Information Technology Institute

Cache Controller Implementation with Write-Through Policy

Group no.10

<u>Group Members:</u>

Name	Track
Ehab Mostafa Farghly	Design
Mohamed Mahmoud Elfeki	Verification

Top level architecture of RISC V with cache memory

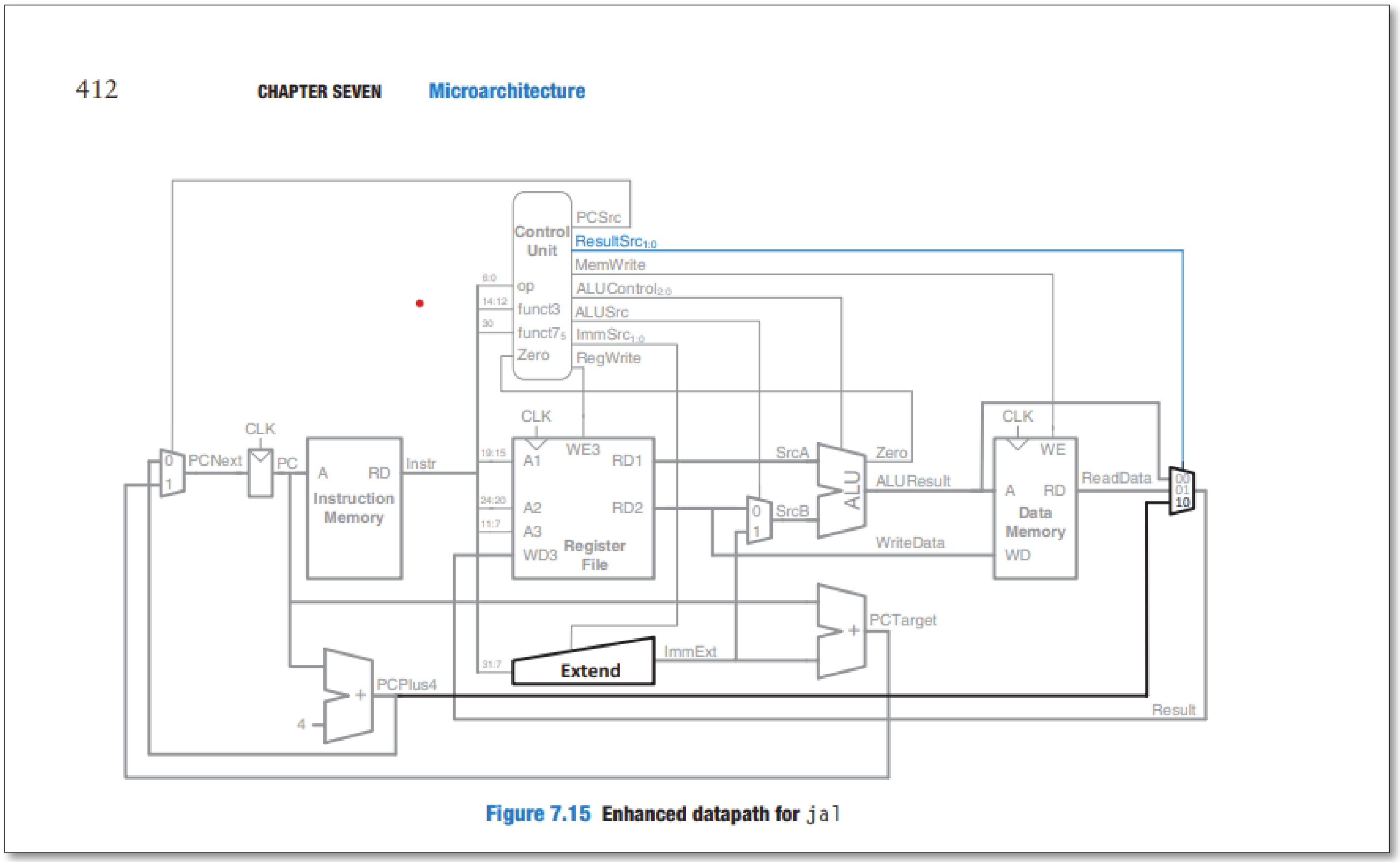

412 CHAPTER SEVEN Microarchitecture

Figure 7.15 Enhanced datapath for jal

Figure 1 RISC-v Architecture

Comments:

