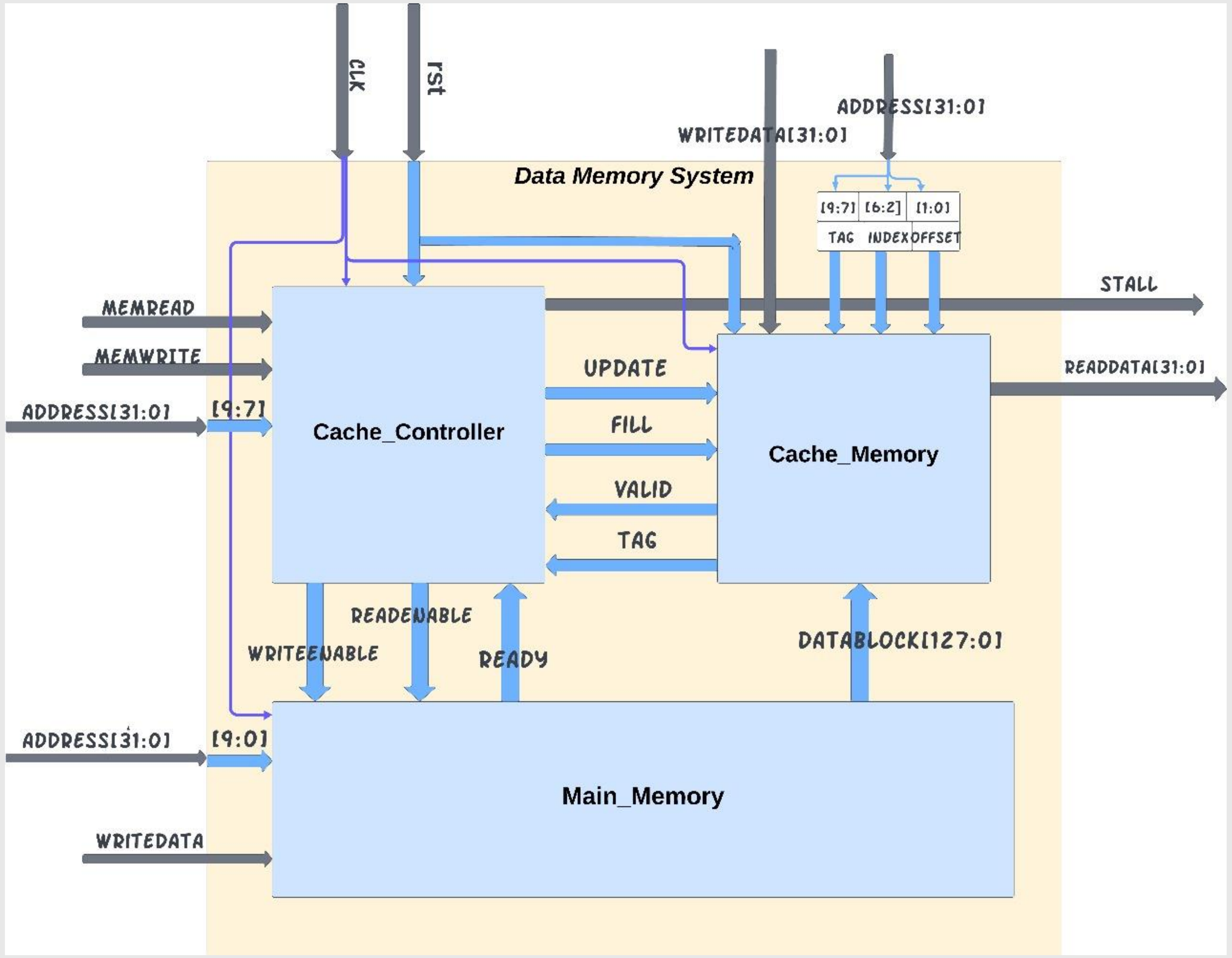
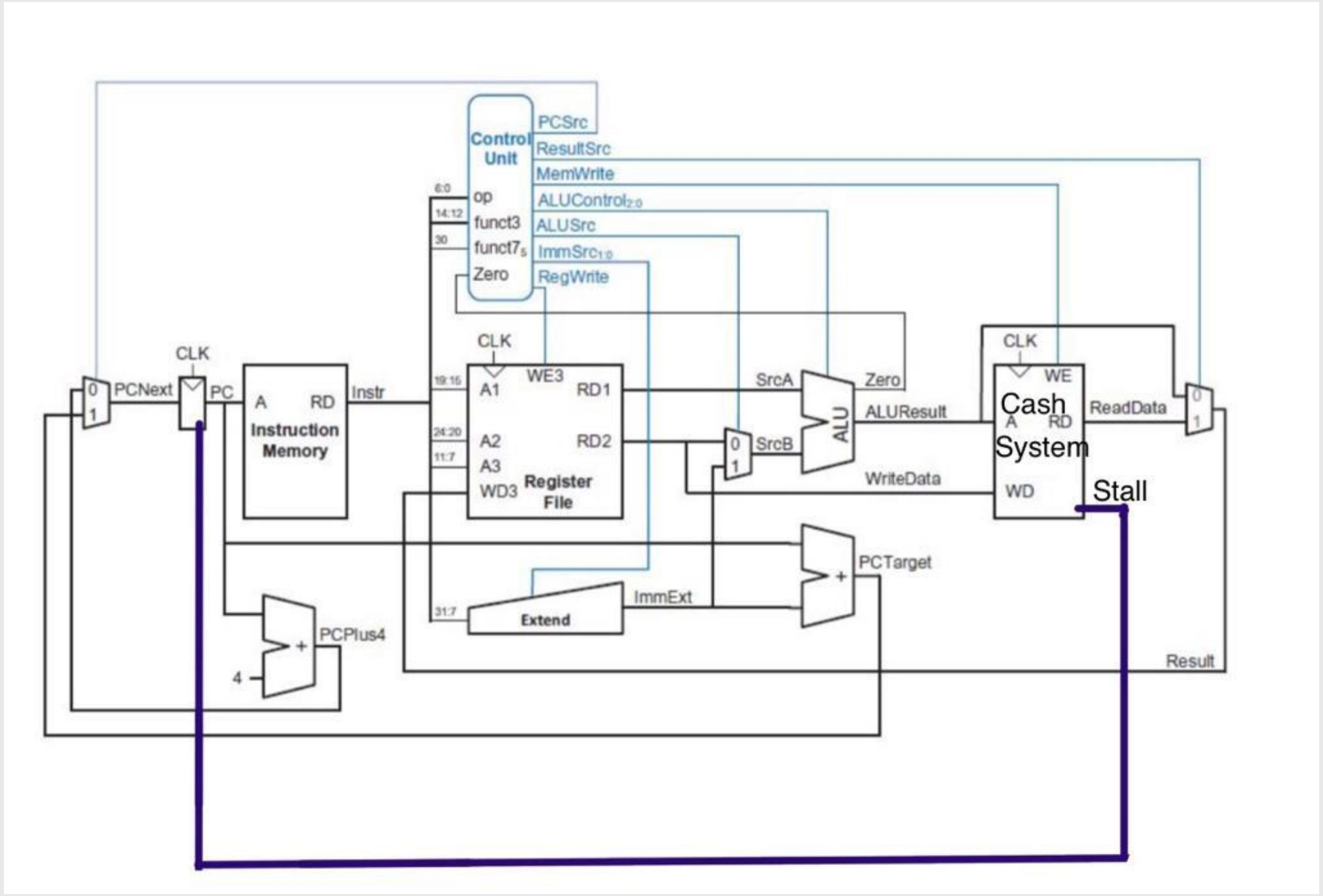
CASH CONTROLLER

