



ELEC 374 – Phase 3 Report

Department of Electrical and Computer Engineering
Queen's University

Group 16

Composed By:
Zeerak Asim (20237955)
Nicholas Seegobin (20246787)

Date of Submission:
31 March 2023

"We do hereby verify that this written lab report is our own work and contains our own original ideas, concepts, and designs. No portion of this report has been copied in whole or in part from another source, with the possible exception of properly referenced material".

Table of Contents

| | |
|-------------------------------------|----|
| Registers..... | 3 |
| General Register..... | 3 |
| Memory Data Register | 3 |
| Revised Register 0..... | 3 |
| Program Counter..... | 4 |
| Arithmetic Logic Unit | 5 |
| Bus System | 9 |
| Datapath | 9 |
| Select Encode Logic..... | 12 |
| Encoder | 13 |
| Multiplexer..... | 14 |
| Con Flip Flop..... | 15 |
| Control Unit..... | 16 |
| Memory Subsystem | 31 |
| Ram | 31 |
| Ram.mif File | 32 |
| Ports | 35 |
| Inport | 35 |
| Outport | 36 |
| Control Unit Testbench..... | 36 |
| IR Storing Data from Ram | 37 |
| Dedicated PC Incrementor..... | 37 |
| PC Changing for BRMI and BRPL | 38 |
| Instruction Fetch and Decode..... | 39 |

Registers

General Register

```
hdl > registers > V register.v
1  /* Representation of a register in Verilog HDL. */
2  /* Declared 3, 1-bit signals, 1, 32-bit D-input, and 1, 32-bit Q-output. */
3  /* Data type of each signal/input is a wire and output is a reg. */
4  module register(input clk, input clr, input enable, input [31:0] D, output reg [31:0] Q);
5
6      /* While loop that iterates every positive clock edge. */
7      always @(posedge clk)
8      begin
9          /* If clear signal is high, set Q output to 0. */
10         if(clr)
11             Q <= 0;
12
13         /* If enable signal is high, set Q output to follow (or equal to) D input. */
14         else if(enable)
15             Q = D;
16     end
17 endmodule // Register end.
```

Memory Data Register

```
hdl > registers > V md_register.v
1  /* Representation of a memory data register with 2 to 1 multiplexer in Verilog */
2  module md_register(input clk, input clr, input enable, input read, input [31:0] MDataIN, input [31:0] bus_Data, output reg [31:0] Q);
3
4      /* While loop that iterates every positive clock edge. */
5      always @(posedge clk)
6      begin
7
8          /* If clear signal is high, set Q output to 0. */
9          if(clr)
10             Q <= 0;
11
12         /* If enable signal is high, set Q output to follow (or equal to) D input. */
13         else if(enable)
14             Q <= read ? MDataIN : bus_Data;
15     end
16 endmodule // md_register end.
```

Revised Register 0

```
hdl > registers > V R0_revised.v
1  /* Representation of a register in Verilog HDL. */
2  /* Declared 3, 1-bit signals, 1, 32-bit D-input, and 1, 32-bit Q-output. */
3  /* Data type of each signal/input is a wire and output is a reg. */
4  module R0_revised(input clk, input clr, input enable, input ba_select, input [31:0] bus_Data, output wire [31:0] R0_Data);
5
6      wire [31:0] registerOutput;
7
8      register R0(clk, clr, enable, bus_Data, registerOutput);
9      assign R0_Data = registerOutput & {32{!ba_select}};
10
11 endmodule // revised_register_R0 end.
```

Program Counter

```
hdl > registers > V program_counter.v
1  module program_counter(
2      input wire clk, clr, enable, incPC,
3      input wire [31:0] PC_Input,
4      output wire [31:0] PC_Output
5  );
6
7      reg [31:0] Q;
8      reg incFlag; // flag to indicate that PC should be incremented on next clock edge
9
10     initial begin
11         Q <= 0;
12         incFlag = 1;
13     end
14
15     always @ (posedge clk) begin
16         if (clr) begin
17             Q <= 0;
18         end
19         else if (enable) begin
20             Q <= PC_Input;
21         end
22         else begin
23             if (incPC == 1 && incFlag == 1) begin
24                 Q <= Q + 1;
25                 incFlag <= 0;
26             end
27             else if (incPC == 0) begin
28                 incFlag <= 1;
29             end
30         end
31     end
32
33     assign PC_Output = Q;
34
35     endmodule
36
```