- Only change we did on RISC-v that we replaced Data Memory Block by our Implementation of Cache Memory.
- The main difference between this architecture and the architecture with cache memory that data memory has 2 more signals stall and memory read.
- stall signal is output from Cache Memory Block and input to the PC. If its value equal 1 it will stall PC and prevent it from incrementing and taking following instructions.
- Memory Read will come from control unit and gives value 1 only when the opcode is LW Instruction.

Top level architecture of cache memory

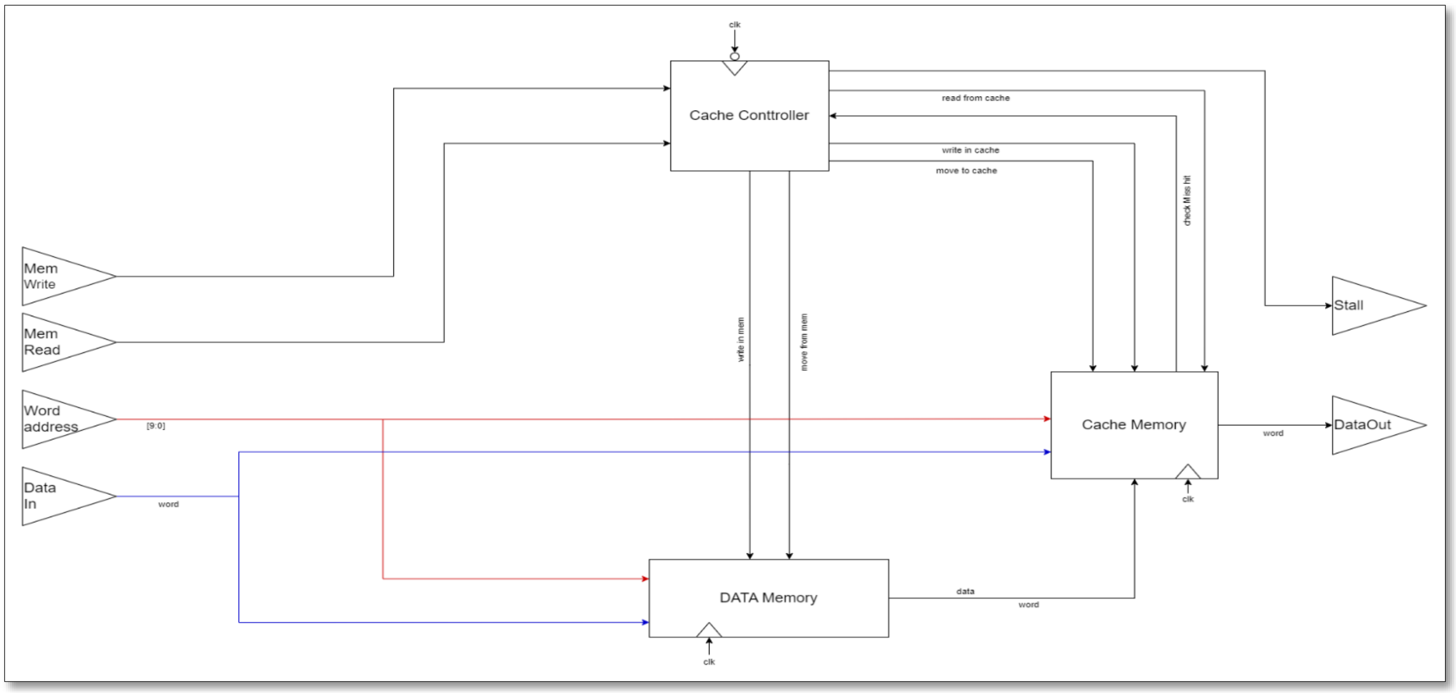


Figure 2 Cache Memory Block Diagram

Comments:

- This is the Block Diagram representing our implementation of the cache controller.
- DATA Memory & Cache Memory works at the Positive edge clk .
- Cache Controller works at the negative edge clk.
- So, we can consider All Architecture works at negedge clk of RISC-v because the Controlller is the brain of the design.

