Information Technology Institute

Ministry of Communications and Information Technology

egypt.makes.electronics

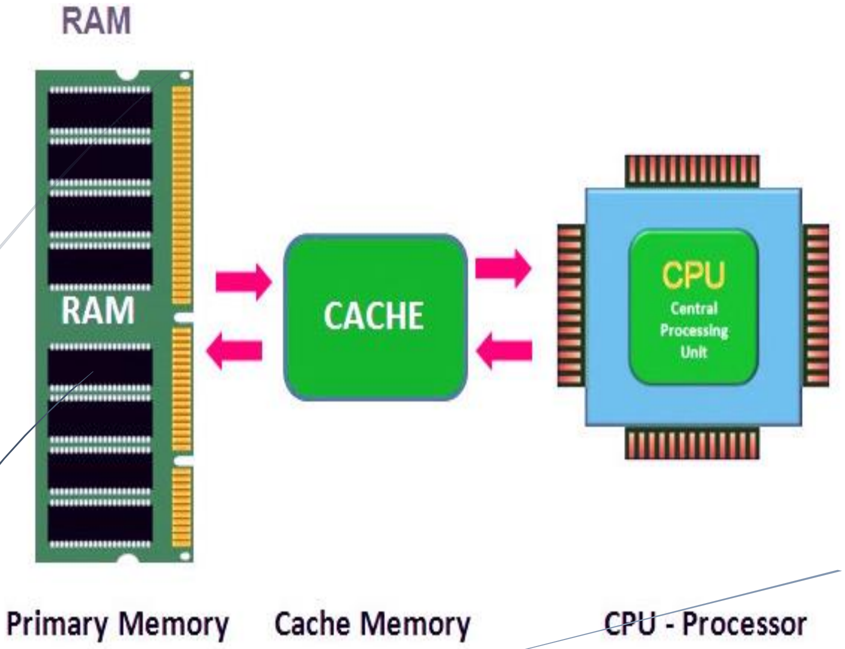
RISC-V Project (Phase 2)

Cache Controller

Mohamed Elsayed Ali Ebrahim Saad

Group Number: (G1)

Digital IC Design (New Capital)



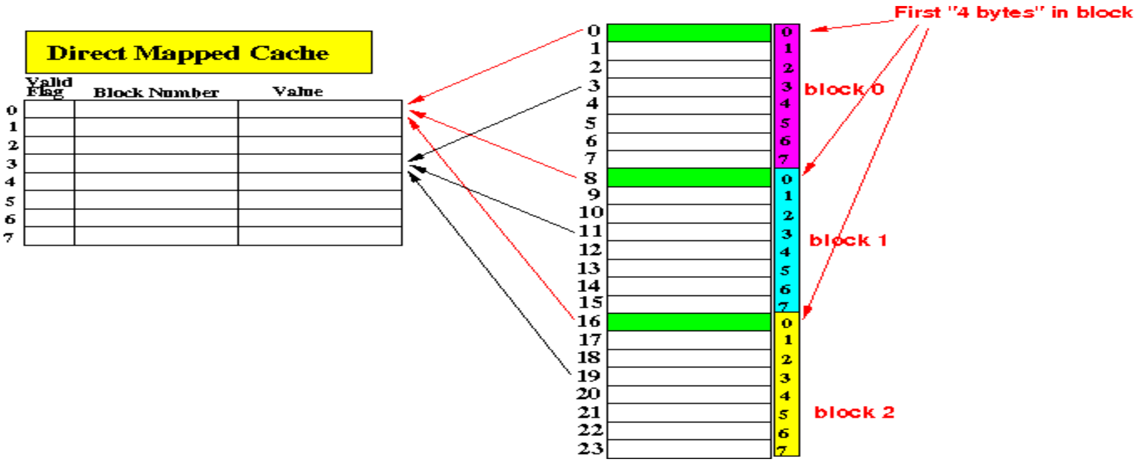
Contents

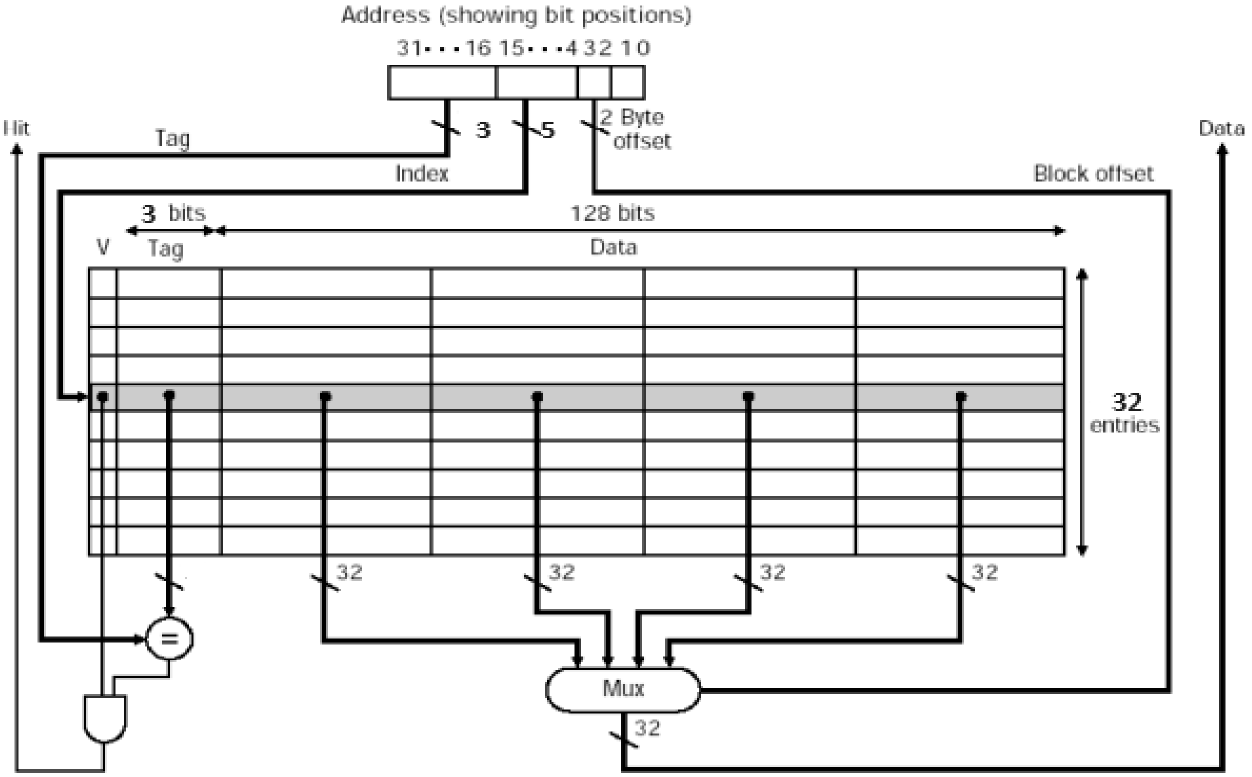
1. Overview:

Cache Controller Implementation with Write-Through Policy

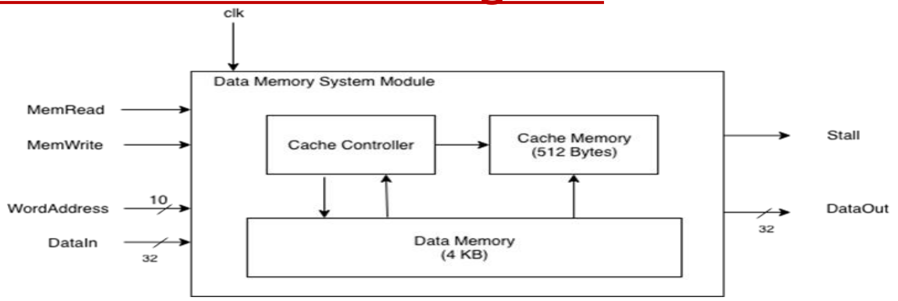
In this project, we will work on implementing a simple caching system for the RISC-V processor. we will integrate the caching system with the **single-cycle implementation**. Additionally, we assume the following:

- Only data memory will be cached. The instruction memory will not be affected.
- We will have only one level of coaching.
- The main memory module is assumed to have a capacity of 4 Kbytes (word addressable using 10 bits or byte addressable using 12 bits)
- Main memory access (for reading or writing) takes 4 clock cycles.
- The data cache geometry is (512, 16, 1). This means that the total cache capacity is 512 bytes, that each cache block is 16 bytes (implying that the cache has 32 blocks in total), and that the cache uses direct mapping.
- The cache uses write-through and write-around policies for write hit and write miss handling and no write buffers exist. This implies that all SW instructions need to stall the processor.
- LW instructions will only stall the processor in case of a miss.



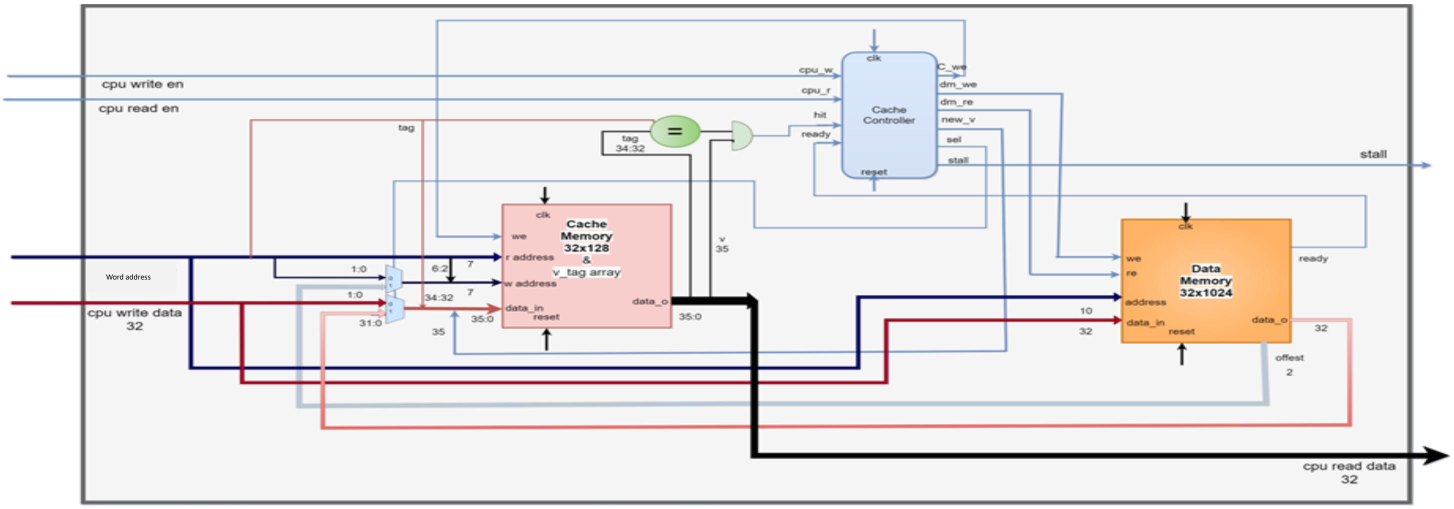


2. Architecture and RTL Block Diagram:

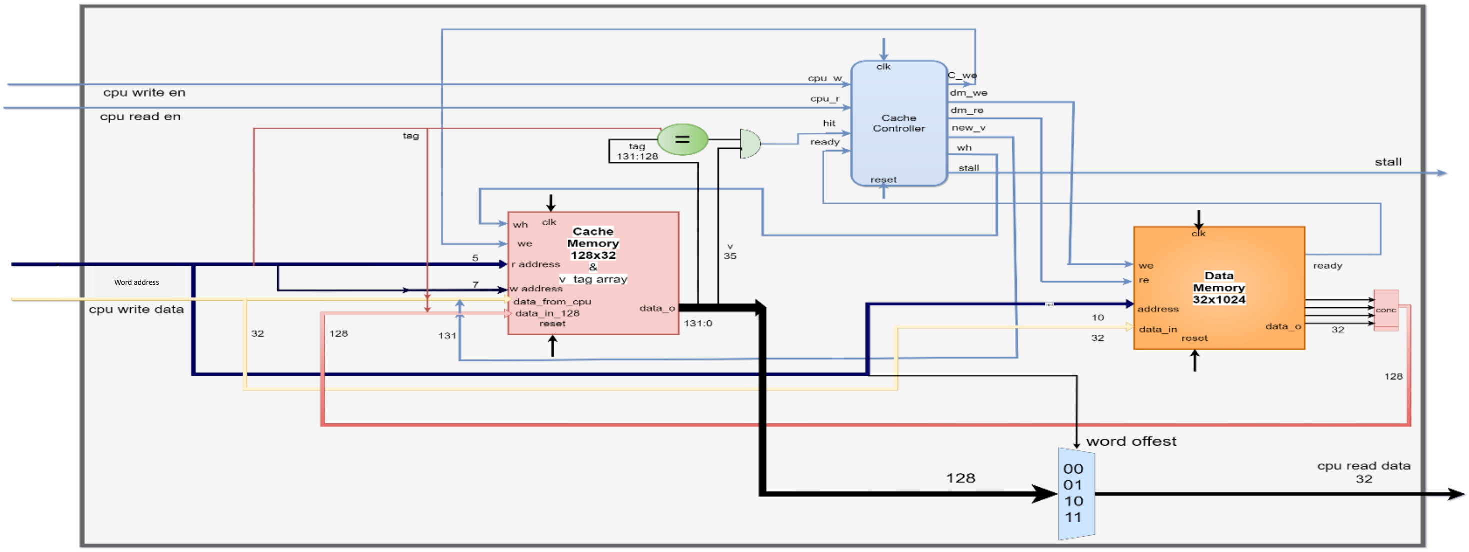


In this report there are two different micro architecture implementations for cache system. The main difference between them:

- **First architecture:** data transferred from the main memory to the cache memory word by word (copy a word each cycle) that's mean that the cache system needs 4 cycle to move 4 words (1 block seat) overall, when risc-v preform write operation or read and miss operation risc-v processor need 4 clock cycles.
- **Second architecture:** data transferred from the main memory to the cache memory 4 words (complete block seat) that's mean that cache system needs less than 4 cycle to move 4 words (1 block seat) overall, when risc-v preform write operation or read and miss operation risc-v need less than 4 clock cycles.

