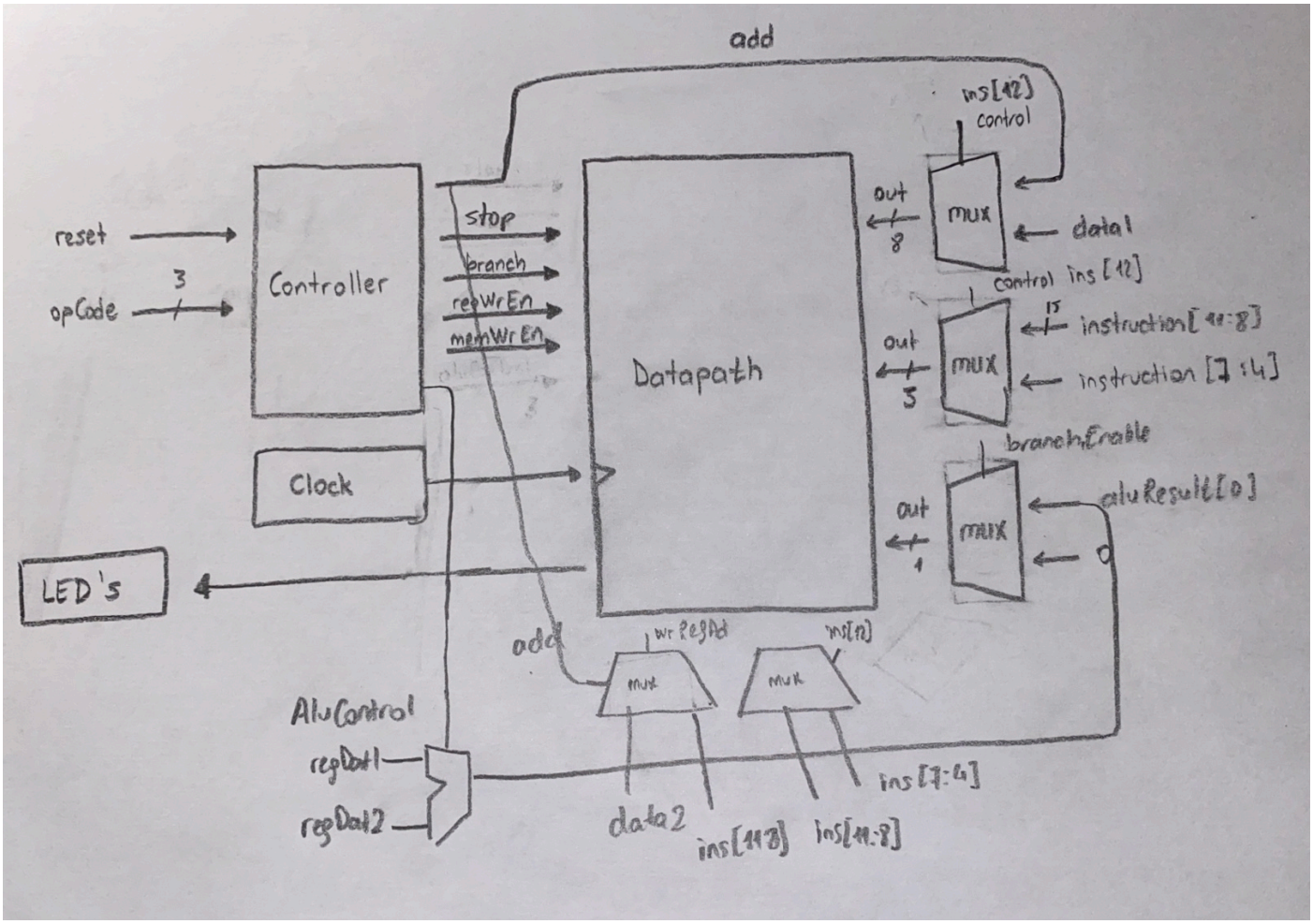
CS223-5: Digital Design Project

Can Avşar 21902111

December 25, 2020

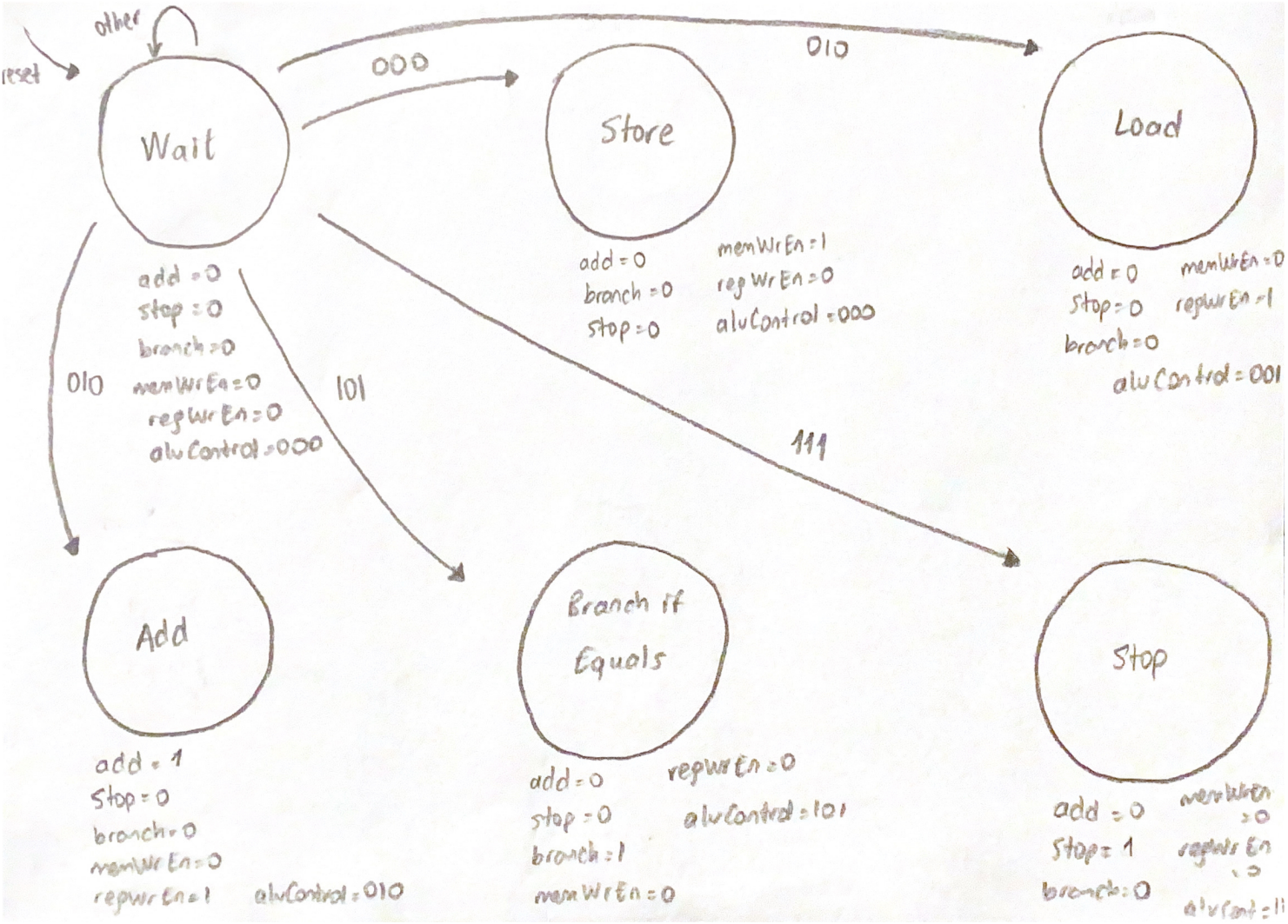
1) Block Diagram of Controller/Datapath



I used some mux and they are connected to some inputs and outputs. I only showed the controller and the datapath. Also clock is going to the datapath and led's are determined by the datapath.