Cache Controller FSM

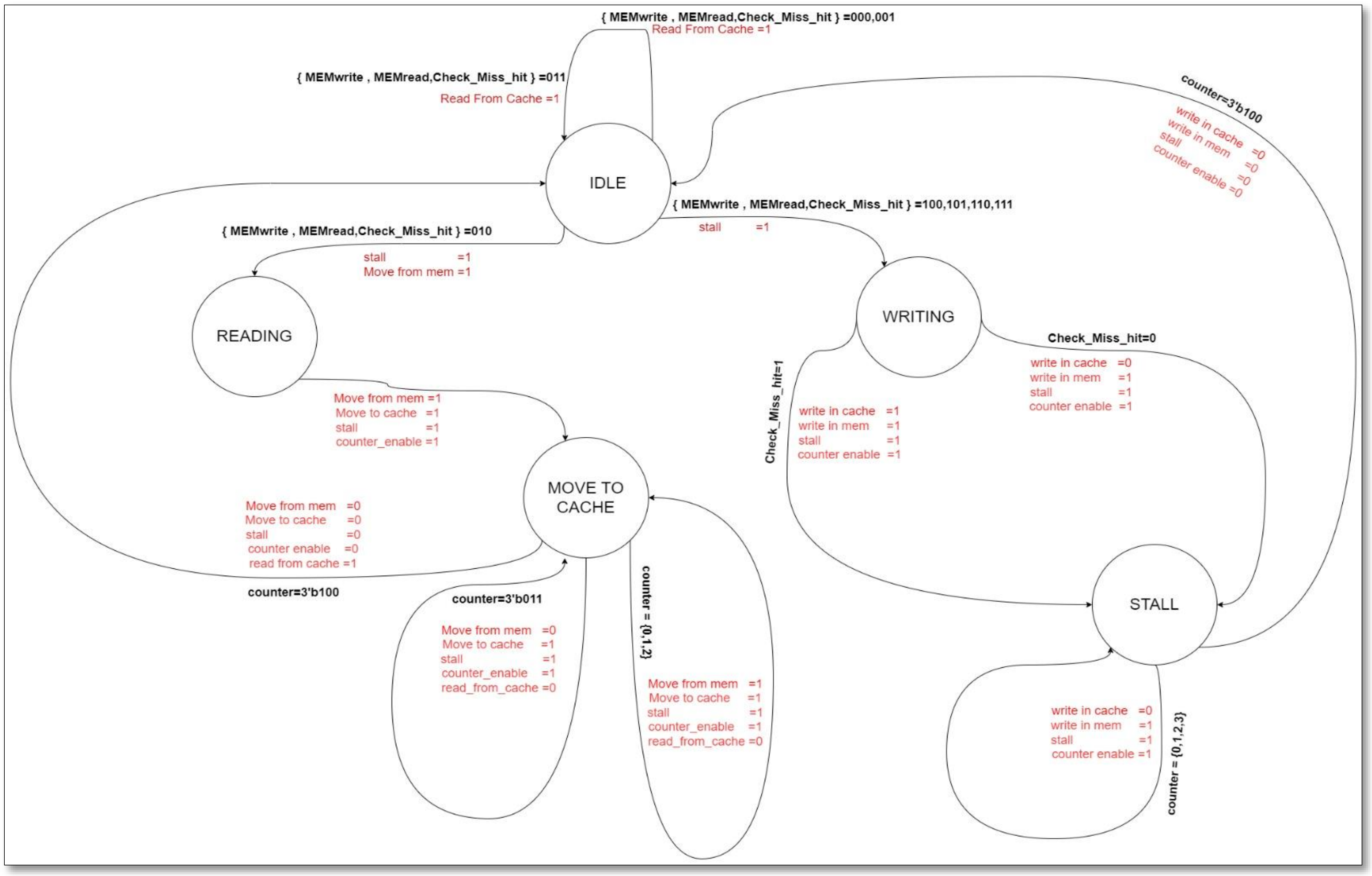


Figure 3 Cache Controller FSM

Simulation Testcases & Results

- **Test cases of cache memory**

```
 /***********************************        testcases     ***********************************
*****/

      /* scenario writing in block and check the reading operations of them miss first then
all hit */
          /** write operations **/

  write_data('b111_00010_01 ,'d16  );
  write_data('b111_00010_10 ,'d8   );
  write_data('b111_00010_00 ,'d4   );
  write_data('b111_00010_11 ,'d2   );


          /** read operations **/
  read_data ('b111_00010_11);
  read_data ('b111_00010_10);
  read_data ('b111_00010_01);
  read_data ('b111_00010_00);


      /* scenario writing in block exist in cache and check the writing operations of them
hit and store in cache and memory in same time */
          /** write operations **/

  write_data('b111_00010_01 ,'d100  );
  write_data('b111_00010_10 ,'d120  );

         /** read operations **/

  read_data ('b111_00010_00);
  read_data ('b111_00010_01);
  read_data ('b111_00010_10);
  read_data ('b111_00010_11);


    /* scenario addressing same block different in tag with different values */
          /** write operations **/

  write_data('b101_00010_01 ,'d200  );
