System Synthesis and Modeling (CSCE 3953)

Final Project (Step 4)

Zack Fravel

10/6/16

zpfravel@uark.edu

**Report**

Step four of the final project had us design a number of finite state machines that are used to generate the control signals along our datapath. With my designs, I broke up the functionality of the CPU into seven different FSMs. I have one FSM for the register-based operations, one for instruction fetch, and dedicated FSMs for Mov, Movi, Load, Store, and Immediate ALU operations.

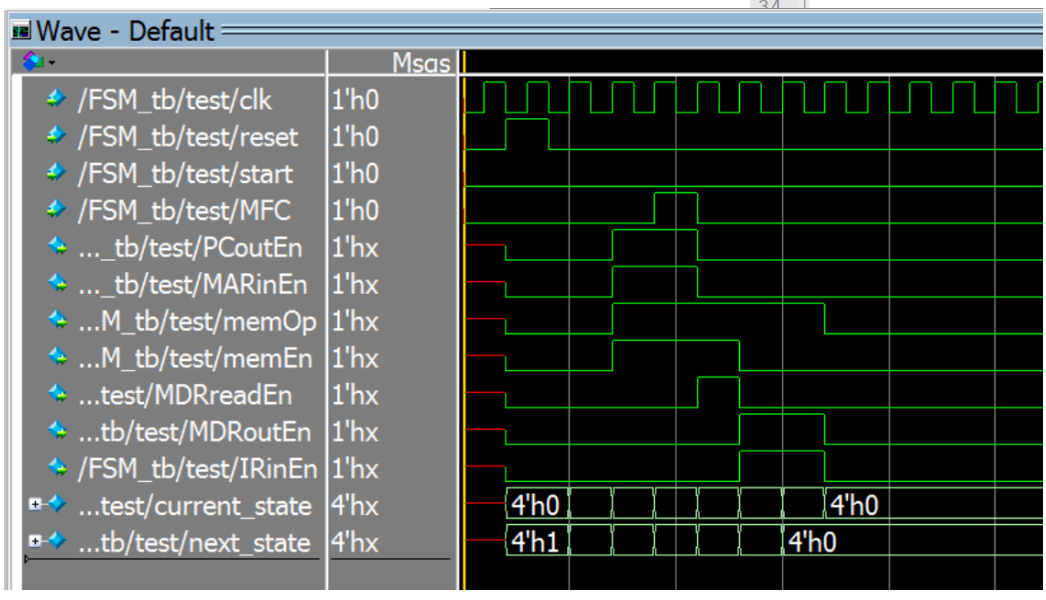
With the Immediate FSM's, I have the decoder pass in the value that needs to be send to the ALU and the FSM is responsible for driving the bus and sending these values. Therefore, I have a tri-state output on my Immediate FSM as well as the MovImm FSM. To simulate MFC, I just assumed MFC would be set to high so the whole simulation can be seen clearly at once. All of the FSM's send out a 'finish' signal that will be used to trigger the instruction fetch FSM at the end of each operation. The instruction fetch FSM is also triggered by reset. It would take a huge amount of pictures to include all of the code for these FSMs so I included the top module declarations with the ports and the output logic for each.

## Instruction Fetch FSM

```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 4)

module fetch_FSM(clk, reset, start, PCoutEn, MARinEn, memOp, memEn, MDRreadEn, MDRoutEn, IRinEn, MFC);

// Port Declartaion

input clk; input reset; input start; input MFC;

output reg PCoutEn; output reg MARinEn; output reg memOp; output reg memEn;
output reg MDRreadEn; output reg MDRoutEn; output reg IRinEn;

reg[3:0] current_state, next_state;
```

```
always @(current_state)                                    // Output Logic
begin

   case (current_state)

   init:  begin
                PCoutEn = 0; MARinEn = 0; memOp = 0; memEn = 0;
                MDRreadEn = 0; MDRoutEn = 0; IRinEn = 0;
          end
   one:   begin
                PCoutEn = 0; MARinEn = 0; memOp = 0; memEn = 0;
                MDRreadEn = 0; MDRoutEn = 0; IRinEn = 0;
          end
   two:   begin
                PCoutEn = 1; MARinEn = 1; memOp = 1; memEn = 1;
                MDRreadEn = 0; MDRoutEn = 0; IRinEn = 0;
          end
   three: begin
                PCoutEn = 1; MARinEn = 1; memOp = 1; memEn = 1;
                MDRreadEn = 0; MDRoutEn = 0; IRinEn = 0;
          end
   four:  begin
                PCoutEn = 0; MARinEn = 0; memOp = 1; memEn = 1;
                MDRreadEn = 1; MDRoutEn = 0; IRinEn = 0;
          end
   five:  begin
                PCoutEn = 0; MARinEn = 0; memOp = 1; memEn = 0;
                MDRreadEn = 0; MDRoutEn = 1; IRinEn = 1;
          end
   six:   begin
                PCoutEn = 0; MARinEn = 0; memOp = 1; memEn = 0;
                MDRreadEn = 0; MDRoutEn = 1; IRinEn = 1;
          end
```