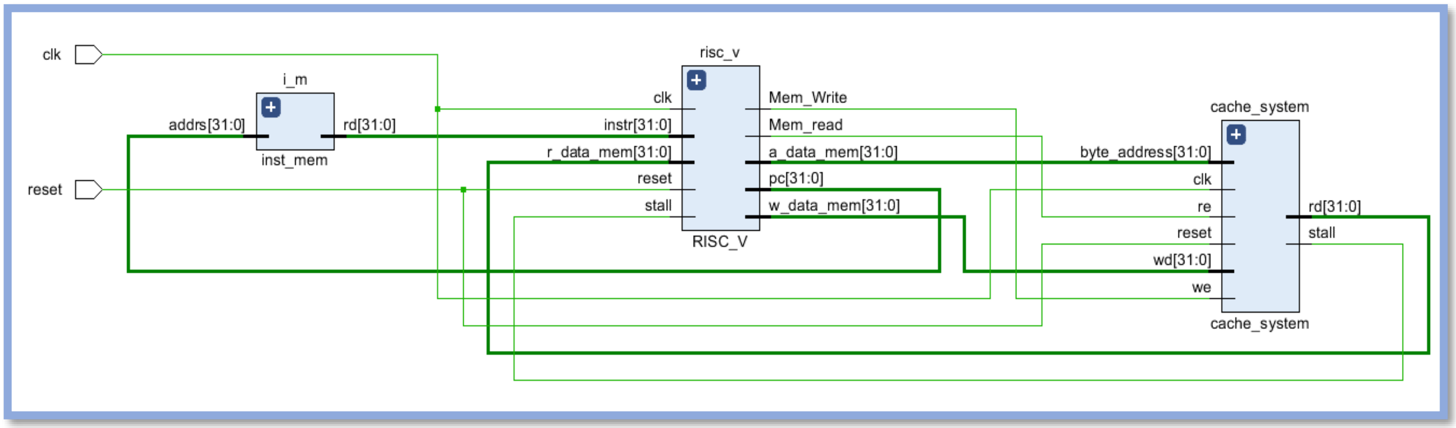


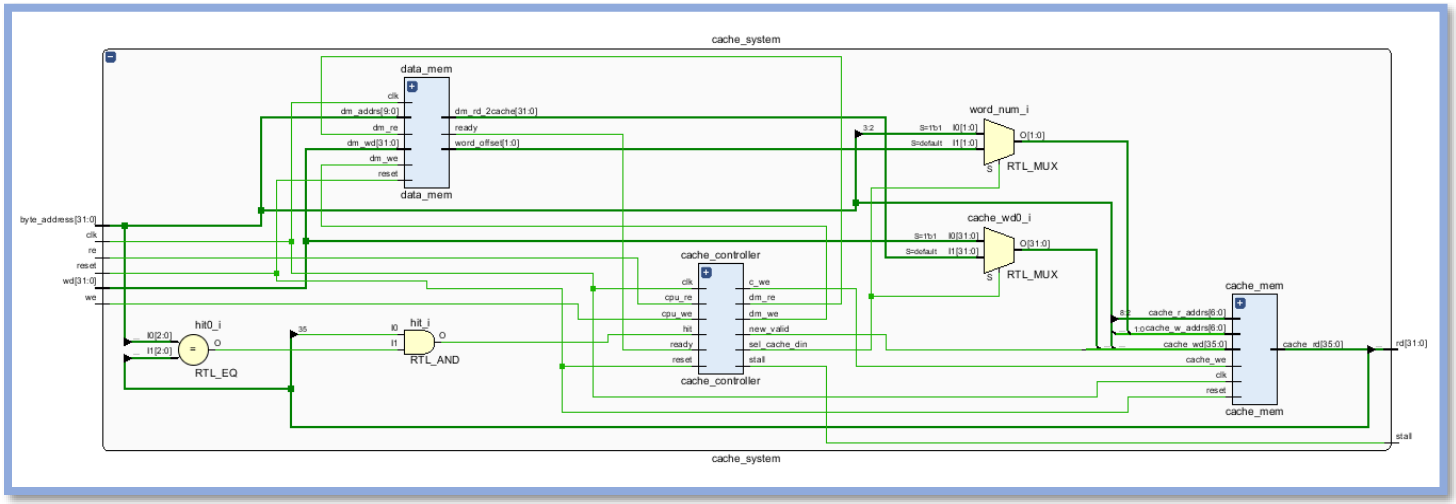
First Architecture

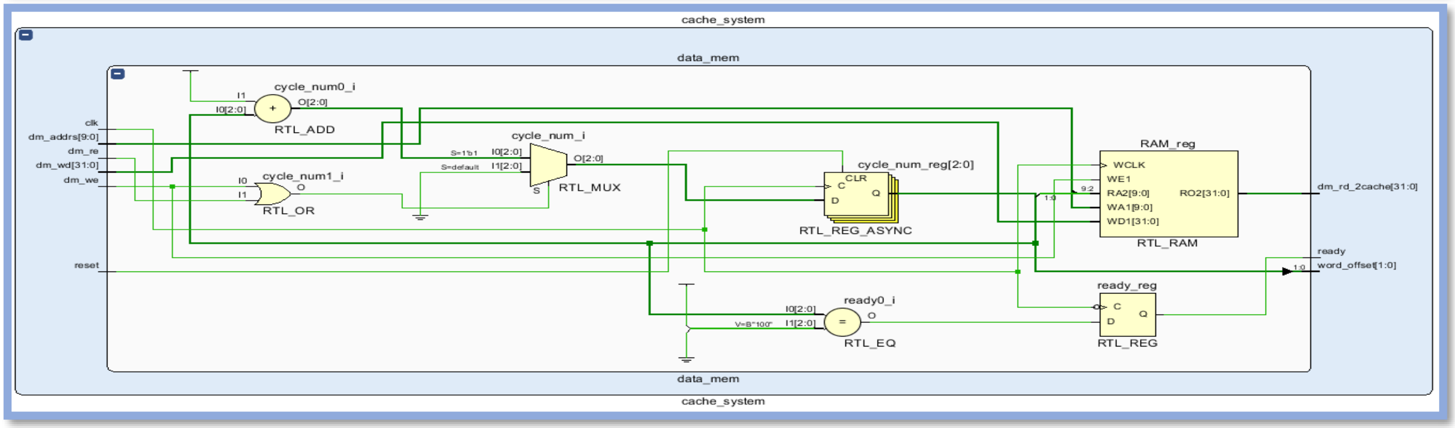


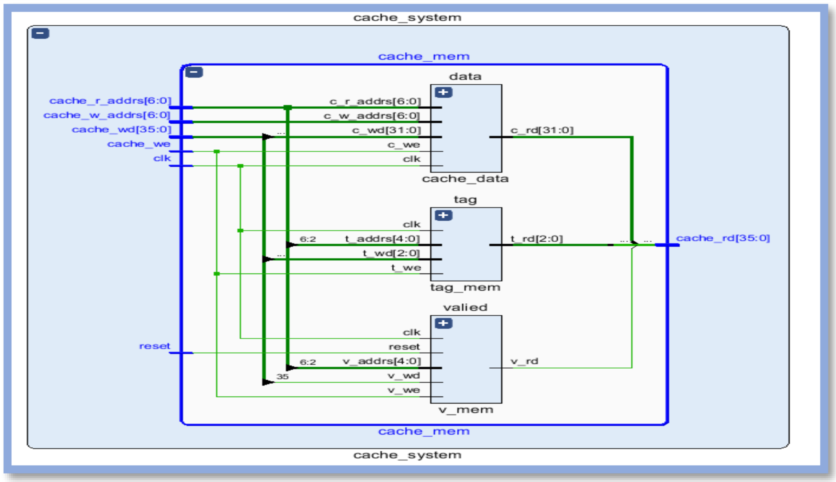
Second Architecture

❖ First Cache System Architecture RTL Block Diagram:

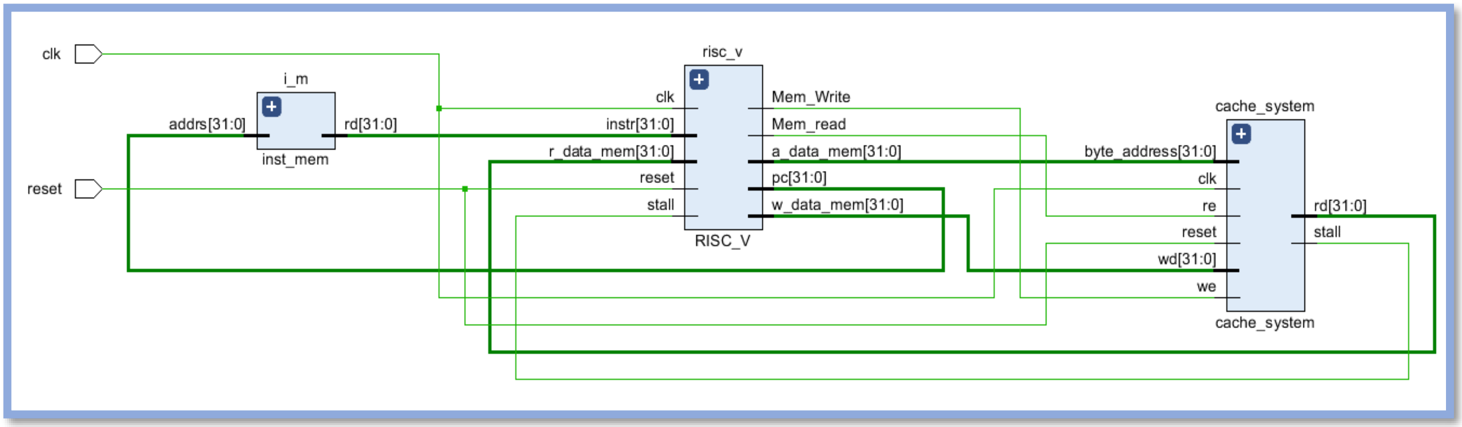


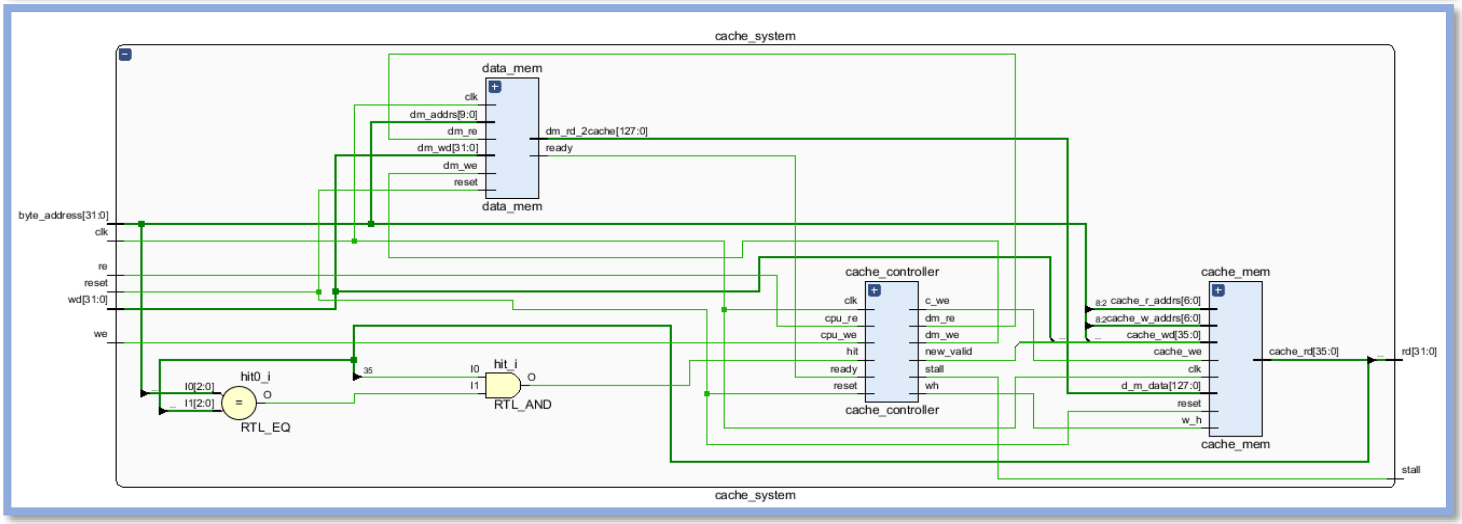


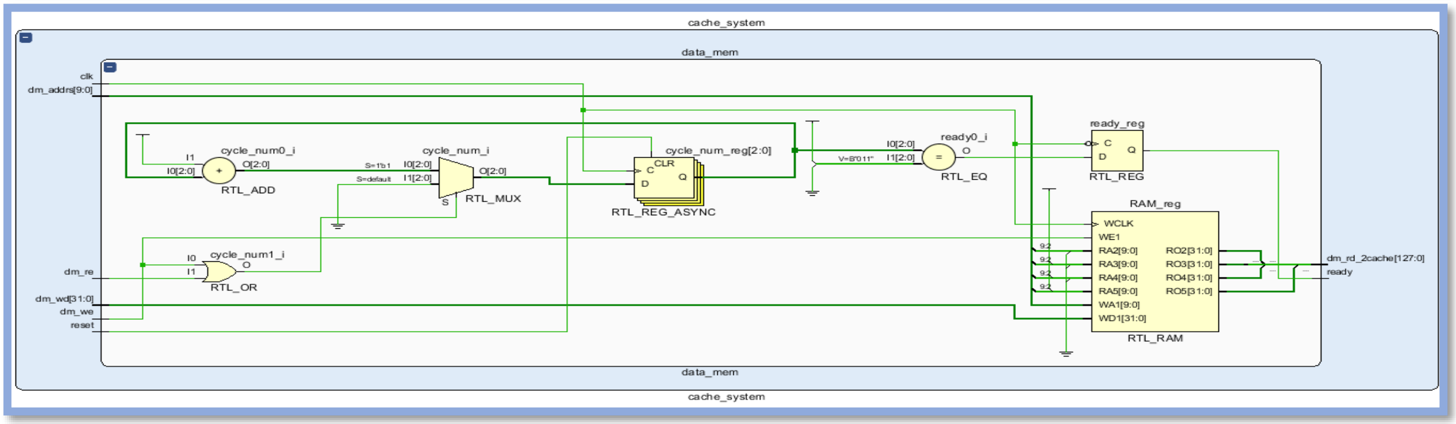


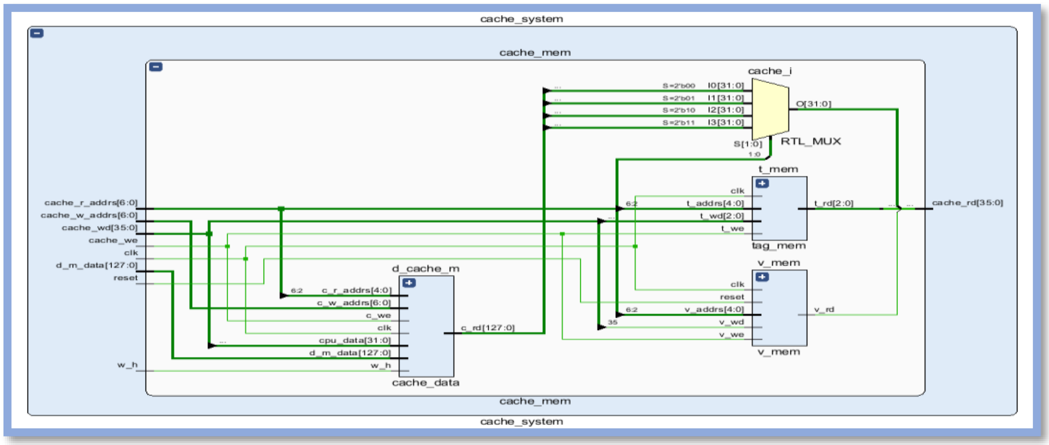


❖ Second Cache System Architecture RTL Block Diagram:

