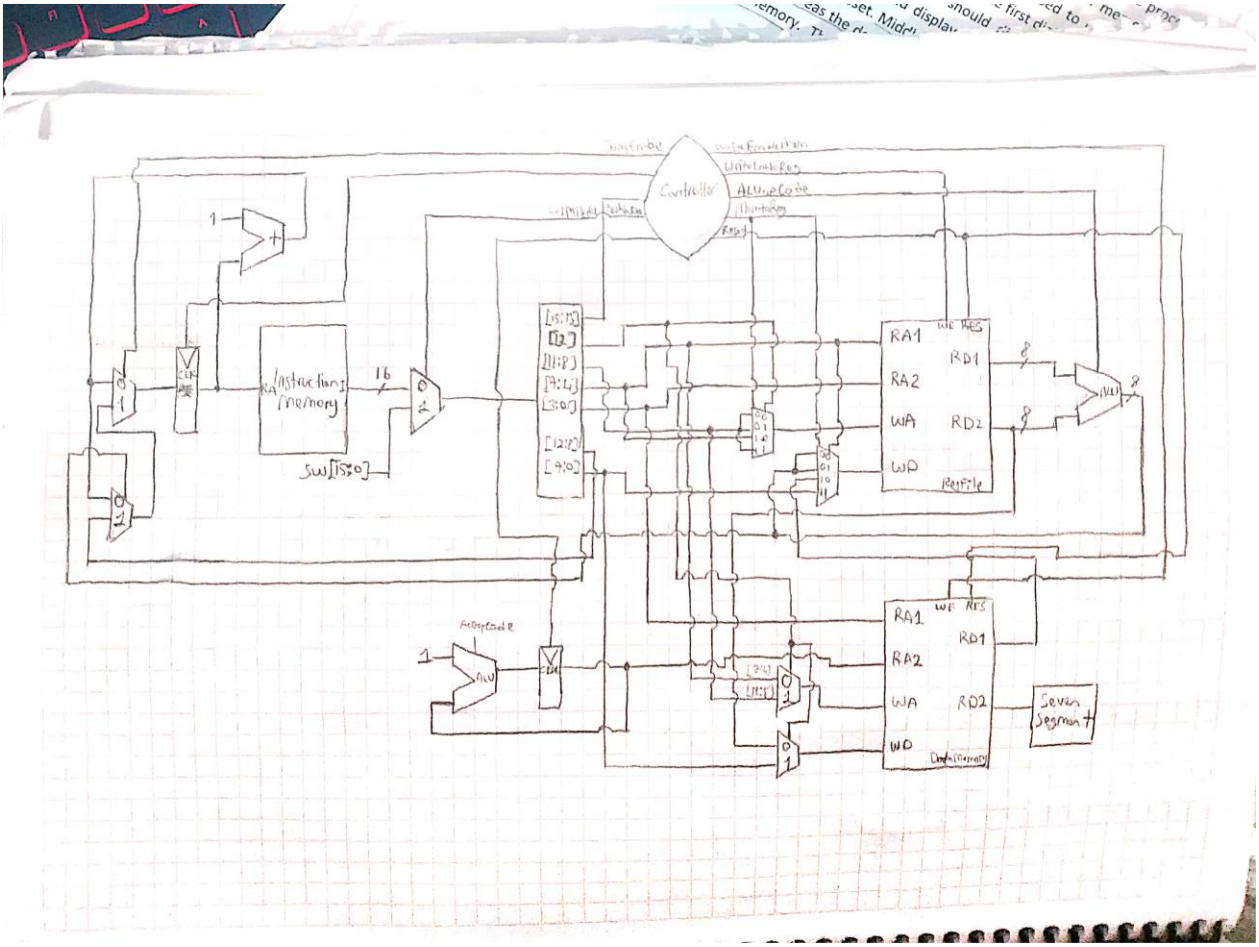
CS223Project

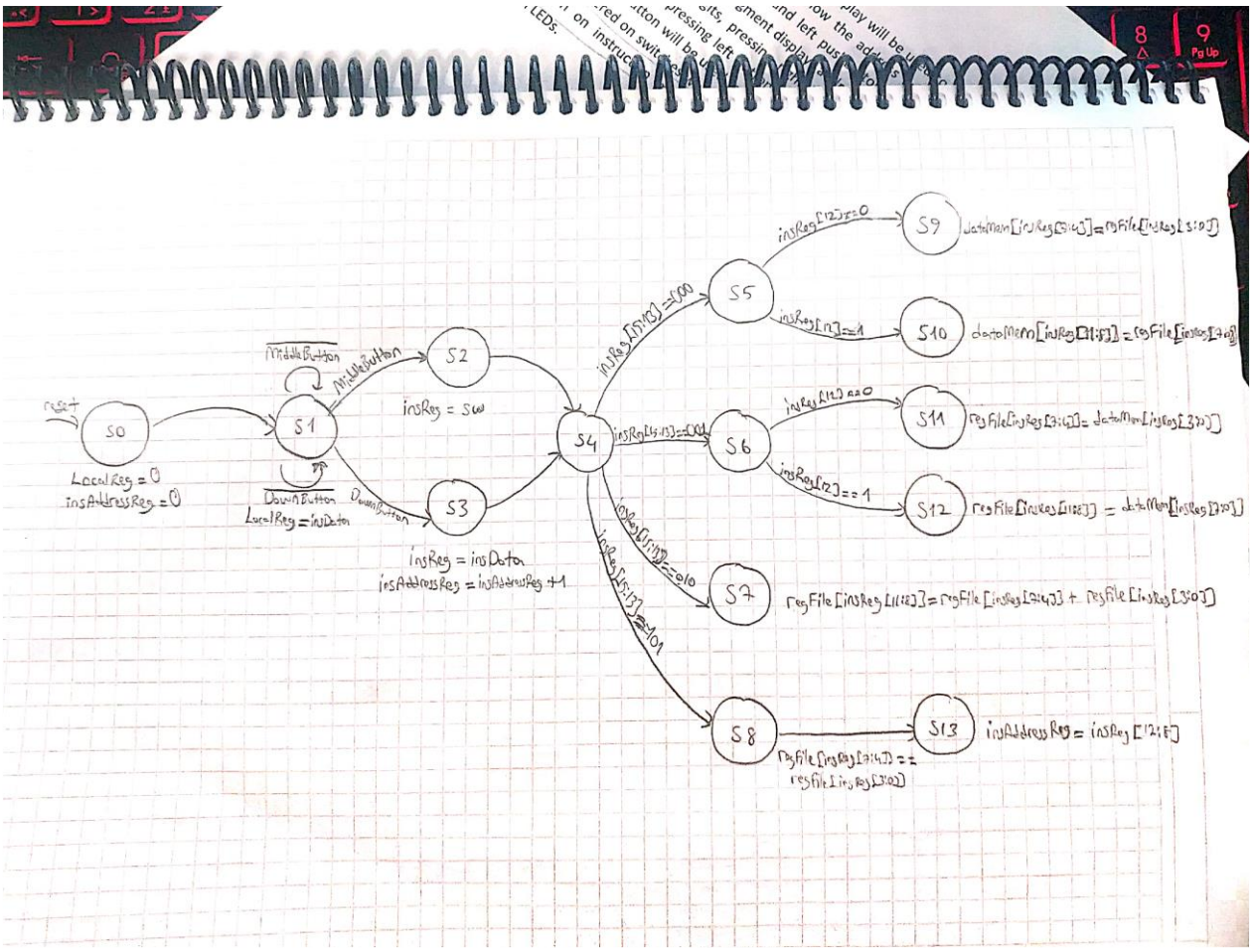
Single Cycle Processor

Deniz Semih Özal
21802414

Block Diagram of the Datapath ot the Single Cycle Processor



State Diagram of the Controller



According to opcode, state transitions are happening and button choices affects instruction register addresses.