It is a bit confusing because of there are only controller and datapath, the report wanted the controller/datapath diagram.

2) State Diagram of the Controller



I used 6 states on my controller and they are wait, store, load, add, branch (if equals) and stop. Each one has different operation codes and when the controller receives the instruction, the operation code which is the first 3 bits of the instruction, determines the next state.

Also, there are some variables and their conditions under each state. These are the details of this diagram. Note: The photo is a bit pale but I hope they are readable.

3) SystemVerilog Code Implementation of Single Cycle Processor and Buttons

I will include all codes with the .zip file but there are some initial codes for the processor.

Top Design:

```
module TopDesign
    (
    input logic clk,
    input logic [15:0] swIns,
    input logic [4:0] debouncer,
    output logic [15:0] insLed,
    output logic [6:0]seg,
    output logic dp,
    output logic[3:0] an
    );

    logic previous, next, getFromSwitches, getFromMemory, reset;
    logic branch, stop, add;
    logic [7:0] data1;
    logic [3:0] data2;
    logic registerWriteEnable;
    logic [3:0] writeRegisterAddress;
    logic [7:0] readRegisterData1, readRegisterData2, writeRegisterData;
    logic [4:0] romMemoryAddress = 5'b00000;
    logic [15:0] instruction;
    logic [15:0] romMemoryInstruction;
    logic memoryWriteEnable;
    logic [3:0] writeDataAddress;
    logic [3:0] memoryAddress = 4'b0000;
    logic [7:0] readRamData1, readRamData2, writeData;
    logic [2:0] aluControl;
    logic [7:0] aluResult;


    // create buttons
    Debouncer upButton( clk, debouncer[1], reset);
    Debouncer centerButton( clk, debouncer[0], getFromSwitches);
    Debouncer rightButton( clk, debouncer[3], next);
    Debouncer leftButton( clk, debouncer[2], previous);
```

```
Debouncer downButton( clk, debouncer[4], getFromMemory);

always_ff@(posedge clk, posedge reset)
begin
   if (reset)
      begin

      romMemoryAddress <= 5'b00000;

      end

   else begin

   if (!stop)
   begin

   instruction <= 16'b1000000000000000;

      if (previous)
         if (memoryAddress == 4'b0000)
            memoryAddress <= 4'b1111;
         else
            memoryAddress <= memoryAddress - 1;

      else if (next)
          if (memoryAddress == 4'b1111)
              memoryAddress <= 4'b0000;
          else
              memoryAddress <= memoryAddress + 1;

       else if (getFromMemory)
         begin
         instruction <= romMemoryInstruction;
         if (romMemoryAddress == 5'b11111)
            romMemoryAddress <= 5'b00000;
         else
            romMemoryAddress <= romMemoryAddress + 1;
      end

      else if (getFromSwitches)
         instruction <= swIns;

      else if (branch)
         romMemoryAddress <= instruction[12:8];
   end
   end
end
```

```
    Controller controlUnit( reset, instruction[15:13], stop, add, branchEnable, registerWriteEnable,
memoryWriteEnable, aluControl);

    Multiplexer multiplexer1(instruction[12], readRegisterData1, instruction[7:0], writeData);
    Multiplexer multiplexer2(instruction[12], readRamData1, instruction[7:0], data1);
    Multiplexer multiplexer3(add, data1, aluResult, writeRegisterData);

    MultiplexerBranch multiplexer4(branchEnable, 0, aluResult[0], branch);

    MultiplexerAddress multiplexer5(instruction[12], instruction[7:4], instruction[11:8],
writeDataAddress);
    MultiplexerAddress multiplexer6(instruction[12], instruction[7:4], instruction[11:8], data2);
    MultiplexerAddress multiplexer7(add, data2, instruction[11:8], writeRegisterAddress);

    DataMemory memory(clk, reset, memoryWriteEnable, writeDataAddress, writeData,
instruction[3:0], memoryAddress, readRamData1, readRamData2 );
    InstructionMemory instructions(romMemoryAddress, romMemoryInstruction, insLed);
    ALU alu(readRegisterData1, readRegisterData2, aluControl, aluResult);
    RegisterFile registerFile(clk, reset, registerWriteEnable, writeRegisterAddress, writeRegisterData,
instruction[3:0], instruction[7:4], readRegisterData1, readRegisterData2);
    SevenSegmentDisplay sevenSegmentDisplay(clk, memoryAddress, 1, readRamData2[7:4],
readRamData2[3:0], seg, dp, an);

endmodule
```