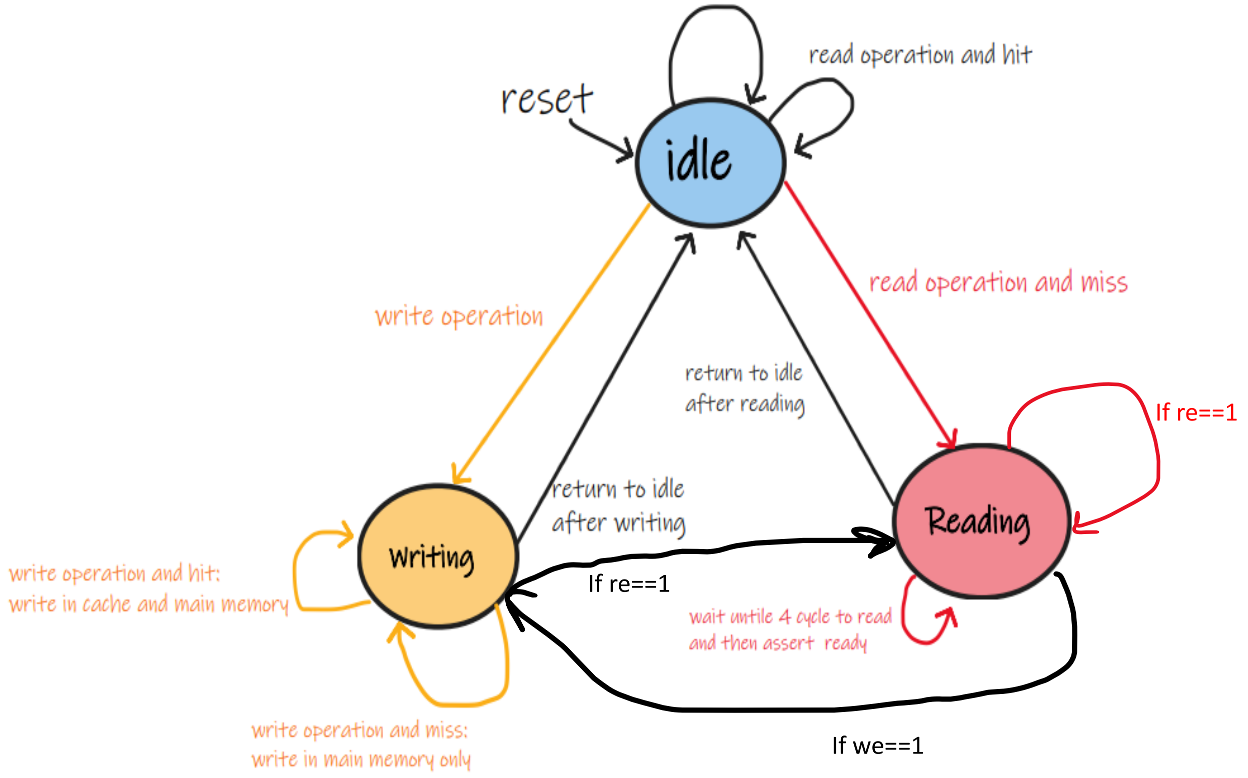


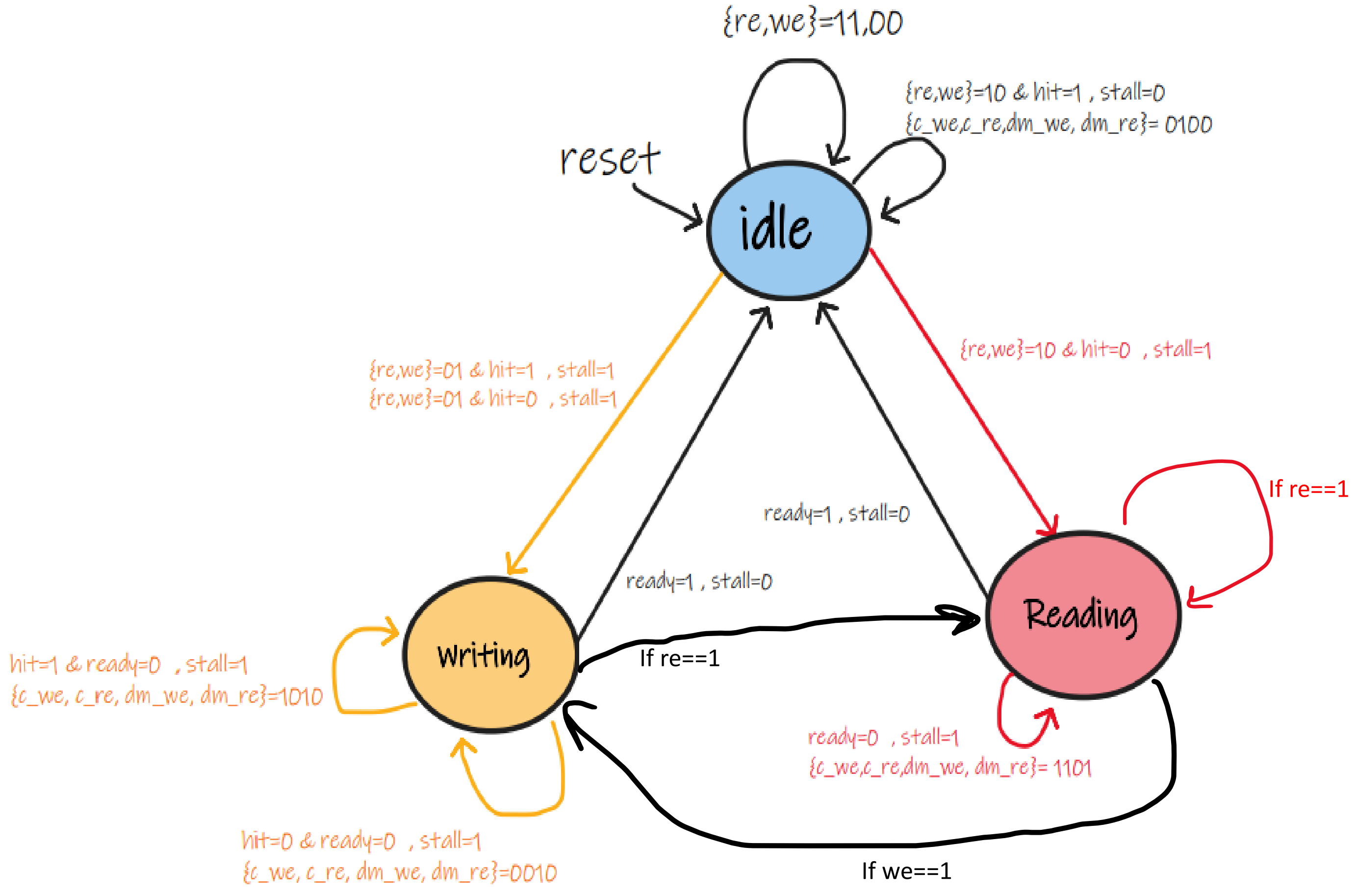






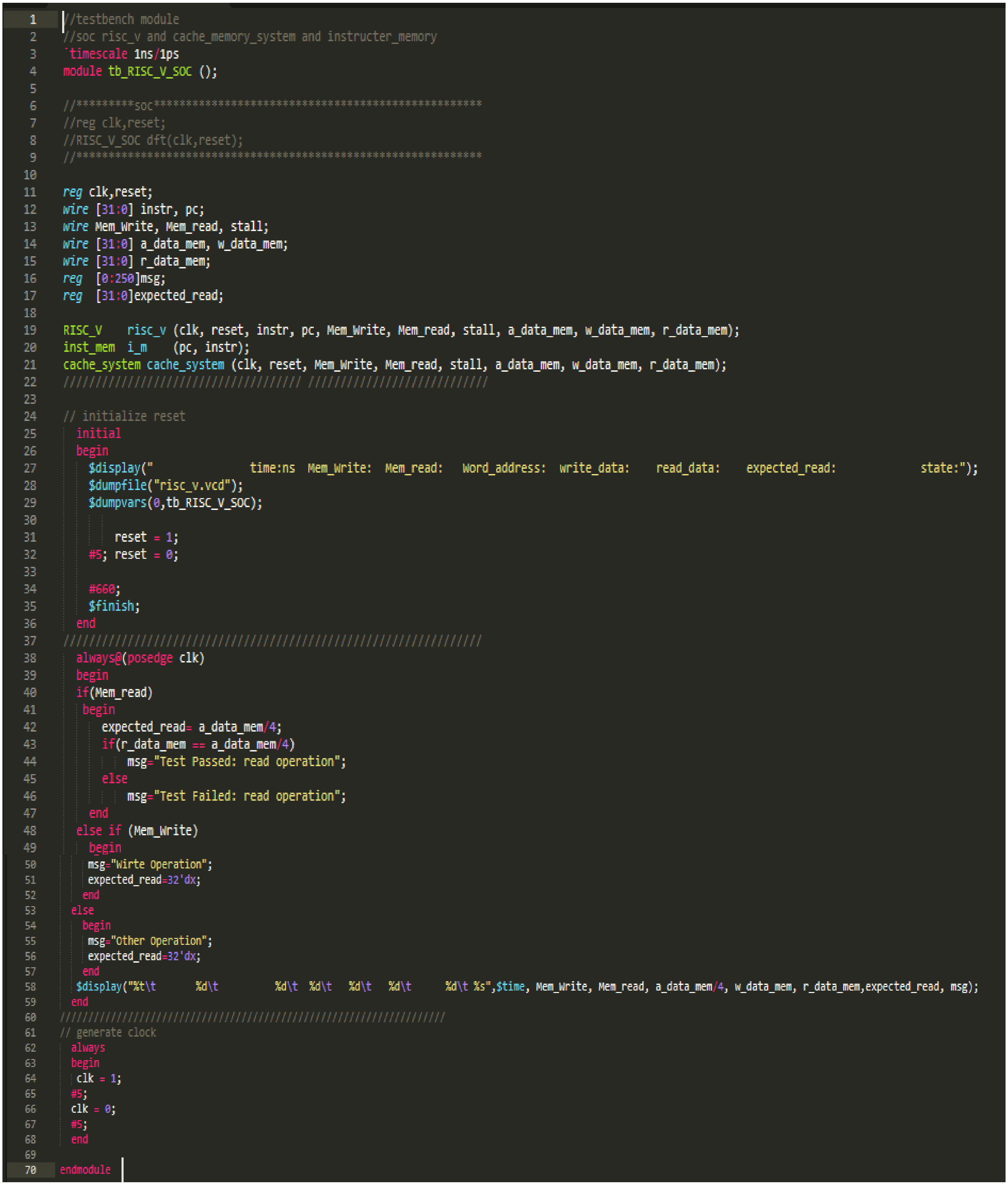
3. Cache Controller FSM:




{re,we}=11,00
{re,we}=10 & hit=1 , stall=0
{c_we,c_re,dm_we, dm_re}= 0100
reset
idle
{re,we}=01 & hit=1 , stall=1
{re,we}=01 & hit=0 , stall=1
{re,we}=10 & hit=0 , stall=1
ready=1 , stall=0
ready=1 , stall=0
If re==1
Writing
Reading
If re==1
hit=1 & ready=0 , stall=1
{c_we, c_re, dm_we, dm_re}=1010
ready=0 , stall=1
{c_we,c_re,dm_we, dm_re}= 1101
hit=0 & ready=0 , stall=1
{c_we, c_re, dm_we, dm_re}=0010
If we==1

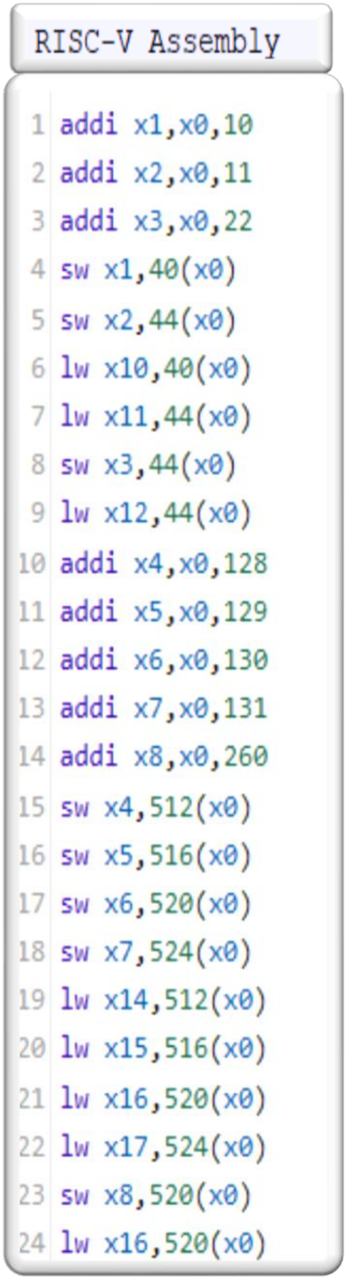
4. Test Bench:

❖ Test Bench Code:



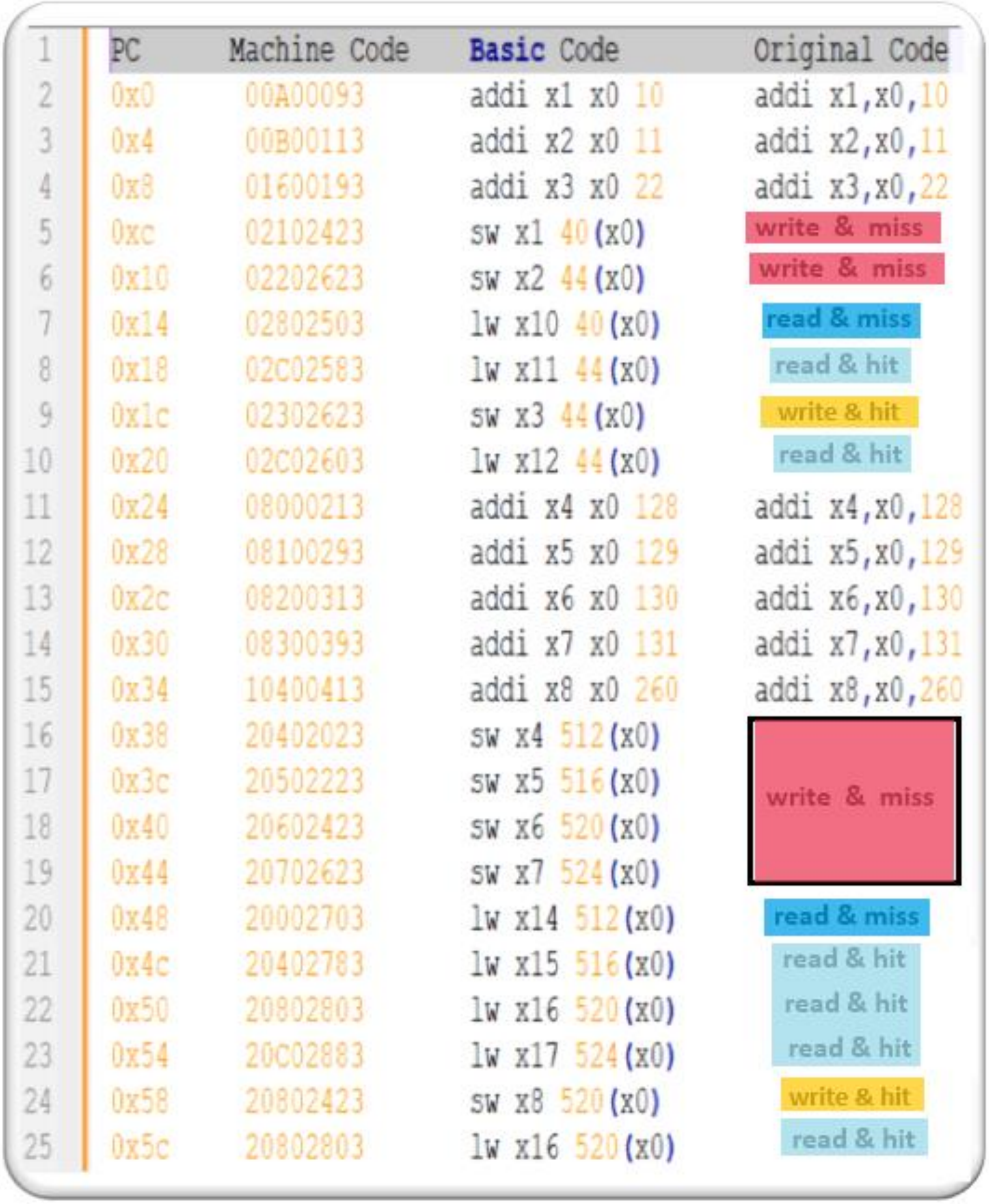
```
//testbench module
//soc risc_v and cache_memory_system and instructer_memory
`timescale 1ns/1ps
module tb_RISC_V_SOC ();

//*********soc**************************************************
//reg clk,reset;
//RISC_V_SOC dft(clk,reset);
//**************************************************************

reg clk,reset;
wire [31:0] instr, pc;
wire Mem_Write, Mem_read, stall;
wire [31:0] a_data_mem, w_data_mem;
wire [31:0] r_data_mem;
reg  [0:250]msg;
reg  [31:0]expected_read;

RISC_V    risc_v (clk, reset, instr, pc, Mem_Write, Mem_read, stall, a_data_mem, w_data_mem, r_data_mem);
inst_mem  i_m    (pc, instr);
cache_system cache_system (clk, reset, Mem_Write, Mem_read, stall, a_data_mem, w_data_mem, r_data_mem);
///////////////////////////////////// ////////////////////////////

// initialize reset
  initial
  begin
    $display("               time:ns  Mem_Write:  Mem_read:   Word_address:  write_data:    read_data:    expected_read:             state:");
    $dumpfile("risc_v.vcd");
    $dumpvars(0,tb_RISC_V_SOC);

        reset = 1;
    #5; reset = 0;

    #660;
    $finish;
  end
////////////////////////////////////////////////////////////////
  always@(posedge clk)
  begin
  if(Mem_read)
   begin
      expected_read= a_data_mem/4;
      if(r_data_mem == a_data_mem/4)
          msg="Test Passed: read operation";
      else
          msg="Test Failed: read operation";
    end
  else if (Mem_Write)
    begin
    msg="Wirte Operation";
    expected_read=32'dx;
   end
  else
    begin
     msg="Other Operation";
     expected_read=32'dx;
    end
   $display("%t\t      %d\t          %d\t  %d\t   %d\t   %d\t      %d\t %s",$time, Mem_Write, Mem_read, a_data_mem/4, w_data_mem, r_data_mem,expected_read, msg);
  end
/////////////////////////////////////////////////////////////////
// generate clock
  always
  begin
   clk = 1;
  #5;
  clk = 0;
  #5;
  end

endmodule
```

❖ Assembly Code Instructions:



