Indian Institute of Technology Palakkad
भारतीय प्रौद्योगिकी संस्थान पालक्काड
Nurturing Minds For a Better World

CO LAB 7 REPORT

Submitted By
Neeraj Krishna N (112101033)
Amithabh A (112101004)
Evans Samuel Biju (112101017)

CS2160 Computer Organisation Laboratory

Computer Science And Engineering

2 May 2023

Contents

1 Problem Statement of Lab 7

We have to add instruction cache (I-Cache) and data cache (D-Cache) to our existing RISV-V datapath RTL. Both I-Cache and D-Cache should be able to store a minimum of eight 32-bit data.

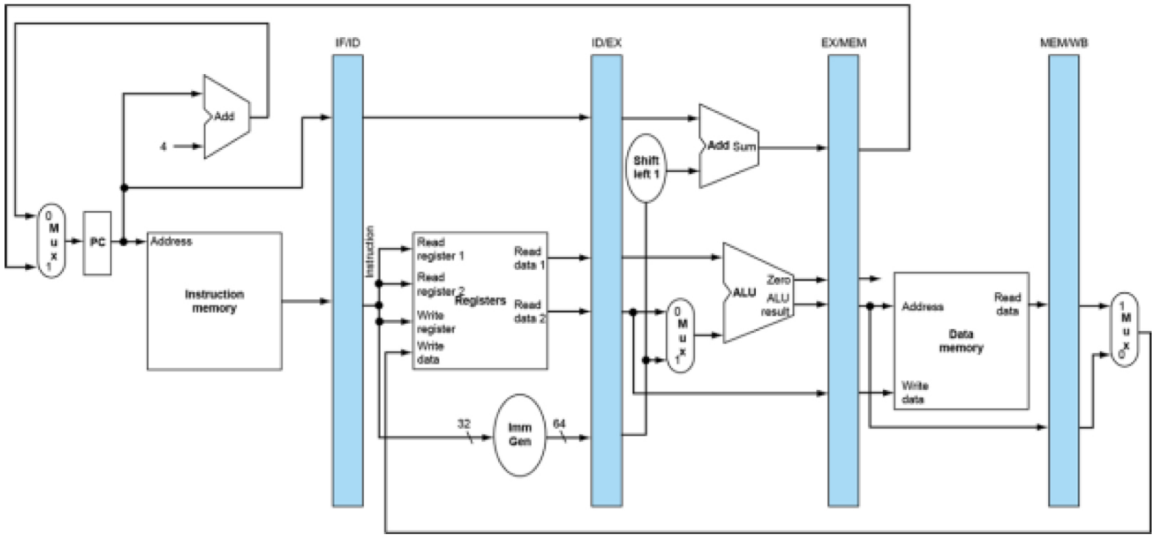


Figure 1: RISC-V Datapath

We have to store the following instructions in I-Cache. We can store 45 and -20 in the D-Cache initially.

Address	Instruction
000	addi x29, x29, 0
001	addi x30, x30, 1
010	lw x28 0(x29)
011	lw x27 0(x30)
100	add x10, x28, x29
101	sw x10, 0(x29)

Table 1: Instructions

2 Solution

Here since we need to add D-Cache and I-Cache, in the existing pipline module we would have to add an 8-length array of 32-bit registers for D-Cache and the instruction is received as an input to the pipeline module.

The complete code for the pipeline can be found in the GITHUB REPO

The code for pipeline and testbench is in the final section of this report 6.

The pipeline receives the instruction as a 256 bit string. The pipeline has an instruction cache which is 8-length array of registers of 32-bit instruction named **instruction_cache**. For the data cache, it is also an 8-length array (named **data_cache**) of registers which can hold 32-bit values which is declared inside the pipeline module. When **rst is 1**, all the values inside the register file resets to zero and the **instruction_cache** inside the pipeline module will have the correct instruction in the correct addresses. And, the whenever posedge clk happens, the pipeline which is modelled as an FSM will go from *STAGE 1* to *STAGE 4* at each posedge clk. **Instruction Fetch** happens when the current stage is *STAGE 1*. The type of instruction (i.e. whether it is load instruction, store instruction, or R-Type instruction or I-Type instruction) is obtained from the decoder. The schematic for the design is shown before the section "Code for Lab 7"[6] .i.e. here 5 (The schematic is in landscape view).