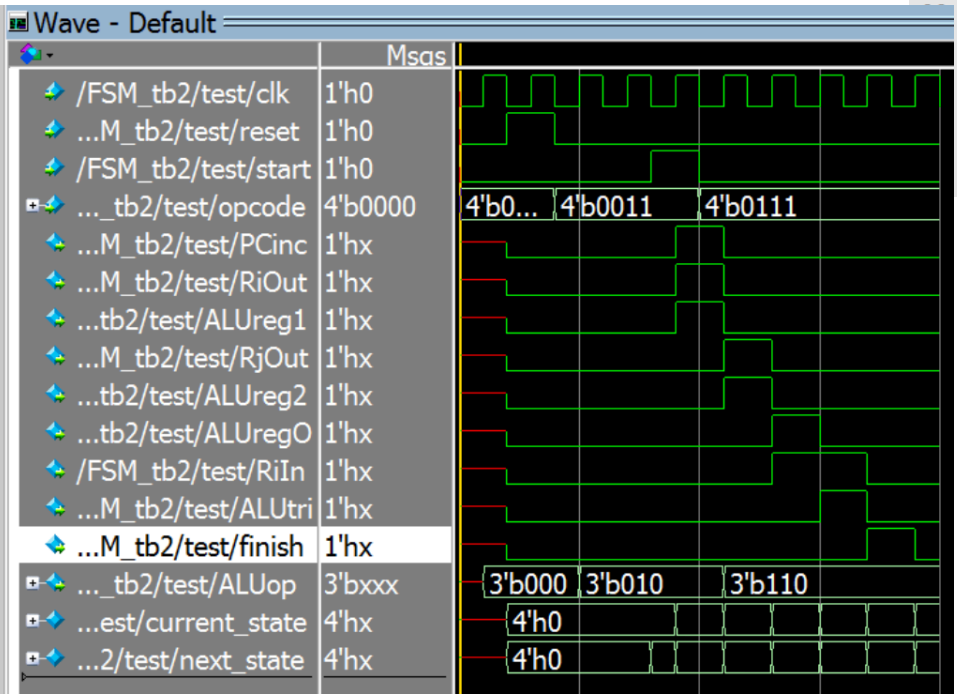
```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 4)

`timescale 1ns/1ns
module FSM_tb;

// Inputs
        reg CLK; reg RESET; reg START; reg MFC;
// Outputs
 wire PCoutEn;  wire MARinEn;  wire memOp;  wire
 wire MDRreadEn;  wire MDRoutEn;  wire IRinEn;

// Instantiate Module
        fetch_FSM test (CLK, RESET, START, PCout

initial
   begin
        CLK = 0;
        RESET = 0;
        START = 0;
        MFC = 0;
        #20;
        RESET = 1;
        #20;
        RESET = 0;
        #50;
        MFC = 1;
        #20;
        MFC = 0;
   end

always #10 CLK = ~CLK;
```



## Register FSM

```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 4)

module Register_FSM(clk, reset, start, PCinc, RiOut, ALUreg1, RjOut, ALUreg2, ALUregO, RiIn, ALUtri, finish, opcode, ALUop);

// Port Declartaion

input clk; input reset; input start; input[3:0] opcode;

output reg PCinc; output reg RiOut; output reg ALUreg1; output reg RjOut;
output reg ALUreg2; output reg ALUregO; output reg RiIn; output reg ALUtri; output reg finish;
output reg[2:0] ALUop;

reg[3:0] current_state, next_state;
```

```
always @(current_state)                                 // Output Logic
begin

   case (current_state)

   init:  begin
                PCinc = 0; RiOut = 0; RjOut = 0; RiIn = 0;
                ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0; ALUtri = 0; finish = 0;
          end
   one:   begin
                PCinc = 1; RiOut = 1; RjOut = 0; RiIn = 0;
                ALUreg1 = 1; ALUreg2 = 0; ALUregO = 0; ALUtri = 0; finish = 0;
          end
   two:   begin
                PCinc = 0; RiOut = 0; RjOut = 1; RiIn = 0;
                ALUreg1 = 0; ALUreg2 = 1; ALUregO = 0; ALUtri = 0; finish = 0;
          end
   three: begin
                PCinc = 0; RiOut = 0; RjOut = 0; RiIn = 1;
                ALUreg1 = 0; ALUreg2 = 0; ALUregO = 1; ALUtri = 0; finish = 0;
          end
   four:  begin
                PCinc = 0; RiOut = 0; RjOut = 0; RiIn = 1;
                ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0; ALUtri = 1; finish = 0;
          end
   five:  begin
                PCinc = 0; RiOut = 0; RjOut = 0; RiIn = 0;
                ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0; ALUtri = 0; finish = 1;
          end
   six:   begin
                PCinc = 0; RiOut = 0; RjOut = 0; RiIn = 0;
                ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0; ALUtri = 0; finish = 0;
          end
```



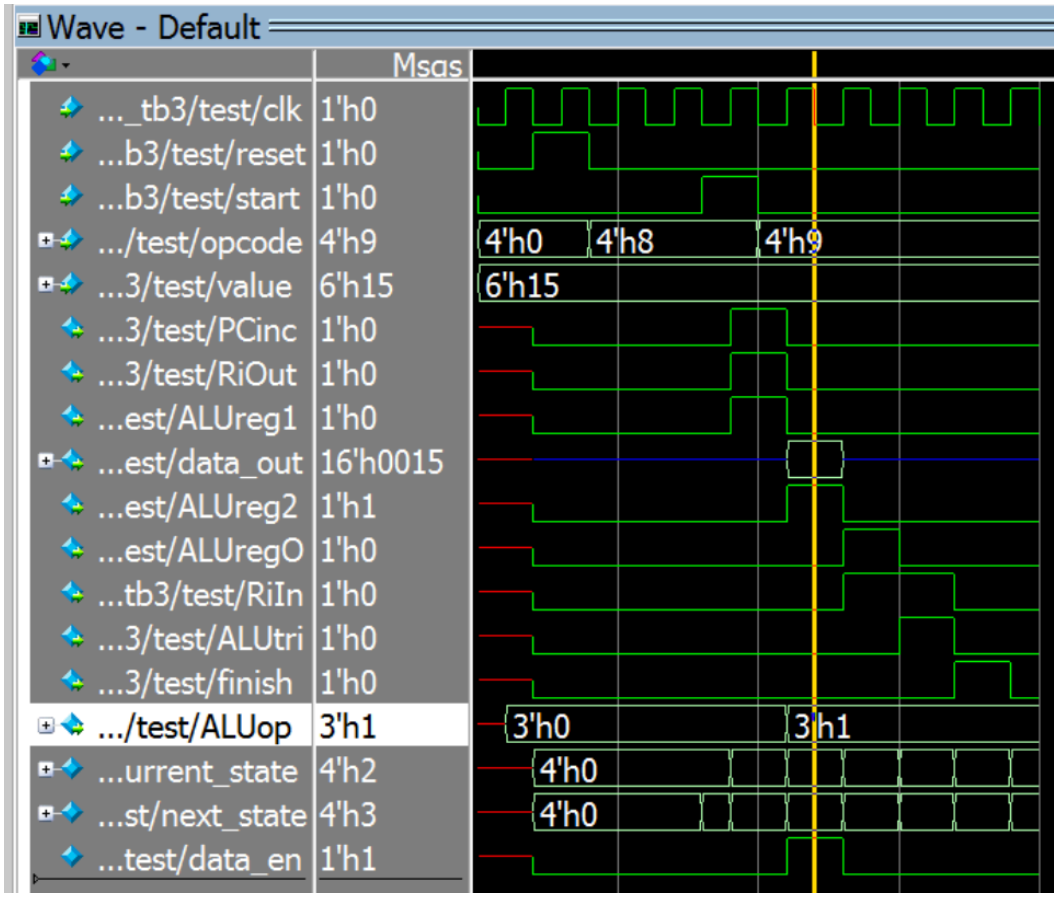

```
`timescale 1ns/1ns
module FSM_tb2;

// Inputs
        reg CLK; reg RESET; reg START; re
// Outputs
 wire PCinc;  wire ALUreg1;  wire ALUreg2
 wire RiIn;  wire RiOut;  wire RjOut; wi

// Instantiate Module
        Register_FSM test (CLK, RESET, ST

initial
   begin
        CLK = 0;
        RESET = 0;
        START = 0;
        OPCODE = 4'b0000;
        #20;
        RESET = 1;
        #20;
        RESET = 0;
        OPCODE = 4'b0011;
        #40;
        START = 1;
        #20;
        START = 0;
        OPCODE = 4'b0111;
   end

always #10 CLK = ~CLK;

endmodule
```