The first step is fetching instruction from instruction memory and the instruction can be chosen from both switches or instruction memory. Then instructions go to decode step passing through register files. Then decoded values executed with ALU or another units and goes to the memory. Once the values are stored in the memory they need to be called back and writeback step starts.

Instruction Memory

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


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


  assign readData = instructionMemory[readAddress];
```

```
initial
begin
    instructionMemory[0] = 16'b001_1_0000_00000100;   // RF[0] = 4
    instructionMemory[1] = 16'b001_1_0001_00000101;   // RF[1] = 5
    instructionMemory[2] = 16'b001_1_0100_00001111;   // RF[4] = 2
    instructionMemory[3] = 16'b001_1_0101_00000110;   // RF[5] = 3
    instructionMemory[4] = 16'b001_0_0000_0000_0000;  // load value from DM[0] to RF[0]
    instructionMemory[5] = 16'b001_0_0000_0001_0001;  // load value from DM[1] to RF[1]
    instructionMemory[6] = 16'b001_1_0010_00000000;  // load 0 immediately to RF[2]
    instructionMemory[7] = 16'b001_1_0011_00000000;  // load 0 immediately to RF[3]
    instructionMemory[8] = 16'b001_1_0100_00000001;   // load 1 immediately to RF[4]
    instructionMemory[9] = 16'b001_1_0011_00000001;   // rf[3] = 1
    instructionMemory[10] = 16'b001_1_1111_00000000;  // rf[15] = 0
    instructionMemory[11] = 16'b101_10011_0010_0101;
    instructionMemory[11] = 16'b101_10011_0010_0101;
    instructionMemory[12] = 16'b010_1_1000_1111_0100;
    instructionMemory[13] = 16'b000_0_0000_1000_1000;
    instructionMemory[14] = 16'b001_0_0000_1111_1000;
    instructionMemory[15] = 16'b010_0_1001_0010_0011;
    instructionMemory[17] = 16'b001_0_0000_0010_1001;
    instructionMemory[18] = 16'b101_01011_0000_0000;
    instructionMemory[19] = 16'b000_0_0000_1111_1111;
    instructionMemory[20] = 16'b111_1111111111111;
    instructionMemory[21] = 16'b0011010101010010;
    instructionMemory[22] = 16'b0001011000100010;
    instructionMemory[23] = 16'b0001010001000010;
    instructionMemory[24] = 16'b0001100110000010;
```

```
        instructionMemory[25] = 16'b0011001000011110;
        instructionMemory[26] = 16'b0101010100100010;
        instructionMemory[27] = 16'b1001010101000010;
        instructionMemory[28] = 16'b0101001010000010;
        instructionMemory[29] = 16'b1011001000010010;
        instructionMemory[30] = 16'b0111000100100010;
        instructionMemory[31] = 16'b1001010111000010;
    end
endmodule
```

Program Counter (PC)

```
module programCounter( input logic clk, en, reset,
                       input logic[4:0] next_insAddr,
                       output logic[4:0] insAddr,
                       input logic pc_control, halt
    );

    always_ff@( posedge clk)
        if( reset)
        begin
            for (int i = 0; i < 16; i = i + 1)
                begin
                    insAddr <= 5'b0;
                end
        end
        else if( (en || pc_control) && !halt)
            insAddr <= next_insAddr;

endmodule
```

Register File

```
module registerFile( input logic clk, reset, writeEnableRegister,
                     input logic[3:0] readAddress1, readAddress2,
                     input logic[3:0] writeAddress,
                     input logic[7:0] writeData,
                     output logic[7:0] readData1, readData2);


   logic[7:0] registerFile[15:0];


   always_ff@(posedge clk)
      if( reset)
      begin
         for (int i = 0; i < 16; i = i + 1)
            begin
               registerFile[i] <= 8'b0;
            end
      end
      else if(writeEnableRegister) registerFile[writeAddress] <= writeData;


   assign readData1 = registerFile[readAddress1];
   assign readData2 = registerFile[readAddress2];
endmodule
```

Data Memory

