Winter 2024
ECEN 122L Lab 3 Report
Ayden Dauenhauer, Cassie Hashemi
Feb. 5, 2024

For your lab report, include the source code for your working state machine. In addition, include answers to the following questions.

```
module l3_SM(input clk,
                    input execute,
                    input [2:0] operation, // opcode (part of instruction)
                    output reg _Extern,
                    output reg Gout,
                    output reg Ain,
                    output reg Gin,
                    output reg RdX,
                    output reg RdY,
                    output reg WrX,
                    output reg add_sub,
                    output reg DPin,
                    output reg Iout,
                    output [3:0] cur_state);

        //defining all my states - 8 total
        parameter IDLE          =  4'b0000;
        parameter LOAD          =  4'b0001;
        parameter READ_Y  =  4'b0010;
        parameter READ_X  =  4'b0011;
        parameter ADD           =  4'b0100;
        parameter SUB           =  4'b0101;
        parameter MV            =  4'b0110;
        parameter WRITE_X       =  4'b0111;
        parameter DONE          =  4'b1000;
        parameter ADDI          =  4'b1001;
        parameter SUBI          =  4'b1010;
        parameter DISPLAY       =  4'b1011;

        reg [3:0] state = IDLE; // initial state being IDLE

        assign cur_state = state;

        initial begin //instead of reset
        state <= IDLE;
        end

        always@(*)
        begin
           case(state)
               IDLE:
                   begin
                       _Extern = 1'b0;
                       Gout = 1'b0;
                       Ain = 1'b0;
                       Gin = 1'b0;
```

```
            RdX = 1'b0;
            RdY = 1'b0;
            WrX = 1'b0;
            add_sub = 1'b0;
            Iout = 1'b0;
            DPin = 1'b0;
        end
    LOAD:
        begin
            _Extern = 1'b1;
            Gout = 1'b0;
            Ain = 1'b0;
            Gin = 1'b0;
            RdX = 1'b0;
            RdY = 1'b0;
            WrX = 1'b1;
            add_sub = 1'b0;
            Iout = 1'b0;
            DPin = 1'b0;
        end
    READ_Y:
        begin
            _Extern = 1'b0;
            Gout = 1'b0;
            Ain = 1'b1;
            Gin = 1'b0;
            RdX = 1'b0;
            RdY = 1'b1;
            WrX = 1'b0;
            add_sub = 1'b0;
            Iout = 1'b0;
            DPin = 1'b0;
        end
    READ_X:
        begin
            _Extern = 1'b0;
            Gout = 1'b0;
            Ain = 1'b1;
            Gin = 1'b0;
            RdX = 1'b1;
            RdY = 1'b0;
            WrX = 1'b0;
            add_sub = 1'b0;
            Iout = 1'b0;
            DPin = 1'b0;
        end
    ADD:
        begin
            _Extern = 1'b0;
```

```
            Gout = 1'b0;
            Ain = 1'b0;
            Gin = 1'b1;
            RdX = 1'b1;
            RdY = 1'b0;
            WrX = 1'b0;
            add_sub = 1'b0;
            Iout = 1'b0;
            DPin = 1'b0;
        end
    SUB:
        begin
            _Extern = 1'b0;
            Gout = 1'b0;
            Ain = 1'b0;
            Gin = 1'b1;
            RdX = 1'b0;
            RdY = 1'b1;
            WrX = 1'b0;
            add_sub = 1'b1;
            Iout = 1'b0;
            DPin = 1'b0;
        end
    MV:
        begin
            _Extern = 1'b0;
            Gout = 1'b0;
            Ain = 1'b0;
            Gin = 1'b1;
            RdX = 1'b0;
            RdY = 1'b0;
            WrX = 1'b0;
            add_sub = 1'b0;
            Iout = 1'b0;
            DPin = 1'b0;
        end
    WRITE_X:
        begin
            _Extern = 1'b0;
            Gout = 1'b1;
            Ain = 1'b0;
            Gin = 1'b0;
            RdX = 1'b0;
            RdY = 1'b0;
            WrX = 1'b1;
            add_sub = 1'b0;
            Iout = 1'b0;
            DPin = 1'b0;
        end
```

```
DONE:
    begin
        _Extern = 1'b0;
        Gout = 1'b0;
        Ain = 1'b0;
        Gin = 1'b0;
        RdX = 1'b0;
        RdY = 1'b0;
        WrX = 1'b0;
        add_sub = 1'b0;
        Iout = 1'b0;
        DPin = 1'b0;
    end
ADDI:
    begin
        _Extern = 1'b0;
        Gout = 1'b0;
        Ain = 1'b0;
        Gin = 1'b1;
        RdX = 1'b0;
        RdY = 1'b0;
        WrX = 1'b0;
        add_sub = 1'b0;
        Iout = 1'b1;
        DPin = 1'b0;
    end
SUBI:
    begin
        _Extern = 1'b0;
        Gout = 1'b0;
        Ain = 1'b0;
        Gin = 1'b1;
        RdX = 1'b0;
        RdY = 1'b0;
        WrX = 1'b0;
        add_sub = 1'b1;
        Iout = 1'b1;
        DPin = 1'b0;
    end
DISPLAY:
    begin
