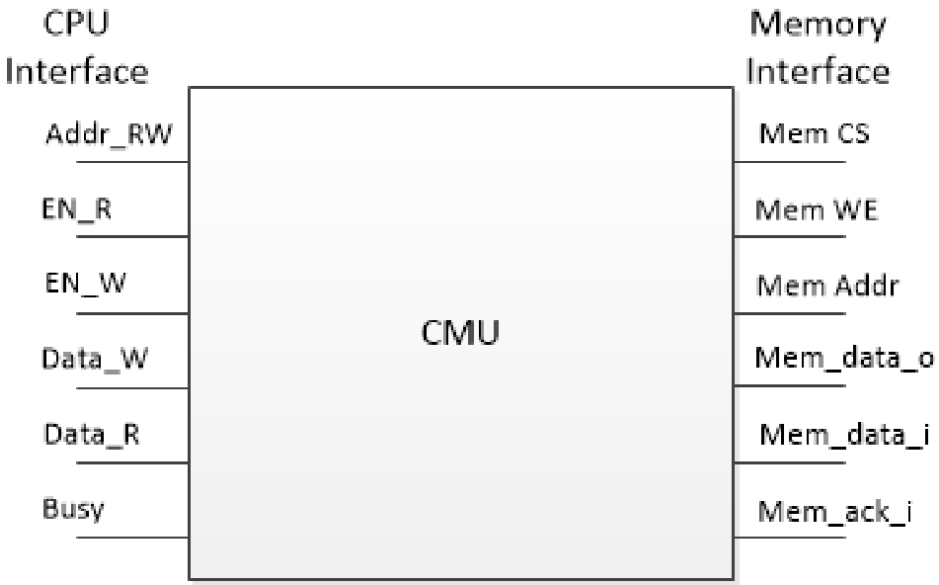
- Cache Overview
- Cache Controller Interfaces
- Cache Controller FSM
- Code: Cache Controller
- Simulation

Cache Overview



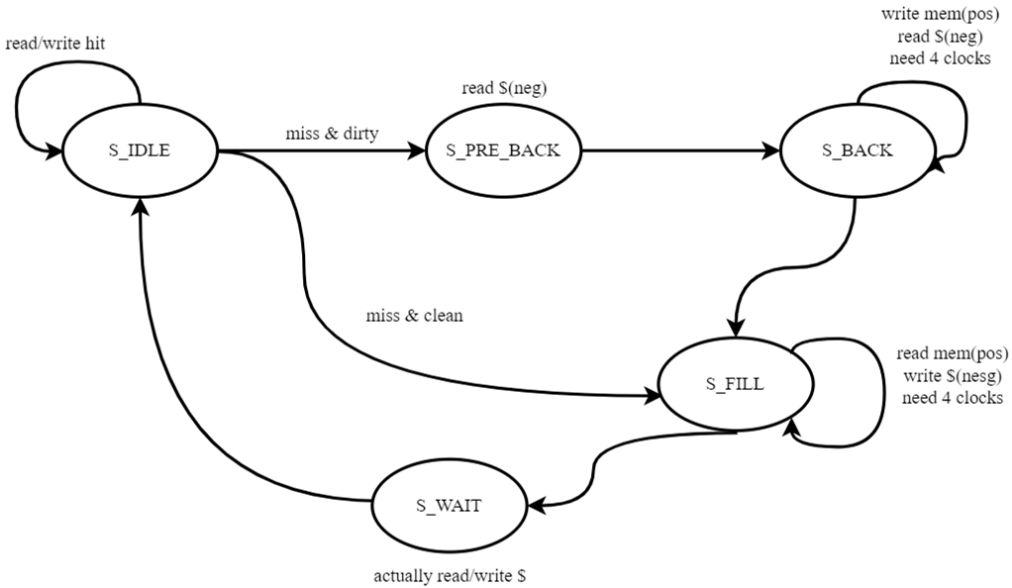
Cache Controller Interfaces

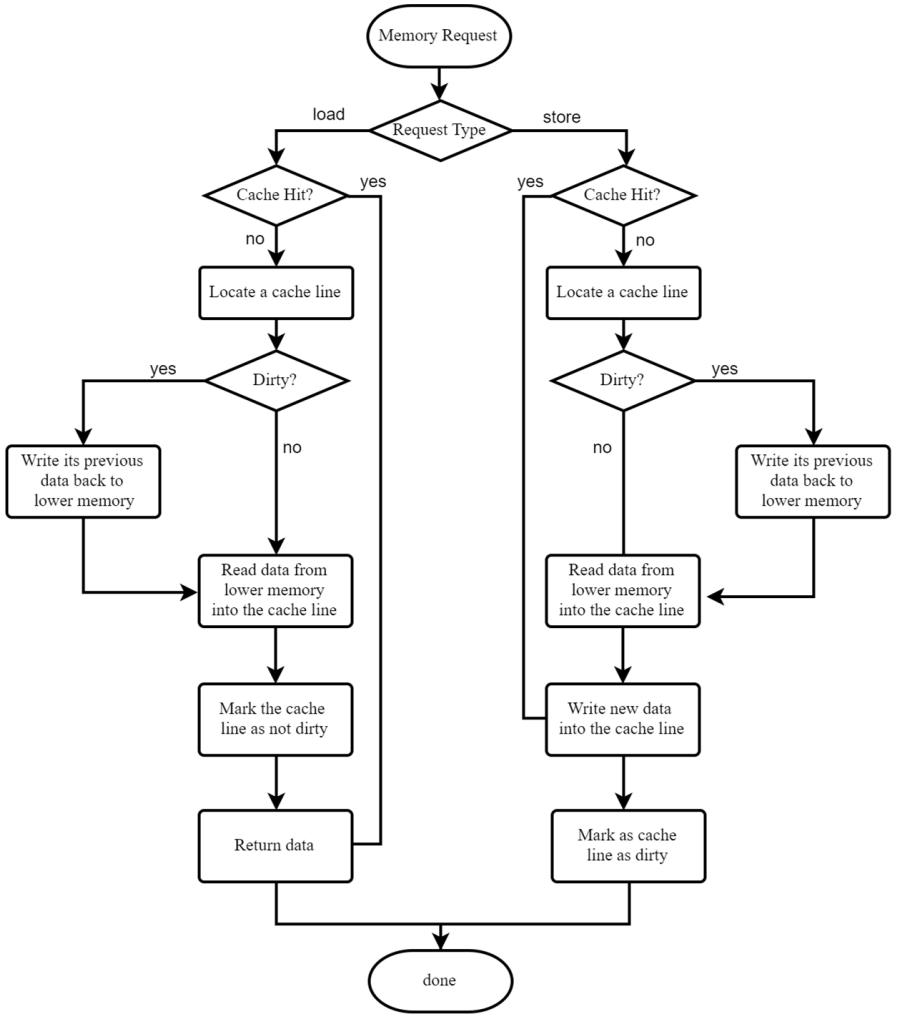
```
cmu CMU(.clk(debug_clk),.rst(rst),.addr_rw(ALUout_MEM),
    .en_r(DatatoReg_MEM),.en_w(mem_w_MEM),.u_b_h_w(u_b_h_w_MEM),
    .data_w(Dataout_MEM),.data_r(Datain_MEM),.stall(cmu_stall),
    .mem_cs_o(ram_cs),.mem_we_o(ram_we),.mem_addr_o(ram_addr),
     .mem_data_i(ram_dout),.mem_data_o(ram_din),.mem_ack_i(ram_ack
),.cmu_state(cmu_state));
```



lb/lh/lw/lbu…?