Arithmetic Logic Unit

```
hdl > alu > V alu.v
1  module alu (
2      // ALU Inputs
3      input wire [31:0] A, B, Y,    //added Y
4      input wire [4:0] opcode,
5
6      // ALU Output
7      output reg [63:0] result);
8
9      parameter load = 5'b00000, loadImm = 5'b00001, store = 5'b00010, add = 5'b00011, sub = 5'b00100,
10     AND = 5'b00101, OR = 5'b00110, right_shift = 5'b00111, arith_shift_right = 5'b01000, left_shift = 5'b01001,
11     right_rotate = 5'b01010, left_rotate = 5'b01011, addImm = 5'b01100, AND_Imm = 5'b01101, OR_Imm = 5'b01110,
12     mul = 5'b01111, div = 5'b10000, negate = 5'b10001, NOT = 5'b10010, branch = 5'b10011, jr = 5'b10100, jal = 5'b10101,
13     in = 5'b10110, out = 5'b10111, mfhi = 5'b11000, mflo = 5'b11001;
14
15     // And Operation
16     wire [31:0] and_result;
17     logical_and andInstance(.A(A), .B(B), .result(and_result));
18
19     // Or Operation
20     wire [31:0] or_result;
21     logical_or orInstance(A, B, or_result);
22
23     // Add Operation
24     wire [31:0] sum_result;
25     wire carryOut;
26     adder adderInstance(A, B, sum_result, carryOut);
27
28     // Subtractor Operation
29     wire [31:0] difference_result;
30     wire borrowOut;
31     subtractor subtractorInstance(A, B, difference_result, borrowOut);
32
33     // Multiplication Operation
34     wire [63:0] product_result;
35     multiplier multiplierInstance(A, B, product_result);
36
37     // Division Operation
38     wire [31:0] quotient_result, remainder_result;
39     divider dividerInstance(A, B, quotient_result, remainder_result);
40
41     // Left Shift Operation
42     wire [31:0] leftShift_result;
43     shift_left leftShiftInstance(A, B, leftShift_result);
44
45     // Right Shift Operation
46     wire [31:0] rightShift_result;
47     shift_right rightShiftInstance(A, B, rightShift_result);
48
```

```
49 // Arithmetic Right Shift Operation
50 wire [31:0] arithShiftRight_result;
51 arithmetic_shift_right arithShiftRightInstance(A, B, arithShiftRight_result);
52
53 // Left Rotate Operation
54 wire [31:0] leftRotate_result;
55 rotate_left leftRotateInstance(A, B, leftRotate_result);
56
57 // Right Rotate Operation
58 wire [31:0] rightRotate_result;
59 rotate_right rightRotateInstance(A, B, rightRotate_result);
60
61 // Negate Operation
62 wire [31:0] negate_result;
63 negate negateInstance(B, negate_result);
64
65 // Not Operation
66 wire [31:0] not_result;
67 logical_not notInstance(B, not_result);
68
```

```

69 // Select operation
70 always @(*) begin
71     case(opcode)
72         // Add Operation
73         add : begin
74             result[31:0] <= sum_result[31:0];
75             result[63:32] <= 32'd0;
76         end
77
78         // Subtract Operation
79         sub : begin
80             result[31:0] <= difference_result[31:0];
81             result[63:32] <= 32'd0;
82         end
83
84         // And Operation
85         AND, AND_Imm : begin
86             result[31:0] <= and_result[31:0];
87             result[63:32] <= 32'd0;
88         end
89
90         // Or Operation
91         OR, OR_Imm : begin
92             result[31:0] <= or_result[31:0];
93             result[63:32] <= 32'd0;
94         end
95
96         // Right Shift Operation
97         right_shift : begin
98             result[31:0] <= rightShift_result[31:0];
99             result[63:32] <= 32'd0;
100         end
101
102         // Arithmetic Right Shift Operation
103         arith_shift_right : begin
104             result[31:0] <= arithShiftRight_result[31:0];
105             result[63:32] <= 32'd0;
106         end
107
108         // Left Shift Operation
109         left_shift : begin
110             result[31:0] <= leftShift_result[31:0];
111             result[63:32] <= 32'd0;
112         end
113

```

```

114 // Right Rotate Operation
115 right_rotate : begin
116     result[31:0] <= rightRotate_result[31:0];
117     result[63:32] <= 32'd0;
118 end
119
120 // Left Rotate Operation
121 left_rotate : begin
122     result[31:0] <= leftRotate_result[31:0];
123     result[63:32] <= 32'd0;
124 end
125
126 // Multiply Operation
127 mul : begin
128     result[63:0] <= product_result[63:0];
129 end
130
131 // Divide Operation
132 div : begin
133     result[31:0] <= quotient_result[31:0];
134     result[63:32] <= remainder_result[31:0];
135 end
136
137 // Negate Operation
138 negate : begin
139     result[31:0] <= negate_result[31:0];
140     result[63:32] <= 32'd0;
141 end
142
143 // Not Operation
144 NOT : begin
145     result[31:0] <= not_result[31:0];
146     result[63:32] <= 32'd0;
147 end
148
149 // Default Case
150 default : begin
151     result[63:0] <= 64'd0;
152 end
153 endcase
154 end
155 endmodule
156