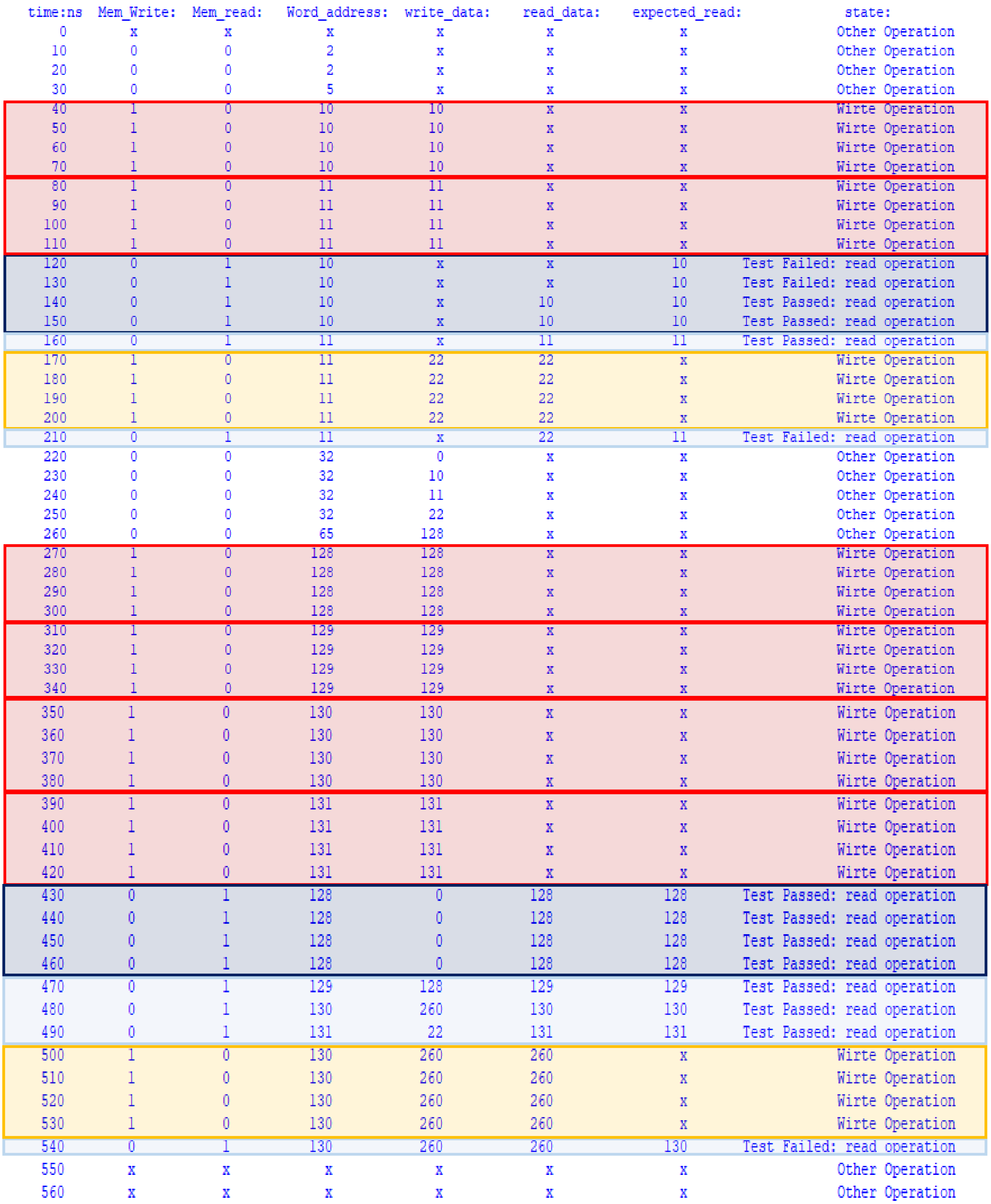
RISC-V Assembly

```
addi x1,x0,10
addi x2,x0,11
addi x3,x0,22
sw x1,40(x0)
sw x2,44(x0)
lw x10,40(x0)
lw x11,44(x0)
sw x3,44(x0)
lw x12,44(x0)
addi x4,x0,128
addi x5,x0,129
addi x6,x0,130
addi x7,x0,131
addi x8,x0,260
sw x4,512(x0)
sw x5,516(x0)
sw x6,520(x0)
sw x7,524(x0)
lw x14,512(x0)
lw x15,516(x0)
lw x16,520(x0)
lw x17,524(x0)
sw x8,520(x0)
lw x16,520(x0)
```

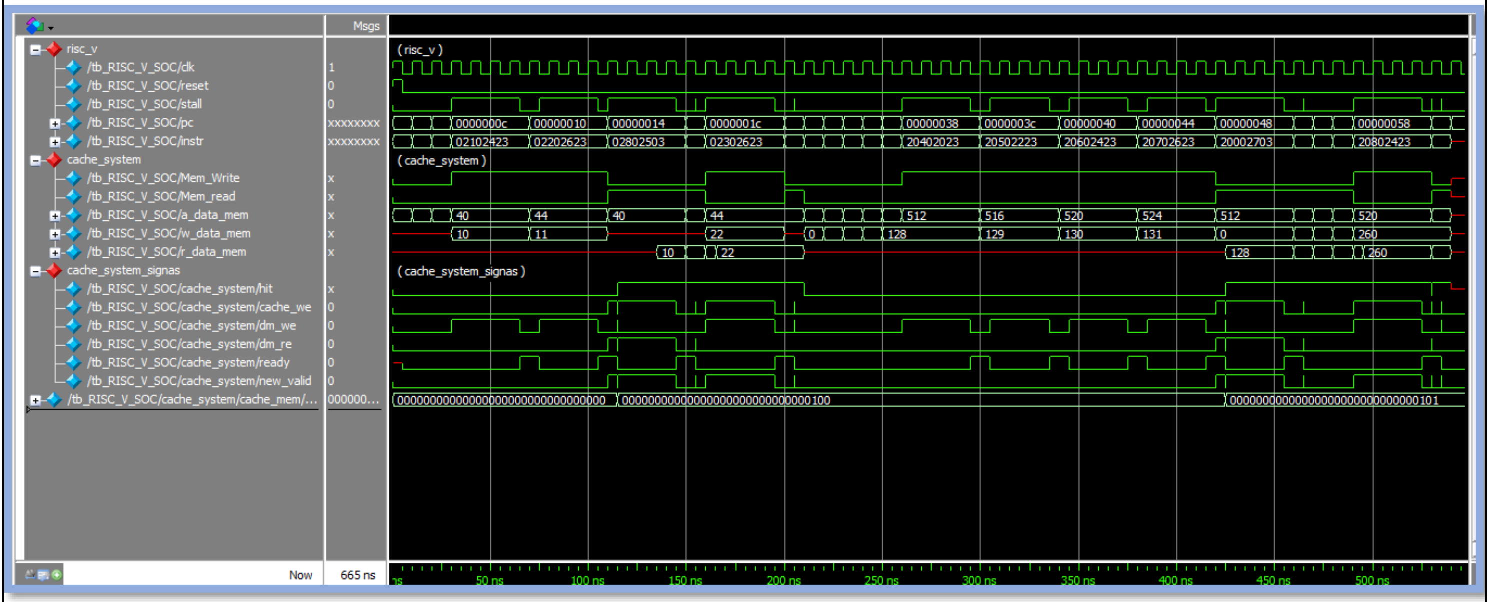


PC	Machine Code	Basic Code	Original Code
0x0	00A00093	addi x1 x0 10	addi x1,x0,10
0x4	00B00113	addi x2 x0 11	addi x2,x0,11
0x8	01600193	addi x3 x0 22	addi x3,x0,22
0xc	02102423	sw x1 40(x0)	write & miss
0x10	02202623	sw x2 44(x0)	write & miss
0x14	02802503	lw x10 40(x0)	read & miss
0x18	02C02583	lw x11 44(x0)	read & hit
0x1c	02302623	sw x3 44(x0)	write & hit
0x20	02C02603	lw x12 44(x0)	read & hit
0x24	08000213	addi x4 x0 128	addi x4,x0,128
0x28	08100293	addi x5 x0 129	addi x5,x0,129
0x2c	08200313	addi x6 x0 130	addi x6,x0,130
0x30	08300393	addi x7 x0 131	addi x7,x0,131
0x34	10400413	addi x8 x0 260	addi x8,x0,260
0x38	20402023	sw x4 512(x0)	write & miss
0x3c	20502223	sw x5 516(x0)	
0x40	20602423	sw x6 520(x0)	
0x44	20702623	sw x7 524(x0)	
0x48	20002703	lw x14 512(x0)	read & miss
0x4c	20402783	lw x15 516(x0)	read & hit
0x50	20802803	lw x16 520(x0)	read & hit
0x54	20C02883	lw x17 524(x0)	read & hit
0x58	20802423	sw x8 520(x0)	write & hit
0x5c	20802803	lw x16 520(x0)	read & hit

First Cache System Architecture Test Bench Results:

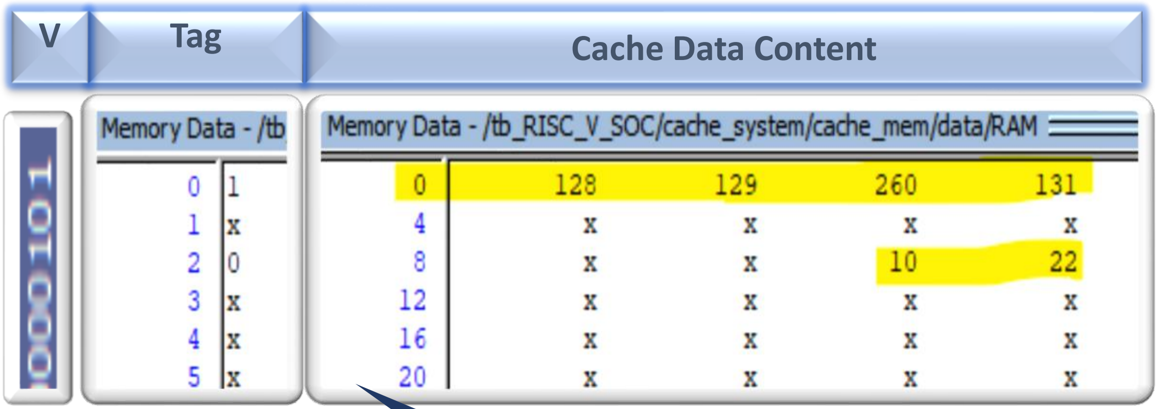


```
 time:ns  Mem_Write:  Mem_read:  Word_address:  write_data:   read_data:   expected_read:           state:
      0         x          x            x            x            x              x          Other Operation
     10         0          0            2            x            x              x          Other Operation
     20         0          0            2            x            x              x          Other Operation
     30         0          0            5            x            x              x          Other Operation
     40         1          0           10           10            x              x          Wirte Operation
     50         1          0           10           10            x              x          Wirte Operation
     60         1          0           10           10            x              x          Wirte Operation
     70         1          0           10           10            x              x          Wirte Operation
     80         1          0           11           11            x              x          Wirte Operation
     90         1          0           11           11            x              x          Wirte Operation
    100         1          0           11           11            x              x          Wirte Operation
    110         1          0           11           11            x              x          Wirte Operation
    120         0          1           10            x            x             10    Test Failed: read operation
    130         0          1           10            x            x             10    Test Failed: read operation
    140         0          1           10            x           10             10    Test Passed: read operation
    150         0          1           10            x           10             10    Test Passed: read operation
    160         0          1           11            x           11             11    Test Passed: read operation
    170         1          0           11           22           22              x          Wirte Operation
    180         1          0           11           22           22              x          Wirte Operation
    190         1          0           11           22           22              x          Wirte Operation
    200         1          0           11           22           22              x          Wirte Operation
    210         0          1           11            x           22             11    Test Failed: read operation
    220         0          0           32            0            x              x          Other Operation
    230         0          0           32           10            x              x          Other Operation
    240         0          0           32           11            x              x          Other Operation
    250         0          0           32           22            x              x          Other Operation
    260         0          0           65          128            x              x          Other Operation
    270         1          0          128          128            x              x          Wirte Operation
    280         1          0          128          128            x              x          Wirte Operation
    290         1          0          128          128            x              x          Wirte Operation
    300         1          0          128          128            x              x          Wirte Operation
    310         1          0          129          129            x              x          Wirte Operation
    320         1          0          129          129            x              x          Wirte Operation
    330         1          0          129          129            x              x          Wirte Operation
    340         1          0          129          129            x              x          Wirte Operation
    350         1          0          130          130            x              x          Wirte Operation
    360         1          0          130          130            x              x          Wirte Operation
    370         1          0          130          130            x              x          Wirte Operation
    380         1          0          130          130            x              x          Wirte Operation
    390         1          0          131          131            x              x          Wirte Operation
    400         1          0          131          131            x              x          Wirte Operation
    410         1          0          131          131            x              x          Wirte Operation
    420         1          0          131          131            x              x          Wirte Operation
    430         0          1          128            0          128            128    Test Passed: read operation
    440         0          1          128            0          128            128    Test Passed: read operation
    450         0          1          128            0          128            128    Test Passed: read operation
    460         0          1          128            0          128            128    Test Passed: read operation
    470         0          1          129          128          129            129    Test Passed: read operation
    480         0          1          130          260          130            130    Test Passed: read operation
    490         0          1          131           22          131            131    Test Passed: read operation
    500         1          0          130          260          260              x          Wirte Operation
    510         1          0          130          260          260              x          Wirte Operation
    520         1          0          130          260          260              x          Wirte Operation
    530         1          0          130          260          260              x          Wirte Operation
    540         0          1          130          260          260            130    Test Failed: read operation
    550         x          x            x            x            x              x          Other Operation
    560         x          x            x            x            x              x          Other Operation
```



	Msgs
risc_v	
/tb_RISC_V_SOC/clk	1
/tb_RISC_V_SOC/reset	0
/tb_RISC_V_SOC/stall	0
/tb_RISC_V_SOC/pc	xxxxxxxx
/tb_RISC_V_SOC/instr	xxxxxxxx
cache_system	
/tb_RISC_V_SOC/Mem_Write	x
/tb_RISC_V_SOC/Mem_read	x
/tb_RISC_V_SOC/a_data_mem	x
/tb_RISC_V_SOC/w_data_mem	x
/tb_RISC_V_SOC/r_data_mem	x
cache_system_signas	
/tb_RISC_V_SOC/cache_system/hit	x
/tb_RISC_V_SOC/cache_system/cache_we	0
/tb_RISC_V_SOC/cache_system/dm_we	0
/tb_RISC_V_SOC/cache_system/dm_re	0
/tb_RISC_V_SOC/cache_system/ready	0
/tb_RISC_V_SOC/cache_system/new_valid	0
/tb_RISC_V_SOC/cache_system/cache_mem/...	000000...
Now	665 ns

(risc_v)
0000000c
00000010
00000014
0000001c
00000038
0000003c
00000040
00000044
00000048
00000058
02102423
02202623
02802503
02302623
20402023
20502223
20602423
20702623
20002703
20802423
(cache_system)
40
44
40
44
512
516
520
524
512
520
10
11
22
0
128
129
130
131
0
260
10
22
128
260
(cache_system_signas)
00000000000000000000000000000000
00000000000000000000000000000100
00000000000000000000000000000101
ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
400 ns
450 ns
500 ns



