Habib University
CS330L Computer Architecture Lab

Lab Project Report

Instructor: Maria Samad

Muhammad Hammad Maqdoom *mm05534*
Zoha Ovais Karim zk05617
Umema Zehra

Spring 2021

Contents

1 Introduction **1**

2 Task 1 **2**
2.1 Modify single-cycle processor 2

3 Task 2 **19**
3.1 5 Stage Pipelining 19

4 Code **41**
4.1 EDA Links 41

Chapter 1

Introduction

This project required us to build a 5-stage pipelined processor capable of executing a bubble sort program.

1. We modified the single-cycle processor to be able to run the bubble sort code on it.
2. We then modified the said processor to make it a pipelined one (5 stages). We then tested and run each instruction separately to verify that the pipelined version can at least execute one instruction correctly in isolation.
3. We then introduced circuitry to detect hazards (data, control, and structural) and tried to handle them in hardware i.e. by forwarding, stalling, and flushing the pipeline.

Chapter 2

Task 1

2.1 Modify single-cycle processor

The tasks that we attempted in lab 11 came in handy for this task. We made changes and alterations to the code in line with task 1 given to us.

We modified the single-cycle processor to be able to run the bubble sort code on it. We had worked on the bubble sort task in lab 4 (task 2). For the instructions, we made use of the venus simulator and then used the same instructions in verilog for instruction memory.

```
// Code your design here

module Adder(
    input [63:0] a, b,
    output reg [63:0] out
);
always@(*)
    out = a + b;
endmodule

module registerFile(
    input [63:0] WriteData,
    input [4:0] RS1,
    input [4:0] RS2,
    input [4:0] RD,
    input RegWrite, clk, reset,
    output reg [63:0] ReadData1,
    output reg [63:0] ReadData2
);
reg [63:0] Registers [31:0];
initial
    begin
    Registers[0] = 64'd 0;
    Registers[1] = 64'd 0;
    Registers[2] = 64'd 0;
    Registers[3] = 64'd 0;
    Registers[4] = 64'd 0;
    Registers[5] = 64'd 0;
    Registers[6] = 64'd 0;
```

```
    Registers[7] = 64'd 0;
    Registers[8] = 64'd 0;
    Registers[9] = 64'd 0;
    Registers[10] = 64'd 0;
    Registers[11] = 64'd 0;
    Registers[12] = 64'd 0;
    Registers[13] = 64'd 0;
    Registers[14] = 64'd 0;
    Registers[15] = 64'd 0;
    Registers[16] = 64'd 0;
    Registers[17] = 64'd 0;
    Registers[18] = 64'd 0;
    Registers[19] = 64'd 0;
    Registers[20] = 64'd 0;
    Registers[21] = 64'd 0;
    Registers[22] = 64'd 0;
    Registers[23] = 64'd 0;
    Registers[24] = 64'd 0;
    Registers[25] = 64'd 0;
    Registers[26] = 64'd 0;
    Registers[27] = 64'd 0;
    Registers[28] = 64'd 0;
    Registers[29] = 64'd 0;
    Registers[30] = 64'd 0;
    Registers[31] = 64'd 0;
    end
  always @(posedge clk)
    if(RegWrite)
        begin
        Registers[RD] = WriteData;
        end
  always @(*)
    if(reset)
        begin
        ReadData1 = 64'b0;
        ReadData2 = 64'b0;
        end
    else
        begin
        ReadData1 = Registers[RS1];
        ReadData2 = Registers[RS2];
        end
endmodule

module Instruction_Parser(
    input [31:0] instruction,
    output [6:0] opcode, funct7,
    output [4:0] rd , rs1 , rs2,
    output [2:0] funct3

);

assign opcode = instruction[6:0];
assign rd = instruction[11:7];
assign funct3 = instruction[14:12];
assign rs1 = instruction[19:15];
assign rs2 = instruction[24:20];
assign funct7 = instruction[31:25];
```

```
88
89  endmodule
90
91  module mux2x1
92  (
93      input [63:0] a,b,
94      input s ,
95      output [63:0] data_out
96  );
97
98  assign data_out = s ? b : a;
99
100 endmodule
101
102 module data_generator
103 (
104 input [31:0] instruction,
105 output reg [63:0] imm_data
106
107 );
108
109 wire [6:0] opcode;
110 assign opcode = instruction[6:0];
111
112 always @(*)
113 begin
114     case (opcode)
115         7'b0000011: imm_data =  {{52{instruction[31]}}, instruction
    [31:20]};
116         7'b0100011: imm_data = {{52{instruction[31]}}, instruction [31:25],
     instruction [11:7]};
117         7'b1100011: imm_data = {{52{instruction[31]}}, instruction [31] ,
    instruction [7], instruction [30:25], instruction [11:8]};
118         7'b0010011: imm_data = {{52{instruction[31]}}, instruction[31:20]};
119         default : imm_data = 64'd0;
120     endcase
121 end
122
123 endmodule
124
125 // reg [63:0] immediate;
126 // wire [6:0] opcode;
127 // assign opcode = instruction[6:0];
128
129 // always @(instruction)
130 // begin
131 //     if(instruction[6]==0) //data transfer
132 //        if(instruction[5]==0) //load
133 //                immediate[11:0] = instruction [31:20];
134 //        else if(instruction[5]==1) //store
135 //                immediate[11:0]= {instruction[31:25],instruction[11:7]};
136 //     else if(instruction[6]==1) //conditional branches
137 //         immediate[11:0] = {instruction[31],instruction[7],instruction
    [30:25] , instruction[11:8]};
138 // end
139
140 // assign imm_data[11:0]= immediate[11:0];
141 // assign imm_data[63:12] = {52{instruction[31]}};
```

```
142
143
144 // endmodule
145
146 module selector(
147     input branch, ZERO,
148     input [63:0] a, b,
149     input [2:0] funct3,
150     output reg sel
151 );
152
153 always@(*)
154 begin
155     if (branch == 1)
156         begin
157             case(funct3)
158             3'b001: //bne
159             begin
160                 if(branch == 1 & ZERO == 0)
161                     sel = 1;
162                 else
163                     sel = 0;
164             end
165             3'b000: //beq
166             begin
167                 if(branch == 1 & ZERO == 1)
168                     sel = 1;
169                 else
170                     sel = 0;
171             end
172             3'b101: //bge
173             begin
174                 if (a >= b)
175                     sel = 1;
176                 else
177                     sel = 0;
178             end
179           endcase
180         end
181    else
182       sel = 0;
183 end
184 endmodule
185
186
187
188
189 module Program_Counter
190 (
191     input clk, reset,
192     input [63:0] PC_In,
193     output reg [63:0] PC_Out
194 );
195
196 reg reset_force; // variable to force 0th value after reset
197
198 initial
199 PC_Out <= 64'd0;
```

```
200
201
202 always @(posedge clk or posedge reset) begin
203     if (reset || reset_force) begin
204         PC_Out = 64'd0;
205         reset_force <= 0;
206         end
207
208     // else if (!PCWrite) begin
209     //     PC_Out = PC_Out;
210     // end
211     else
212     PC_Out = PC_In;
213
214 end
215
216 always @(negedge reset) reset_force <= 1;
217
218 endmodule // Program_Counter
219
220 module Data_Memory(
221     input [63:0] mem_addr,
222     input [63:0] write_data,
223     input clk, mem_write, mem_read,
224   output reg [63:0] read_data,
225   output [63:0] element1,
226   output [63:0] element2 ,
227   output [63:0] element3,
228   output [63:0] element4,
229   output [63:0] element5,
230   output [63:0] element6,
231   output [63:0] element7,
232   output [63:0] element8);
233 reg [0:7] data_mem[63:0];
234 initial
235 begin
236     data_mem[0] = 64'd0;
237     data_mem[1] = 64'd0;
238     data_mem[2] = 64'd0;
239     data_mem[3] = 64'd0;
240     data_mem[4] = 64'd0;
241     data_mem[5] = 64'd0;
242     data_mem[6] = 64'd0;
243     data_mem[7] = 64'd0;
244     data_mem[8] = 64'd0;
245     data_mem[9] = 64'd0;