```


Bus System

Datapath

```
hdl > bus > V datapath.v
1  /* Representation of the datapath in Verilog HDL. */
2  module datapath(
3      // CPU signals
4      input wire clk, clr,
5      input wire [31:0] input_Data,
6      output wire [31:0] outport_Data
7  );
8
9      /* Enable Signals */
10     // General Register Enable Signals
11     wire R15_enable;
12     wire manual_R15_enable;
13     wire [15:0] register_enable;
14
15     // Program Counter Enable Signals
16     wire PC_enable, PC_increment_enable;
17
18     // Instruction Register Enable Signal
19     wire IR_enable;
20
21     // CON Flip Flop Enable Signal
22     wire con_enable;
23
24     // ALU 'Y' Register Enable Signal
25     wire Y_enable;
26
27     // Z_HI and Z_LO Enable Signal
28     wire Z_enable;
29
30     // HI and LO Register Enable Signals
31     wire HI_enable, LO_enable;
32
33     // Memory Register Enable Signals
34     wire MAR_enable, MDR_enable, read, write;
35
36     // Outport Register Enable Signal
37     wire outport_enable;
38
39     /* Select Signals */
40     // General Register Select Signals
41     wire [15:0] register_select;
42
43     // 32-to-5 Encoder Output
44     wire [4:0] bus_select;
45
46     // PC Register Select Signal
47     wire PC_select;
48
```

```

49 // HI/LO Register Select Signal
50 wire HI_select, LO_select;
51
52 // Z_HI/Z_LO Register Select Signal
53 wire Z_HI_select, Z_LO_select;
54
55 // MDR Select Signal
56 wire MDR_select;
57
58 // Inport Select Signal
59 wire inport_select;
60
61 // C_Sign_Extended Select Signal
62 wire c_select;
63
64 /* Data Signals */
65 wire [31:0] bus_Data; // Data currently in the bus
66
67 wire [63:0] aluResult;
68 wire [4:0] alu_instruction; // ALU Opcode
69
70 // General Register Contents
71 wire [31:0] R0_Data, R1_Data, R2_Data, R3_Data, R4_Data, R5_Data, R6_Data, R7_Data,
72 R8_Data, R9_Data, R10_Data, R11_Data, R12_Data, R13_Data, R14_Data, R15_Data;
73
74 // Instruction Register and Program Counter Contents
75 wire [31:0] PC_Data, IR_Data;
76
77 // ALU Input Register 'Y' Contents
78 wire [31:0] Y_Data;
79
80 // ALU Output Register Contents
81 wire [31:0] Z_HI_Data, Z_LO_Data;
82 wire [31:0] HI_Data, LO_Data;
83
84 // RAM Memory Address Register (MAR) Contents
85 wire [8:0] MAR_Data;
86
87 // RAM Memory Data Register (MDR) Contents
88 wire [31:0] MDR_Data, MDataIN;
89
90 // C Sign Extended Data
91 wire [31:0] C_sign_ext_Data;
92
93 // Port Register Contents
94 wire [31:0] inport_Data;
95