## ADD/SUB Imm FSM

```
// System Synthesis and Modeling
// Final Project (Step 4)

module Immediate_FSM(clk, reset, start, value, PCinc, RiOut, ALUreg1, data_out, ALUreg2, ALUregO, RiIn, ALUtri, finish, opcode, ALUop);

// Port Declartaion

input clk; input reset; input start; input[3:0] opcode; input[5:0] value;

output reg PCinc; output reg RiOut; output reg ALUreg1; output[15:0] data_out;
output reg ALUreg2; output reg ALUregO; output reg RiIn; output reg ALUtri; output reg finish;
output reg[2:0] ALUop;

always @(current_state)                                 // Output Logic
begin

   case (current_state)

   init:  begin
                PCinc = 0; RiOut = 0; ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0;
                RiIn = 0; ALUtri = 0; finish = 0; data_en = 0;
          end
   one:   begin
                PCinc = 1; RiOut = 1; ALUreg1 = 1; ALUreg2 = 0; ALUregO = 0;
                RiIn = 0; ALUtri = 0; finish = 0; data_en = 0;
          end
   two:   begin
                PCinc = 0; RiOut = 0; ALUreg1 = 0; ALUreg2 = 1; ALUregO = 0;
                RiIn = 0; ALUtri = 0; finish = 0; data_en = 1;
          end
   three: begin
                PCinc = 0; RiOut = 0; ALUreg1 = 0; ALUreg2 = 0; ALUregO = 1;
                RiIn = 1; ALUtri = 0; finish = 0; data_en = 0;
          end
   four:  begin
                PCinc = 0; RiOut = 0; ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0;
                RiIn = 1; ALUtri = 1; finish = 0; data_en = 0;
          end
   five:  begin
                PCinc = 0; RiOut = 0; ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0;
                RiIn = 0; ALUtri = 0; finish = 1; data_en = 0;
          end
   six:   begin
                PCinc = 0; RiOut = 0; ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0;
                RiIn = 0; ALUtri = 0; finish = 0; data_en = 0;
          end

   endcase

end
```

```
always@(posedge clk)                                    // Convert Opcode for ALU
begin

        case(opcode)
                4'b1000: begin ALUop = 3'b000; end      //Addi
                4'b1001: begin ALUop = 3'b001; end      //Subi
                default: begin ALUop = 3'b000; end
        endcase

end

        assign data_out = (data_en) ? value:16'hzzzz;


endmodule
```

```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 4)

`timescale 1ns/1ns
module FSM_tb3;

// Inputs
        reg CLK; reg RESET; reg START; reg[3:0] OPCODE; reg[5:0] value;
// Outputs
 wire PCinc;  wire ALUreg1;  wire ALUreg2;  wire ALUregO;
 wire RiIn;  wire RiOut;  wire RjOut; wire finish; wire ALUtri; wire[2:0] ALUop; wire[15:0] data_out;

// Instantiate Module
        Immediate_FSM test (CLK, RESET, START, value, PCinc, RiOut, ALUreg1, data_out, ALUreg2, ALUregO, RiIn, ALUtri, fin

initial
   begin
        CLK = 0;
        RESET = 0;
        START = 0;
        OPCODE = 4'b0000;
        value = 6'b010101;
        #20;
        RESET = 1;
        #20;
        RESET = 0;
        OPCODE = 4'b1000;
        #40;
        START = 1;
        #20;
        START = 0;
        OPCODE = 4'b1001;
   end

always #10 CLK = ~CLK;

endmodule
```