246     data_mem[10] = 64'd0;
247     data_mem[11] = 64'd0;
248     data_mem[12] = 64'd0;
249     data_mem[13] = 64'd0;
250     data_mem[14] = 64'd0;
251     data_mem[15] = 64'd0;
252     data_mem[16] = 64'd0;
253     data_mem[17] = 64'd0;
254     data_mem[18] = 64'd0;
255     data_mem[19] = 64'd0;
256     data_mem[20] = 64'd0;
257     data_mem[21] = 64'd0;
```

```
    data_mem[22] = 64'd0;
    data_mem[23] = 64'd0;
    data_mem[24] = 64'd0;
    data_mem[25] = 64'd0;
    data_mem[26] = 64'd0;
    data_mem[27] = 64'd0;
    data_mem[28] = 64'd0;
    data_mem[29] = 64'd0;
    data_mem[30] = 64'd0;
    data_mem[31] = 64'd0;
    data_mem[32] = 64'd0;
    data_mem[33] = 64'd0;
    data_mem[34] = 64'd0;
    data_mem[35] = 64'd0;
    data_mem[36] = 64'd0;
    data_mem[37] = 64'd0;
    data_mem[38] = 64'd0;
    data_mem[39] = 64'd0;
    data_mem[40] = 64'd0;
    data_mem[41] = 64'd0;
    data_mem[42] = 64'd0;
    data_mem[43] = 64'd0;
    data_mem[44] = 64'd0;
    data_mem[45] = 64'd0;
    data_mem[46] = 64'd0;
    data_mem[47] = 64'd0;
    data_mem[48] = 64'd0;
    data_mem[49] = 64'd0;
    data_mem[50] = 64'd0;
    data_mem[51] = 64'd0;
    data_mem[52] = 64'd0;
    data_mem[53] = 64'd0;
    data_mem[54] = 64'd0;
    data_mem[55] = 64'd0;
    data_mem[56] = 64'd0;
    data_mem[57] = 64'd0;
    data_mem[58] = 64'd0;
    data_mem[59] = 64'd0;
    data_mem[60] = 64'd0;
    data_mem[61] = 64'd0;
    data_mem[62] = 64'd0;
    data_mem[63] = 64'd0;
end
always @(negedge clk)
begin
    if (mem_write)
        begin
            data_mem[mem_addr] = write_data[7:0];
            data_mem[mem_addr+1] = write_data[15:8];
            data_mem[mem_addr+2] = write_data[23:16];
            data_mem[mem_addr+3] = write_data[31:24];
            data_mem[mem_addr+4] = write_data[39:32];
            data_mem[mem_addr+5] = write_data[47:40];
            data_mem[mem_addr+6] = write_data[55:48];
            data_mem[mem_addr+7] = write_data[63:56];
        end
end
  always @(*)
```

```
316     begin
317         if(mem_read)
318         begin
319             read_data = {data_mem[mem_addr+7],data_mem[mem_addr+6],
    data_mem[mem_addr+5], data_mem[mem_addr+4], data_mem[mem_addr+3],
    data_mem[mem_addr+2], data_mem[mem_addr+1], data_mem[mem_addr]};
320         end
321     end
322   assign element1= {data_mem[7],data_mem[6],data_mem[5], data_mem[4],
    data_mem[3], data_mem[2], data_mem[1], data_mem[0]};
323   assign element2= {data_mem[15],data_mem[14],data_mem[13], data_mem[12],
    data_mem[11], data_mem[10], data_mem[9], data_mem[8]};
324   assign element3= {data_mem[23],data_mem[22],data_mem[21], data_mem[20],
    data_mem[19], data_mem[18], data_mem[17], data_mem[16]};
325   assign element4= {data_mem[31],data_mem[30],data_mem[29], data_mem[28],
    data_mem[27], data_mem[26], data_mem[25], data_mem[24]};
326   assign element5= {data_mem[39],data_mem[38],data_mem[37],data_mem[36],
    data_mem[35], data_mem[34], data_mem[33], data_mem[32]};
327   assign element6= {data_mem[47], data_mem[46],data_mem[45],data_mem[44],
    data_mem[43], data_mem[42], data_mem[41], data_mem[40]};
328   assign element7= {data_mem[55], data_mem[54],data_mem[53],data_mem[52],
    data_mem[51], data_mem[50], data_mem[49], data_mem[48]};
329   assign element8= {data_mem[63], data_mem[62],data_mem[61],data_mem[60],
    data_mem[59], data_mem[58], data_mem[57], data_mem[56]};
330 endmodule
331
332 module Instruction_Memory(
333     input [63:0] Inst_Address,
334     output reg [31:0] Instruction
335 );
336 reg [7:0] inst_memory [131:0];
337
338 initial
339 begin
340     inst_memory[0] = 8'b10010011;
341     inst_memory[1] = 8'b00000010;
342     inst_memory[2] = 8'b00110000;
343     inst_memory[3] = 8'b00000000;
344
345     inst_memory[4] = 8'b00100011;
346     inst_memory[5] = 8'b00110010;
347     inst_memory[6] = 8'b01010000;
348     inst_memory[7] = 8'b00000000;
349
350     inst_memory[8] = 8'b10010011;
351     inst_memory[9] = 8'b00000010;
352     inst_memory[10] = 8'b00100000;
353     inst_memory[11] = 8'b00000000;
354
355     inst_memory[12] = 8'b00100011;
356     inst_memory[13] = 8'b00110110;
357     inst_memory[14] = 8'b01010000;
358     inst_memory[15] = 8'b00000000;
359
360     inst_memory[16] = 8'b10010011;
361     inst_memory[17] = 8'b00000010;
362     inst_memory[18] = 8'b10100000;
363     inst_memory[19] = 8'b00000000;
```

```
364
365     inst_memory[20] = 8'b00100011;
366     inst_memory[21] = 8'b00111010;
367     inst_memory[22] = 8'b01010000;
368     inst_memory[23] = 8'b00000000;
369
370     inst_memory[24] = 8'b00010011;
371     inst_memory[25] = 8'b00000101;
372     inst_memory[26] = 8'b01000000;
373     inst_memory[27] = 8'b00000000;
374
375     inst_memory[28] = 8'b10010011;
376     inst_memory[29] = 8'b00000101;
377     inst_memory[30] = 8'b00110000;
378     inst_memory[31] = 8'b00000000;
379
380     inst_memory[32] = 8'b01100011;
381     inst_memory[33] = 8'b00010110;
382     inst_memory[34] = 8'b00000101;
383     inst_memory[35] = 8'b00000000;
384
385     //bne 101
386     inst_memory[36] = 8'b01100011;
387     inst_memory[37] = 8'b10010100;
388     inst_memory[38] = 8'b00000101;
389     inst_memory[39] = 8'b00000000;
390
391     //beq 011
392     inst_memory[40] = 8'b01100011;
393     inst_memory[41] = 8'b00001100;
394     inst_memory[42] = 8'b00000000;
395     inst_memory[43] = 8'b00000100;
396
397     inst_memory[44] = 8'b00010011;
398     inst_memory[45] = 8'b00001001;
399     inst_memory[46] = 8'b00000000;
400     inst_memory[47] = 8'b00000000;
401
402     inst_memory[48] = 8'b01100011;
403     inst_memory[49] = 8'b00000110;
404     inst_memory[50] = 8'b10111001;
405     inst_memory[51] = 8'b00000100;
406
407     inst_memory[52] = 8'b10110011;
408     inst_memory[53] = 8'b00001001;
409     inst_memory[54] = 8'b00100000;
410     inst_memory[55] = 8'b00000001;
411
412     inst_memory[56] = 8'b01100011;
413     inst_memory[57] = 8'b10001110;
414     inst_memory[58] = 8'b10111001;
415     inst_memory[59] = 8'b00000010;
416
417     inst_memory[60] = 8'b10010011;
418     inst_memory[61] = 8'b00010010;
419     inst_memory[62] = 8'b00111001;
420     inst_memory[63] = 8'b00000000;
421
```

```
422     inst_memory[64] = 8'b00010011;
423     inst_memory[65] = 8'b10010011;
424     inst_memory[66] = 8'b00111001;
425     inst_memory[67] = 8'b00000000;
426 
427     inst_memory[68] = 8'b10110011;
428     inst_memory[69] = 8'b10000010;
429     inst_memory[70] = 8'b10100010;
430     inst_memory[71] = 8'b00000000;
431 
432     inst_memory[72] = 8'b00110011;
433     inst_memory[73] = 8'b00000011;
434     inst_memory[74] = 8'b10100011;
435     inst_memory[75] = 8'b00000000;
436 
437     inst_memory[76] = 8'b00000011;
438     inst_memory[77] = 8'b10111110;
439     inst_memory[78] = 8'b00000010;
440     inst_memory[79] = 8'b00000000;
441 
442     inst_memory[80] = 8'b10000011;
443     inst_memory[81] = 8'b00111110;
444     inst_memory[82] = 8'b00000011;
445     inst_memory[83] = 8'b00000000;
446     //bge 111
447     inst_memory[84] = 8'b01100011;
448     inst_memory[85] = 8'b01011100;
449     inst_memory[86] = 8'b11011110;
450     inst_memory[87] = 8'b00000001;
451 
452     inst_memory[88] = 8'b00110011;
453     inst_memory[89] = 8'b00001111;
454     inst_memory[90] = 8'b11000000;
455     inst_memory[91] = 8'b00000001;
456 
457     inst_memory[92] = 8'b00110011;
458     inst_memory[93] = 8'b00001110;