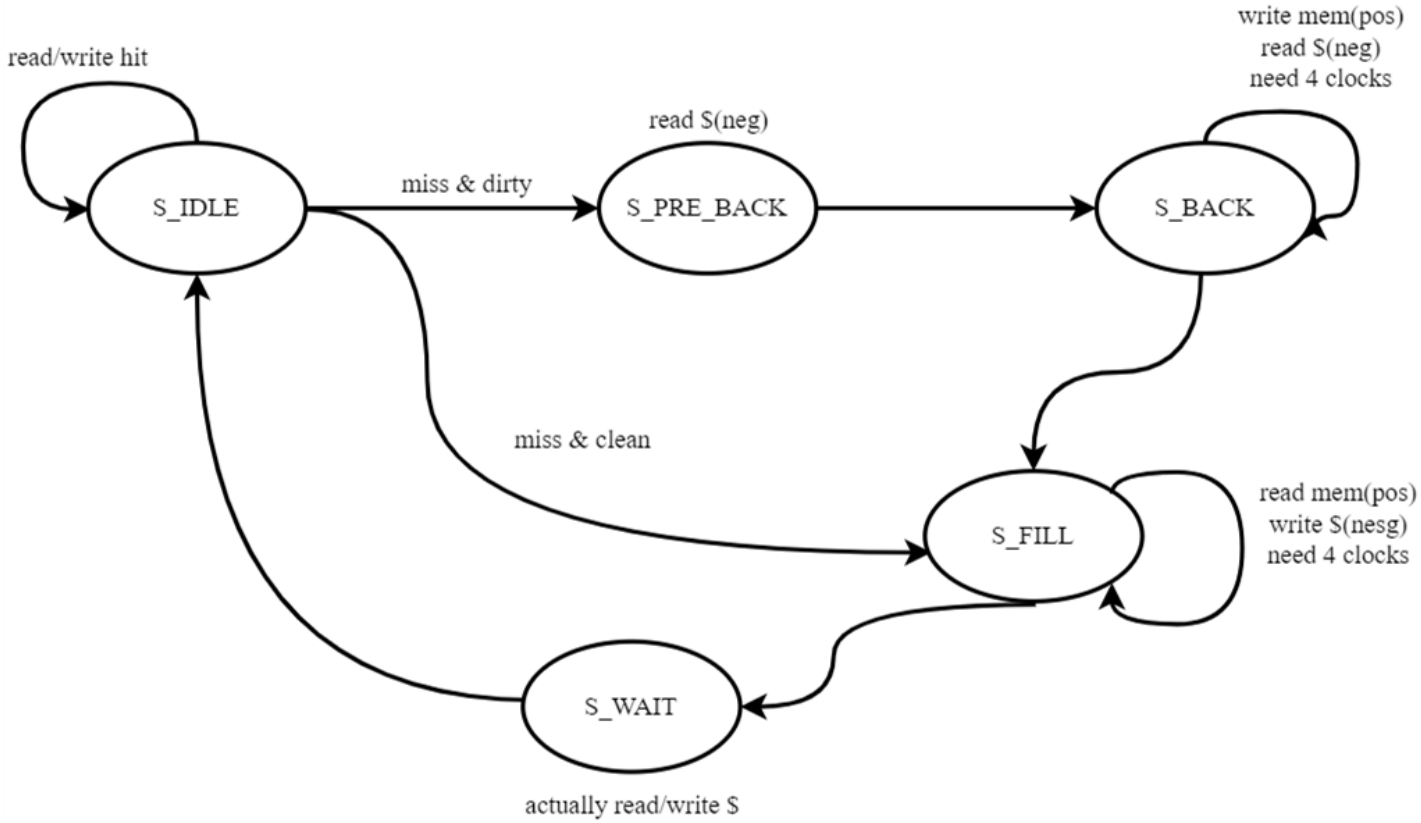
Cache Controller FSM



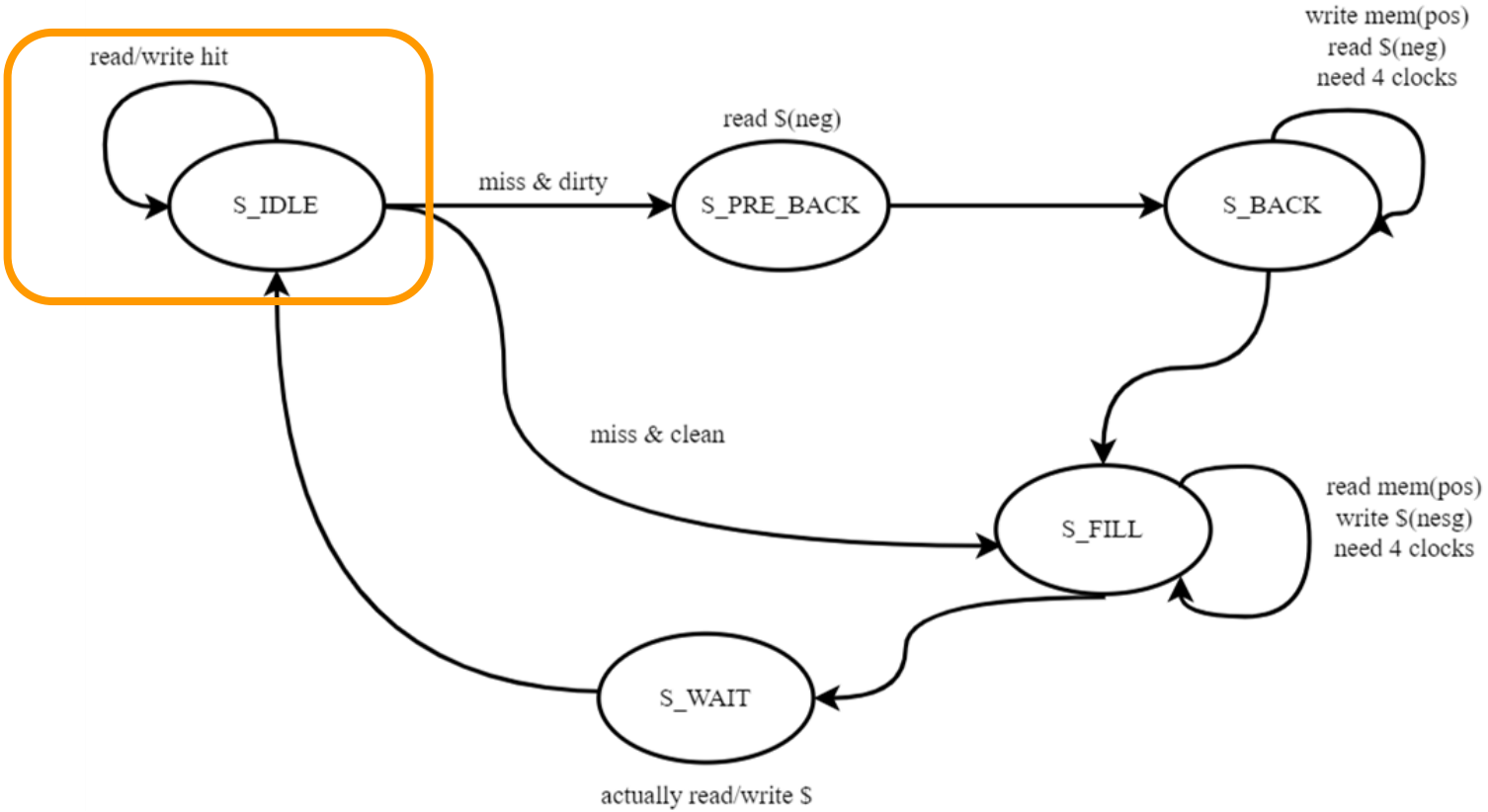


Cache Controller FSM

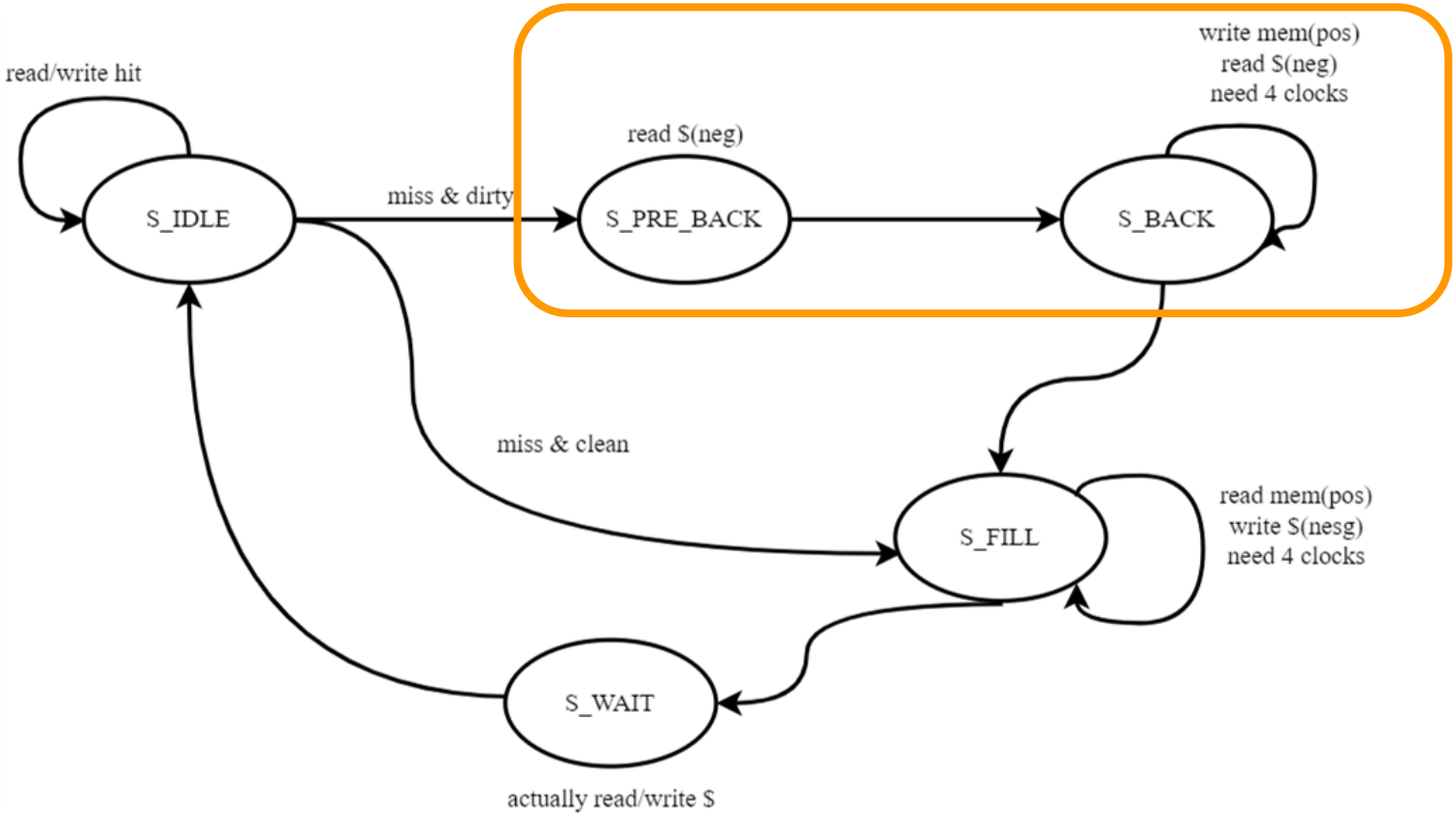
- S_IDLE
- S_PRE_BACK
- S_BACK
- S_FILL
- S_WAIT