Omnia Mohamed , Shohrt Helmy
Group 2

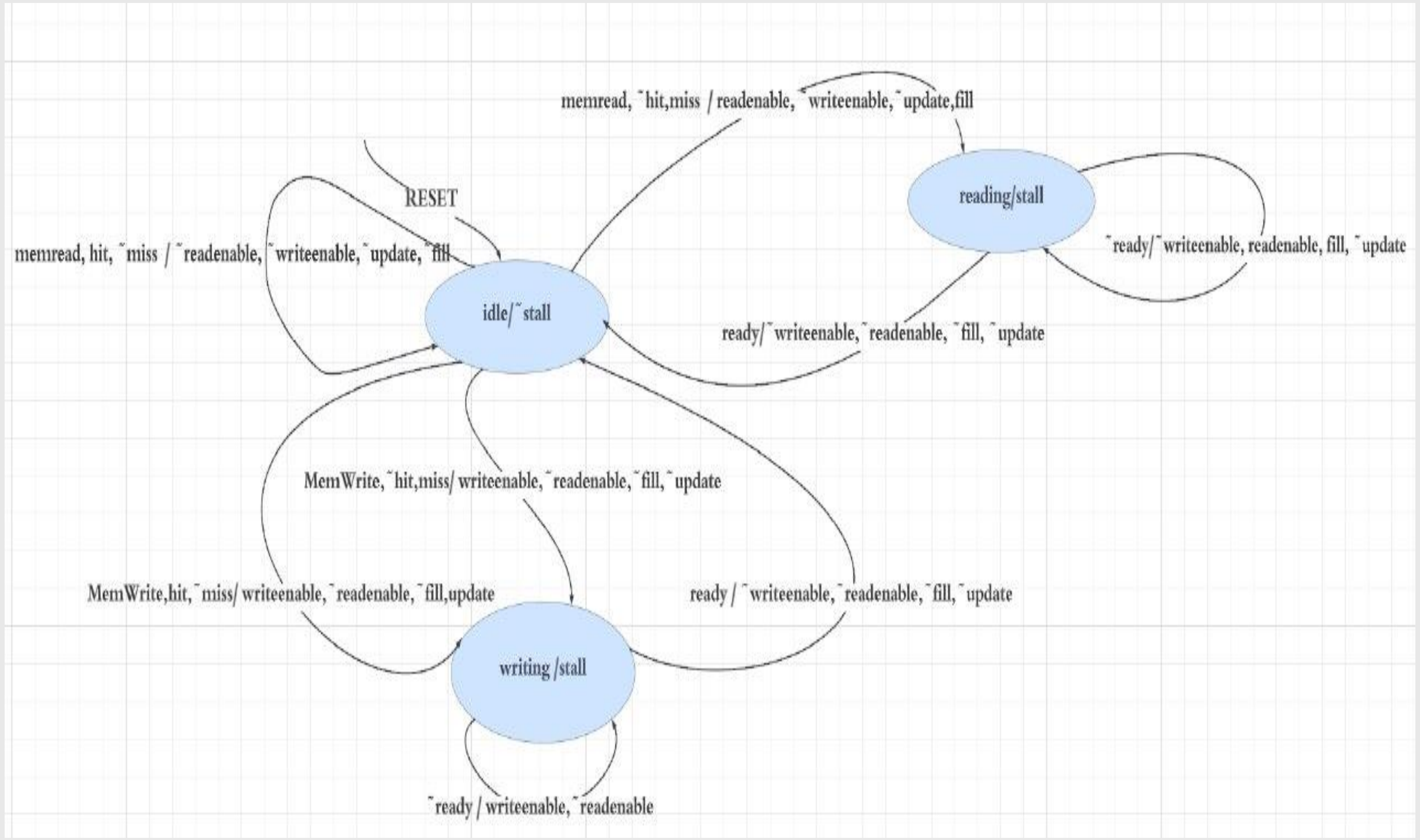
Top level of cash



Top RISC with cash



Controller FSM

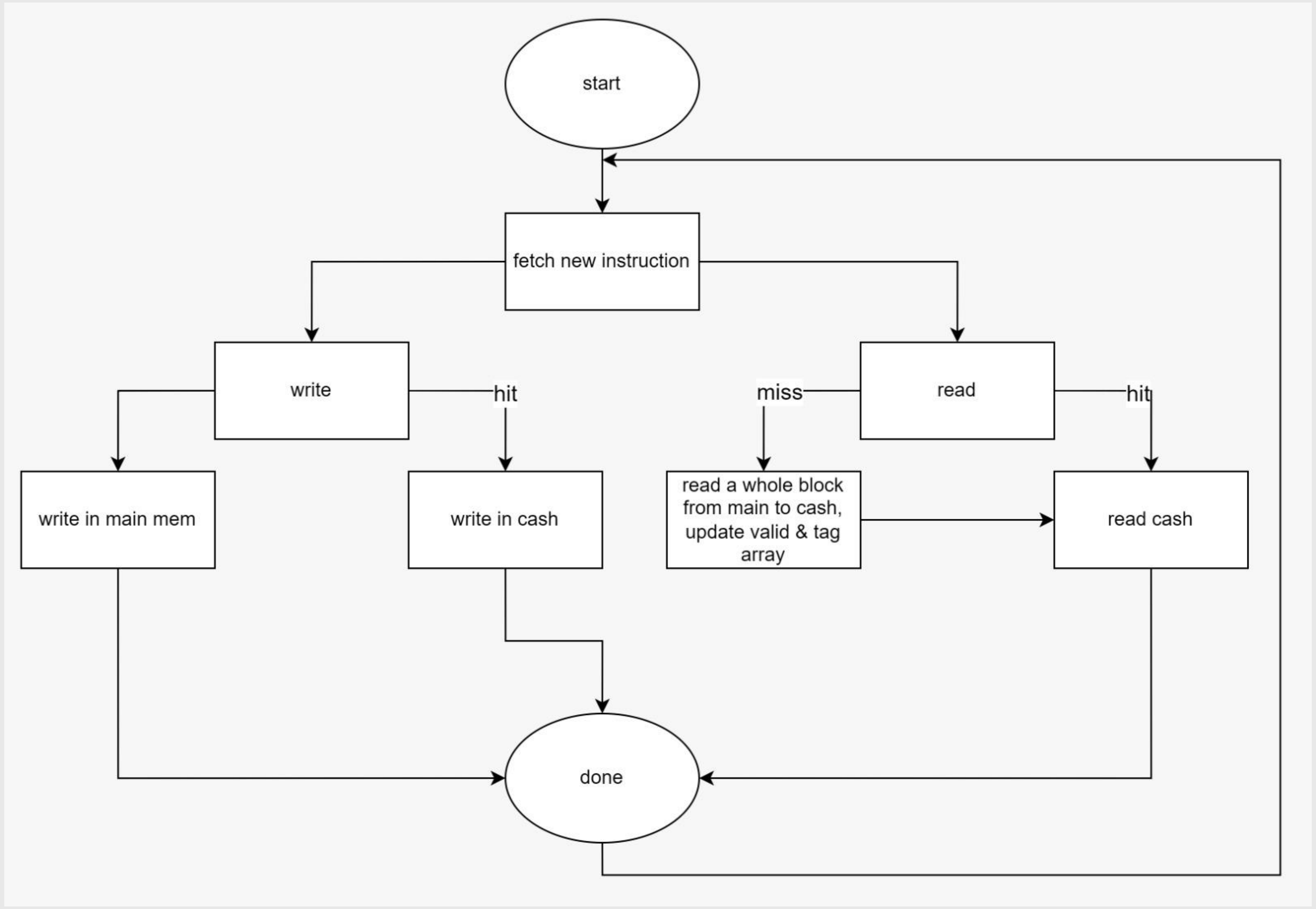

memread, ˜hit,miss / readenable, ˜writeenable,˜update,fill
RESET
reading/stall
memread, hit, ˜miss / ˜readenable, ˜writeenable,˜update, ˜fill
idle/˜stall
˜ready/˜writeenable, readenable, fill, ˜update
ready/˜writeenable,˜readenable, ˜fill, ˜update
MemWrite,˜hit,miss/ writeenable,˜readenable,˜fill,˜update
MemWrite,hit,˜miss/ writeenable,˜readenable,˜fill,update
ready / ˜writeenable,˜readenable,˜fill,˜update
writing /stall
˜ready / writeenable,˜readenable

