

Cache Controller Implementation with Write-Through Policy

Group no.10

***Group Members:***

|  |  |
| --- | --- |
| Name | Track |
| Ehab Mostafa Farghly | Design |
| Mohamed Mahmoud Elfeki | Verification |

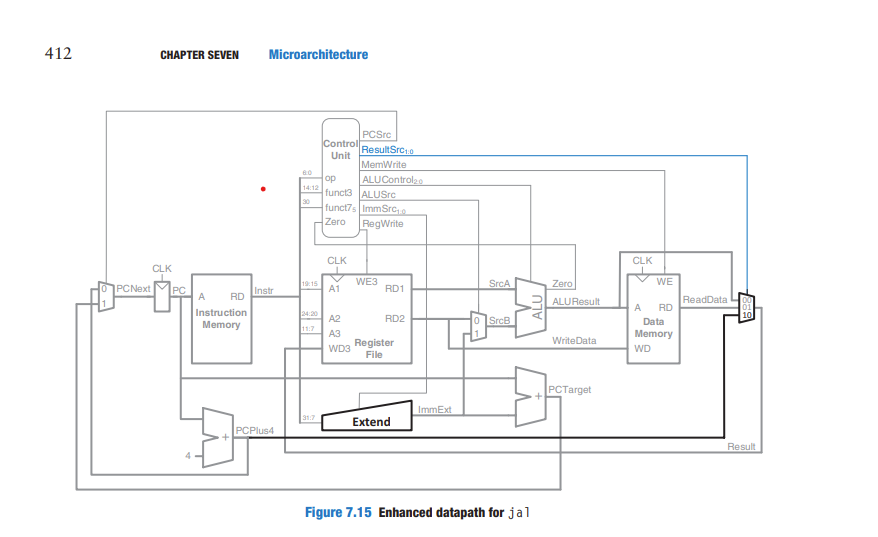
**Top level architecture of RISC V with cache memory**