```

```
hdl > bus > V datapath.v
```

```
96  /* Select and Encode Signals */
97  wire Gra, Grb, Grc, r_enable, r_select, ba_select;
98
99  /* CON FF Output */
100 wire con_output;
101
102  /* Bus Components */
103  // 32-to-5 Encoder
104  encoder encoder_instance(.register_select(register_select), .HI_select(HI_select), .LO_select(LO_select),
105  .Z_HI_select(Z_HI_select), .Z_LO_select(Z_LO_select), .PC_select(PC_select), .MDR_select(MDR_select),
106  .inport_select(inport_select), .c_select(c_select), .selectSignal(bus_select));
107
108  // 32-to-1 Multiplexer
109  multiplexer multiplexer_instance(.selectSignal(bus_select), .muxIN_r0(R0_Data),
110  .muxIN_r1(R1_Data), .muxIN_r2(R2_Data), .muxIN_r3(R3_Data), .muxIN_r4(R4_Data),
111  .muxIN_r5(R5_Data), .muxIN_r6(R6_Data), .muxIN_r7(R7_Data), .muxIN_r8(R8_Data),
112  .muxIN_r9(R9_Data), .muxIN_r10(R10_Data), .muxIN_r11(R11_Data), .muxIN_r12(R12_Data),
113  .muxIN_r13(R13_Data), .muxIN_r14(R14_Data), .muxIN_r15(R15_Data), .muxIN_HI(HI_Data),
114  .muxIN_LO(LO_Data), .muxIN_Z_HI(Z_HI_Data), .muxIN_Z_LO(Z_LO_Data), .muxIN_PC(PC_Data),
115  .muxIN_MDR(MDR_Data), .muxIN_inport(inport_Data), .muxIN_C_sign_ext(C_sign_ext_Data), .muxOut(bus_Data));
116
117  // General purpose registers r0 -> r15
118  R0_revised r0 (.clk(clk), .clr(clr), .enable(register_enable[0]), .ba_select(ba_select), .bus_Data(bus_Data), .R0_Data(R0_Data));
119  register r1 (.clk(clk), .clr(clr), .enable(register_enable[1]), .D(bus_Data), .Q(R1_Data));
120  register r2 (.clk(clk), .clr(clr), .enable(register_enable[2]), .D(bus_Data), .Q(R2_Data));
121  register r3 (.clk(clk), .clr(clr), .enable(register_enable[3]), .D(bus_Data), .Q(R3_Data));
122  register r4 (.clk(clk), .clr(clr), .enable(register_enable[4]), .D(bus_Data), .Q(R4_Data));
123  register r5 (.clk(clk), .clr(clr), .enable(register_enable[5]), .D(bus_Data), .Q(R5_Data));
124  register r6 (.clk(clk), .clr(clr), .enable(register_enable[6]), .D(bus_Data), .Q(R6_Data));
125  register r7 (.clk(clk), .clr(clr), .enable(register_enable[7]), .D(bus_Data), .Q(R7_Data));
126  register r8 (.clk(clk), .clr(clr), .enable(register_enable[8]), .D(bus_Data), .Q(R8_Data));
127  register r9 (.clk(clk), .clr(clr), .enable(register_enable[9]), .D(bus_Data), .Q(R9_Data));
128  register r10 (.clk(clk), .clr(clr), .enable(register_enable[10]), .D(bus_Data), .Q(R10_Data));
129  register r11 (.clk(clk), .clr(clr), .enable(register_enable[11]), .D(bus_Data), .Q(R11_Data));
130  register r12 (.clk(clk), .clr(clr), .enable(register_enable[12]), .D(bus_Data), .Q(R12_Data));
131  register r13 (.clk(clk), .clr(clr), .enable(register_enable[13]), .D(bus_Data), .Q(R13_Data));
132  register r14 (.clk(clk), .clr(clr), .enable(register_enable[14]), .D(bus_Data), .Q(R14_Data));
133
134  assign R15_enable = manual_R15_enable | register_enable[15];
135  register r15 (.clk(clk), .clr(clr), .enable(R15_enable), .D(bus_Data), .Q(R15_Data));
136
137  // ALU Output Registers
138  register HI (.clk(clk), .clr(clr), .enable(HI_enable), .D(bus_Data), .Q(HI_Data));
139  register LO (.clk(clk), .clr(clr), .enable(LO_enable), .D(bus_Data), .Q(LO_Data));
140  register Z_HI (.clk(clk), .clr(clr), .enable(Z_enable), .D(aluResult[63:32]), .Q(Z_HI_Data));
141  register Z_LO (.clk(clk), .clr(clr), .enable(Z_enable), .D(aluResult[31:0]), .Q(Z_LO_Data));
142  register Y (.clk(clk), .clr(clr), .enable(Y_enable), .D(bus_Data), .Q(Y_Data));
143
144  // ALU Instance
145  alu alu_instance(.A(Y_Data), .B(bus_Data), .opcode(alu_instruction), .result(aluResult));
146
147  // PC and IR Registers
148  program_counter PC (.clk(clk), .clr(clr), .enable(PC_enable), .incPC(PC_increment_enable), .PC_Input(bus_Data), .PC_Output(PC_Data));
149  register IR (.clk(clk), .clr(clr), .enable(IR_enable), .D(bus_Data), .Q(IR_Data));
150
151  // Memory Registers
152  register MAR (.clk(clk), .clr(clr), .enable(MAR_enable), .D(bus_Data), .Q(MAR_Data));
153  md_register MDR (.clk(clk), .clr(clr), .enable(MDR_enable), .read(read), .MDataIN(MDataIN), .bus_Data(bus_Data), .Q(MDR_Data));
154
155  // Ram Instance
156  ram ram_instance(.debug_port_01(debug_port_01), .debug_port_02(debug_port_02), .clk(clk), .read(read), .write(write), .data_in(MDR_Data), .address_in(MAR_Data), .data_out(MDataIN));
157
158  select_encode_logic sel_instance(.instruction(IR_Data), .Gra(Gra), .Grb(Grb), .Grc(Grc), .r_enable(r_enable), .r_select(r_select),
159  .ba_select(ba_select), .C_sign_ext_Data(C_sign_ext_Data), .register_enable(register_enable), .register_select(register_select));
160
161  con_ff con_instance(.bus_Data(bus_Data), .instruction(IR_Data), .con_enable(con_enable), .con_output(con_output));
162
163  inport inport_instance(.clk(clk), .clr(clr), .input_Data(input_Data), .inport_Data(inport_Data));
164  outport outport_instance(.clk(clk), .clr(clr), .enable(outport_enable), .bus_Data(bus_Data), .outport_Data(outport_Data));
165
166  control_unit cu_instance(
167  clk, reset, con_output, IR_Data, alu_instruction, read, write, Gra, Grb, Grc, r_enable, r_select, ba_select,
168  PC_enable, PC_increment_enable, IR_enable, con_enable, Y_enable, Z_enable, HI_enable, LO_enable,
169  MAR_enable, MDR_enable, outport_enable, manual_R15_enable,
170  MDR_select, Z_HI_select, Z_LO_select, HI_select, LO_select, PC_select, inport_select, c_select);
171
172  endmodule // Datapath end.
```