  write_data('b101_00010_10 ,'d220  );

        /** read operations **/

  read_data ('b101_00010_00);
  read_data ('b101_00010_01);
  read_data ('b101_00010_10);
  read_data ('b111_00010_11);
  read_data ('b111_00010_01);


    /********************  Random processes ******************/
             /** write operations **/

  write_data('b000_00110_01 ,'d60  );  // Block 6    tag =0  ofsset=1
  write_data('b011_01010_01 ,'d100 );  // Block 10   tag =3  offset=1
  write_data('b010_00111_01 ,'d20  );  // Block 2    tag=2   offset=1
  write_data('b110_10010_10 ,'d60  );  // Blcok
18   tag=6   offset=2                  /*******/
  write_data('b000_01011_00 ,'d110 );  // BLock 11   tag =0  offset=0
  write_data('b101_10010_11 ,'d200 );  // Block 18   tag=5   offset=3
  write_data('b001_00000_11 ,'d1000);  // Block 0    tag =1  offset=3
  write_data('b001_10011_01 ,'d190 );  // Block 19   tag =1  offset=1
  write_data('b111_11010_10 ,'d260 );  // Block 26   tag =7  offset=2
```

```
    /** read operations **/

read_data ('b000_00110_01);
read_data ('b011_01010_01);
read_data ('b010_00111_01);
read_data ('b110_10010_10);
read_data ('b110_10010_10);
read_data ('b000_01011_00);
read_data ('b101_10010_11);
read_data ('b001_00000_11);
read_data ('b001_10011_01);
read_data ('b111_11010_10);
```