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**

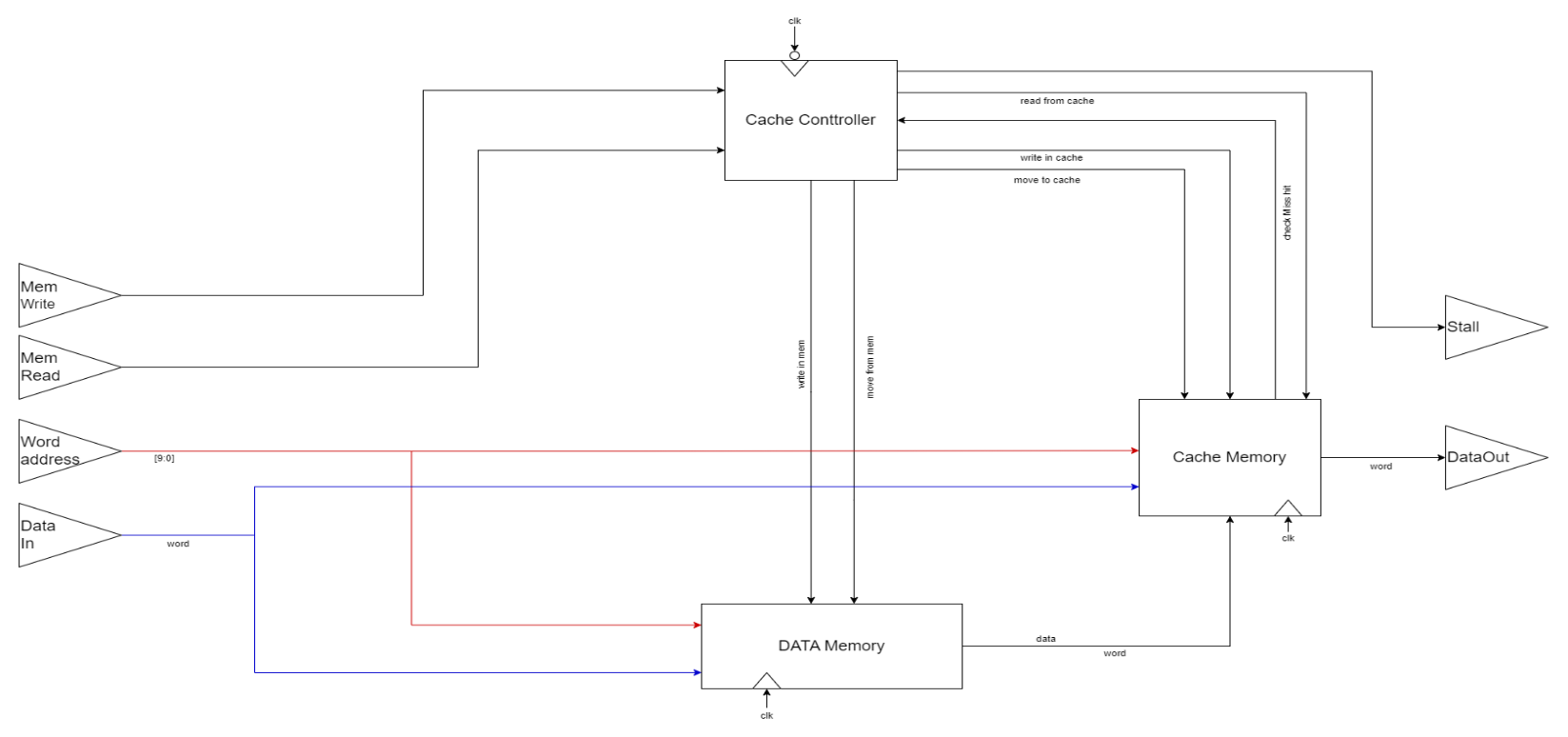


Figure 2 Cache Memory Block Diagram

**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**

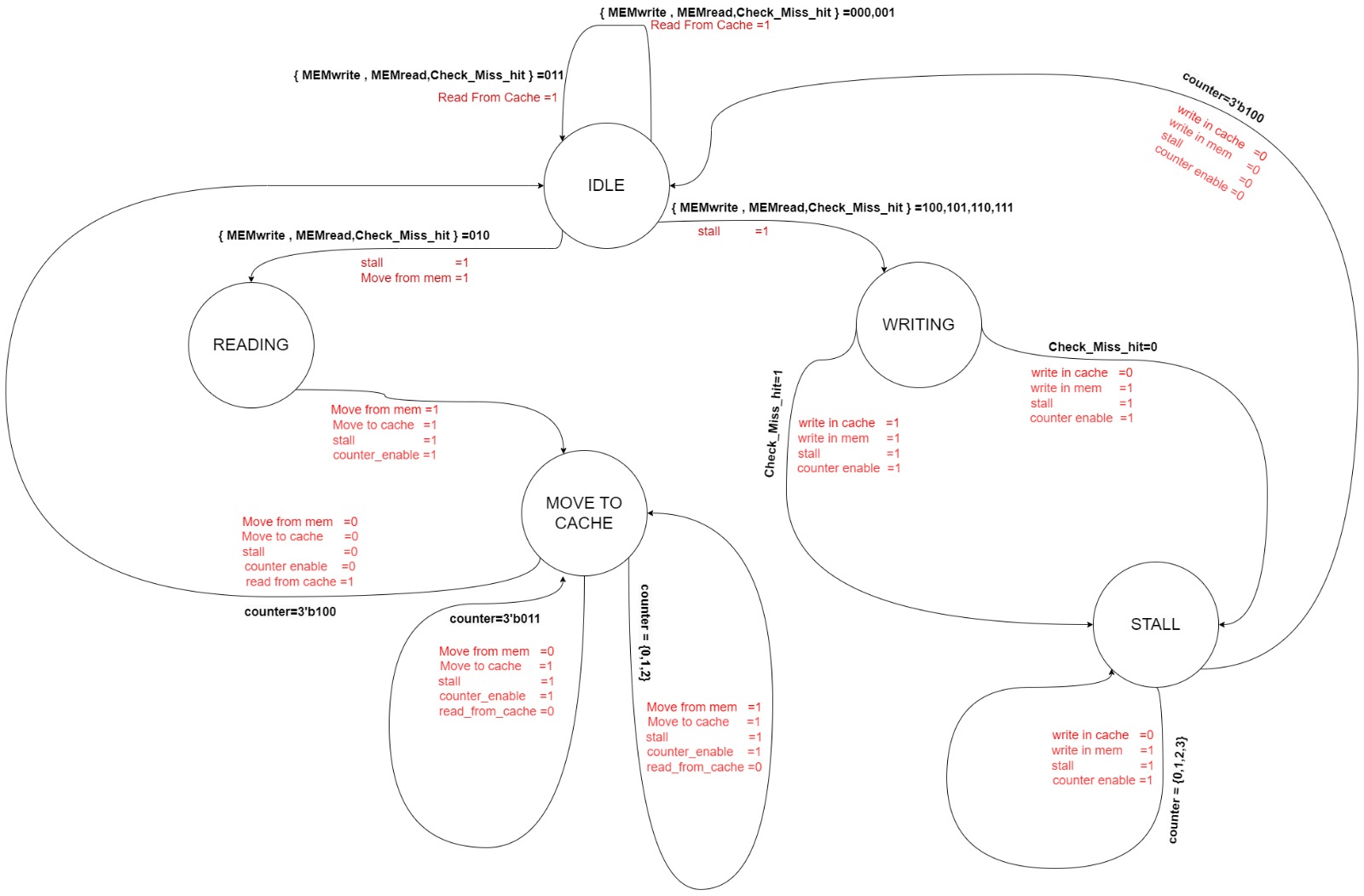


