Ayden Dauenhauer and Leo Chen

Prof. Yang

ELEN 122L

13 February 2024

Postlab 4

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

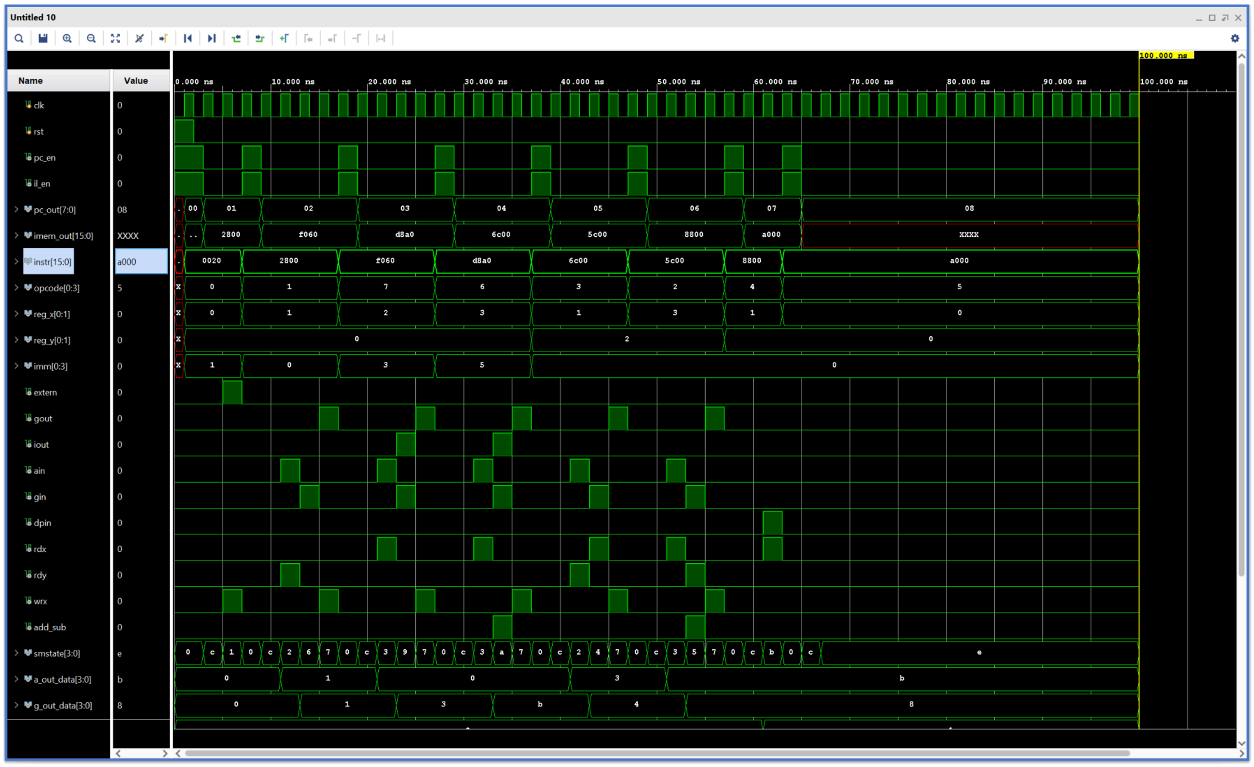
At first, we were having trouble understanding how to implement the instruction file because we thought that there would have to be an additional destination register, however the destination register was reg_x, the first register in our add/sub instruction. We also had a small error where addi was coming from read_y instead of read_x which caused the instruction to use the wrong register.

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.

We prepared well by making sure we are using every single register, as well as using every single operation. However, we did wonder where we can see the register value, as it is not apparent in the waveform.

What would happen if you didn't implement the HALT instruction? What would your system likely have done?