Address format

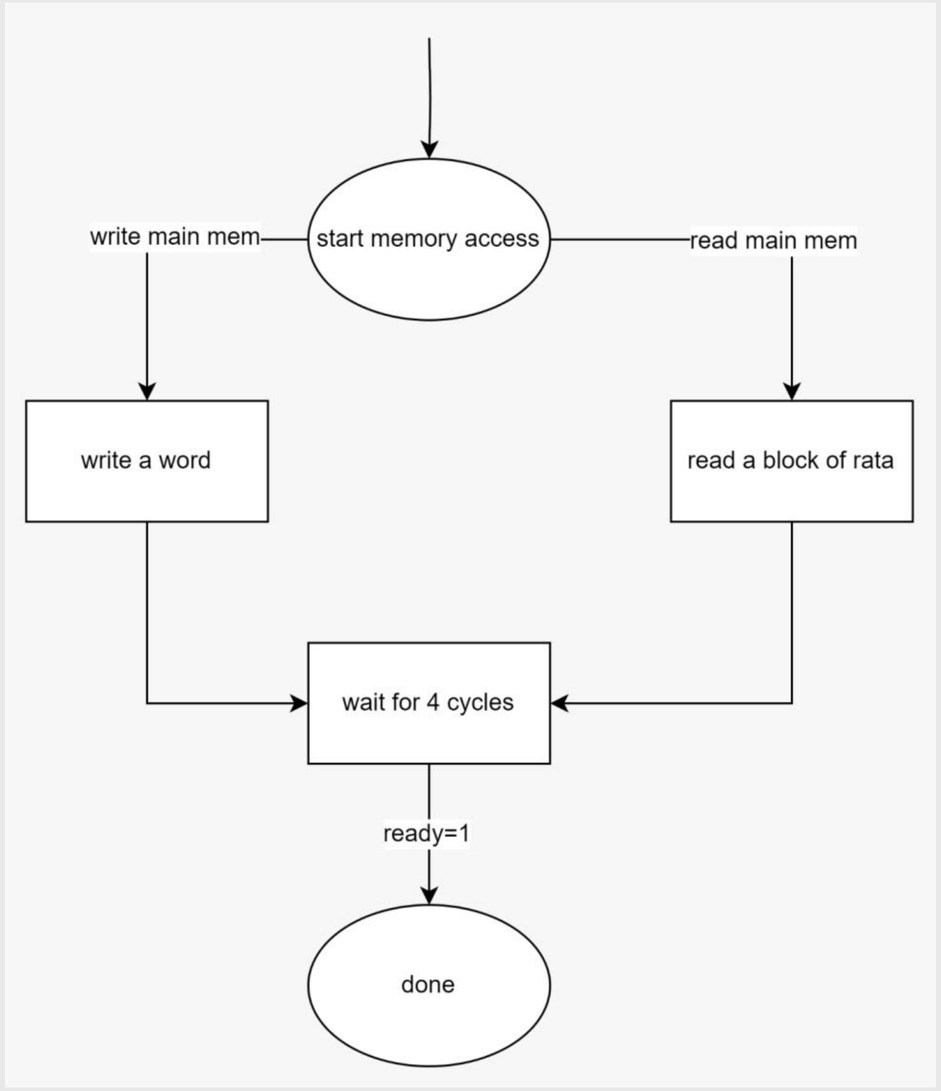
- The cash has 32 blocks → needs a 5 bit index to select a block.
- Each block has is 128 bit (4 words) → needs a 2 bit offset to select a certain word
- The main memory has 256 block each of 4 words, it has 8 groups each of 32 block → needs a 3 bit tag to determine the exact place of each block.

tag			index					ofset	
9	8	7	6	5	4	3	2	1	0

Flow chart of the system



Main memory flow chart



Main memory code

```
module main_memory (clk, read_main, write_main, address, wdata, ready, data_block);


input clk, read_main, write_main;
input [9:0] address;
input [31:0] wdata;
output reg ready;
output [127:0] data_block;


reg [2:0] count =0;
reg [31:0] main [0:2**10-1];
always @ (negedge clk) begin
            if (read_main || write_main)
                        if (count!=4) begin
                                    count<=count+1;
                        ready<=0;
```

Main memory code

```
end
                              else begin
                                             count<=0;
                                             ready<=1;
                              end
               end
always @ (negedge clk) begin
if (write_main) begin
               main[address] <= wdata;
end
end
assign data_block [31:0] = main [{address [ 9:2], 2'b00}];
assign data_block [63:32] = main [{address [ 9:2], 2'b01}];
assign data_block [95:64] = main [{address [ 9:2], 2'b10}];
assign data_block [127:96] = main [{address [ 9:2], 2'b11}];
endmodule
```