        _Extern = 1'b0;
        Gout = 1'b0;
        Ain = 1'b0;
        Gin = 1'b0;
        RdX = 1'b1;
        RdY = 1'b0;
        WrX = 1'b0;
        add_sub = 1'b0;
```

```
                    Iout = 1'b0;
                    DPin = 1'b1;
                end
        endcase
    end

/*
opcode encodings
000 - load
001 - move
010 - subtract
011 - add
100 - disp
101 - reserved
110 - subi
111 - addi
*/

    always@(posedge clk)
    begin

        case(state)
            IDLE: begin
                    if(execute == 1'b1 && operation == 3'b000)
state <= LOAD;
                    if(execute == 1'b1 && operation == 3'b001)
state <= READ_Y;
                    if(execute == 1'b1 && operation == 3'b011)
state <= READ_Y;
                    if(execute == 1'b1 && operation == 3'b010)
state <= READ_X;
                    if(execute == 1'b1 && operation == 3'b110)
state <= READ_X;
                    if(execute == 1'b1 && operation == 3'b111)
state <= READ_X;
                    if(execute == 1'b1 && operation == 3'b100)
state <= DISPLAY;
                end

            LOAD: begin
                    state <= DONE; // always go to the DONE state
at the next clock tick
                end

            READ_Y: begin
                    if (operation == 3'b001) state <= MV;
                    if (operation == 3'b011) state <= ADD;
                    else state <= READ_Y;
                end
```

```
                READ_X: begin
                            if (operation == 3'b010) state <= SUB;
                            if (operation == 3'b110) state <= SUBI;
                            if (operation == 3'b111) state <= ADDI;
                            else state <= READ_X;
                        end

                ADD: begin
                            state <= WRITE_X;
                 end

                SUB: begin
                            state <= WRITE_X;
                       end

                MV: begin
                            state <= WRITE_X;
                      end

                WRITE_X: begin
                            state <= DONE;
                       end

                SUBI: begin
                            state <= WRITE_X;
                       end

              ADDI: begin
                            state <= WRITE_X;
                       end

              DISPLAY: begin
                            state <= DONE;
                       end

                DONE: begin
                            //back to idle if execute back to low
                            if(execute == 1'b0) state <= IDLE;
                        end
                default: state <= IDLE;
          endcase

      end //end always
endmodule
```

```
`timescale 1ns / 1ps

module l3_tb();

    /* clock and instruction control */
    reg clk, exec;

    /* instruction: to be generated in the testbench part */
    reg [0:10] instr;
    /*  4 different fields in an instruction */
    wire [0:2] opcode;  /* instr [0:2] */
    wire [1:0] reg_x;   /* instr [3:4] */
    wire [1:0] reg_y;   /* instr [5:6] */
    wire [3:0] imm;     /* instr [7:10] */

    /* control state machine outputs */
    wire extern, gout, iout, ain, gin, dpin, rdx, rdy, wrx, add_sub;

    /* state machine states */
    wire [3:0] smstate;

    /* latch a output */
    wire [3:0] a_out_data;
    /* latch g output */
    wire [3:0] g_out_data;
    /* latch dp output */
    wire [3:0] dp_out_data;

    /* mux 2 output */
    wire [3:0] mux2_output;

    /* adder output */
    wire [3:0] adder_out;



    l3_SM sm(.clk(clk),
            .execute(exec),
            .operation(opcode),
            ._Extern(extern),
            .Gout(gout),
            .Iout(iout),
            .Ain(ain),
            .Gin(gin),
```

```
        .DPin(dpin),
        .RdX(rdx),
        .RdY(rdy),
        .WrX(wrx),
        .add_sub(add_sub),
        .cur_state(smstate));

wire [3:0] rf_datain, rf_dataout;
wire [6:0] seg;
wire [7:0] an;

RF rf(.fpga_clk(clk),
      .DataIn(rf_datain),
      .AddrX(reg_x),
      .AddrY(reg_y),
      .RdX(rdx),
      .RdY(rdy),
      .WrX(wrx),
      .sm_state(smstate),
      .Dataout(rf_dataout),
      .seg(seg),
      .an(an));

 A a(.Ain(rf_dataout),
    .load_en(ain),
    .Aout(a_out_data));

 A g(.Ain(adder_out),
    .load_en(gin),
    .Aout(g_out_data));

        A dp(.Ain(rf_dataout),
    .load_en(dpin),
    .Aout(dp_out_data));

        mux_2_to_1 m2(.in1(imm),
          .in0(rf_dataout),
          .sel(iout),
          .mux_output(mux2_output));

 l2_adder adder(.in_A(a_out_data),
              .in_B(mux2_output),
              .add_sub(add_sub),
              .adder_out(adder_out));
```