If we remove the last halt instruction — if there are no halt instructions at all, our state machine will be unable to fetch the next instruction as it does not exist, and the instruction displays xxxx on the waveform. This is a similar behavior to if we had implemented a halt instruction, but less graceful.



```
module l4_SM(input clk,
   input reset,
   input [0:2] operation, // opcode (part of instruction)
   output reg _Extern,
   output reg Gout,
   output reg Iout,
   output reg Ain,
   output reg Gin,
   output reg DPin,
   output reg RdX,
   output reg RdY,
   output reg WrX,
   output reg add_sub,
   output reg pc_en, // PC enable
   output reg ILin, // Latch I enable
   output [3:0] cur_state);

   /* in total, 13 states are defined */
   parameter FETCH     =  4'b0000;
   parameter LOAD      =  4'b0001;
   parameter READ_Y    =  5'b0010;
   parameter READ_X    =  4'b0011;
   parameter ADD       =  4'b0100;
   parameter SUB       =  4'b0101;
   parameter MV        =  4'b0110;
   parameter WRITE_X   =  4'b0111;
   parameter ADDI      =  4'b1001;
   parameter SUBI      =  4'b1010;
   parameter DISP      =  4'b1011;
   parameter DECODE    =  4'b1100;
   parameter HALT      =  4'b1110;

   reg [3:0] state = FETCH; // initial state being FETCH
   assign cur_state = state;




   /* TODO #3: complete the following always statement */

   always@(*)
   begin
       case(state)
           FETCH:
           /* TODO #3: control signal output for FETCH */
           begin
               _Extern = 1'b0;
               Gout = 1'b0;
               Iout = 1'b0;
               Ain = 1'b0;
               Gin = 1'b0;
               DPin = 1'b0;
               RdX = 1'b0;
               RdY = 1'b0;
               WrX = 1'b0;
               add_sub = 1'b0;
               pc_en = 1'b1;
               ILin = 1'b1;
           end
           LOAD:
           /* TODO #3: control signal output for LOAD */
           begin
               _Extern = 1'b1;
               Gout = 1'b0;
               Iout = 1'b0;
               Ain = 1'b0;
               Gin = 1'b0;
               DPin = 1'b0;
               RdX = 1'b0;
               RdY = 1'b0;
               WrX = 1'b1;
               add_sub = 1'b0;
               pc_en = 1'b0;
               ILin = 1'b0;
           end
           READ_Y:
           /* TODO #3: control signal output for READ_Y */
           begin
               _Extern = 1'b0;
               Gout = 1'b0;
               Iout = 1'b0;
               Ain = 1'b1;
               Gin = 1'b0;
               DPin = 1'b0;
               RdX = 1'b0;
               RdY = 1'b1;
               WrX = 1'b0;
               add_sub = 1'b0;
               pc_en = 1'b0;
               ILin = 1'b0;
           end
           READ_X:
           /* TODO #3: control signal output for READ_X */
```

```
begin
    _Extern = 1'b0;
    Gout = 1'b0;
    Iout = 1'b0;
    Ain = 1'b1;
    Gin = 1'b0;
    DPin = 1'b0;
    RdX = 1'b1;
    RdY = 1'b0;
    WrX = 1'b0;
    add_sub = 1'b0;
    pc_en = 1'b0;
    ILin = 1'b0;
end
ADD:
/* TODO #3: control signal output for ADD */
begin
    _Extern = 1'b0;
    Gout = 1'b0;
    Iout = 1'b0;
    Ain = 1'b0;
    Gin = 1'b1;
    DPin = 1'b0;
    RdX = 1'b1;
    RdY = 1'b0;
    WrX = 1'b0;
    add_sub = 1'b0;
    pc_en = 1'b0;
    ILin = 1'b0;
end
SUB:
/* TODO #3: control signal output for SUB */
begin
    _Extern = 1'b0;
    Gout = 1'b0;
    Iout = 1'b0;
```

```
            Ain = 1'b0;
            Gin = 1'b1;
            DPin = 1'b0;
            RdX = 1'b0;
            RdY = 1'b1;
            WrX = 1'b0;
            add_sub = 1'b1;
            pc_en = 1'b0;
            ILin = 1'b0;
        end
        /* TODO #3: control signal output for MV */
        MV:
        begin
            _Extern = 1'b0;
            Gout = 1'b0;
            Iout = 1'b0;
            Ain = 1'b0;
            Gin = 1'b1;
            DPin = 1'b0;
            RdX = 1'b0;
            RdY = 1'b0;
            WrX = 1'b0;
            add_sub = 1'b0;
            pc_en = 1'b0;
            ILin = 1'b0;
        end
        /* TODO #3: control signal output for WRITE_X */
        WRITE_X:
        begin
            _Extern = 1'b0;
            Gout = 1'b1;
            Iout = 1'b0;
            Ain = 1'b0;