- **Simulation result in this case**

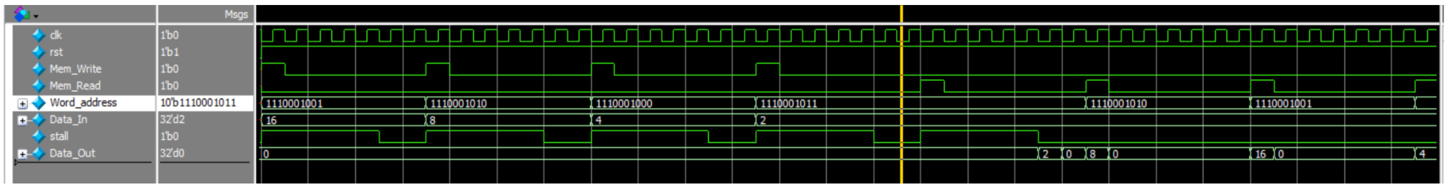


Figure 4 Cache simulation Results

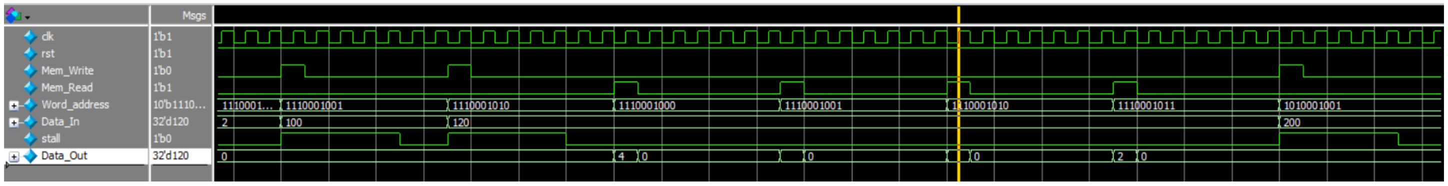


Figure 5 Cache simulation Results

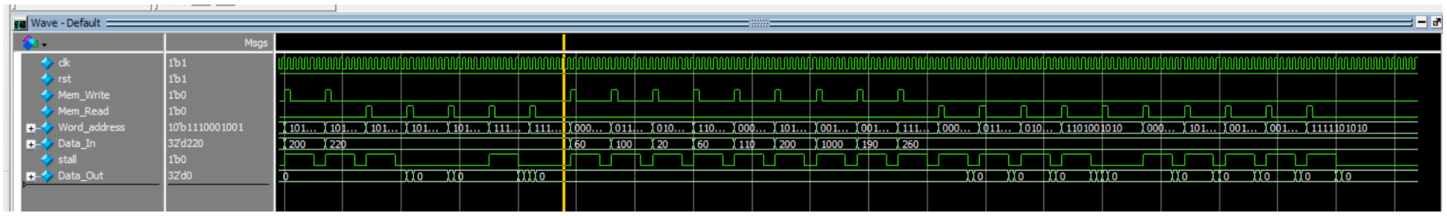


Figure 6 Cache simulation Results

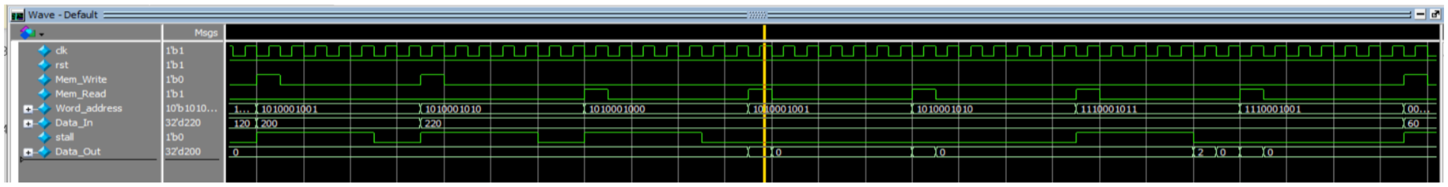


Figure 7 Cache simulation Results

- **Test cases RISC-v integrated with Cache Memory Block**

Machine Code	Assembly Code
00F00093	addi x1 x0 15
01400113	addi x2 x0 20
01900193	addi x3 x0 25
01E00213	addi x4 x0 30
01900293	addi x5 x0 25
0020A2A3	sw x2 5(x1)
0030A323	sw x3 6(x1)
0040A3A3	sw x4 7(x1)
0050A423	sw x5 8(x1)
0050A303	lw x6 5(x1)
0060A383	lw x7 6(x1)
0070A403	lw x8 7(x1)
0080A483	lw x9 8(x1)