459     inst_memory[94] = 8'b11010000;
460     inst_memory[95] = 8'b00000001;
461 
462     inst_memory[96] = 8'b10110011;
463     inst_memory[97] = 8'b00001110;
464     inst_memory[98] = 8'b11100000;
465     inst_memory[99] = 8'b00000001;
466 
467     inst_memory[100] =8'b00100011;
468     inst_memory[101] =8'b10110000;
469     inst_memory[102] =8'b11000010;
470     inst_memory[103] =8'b00000001;
471 
472     inst_memory[104] =8'b00100011;
473     inst_memory[105] =8'b00110000;
474     inst_memory[106] =8'b11010011;
475     inst_memory[107] =8'b00000001;
476 
477     inst_memory[108] =8'b10010011;
478     inst_memory[109] =8'b10001001;
479     inst_memory[110] =8'b00011001;
```

```
    inst_memory[111] =8'b00000000;

    inst_memory[112] =8'b11100011;
    inst_memory[113] =8'b00000100;
    inst_memory[114] =8'b00000000;
    inst_memory[115] =8'b11111100;

    inst_memory[116] =8'b00010011;
    inst_memory[117] =8'b00001001;
    inst_memory[118] =8'b00011001;
    inst_memory[119] =8'b00000000;

    inst_memory[120] =8'b11100011;
    inst_memory[121] =8'b00001100;
    inst_memory[122] =8'b00000000;
    inst_memory[123] =8'b11111010;

    inst_memory[124] =8'b01100011;
    inst_memory[125] =8'b00000010;
    inst_memory[126] =8'b00000000;
    inst_memory[127] =8'b00000000;

    inst_memory[128] =8'b00010011;
    inst_memory[129] =8'b00000000;
    inst_memory[130] =8'b00000000;
    inst_memory[131] =8'b00000000;

end
always@(Inst_Address)
    Instruction = {inst_memory[Inst_Address+3],inst_memory[Inst_Address+2],
   inst_memory[Inst_Address+1], inst_memory[Inst_Address]};
endmodule

module Control_Unit
(
    input [6:0] Opcode,
    output reg [1:0] ALUOp,
    output reg Branch, MemRead, MemtoReg, MemWrite, ALUSrc, Regwrite
);
always @(*)
begin
    case (Opcode)
    7'b0110011: // R-type (add/sub)
        begin
            ALUSrc = 1'b0;
            MemtoReg = 1'b0;
            Regwrite = 1'b1;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            Branch = 1'b0;
            ALUOp = 2'b10;
        end
    7'b0000011: // I-type (ld)
        begin
            ALUSrc = 1'b1;
            MemtoReg = 1'b1;
            Regwrite = 1'b1;
            MemRead = 1'b1;
```

```
            MemWrite = 1'b0;
            Branch = 1'b0;
            ALUOp = 2'b00;
        end
    7'b0100011: // S-type(sd)
        begin
            ALUSrc = 1'b1;
            MemtoReg = 1'bx;
            Regwrite = 1'b0;
            MemRead = 1'b0;
            MemWrite = 1'b1;
            Branch = 1'b0;
            ALUOp = 2'b00;
        end
    7'b0010011: // I-type (addi)
        begin
            ALUSrc = 1'b1;
            MemtoReg = 1'b0;
            Regwrite = 1'b1;
            MemRead = 1'b1;
            MemWrite = 1'b0;
            Branch = 1'b0;
            ALUOp = 2'b00;
        end
    7'b1100011: // SB-type (beq/bne/bge)
        begin
            ALUSrc = 1'b0;
            MemtoReg = 1'bx;
            Regwrite = 1'b0;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            Branch = 1'b1;
            ALUOp = 2'b01;
        end
    default: begin
            ALUSrc = 1'b0;
            MemtoReg = 1'b0;
            Regwrite = 1'b0;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            Branch = 1'b0;
            ALUOp = 2'b00;
    end
    endcase
end
endmodule

module ALU_Control
(
    input [1:0] ALUOp,
    input [3:0] Funct,
    output reg [3:0] Operation
);
always@(*)
begin
    case(ALUOp)
        2'b00: //for both addi and slli
            begin
```

```
                case(Funct[2:0])
                3'b000: //addi
                begin
                    Operation = 4'b0010;
                end
                3'b001: //slli
                begin
                    Operation = 4'b1000;
                end
              endcase
            end
        2'b01:
            begin
            Operation = 4'b0110;
            end
        2'b10:
            begin
            case(Funct)
            4'b0000:
                begin
                Operation = 4'b0010;
                end
            4'b1000:
                begin
                Operation = 4'b0110;
                end
            4'b0111:
                begin
                Operation = 4'b0000;
                end
            4'b0110:
                begin
                Operation = 4'b0001;
                end
            endcase
            end
    endcase
end
endmodule

module alu_64(
  input [63:0] a,
    input [63:0] b,
    input [3:0] ALUOp,
  output reg [63:0] Result,
    output reg ZERO
);
always @ (*)
begin
    case(ALUOp)
    4'b0000 :
        begin
        Result = a&b;
        end
    4'b0001 :
        begin
        Result = a|b;
        end
```

```
    4'b0010 :
        begin
        Result = a+b;
        end
    4'b0110:
        begin
        Result = a-b;
        end
    4'b1100:
        begin
        Result = ~(a|b);
        end
    4'b1000:
        begin
            Result = a << b;
        end
    default : Result = 0;
endcase

if (Result == 64'b0)
    ZERO = 1'b1;
else
    ZERO = 1'b0;

end
endmodule



module RISC_V_Processor(
    input clk, reset
);

//Mux output
wire [63:0] PC_In_from_mux;
//Program counter output
wire [63:0] PC_Out;
//Adders outputs
wire [63:0] a1_out;
wire [63:0] a2_out;
//Input to Adder a1
wire [63:0] b_in = 64'd4;
//Output from IM
wire [31:0] Instruction;
//Output from IP
wire [4:0] rd;
wire [4:0] rs1;
wire [4:0] rs2;
wire [6:0] opcode;
wire [6:0] funct7;
wire [2:0] funct3;
//Outputs from RegisterFile
wire [63:0] ReadData1;
wire [63:0] ReadData2;
//Outputs from Control Unit
wire [1:0] ALUOp;
wire Branch, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite;
//Outputs from ALU Control
```

```
711 wire [3:0] Operation;
712 //Funct input to ALU_Control
713 wire [3:0] Funct;
714 assign Funct = {Instruction[30], Instruction[14:12]};
715 //Output from ALU
716 wire [63:0] Result_from_alu;
717 wire zero_output;
718 //Output from Data generator
719 wire [63:0] imm_data;
720 //Output from mux2
721 wire [63:0] out_from_mux2;
722 //sel for mux1
723 wire sel;
724 //Output from Data memory
725 wire [63:0] out_from_DM;
726 //Output from mux3
727 wire [63:0] out_from_mux3;
728 //Input to Adder a2
729 wire [63:0] b_adder2;
730 assign b_adder2 = imm_data << 1;
731
732 Program_Counter pc (.clk(clk), .reset(reset), .PC_In(PC_In_from_mux), .
        PC_Out(PC_Out));
733
734 Adder a1 (.a(PC_Out), .b(b_in), .out(a1_out));
735
736 Adder a2(.a(PC_Out), .b(b_adder2), .out(a2_out));
737
738 mux2x1 mux1(.a(a1_out), .b(a2_out), .s(sel), .data_out(PC_In_from_mux));
739
740 Instruction_Memory im(.Inst_Address(PC_Out), .Instruction(Instruction));
741
742 Instruction_Parser ip(.instruction(Instruction), .rd(rd), .rs1(rs1), .rs2(
        rs2), .funct3(funct3), .funct7(funct7), .opcode(opcode));
743
744 registerFile rf(.WriteData(out_from_mux3), .RS1(rs1), .RS2(rs2), .RD(rd), .
        clk(clk), .reset(reset), .RegWrite(RegWrite), .ReadData1(ReadData1), .
        ReadData2(ReadData2));
745
746 Control_Unit cu(
747     .Opcode(opcode),
748     .ALUOp(ALUOp),
749     .Branch(Branch),
750     .MemRead(MemRead),
751     .MemtoReg(MemtoReg),
752     .MemWrite(MemWrite),
753     .ALUSrc(ALUSrc),
754     .Regwrite(RegWrite));
755
756 ALU_Control aluc(
757     .ALUOp(ALUOp),
758     .Funct(Funct),
759     .Operation(Operation));
760
761 data_generator dg(
762     .instruction(Instruction),
763     .imm_data(imm_data));
764
```