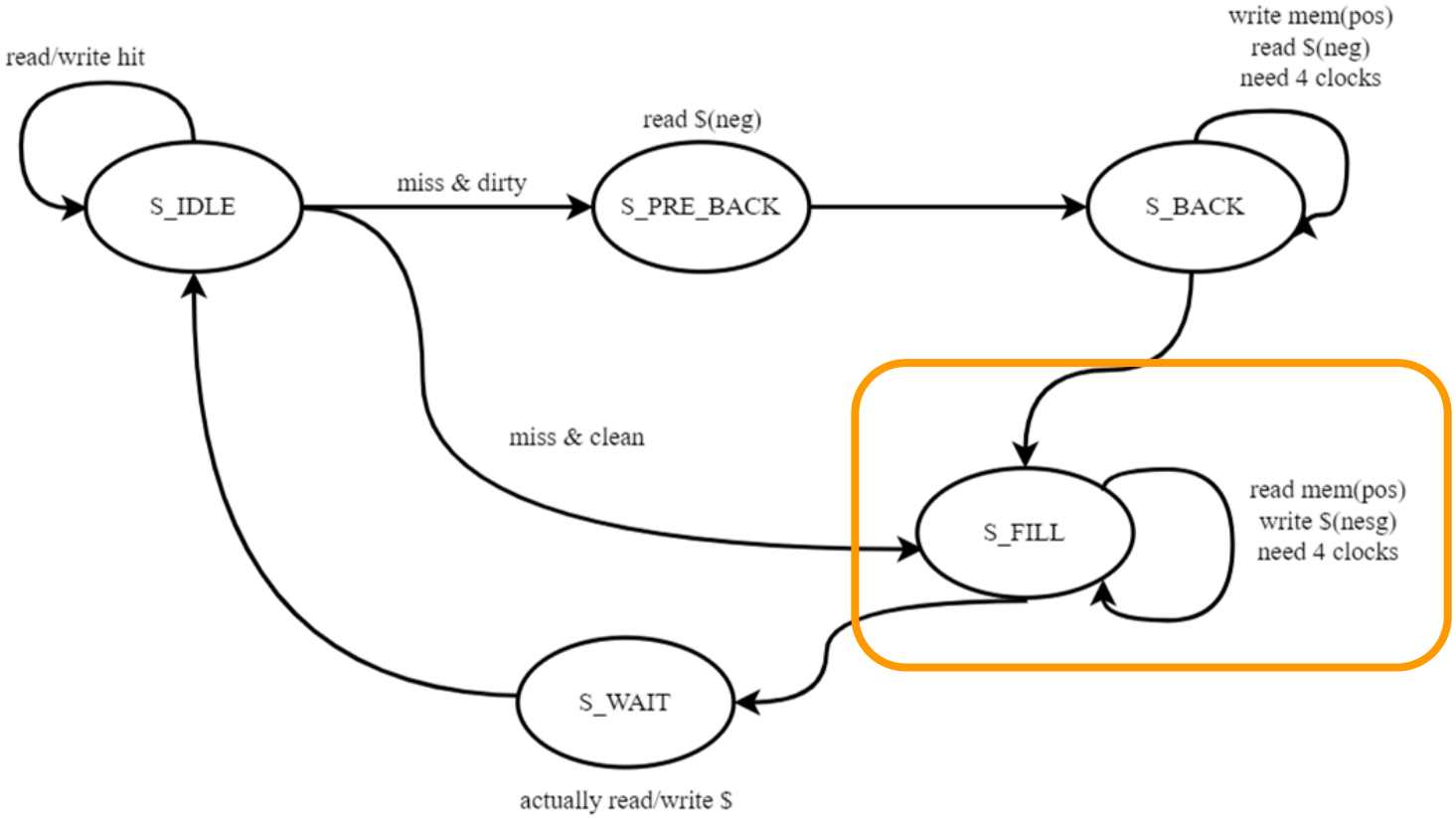
Cache Controller FSM – Hit



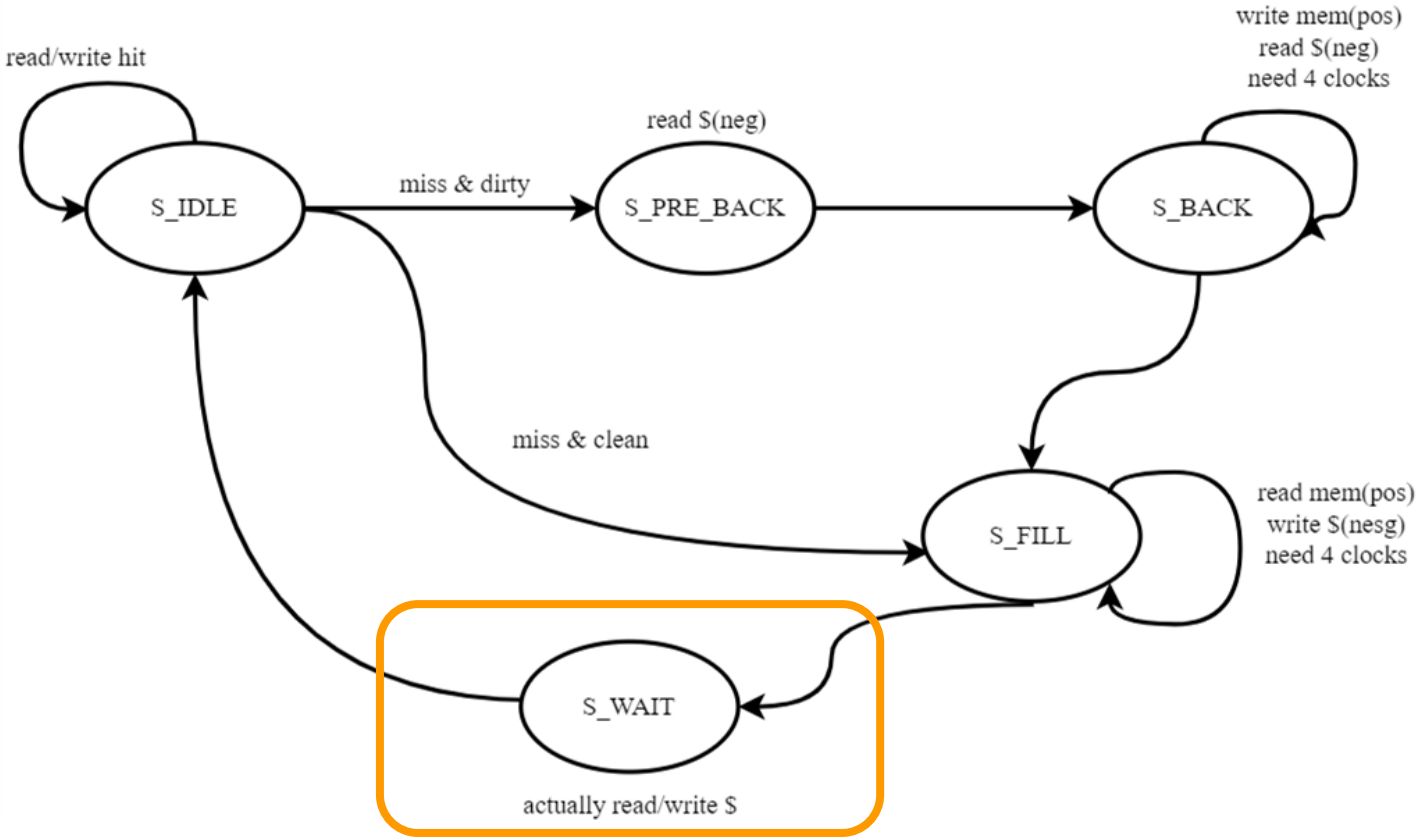
Cache Controller FSM – Miss & Dirty



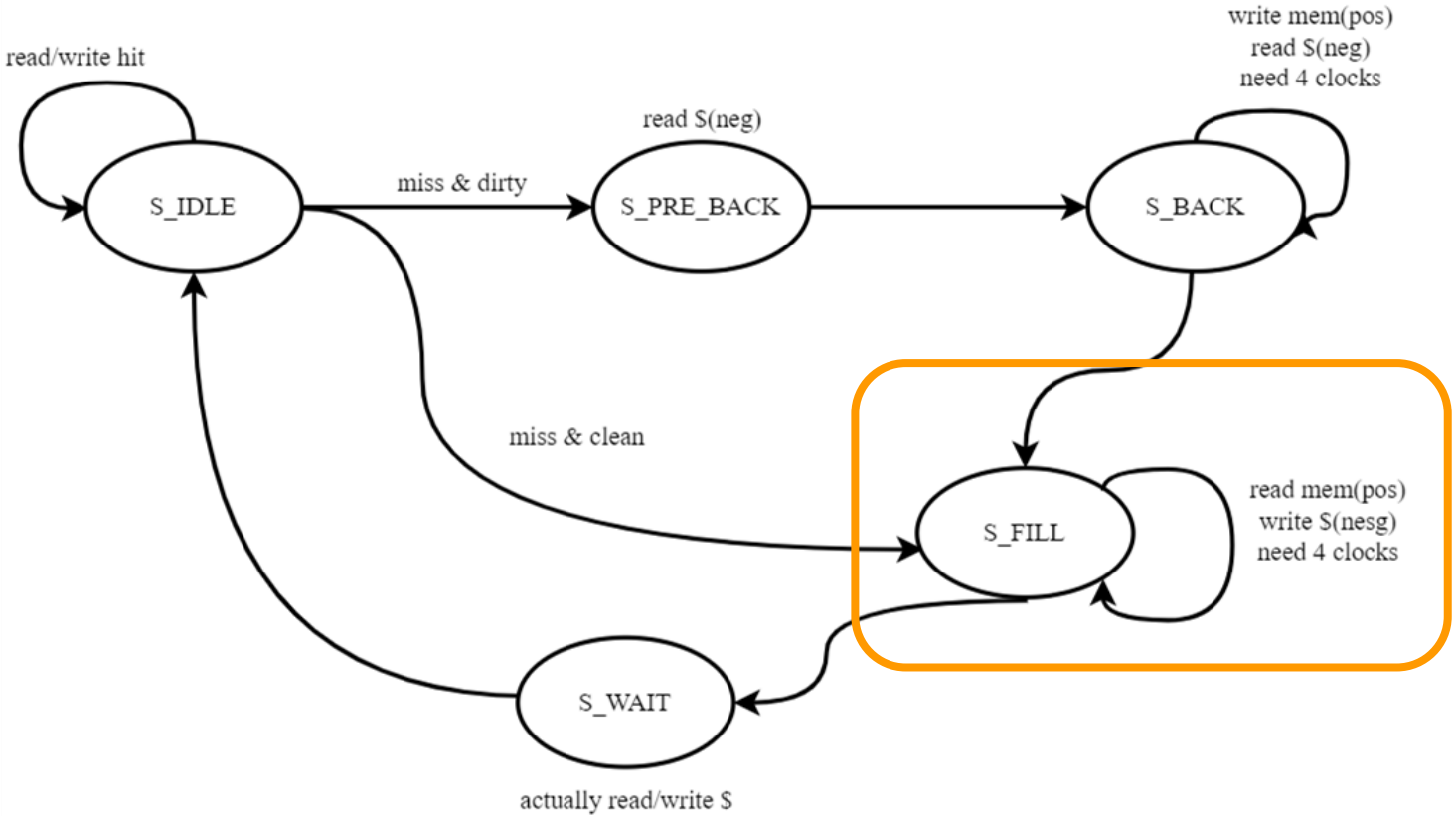
Cache Controller FSM – Miss & Dirty



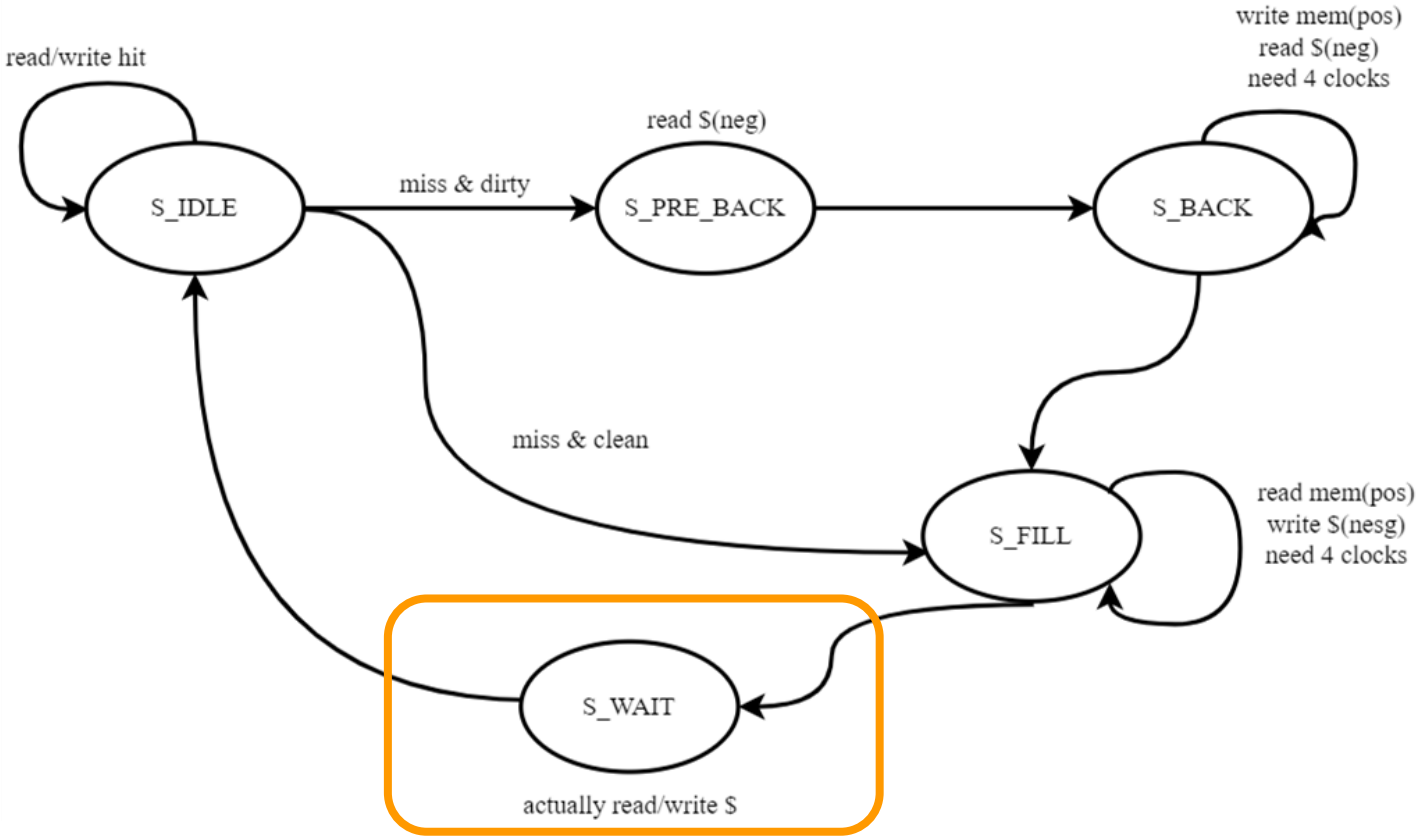
Cache Controller FSM – Miss & Dirty



Cache Controller FSM – Miss & Clean



Cache Controller FSM – Miss & Clean



Code: Cache Controller

```
localparam
    S_IDLE = 0,
    S_PRE_BACK = 1,
    S_BACK = 2,
    S_FILL = 3,
    S_WAIT = 4;

reg [2:0]state = 0;
reg [2:0]next_state = 0;
reg [ELEMENT_WORDS_WIDTH-1:0]word_count = 0;
reg [ELEMENT_WORDS_WIDTH-1:0]next_word_count = 0;
```

```
always @ (posedge clk) begin
    if (rst) begin
        state <= S_IDLE;
        word_count <= 2'b00;
    end
    else begin
        state <= next_state;
        word_count <= next_word_count;
    end
end
```

Code: Cache Controller