V
0000101

Tag

Memory Data - /tb

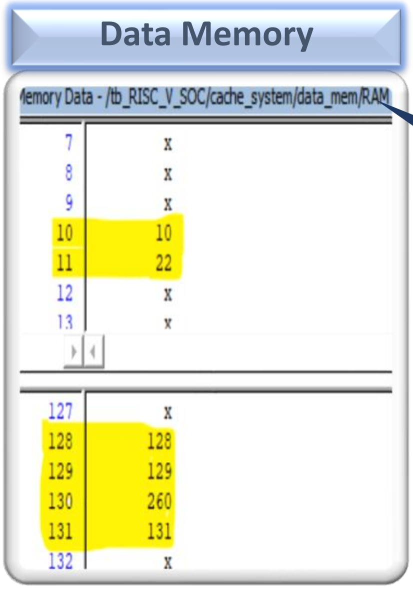
0	1
1	x
2	0
3	x
4	x
5	x

Cache Data Content

Memory Data - /tb_RISC_V_SOC/cache_system/cache_mem/data/RAM

0	128	129	260	131
4	x	x	x	x
8	x	x	10	22
12	x	x	x	x
16	x	x	x	x
20	x	x	x	x

Cache Memory
Word Address: **decimal**
data: **decimal**

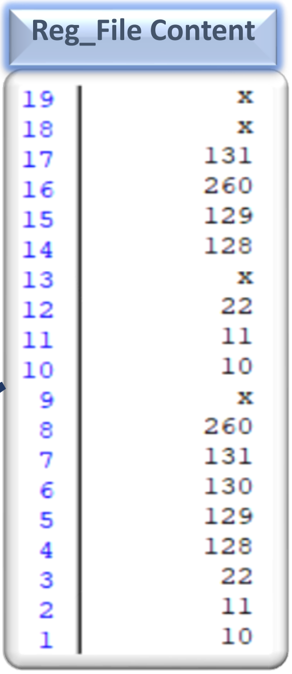


Data Memory

Memory Data - /tb_RISC_V_SOC/cache_system/data_mem/RAM

7	x
8	x
9	x
10	10
11	22
12	x
13	x
127	x
128	128
129	129
130	260
131	131
132	x

Data Memory
Word Address: **decimal**
data: **decimal**

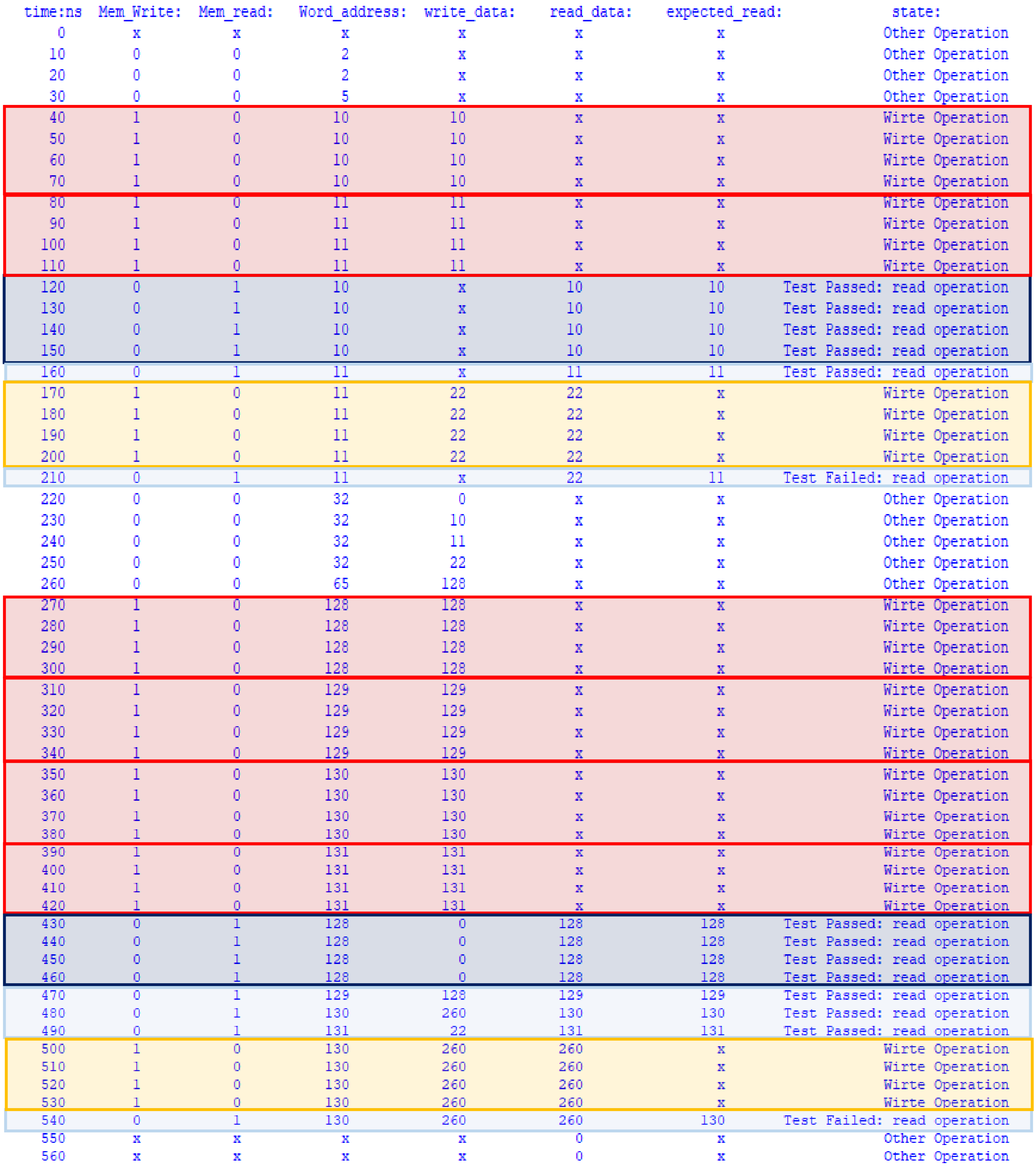


Reg_File Content

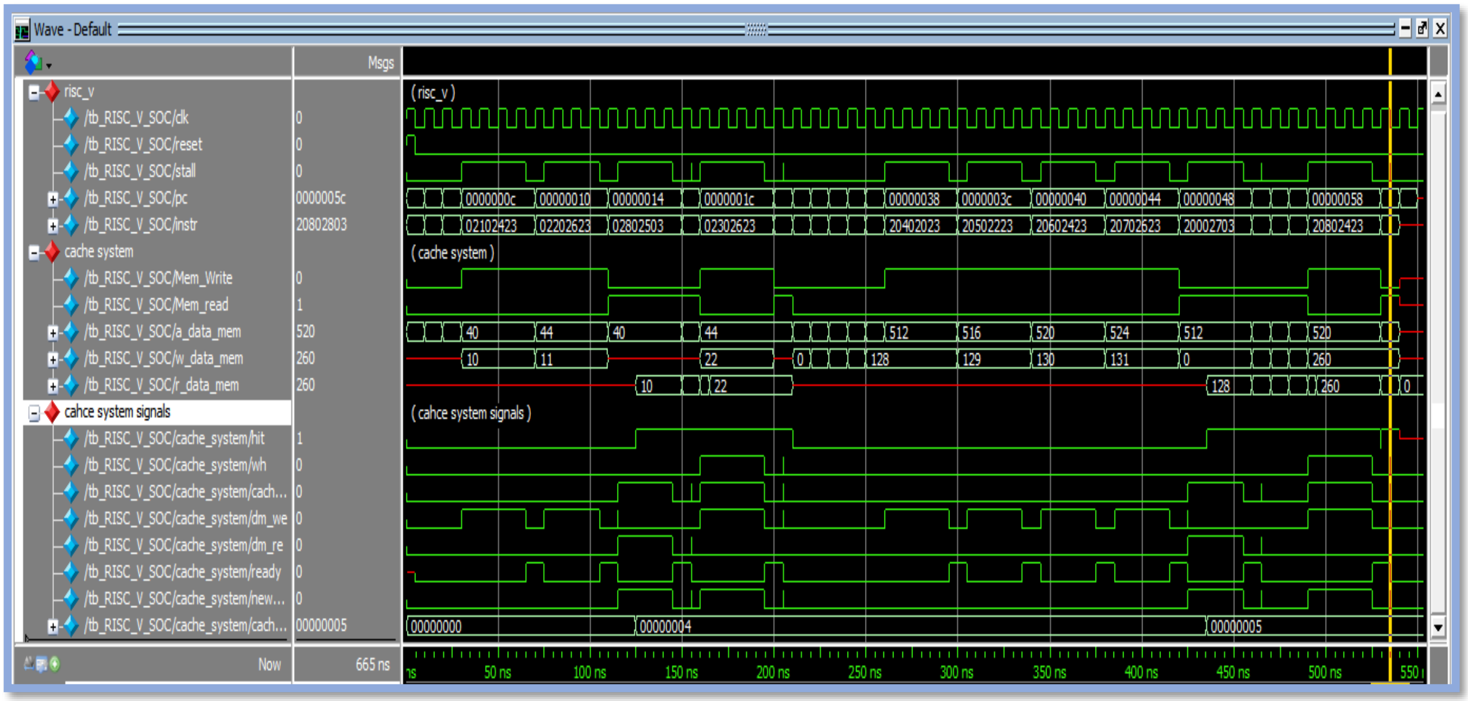
19	x
18	x
17	131
16	260
15	129
14	128
13	x
12	22
11	11
10	10
9	x
8	260
7	131
6	130
5	129
4	128
3	22
2	11
1	10

Reg File
Reg Address: **decimal**
data: **decimal**

❖ Second Cache System Architecture Test Bench Results:

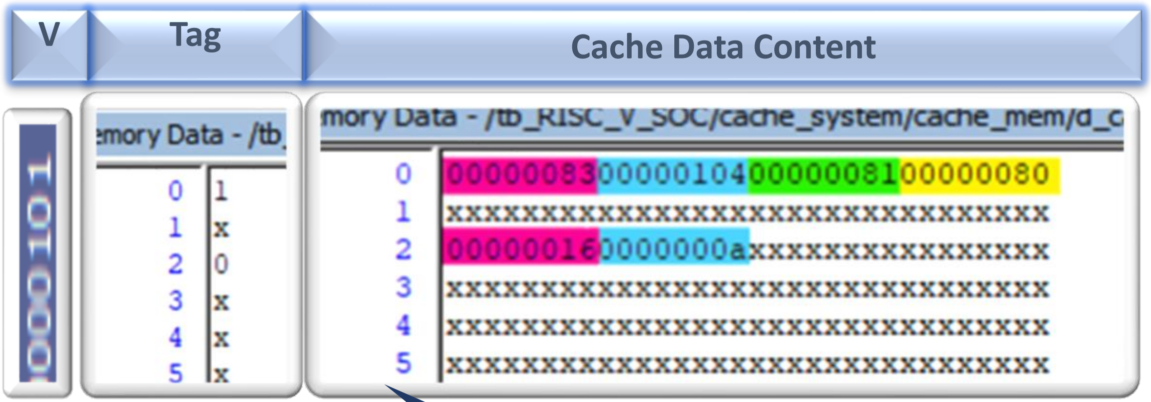


```
time:ns  Mem_Write:  Mem_read:  Word_address:  write_data:   read_data:   expected_read:           state:
   0        x          x            x             x             x             x              Other Operation
  10        0          0            2             x             x             x              Other Operation
  20        0          0            2             x             x             x              Other Operation
  30        0          0            5             x             x             x              Other Operation
  40        1          0           10            10             x             x              Wirte Operation
  50        1          0           10            10             x             x              Wirte Operation
  60        1          0           10            10             x             x              Wirte Operation
  70        1          0           10            10             x             x              Wirte Operation
  80        1          0           11            11             x             x              Wirte Operation
  90        1          0           11            11             x             x              Wirte Operation
 100        1          0           11            11             x             x              Wirte Operation
 110        1          0           11            11             x             x              Wirte Operation
 120        0          1           10             x            10            10        Test Passed: read operation
 130        0          1           10             x            10            10        Test Passed: read operation
 140        0          1           10             x            10            10        Test Passed: read operation
 150        0          1           10             x            10            10        Test Passed: read operation
 160        0          1           11             x            11            11        Test Passed: read operation
 170        1          0           11            22            22             x              Wirte Operation
 180        1          0           11            22            22             x              Wirte Operation
 190        1          0           11            22            22             x              Wirte Operation
 200        1          0           11            22            22             x              Wirte Operation
 210        0          1           11             x            22            11        Test Failed: read operation
 220        0          0           32             0             x             x              Other Operation
 230        0          0           32            10             x             x              Other Operation
 240        0          0           32            11             x             x              Other Operation
 250        0          0           32            22             x             x              Other Operation
 260        0          0           65           128             x             x              Other Operation
 270        1          0          128           128             x             x              Wirte Operation
 280        1          0          128           128             x             x              Wirte Operation
 290        1          0          128           128             x             x              Wirte Operation
 300        1          0          128           128             x             x              Wirte Operation
 310        1          0          129           129             x             x              Wirte Operation
 320        1          0          129           129             x             x              Wirte Operation
 330        1          0          129           129             x             x              Wirte Operation
 340        1          0          129           129             x             x              Wirte Operation
 350        1          0          130           130             x             x              Wirte Operation
 360        1          0          130           130             x             x              Wirte Operation
 370        1          0          130           130             x             x              Wirte Operation
 380        1          0          130           130             x             x              Wirte Operation
 390        1          0          131           131             x             x              Wirte Operation
 400        1          0          131           131             x             x              Wirte Operation
 410        1          0          131           131             x             x              Wirte Operation
 420        1          0          131           131             x             x              Wirte Operation
 430        0          1          128             0           128           128        Test Passed: read operation
 440        0          1          128             0           128           128        Test Passed: read operation
 450        0          1          128             0           128           128        Test Passed: read operation
 460        0          1          128             0           128           128        Test Passed: read operation
 470        0          1          129           128           129           129        Test Passed: read operation
 480        0          1          130           260           130           130        Test Passed: read operation
 490        0          1          131            22           131           131        Test Passed: read operation
 500        1          0          130           260           260             x              Wirte Operation
 510        1          0          130           260           260             x              Wirte Operation
 520        1          0          130           260           260             x              Wirte Operation
 530        1          0          130           260           260             x              Wirte Operation
 540        0          1          130           260           260           130        Test Failed: read operation
 550        x          x            x             x             0             x              Other Operation
 560        x          x            x             x             0             x              Other Operation
```



Wave - Default

```
                                              Msgs
risc_v
  /tb_RISC_V_SOC/clk                          0
  /tb_RISC_V_SOC/reset                        0
  /tb_RISC_V_SOC/stall                        0
  /tb_RISC_V_SOC/pc                           0000005c
  /tb_RISC_V_SOC/instr                        20802803
cache system
  /tb_RISC_V_SOC/Mem_Write                    0
  /tb_RISC_V_SOC/Mem_read                     1
  /tb_RISC_V_SOC/a_data_mem                   520
  /tb_RISC_V_SOC/w_data_mem                   260
  /tb_RISC_V_SOC/r_data_mem                   260
cahce system signals
  /tb_RISC_V_SOC/cache_system/hit             1
  /tb_RISC_V_SOC/cache_system/wh              0
  /tb_RISC_V_SOC/cache_system/cach...         0
  /tb_RISC_V_SOC/cache_system/dm_we           0
  /tb_RISC_V_SOC/cache_system/dm_re           0
  /tb_RISC_V_SOC/cache_system/ready           0
  /tb_RISC_V_SOC/cache_system/new...          0
  /tb_RISC_V_SOC/cache_system/cach...         00000005
Now                                           665 ns
```



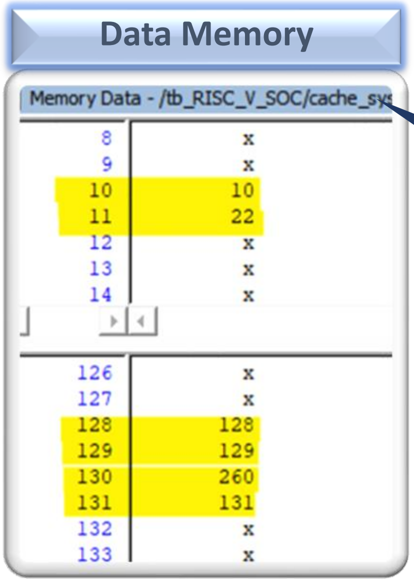
mory Data - /tb_

0	1
1	x
2	0
3	x
4	x
5	x

mory Data - /tb_RISC_V_SOC/cache_system/cache_mem/d_c

0	00000083000001040000008100000080
1	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2	000000160000000axxxxxxxxxxxxxxxx
3	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

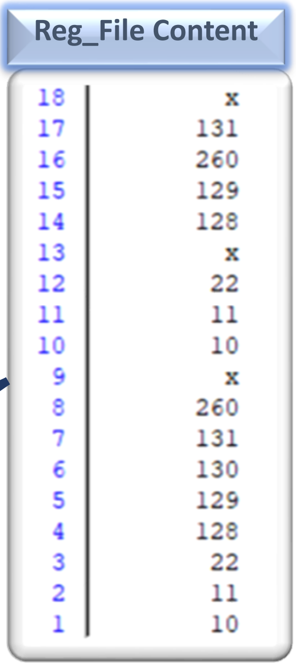
Cache Memory
Seat Address: **decimal**
data: **Hex**



Memory Data - /tb_RISC_V_SOC/cache_sys

8	x
9	x
10	10
11	22
12	x
13	x
14	x
126	x
127	x
128	128
129	129
130	260
131	131
132	x
133	x