Controller:

```
module Controller(
    input logic reset,
    input logic [2:0] operationCode,
    output logic stop, add, branch, registerWriteEnable, memoryWriteEnable,
    output logic [2:0] aluControl);

    always_comb
    begin
        if( reset)
        begin
            add = 0;
            branch = 0;
            stop = 0;
            memoryWriteEnable = 0;
            registerWriteEnable = 0;
            aluControl = 3'b000;
        end
        else begin
```

```
case(operationCode)
    3'b000: // store
    begin
        add = 0;
        branch = 0;
        stop = 0;
        memoryWriteEnable = 1;
        registerWriteEnable = 0;
        aluControl = 3'b000;
    end

    3'b001:  //load
    begin
        add = 0;
        branch = 0;
        stop = 0;
        memoryWriteEnable = 0;
        registerWriteEnable = 1;
        aluControl = 3'b001;
    end


    3'b010: // add
    begin
        add = 1;
        stop = 0;
        branch = 0;
        memoryWriteEnable = 0;
        registerWriteEnable = 1;
        aluControl = 3'b010;
    end

    3'b101: // branch if equals
    begin
        add = 0;
        stop = 0;
        branch = 1;
        memoryWriteEnable = 0;
        registerWriteEnable = 0;
        aluControl = 3'b101;
    end

    3'b111: // stop
    begin
        add = 0;
        stop = 1;
        branch = 0;
        memoryWriteEnable = 0;
        registerWriteEnable = 0;
```

```
            aluControl = 3'b000;
        end

        default:
        begin
            add = 0;
            stop = 0;
            branch = 0;
            memoryWriteEnable = 0;
            registerWriteEnable = 0;
            aluControl = 3'b000;
        end
      endcase
    end
  end
endmodule
```

Multiplexer:

```
module Multiplexer(
  input logic controlInput,
  input logic [7:0] a, b,
  output logic [7:0] out);

always_comb
  begin
    case (controlInput)
      0: out = a;
      1: out = b;
    endcase
  end
endmodule
```

ALU:

```
module ALU(
input logic [7:0] a,
input logic [7:0] b,
input logic [2:0] aluControl,
output logic [7:0] aluResult);

always_comb
 begin
    case(aluControl)
    3'b000: aluResult = 0;
```

```
    3'b001: aluResult = 0;
    3'b010: aluResult = a + b;
    3'b101: aluResult = ~(a ^ b); // xnor
    3'b111: aluResult = 0;
    default: aluResult = 0;
    endcase
end
endmodule
```

Data Memory:

```
module DataMemory(
input logic clk,
input logic reset,
input logic writeEnable,
input logic [3:0] writeAddress,
input logic [7:0] writeData,
input logic [3:0] readAddress1,
input logic [3:0] readAddress2,
output logic [7:0] readData1,
output logic [7:0] readData2);
logic [7:0] mem [15:0];
always_ff @(posedge clk)
   begin
   if(reset) begin
            mem[0] <= 8'b0;
            mem[1] <= 8'b0;
            mem[2] <= 8'b0;
            mem[3] <= 8'b0;
            mem[4] <= 8'b0;
            mem[5] <= 8'b0;
            mem[6] <= 8'b0;
            mem[7] <= 8'b0;
            mem[8] <= 8'b0;
            mem[9] <= 8'b0;
            mem[10] <= 8'b0;
            mem[11] <= 8'b0;
            mem[12] <= 8'b0;
            mem[13] <= 8'b0;
            mem[14] <= 8'b0;
            mem[15] <= 8'b0;
        end
   else
      if (writeEnable == 1)
      mem[writeAddress] <= writeData;
   end

   assign readData1 = mem[readAddress1];
```

```
    assign readData2 = mem[readAddress2];


endmodule
```