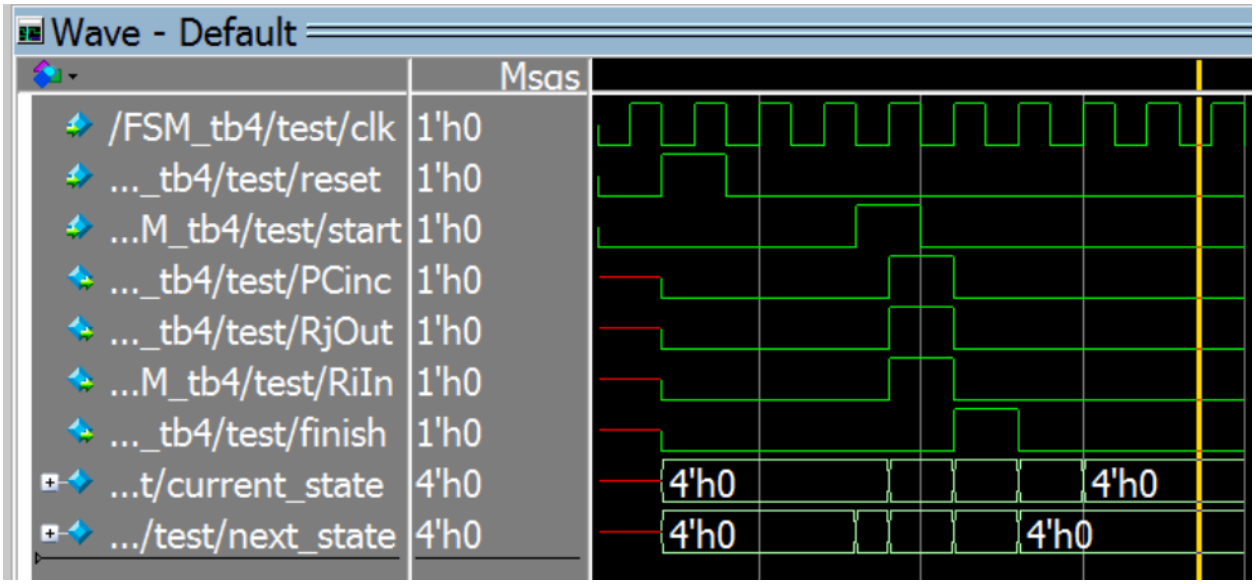


## Move FSM

```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 4)

module move_FSM(clk, reset, start, PCinc, RjOut, RiIn, finish);

// Port Declartaion

input clk; input reset; input start;

output reg PCinc; output reg RjOut; output reg RiIn; output reg finish;

reg[3:0] current_state, next_state;
```

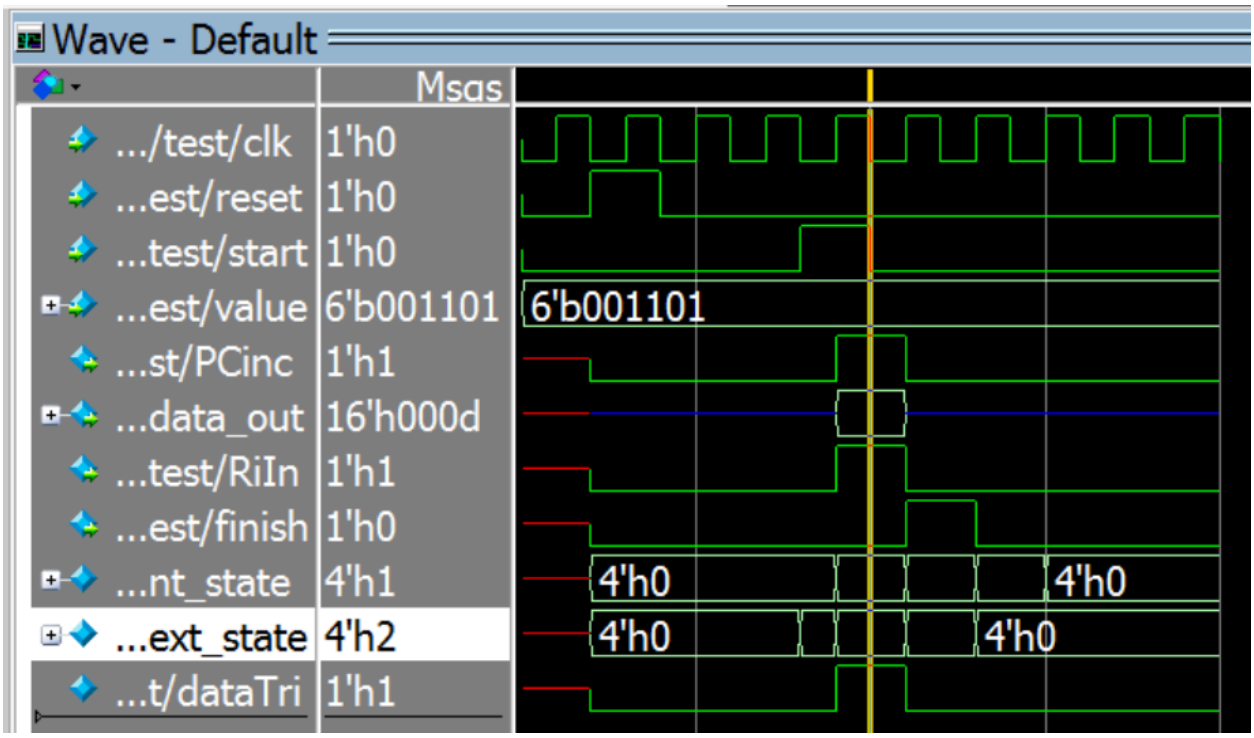
```
always @(current_state)                                    // Output Logic
begin

   case (current_state)

   init:  begin
                PCinc = 0; RjOut = 0; RiIn = 0; finish = 0;
          end
   one:   begin
                PCinc = 1; RjOut = 1; RiIn = 1; finish = 0;
          end
   two:   begin
                PCinc = 0; RjOut = 0; RiIn = 0; finish = 1;
          end
   three: begin
                PCinc = 0; RjOut = 0; RiIn = 0; finish = 0;
          end
   endcase
```

```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 4)

`timescale 1ns/1ns
module FSM_tb4;

// Inputs
       reg CLK; reg RESET; reg START; reg[5:0] value;
// Outputs
 wire PCinc;
 wire RiIn;  wire[15:0] data_out; wire finish;

// Instantiate Module
       moveImm_FSM test (CLK, RESET, START, value, PCinc, data_out, RiIn, finish);

initial
   begin
        CLK = 0;
        RESET = 0;
        START = 0;
        value = 6'b001101;
        #20;
        RESET = 1;
        #20;
        RESET = 0;
        #40;
        START = 1;
        #20;
        START = 0;
   end

always #10 CLK = ~CLK;

endmodule
```