```
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 = ??;
                    else if (cache_valid && cache_dirty)
                        next_state = ??;
                    else
                        next_state = ??;
                end
                next_word_count = 2'b00;
            end
```

```
S_PRE_BACK: begin
    next_state = ??;
    next_word_count = 2'b00;
end

S_BACK: begin
    if (mem_ack_i && word_count == ...)
        next_state = ??;
    else
        next_state = ??;

    if (mem_ack_i)
        next_word_count = ??;
    else
        next_word_count = word_count;
end
```

Code: Cache Controller

```
S_FILL: begin
    if (mem_ack_i && word_count == ...)
        next_state = ??;
    else
        next_state = ??;

    if (mem_ack_i)
        next_word_count = ??;
    else
        next_word_count = word_count;
end

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

Code: Cache Controller

```
always @ (*) begin
    case(state)
        S_IDLE, S_WAIT: begin
            cache_addr = addr_rw;
            cache_load = en_r;
            cache_edit = en_w;
            cache_store = 1'b0;
            cache_u_b_h_w = u_b_h_w;
            cache_din = data_w;
        end
        S_BACK, S_PRE_BACK: begin
            cache_addr = ...
            cache_load = 1'b0;
            cache_edit = 1'b0;
            cache_store = 1'b0;
            cache_u_b_h_w = 3'b010;
            cache_din = 32'b0;
        end
        S_FILL: begin
            cache_addr = ...
            cache_load = 1'b0;
            cache_edit = 1'b0;
            cache_store = mem_ack_i;
            cache_u_b_h_w = 3'b010;
            cache_din = mem_data_i;
        end
    endcase
end
assign data_r = cache_dout;
```

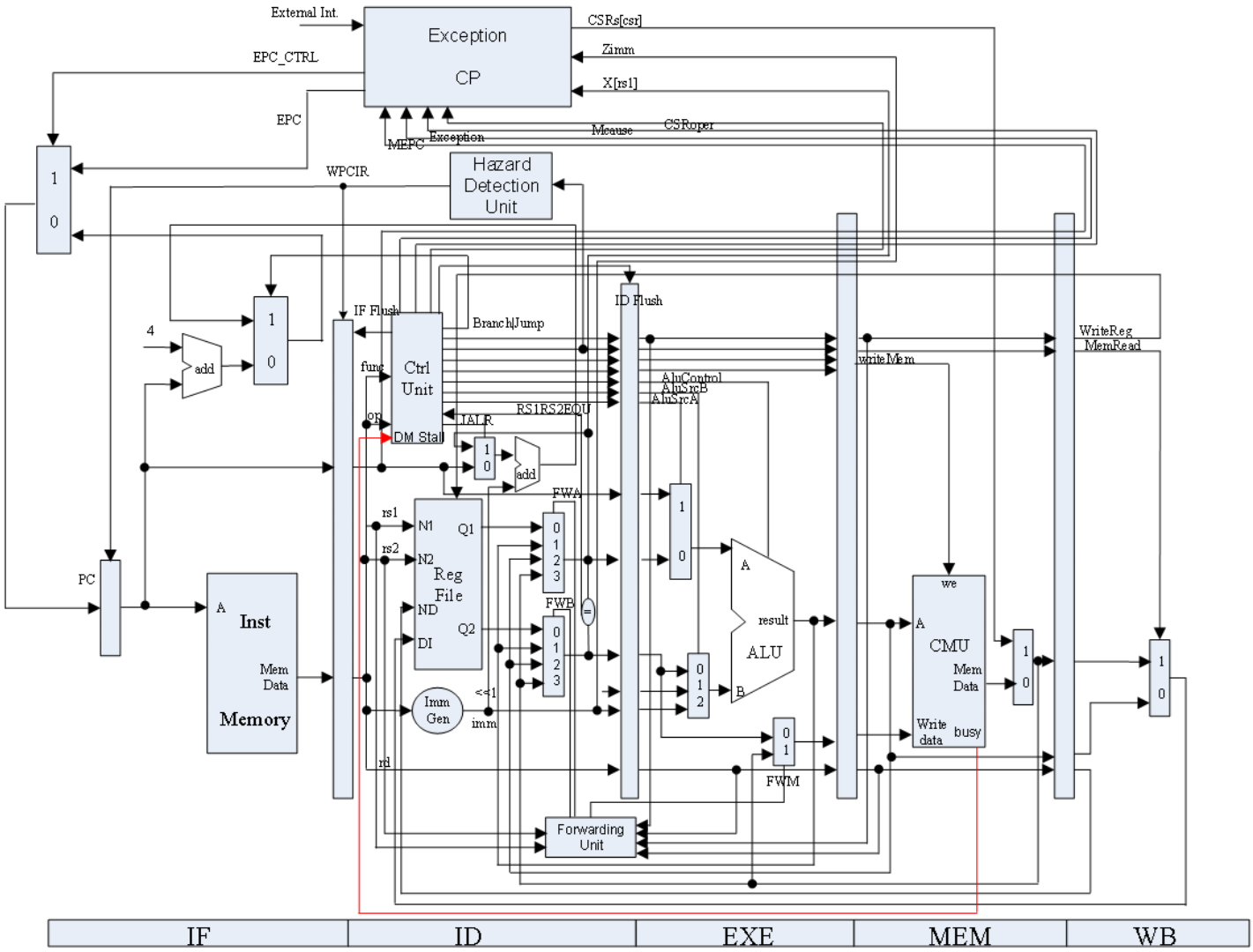
Code: Cache Controller