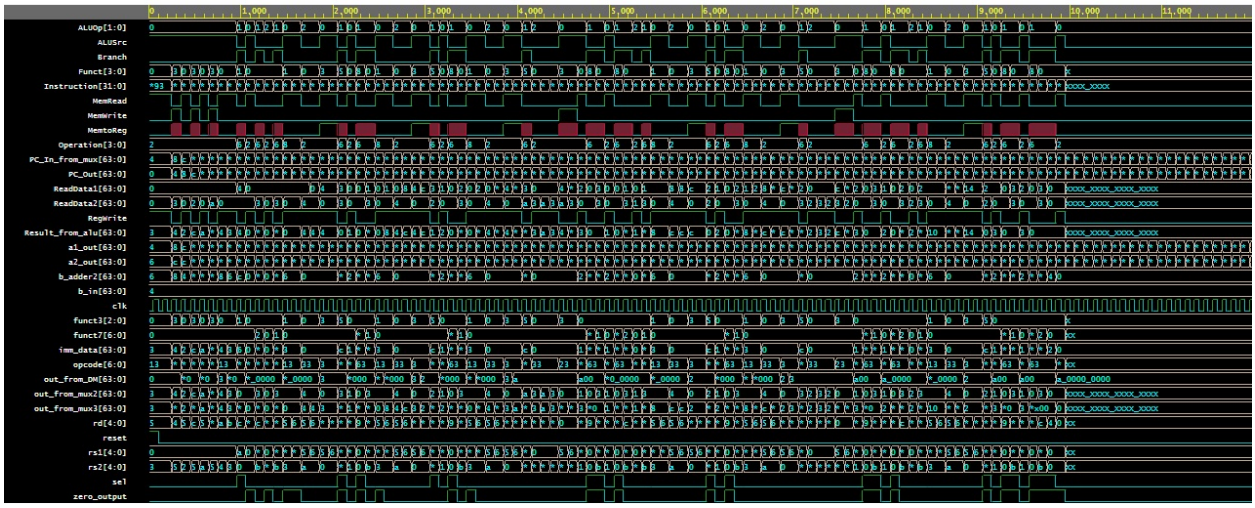
```
765 mux2x1 mux2(
766     .a(ReadData2),
767     .b(imm_data),
768     .s(ALUSrc),
769     .data_out(out_from_mux2));
770 
771 alu_64 alu(
772     .a(ReadData1),
773     .b(out_from_mux2),
774     .ALUOp(Operation),
775     .Result(Result_from_alu),
776     .ZERO(zero_output));
777 
778 selector s(
779     .branch(Branch),
780     .ZERO(zero_output),
781     .a(ReadData1),
782     .b(out_from_mux2),
783     .funct3(funct3),
784     .sel(sel));
785 
786   Data_Memory dm(.mem_addr(Result_from_alu), .write_data(ReadData2), .clk(
    clk), .mem_write(MemWrite), .mem_read(MemRead), .read_data(out_from_DM),
     .element1(el1), .element2(el2), .element3(el3), .element4(el4), .
    element5(el5), .element6(el6), .element7(el7), .element8(el8));
787 
788 mux2x1 mux3(
789     .b(out_from_DM),
790     .a(Result_from_alu),
791     .s(MemtoReg),
792     .data_out(out_from_mux3));
793 
794 always @(posedge clk)
795     begin
796         $monitor(
797         "PC_In = ", PC_In_from_mux,
798         ", PC_Out = ", PC_Out,
799         ", Instruction = %b", Instruction,
800         ", Opcode = %b", opcode,
801         ", Funct3 = %b", funct3,
802         ", Zero = %b", zero_output,
803         ", Branch = %d", Branch,
804         ", sel = %d", sel,
805         ", rs1 = %d", rs1,
806         ", rs2 = %d", rs2,
807         ", rd = %d", rd,
808         ", funct7 = %b", funct7,
809         ", ALUOp = %b", ALUOp,
810         ", imm_data = %d", imm_data,
811         ", Operation = %b", Operation
812         );
813     end
814 endmodule
```