```
module dataMemory( input logic clk, reset, writeEnableMemory,
                   input logic[3:0] readAddress1, readAddress2,
                   input logic[3:0] writeAddress,
                   input logic[7:0] writeData,
                   output logic[7:0] readData1, readData2);
```

```
    logic[7:0] dataMemory[15:0];

    always_ff@(posedge clk)
        if( reset)
        begin
            for (int i = 0; i < 16; i = i + 1)
                begin
                    dataMemory[i] <= 8'b0;
                end
        end
        else if ( writeEnableMemory) dataMemory[writeAddress] <= writeData;

    assign readData1 = dataMemory[readAddress1];
    assign readData2 = dataMemory[readAddress2];

endmodule
```

Arithmetic Logic Unit

```
module ALU( input logic[7:0] in0, in1,
            input logic [1:0] ALUcontrol,
            output logic[7:0] result,
            output logic zero
    );
    logic[7:0] sum;
    adder add( in0, in1, sum);

    always_comb
        case(ALUcontrol)
```

```
        0: result = sum; // add
        1: zero = ( in0 == in1);
    endcase
endmodule
```

Datapath

```
module datapath( input logic clk, reset,
                 input logic[15:0] instruction,
                 input logic writeEnableReg,
                 input logic[7:0] rData1_data,
                 input logic memToReg, aluControl, add, beq,
                 input logic[4:0]  insAddr,
                 output logic[3:0] wAddr_data,
                 output logic[4:0] next_insAddr,
                 input logic[3:0]  wAddr_reg,
                 output logic[7:0] wData_reg,
                 output logic pc_beq,
                 output logic[7:0] wData_data
   );


   logic[7:0] alu_A, alu_B, aluResult;
   mux2to1#(4) wAddr_data_mux( instruction[7:4], instruction[11:8], instruction[12],
wAddr_data);
   mux2to1#(8) wData_data_mux( aluB, instruction[7:0], instruction[12], wData_data);


   logic zeroFlag;
   assign pc_beq = zeroFlag & beq;


   logic temp_add;
   always_comb
```

```
    if( instruction[12] || add)
        temp_add = 1;
    else temp_add = 0;

  logic not_add;
  assign not_add = instruction[12] && !add;

  logic[7:0] res_mux;
  logic[7:0] wData_regFile;
  logic[3:0] wAddress_regFile;

  logic[4:0] next_addr;
  logic [7:0] rData1_reg;
  logic [7:0] rData2_reg;

  adder#(5) adder1( insAddr, 5'b00001, next_addr);
  mux2to1#(5) counter_mux( next_addr, instruction[12:8], pcBranch, next_insAddr);

  registerFile regFile( clk, reset, writeEnableReg, instruction[7:4], instruction[3:0], wAddr_reg, wData_regFile, rData1_reg, rData2_reg );
  mux2to1#(4) wAddr_reg_mux( instruction[7:4], instruction[11:8], temp_add, wAddress_regFile);
  mux2to1#(8) wData_reg_mux( res_mux, instruction[7:0], not_add, wData_regFile);
  mux2to1#(8) reset_mux( aluResult, rData1_data, memToReg, res_mux);

  ALU alu( aluA, aluB, aluControl, aluResult, zeroFlag);
Endmodule
```

Mux2to1

```
module mux2to1 #(parameter WIDTH = 8)(
    input logic[WIDTH - 1:0] in0, in1,
    input logic sel,
    output logic[WIDTH - 1:0] out
  );
  assign out = sel ? in1 : in0;
endmodule
```

Controller