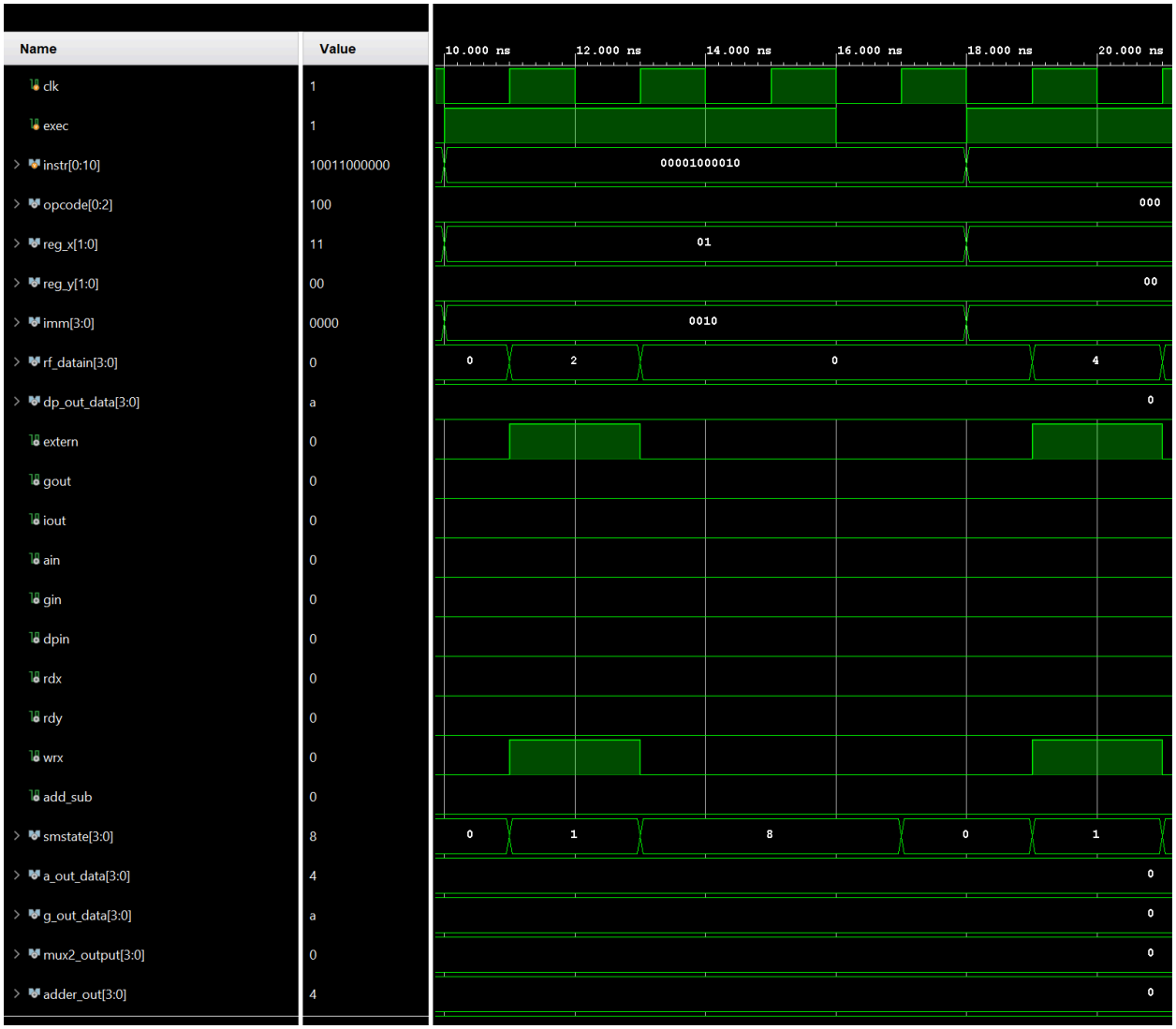
```
 modified_mux m1(.input_data(imm),
          .G_data(g_out_data),
          .Gout(gout),
          ._Extern(extern),
          .mux_output(rf_datain));


// clock generation
initial begin
   clk = 0;
end
always #1 clk = !clk;


assign opcode = instr[0:2];
assign reg_x = instr[3:4];
assign reg_y = instr[5:6];
assign imm = instr[7:10];

// insturction test bench
always
begin
   // initialization
   exec = 0;
   //  op = 3'b000, reg_x = 2'b00, reg_y = 2'b00, imm = 4'b0000
   instr = 11'b00000000000;
   #2;

   // load 1 to reg 0
   exec = 1;
   // op = 3'b000, reg_x = 2'b00, reg_y = 2'b00, imm = 4'b0001
   instr = 11'b00000000001;
   #6;
   // end of load 1 to reg 0
   // now, r0 = 1
   exec = 0;
   #2;

   // load 2 to reg 1
   exec = 1;
   // op = 3'b000, reg_x = 2'b01, reg_y = 2'b00, imm = 4'b0010