- **Simulation result in this case**

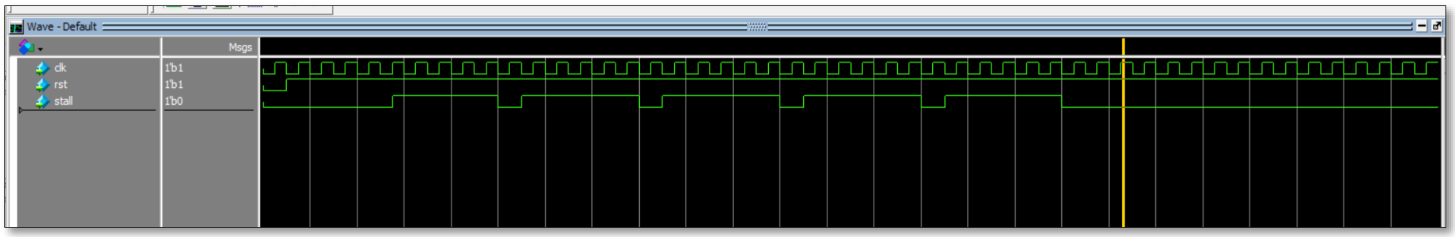
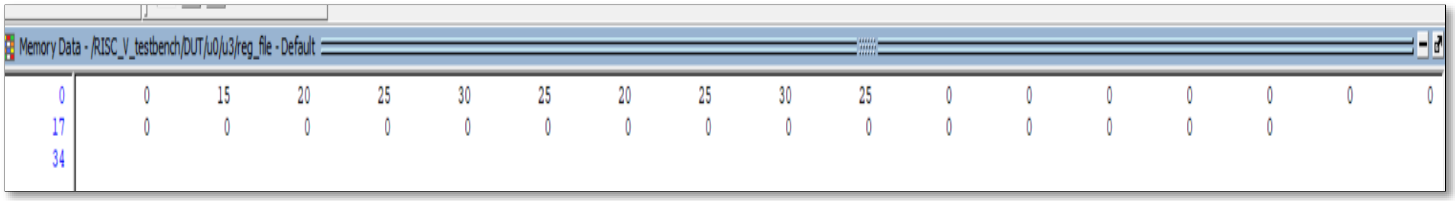


Figure 8 RISC-V with Cache memory simulation Results


Memory Data - /RISC_V_testbench/DUT/u0/u3/reg_file - Default

```
 0    0  15  20  25  30  25  20  25  30  25   0   0   0   0   0   0   0
17    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
34
```

Figure 9 Reg File Memory

Comments:

- As we see in the waveform stall signal is high five times as miss occur in all 4 times of storing and miss occur in first time of loading then hit occur and register files contain the proper results.