- When it is a load instruction, say for example *lw x28, 0(x29)*, it takes the value of *x29*, adds the offset 0 and stores the value in this address of the **data_cache** to *x28*.

- When it is a store instruction, say for example *sw x28, 0(x29)*, it stores the value in the register *x28* to the **data_cache** at the address *x28* added with the offset 0.
- When it is an R-Type instruction, say for example, *add x29, x28, x28*, it takes the value of *x28* and adds it with itself and writes it to the register *x29*.
- When it is an I-Type instruction, say for example, *add x29, x28, 5*, it takes the value of *x28* and adds it to *5* and writes it to the register *x29*.

3 Timing Diagrams

The timing diagram for the testbench is shown in Figure[2]

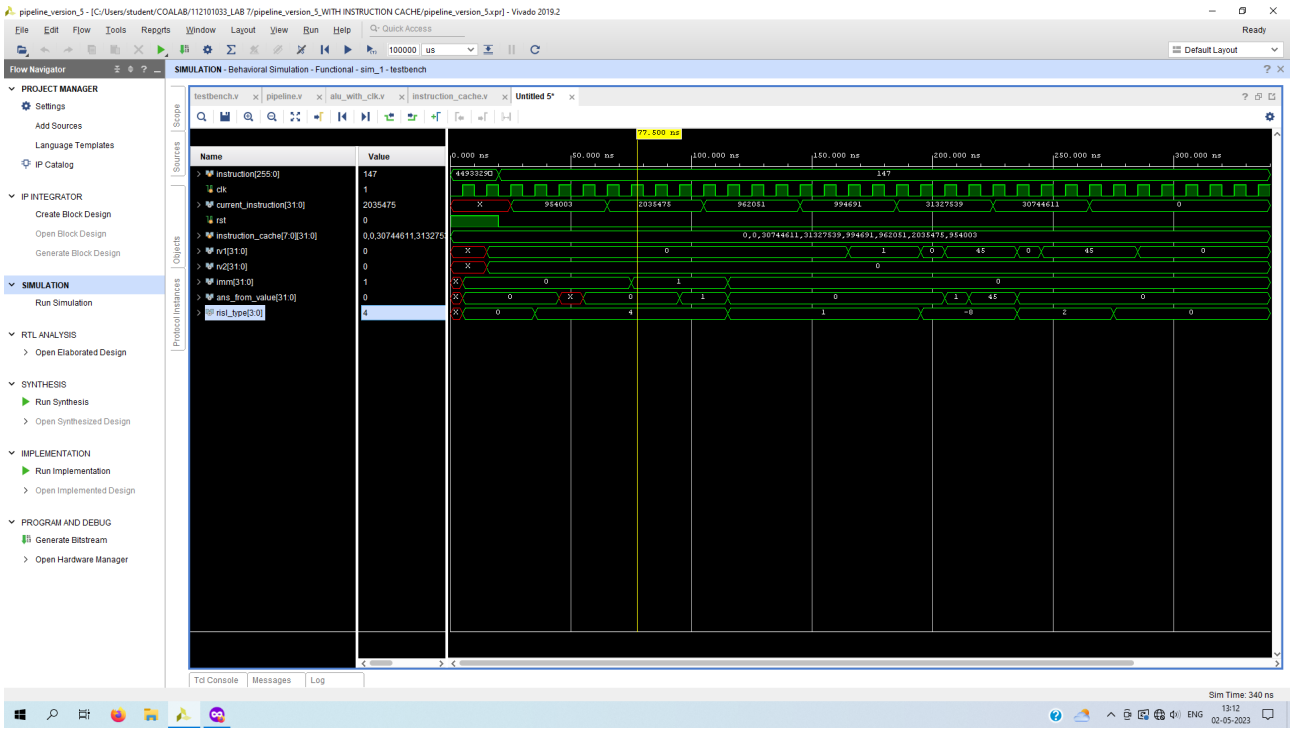


Figure 2: Simulation of Testbench

In the timing diagram, at the first posedge clk after **rst becomes zero**, the first instruction is fetched as shown by the **current_instruction** in the simulation image[2] and it completes it in 4 clock cycles, and the subsequent instruction is fetched and this goes on till the program counter reaches the end of the instruction.