Figure 3 Cache Controller FSM

**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**

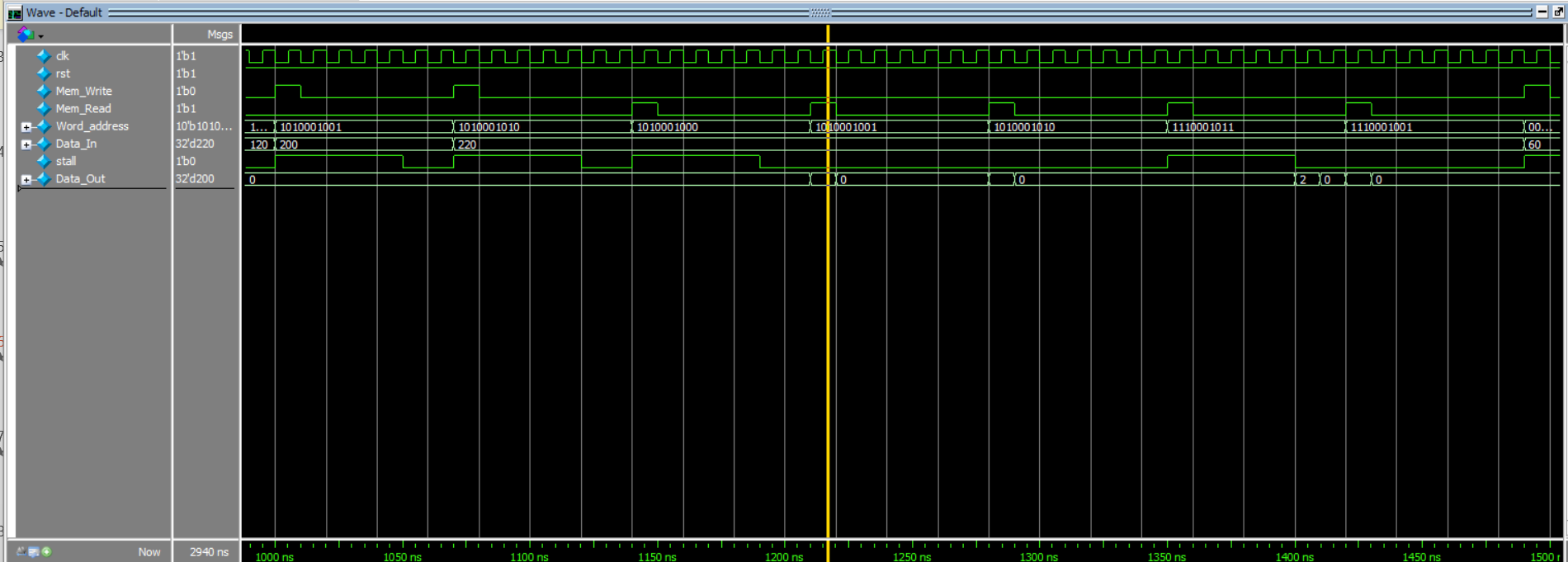


Figure 7 Cache simulation Results

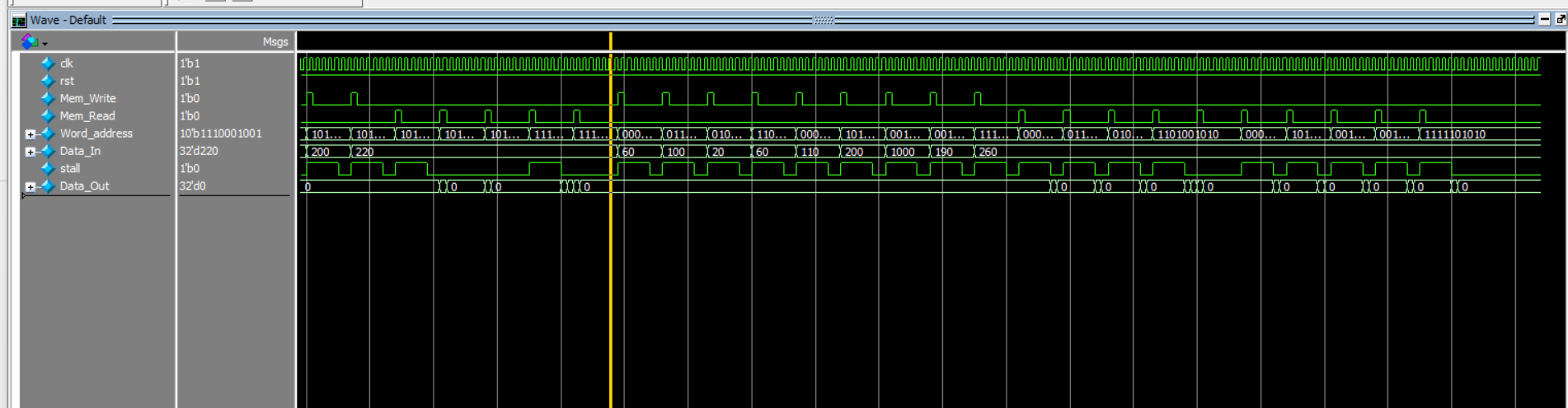


Figure 6 Cache simulation Results