Data Memory
Word Address: **decimal**
data: **decimal**

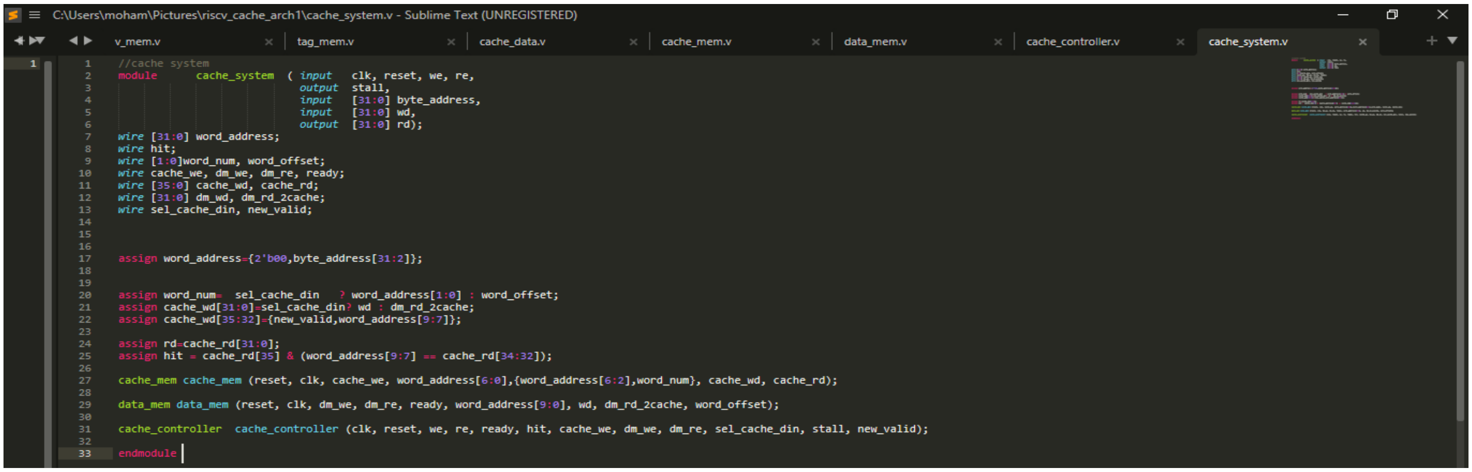


18	x
17	131
16	260
15	129
14	128
13	x
12	22
11	11
10	10
9	x
8	260
7	131
6	130
5	129
4	128
3	22
2	11
1	10

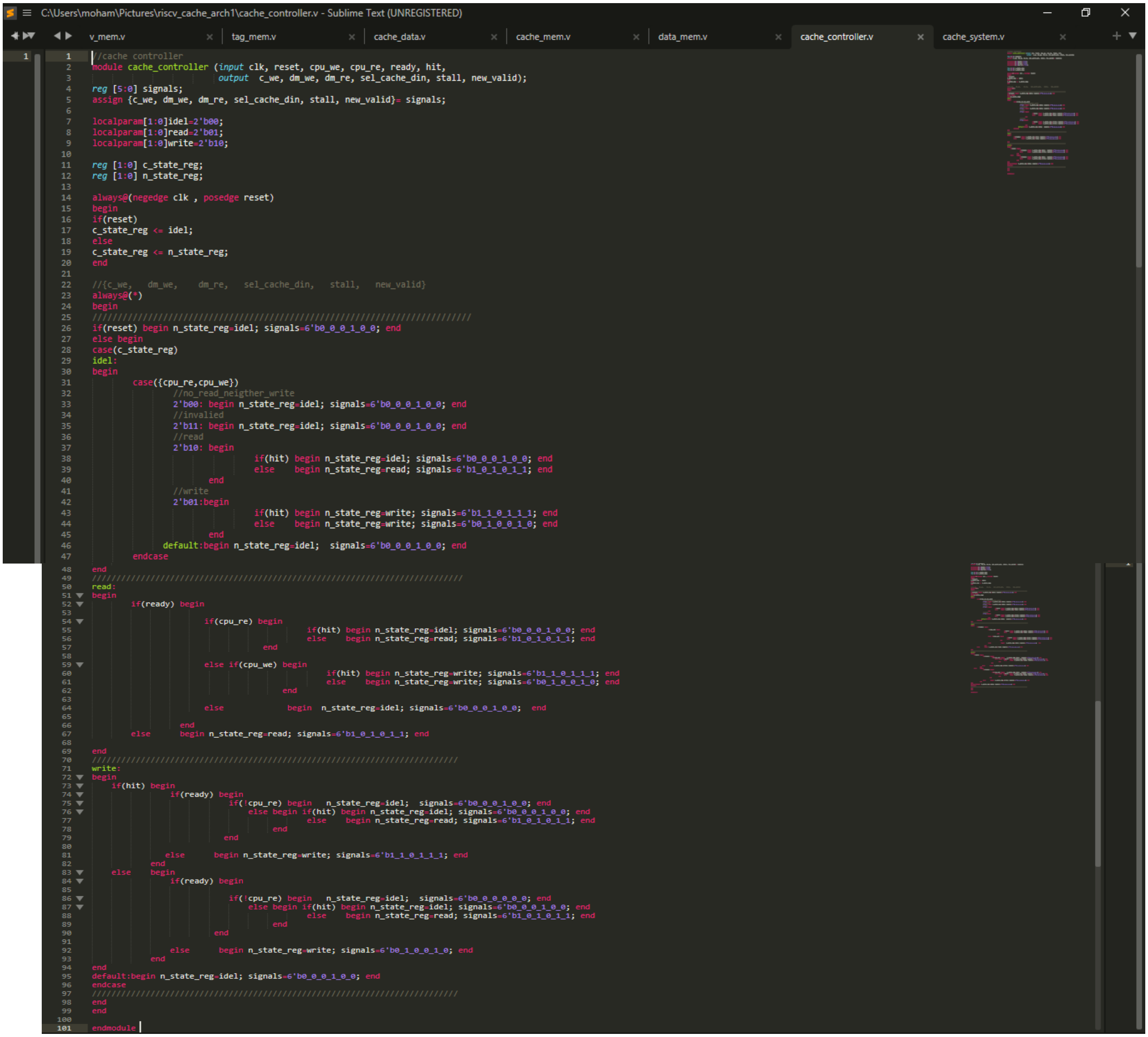
Reg File
Reg Address: **decimal**
data: **decimal**

5. Verilog Codes:

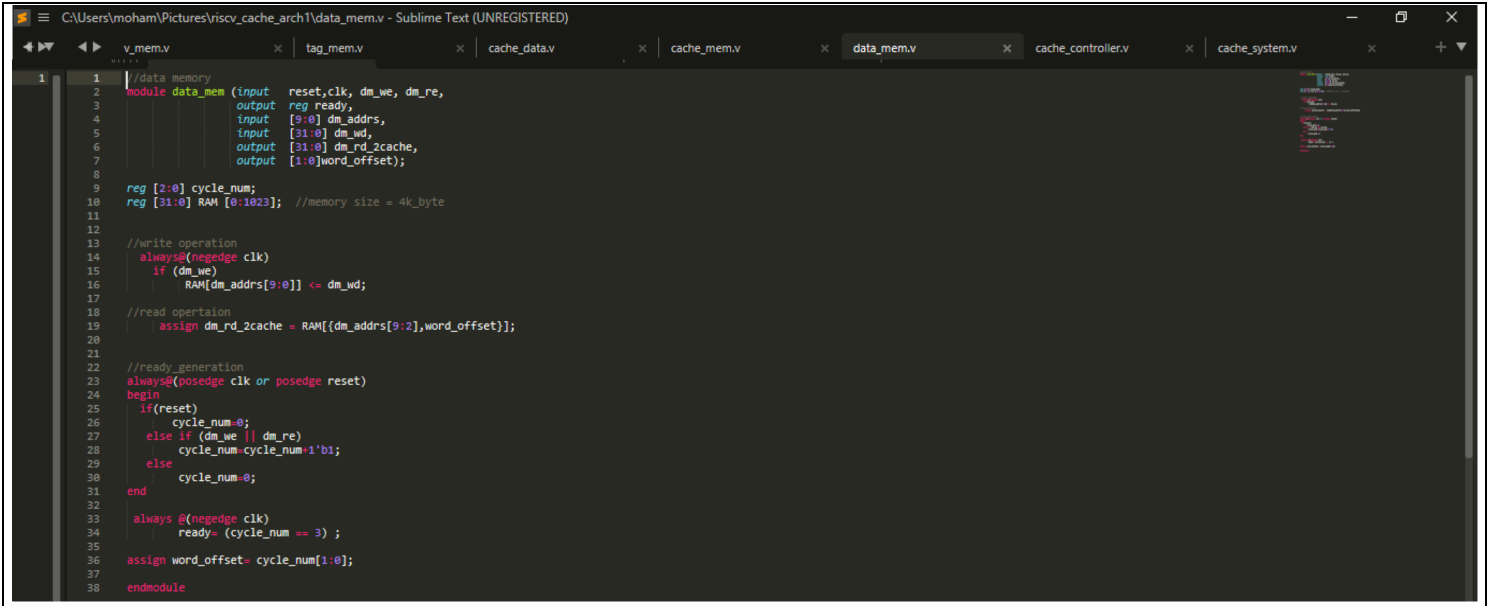
❖ First Cache System Architecture Verilog Codes:


C:\Users\moham\Pictures\riscv_cache_arch1\cache_system.v - Sublime Text (UNREGISTERED)
v_mem.v
tag_mem.v
cache_data.v
cache_mem.v
data_mem.v
cache_controller.v
cache_system.v

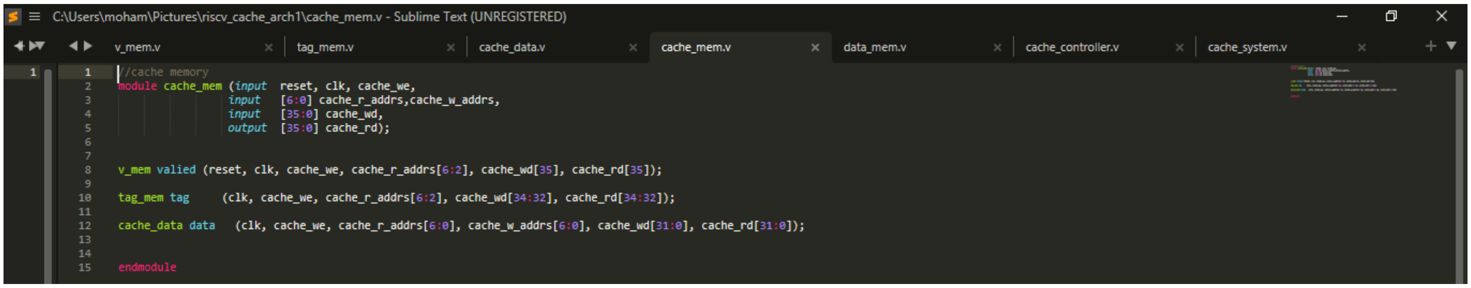
```verilog
//cache system
module      cache_system  ( input   clk, reset, we, re,
                            output  stall,
                            input   [31:0] byte_address,
                            input   [31:0] wd,
                            output  [31:0] rd);
wire [31:0] word_address;
wire hit;
wire [1:0]word_num, word_offset;
wire cache_we, dm_we, dm_re, ready;
wire [35:0] cache_wd, cache_rd;
wire [31:0] dm_wd, dm_rd_2cache;
wire sel_cache_din, new_valid;



assign word_address={2'b00,byte_address[31:2]};


assign word_num=  sel_cache_din   ? word_address[1:0] : word_offset;
assign cache_wd[31:0]=sel_cache_din? wd : dm_rd_2cache;
assign cache_wd[35:32]={new_valid,word_address[9:7]};

assign rd=cache_rd[31:0];
assign hit = cache_rd[35] & (word_address[9:7] == cache_rd[34:32]);

cache_mem cache_mem (reset, clk, cache_we, word_address[6:0],{word_address[6:2],word_num}, cache_wd, cache_rd);

data_mem data_mem (reset, clk, dm_we, dm_re, ready, word_address[9:0], wd, dm_rd_2cache, word_offset);

cache_controller  cache_controller (clk, reset, we, re, ready, hit, cache_we, dm_we, dm_re, sel_cache_din, stall, new_valid);

endmodule
```


C:\Users\moham\Pictures\riscv_cache_arch1\cache_controller.v - Sublime Text (UNREGISTERED)
v_mem.v
tag_mem.v
cache_data.v
cache_mem.v
data_mem.v
cache_controller.v
cache_system.v

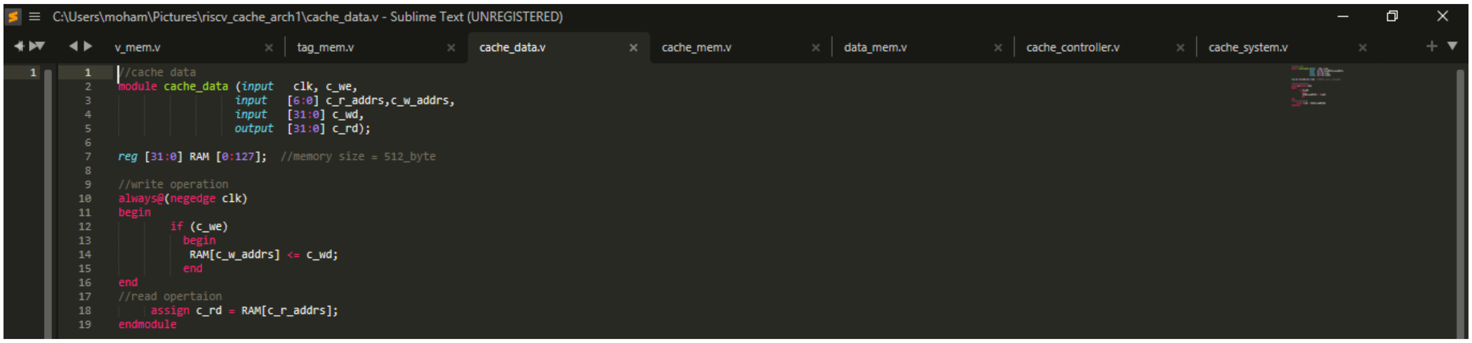
```verilog
//cache controller
module cache_controller (input clk, reset, cpu_we, cpu_re, ready, hit,
                         output  c_we, dm_we, dm_re, sel_cache_din, stall, new_valid);
reg [5:0] signals;
assign {c_we, dm_we, dm_re, sel_cache_din, stall, new_valid}= signals;

localparam[1:0]idel=2'b00;
localparam[1:0]read=2'b01;
localparam[1:0]write=2'b10;

reg [1:0] c_state_reg;
reg [1:0] n_state_reg;

always@(negedge clk , posedge reset)
begin
if(reset)
c_state_reg <= idel;
else
c_state_reg <= n_state_reg;
end

//{c_we,   dm_we,    dm_re,   sel_cache_din,   stall,   new_valid}
always@(*)
begin
///////////////////////////////////////////////////////////////////////////
if(reset) begin n_state_reg=idel; signals=6'b0_0_0_1_0_0; end
else begin
case(c_state_reg)
idel:
begin
        case({cpu_re,cpu_we})
                //no_read_neigther_write
                2'b00: begin n_state_reg=idel; signals=6'b0_0_0_1_0_0; end
                //invalied
                2'b11: begin n_state_reg=idel; signals=6'b0_0_0_1_0_0; end
                //read
                2'b10: begin
                                if(hit) begin n_state_reg=idel; signals=6'b0_0_0_1_0_0; end
                                else    begin n_state_reg=read; signals=6'b1_0_1_0_1_1; end
                       end
                //write
                2'b01:begin
                                if(hit) begin n_state_reg=write; signals=6'b1_1_0_1_1_1; end
                                else    begin n_state_reg=write; signals=6'b0_1_0_0_1_0; end
                       end
              default:begin n_state_reg=idel;  signals=6'b0_0_0_1_0_0; end
        endcase
end
//////////////////////////////////////////////////////////////////////////
read:
begin
        if(ready) begin

                        if(cpu_re) begin
                                             if(hit) begin n_state_reg=idel; signals=6'b0_0_0_1_0_0; end
                                             else    begin n_state_reg=read; signals=6'b1_0_1_0_1_1; end
                                    end

                        else if(cpu_we) begin
                                                 if(hit) begin n_state_reg=write; signals=6'b1_1_0_1_1_1; end
                                                 else    begin n_state_reg=write; signals=6'b0_1_0_0_1_0; end
                                        end

                        else             begin  n_state_reg=idel; signals=6'b0_0_0_1_0_0;  end

                  end
        else      begin n_state_reg=read; signals=6'b1_0_1_0_1_1; end

end
//////////////////////////////////////////////////////////////////////////
write:
begin
    if(hit) begin
                if(ready) begin
                             if(!cpu_re) begin   n_state_reg=idel;  signals=6'b0_0_0_1_0_0; end
                                 else begin if(hit) begin n_state_reg=idel; signals=6'b0_0_0_1_0_0; end
                                             else    begin n_state_reg=read; signals=6'b1_0_1_0_1_1; end
                                      end
                            end

               else      begin n_state_reg=write; signals=6'b1_1_0_1_1_1; end
            end
    else    begin
                if(ready) begin

                             if(!cpu_re) begin   n_state_reg=idel;  signals=6'b0_0_0_0_0_0; end
                                 else begin if(hit) begin n_state_reg=idel; signals=6'b0_0_0_1_0_0; end
                                             else    begin n_state_reg=read; signals=6'b1_0_1_0_1_1; end
                                      end
                          end

                else      begin n_state_reg=write; signals=6'b0_1_0_0_1_0; end
            end
end
default:begin n_state_reg=idel; signals=6'b0_0_0_1_0_0; end
endcase
//////////////////////////////////////////////////////////////////////////
end
end

endmodule
```