**Move Imm FSM** (similar tb)

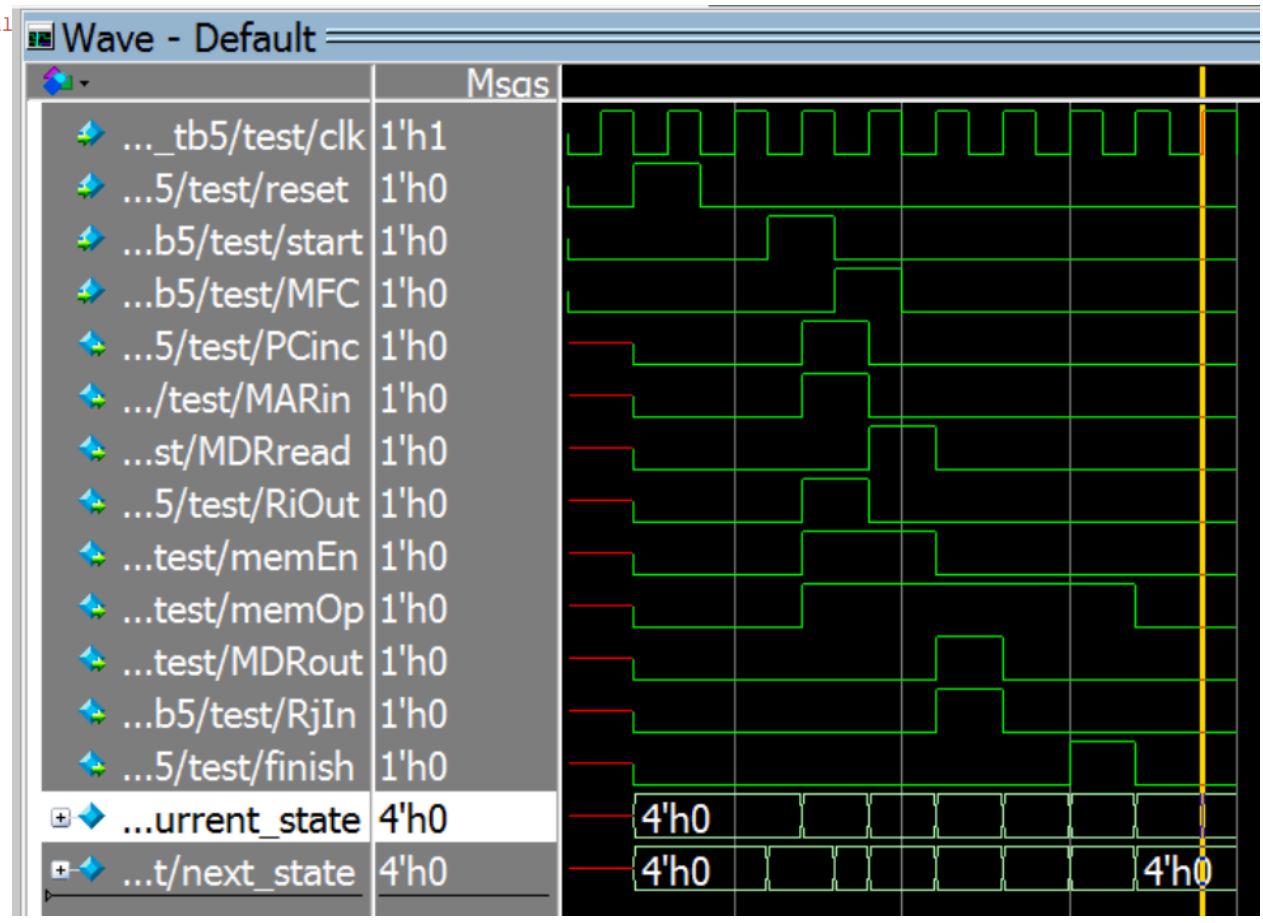
```
Ln#
  1 // Zack Fravel
  2 // System Synthesis and Modeling
  3 // Final Project (Step 4)
  4
  5 module moveImm_FSM(clk, reset, start, value, PCinc, data_out, RiIn, finish);
  6
  7 // Port Declartaion
  8
  9 input clk; input reset; input start; input[5:0] value;
 10
 11 output reg PCinc; output[15:0] data_out; output reg RiIn; output reg finish;
 12
 13 reg[3:0] current_state, next_state;
 14 reg dataTri;
 15

 53
 54 always @(current_state)                                    // Output Logic
 55 begin
 56
 57    case (current_state)
 58
 59    init:  begin
 60                 PCinc = 0; dataTri = 0; RiIn = 0; finish = 0;
 61           end
 62    one:   begin
 63                 PCinc = 1; dataTri = 1; RiIn = 1; finish = 0;
 64           end
 65    two:   begin
 66                 PCinc = 0; dataTri = 0; RiIn = 0; finish = 1;
 67           end
 68    three: begin
 69                 PCinc = 0; dataTri = 0; RiIn = 0; finish = 0;
 70           end
 71    endcase
 72
 73 end
 74
 75         assign data_out = (dataTri) ? value:16'hzzzz;
 76
 77
 78 endmodule
```




**Load FSM**

```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 4)

module Load_FSM(clk, reset, start, MFC, PCinc, RiOut, MARin, MDRread, memEn, memOp, MDRout, RjIn, finish);

// Port Declartaion

input clk; input reset; input start; input MFC;

output reg PCinc; output reg MARin; output reg MDRread; output reg RiOut;
output reg memEn; output reg memOp; output reg MDRout; output reg RjIn; output reg finish;

always @(current_state)                                 // Output Logic
begin

   case (current_state)

   init:  begin
                  PCinc = 0; MARin = 0; RiOut = 0; MDRread = 0;
                  memEn = 0; memOp = 0; MDRout = 0; RjIn = 0; finish = 0;
          end
   one:   begin
                  PCinc = 1; MARin = 1; RiOut = 1; MDRread = 0;
                  memEn = 1; memOp = 1; MDRout = 0; RjIn = 0; finish = 0;
          end
   two:   begin
                  PCinc = 0; MARin = 0; RiOut = 0; MDRread = 1;
                  memEn = 1; memOp = 1; MDRout = 0; RjIn = 0; finish = 0;
          end
   three: begin
                  PCinc = 0; MARin = 0; RiOut = 0; MDRread = 0;
                  memEn = 0; memOp = 1; MDRout = 1; RjIn = 1; finish = 0;
          end
   four:  begin
                  PCinc = 0; MARin = 0; RiOut = 0; MDRread = 0;
                  memEn = 0; memOp = 1; MDRout = 0; RjIn = 0; finish = 0;
          end
   five:  begin
                  PCinc = 0; MARin = 0; RiOut = 0; MDRread = 0;
                  memEn = 0; memOp = 1; MDRout = 0; RjIn = 0; finish = 1;
          end
   six:   begin
                  PCinc = 0; MARin = 0; RiOut = 0; MDRread = 0;
                  memEn = 0; memOp = 0; MDRout = 0; RjIn = 0; finish = 0;
          end

   endcase

end
```