```
module controller( input logic[2:0] operation,
                   output logic writeEnableMem, writeEnableReg,
                   output logic aluControl, memToReg, add, branch, halt
  );

  logic[6:0] control_signals;

  assign { writeEnableReg, writeEnableMem, memToReg, add, aluControl, branch, halt} =
control_signals;

  always_comb
    case(operation)
      3'b000: control_signals <= 7'b0100000;
      3'b001: control_signals <= 7'b1010000;
      3'b010: control_signals <= 7'b1001000;
      3'b101: control_signals <= 7'b0000110;
      3'b111: control_signals <= 7'b0000001;
      default: control_signals <= 7'bxxxxxxx;
    endcase

endmodule
```

Top Level Design

```
module topDesign( input logic clk,
                  input logic[15:0] instruction,
                  input logic resetButton, displayNextButton, displayPrevButton, runInstructionButton, nextInstructionButton,
                  output logic[15:0] instructionFromMemory,
                  output logic [6:0] seg,
                  output logic dp,
                  output logic [3:0] an
                  );


    logic[3:0] addr_reg;
    logic[7:0] data_reg;


    logic pulse1, pulse2, pulse3, pulse4, pulse5;
    debounce d1( clk, resetButton, pulse1); // reset
    debounce d2( clk, displayNextButton, pulse2); // displayNext
    debounce d3( clk, displayPrevButton, pulse3); // displayPrev
    debounce d4( clk, runInstructionButton, pulse4); // instructionRun
    debounce d5( clk, nextInstructionButton, pulse5); // instructionNext


    logic[4:0] insAddr;
    logic[4:0] next_insAddr;
    logic[3:0] addr_data;
    logic[7:0] data_data;
    logic beq;
    logic writeEnableMem;
    logic halt;
    logic[3:0] wAddr_data;
```

```
    logic[7:0] wData_data;
    logic[7:0] rData1_data;
    logic reset;
    logic [15:0] instruction_selection;
    logic read_addr = instruction_selection [3:0];
    programCounter pc( clk, pulse5, pulse1, next_insAddr, insAddr, beq, halt);
    instructionMemory instruction_memory( insAddr, instructionFromMemory);


    always_ff@(posedge clk)
        if(reset) instruction_selection <= 16'b0;
        else if (pulse4) instruction_selection <= instruction;
        else if (pulse5) instruction_selection <= instructionFromMemory;
        else instruction_selection <= instruction_selection;


    singleCyleProcessor mips( clk, pulse1, instruction_selection, rData1_data, insAddr,
next_insAddr, writeEnableMem, wAddr_data,
                    addr_reg, data_reg, beq, halt, wData_data);


    dataMemory dataMemory( clk, writeEnableMem, pulse1, wAddr_data, wData_data,
read_addr, addr_data, rData1_data, data_data);




    SevSeg_4digit sevenSeg( clk, addr_data, 5'b11111, data_data[7:4], data_data[3:0], seg, dp,
an); //mem



      always_ff @(posedge clk)
```

```
    begin
     if( pulse2) addr_data <= addr_data + 1;
        else if( pulse3) addr_data <= addr_data - 1;
        else addr_data <= addr_data;
    end


endmodule
```

Seven Segment

```
module SevSeg_4digit(
 input clk,
 input [3:0] in3, in2, in1, in0, //user inputs for each digit (hexadecimal value)
 output [6:0]seg, logic dp, // just connect them to FPGA pins (individual LEDs).
 output [3:0] an   // just connect them to FPGA pins (enable vector for 4 digits active low)
 );