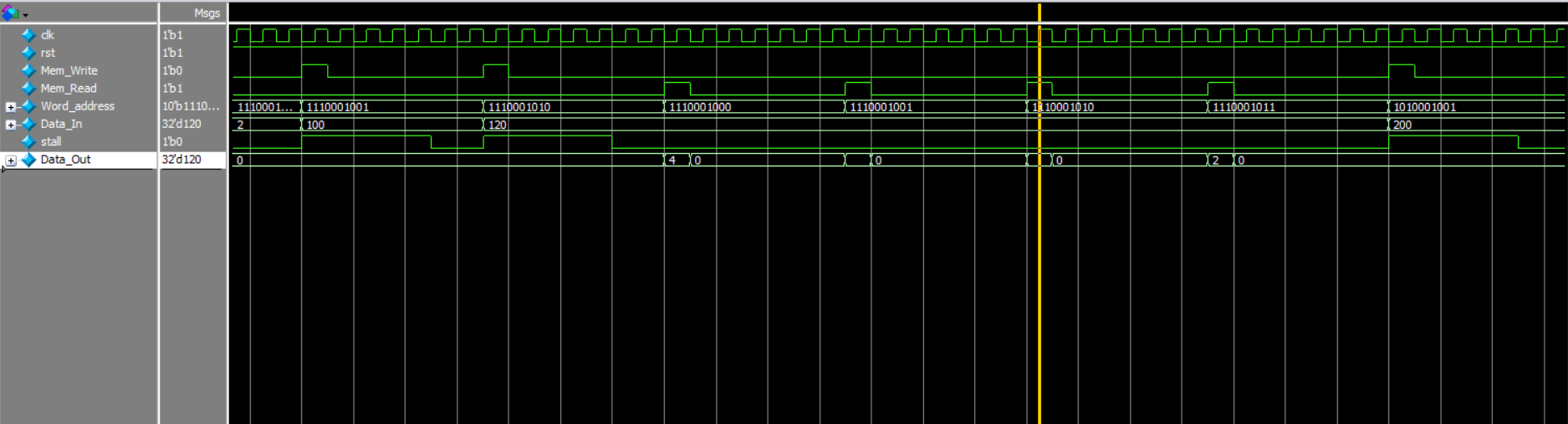


Figure 5 Cache simulation Results

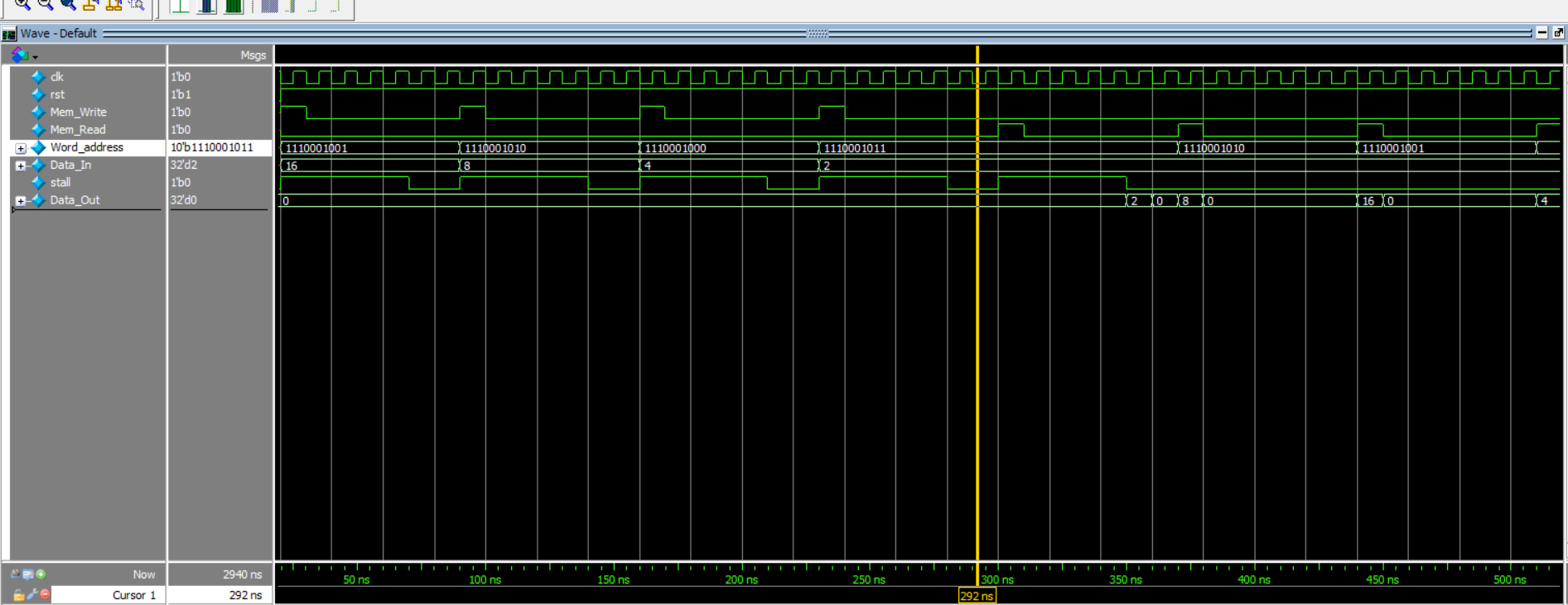


Figure 4 Cache simulation Results

* **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**

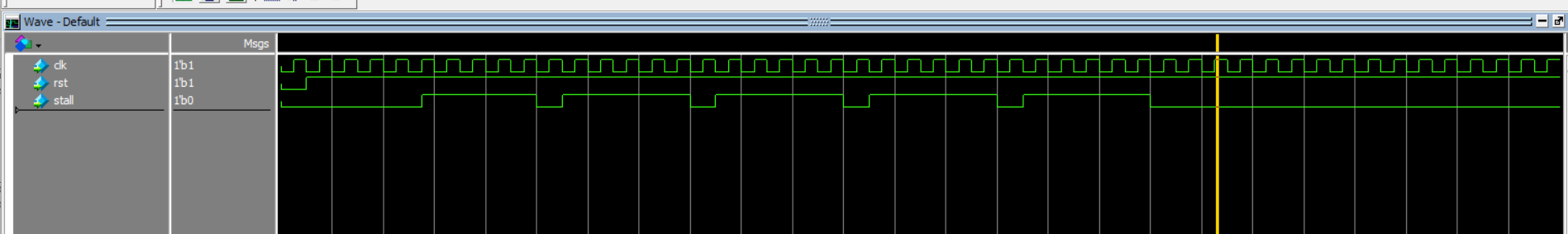
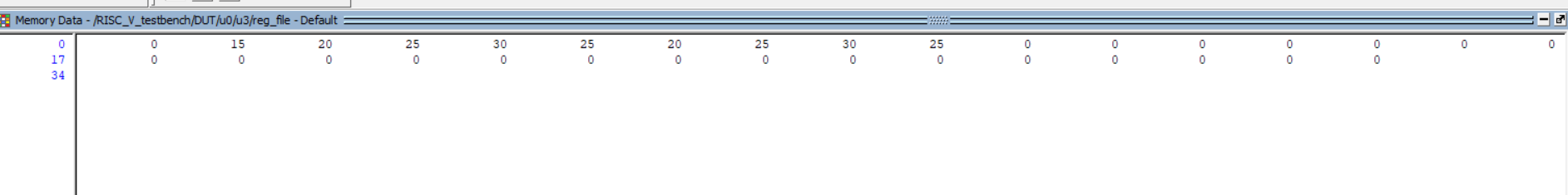


Figure 9 Reg File Memory

Figure 8 RISC-V with Cache memory simulation Results

**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**

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