Cash array code

```
module cash_array (clk, reset, offset, index, tag, refill, update, w_data, r_data, main_data, valid, cash_tagged);

input clk, reset, refill, update;
input [1:0] offset;
input [4:0] index;
input [2:0] tag;
input [31:0] w_data;
input [127:0] main_data;
output valid;
output [2:0] cash_tagged;
output reg [31:0] r_data;

reg [131:0] cash [0:31];    //32 block each of 128 bit data + 3 bit tag + 1 valid bit
integer i;
```

Cash array code

```
always @ (posedge clk or negedge reset) begin
        if (reset==0) begin          //reset all valid bits and tag fields to zero
                for (i=0; i<32; i=i+1)
                cash[i] [132:128] <=0;
        end
        else if (refill) begin          //fill a whole block of cash with data from main (in case of read miss)
                cash [index] [127:0] = main_data;
                cash [index] [131]=1'b1;          //update valid to 1
                cash [index] [130:128]= tag;          //update tag
        end
else if (update) begin                    //write new data word in its place in cash (in case of write hit)
                case (offset)
                2'b00: cash [index] [31:0] = w_data;
                2'b01: cash [index] [63:32] = w_data;
                2'b10: cash [index] [95:64] = w_data;
                2'b11: cash [index] [127:96] = w_data;
        endcase
        end
end
```

Cash array code

```
//read data
always @(*) begin
        case (offset)
                2'b00: r_data = cash [index] [31:0];
                2'b01: r_data = cash [index] [63:32];
                2'b10: r_data = cash [index] [95:64];
                2'b11: r_data = cash [index] [127:96];
        endcase

end



assign valid = cash [index] [131] ;
assign cash_tagged = cash [index] [130:128];

endmodule
```

Cash controller code

```
module cash_controller (clk, rst_n, tag, valid, read, write, cash_tag, ready, stall, refill, cashe_update,
main_write, main_read);

input [2:0] tag, cash_tag;


input clk, rst_n, read, write, ready, valid;

output reg stall, refill, cashe_update, main_write, main_read;


wire hit;


parameter idle_state = 2'b00;

parameter read_state= 2'b01;

parameter write_state= 2'b10;


reg [1:0] current_state, next_state;
```

Cash controller code

```
always @(negedge clk or negedge rst_n) begin
        if(~rst_n) begin
                current_state<= idle_state;
        end
        else begin
                current_state <= next_state ;
        end
end

always @(*) begin
        case (current_state)
                idle_state: begin
if (read  && hit ) begin
                refill=0;
                cashe_update=0;
```

Cash controller code

```
        main_write=0;
        main_read=0;
        next_state=idle_state;
end

else if (read && ~hit) begin

        refill=1;
        cashe_update=0;
        main_write=0;
        main_read=1;
        next_state=read_state;
end
```

Cash controller code

```
else if (write && hit) begin

                refill=0;
                cashe_update=1;          //WRITE THROYGH
                main_write=1;
                main_read=0;
                next_state=write_state;
        end

        else if (write && ~hit) begin

                refill=0;
                cashe_update=0;
                main_write=1;                  //WRITE AROUND
                main_read=0;
                next_state=write_state;
        end

    end
```

Cash controller code

```
read_state:  if (ready) begin
                 refill=0;
                 cashe_update=0;
                 main_write=0;
                 main_read=0;
                 next_state=idle_state;
             end
             else begin
                 refill=1;
                 cashe_update=0;
                 main_write=0;
                 main_read=1;
                 next_state=read_state;
             end
```

Cash controller code

```
write_state: if (ready) begin

                                refill=0;
                                cashe_update=0;
                                main_write=0;
                                main_read=0;
                                next_state=idle_state;
        end
        else begin

                next_state= write_state;
        end
endcase // current_state
end
```

Cash controller code

```
//moor state output
always @(*)
case (current_state)
        2'b00: stall=0;
        2'b01:stall=1;
        2'b10:stall=1;
endcase
assign hit= ((tag==cash_tag)&&(valid==1));
endmodule
```

Top cash code

```
module top_cash (clk, reset_neg, address, write_data, write_en, read_en, stall, read_data);

input clk, reset_neg, write_en, read_en;

input [31:0] address, write_data;

output stall;

output [31:0] read_data;

wire valid, ready, cash_refill, cash_update, main_write, main_read;

wire [2:0] cash_tag;

wire [31:0] main_data;

wire [127:0] main_data_block;

//intantiate modules

cash_controller cc (clk, reset_neg, address[9:7], valid, read_en, write_en, cash_tag, ready, stall,
cash_refill, cash_update, main_write, main_read);

cash_array ca (clk, reset_neg, address[1:0], address[6:2], address[9:7], cash_refill, cash_update,
write_data, read_data, main_data_block, valid, cash_tag);

main_memory mm (clk, main_read, main_write, address[9:0], write_data, ready, main_data_block );

endmodule
```

Top RISC with cash module

```
module RISCV_cash (
        input clk, reset);


wire [31:0] PC, INSTR, IMEXT, PCnxt, Result, SrcA, SrcB, AluResult, ReadData, PCplus4, PCtarget,
WriteData;

wire Sign, ALUsrc, WE, PCSrc,  Zero, ResultSrc, RegWrite, stall ;

wire [2:0] AluControl;

wire [1:0] IMMSRC;


instruction_memory IM (PC, INSTR);

program_counter_cash pc (PCnxt, clk, reset, stall, PC);

extend ex (IMMSRC, INSTR[31:7], IMEXT);

register_file regfile (INSTR[19:15], INSTR[24:20], INSTR[11:7], Result, RegWrite, clk, reset, SrcA,
WriteData);

mux2 pcmux (PCplus4, PCtarget, PCSrc, PCnxt);
```