RTL of the Cache Memory Controller

Cache Controller

```
module cache_controller(
    input  wire             clk,rst,
    input  wire             Mem_Write,Mem_Read,
    input  wire             check_Miss_hit,

    output reg              write_in_mem,
    output reg              write_in_cache,
    output reg              move_from_mem,
    output reg              move_to_cache,
    output reg              read_from_cache,
    output reg              stall
);
parameter  IDLE          =3'b000,
           READING       =3'b001,
           MOVE_TO_CACHE =3'b010,
           WRITING       =3'b011,
           STALL         =3'b100;

reg     [2:0] current_state,next_state;
reg     [2:0] counter;
reg           counter_enable;


always @(negedge clk or negedge rst)
    begin
        if(!rst)
          begin
            counter <='b0;
          end
        else if (counter_enable)
            begin
            counter <= counter +'b1;
            end
       else
          begin
            counter <= 'b0;
          end
    end

always @(negedge clk or negedge rst)
    begin
        if(!rst)
          begin
            current_state <=IDLE;
          end
        else
          begin
            current_state <= next_state;
          end
    end

always@(*)
    begin
        write_in_mem    ='b0;
        write_in_cache  ='b0;
        move_from_mem   ='b0;
        move_to_cache   ='b0;
        read_from_cache ='b0;
        counter_enable  ='b0;

        case(current_state)
             IDLE        : begin
                            if((Mem_Read=='b1) && (Mem_Write =='b0) && (check_Miss_hit=='b1)
)
                                begin
                                    next_state = IDLE;
                                    stall           ='b0;
                                    read_from_cache='b1;
                                end

                            else if((Mem_Read=='b1) && (Mem_Write =='b0) &&
(check_Miss_hit=='b0) )
                                begin
                                    next_state = READING;
                                    stall='b1;
                                    move_from_mem = 'b1;

                                end

                            else if((Mem_Read=='b0) && (Mem_Write =='b1) )
                                begin
                                    next_state = WRITING;
                                    stall='b1;
                                end

                            else
                                begin
                                    next_state =IDLE;
                                    stall='b0;

                                end
                          end

           READING      : begin
```

```
                                 next_state    = MOVE_TO_CACHE;
                                 move_from_mem = 'b1;
                                 move_to_cache = 'b1;
                                 stall         = 'b1;  /*****/
                                 counter_enable= 'b1;
                         end

        MOVE_TO_CACHE  : begin

                        if(counter == 3'b100)
                             begin
                                 next_state      = IDLE;
                                 read_from_cache = 'b1;
                                 stall           = 'b0;
                                 counter_enable  = 'b0;
                                 move_from_mem   = 'b0;
                                 move_to_cache   = 'b0;
                             end
                        else if(counter == 3'b011)
                             begin
                                 next_state      = MOVE_TO_CACHE;
                                 read_from_cache = 'b0;
                                 stall           = 'b1;
                                 counter_enable  = 'b1;
                                 move_from_mem   = 'b0;
                                 move_to_cache   = 'b1;
                             end
                        else
                             begin
                                 next_state    = MOVE_TO_CACHE;
                                 move_from_mem = 'b1;
                                 move_to_cache = 'b1;
                                 stall         = 'b1;
                                 counter_enable= 'b1;
                             end
                         end

        WRITING      : begin
                         counter_enable='b1;

                        if(check_Miss_hit )      // 1 hit       0 Miss
                              begin
                                 write_in_cache ='b1;
                                 write_in_mem   ='b1;
                                 stall          ='b1;
                                 next_state     =STALL;
                              end
                        else
                             begin
                                 write_in_cache ='b0;
```

```
                                    write_in_mem   ='b1;
                                    stall          ='b1;
                                    next_state     =STALL;
                                end
                              end

            STALL      : begin
                    counter_enable  ='b1;

                            if(counter == 'b100)
                                begin
                                    stall='b0;
                                    write_in_cache='b0;
                                    write_in_mem='b0;
                                    next_state =IDLE;
                                    counter_enable  ='b0;
                                end
                            else
                                begin
                                    stall='b1;
                                    write_in_cache='b0;
                                    write_in_mem='b1;
                                    next_state =STALL;
                                    counter_enable  ='b1;
                                 end
                          end

            default     : begin
                                write_in_mem    ='b0;
                                write_in_cache  ='b0;
                                move_from_mem   ='b0;
                                read_from_cache ='b0;
                                stall           ='b0;
                                counter_enable  ='b0;
                                next_state      =IDLE;
                            end
        endcase
    end
endmodule
```