   instr = 11'b00001000010;
   #6;
```

```
 // end of load 2 to reg 1
 // now r1 = 2
 exec = 0;
 #2;

 // load 4 to reg 2
 exec = 1;
 // op = 3'b000, reg_x = 2'b10, reg_y = 2'b00, imm = 4'b0100
 instr = 11'b00010000100;
 #6;
 // end of load 4 to reg 2
 // now r2 = 4
 exec = 0;
 #2;

// load 8 to reg 3
 exec = 1;
 // op = 3'b000, reg_x = 2'b11, reg_y = 2'b00, imm = 4'b1000
 instr = 11'b00011001000;
 #6;
 // end of load 8 to reg 3
 // now r3 = 8
 exec = 0;
 #2;

 // move r3 to r2
 exec = 1;
 // op = 3'b001, reg_x = 2'b10, reg_y = 2'b11, imm = 4'b1000;
 instr = 11'b00110111000;
 #10;
 // end of move r3 to r2
 // now r2 = 8
 exec = 0;
 #2;

 //  add: r2 = r2 + r1
 exec = 1;
 // op = 3'b011, reg_x = 2'b10, reg_y = 2'b01, imm = 4'b1000;
 instr = 11'b01110011000;
 #10;
 // end of r2 = r2 + r1
 // now r2 = 10
 exec = 0;
 #2;
```