Test Bench:

```
1 module tb();
2 
```

```
reg clk, reset;

RISC_V_Processor processor(.clk(clk), .reset(reset));

initial
begin
  $dumpfile("dump.vcd");
  $dumpvars();
  clk = 1'd0;
  reset = 1'd1;
  #10
  reset = 1'd0;
  #1200
  $finish;
end

always #5 clk=~clk;

endmodule
```

Result from Test Bench:
Wave Diagram



Chapter 3

Task 2

3.1 5 Stage Pipelining

We now modified the said processor to make it a pipelined one (5 stages). We then tested and ran each instruction separately to verify that the pipelined version can at least execute one instruction correctly in isolation.

```
1
2  module Adder(
3      input [63:0] a, b,
4      output reg [63:0] out
5  );
6  always@(*)
7      out = a + b;
8  endmodule
9
10
11 module alu_64(
12   input [63:0] a,
13     input [63:0] b,
14     input [3:0] ALUOp,
15     output reg ZERO,
16   output reg [63:0] Result
17 );
18 always @ (*)
19 begin
20     case(ALUOp)
21     4'b0000 :
22         begin
23         Result = a&b;
24         end
25     4'b0001 :
26         begin
27         Result = a|b;
28         end
29     4'b0010 :
30         begin
31         Result = a+b;
32         end
33     4'b0110:
```

```
        begin
        Result = a-b;
        end
    4'b1100:
        begin
        Result = ~(a|b);
        end
    4'b1000:
        begin
            Result = a << b;
        end
    default : Result = 0;
endcase

if (Result == 64'd0)
    ZERO = 1'b0;
else
    ZERO = 1'b0;

end
endmodule


module ALU_Control
(
    input [1:0] ALUOp,
    input [3:0] Funct,
    output reg [3:0] Operation
);
always@(*)
begin
    case(ALUOp)
        2'b00: //for both addi and slli
            begin
                case(Funct[2:0])
                3'b000: //addi
                begin
                    Operation = 4'b0010;
                end
                3'b001: //slli
                begin
                    Operation = 4'b1000;
                end
              endcase
            end
        2'b01:
            begin
            Operation = 4'b0110;
            end
        2'b10:
            begin
            case(Funct)
            4'b0000:
                begin
                Operation = 4'b0010;
                end
            4'b1000:
                begin
```

```
                Operation = 4'b0110;
                end
            4'b0111:
                begin
                Operation = 4'b0000;
                end
            4'b0110:
                begin
                Operation = 4'b0001;
                end
            endcase
            end
    endcase
end
endmodule




module Control_Unit
(
    input [6:0] Opcode,
    output reg [1:0] ALUOp,
    output reg Branch, MemRead, MemtoReg, MemWrite, ALUSrc, Regwrite
);
always @(*)
begin
    case (Opcode)
    7'b0110011: // R-type (add/sub)
        begin
            ALUSrc = 1'b0;
            MemtoReg = 1'b0;
            Regwrite = 1'b1;
            MemRead = 1'b0;
            MemWrite = 1'b0;
            Branch = 1'b0;
            ALUOp = 2'b10;
        end
    7'b0000011: // I-type (ld)
        begin
            ALUSrc = 1'b1;
            MemtoReg = 1'b1;
            Regwrite = 1'b1;
            MemRead = 1'b1;
            MemWrite = 1'b0;
            Branch = 1'b0;
            ALUOp = 2'b00;
        end
    7'b0100011: // S-type(sd)
        begin
            ALUSrc = 1'b1;
            MemtoReg = 1'bx;
            Regwrite = 1'b0;
            MemRead = 1'b0;
            MemWrite = 1'b1;
            Branch = 1'b0;
            ALUOp = 2'b00;
        end
```

```
150     7'b0010011: // I-type (addi)
151         begin
152             ALUSrc = 1'b1;
153             MemtoReg = 1'b0;
154             Regwrite = 1'b1;
155             MemRead = 1'b1;
156             MemWrite = 1'b0;
157             Branch = 1'b0;
158             ALUOp = 2'b00;
159         end
160     7'b1100011: // SB-type (beq/bne/bge)
161         begin
162             ALUSrc = 1'b0;
163             MemtoReg = 1'bx;
164             Regwrite = 1'b0;
165             MemRead = 1'b0;
166             MemWrite = 1'b0;
167             Branch = 1'b1;
168             ALUOp = 2'b01;
169         end
170     default: begin
171             ALUSrc = 1'b0;
172             MemtoReg = 1'b0;
173             Regwrite = 1'b0;
174             MemRead = 1'b0;
175             MemWrite = 1'b0;
176             Branch = 1'b0;
177             ALUOp = 2'b00;
178     end
179     endcase
180 end
181 endmodule
182 
183 module Forwarding_Unit
184 (
185     input EXMEM_ReadData2, MEMWB_read_data,
186   input rs1, rs2,
187     input EXMEM_Regwrite,
188     input MEMWB_RegWrite,
189 
190     output reg [1:0] fwd_A,
191     output reg [1:0] fwd_B
192 );
193 
194 
195 
196 always @(*) begin
197 
198     if (EXMEM_ReadData2 == rs1 && EXMEM_Regwrite && EXMEM_ReadData2 != 0)
199         begin
200             fwd_A = 2'b10;
201         end
202 
203     else if (((MEMWB_read_data == rs1) && MEMWB_RegWrite && (
    MEMWB_read_data != 0))
204             &&
205             !(EXMEM_Regwrite && (EXMEM_ReadData2 != 0) && (EXMEM_ReadData2
    == rs1)))
```

```
206         begin
207             fwd_A = 2'b01;
208         end
209 
210     else
211         begin
212             fwd_A = 2'b00;
213         end
214 
215 
216   if ((EXMEM_ReadData2 == rs2) && (EXMEM_Regwrite) && (EXMEM_ReadData2 !=
      0))
217         begin
218             fwd_B = 2'b10;
219         end
220 
221     else if (((MEMWB_read_data == rs2) && (MEMWB_RegWrite == 1) && (
    MEMWB_read_data != 0))
222             &&
223             !(EXMEM_Regwrite && (EXMEM_ReadData2 != 0) && (EXMEM_ReadData2
    == rs2)
224             ))
225         begin
226             fwd_B = 2'b01;
227         end
228 
229     else
230         begin
231             fwd_B = 2'b00;
232         end
233 end
234 
235 endmodule // Forwarding Unit
236 
237 module MUX_Triple
238 (
239     input [63:0] a, b, c,
240     input [1:0] sel,
241     output reg [63:0] Res
242 );
243 
244 always@(*)
245 begin
246     case (sel)
247     2'b00: Res = a;
248     2'b01: Res = b;
249     2'b10: Res = c;
250         default: Res = 2'bX;
251     endcase
252 
253 end
254 
255 
256 endmodule // Triple MUX
257 
258 
259 
260 module data_generator
```

```
(
input [31:0] instruction,
output reg [63:0] imm_data

);

wire [6:0] opcode;
assign opcode = instruction[6:0];

always @(*)
begin
    case (opcode)
        7'b0000011: imm_data =  {{52{instruction[31]}}, instruction [31:20]};
        7'b0100011: imm_data = {{52{instruction[31]}}, instruction [31:25], instruction [11:7]};
        7'b1100011: imm_data = {{52{instruction[31]}}, instruction [31] , instruction [7], instruction [30:25], instruction [11:8]};
        7'b0010011: imm_data = {{52{instruction[31]}}, instruction[31:20]};
        default : imm_data = 64'd0;
    endcase
end

endmodule

module Data_Memory(
    input [63:0] mem_addr,
    input [63:0] write_data,
    input clk, mem_write, mem_read,
    output reg [63:0] read_data
);
reg [0:7] data_mem[63:0];
initial
begin
    data_mem[0] = 64'd0;
    data_mem[1] = 64'd0;
    data_mem[2] = 64'd0;
    data_mem[3] = 64'd0;