Select Encode Logic

```
hdl > bus > select_encode_logic > V select_encode_logic.v
1  module select_encode_logic(
2      // Instruction Register Data
3      input [31:0] instruction,
4
5      // Select and Encode Logic Control Signals
6      input Gra, Grb, Grc, r_enable, r_select, ba_select,
7
8      // Array of Enable and Select Output Signals
9      output [15:0] register_enable, register_select,
10
11     // Sign Extension of Constant C
12     output [31:0] C_sign_ext_Data);
13
14     // Instruction Register Content Variables
15     reg [3:0] Ra, Rb, Rc;
16
17     // 4 to 16 Decoder Input/Output Variable
18     reg [3:0] decoder_input;
19     reg [15:0] decoder_out;
20
21     always @ (instruction, Gra, Grb, Grc) begin
22         // Assigning Values to Instruction Register Content Variables
23         Ra = instruction [26:23]; // Bit 23 -> 26 of the instruction register is register a
24         Rb = instruction [22:19]; // Bit 19 -> 22 of the instruction register is register b
25         Rc = instruction [19:15]; // Bit 15 -> 19 of the instruction register is register c
26
27         if (Gra) decoder_input = Ra;
28         else if (Grb) decoder_input = Rb;
29         else if (Grc) decoder_input = Rc;
30
31         case (decoder_input)
32             4'd0: decoder_out = 16'd1;
33             4'd1: decoder_out = 16'd2;
34             4'd2: decoder_out = 16'd4;
35             4'd3: decoder_out = 16'd8;
36             4'd4: decoder_out = 16'd16;
37             4'd5: decoder_out = 16'd32;
38             4'd6: decoder_out = 16'd64;
39             4'd7: decoder_out = 16'd128;
40             4'd8: decoder_out = 16'd256;
41             4'd9: decoder_out = 16'd512;
42             4'd10: decoder_out = 16'd1024;
43             4'd11: decoder_out = 16'd2048;
44             4'd12: decoder_out = 16'd4096;
45             4'd13: decoder_out = 16'd8192;
46             4'd14: decoder_out = 16'd16384;
47             4'd15: decoder_out = 16'd32768;
48         endcase
49     end
50
51     // Assigning Values to the Outputs
52     assign register_enable = {16{r_enable}} & decoder_out;
53     assign register_select = ({16{ba_select}} | {16{r_select}}) & decoder_out;
54     assign C_sign_ext_Data = {{13{instruction[18]}}, instruction[18:0]};
55
56 endmodule
```

Encoder

```
hdl> bus> V encoder.v
1  /* Representation of an encoder in Verilog HDL. */
2  /* Declared 32, 1-bit inputs and 5 output select signals. */
3  /* Data type of each input is a wire. */
4  module encoder(input [15:0] register_select, input HI_select, input LO_select, input Z_HI_select, input Z_LO_select,
5  input PC_select, input MDR_select, input inport_select, input c_select, output reg [4:0] selectSignal);
6
7      /* While loop to update the selectSignal output wire. */
8      always @* begin
9          if (register_select[0]) selectSignal <= 5'b00000;
10         else if (register_select[1]) selectSignal <= 5'b00001;
11         else if (register_select[2]) selectSignal <= 5'b00010;
12         else if (register_select[3]) selectSignal <= 5'b00011;
13         else if (register_select[4]) selectSignal <= 5'b00100;
14         else if (register_select[5]) selectSignal <= 5'b00101;
15         else if (register_select[6]) selectSignal <= 5'b00110;
16         else if (register_select[7]) selectSignal <= 5'b00111;
17         else if (register_select[8]) selectSignal <= 5'b01000;
18         else if (register_select[9]) selectSignal <= 5'b01001;
19         else if (register_select[10]) selectSignal <= 5'b01010;
20         else if (register_select[11]) selectSignal <= 5'b01011;
21         else if (register_select[12]) selectSignal <= 5'b01100;
22         else if (register_select[13]) selectSignal <= 5'b01101;
23         else if (register_select[14]) selectSignal <= 5'b01110;
24         else if (register_select[15]) selectSignal <= 5'b01111;
25         else if (HI_select) selectSignal <= 5'b10000;
26         else if (LO_select) selectSignal <= 5'b10001;
27         else if (Z_HI_select) selectSignal <= 5'b10010;
28         else if (Z_LO_select) selectSignal <= 5'b10011;
29         else if (PC_select) selectSignal <= 5'b10100;
30         else if (MDR_select) selectSignal <= 5'b10101;
31         else if (inport_select) selectSignal <= 5'b10110;
32         else if (c_select) selectSignal <= 5'b10111;
33         else selectSignal <= 5'b00000; // optional, to avoid latch.
34     end
35 endmodule // Encoder end.
```

Multiplexer

```
hdl> bus > V multiplexer.v
1  /* Representation of a multiplexer in Verilog HDL. */
2  /* Declared 32, 32-bit inputs, 5 select signals, and 1, 32-bit output. */
3  /* Data type of each input/select signal is a wire and output is a reg. */
4
5  module multiplexer(input [4:0] selectSignal, input [31:0] muxIN_r0,
6  input [31:0] muxIN_r1, input [31:0] muxIN_r2, input [31:0] muxIN_r3,
7  input [31:0] muxIN_r4, input [31:0] muxIN_r5, input [31:0] muxIN_r6,
8  input [31:0] muxIN_r7, input [31:0] muxIN_r8, input [31:0] muxIN_r9,
9  input [31:0] muxIN_r10, input [31:0] muxIN_r11, input [31:0] muxIN_r12,
10 input [31:0] muxIN_r13, input [31:0] muxIN_r14, input [31:0] muxIN_r15,
11 input [31:0] muxIN_HI, input [31:0] muxIN_LO, input [31:0] muxIN_Z_HI,
12 input [31:0] muxIN_Z_LO, input [31:0] muxIN_PC, input [31:0] muxIN_MDR,
13 input [31:0] muxIN_inport, input [31:0] muxIN_C_sign_ext, output reg [31:0] muxOut);
14
15     /* While loop to check for updates in the select signals, which updates the mux output. */
16     always @(*) begin
17         case (selectSignal)
18             5'b00000: muxOut <= muxIN_r0;
19             5'b00001: muxOut <= muxIN_r1;
20             5'b00010: muxOut <= muxIN_r2;
21             5'b00011: muxOut <= muxIN_r3;
22             5'b00100: muxOut <= muxIN_r4;
23             5'b00101: muxOut <= muxIN_r5;
24             5'b00110: muxOut <= muxIN_r6;
25             5'b00111: muxOut <= muxIN_r7;
26             5'b01000: muxOut <= muxIN_r8;
27             5'b01001: muxOut <= muxIN_r9;
28             5'b01010: muxOut <= muxIN_r10;
29             5'b01011: muxOut <= muxIN_r11;
30             5'b01100: muxOut <= muxIN_r12;
31             5'b01101: muxOut <= muxIN_r13;
32             5'b01110: muxOut <= muxIN_r14;
33             5'b01111: muxOut <= muxIN_r15;
34             5'b10000: muxOut <= muxIN_HI;
35             5'b10001: muxOut <= muxIN_LO;
36             5'b10010: muxOut <= muxIN_Z_HI;
37             5'b10011: muxOut <= muxIN_Z_LO;
38             5'b10100: muxOut <= muxIN_PC;
39             5'b10101: muxOut <= muxIN_MDR;
40             5'b10110: muxOut <= muxIN_inport;
41             5'b10111: muxOut <= muxIN_C_sign_ext;
42             default: muxOut <= 32'd0;
43         endcase
44     end
45 endmodule // Multiplexer end.
46
```