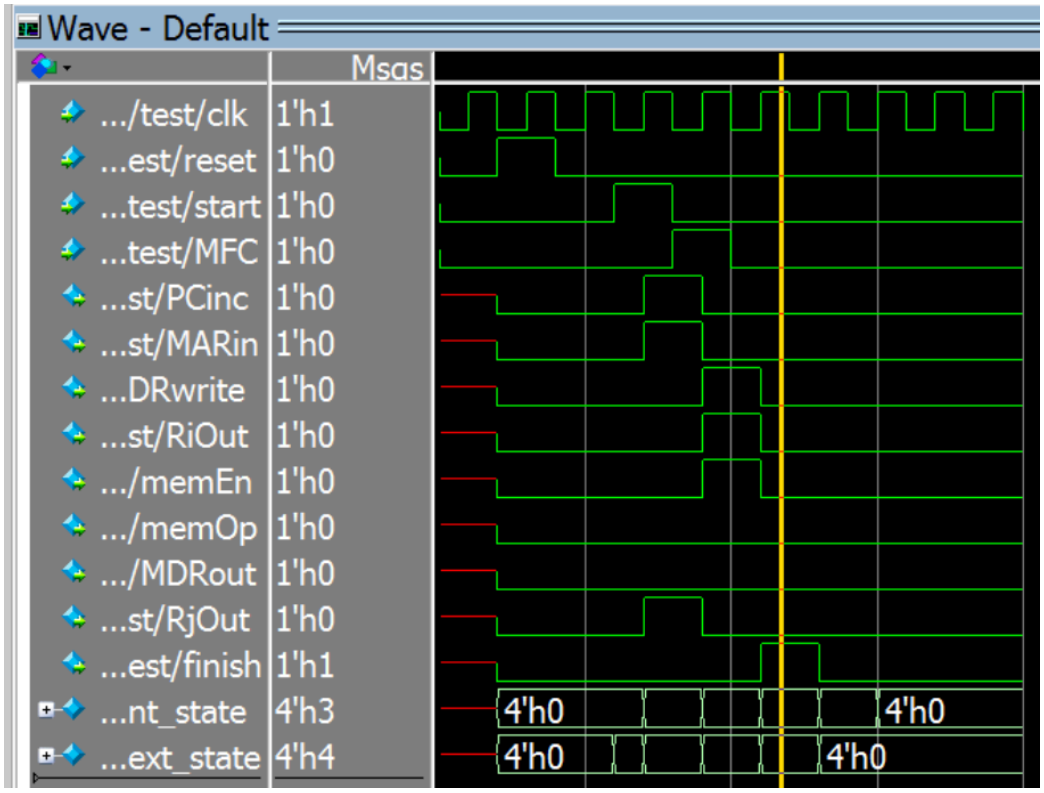
```
1  // Zack Fravel
2  // System Synthesis and Modeling
3  // Final Project (Step 4)
4
5  `timescale 1ns/1ns
6  module FSM_tb5;
7
8  // Inputs
9          reg CLK; reg RESET; reg START; reg MFC;
10 // Outputs
11 wire PCinc; wire MARin; wire MDRwrite; wire RiOut;
12 wire memEn; wire memOp; wire MDRout; wire RjOut; wire finish;
13
14 // Instantiate Module
15         Store_FSM test (CLK, RESET, START, MFC, PCinc, RiOut, MARin, MDRwrite, memEn, memOp, MDRout, RjOut, finish);
16
17 initial
18    begin
19         CLK = 0;
20         RESET = 0;
21         START = 0;
22         MFC = 0;
23         #20;
24         RESET = 1;
25         #20;
26         RESET = 0;
27         #20;
28         START = 1;
29         #20;
30         START = 0;
31         MFC = 1;
32         #20;
33         MFC = 0;
34    end
35
36 always #10 CLK = ~CLK;
37
38 endmodul
39
```



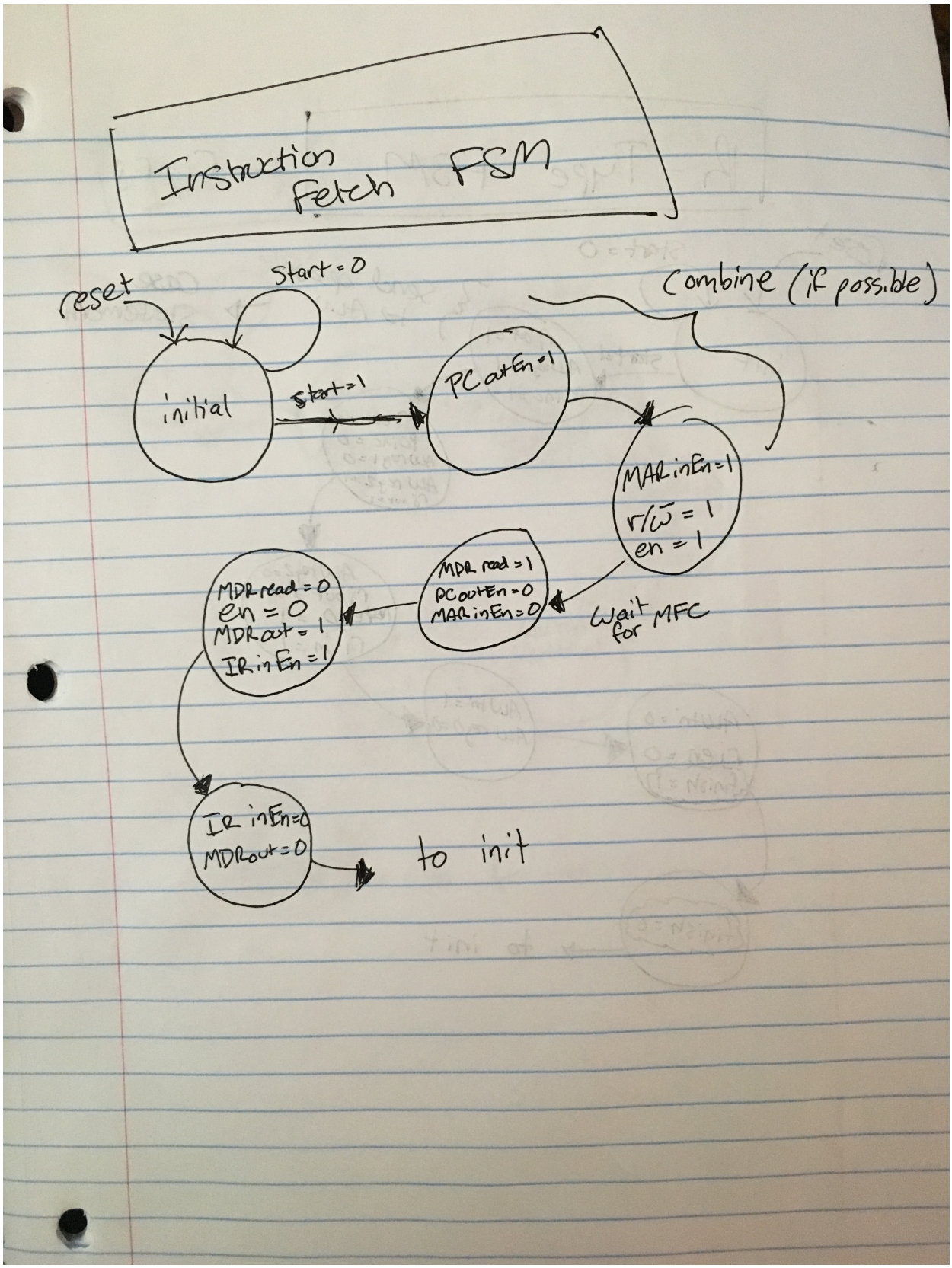
**Store FSM** (similar tb)