C:\Users\moham\Pictures\riscv_cache_arch1\data_mem.v - Sublime Text (UNREGISTERED)
v_mem.v
tag_mem.v
cache_data.v
cache_mem.v
data_mem.v
cache_controller.v
cache_system.v

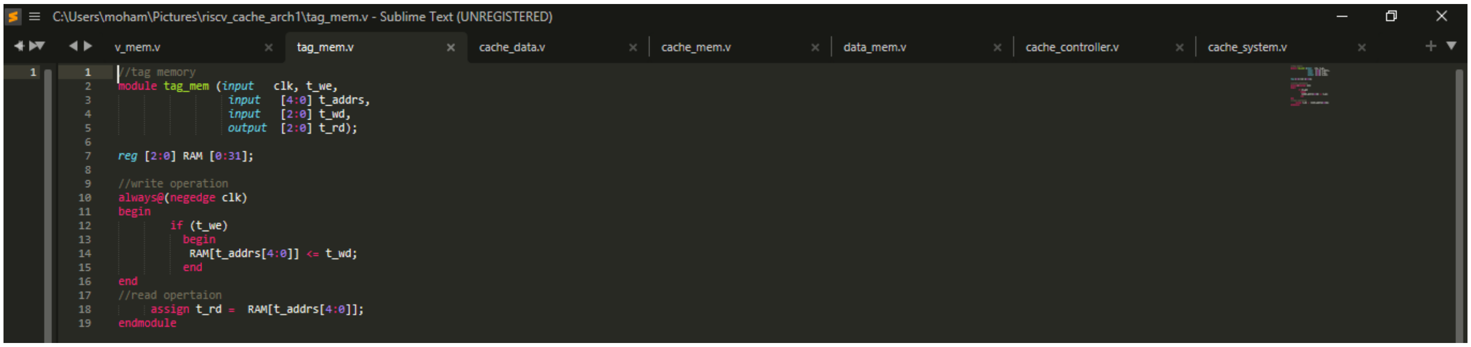
```
//data memory
module data_mem (input   reset,clk, dm_we, dm_re,
                output  reg ready,
                input   [9:0] dm_addrs,
                input   [31:0] dm_wd,
                output  [31:0] dm_rd_2cache,
                output  [1:0]word_offset);

reg [2:0] cycle_num;
reg [31:0] RAM [0:1023];  //memory size = 4k_byte


//write operation
  always@(negedge clk)
    if (dm_we)
         RAM[dm_addrs[9:0]] <= dm_wd;

//read opertaion
     assign dm_rd_2cache = RAM[{dm_addrs[9:2],word_offset}];


//ready_generation
always@(posedge clk or posedge reset)
begin
  if(reset)
       cycle_num=0;
   else if (dm_we || dm_re)
        cycle_num=cycle_num+1'b1;
   else
        cycle_num=0;
end

 always @(negedge clk)
        ready= (cycle_num == 3) ;

assign word_offset= cycle_num[1:0];

endmodule
```


C:\Users\moham\Pictures\riscv_cache_arch1\cache_mem.v - Sublime Text (UNREGISTERED)
v_mem.v
tag_mem.v
cache_data.v
cache_mem.v
data_mem.v
cache_controller.v
cache_system.v

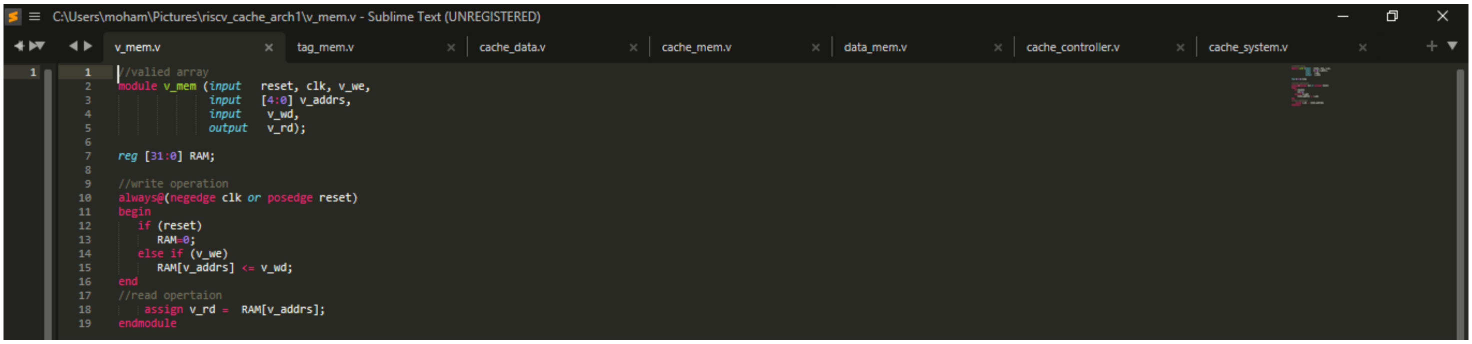
```
//cache memory
module cache_mem (input  reset, clk, cache_we,
                 input   [6:0] cache_r_addrs,cache_w_addrs,
                 input   [35:0] cache_wd,
                 output  [35:0] cache_rd);


v_mem valied (reset, clk, cache_we, cache_r_addrs[6:2], cache_wd[35], cache_rd[35]);

tag_mem tag     (clk, cache_we, cache_r_addrs[6:2], cache_wd[34:32], cache_rd[34:32]);

cache_data data   (clk, cache_we, cache_r_addrs[6:0], cache_w_addrs[6:0], cache_wd[31:0], cache_rd[31:0]);


endmodule
```


C:\Users\moham\Pictures\riscv_cache_arch1\cache_data.v - Sublime Text (UNREGISTERED)
v_mem.v
tag_mem.v
cache_data.v
cache_mem.v
data_mem.v
cache_controller.v
cache_system.v

```
//cache data
module cache_data (input   clk, c_we,
                  input   [6:0] c_r_addrs,c_w_addrs,
                  input   [31:0] c_wd,
                  output  [31:0] c_rd);

reg [31:0] RAM [0:127];  //memory size = 512_byte

//write operation
always@(negedge clk)
begin
        if (c_we)
          begin
           RAM[c_w_addrs] <= c_wd;
          end
end
//read opertaion
     assign c_rd = RAM[c_r_addrs];
endmodule
```

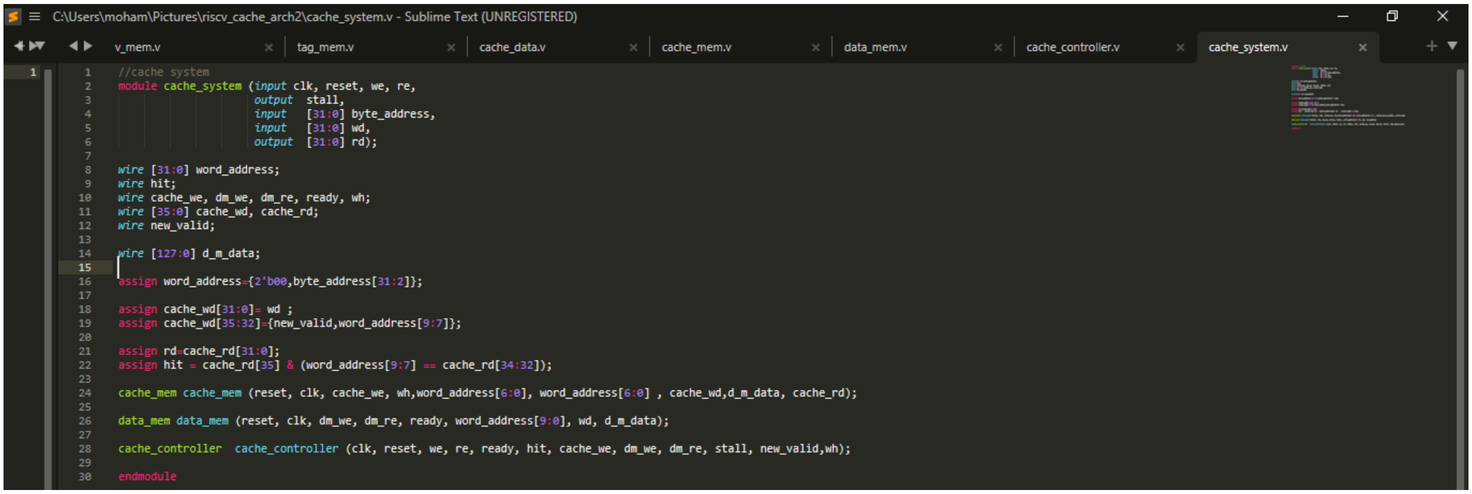


```
//tag memory
module tag_mem (input   clk, t_we,
               input   [4:0] t_addrs,
               input   [2:0] t_wd,
               output  [2:0] t_rd);

reg [2:0] RAM [0:31];

//write operation
always@(negedge clk)
begin
        if (t_we)
          begin
           RAM[t_addrs[4:0]] <= t_wd;
          end
end
//read opertaion
     assign t_rd =  RAM[t_addrs[4:0]];
endmodule
```



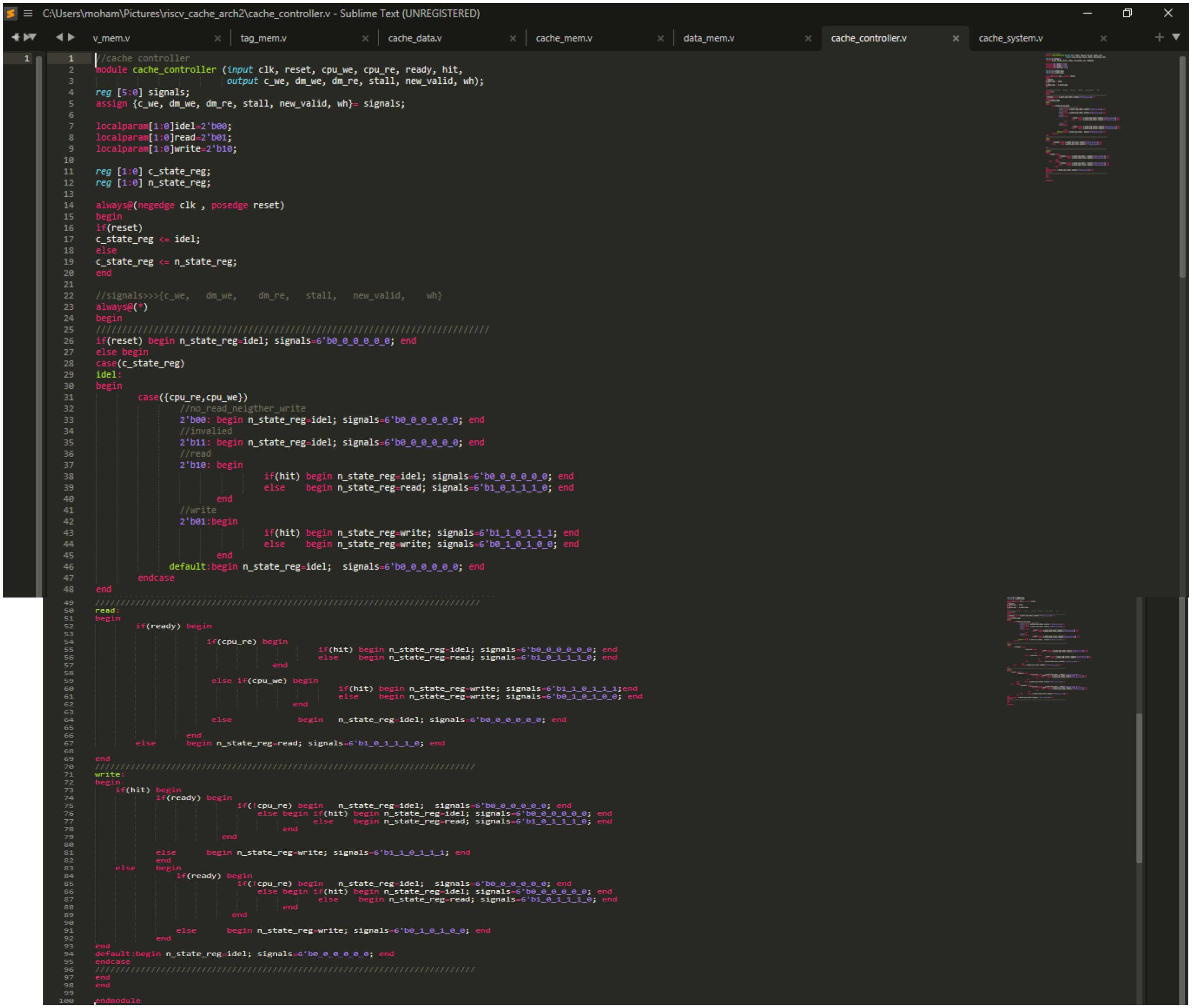
```
//valied array
module v_mem (input   reset, clk, v_we,
             input   [4:0] v_addrs,
             input    v_wd,
             output   v_rd);

reg [31:0] RAM;

//write operation
always@(negedge clk or posedge reset)
begin
   if (reset)
      RAM=0;
   else if (v_we)
      RAM[v_addrs] <= v_wd;
end
//read opertaion
    assign v_rd =  RAM[v_addrs];
endmodule
```

❖ Second Cache System Architecture Verilog Codes:



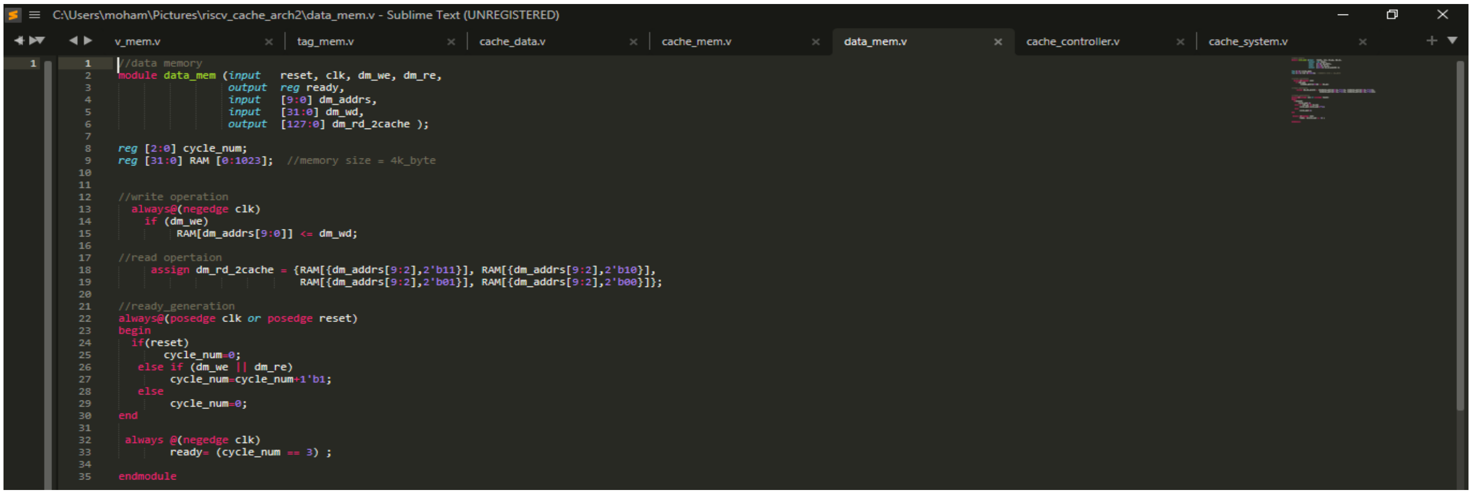
C:\Users\moham\Pictures\riscv_cache_arch2\cache_system.v - Sublime Text (UNREGISTERED)
v_mem.v
tag_mem.v
cache_data.v
cache_mem.v
data_mem.v
cache_controller.v
cache_system.v

```verilog
//cache system
module cache_system (input clk, reset, we, re,
                     output stall,
                     input  [31:0] byte_address,
                     input  [31:0] wd,
                     output [31:0] rd);

wire [31:0] word_address;
wire hit;
wire cache_we, dm_we, dm_re, ready, wh;
wire [35:0] cache_wd, cache_rd;
wire new_valid;

wire [127:0] d_m_data;

assign word_address={2'b00,byte_address[31:2]};

assign cache_wd[31:0]= wd ;
assign cache_wd[35:32]={new_valid,word_address[9:7]};

assign rd=cache_rd[31:0];
assign hit = cache_rd[35] & (word_address[9:7] == cache_rd[34:32]);

cache_mem cache_mem (reset, clk, cache_we, wh,word_address[6:0], word_address[6:0] , cache_wd,d_m_data, cache_rd);

data_mem data_mem (reset, clk, dm_we, dm_re, ready, word_address[9:0], wd, d_m_data);

cache_controller  cache_controller (clk, reset, we, re, ready, hit, cache_we, dm_we, dm_re, stall, new_valid,wh);

endmodule
```