    data_mem[4] = 64'd0;
    data_mem[5] = 64'd0;
    data_mem[6] = 64'd0;
    data_mem[7] = 64'd0;
  data_mem[8] = 64'h8;
    data_mem[9] = 64'd0;
  data_mem[10] = 64'h10;
    data_mem[11] = 64'd0;
    data_mem[12] = 64'd0;
    data_mem[13] = 64'd0;
    data_mem[14] = 64'd0;
    data_mem[15] = 64'd0;
    data_mem[16] = 64'd0;
    data_mem[17] = 64'd0;
    data_mem[18] = 64'd0;
    data_mem[19] = 64'd0;
    data_mem[20] = 64'd0;
    data_mem[21] = 64'd0;
    data_mem[22] = 64'd0;
    data_mem[23] = 64'd0;
```

```
316     data_mem[24] = 64'd0;
317     data_mem[25] = 64'd0;
318     data_mem[26] = 64'd0;
319     data_mem[27] = 64'd0;
320     data_mem[28] = 64'd0;
321     data_mem[29] = 64'd0;
322     data_mem[30] = 64'd0;
323     data_mem[31] = 64'd0;
324     data_mem[32] = 64'd0;
325     data_mem[33] = 64'd0;
326     data_mem[34] = 64'd0;
327     data_mem[35] = 64'd0;
328     data_mem[36] = 64'd0;
329     data_mem[37] = 64'd0;
330     data_mem[38] = 64'd0;
331     data_mem[39] = 64'd0;
332     data_mem[40] = 64'd8;
333     data_mem[41] = 64'd0;
334     data_mem[42] = 64'd0;
335     data_mem[43] = 64'd0;
336     data_mem[44] = 64'd0;
337     data_mem[45] = 64'd0;
338     data_mem[46] = 64'd0;
339     data_mem[47] = 64'd0;
340     data_mem[48] = 64'd0;
341     data_mem[49] = 64'd0;
342     data_mem[50] = 64'd0;
343     data_mem[51] = 64'd0;
344     data_mem[52] = 64'd0;
345     data_mem[53] = 64'd0;
346     data_mem[54] = 64'd0;
347     data_mem[55] = 64'd0;
348     data_mem[56] = 64'd0;
349     data_mem[57] = 64'd0;
350     data_mem[58] = 64'd0;
351     data_mem[59] = 64'd0;
352     data_mem[60] = 64'd0;
353     data_mem[61] = 64'd0;
354     data_mem[62] = 64'd0;
355     data_mem[63] = 64'd0;
356 end
357 always @(negedge clk)
358 begin
359     if (mem_write)
360         begin
361             data_mem[mem_addr] = write_data[7:0];
362             data_mem[mem_addr+1] = write_data[15:8];
363             data_mem[mem_addr+2] = write_data[23:16];
364             data_mem[mem_addr+3] = write_data[31:24];
365             data_mem[mem_addr+4] = write_data[39:32];
366             data_mem[mem_addr+5] = write_data[47:40];
367             data_mem[mem_addr+6] = write_data[55:48];
368             data_mem[mem_addr+7] = write_data[63:56];
369         end
370 end
371   always @(*)
372     begin
373         if(mem_read)
```

```
374         begin
375             read_data = {data_mem[mem_addr+7],data_mem[mem_addr+6],
    data_mem[mem_addr+5], data_mem[mem_addr+4], data_mem[mem_addr+3],
    data_mem[mem_addr+2], data_mem[mem_addr+1], data_mem[mem_addr]};
376         end
377     end
378 endmodule
379
380
381
382 module EX_MEM(
383     input clk, reset, ZERO,
384     input [63:0] out, Result, IDEX_ReadData2,
385     input [4:0] IDEX_inst2,
386     input IDEX_Branch, IDEX_MemRead, IDEX_MemtoReg, IDEX_MemWrite,
    IDEX_Regwrite,
387     output reg EXMEM_ZERO,
388     output reg [4:0] EXMEM_inst2,
389     output reg [63:0] EXMEM_out, EXMEM_Result, EXMEM_ReadData2,
390     output reg EXMEM_Branch, EXMEM_MemRead, EXMEM_MemtoReg, EXMEM_MemWrite,
     EXMEM_Regwrite
391 );
392
393     always @(posedge clk or reset)
394     begin
395         if(clk)
396            begin
397                 EXMEM_out = out;
398                 EXMEM_ZERO = ZERO;
399                 EXMEM_Result = Result;
400                 EXMEM_ReadData2 = IDEX_ReadData2;
401                 EXMEM_inst2 = IDEX_inst2;
402                 EXMEM_Branch = IDEX_Branch;
403                 EXMEM_MemRead = IDEX_MemRead;
404                 EXMEM_MemtoReg= IDEX_MemtoReg;
405                 EXMEM_MemWrite= IDEX_MemWrite;
406                 EXMEM_Regwrite= IDEX_Regwrite;
407             end
408         else
409             begin
410                 EXMEM_out = 0;
411                 EXMEM_ZERO = 0;
412                 EXMEM_Result = 0;
413                 EXMEM_ReadData2 = 0;
414                 EXMEM_inst2 = 0;
415                 EXMEM_Branch = 0;
416                 EXMEM_MemRead = 0;
417                 EXMEM_MemtoReg= 0;
418                 EXMEM_MemWrite= 0;
419                 EXMEM_Regwrite=0;
420             end
421     end
422 endmodule
423
424
425
426
427
```

```
module ID_EX(
    input clk, reset,
    input [3:0] inst1,
    input [4:0] inst2,
    input [63:0] ReadData1, ReadData2, PC_Out, imm_data,
    input [1:0] ALUOp,
    input Branch, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite,
    output reg [3:0] IDEX_inst1,
    output reg [4:0] IDEX_inst2,
    output reg [63:0] IDEX_PC_Out, IDEX_ReadData1, IDEX_ReadData2,
   IDEX_imm_data,
    output reg [1:0] IDEX_ALUOp,
    output reg IDEX_Branch, IDEX_MemRead, IDEX_MemtoReg, IDEX_MemWrite,
   IDEX_ALUSrc, IDEX_Regwrite
);

always @(posedge clk or reset)
    begin
        if(clk)
            begin
                IDEX_PC_Out = PC_Out;
                IDEX_ReadData1 = ReadData1;
                IDEX_ReadData2 = ReadData2;
                IDEX_imm_data = imm_data;
                IDEX_inst1 = inst1;
                IDEX_inst2 = inst2;
                IDEX_Branch = Branch;
                IDEX_MemRead = MemRead;
                IDEX_MemWrite = MemWrite;
                IDEX_ALUSrc = ALUSrc;
                IDEX_Regwrite = RegWrite;
                IDEX_ALUOp = ALUOp;
                IDEX_MemtoReg = MemtoReg;

            end
        else
            begin
                IDEX_PC_Out = 0;
                IDEX_ReadData1 = 0;
                IDEX_ReadData2 = 0;
                IDEX_imm_data = 0;
                IDEX_inst1 = 0;
                IDEX_inst2 = 0;
                IDEX_Branch = 0;
                IDEX_MemRead = 0;
                IDEX_MemWrite = 0;
                IDEX_ALUSrc = 0;
                IDEX_Regwrite = 0;
                IDEX_ALUOp = 0;
                IDEX_MemtoReg = 0;
            end
    end

endmodule



module IF_ID(
```

```
    input clk, reset,
    input [31:0] instruction,
    input [63:0] PC_Out,
    output reg [31:0] IFID_instruction,
    output reg [63:0] IFID_PC_Out
);



always @(posedge clk or reset)
begin
    if(clk)
        begin
            IFID_instruction = instruction;
            IFID_PC_Out = PC_Out;
        end
    else
        begin
            IFID_instruction = 0;
            IFID_PC_Out = 0;
        end
end
endmodule



module Instruction_Memory(
    input [63:0] Inst_Address,
    output reg [31:0] Instruction
);
reg [7:0] inst_memory [131:0];

initial
begin

    inst_memory[0] = 8'b10000011;
    inst_memory[1] = 8'b00110100;
    inst_memory[2] = 8'b10000101;
    inst_memory[3] = 8'b00000010;


  {inst_memory[7],inst_memory[6],inst_memory[5],inst_memory[4]}=32'
    h00548513;
    // inst_memory[0] = 8'b10010011;
    // inst_memory[1] = 8'b00000010;
    // inst_memory[2] = 8'b00110000;
    // inst_memory[3] = 8'b00000000;

    // inst_memory[4] = 8'b00100011;
    // inst_memory[5] = 8'b00110010;
    // inst_memory[6] = 8'b01010000;
    // inst_memory[7] = 8'b00000000;

    // inst_memory[8] = 8'b10010011;
    // inst_memory[9] = 8'b00000010;
    // inst_memory[10] = 8'b00100000;
    // inst_memory[11] = 8'b00000000;