Con Flip Flop

```
hdl> con_ff > V con_ff.v
1  /* This module determines whether the correct condition has been met to cause branching to take place in a conditional branch instruction. */
2  module con_ff(
3      // Bus Input
4      input [31:0] bus_Data,
5
6      // Instruction Register Input
7      input [31:0] instruction,
8
9      // Enable Signal for the CON Flip Flop
10     input con_enable,
11
12     // Q Output Signal from the CON Flip Flop
13     output con_output);
14
15     reg [3:0] decoder_out;
16     reg FF_Output;
17     assign con_output = FF_Output;
18
19     always @ (instruction[20:19]) begin
20         case (instruction[20:19])
21             2'b00: begin
22                 decoder_out = 4'b0001;
23             end
24
25             2'b01: begin
26                 decoder_out = 4'b0010;
27             end
28
29             2'b10: begin
30                 decoder_out = 4'b0100;
31             end
32
33             2'b11: begin
34                 decoder_out = 4'b1000;
35             end
36         endcase
37     end
38
39     always @ (bus_Data && con_enable) begin
40         if (bus_Data == 0 && decoder_out[0]) begin
41             FF_Output = 1;
42         end else if (bus_Data != 0 && decoder_out[1]) begin
43             FF_Output = 1;
44         end else if (bus_Data >= 0 && decoder_out[2]) begin
45             FF_Output = 1;
46         end else if (bus_Data[31] && decoder_out[3]) begin
47             FF_Output = 1;
48         end else begin
49             FF_Output = 0;
50         end
51     end
52
53 endmodule
```