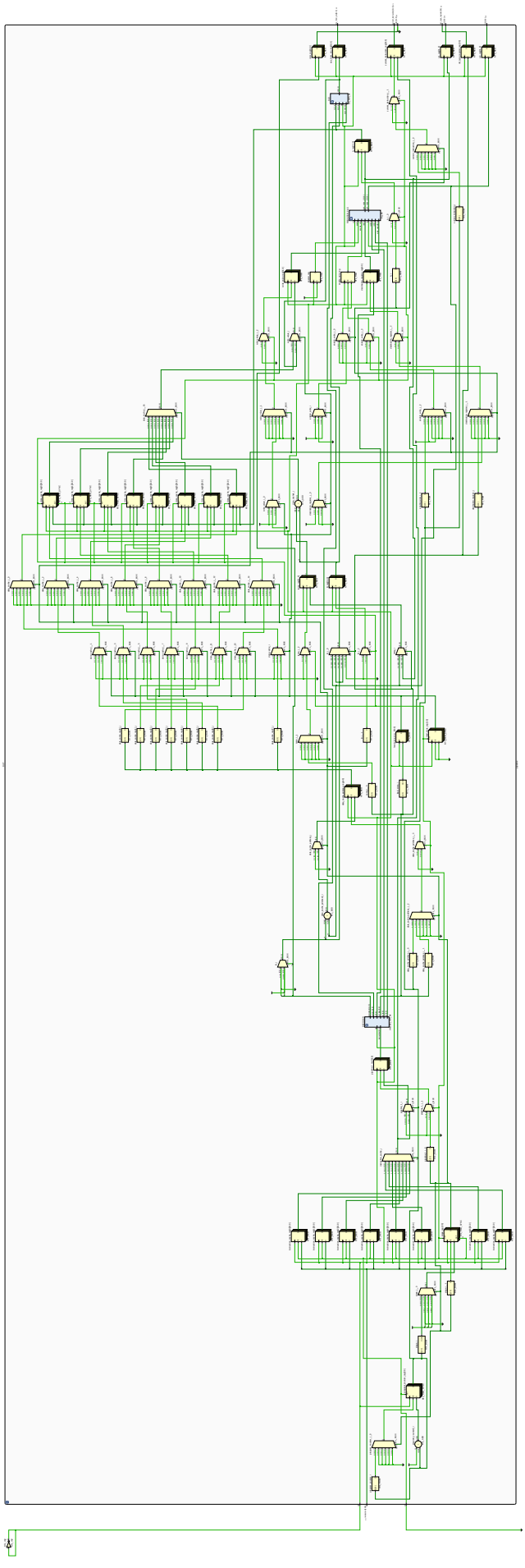
- The read value 1 is shown as rv1.
- The read value 2 is shown as rv2.
- The immediate value is shown as imm.

4 Conclusion

We have developed a complete pipeline in Verilog with instruction cache and data cache.

5 Contributions

The instruction cache and the data cache was done together and there was equal splitting of the work to the team members.