Register File:

```
module RegisterFile(
input logic clk,
input logic reset,
input logic regWriteEnable,
input logic [3:0] regWriteAddress,
input logic [7:0] regWriteData,
input logic [3:0] regReadAddress1,
input logic [3:0] regReadAddress2,
output logic [7:0] regReadData1,
output logic [7:0] regReadData2);
      logic [7:0] regMem[15:0];

      always_ff@ (posedge clk, posedge reset) begin
        if(reset)
            begin
                regMem[0] <= 8'b0;
                regMem[1] <= 8'b0;
                regMem[2] <= 8'b0;
                regMem[3] <= 8'b0;
                regMem[4] <= 8'b0;
                regMem[5] <= 8'b0;
                regMem[6] <= 8'b0;
                regMem[7] <= 8'b0;
                regMem[8] <= 8'b0;
                regMem[9] <= 8'b0;
                regMem[10] <= 8'b0;
                regMem[11] <= 8'b0;
                regMem[12] <= 8'b0;
                regMem[13] <= 8'b0;
                regMem[14] <= 8'b0;
                regMem[15] <= 8'b0;
            end
            else
             begin
                if(regWriteEnable == 1)
                    begin
                     regMem[regWriteAddress] <= regWriteData;
                end
            end
            end
```

```
    assign regReadData1 = regMem[regReadAddress1];
    assign regReadData2 = regMem[regReadAddress2];
endmodule
```

Instruction Memory:

```
module InstructionMemory(
input logic [4:0] readAddress,
output logic [15:0] readData, readData2);

logic [15:0] rom[31:0];

always_comb
    begin
      rom[0] = 16'b001_0_0000_00000000;
      rom[1] = 16'b001_0_0000_00010001;
      rom[2] = 16'b001_1_0010_00000000;
      rom[3] = 16'b001_1_0011_00000000;
      rom[4] = 16'b001_1_0100_00000001;
      rom[5] = 16'b101_0_1001_00100001;
      rom[6] = 16'b010_0_0011_00110000;
      rom[7] = 16'b010_0_0010_00100100;
      rom[8] = 16'b101_0_0101_0000_0000;
      rom[9] = 16'b000_0_0000_0011_0011;
      rom[10] = 16'b001_0_0000_00000000;
      rom[11] = 16'b001_0_0000_00010001;
      rom[12] = 16'b001_1_0010_00000000;
      rom[13] = 16'b001_1_0011_00000000;
      rom[14] = 16'b001_1_0100_00000001;
      rom[15] = 16'b101_0_1001_00100001;
      rom[16] = 16'b010_0_0011_00110000;
      rom[17] = 16'b010_0_0010_00100100;
      rom[18] = 16'b101_0_0101_0000_0000;
      rom[19] = 16'b000_0_0000_0011_0011;
      rom[20] = 16'b010_x_1100_1000_1001;
      rom[21] = 16'b010_x_1101_1010_1011;
      rom[22] = 16'b101_00110_1000_1101;
      rom[23] = 16'b101_00010_1001_1011;
      rom[24] = 16'b101_xxxxxxxxxxxxx;
      rom[25] = 16'b101_xxxxxxxxxxxxx;
      rom[26] = 16'b101_xxxxxxxxxxxxx;
      rom[27] = 16'b101_xxxxxxxxxxxxx;
      rom[28] = 16'b011_xxxxxxxxxxxxx;
      rom[29] = 16'b100_xxxxxxxxxxxxx;
      rom[30] = 16'b101_xxxxxxxxxxxxx;
      rom[31] = 16'b111_xxxxxxxxxxxxx;
```

```
    end

    assign readData = rom[readAddress];
    assign readData2 = rom[readAddress+1];
endmodule
```