// divide system clock (100Mhz for Basys3) by 2^N using a counter, which allows us to multiplex at lower speed
localparam N = 18;
logic [N-1:0] count = {N{1'b0}}; //initial value
always@ (posedge clk)
        count <= count + 1;



logic [4:0]digit_val; // 7-bit register to hold the current data on output
logic [3:0]digit_en;  //register for the 4 bit enable

always@ (*)
 begin
 digit_en = 4'b1111; //default
```

```
digit_val = in0; //default

 case(count[N-1:N-2]) //using only the 2 MSB's of the counter

  2'b00 :  //select first 7Seg.
   begin
    digit_val = {1'b0, in0};
    digit_en = 4'b1110;
   end

  2'b01:  //select second 7Seg.
   begin
    digit_val = {1'b0, in1};
    digit_en = 4'b1101;
   end

  2'b10:  //select third 7Seg.
   begin
    digit_val = {1'b1, in2};
    digit_en = 4'b1011;
   end

  2'b11:  //select forth 7Seg.
   begin
    digit_val = {1'b0, in3};
    digit_en = 4'b0111;
   end
 endcase
```

```
 end


//Convert digit number to LED vector. LEDs are active low.
logic [6:0] sseg_LEDs;
always @(*)
 begin
  sseg_LEDs = 7'b1111111; //default
  case( digit_val)
   5'd0 : sseg_LEDs = 7'b1000000; //to display 0
   5'd1 : sseg_LEDs = 7'b1111001; //to display 1
   5'd2 : sseg_LEDs = 7'b0100100; //to display 2
   5'd3 : sseg_LEDs = 7'b0110000; //to display 3
   5'd4 : sseg_LEDs = 7'b0011001; //to display 4
   5'd5 : sseg_LEDs = 7'b0010010; //to display 5
   5'd6 : sseg_LEDs = 7'b0000010; //to display 6
   5'd7 : sseg_LEDs = 7'b1111000; //to display 7
   5'd8 : sseg_LEDs = 7'b0000000; //to display 8
   5'd9 : sseg_LEDs = 7'b0010000; //to display 9
   5'd10: sseg_LEDs = 7'b0001000; //to display a
   5'd11: sseg_LEDs = 7'b0000011; //to display b
   5'd12: sseg_LEDs = 7'b1000110; //to display c
   5'd13: sseg_LEDs = 7'b0100001; //to display d
   5'd14: sseg_LEDs = 7'b0000110; //to display e
   5'd15: sseg_LEDs = 7'b0001110; //to display f
   5'd16: sseg_LEDs = 7'b0110111; //to display "="
   default : sseg_LEDs = 7'b0111111; //dash
  endcase
```

```
 end


assign an = digit_en;
assign seg = sseg_LEDs;
assign dp = 1'b1; //turn dp off



endmodule
```

Debouncer

```
module debounce(input logic clk, input logic button,output logic pulse );

logic [24:0] timer;
typedef enum logic [1:0]{S0,S1,S2,S3} states;
states state, nextState;
logic gotInput;

always_ff@(posedge clk)
   begin
      state <= nextState;
      if(gotInput)
         timer <= 25000000;
      else
         timer <= timer - 1;
   end
always_comb
   case(state)
      S0: if(button)
         begin //startTimer
```

```
            nextState = S1;
            gotInput = 1;
        end
        else begin nextState = S0; gotInput = 0; end
    S1: begin nextState = S2; gotInput = 0; end
    S2: begin nextState = S3; gotInput = 0; end
    S3: begin if(timer == 0) nextState = S0; else nextState = S3; gotInput = 0; end
    default: begin nextState = S0; gotInput = 0; end
    endcase

assign pulse = ( state == S1 );
endmodule
```

Constraint

Clock signal

```
set_property PACKAGE_PIN W5 [get_ports clk]
        set_property IOSTANDARD LVCMOS33 [get_ports clk]
 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

# Switches
set_property PACKAGE_PIN V17 [get_ports {instruction[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[0]}]
set_property PACKAGE_PIN V16 [get_ports {instruction[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[1]}]
set_property PACKAGE_PIN W16 [get_ports {instruction[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[2]}]
set_property PACKAGE_PIN W17 [get_ports {instruction[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[3]}]
set_property PACKAGE_PIN W15 [get_ports {instruction[4]}]
```