Cache Memory

```
module cache_Memory (
    input    wire          clk,rst,
    input    wire  [9:0]   Word_address,
    input    wire  [31:0]  data,
    input    wire  [31:0]  Data_IN,
    input    wire          write_in_cache,read_from_cache,
    input    wire          move_to_cache,

    output   reg           check_Miss_hit,
    output   reg   [31:0]  Data_Out
);

reg [31:0]  cache_mem [0:127];
reg [3:0]   valid_tag [0:31];
reg [1:0]   move_count;
integer i,k;

always @(posedge clk or negedge rst )
    begin
      if(!rst)
         begin
            for (i=0;i<128;i=i+1)
               begin
                cache_mem[i]<='b0;
               end
            for (k=0;k<32;k=k+1)
               begin
                valid_tag[k]<='b0;
               end
               move_count <= 'b0;
         end

      else if (write_in_cache)
         begin
            cache_mem [Word_address[6:0]] <= Data_IN;
            valid_tag [Word_address[6:2]] <={1'b1,Word_address[9:7]};
         end

      else if (move_to_cache)
         begin
            cache_mem [{Word_address[6:2],move_count}] <= data;
            valid_tag [Word_address[6:2]]              <={1'b1,Word_address[9:7]};
            move_count                                 <= move_count +'b1;
         end


    end
```

```
   always @(*)
      begin
         if (valid_tag[Word_address[6:2]] == {1'b1 , Word_address [9:7]})
            begin
             check_Miss_hit ='b1;
            end
         else
            begin
             check_Miss_hit ='b0;
            end
      end
       always @(*) begin
         if (!rst) begin
            Data_Out='b0;

         end
         else if (read_from_cache)
         begin
            Data_Out = cache_mem [Word_address[6:0]];
         end
         else begin
            Data_Out='b0;
         end

 end
endmodule
```

Data Memory

```
module  data_memory (
    input        wire          clk,rst,
    input        wire   [9:0]    Word_address,
    input        wire   [31:0]   Data_In,
    input        wire            write_in_mem,
    input        wire            move_to_cache,

    output       reg    [31:0]         data
);

    reg [31:0]  main_memory [0:1023];
    reg [1:0]   address_count;
    integer     i;

always@(posedge clk or negedge rst)
    begin
        if(!rst)
          begin
            for(i=0;i<1024;i=i+1)
              main_memory[i] <='b0;
              address_count  <='b0;
              data           <='b0;
          end
        else
          begin
             if(write_in_mem)
               begin
                main_memory[Word_address] <= Data_In;
                address_count             <='b0;
               end
             else if (move_to_cache)
               begin
                address_count <=address_count+1;
                data          <= main_memory[{Word_address[9:2],address_count}];
               end
               else
               begin
                address_count <='b0;
               end
          end
    end

endmodule
```

Cache Memory TOP

```verilog
module cache_Memory_top (
    input  wire         clk,rst,
    input  wire         Mem_Write,Mem_Read,
    input  wire  [9:0]  Word_address,
    input  wire  [31:0]  Data_In,

    output wire         stall,
    output wire  [31:0]  Data_Out
) ;

/************************************/
wire         check_Miss_hit;
wire         write_in_mem,write_in_cache;
wire         move_from_mem;
wire         move_to_cache;
wire         read_from_cache;

cache_controller u0(
    .clk(clk),
    .rst(rst),
    .Mem_Write(Mem_Write),
    .Mem_Read(Mem_Read),
    .check_Miss_hit(check_Miss_hit),

    .write_in_mem(write_in_mem),
    .write_in_cache(write_in_cache),
    .move_from_mem(move_from_mem),
    .move_to_cache(move_to_cache),
    .read_from_cache(read_from_cache),
    .stall(stall)
);

/*************************************/
wire  [31:0]     data;

cache_Memory u1(
    .clk(clk),
    .rst(rst),
    .Word_address(Word_address),
    .data(data),
    .Data_IN(Data_In),
    .write_in_cache(write_in_cache),
    .read_from_cache(read_from_cache),
    .move_to_cache(move_to_cache),

    .check_Miss_hit(check_Miss_hit),
    .Data_Out(Data_Out)
);
```

```
/************************************/

data_memory u2(
    .clk(clk),
    .rst(rst),
    .Word_address(Word_address),
    .Data_In(Data_In),
    .write_in_mem(write_in_mem),
    .move_to_cache(move_from_mem),

    .data(data)
);

endmodule
```