Control Unit

```
hdl > control_unit > V control_unit.v
1  `timescale 1ns/10ps
2  module control_unit(
3      // Control Unit Inputs
4      input wire clk, reset, con_output,
5      input wire [31:0] IR_Data,
6
7      // ALU Opcode
8      output reg [4:0] alu_instruction,
9
10     // RAM Read/Write Signals
11     output reg read, write,
12
13     // Select Encode Logic Signals
14     output reg Gra, Grb, Grc, r_enable, r_select, ba_select,
15
16     // Control Unit Enable Signal Outputs
17     output reg PC_enable, PC_increment_enable, IR_enable, con_enable, Y_enable, Z_enable, HI_enable, LO_enable,
18     MAR_enable, MDR_enable, output_enable, manual_R15_enable,
19
20     // Control Unit Select Signal Outputs
21     output reg MDR_select, Z_HI_select, Z_LO_select, HI_select, LO_select, PC_select, inport_select, c_select);
22
23 parameter
24 // Initial State
25 reset_state = 8'b00000000,
26
27 // Instruction Fetch
28 fetch0 = 8'b00000001, fetch1 = 8'b00000010, fetch2= 8'b00000011,
29
30 // Add Instruction
31 add3 = 8'b00000100, add4= 8'b00000101, add5= 8'b00000110,
32
33 // Sub Instruction
34 sub3 = 8'b00000111, sub4 = 8'b00001000, sub5 = 8'b00001001,
35
36 // Multiply Instruction
37 mul3 = 8'b00001010, mul4 = 8'b00001011, mul5 = 8'b00001100, mul6 = 8'b00001101,
38
39 // Divide Instruction
40 div3 = 8'b00001110, div4 = 8'b00001111, div5 = 8'b00010000, div6 = 8'b00010001,
41
42 // Or Instruction
43 or3 = 8'b00010010, or4 = 8'b00010011, or5 = 8'b00010100, and3 = 8'b00010101,
44
45 // And Instruction
46 and4 = 8'b00010110, and5 = 8'b00010111,
47
```




```
hdl > control_unit > V control_unit.v
```

```
47
48 // Shift Left Instruction
49 shl3 = 8'b00011000, shl4 = 8'b00011001, shl5 = 8'b00011010,
50
51 // Shift Right Instruction
52 shr3 = 8'b00011011, shr4 = 8'b00011100, shr5 = 8'b00011101,
53
54 // Rotate Left Instruction
55 rol3 = 8'b00011110, rol4 = 8'b00011111, rol5 = 8'b00100000,
56
57 // Rotate Right Instruction
58 ror3 = 8'b00100001, ror4 = 8'b00100010, ror5 = 8'b00100011,
59
60 // Negate Instruction
61 neg3 = 8'b00100100, neg4 = 8'b00100101, neg5 = 8'b00100110,
62
63 // Not Instruction
64 not3 = 8'b00100111, not4 = 8'b00101000, not5 = 8'b00101001,
65
66 // Load Instruction
67 ld3 = 8'b00101010, ld4 = 8'b00101011, ld5 = 8'b00101100, ld6 = 8'b00101101, ld7 = 8'b00101110,
68
69 // Load Immediate Instruction
70 ldi3 = 8'b00101111, ldi4 = 8'b00110000, ldi5 = 8'b00110001,
71
72 // Store Instruction
73 st3 = 8'b00110010, st4 = 8'b00110011, st5 = 8'b00110100, st6 = 8'b00110101,
74
75 // Add Immediate Instruction
76 addi3 = 8'b00110111, addi4 = 8'b00111000, addi5 = 8'b00111001,
77
78 // And Immediate Instruction
79 andi3 = 8'b00111010, andi4 = 8'b00111011, andi5 = 8'b00111100,
80
81 // Or Immediate Instruction
82 ori3 = 8'b00111101, ori4 = 8'b00111110, ori5 = 8'b00111111,
83
84 // Branch Instruction
85 br3 = 8'b01000000, br4 = 8'b01000001, br5 = 8'b01000010, br6 = 8'b01000011,
86
87 // Jump Instructions
88 jr3 = 8'b01000100,
89
90 jal3 = 8'b01000101, jal4 = 8'b01000110,
91
92 // Move from LO/HI Instruction
93 mfhi3 = 8'b01000111, mflo3 = 8'b01001000,
94
```

```
hdl > control_unit > V control_unit.v
```


```
95 // In/Out Port Instruction
96 in3 = 8'b01001001, out3 = 8'b01001010,
97
98 // No Instruction
99 nop3 = 8'b01001011, halt3 = 8'b01001100,
100
101 // Shift Right Arithmetic Instruction
102 shra3 = 8'b01001101, shra4 = 8'b01001110, shra5 = 8'b01001111;
103
104 parameter ld = 5'b00000, ldi = 5'b00001, st = 5'b00010,
105 add = 5'b00011, sub = 5'b00100, AND = 5'b00101, OR = 5'b00110,
106 shr = 5'b00111, shra = 5'b01000, shl = 5'b01001, ror = 5'b01010, rol = 5'b01011,
107 mul = 5'b01111, div = 5'b10000, neg = 5'b10001, NOT = 5'b10010,
108 addi = 5'b01100, andi = 5'b01101, ori = 5'b01110,
109 br = 5'b10011, jr = 5'b10100, jal = 5'b10101,
110 in = 5'b10110, out = 5'b10111, mfhi = 5'b11000, mflo = 5'b11001,
111 nop = 5'b11010, halt = 5'b11011;
112
113 reg [7:0] present_state = reset_state; // adjust the bit pattern based on the number of states
114
115 always @(posedge clk, posedge reset) //con_ff finite state machine; if clk or reset rising-edge
116 begin
117     if (reset == 1'b1) present_state = reset_state;
118     else case (present_state)
119         reset_state : present_state = fetch0;
120         fetch0 : present_state = fetch1;
121         fetch1 : present_state = fetch2;
122         fetch2 : begin
123             case (IR_Data[31:27]) // inst. decoding based on the opcode to set the next state
124                 5'b00011 : present_state = add3;
125                 5'b00100 : present_state = sub3;
126                 5'b00111 : present_state = mul3;
127                 5'b10000 : present_state = div3;
128                 5'b00111 : present_state = shr3;
129                 5'b01000 : present_state = shra3;
130                 5'b01001 : present_state = shl3;
131                 5'b01010 : present_state = ror3;
132                 5'b01011 : present_state = rol3;
133                 5'b00101 : present_state = and3;
134                 5'b00110 : present_state = or3;
135                 5'b10001 : present_state = neg3;
136                 5'b10010 : present_state = not3;
137                 5'b00000 : present_state = ld3;
138                 5'b00001 : present_state = ldi3;
139                 5'b00010 : present_state = st3;
140                 5'b01100 : present_state = addi3;
141                 5'b01100 : present_state = andi3;
142                 5'b01110 : present_state = ori3;
143                 5'b10011 : present_state = br3;
```