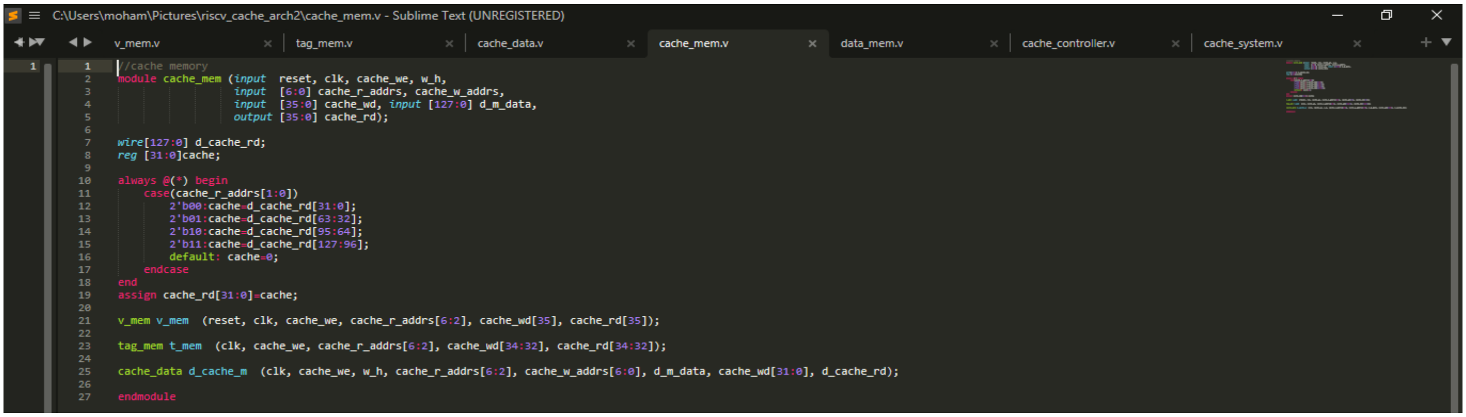


C:\Users\moham\Pictures\riscv_cache_arch2\cache_controller.v - Sublime Text (UNREGISTERED)
v_mem.v
tag_mem.v
cache_data.v
cache_mem.v
data_mem.v
cache_controller.v
cache_system.v

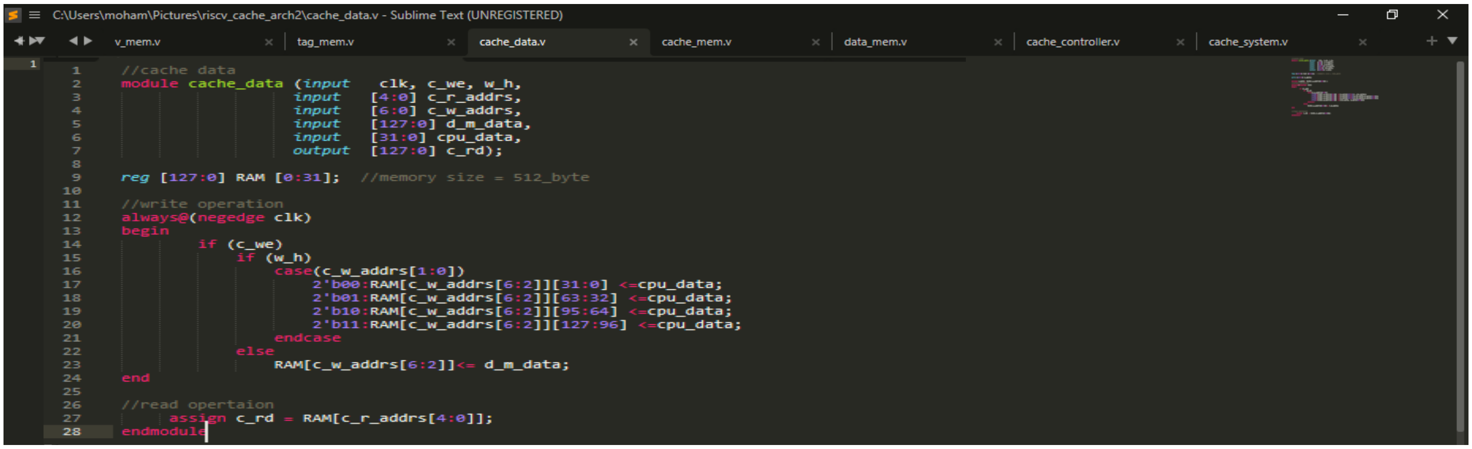
```verilog
//cache controller
module cache_controller (input clk, reset, cpu_we, cpu_re, ready, hit,
                         output c_we, dm_we, dm_re, stall, new_valid, wh);
reg [5:0] signals;
assign {c_we, dm_we, dm_re, stall, new_valid, wh}= signals;

localparam[1:0]idel=2'b00;
localparam[1:0]read=2'b01;
localparam[1:0]write=2'b10;

reg [1:0] c_state_reg;
reg [1:0] n_state_reg;

always@(negedge clk , posedge reset)
begin
if(reset)
c_state_reg <= idel;
else
c_state_reg <= n_state_reg;
end

//signals>>>{c_we,   dm_we,    dm_re,   stall,   new_valid,    wh}
always@(*)
begin
/////////////////////////////////////////////////////////////////////////////
if(reset) begin n_state_reg=idel; signals=6'b0_0_0_0_0_0; end
else begin
case(c_state_reg)
idel:
begin
        case({cpu_re,cpu_we})
                //no_read_neigther_write
                2'b00: begin n_state_reg=idel; signals=6'b0_0_0_0_0_0; end
                //invalied
                2'b11: begin n_state_reg=idel; signals=6'b0_0_0_0_0_0; end
                //read
                2'b10: begin
                                if(hit) begin n_state_reg=idel; signals=6'b0_0_0_0_0_0; end
                                else    begin n_state_reg=read; signals=6'b1_0_1_1_1_0; end
                        end
                //write
                2'b01:begin
                                if(hit) begin n_state_reg=write; signals=6'b1_1_0_1_1_1; end
                                else    begin n_state_reg=write; signals=6'b0_1_0_1_0_0; end
                        end
              default:begin n_state_reg=idel;  signals=6'b0_0_0_0_0_0; end
        endcase
end
/////////////////////////////////////////////////////////////////////////////
read:
begin
        if(ready) begin

                    if(cpu_re) begin
                                     if(hit) begin n_state_reg=idel; signals=6'b0_0_0_0_0_0; end
                                     else    begin n_state_reg=read; signals=6'b1_0_1_1_1_0; end
                            end

                     else if(cpu_we) begin
                                        if(hit) begin n_state_reg=write; signals=6'b1_1_0_1_1_1;end
                                        else    begin n_state_reg=write; signals=6'b0_1_0_1_0_0; end
                                end

                     else                begin   n_state_reg=idel; signals=6'b0_0_0_0_0_0; end

                end
        else      begin n_state_reg=read; signals=6'b1_0_1_1_1_0; end

end
/////////////////////////////////////////////////////////////////////////////
write:
begin
    if(hit) begin
          if(ready) begin
                      if(!cpu_re) begin   n_state_reg=idel;  signals=6'b0_0_0_0_0_0; end
                         else begin if(hit) begin n_state_reg=idel; signals=6'b0_0_0_0_0_0; end
                                    else    begin n_state_reg=read; signals=6'b1_0_1_1_1_0; end
                                 end
                    end

          else      begin n_state_reg=write; signals=6'b1_1_0_1_1_1; end
          end
    else    begin
              if(ready) begin
                      if(!cpu_re) begin   n_state_reg=idel;  signals=6'b0_0_0_0_0_0; end
                         else begin if(hit) begin n_state_reg=idel; signals=6'b0_0_0_0_0_0; end
                                     else    begin n_state_reg=read; signals=6'b1_0_1_1_1_0; end
                                 end
                       end

              else      begin n_state_reg=write; signals=6'b0_1_0_1_0_0; end
          end
end
default:begin n_state_reg=idel; signals=6'b0_0_0_0_0_0; end
endcase
/////////////////////////////////////////////////////////////////////////////
end
end

endmodule
```


C:\Users\moham\Pictures\riscv_cache_arch2\data_mem.v - Sublime Text (UNREGISTERED)
v_mem.v
tag_mem.v
cache_data.v
cache_mem.v
data_mem.v
cache_controller.v
cache_system.v

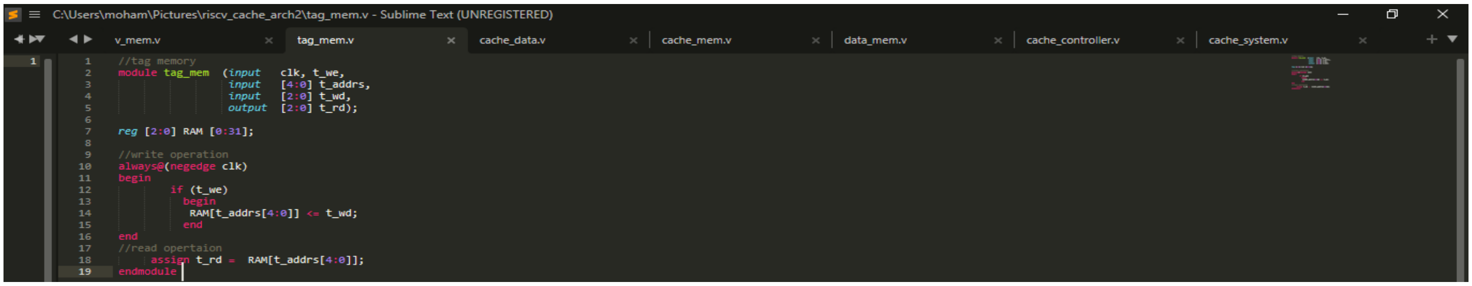
```
//data memory
module data_mem (input   reset, clk, dm_we, dm_re,
                 output  reg ready,
                 input   [9:0] dm_addrs,
                 input   [31:0] dm_wd,
                 output  [127:0] dm_rd_2cache );

reg [2:0] cycle_num;
reg [31:0] RAM [0:1023];  //memory size = 4k_byte


//write operation
  always@(negedge clk)
    if (dm_we)
         RAM[dm_addrs[9:0]] <= dm_wd;

//read opertaion
     assign dm_rd_2cache = {RAM[{dm_addrs[9:2],2'b11}], RAM[{dm_addrs[9:2],2'b10}],
                            RAM[{dm_addrs[9:2],2'b01}], RAM[{dm_addrs[9:2],2'b00}]};

//ready_generation
always@(posedge clk or posedge reset)
begin
  if(reset)
       cycle_num=0;
   else if (dm_we || dm_re)
        cycle_num=cycle_num+1'b1;
   else
        cycle_num=0;
end

 always @(negedge clk)
        ready= (cycle_num == 3) ;

endmodule
```


C:\Users\moham\Pictures\riscv_cache_arch2\cache_mem.v - Sublime Text (UNREGISTERED)
v_mem.v
tag_mem.v
cache_data.v
cache_mem.v
data_mem.v
cache_controller.v
cache_system.v

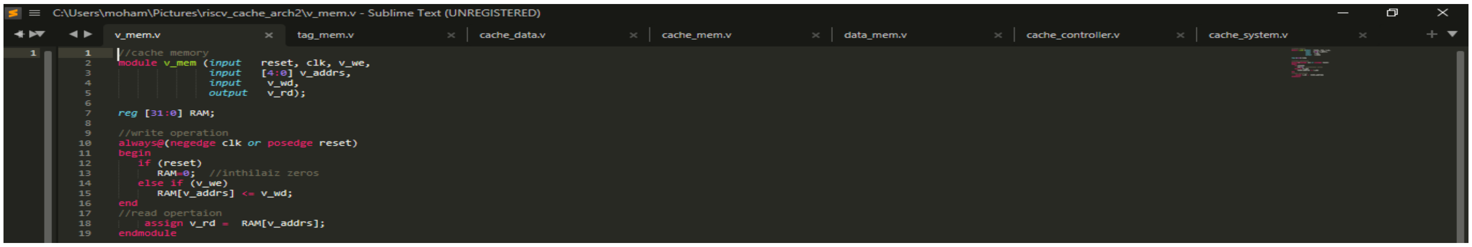
```
//cache memory
module cache_mem (input  reset, clk, cache_we, w_h,
                  input  [6:0] cache_r_addrs, cache_w_addrs,
                  input  [35:0] cache_wd, input [127:0] d_m_data,
                  output [35:0] cache_rd);

wire[127:0] d_cache_rd;
reg [31:0]cache;

always @(*) begin
    case(cache_r_addrs[1:0])
        2'b00:cache=d_cache_rd[31:0];
        2'b01:cache=d_cache_rd[63:32];
        2'b10:cache=d_cache_rd[95:64];
        2'b11:cache=d_cache_rd[127:96];
        default: cache=0;
    endcase
end
assign cache_rd[31:0]=cache;

v_mem v_mem  (reset, clk, cache_we, cache_r_addrs[6:2], cache_wd[35], cache_rd[35]);

tag_mem t_mem  (clk, cache_we, cache_r_addrs[6:2], cache_wd[34:32], cache_rd[34:32]);

cache_data d_cache_m  (clk, cache_we, w_h, cache_r_addrs[6:2], cache_w_addrs[6:0], d_m_data, cache_wd[31:0], d_cache_rd);

endmodule
```


C:\Users\moham\Pictures\riscv_cache_arch2\cache_data.v - Sublime Text (UNREGISTERED)
v_mem.v
tag_mem.v
cache_data.v
cache_mem.v
data_mem.v
cache_controller.v
cache_system.v

```
//cache data
module cache_data (input   clk, c_we, w_h,
                   input   [4:0] c_r_addrs,
                   input   [6:0] c_w_addrs,
                   input   [127:0] d_m_data,
                   input   [31:0] cpu_data,
                   output  [127:0] c_rd);

reg [127:0] RAM [0:31];  //memory size = 512_byte

//write operation
always@(negedge clk)
begin
        if (c_we)
            if (w_h)
                case(c_w_addrs[1:0])
                    2'b00:RAM[c_w_addrs[6:2]][31:0] <=cpu_data;
                    2'b01:RAM[c_w_addrs[6:2]][63:32] <=cpu_data;
                    2'b10:RAM[c_w_addrs[6:2]][95:64] <=cpu_data;
                    2'b11:RAM[c_w_addrs[6:2]][127:96] <=cpu_data;
                endcase
            else
                RAM[c_w_addrs[6:2]]<= d_m_data;
end

//read opertaion
     assign c_rd = RAM[c_r_addrs[4:0]];
endmodule
```


C:\Users\moham\Pictures\riscv_cache_arch2\tag_mem.v - Sublime Text (UNREGISTERED)
v_mem.v
tag_mem.v
cache_data.v
cache_mem.v
data_mem.v
cache_controller.v
cache_system.v

```
//tag memory
module tag_mem  (input   clk, t_we,
                 input   [4:0] t_addrs,
                 input   [2:0] t_wd,
                 output  [2:0] t_rd);

reg [2:0] RAM [0:31];

//write operation
always@(negedge clk)
begin
        if (t_we)
          begin
           RAM[t_addrs[4:0]] <= t_wd;
          end
end
//read opertaion
     assign t_rd =  RAM[t_addrs[4:0]];
endmodule
```



```
//cache memory
module v_mem (input   reset, clk, v_we,
              input   [4:0] v_addrs,
              input    v_wd,
              output   v_rd);

reg [31:0] RAM;

//write operation
always@(negedge clk or posedge reset)
begin
   if (reset)
      RAM=0;  //inthilaiz zeros
   else if (v_we)
      RAM[v_addrs] <= v_wd;
end
//read opertaion
    assign v_rd =  RAM[v_addrs];
endmodule
```