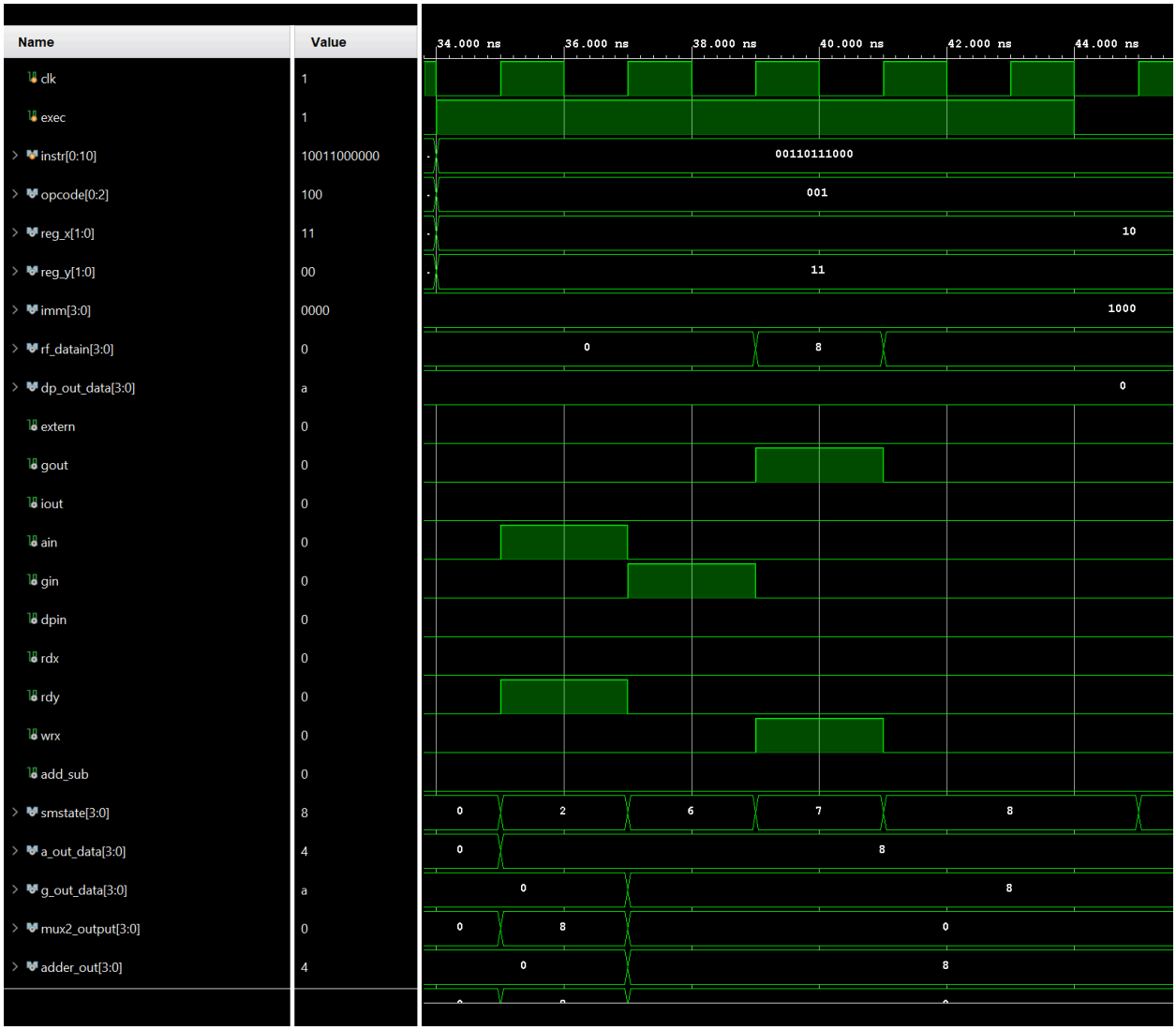
```
        //  sub: r3 = r3 - r0
        exec = 1;
        // op = 3'b010, reg_x = 2'b11, reg_y = 2'b00, imm = 4'b1000
        instr = 11'b01011001000;
        #10;
        // end of r3 = r3 - r0
        // now r3 = 7
        exec = 0;
        #2;

        //  subi: r3 = r3 - imm
        exec = 1;
        // op = 3'b110, reg_x = 2'b11, reg_y = 2'b00, imm = 4'b1000
        instr = 11'b11011000011;
        #10;
        // end of r3 = r3 - imm
        // now r3 = 4
        exec = 0;
        #2;

        //  addi: r3 = r3 + imm
        exec = 1;
        // op = 3'b111, reg_x = 2'b11, reg_y = 2'b00, imm = 4'b1000
        instr = 11'b11111000110;
        #10;
        // end of r3 = r3 + imm
        // now r3 = 10
        exec = 0;
        #2;

        //  display:
        exec = 1;
        // op = 3'b100, reg_x = 2'b11, reg_y = 2'b00, imm = 4'b0000
        instr = 11'b10011000000;
        #6;
        // end of display
        exec = 0;
        #2;

    end

endmodule
```