```
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[4]}]
set_property PACKAGE_PIN V15 [get_ports {instruction[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[5]}]
set_property PACKAGE_PIN W14 [get_ports {instruction[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[6]}]
set_property PACKAGE_PIN W13 [get_ports {instruction[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[7]}]
set_property PACKAGE_PIN V2 [get_ports {instruction[8]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[8]}]
set_property PACKAGE_PIN T3 [get_ports {instruction[9]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[9]}]
set_property PACKAGE_PIN T2 [get_ports {instruction[10]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[10]}]
set_property PACKAGE_PIN R3 [get_ports {instruction[11]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[11]}]
set_property PACKAGE_PIN W2 [get_ports {instruction[12]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[12]}]
set_property PACKAGE_PIN U1 [get_ports {instruction[13]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[13]}]
set_property PACKAGE_PIN T1 [get_ports {instruction[14]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[14]}]
set_property PACKAGE_PIN R2 [get_ports {instruction[15]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {instruction[15]}]

# LEDs
set_property PACKAGE_PIN U16 [get_ports {instructionFromMemory[0]}]

        set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[0]}]
```

```
set_property PACKAGE_PIN E19 [get_ports {instructionFromMemory[1]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[1]}]
set_property PACKAGE_PIN U19 [get_ports {instructionFromMemory[2]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[2]}]
set_property PACKAGE_PIN V19 [get_ports {instructionFromMemory[3]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[3]}]
set_property PACKAGE_PIN W18 [get_ports {instructionFromMemory[4]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[4]}]
set_property PACKAGE_PIN U15 [get_ports {instructionFromMemory[5]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[5]}]
set_property PACKAGE_PIN U14 [get_ports {instructionFromMemory[6]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[6]}]
set_property PACKAGE_PIN V14 [get_ports {instructionFromMemory[7]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[7]}]
set_property PACKAGE_PIN V13 [get_ports {instructionFromMemory[8]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[8]}]
set_property PACKAGE_PIN V3 [get_ports {instructionFromMemory[9]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[9]}]
set_property PACKAGE_PIN W3 [get_ports {instructionFromMemory[10]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[10]}]
set_property PACKAGE_PIN U3 [get_ports {instructionFromMemory[11]}]
```

set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[11]}]
set_property PACKAGE_PIN P3 [get_ports {instructionFromMemory[12]}]

set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[12]}]
set_property PACKAGE_PIN N3 [get_ports {instructionFromMemory[13]}]

set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[13]}]
set_property PACKAGE_PIN P1 [get_ports {instructionFromMemory[14]}]

set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[14]}]
set_property PACKAGE_PIN L1 [get_ports {instructionFromMemory[15]}]

set_property IOSTANDARD LVCMOS33 [get_ports {instructionFromMemory[15]}]

7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

```
set_property PACKAGE_PIN V7 [get_ports dp]
        set_property IOSTANDARD LVCMOS33 [get_ports dp]
set_property PACKAGE_PIN U2 [get_ports {an[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]

#Buttons
set_property PACKAGE_PIN U18 [get_ports runInstructionButton]

        set_property IOSTANDARD LVCMOS33 [get_ports runInstructionButton]
set_property PACKAGE_PIN T18 [get_ports resetButton]
        set_property IOSTANDARD LVCMOS33 [get_ports resetButton]
set_property PACKAGE_PIN W19 [get_ports displayPrevButton]

        set_property IOSTANDARD LVCMOS33 [get_ports displayPrevButton]
set_property PACKAGE_PIN T17 [get_ports displayNextButton]
        set_property IOSTANDARD LVCMOS33 [get_ports displayNextButton]
set_property PACKAGE_PIN U17 [get_ports nextInstructionButton]

        set_property IOSTANDARD LVCMOS33 [get_ports nextInstructionButton]
```