6 Code for Lab 7

6.1 Code for the pipeline

```
module pipeline(
  input [255:0] instruction,
  input clk, rst,
  output reg [31:0] rv1, rv2, imm, ans_from_alu,current_instruction,
  output reg [3:0] risl_type_register
  );
  reg [31:0] pipeline_1;
  reg [31:0] pipeline_2 [2:0];
  reg [31:0] pipeline_3 [1:0];
  wire [16:0] opcode_wire;
  wire [4:0] rs1_wire, rs2_wire, rd_wire;
  reg [4:0] destination_register;
  wire [11:0] immediate_wire;
  wire [3:0] risl_type_wire;

  parameter [3:0]
  R_type = 4'b1000,
  I_type = 4'b0100,
  S_type = 4'b0010,
  L_type = 4'b0001;
  reg [31:0] data_cache [7:0];

  decoder_with_clk DECODER(
    pipeline_1,clk,  opcode_wire, rs1_wire,
    rs2_wire, rd_wire, immediate_wire, risl_type_wire
  );

  reg [4:0] rs1, rs2, rd;
  reg enable_write, enable;
  reg [31:0] write_value;
  wire [31:0] read_value_1_wire, read_value_2_wire;
  always @ (posedge clk)
  begin
  enable <= 1'b1;
  end


  reg_file REGISTER_FILE(
    rs1_wire, rs2_wire, destination_register ,
    enable_write, write_value, clk, rst, enable,
    read_value_1_wire, read_value_2_wire);


  reg [31:0] A, B;
  wire [31:0] UH_wire, LH_wire;
  wire overflow;
  reg [4:0] data_cache_address;


  alu_with_clk ALU(A, B, opcode_wire, clk, UH_wire, LH_wire, overflow_wire);

//     always @
```

```
//     reg store_instruction;
//     reg load_instruction;
  reg [1:0] load_store;


  parameter [2:0]
  STAGE_1 = 3'd1,
  STAGE_2 = 3'd2,
  STAGE_3 = 3'd3,
  STAGE_4 = 3'd4;

  reg [2:0] stage;

  always @ (posedge clk) begin
      rv1 <= read_value_1_wire;
      rv2 <= read_value_2_wire;
      imm <= immediate_wire;
      ans_from_alu <= LH_wire;
      risl_type_register <= risl_type_wire;
  end

  always @ (posedge clk) begin
      case(risl_type_wire)
      S_type: load_store = 2'b01;
      L_type: load_store = 2'b10;
      default: load_store = 2'b00;
      endcase
  end

  reg [31:0] instruction_cache [7:0];
  reg [3:0] program_counter;

  reg [31:0] temp;
  integer i;
  always @ (posedge clk) begin
      if (rst) begin
          stage = 3'd1;
          for (i=0;i<8;i=i+1) data_cache[i] = 32'h0;
          load_store = 2'b0;
          data_cache[0] = 45;
          data_cache[1] = -20;
          program_counter = 4'b0;
          for (i=0;i<8;i = i + 1) instruction_cache[i] = 32'h0;
          instruction_cache[0] = instruction[31:0];
          instruction_cache[1] = instruction[63:32];
          instruction_cache[2] = instruction[95:64];
          instruction_cache[3] = instruction[127:96];
          instruction_cache[4] = instruction[159:128];
          instruction_cache[5] = instruction[191:160];
          instruction_cache[6] = instruction[223:192];
          instruction_cache[7] = instruction[255:224];
// for(i=0; i<8; i=i+1)
//   begin
//   $display("INSTRUCTION FROM PIPELINE IS ", instruction_cache[i]);
//   end
```

```
        end
        else begin

            case(stage)
                STAGE_1: begin
                    if(program_counter == 9) begin
                        stage = STAGE_1;
                        pipeline_1 = 32'h0;
                    end

                else begin

//                      pipeline_1 = instruction[31: 0];
                    pipeline_1 = instruction_cache[program_counter];
                    current_instruction = pipeline_1;
                    program_counter = program_counter + 1;
                    enable_write = 1'b0;
//                      pipeline_1 = instruction;
                    data_cache_address = 5'h0;
                    $display("instruction is ",pipeline_1);
  //                  $display("first cycle");
                    stage = STAGE_2;
                end


                end
                STAGE_2: begin
                // waiting for output from decoder
//                      $display("second cycle");

                stage = STAGE_3;
//                      $display("A and B are     ",A, B);

                end
                STAGE_3: begin
                A = read_value_1_wire;
//                      $display(" A is ", A);
//                      B = read_value_2_wire;
                case(risl_type_wire)
                    R_type:
                    begin
                    B = read_value_2_wire;
                    destination_register = rd_wire;
                    end
                    I_type: begin

                            destination_register = rd_wire;
                            if (~immediate_wire[11]) B = {20'b0, immediate_wire};
                            else B = {20'hfffff, immediate_wire};

                    end
                    S_type:begin

                        destination_register = rd_wire;
                        temp = read_value_2_wire;
```

```
                    if (~immediate_wire[11]) B = {20'b0, immediate_wire};
                        else B = {20'hfffff, immediate_wire};
//                      $display(" A and B for store instruction are ", A, B);
//                      $display("rv1 is ", read_value_1_wire);


                    data_cache_address = B + temp;
//                      $display("dca is ..........", data_cache_address);

                end
                L_type:
                begin
                    temp = read_value_1_wire;
                    if (~immediate_wire[11]) B = {20'b0, immediate_wire};
                        else B = {20'hfffff, immediate_wire};


                end


                default:
                begin
                B = 32'b0;
                destination_register = rd_wire;

                end



            endcase
//              $display("A and B are     ",A, B);
                stage = STAGE_4;
//                  enable_write = 1'b1;
//                  write_value = LH_wire;
////                  $display("lh wire in third cycle is ", LH_wire);
////                  $display("write value is ",write_value);
//                  stage = STAGE_1;


            end
            STAGE_4: begin
//              $display("LH wire in 4th cycle is ",LH_wire);
//                      enable_write = 1'b1;

//                  write_value = LH_wire;
//                  $display("lh wire in third cycle is ", LH_wire);
//                  $display("write value is ",write_value);


                case(load_store)
                    2'b10:begin
                    // load instruction
                    enable_write = 1'b1;
// lh wire from alu is the address from where we
//have to take the value from data cache and place it in destination register
```

```
                        destination_register = rd_wire;
                        write_value = data_cache[temp+B];
                        $display("write value is ", write_value);


                        end
                        2'b01:
                        begin
                        //store instruction
//                          enable_write = 1'b1;
                        enable_write = 1'b0;
//                          write_value
// address is in lh wire, we have to write the
// read data from register file to data cache[lh wire]
                        data_cache[data_cache_address] = A;

                        end
                        default:
                        begin
                            enable_write = 1'b1;
                            write_value = LH_wire;


                        end


                    endcase
                    stage = STAGE_1;
                  for (i=0;i<8;i=i+1) $display("%d value is %b",i, data_cache[i]);
                end

                default: begin
                end
            endcase
        end
    end

endmodule
```