```

```
    // inst_memory[12] = 8'b00100011;
    // inst_memory[13] = 8'b00110110;
    // inst_memory[14] = 8'b01010000;
    // inst_memory[15] = 8'b00000000;

    // inst_memory[16] = 8'b10010011;
    // inst_memory[17] = 8'b00000010;
    // inst_memory[18] = 8'b10100000;
    // inst_memory[19] = 8'b00000000;

    // inst_memory[20] = 8'b00100011;
    // inst_memory[21] = 8'b00111010;
    // inst_memory[22] = 8'b01010000;
    // inst_memory[23] = 8'b00000000;

    // inst_memory[24] = 8'b00010011;
    // inst_memory[25] = 8'b00000101;
    // inst_memory[26] = 8'b01000000;
    // inst_memory[27] = 8'b00000000;

    // inst_memory[28] = 8'b10010011;
    // inst_memory[29] = 8'b00000101;
    // inst_memory[30] = 8'b00110000;
    // inst_memory[31] = 8'b00000000;

    // inst_memory[32] = 8'b01100011;
    // inst_memory[33] = 8'b00010110;
    // inst_memory[34] = 8'b00000101;
    // inst_memory[35] = 8'b00000000;

    // //bne 101
    // inst_memory[36] = 8'b01100011;
    // inst_memory[37] = 8'b10010100;
    // inst_memory[38] = 8'b00000101;
    // inst_memory[39] = 8'b00000000;

    // //beq 011
    // inst_memory[40] = 8'b01100011;
    // inst_memory[41] = 8'b00001100;
    // inst_memory[42] = 8'b00000000;
    // inst_memory[43] = 8'b00000100;

    // inst_memory[44] = 8'b00010011;
    // inst_memory[45] = 8'b00001001;
    // inst_memory[46] = 8'b00000000;
    // inst_memory[47] = 8'b00000000;

    // inst_memory[48] = 8'b01100011;
    // inst_memory[49] = 8'b00000110;
    // inst_memory[50] = 8'b10111001;
    // inst_memory[51] = 8'b00000100;

    // inst_memory[52] = 8'b10110011;
    // inst_memory[53] = 8'b00001001;
    // inst_memory[54] = 8'b00100000;
    // inst_memory[55] = 8'b00000001;

    // inst_memory[56] = 8'b01100011;
```

```
599     // inst_memory[57] = 8'b10001110;
600     // inst_memory[58] = 8'b10111001;
601     // inst_memory[59] = 8'b00000010;
602 
603     // inst_memory[60] = 8'b10010011;
604     // inst_memory[61] = 8'b00010010;
605     // inst_memory[62] = 8'b00111001;
606     // inst_memory[63] = 8'b00000000;
607 
608     // inst_memory[64] = 8'b00010011;
609     // inst_memory[65] = 8'b10010011;
610     // inst_memory[66] = 8'b00111001;
611     // inst_memory[67] = 8'b00000000;
612 
613     // inst_memory[68] = 8'b10110011;
614     // inst_memory[69] = 8'b10000010;
615     // inst_memory[70] = 8'b10100010;
616     // inst_memory[71] = 8'b00000000;
617 
618     // inst_memory[72] = 8'b00110011;
619     // inst_memory[73] = 8'b00000011;
620     // inst_memory[74] = 8'b10100011;
621     // inst_memory[75] = 8'b00000000;
622 
623     // inst_memory[76] = 8'b00000011;
624     // inst_memory[77] = 8'b10111110;
625     // inst_memory[78] = 8'b00000010;
626     // inst_memory[79] = 8'b00000000;
627 
628     // inst_memory[80] = 8'b10000011;
629     // inst_memory[81] = 8'b00111110;
630     // inst_memory[82] = 8'b00000011;
631     // inst_memory[83] = 8'b00000000;
632     // //bge 111
633     // inst_memory[84] = 8'b01100011;
634     // inst_memory[85] = 8'b01011100;
635     // inst_memory[86] = 8'b11011110;
636     // inst_memory[87] = 8'b00000001;
637 
638     // inst_memory[88] = 8'b00110011;
639     // inst_memory[89] = 8'b00001111;
640     // inst_memory[90] = 8'b11000000;
641     // inst_memory[91] = 8'b00000001;
642 
643     // inst_memory[92] = 8'b00110011;
644     // inst_memory[93] = 8'b00001110;
645     // inst_memory[94] = 8'b11010000;
646     // inst_memory[95] = 8'b00000001;
647 
648     // inst_memory[96] = 8'b10110011;
649     // inst_memory[97] = 8'b00001110;
650     // inst_memory[98] = 8'b11100000;
651     // inst_memory[99] = 8'b00000001;
652 
653     // inst_memory[100] =8'b00100011;
654     // inst_memory[101] =8'b10110000;
655     // inst_memory[102] =8'b11000010;
656     // inst_memory[103] =8'b00000001;
```

```
657
658     // inst_memory[104] =8'b00100011;
659     // inst_memory[105] =8'b00110000;
660     // inst_memory[106] =8'b11010011;
661     // inst_memory[107] =8'b00000001;
662
663     // inst_memory[108] =8'b10010011;
664     // inst_memory[109] =8'b10001001;
665     // inst_memory[110] =8'b00011001;
666     // inst_memory[111] =8'b00000000;
667
668     // inst_memory[112] =8'b11100011;
669     // inst_memory[113] =8'b00000100;
670     // inst_memory[114] =8'b00000000;
671     // inst_memory[115] =8'b11111100;
672
673     // inst_memory[116] =8'b00010011;
674     // inst_memory[117] =8'b00001001;
675     // inst_memory[118] =8'b00011001;
676     // inst_memory[119] =8'b00000000;
677
678     // inst_memory[120] =8'b11100011;
679     // inst_memory[121] =8'b00001100;
680     // inst_memory[122] =8'b00000000;
681     // inst_memory[123] =8'b11111010;
682
683     // inst_memory[124] =8'b01100011;
684     // inst_memory[125] =8'b00000010;
685     // inst_memory[126] =8'b00000000;
686     // inst_memory[127] =8'b00000000;
687
688     // inst_memory[128] =8'b00010011;
689     // inst_memory[129] =8'b00000000;
690     // inst_memory[130] =8'b00000000;
691     // inst_memory[131] =8'b00000000;
692
693 end
694 always@(Inst_Address)
695     Instruction = {inst_memory[Inst_Address+3],inst_memory[Inst_Address+2],
        inst_memory[Inst_Address+1], inst_memory[Inst_Address]};
696 endmodule
697
698
699
700
701 module Instruction_Parser(
702     input [31:0] instruction,
703     output [6:0] opcode, funct7,
704     output [4:0] rd , rs1 , rs2,
705     output [2:0] funct3
706
707 );
708
709 assign opcode = instruction[6:0];
710 assign rd = instruction[11:7];
711 assign funct3 = instruction[14:12];
712 assign rs1 = instruction[19:15];
713 assign rs2 = instruction[24:20];
```

```
714 assign funct7 = instruction[31:25];
715
716 endmodule
717
718
719
720
721 module MEM_WB(
722     input clk, reset,
723     input [63:0] read_data, Result,
724     input EXMEM_MemtoReg, EXMEM_RegWrite,
725     input [4:0] EXMEM_inst2,
726     output reg [63:0] MEMWB_read_data, MEMWB_Result,
727     output reg MEMWB_MemtoReg, MEMWB_RegWrite,
728     output reg [4:0] MEMWB_inst2
729 );
730     always @(posedge clk or reset)
731     begin
732         if(clk)
733             begin
734                 MEMWB_read_data = read_data;
735                 MEMWB_Result = Result;
736                 MEMWB_MemtoReg = EXMEM_MemtoReg;
737                 MEMWB_RegWrite = EXMEM_RegWrite;
738                 MEMWB_inst2 = EXMEM_inst2;
739
740             end
741         else
742             begin
743                 MEMWB_read_data = 0;
744                 MEMWB_Result = 0;
745                 MEMWB_MemtoReg = 0;
746                 MEMWB_RegWrite = 0;
747                 MEMWB_inst2 = 0;
748             end
749     end
750 endmodule
751
752
753 module mux2x1
754 (
755     input [63:0] a,b,
756     input s ,
757     output [63:0] data_out
758 );
759
760 assign data_out = s ? b : a;
761
762 endmodule
763
764
765 module Program_Counter
766 (
767     input clk, reset,
768     input [63:0] PC_In,
769     output reg [63:0] PC_Out
770 );
771
```

```
reg reset_force; // variable to force 0th value after reset

initial
PC_Out <= 64'd0;


always @(posedge clk or posedge reset) begin
    if (reset || reset_force) begin
        PC_Out = 64'd0;
        reset_force <= 0;
        end

    // else if (!PCWrite) begin
    //     PC_Out = PC_Out;
    // end
    else
    PC_Out = PC_In;

end

always @(negedge reset) reset_force <= 1;

endmodule // Program_Counter



module registerFile(
    input [63:0] WriteData,
    input [4:0] RS1,
    input [4:0] RS2,
    input [4:0] RD,
    input RegWrite, clk, reset,
    output reg [63:0] ReadData1,
    output reg [63:0] ReadData2
);
reg [63:0] Registers [31:0];
initial
    begin
    Registers[0] = 64'd 0;
    Registers[1] = 64'd 0;
    Registers[2] = 64'd 0;
    Registers[3] = 64'd 0;
    Registers[4] = 64'd 0;
    Registers[5] = 64'd 0;
    Registers[6] = 64'd 0;
    Registers[7] = 64'd 0;
    Registers[8] = 64'd 0;
    Registers[9] = 64'd 0;
    Registers[10] = 64'd 0;
    Registers[11] = 64'd 0;
    Registers[12] = 64'd 0;
    Registers[13] = 64'd 0;
    Registers[14] = 64'd 0;
    Registers[15] = 64'd 0;
    Registers[16] = 64'd 0;
    Registers[17] = 64'd 0;
    Registers[18] = 64'd 0;
    Registers[19] = 64'd 0;
```

```
    Registers[20] = 64'd 0;
    Registers[21] = 64'd 0;
    Registers[22] = 64'd 0;
    Registers[23] = 64'd 0;
    Registers[24] = 64'd 0;
    Registers[25] = 64'd 0;
    Registers[26] = 64'd 0;
    Registers[27] = 64'd 0;
    Registers[28] = 64'd 0;
    Registers[29] = 64'd 0;
    Registers[30] = 64'd 0;
    Registers[31] = 64'd 0;
    end
always @(negedge clk)
    if(RegWrite)
        begin
        Registers[RD] = WriteData;
        end
  always @(*)
    if(reset)
        begin
        ReadData1 = 64'b0;
        ReadData2 = 64'b0;
        end
    else
        begin
        ReadData1 = Registers[RS1];
        ReadData2 = Registers[RS2];
        end
endmodule

module selector(
    input branch, ZERO,
    input [63:0] a, b,
    input [2:0] funct3,
    output reg sel