```
always @ (*) begin
    case (next_state)
        S_IDLE, S_PRE_BACK, 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 = ...
        end

        S_FILL: begin
            mem_cs_o = 1'b1;
            mem_we_o = 1'b0;
            mem_addr_o = ...
        end
    endcase
end
assign mem_data_o = cache_dout;
```

Pipelined CPU with Cache



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)	# 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		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	ffdff06f	40		jal x0, loop	

RAM

NO.	Data	Addr.
0	00000000	0
1	00000004	4
2	00000008	8
3	0000000C	C
4	00000010	10
5	00000014	14
6	00000018	18
7	0000001C	1C
8	00000020	20
9	00000024	24
10	00000028	28
11	0000002C	2C

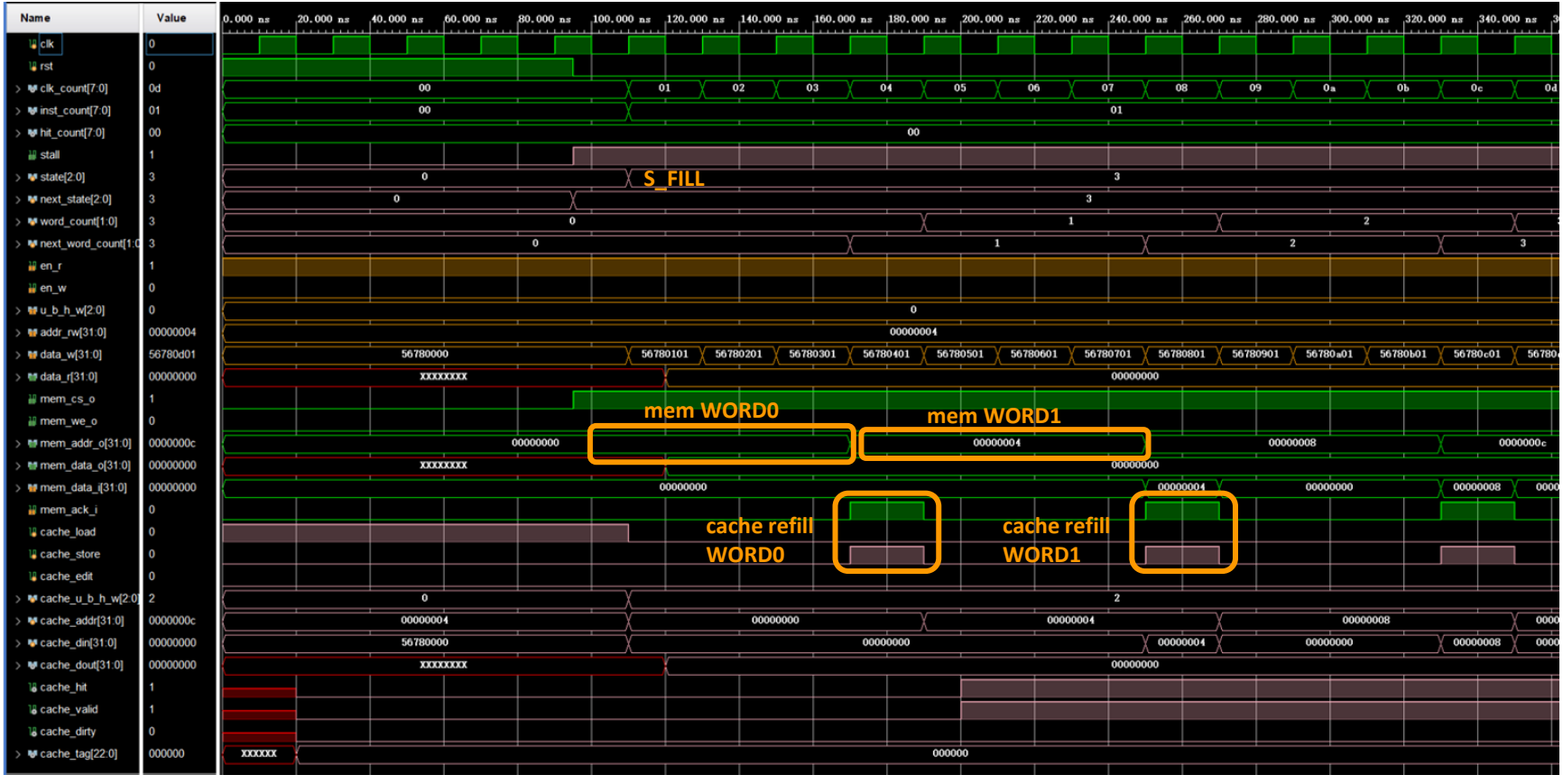
Simulation

inst.v

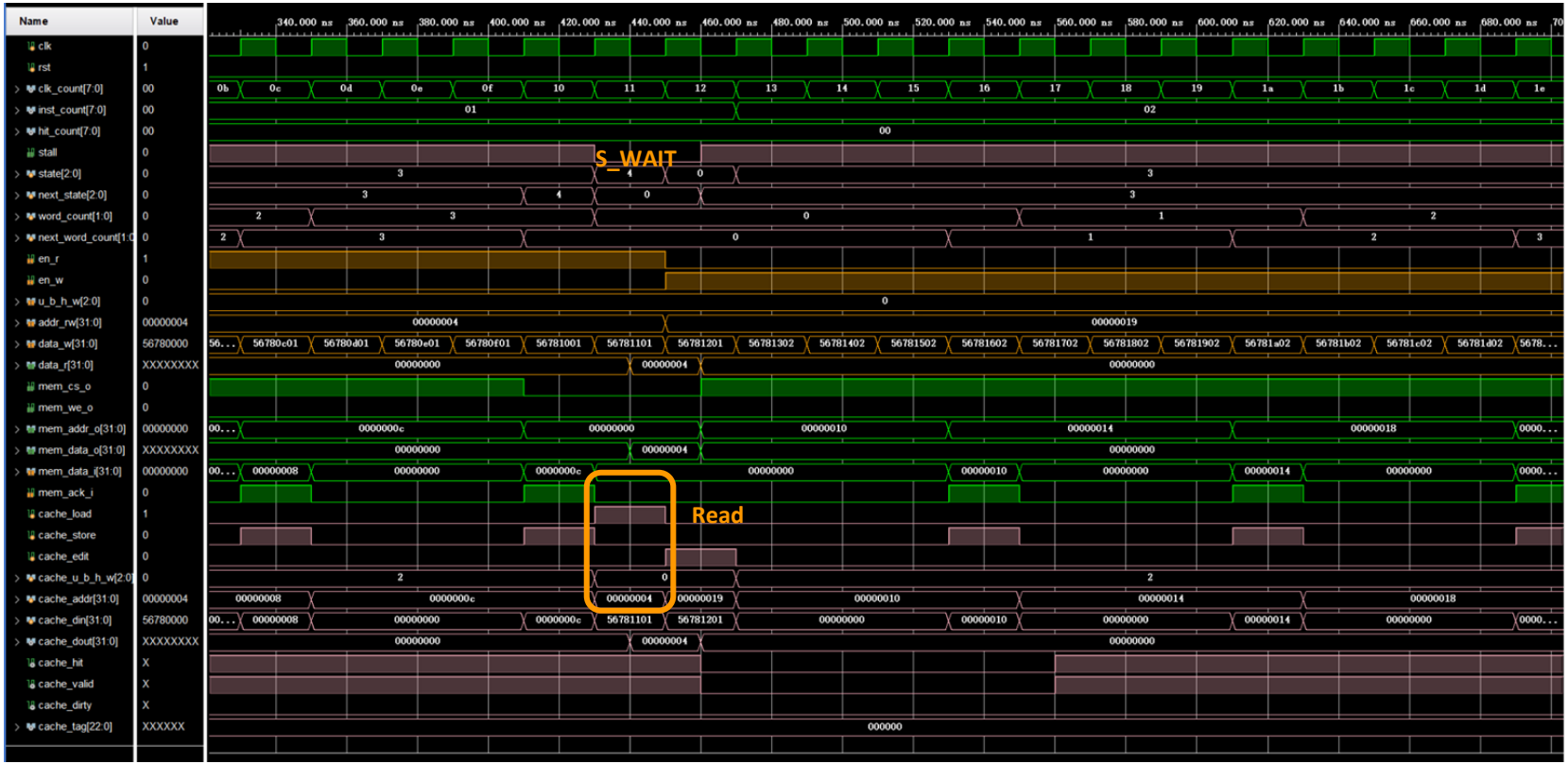
```
reg [39:0] data [0:9];
initial begin
    data[0] = 40'h0_2_00000004;  // read miss              1+17
    data[1] = 40'h0_3_00000019;  // write miss             1+17
    data[2] = 40'h1_2_00000008;  // read hit               1
    data[3] = 40'h1_3_00000014;  // write hit              1