6.2 Code for Testbench

```
module testbench;
reg [255:0] instruction;
reg clk, rst;
reg [31:0] instruction_cache [7:0];
// the above is the instruction cache which holds the instruction




wire [31:0] rv1, rv2, imm, ans_from_value,current_instruction;
wire [3:0] risl_type;

pipeline DUT(
```

```
  instruction , clk , rst , rv1 , rv2 , imm,
  ans_from_value , current_instruction , risl_type
);
// the instruction passed to the pipeline
// module is a single string of 8*32 = 256 bits


initial begin
clk = 1'b0;
rst = 1'b1;
instruction_cache[0] = {12'd0,5'd29,3'b000,5'd29,7'b0010011};
instruction_cache[1] = {12'd1,5'd30,3'b000,5'd30,7'b0010011};
instruction_cache[2] = {12'd0,5'd29, 3'b010, 5'd28, 7'b0000011};
instruction_cache[3] = {12'd0,5'd30, 3'b010, 5'd27, 7'b0000011};
instruction_cache[4] = {7'b0, 5'd29, 5'd28, 3'b0, 5'd10, 7'b0110011};
instruction_cache[5] = {7'b0,5'd29,5'd10, 3'b010, 5'd0, 7'b0100011};
instruction_cache[6] = 32'h0;
instruction_cache[7] = 32'h0;
instruction = {instruction_cache[7],instruction_cache[6],instruction_cache[5],instruction_c
// instruction = {
//      instruction_cache[7],instruction_cache[6],
//      instruction_cache[5],instruction_cache[4],
//      instruction_cache[3],instruction_cache[2],
//      instruction_cache[1],instruction_cache[0]
// };

#20
rst = 1'b0;
//    $display("instruction from testbench is ", instruction);




#320 $finish;

end

always #5 clk = ~clk;

endmodule
```