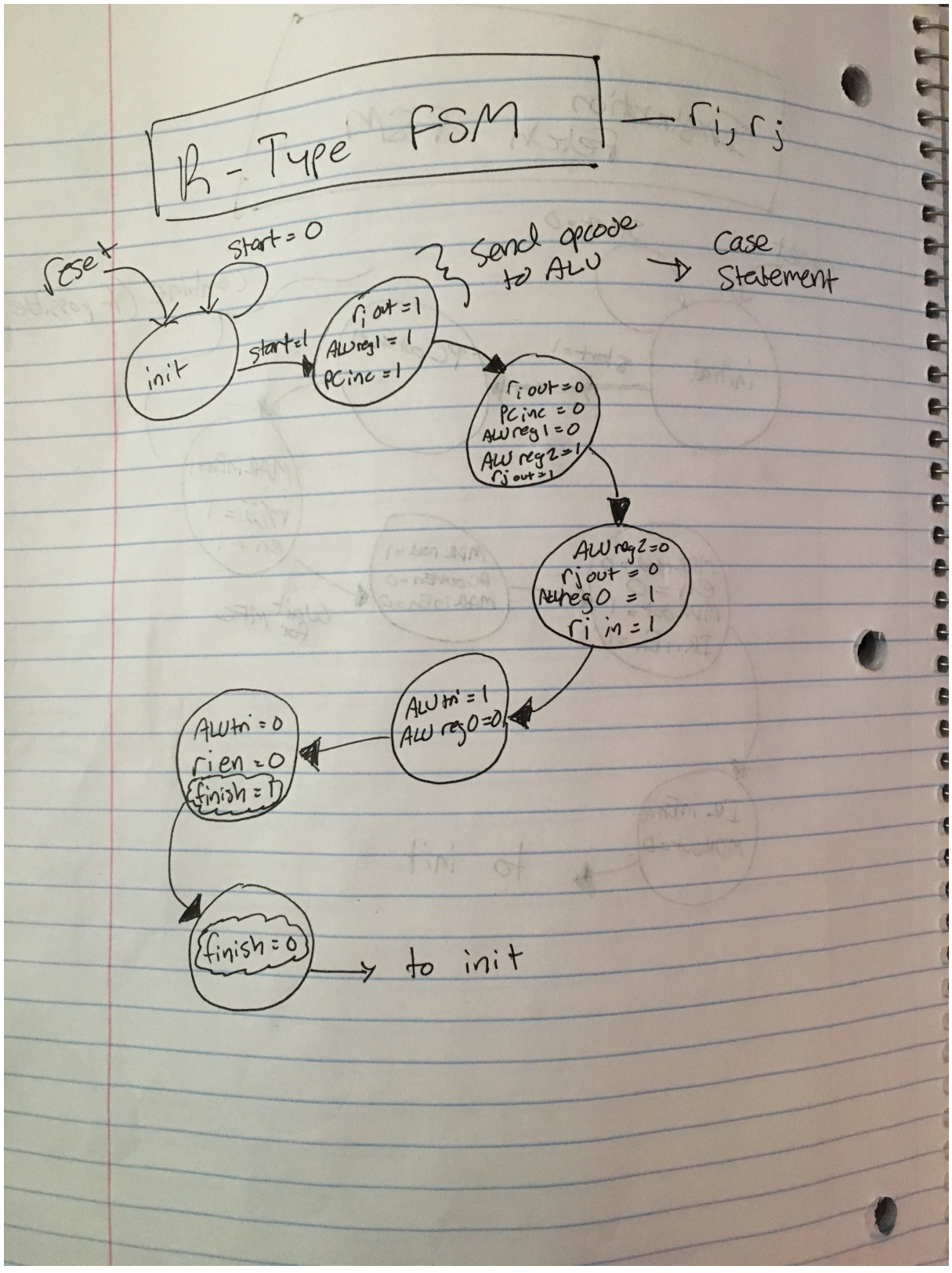
```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 4)

module Store_FSM(clk, reset, start, MFC, PCinc, RiOut, MARin, MDRwrite, memEn, memOp, MDRout, RjOut, finish);

// Port Declartaion

input clk; input reset; input start; input MFC;

output reg PCinc; output reg MARin; output reg MDRwrite; output reg RiOut;
output reg memEn; output reg memOp; output reg MDRout; output reg RjOut; output reg finish;

reg[3:0] current_state, next_state;
```

```
always @(current_state)                                 // Output Logic
begin

    case (current_state)

    init:  begin
                PCinc = 0; MARin = 0; RiOut = 0; MDRwrite = 0;
                memEn = 0; memOp = 0; MDRout = 0; RjOut = 0; finish = 0;
           end
    one:   begin
                PCinc = 1; MARin = 1; RiOut = 0; MDRwrite = 0;
                memEn = 0; memOp = 0; MDRout = 0; RjOut = 1; finish = 0;
           end
    two:   begin
                PCinc = 0; MARin = 0; RiOut = 1; MDRwrite = 1;
                memEn = 1; memOp = 0; MDRout = 0; RjOut = 0; finish = 0;
           end
    three: begin
                PCinc = 0; MARin = 0; RiOut = 0; MDRwrite = 0;
                memEn = 0; memOp = 0; MDRout = 0; RjOut = 0; finish = 1;
           end
    four:  begin
                PCinc = 0; MARin = 0; RiOut = 0; MDRwrite = 0;
                memEn = 0; memOp = 0; MDRout = 0; RjOut = 0; finish = 0;
           end

    endcase

end

endmodule
```