Top RISC with cash module

```
adder pc4 (PC, 32'd4, PCplus4);

adder pct (PC, IMEXT, PCtarget);

ALU alu (SrcA, SrcB, AluControl, Zero, Sign, AluResult);

mux2 alu_srcb (WriteData, IMEXT, ALUsrc, SrcB);



top_cash cash_system (clk, reset, AluResult, WriteData, WE, ~WE, stall, ReadData);


mux2 WD3 (AluResult, ReadData, ResultSrc, Result);


control_unit cu (INSTR[6:0], INSTR[14:12], INSTR[30], Zero, Sign, PCSrc, ResultSrc, WE, ALUsrc,
RegWrite, AluControl, IMMSRC );


endmodule
```

Test cases for testing cash top system

Another read hit

address=131.

Test write hit

write_en=1.

read_en=0.

address=130.

write_data= 15;

Test reading that last data:

write_en=1.

read_en=0.

Test cases for testing cash top system

Another read hit

address=131.

Test write hit

write_en=1.

read_en=0.

address=130.

write_data= 15;

Test reading that last data:

write_en=1.

read_en=0.

Test cases for testing cash top system

Reset = 0 to initialize state to idlec.

Check write miss:

Reset= 1.

write_en=1.

read_en=0.

address=128.

write_data= 1;

Continue writing the rest of the block:

address=129.

write_data= 2.

address=130.

write_data= 3.

address=131.

write_data= 4.

Check read miss Read the last location (128)

address=128.

write_en=0.

read_en=1.

(the whole bloch 128:131 is in the cash now)

Check read hit

address=129.

write_data= 2.

address=130.

write_data= 3.

Test cases for testing cash top system

Another read hit

address=131.

Test write hit

write_en=1.

read_en=0.

address=130.

write_data= 15;

Test reading that last data:

write_en=1.

read_en=0.

Cash top testbench

```
`timescale 1ns / 1ps
module cash_top_tb ();
reg clk, reset_neg, write_en, read_en;
reg [31:0] address, write_data;
wire stall;
wire [31:0] read_data;

top_cash uut (clk, reset_neg, address, write_data, write_en, read_en, stall, read_data);

always begin      //clk period 20 ns
        #10;
        clk=~clk;
end
```

Cash top testbench

```
initial begin
            //testing reset
            clk=0;
            reset_neg=0;
            address= 32'b0010000000;                  //address 128
            write_data= 1;
            write_en=1;
            read_en=0;
//wait 1 clk cycle and test write miss
#20;
reset_neg=1;
//wait 5 cycles: 1 for idle state 4 for memory access
#100;
//wait an extra cycle to memic the time the new unstructipn will come in
//check reading the same location --> read miss
//to get new ins
#10;
```

Cash top testbench

```
/write another location

address= 32'b0010000001;                    //address 129
write_data= 2;
//to change state
#10;
#100;
#10;     //to get a new instruction
// put new inst
address= 32'b0010000010;                        //address 130
write_data= 3;
#10;     //wait 10 to change state
#100;
#10;
```

Cash top testbench

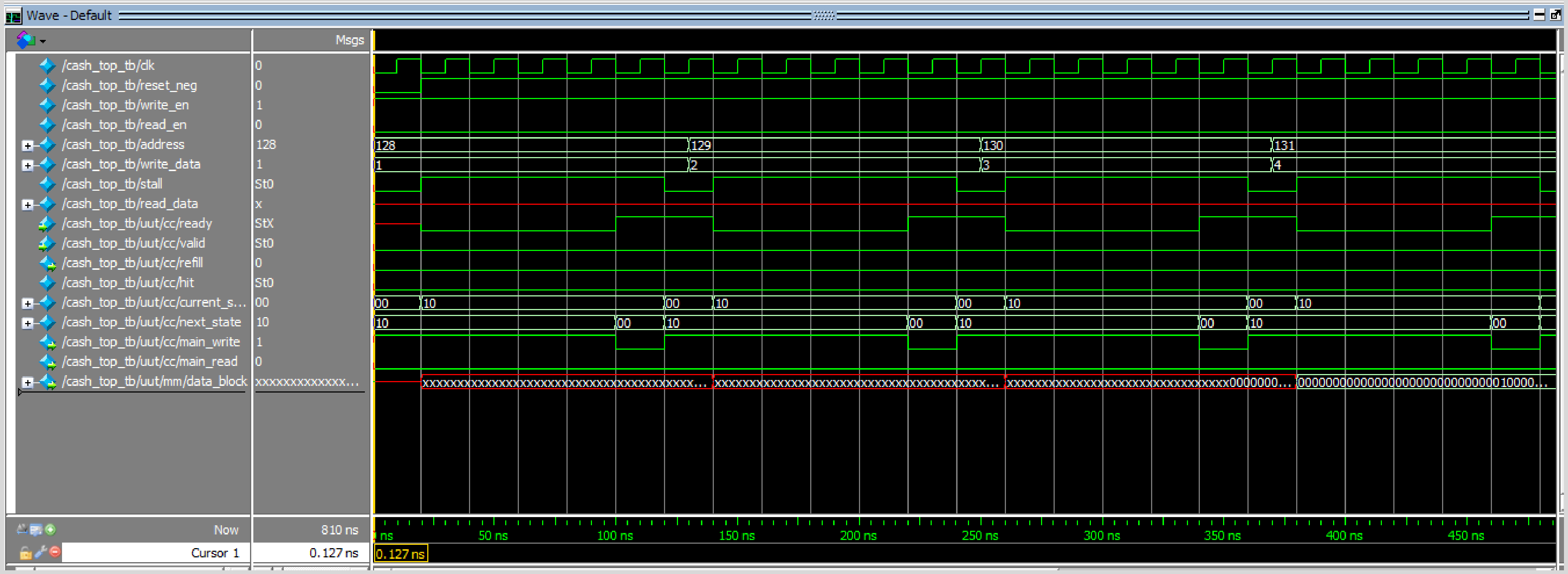
```
address= 32'b0010000011;                              //address 131
write_data= 4;
#10;
#100;
//wait 10 to fetch a new instruction
#10;
//testing read miss (nothing in cash yet)
address= 32'b0010000000;
write_en=0;
read_en=1;
//wait for neg edeg for controller to change to change state (go to read)
#10;
#100;            //for main mem access
#10; //testing read hit ------------> the whole block was transmitted
address= 32'b0010000001;
//another read hit
#20;
address= 32'b0010000010;
#10;
```