    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];
```

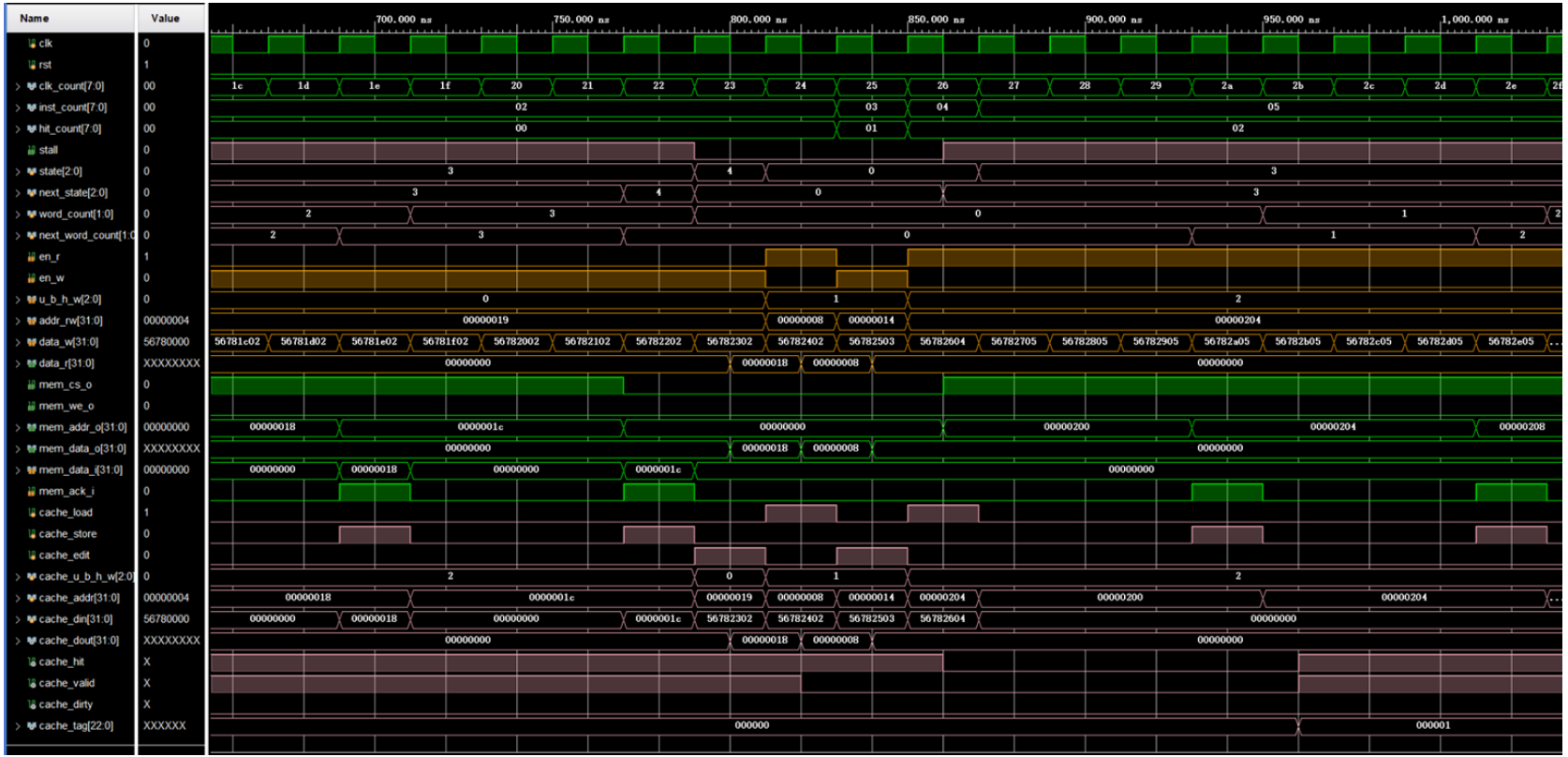
Simulation (1)



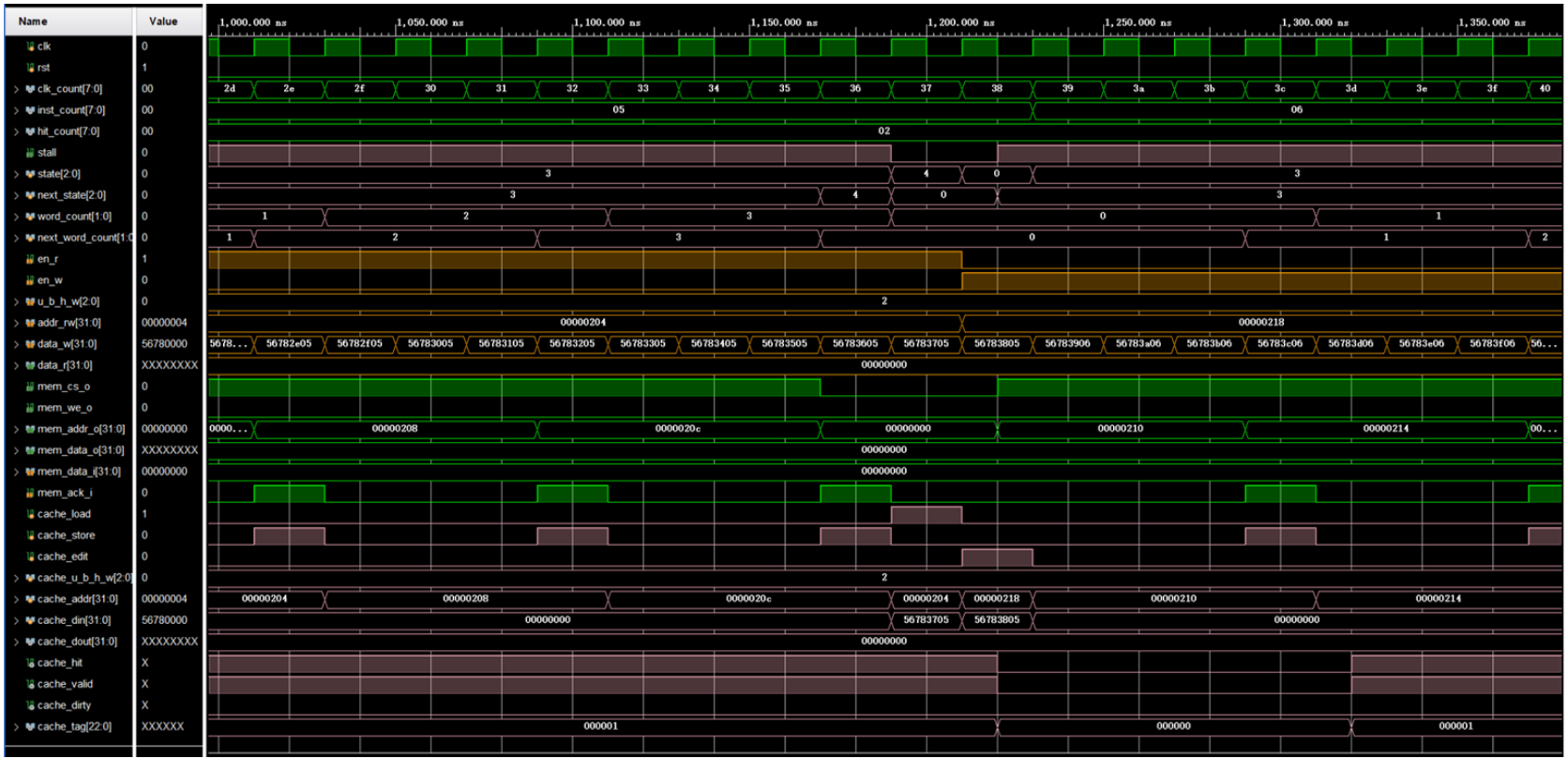
Simulation (2)