1st instruction: Load 1 to reg 0

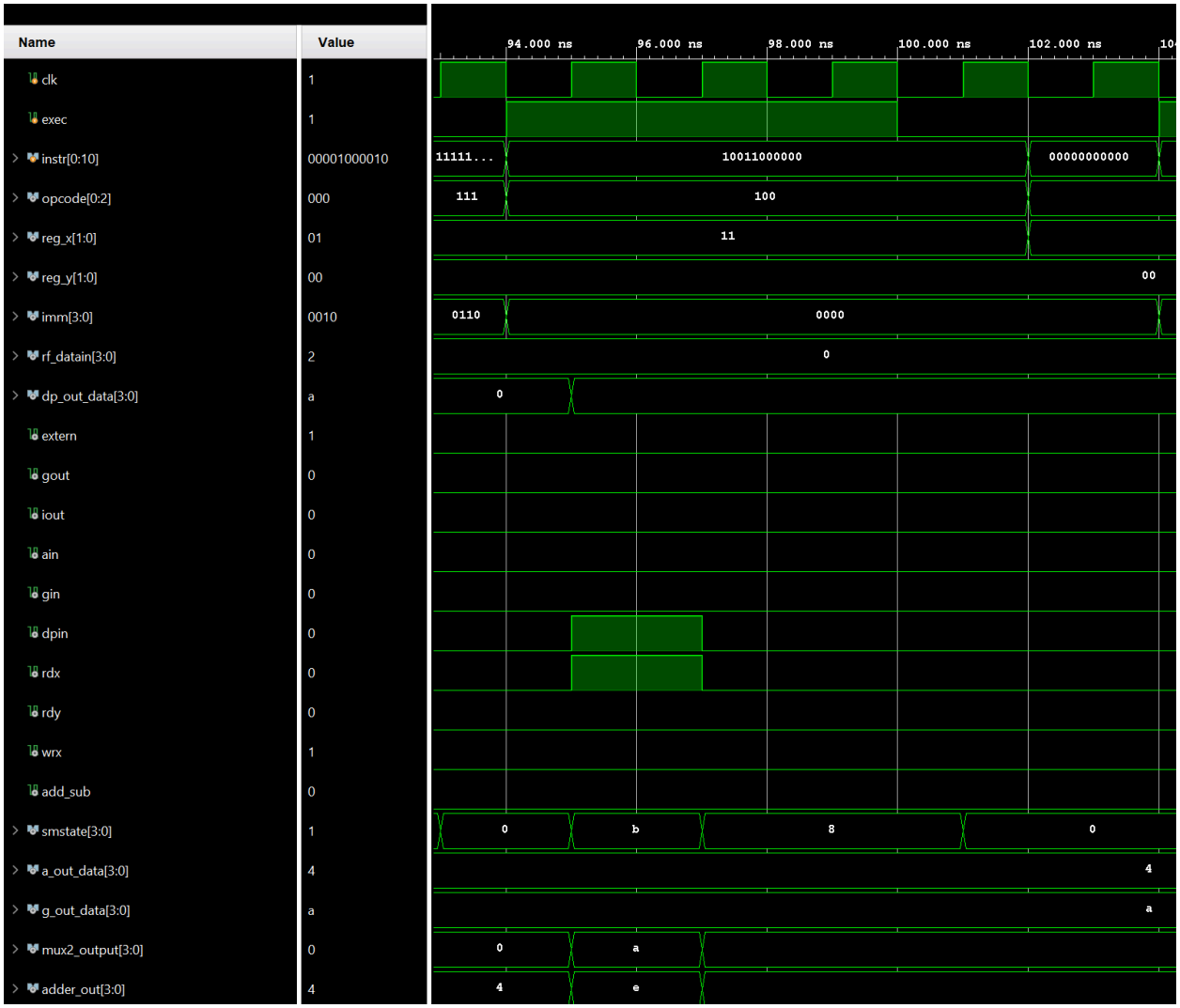
2nd instruction: Load 2 to reg 1



5th instruction: Move (copy) reg 3 to reg 1



last instruction: display reg 3



• What problems did you encounter while implementing and testing your system?

We did not use the predetermined name for the output of mux 2 which was causing undefined wire errors. For some time, we instantiated states, but did not define them which caused the simulation to obviously not enter states that we expected it to enter. We also called “add” instead of “addi” in the portion of the testbench that was supposed to test “addi” which caused unexpected behavior.

• Did any problems arise when demonstrating for the TA? In other words, did the TA ask you to demonstrate something that you did not think of yourself? What was the scenario that you were asked to demonstrate? Provide some thoughts about why you didn’t think of this yourself.

When asked about the meaning behind the delays in the testbench, we were not originally sure whether it was the states that needed time or the “blocks” in the block diagram that were needing the time. Rikesh clarified that each state needs time to process its job. We allocated 2 units of time for each state.

• In this design, we moved to a 3-bit operation encoding yet only implemented 7 operations in total. That leaves one encoding unused. Think of another operation you could envision implementing with this unused operation encoding, and describe/explain the datapath changes that would be necessary to support execution of this new operation.

We could implement a compare command by implementing the subtraction command and comparing it with the zero register to check if the values are the same. We could also include and "if equals" instructions using this method - we would add a third register Z which would only be activated for compare purposes.