            Gin = 1'b0;
            DPin = 1'b0;
            RdX = 1'b0;
            RdY = 1'b0;
            WrX = 1'b1;
            add_sub = 1'b0;
            pc_en = 1'b0;
            ILin = 1'b0;
        end
        ADDI:
        /* TODO #3: control signal output for ADDI */
        begin
            _Extern = 1'b0;
            Gout = 1'b0;
            Iout = 1'b1;
            Ain = 1'b0;
            Gin = 1'b1;
            DPin = 1'b0;
            RdX = 1'b0;
            RdY = 1'b0;
            WrX = 1'b0;
            add_sub = 1'b0;
            pc_en = 1'b0;
            ILin = 1'b0;
        end
        SUBI:
        /* TODO #3: control signal output for SUBI */
        begin
            _Extern = 1'b0;
            Gout = 1'b0;
            Iout = 1'b1;
            Ain = 1'b0;
            Gin = 1'b1;
            DPin = 1'b0;
            RdX = 1'b0;
            RdY = 1'b0;
            WrX = 1'b0;
            add_sub = 1'b1;
            pc_en = 1'b0;
```

```
            ILin = 1'b0;
        end
        DISP:
        /* TODO #3: control signal output for DISP */
        begin
            _Extern = 1'b0;
            Gout = 1'b0;
            Iout = 1'b0;
            Ain = 1'b0;
            Gin = 1'b0;
            DPin = 1'b1;
            RdX = 1'b1;
            RdY = 1'b0;
            WrX = 1'b0;
            add_sub = 1'b0;
            pc_en = 1'b0;
            ILin = 1'b0;
        end
        DECODE:
        /* TODO #3: control signal output for DECODE */
        begin
            _Extern = 1'b0;
            Gout = 1'b0;
            Iout = 1'b0;
            Ain = 1'b0;
            Gin = 1'b0;
            DPin = 1'b0;
            RdX = 1'b0;
            RdY = 1'b0;
            WrX = 1'b0;
            add_sub = 1'b0;
            pc_en = 1'b0;
            ILin = 1'b0;
        end
        /* TODO #3: control signal output for HALT */
        HALT:
           begin
               _Extern = 1'b0;
               Gout = 1'b0;
               Iout = 1'b0;
               Ain = 1'b0;
               Gin = 1'b0;
               DPin = 1'b0;
               RdX = 1'b0;
               RdY = 1'b0;
               WrX = 1'b0;
               add_sub = 1'b0;
               pc_en = 1'b0;
               ILin = 1'b0;
           end
       endcase
   end

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

   /* TODO #2
      based on the state diagram you drew in the pre-lab
      complete the following always statement in which all state transitions are
specified */

   always@(posedge clk or posedge reset)
   begin
```

```
if (reset == 1) state <= FETCH;
else
    case(state)

        FETCH:
        begin
            /* TODO #2: e.g., in FETCH, the next state is always DECODE */
            state <= DECODE;
        end

        LOAD:
        begin
            state <= FETCH;
        end

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

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

        ADD:
        begin
            state <= WRITE_X;
        end

        SUB:
        begin
            state <= WRITE_X;
        end

        MV:
        begin
            state <= WRITE_X;
        end

        WRITE_X:
        begin
            state <= FETCH;
        end

        ADDI:
        begin
            state <= WRITE_X;
        end

        SUBI:
        begin
            state <= WRITE_X;
        end

        DISP:
        begin
            state <= FETCH;
        end

        DECODE:
        begin
            /* TODO #2: in DECODE, if opcode is 000, switch to LOAD
                    complete all other transitions ... */
            if (operation == 3'b000) state <= LOAD;
            if (operation == 3'b001) state <= READ_Y;
            if (operation == 3'b010) state <= READ_X;
            if (operation == 3'b011) state <= READ_Y;
```

```
                    if (operation == 3'b100) state <= DISP;
                    if (operation == 3'b101) state <= HALT;
                    if (operation == 3'b110) state <= READ_X;
                    if (operation == 3'b111) state <= READ_X;
                end

                HALT:
                begin
                    state <= HALT;
                end

                default: state <= FETCH;

            endcase

    end //end always

endmodule
```