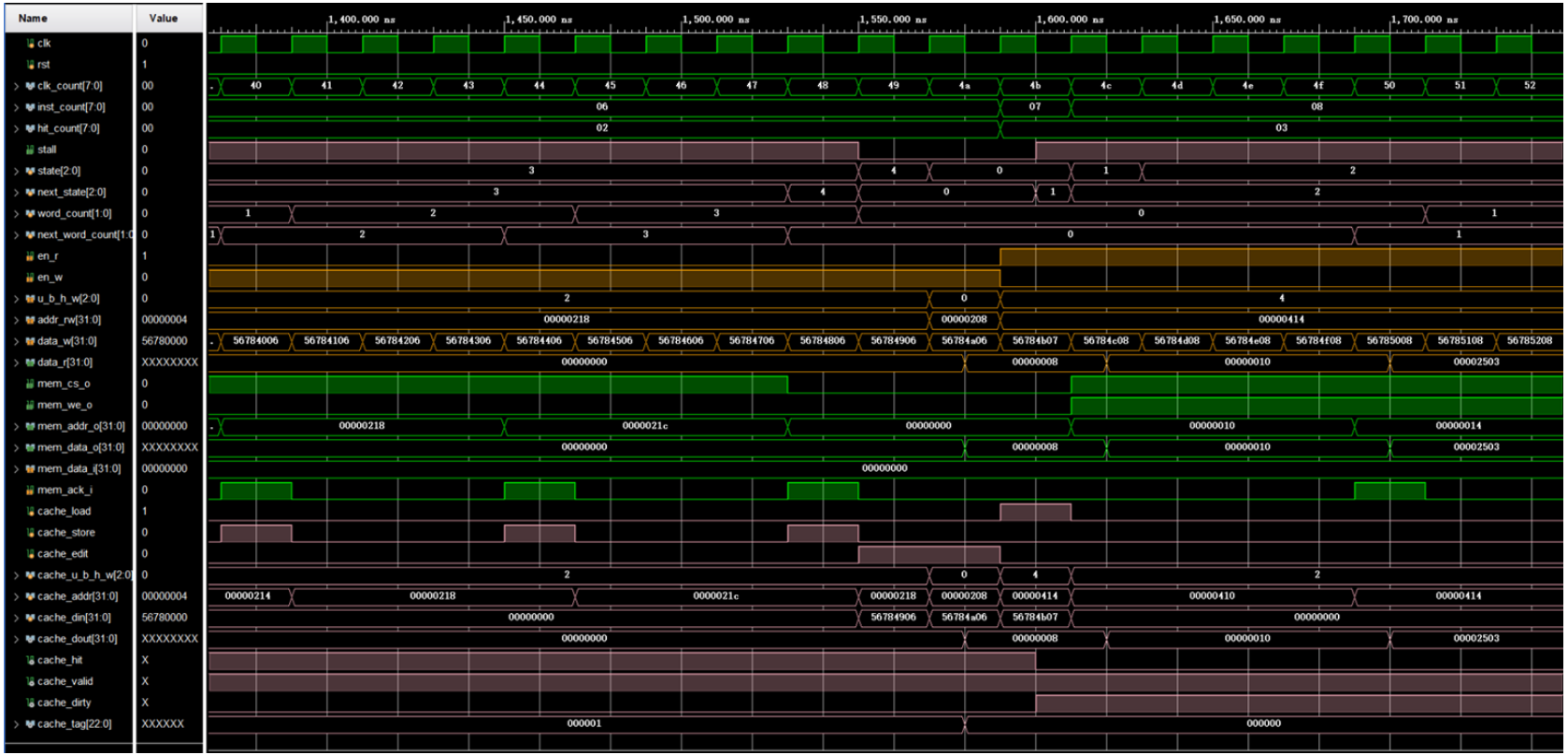
Simulation (3)



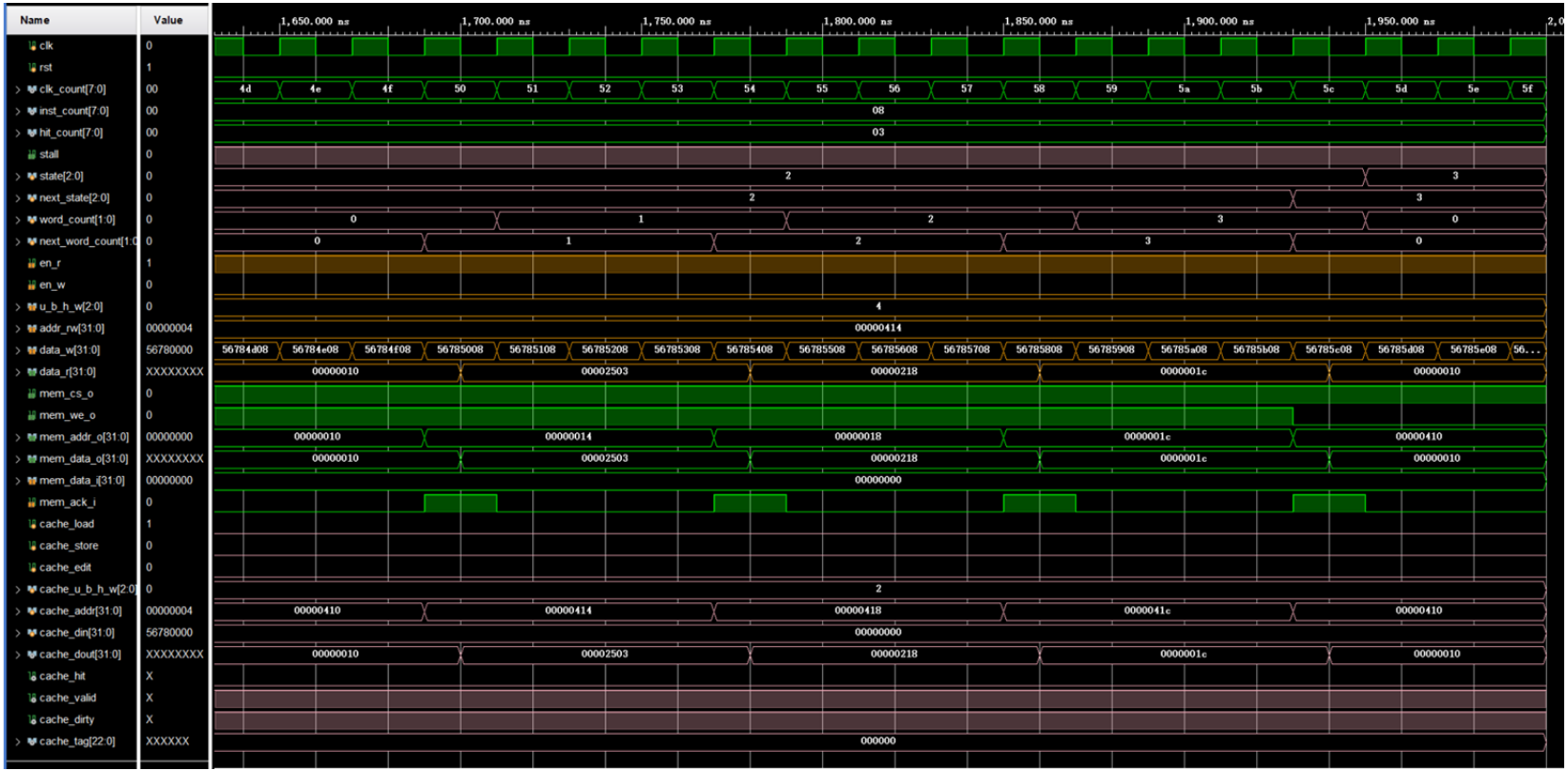
Simulation (4)



Simulation (5)



Simulation (6)



References

- 超标量处理器设计
- Modern Processor Design