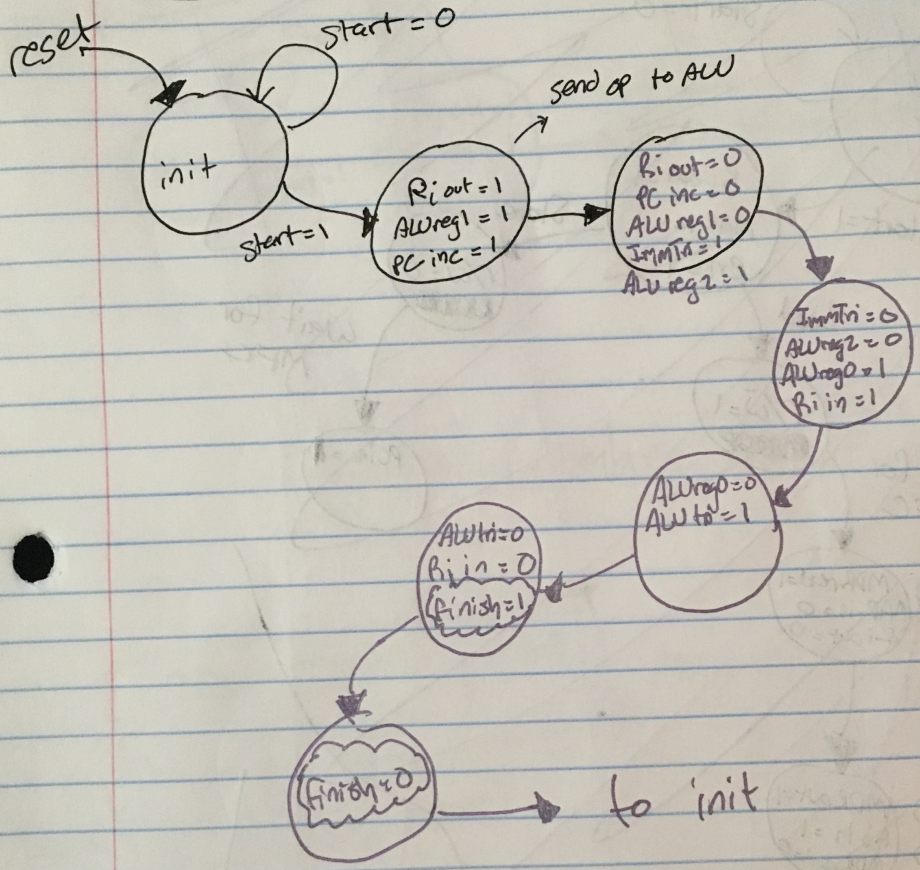
**STG's**


Instruction Fetch FSM
reset
start = 0
initial
start = 1
PCoutEn = 1
Combine (if possible)
MARinEn = 1
r/w = 1
en = 1
MDR read = 1
PCoutEn = 0
MARinEn = 0
Wait for MFC
MDR read = 0
en = 0
MDRout = 1
IRinEn = 1
IR inEn = 0
MDRout = 0
to init


R-Type FSM
ri, rj
reset
start = 0
start=1
init
ri out = 1
ALU reg1 = 1
PC inc = 1
Send opcode to ALU
Case Statement
ri out = 0
PC inc = 0
ALU reg1 = 0
ALU reg2 = 1
rj out = 1
ALU reg2 = 0
rj out = 0
ALU reg0 = 1
ri in = 1
ALU ri = 1
ALU reg0 = 0
ALU ri = 0
ri en = 0
finish = 1
finish = 0
to init

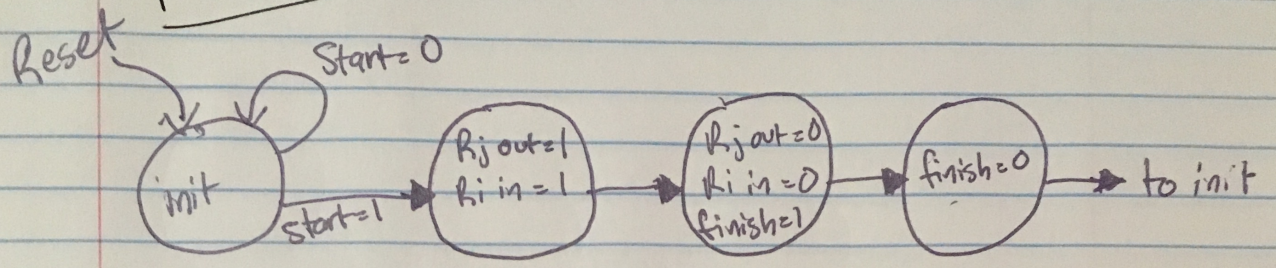
# Imm - ALU op FSM

add/sub Ri, #



MOV FSM

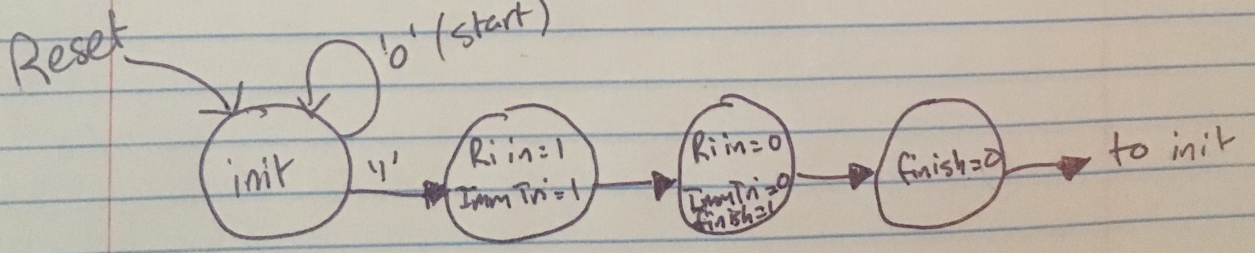
MOV Ri, Rj




(potentially combine these?)

MOV-IMM FSM

MOVi Ri, #



(ended up splitting these up)