);

always@(*)
begin
    if (branch == 1)
        begin
            case(funct3)
            3'b101: //bne
            begin
                if(branch == 1 & ZERO == 0)
                    sel = 1;
            end
            3'b011: //beq
            begin
                if(branch == 1 & ZERO == 1)
                    sel = 1;
            end
            3'b111: //bge
            begin
                if (a >= b)
                    sel = 1;
            end
```

```
          endcase
        end
    else
        sel <= 0;
end
endmodule


module RISC_V_Processor(
    input clk, reset
);

//Mux output
wire [63:0] PC_In_from_mux;
//Program counter output
wire [63:0] PC_Out;
//Adders outputs
wire [63:0] a1_out;
wire [63:0] a2_out;
//Input to Adder a1
wire [63:0] b_in = 64'd4;
//Output from IM
wire [31:0] Instruction;

//Outputs from RegisterFile
wire [63:0] ReadData1;
wire [63:0] ReadData2;
//Outputs from Control Unit
wire [1:0] ALUOp;
wire Branch, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite;
//Outputs from ALU Control
wire [3:0] Operation;
//Funct input to ALU_Control
wire [3:0] Funct;
assign Funct = {Instruction[30], Instruction[14:12]};
//Output from ALU
wire [63:0] Result_from_alu;
wire zero_output;
//Output from Data generator
wire [63:0] imm_data;
//Output from mux2
wire [63:0] out_from_mux2;
//Output from Data memory
wire [63:0] out_from_DM;
//Output from mux3
wire [63:0] out_from_mux3;
//Outputs from IF_ID
wire [31:0] IFID_instruction;
wire [63:0] IFID_PC_Out;
wire [4:0] rd;
wire [4:0] rs1;
wire [4:0] rs2;
wire [6:0] opcode;
wire [6:0] funct7;
wire [2:0] funct3;
//Outputs from ID_EX
wire [63:0] IDEX_PC_Out;
wire [63:0] IDEX_ReadData1;
```

```
wire [63:0] IDEX_ReadData2;
wire [63:0] IDEX_imm_data;
wire [3:0] IDEX_inst1;
wire [4:0] IDEX_inst2;
wire [1:0] IDEX_ALUOp;
wire IDEX_Branch, IDEX_MemRead, IDEX_MemtoReg, IDEX_MemWrite, IDEX_ALUSrc,
    IDEX_Regwrite;
//Input to Adder a2
wire [63:0] b_adder2;
assign b_adder2 = IDEX_imm_data << 1;
//Outputs from EX_MEM
wire [63:0] EXMEM_out;
wire EXMEM_ZERO;
wire [63:0] EXMEM_Result;
wire [63:0] EXMEM_ReadData2;
wire [4:0] EXMEM_inst2;
wire EXMEM_Branch, EXMEM_MemRead, EXMEM_MemtoReg, EXMEM_MemWrite,
    EXMEM_RegWrite;
//Outputs from MEM_WB
wire [63:0] MEMWB_read_data;
wire [63:0] MEMWB_Result;
wire MEMWB_MemtoReg, MEMWB_RegWrite;
wire [4:0] MEMWB_inst2;
//sel to mux1
wire PC_src;
// Outputs from Forwarding Unit
  wire [1:0] FU_fwdA;
  wire [1:0] FU_fwdB;
//Outputs from Triple MUX
wire [63:0] Res;
// wire [63:0] ResB;
  wire [63:0] Resa;
  wire [63:0] Resb;
//addi x5, x0, 3
IF_ID ifid
        (.clk(clk),
        .reset(reset),
        .instruction(Instruction),
        .PC_Out(PC_Out),
        .IFID_instruction(IFID_instruction),
        .IFID_PC_Out(IFID_PC_Out));

ID_EX idex
        (.clk(clk),
         .reset(reset),
         .ALUOp(ALUOp),
         .Branch(Branch),
         .MemRead(MemRead),
         .MemtoReg(MemtoReg),
         .MemWrite(MemWrite),
         .ALUSrc(ALUSrc),
         .RegWrite(RegWrite),
         .ReadData1(ReadData1),
         .ReadData2(ReadData2),
         .PC_Out(IFID_PC_Out),
         .imm_data(imm_data),
         .inst1({IFID_instruction[30], IFID_instruction[14:12]}),
         .inst2(rd),
```

```
1002
1003          .IDEX_Branch(IDEX_Branch),
1004          .IDEX_MemRead(IDEX_MemRead),
1005          .IDEX_MemtoReg(IDEX_MemtoReg),
1006          .IDEX_MemWrite(IDEX_MemWrite),
1007          .IDEX_ALUSrc(IDEX_ALUSrc),
1008          .IDEX_Regwrite(IDEX_Regwrite),
1009          .IDEX_ALUOp(IDEX_ALUOp),
1010          .IDEX_ReadData1(IDEX_ReadData1),
1011          .IDEX_ReadData2(IDEX_ReadData2),
1012          .IDEX_PC_Out(IDEX_PC_Out),
1013          .IDEX_imm_data(IDEX_imm_data),
1014          .IDEX_inst1(IDEX_inst1),
1015          .IDEX_inst2(IDEX_inst2));
1016
1017
1018
1019 EX_MEM exmem(.clk(clk),
1020              .reset(reset),
1021              .IDEX_Branch(IDEX_Branch),
1022              .IDEX_MemRead(IDEX_MemRead),
1023              .IDEX_MemtoReg(IDEX_MemtoReg),
1024              .IDEX_MemWrite(IDEX_MemWrite),
1025              .IDEX_Regwrite(IDEX_Regwrite),
1026              .out(a2_out),
1027              .ZERO(zero_output),
1028              .Result(Result_from_alu),
1029              .IDEX_ReadData2(IDEX_ReadData2),
1030              .IDEX_inst2(IDEX_inst2),
1031
1032              .EXMEM_out(EXMEM_out),
1033              .EXMEM_Result(EXMEM_Result),
1034              .EXMEM_ReadData2(EXMEM_ReadData2),
1035              .EXMEM_inst2(EXMEM_inst2),
1036              .EXMEM_ZERO(EXMEM_ZERO),
1037              .EXMEM_Branch(EXMEM_Branch),
1038              .EXMEM_MemRead(EXMEM_MemRead),
1039              .EXMEM_MemtoReg(EXMEM_MemtoReg),
1040              .EXMEM_MemWrite(EXMEM_MemWrite),
1041              .EXMEM_Regwrite(EXMEM_RegWrite));
1042
1043
1044 MEM_WB memwb(.clk(clk),
1045               .reset(reset),
1046               .EXMEM_MemtoReg(EXMEM_MemtoReg),
1047               .EXMEM_RegWrite(EXMEM_RegWrite),
1048               .read_data(out_from_DM),
1049               .Result(EXMEM_Result),
1050               .EXMEM_inst2(EXMEM_inst2),
1051
1052               .MEMWB_read_data(MEMWB_read_data),
1053               .MEMWB_Result(MEMWB_Result),
1054               .MEMWB_MemtoReg(MEMWB_MemtoReg),
1055               .MEMWB_RegWrite(MEMWB_RegWrite),
1056               .MEMWB_inst2(MEMWB_inst2));
1057
1058 Instruction_Parser ip
1059         (.instruction(IFID_instruction),
```

```
        .rd(rd),
        .rs1(rs1),
        .rs2(rs2),
        .funct3(funct3),
        .funct7(funct7),
        .opcode(opcode));

Program_Counter pc
            (.clk(clk),
            .reset(reset),
            .PC_In(PC_In_from_mux),
            .PC_Out(PC_Out));

Adder a1
        (.a(PC_Out),
        .b(b_in),
        .out(a1_out));

Adder a2
        (.a(IDEX_PC_Out),
        .b(b_adder2),
        .out(a2_out));

mux2x1 mux1
        (.a(a1_out),
        .b(EXMEM_out),
        .s(PC_src),
        .data_out(PC_In_from_mux));

mux2x1 mux2
        (.a(IDEX_ReadData2),
        .b(IDEX_imm_data),
        .s(IDEX_ALUSrc),
        .data_out(out_from_mux2));

mux2x1 mux3
        (.b(MEMWB_read_data),
        .a(MEMWB_Result),
        .s(MEMWB_MemtoReg),
        .data_out(out_from_mux3));

Instruction_Memory im
        (.Inst_Address(PC_Out),
        .Instruction(Instruction));

registerFile rf
        (.WriteData(out_from_mux3),
        .RS1(rs1),
        .RS2(rs2),
        .RD(MEMWB_inst2),
        .clk(clk),
        .reset(reset),
        .RegWrite(MEMWB_RegWrite),
        .ReadData1(ReadData1),
        .ReadData2(ReadData2));

Control_Unit cu
        (.Opcode(opcode),
```

```
        .ALUOp(ALUOp),
        .Branch(Branch),
        .MemRead(MemRead),
        .MemtoReg(MemtoReg),
        .MemWrite(MemWrite),
        .ALUSrc(ALUSrc),
        .Regwrite(RegWrite));

ALU_Control aluc
        (.ALUOp(IDEX_ALUOp),
        .Funct(IDEX_inst1),
        .Operation(Operation));

data_generator dg
        (.instruction(IFID_instruction),
        .imm_data(imm_data));

alu_64 alu
  (.a(Resa),
   .b(Resb),
        .ALUOp(Operation),
        .Result(Result_from_alu),
        .ZERO(zero_output));

selector s
        (.branch(EXMEM_Branch),
        .ZERO(EXMEM_ZERO),
        .a(IDEX_ReadData1),
        .b(out_from_mux2),
        .funct3(funct3),
        .sel(PC_src)
        );

Data_Memory dm
        (.mem_addr(EXMEM_Result),
        .write_data(EXMEM_ReadData2),
        .clk(clk),
        .mem_write(EXMEM_MemWrite),
        .mem_read(EXMEM_MemRead),
        .read_data(out_from_DM));

Forwarding_Unit fu
  (.EXMEM_ReadData2(FU_EXMEM_ReadData2),
   .MEMWB_read_data(FU_MEMWB_read_data),
   .rs1(FU_IDEX_inst1),
   .rs2(FU_IDEX_inst2),
   .EXMEM_Regwrite(FU_EXMEM_Regwrite),
   .MEMWB_RegWrite(FU_MEMWB_RegWrite),
   .fwd_A(FU_fwdA),
   .fwd_B(FU_fwdB));


MUX_Triple mux_for_a
(
  .a(IDEX_ReadData1), //00
    .b(out_from_mux3), //01
    .c(EXMEM_Result),   //10
    .sel(FU_fwdA),
```

```
  .Res(Resa)
);

MUX_Triple mux_for_b
(
  .a(IDEX_ReadData2), //00
    .b(out_from_mux3), //01
    .c(EXMEM_Result),   //10
    .sel(FU_fwdB),
  .Res(Resb)
);



always @(posedge clk)
    begin
        $monitor(
        "PC_In = ", PC_In_from_mux,
        ", PC_Out = ", PC_Out,
        ", Instruction = %b", Instruction,
        ", Opcode = %b", opcode,
        ", Funct3 = %b", funct3,
        ", rs1 = %d", rs1,
        ", rs2 = %d", rs2,
        ", rd = %d", rd,
        ", funct7 = %b", IFID_instruction[31:25],
        ", ALUOp = %b", IDEX_ALUOp,
        ", imm_data = %d", imm_data,
        ", Operation = %b", Operation
        );
    end
endmodule
```

Test Bench:

```

module tb();

reg clk, reset;

RISC_V_Processor processor(.clk(clk), .reset(reset));

initial
begin
  $dumpfile("dump.vcd");
  $dumpvars();
  clk = 1'd0;
  reset = 1'd1;
  #10
  reset = 1'd0;
  #500
  $finish;
end

always #5 clk=~clk;

endmodule
```

Chapter 4

Code

4.1 EDA Links

Task 1 EDA Link

Task 2 EDA Link