Cash top testbench

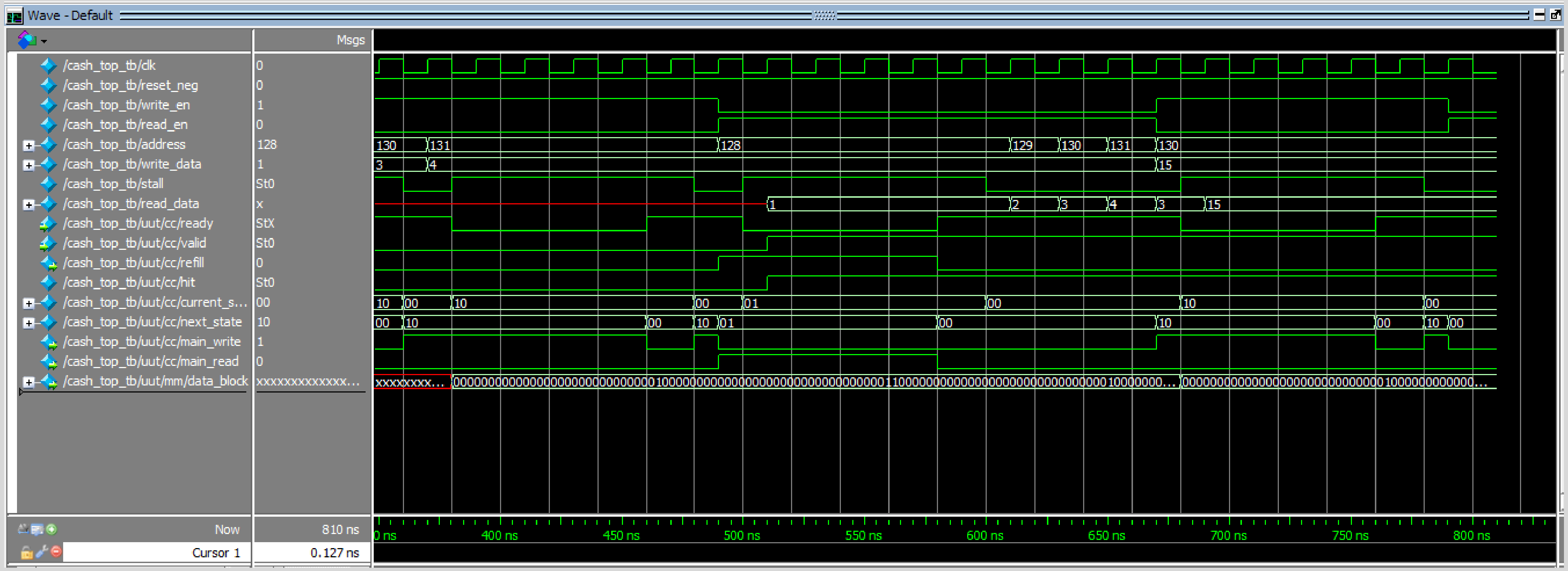
```
//testing read hit
#10;
address= 32'b0010000011;

#20; //to fetch a new inst
//test write hit in
address=32'b0010000010;
write_en=1;
read_en=0;
write_data=15;
#10;
#100;
```

Cash top testbench

```
//read that last location again --> read hit and make sure it's the updated data 15
#10;
//testing read miss (nothing in cash yet)
write_en=0;
read_en=1;
//wait for neg edeg for controller to change to change state (go to read)
#20;



$stop;
        end
        endmodule
```



```
/cash_top_tb/clk                 0
/cash_top_tb/reset_neg           0
/cash_top_tb/write_en            1
/cash_top_tb/read_en             0
/cash_top_tb/address             128
/cash_top_tb/write_data          1
/cash_top_tb/stall               St0
/cash_top_tb/read_data           x
/cash_top_tb/uut/cc/ready        StX
/cash_top_tb/uut/cc/valid        St0
/cash_top_tb/uut/cc/refill       0
/cash_top_tb/uut/cc/hit          St0
/cash_top_tb/uut/cc/current_s... 00
/cash_top_tb/uut/cc/next_state   10
/cash_top_tb/uut/cc/main_write   1
/cash_top_tb/uut/cc/main_read    0
/cash_top_tb/uut/mm/data_block   xxxxxxxxxxxxx...

Now                              810 ns
Cursor 1                         0.127 ns
```


Wave - Default

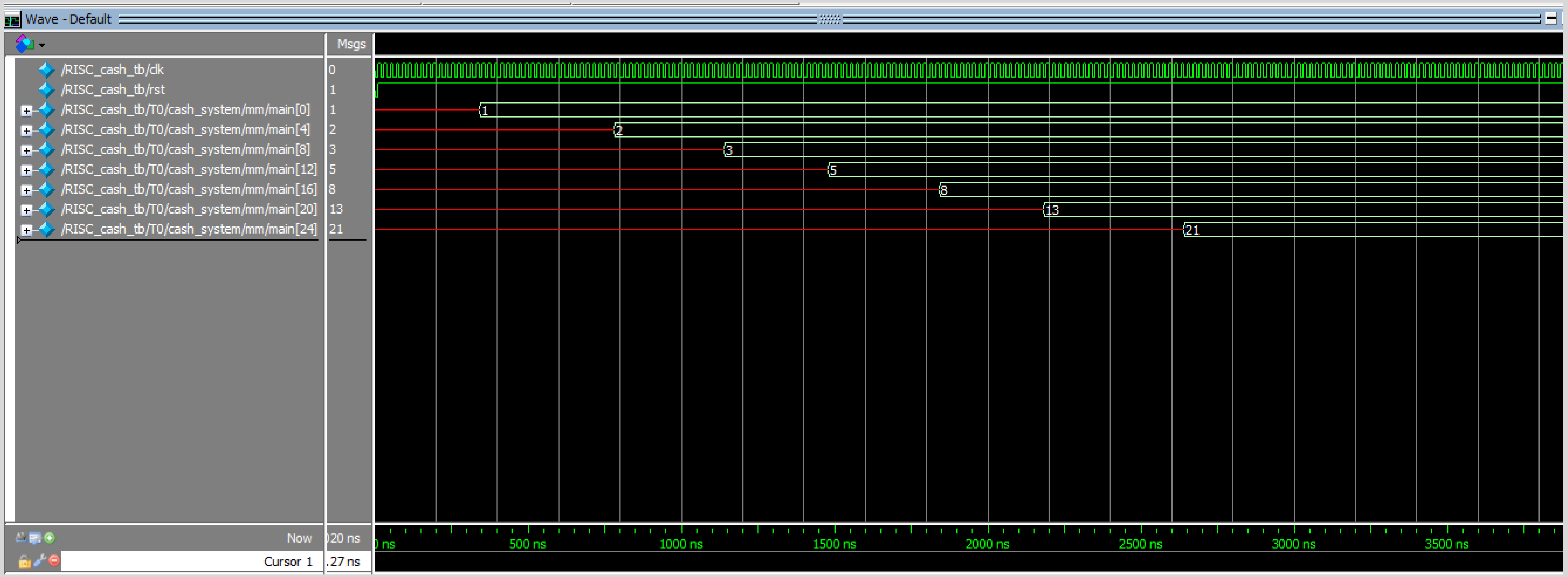
	Msgs
/cash_top_tb/clk	0
/cash_top_tb/reset_neg	0
/cash_top_tb/write_en	1
/cash_top_tb/read_en	0
/cash_top_tb/address	128
/cash_top_tb/write_data	1
/cash_top_tb/stall	St0
/cash_top_tb/read_data	x
/cash_top_tb/uut/cc/ready	StX
/cash_top_tb/uut/cc/valid	St0
/cash_top_tb/uut/cc/refill	0
/cash_top_tb/uut/cc/hit	St0
/cash_top_tb/uut/cc/current_s...	00
/cash_top_tb/uut/cc/next_state	10
/cash_top_tb/uut/cc/main_write	1
/cash_top_tb/uut/cc/main_read	0
/cash_top_tb/uut/mm/data_block	xxxxxxxxxxxxx...

Now	810 ns	
Cursor 1	0.127 ns	

Integrating with RISCV

- Used stall signal to disable PC from getting a new one.
- Used the whole cash system instead of the data memory

FIBONACCI series program result



Wave - Default

```
                                              Msgs
/RISC_cash_tb/clk                             0
/RISC_cash_tb/rst                             1
/RISC_cash_tb/T0/cash_system/mm/main[0]       1
/RISC_cash_tb/T0/cash_system/mm/main[4]       2
/RISC_cash_tb/T0/cash_system/mm/main[8]       3
/RISC_cash_tb/T0/cash_system/mm/main[12]      5
/RISC_cash_tb/T0/cash_system/mm/main[16]      8
/RISC_cash_tb/T0/cash_system/mm/main[20]      13
/RISC_cash_tb/T0/cash_system/mm/main[24]      21

Now        20 ns
Cursor 1   .27 ns
```