hdl > control_unit >  control_unit.v

```
144         5'b10100 : present_state = jr3;
145         5'b10101 : present_state = jal3;
146         5'b11000 : present_state = mfhi3;
147         5'b11001 : present_state = mflo3;
148         5'b10110 : present_state = in3;
149         5'b10111 : present_state = out3;
150         5'b11010 : present_state = nop3;
151         5'b11011 : present_state = halt3;
152     endcase
153 end
154
155     add3 : present_state = add4;
156     add4 : present_state = add5;
157     add5 : present_state = fetch0;
158
159     addi3 : present_state = addi4;
160     addi4 : present_state = addi5;
161     addi5 : present_state = fetch0;
162
163     sub3 : present_state = sub4;
164     sub4 : present_state = sub5;
165     sub5 : present_state = fetch0;
166
167     mul3 : present_state = mul4;
168     mul4 : present_state = mul5;
169     mul5 : present_state = mul6;
170     mul6 : present_state = fetch0;
171
172     div3 : present_state = div4;
173     div4 : present_state = div5;
174     div5 : present_state = div6;
175     div6 : present_state = fetch0;
176
177     or3 : present_state = or4;
178     or4 : present_state = or5;
179     or5 : present_state = fetch0;
180
181     and3 : present_state = and4;
182     and4 : present_state = and5;
183     and5 : present_state = fetch0;
184
185     shl3 : present_state = shl4;
186     shl4 : present_state = shl5;
187     shl5 : present_state = fetch0;
188
189     shr3 : present_state = shr4;
190     shr4 : present_state = shr5;
191     shr5 : present_state = fetch0;
192
```

hdl > control_unit > V control_unit.v

```
192
193     shra3 : present_state = shra4;
194     shra4 : present_state = shra5;
195     shra5 : present_state = fetch0;
196
197     rol3 : present_state = rol4;
198     rol4 : present_state = rol5;
199     rol5 : present_state = fetch0;
200
201     ror3 : present_state = ror4;
202     ror4 : present_state = ror5;
203     ror5 : present_state = fetch0;
204
205     neg3 : present_state = neg4;
206     neg4 : present_state = fetch0;
207
208     not3 : present_state = not4;
209     not4 : present_state = fetch0;
210
211     ld3 : present_state = ld4;
212     ld4 : present_state = ld5;
213     ld5 : present_state = ld6;
214     ld6 : present_state = ld7;
215     ld7 : present_state = fetch0;
216
217     ldi3 : present_state = ldi4;
218     ldi4 : present_state = ldi5;
219     ldi5 : present_state = fetch0;
220
221     st3 : present_state = st4;
222     st4 : present_state = st5;
223     st5 : present_state = st6;
224     st6 : present_state = fetch0;
225
226     andi3 : present_state = andi4;
227     andi4 : present_state = andi5;
228     andi5 : present_state = fetch0;
229
230     ori3 : present_state = ori4;
231     ori4 : present_state = ori5;
232     ori5 : present_state = fetch0;
233
234     jal3 : present_state = jal4;
235     jal4 : present_state = fetch0;
236
237     jr3 : present_state = fetch0;
238
```

hdl > control_unit >  control_unit.v

```
238
239     br3 : present_state = br4;
240     br4 : present_state = br5;
241     br5 : present_state = br6;
242     br6 : present_state = fetch0;
243
244     out3 : present_state = fetch0;
245
246     in3 : present_state = fetch0;
247
248     mflo3 : present_state = fetch0;
249
250     mfhi3 : present_state = fetch0;
251
252     nop3 : present_state = fetch0;
253 endcase
254 end
255
256 always @(present_state) // do the job for each state
257 begin
258     case (present_state) // assert the required signals in each state
259     reset_state: begin
260         // RAM Read/Write Signals
261         read <= 0; write <= 0;
262
263         // Select Encode Logic Signals
264         Gra <= 0; Grb <= 0; Grc <= 0; r_enable <= 0; r_select <= 0; ba_select <= 0;
265
266         // Control Unit Enable Signal Outputs
267         PC_enable <= 0; PC_increment_enable <= 0; IR_enable <= 0; con_enable <= 0; Y_enable <= 0;
268         Z_enable <= 0; HI_enable <= 0; LO_enable <= 0; MAR_enable <= 0; MDR_enable <= 0; outport_enable <= 0;
269         manual_R15_enable <= 0;
270
271         // Control Unit Select Signal Outputs
272         MDR_select <= 0; Z_HI_select <= 0; Z_LO_select <= 0; HI_select <= 0; LO_select <= 0; PC_select <= 0;
273         inport_select <= 0; c_select <= 0;
274
275         // ALU Instruction
276         alu_instruction <= 0;
277
278     end
279
280     fetch0: begin
281         #10 PC_select <= 1; MAR_enable <= 1;
282         #75 PC_select <= 0; MAR_enable <= 0;
283     end
284 end
```

hdl > control_unit > **V** control_unit.v

```
285     fetch1: begin
286         #10 PC_increment_enable <= 1; read <= 1; MDR_enable <= 1;
287         #75 PC_increment_enable <= 0; read <= 0; MDR_enable <= 0;
288     end
289
290     fetch2: begin
291         #10 MDR_select <= 1; IR_enable <= 1;
292         #75 MDR_select <= 0; IR_enable <= 0;
293     end
294
295     // Addition Operation
296     add3: begin
297         #10 Grb <= 1; r_select <= 1; Y_enable <= 1;
298         #75 Grb <= 0; r_select <= 0; Y_enable <= 0;
299     end
300
301     add4: begin
302         #10 Grc <= 1; r_select <= 1; alu_instruction <= add; Z_enable <= 1;
303         #75 Grc <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
304     end
305
306     add5: begin
307         #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
308         #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
309     end
310
311     // Subtraction Operation
312     sub3: begin
313         #10 Grb <= 1; r_select <= 1; Y_enable <= 1;
314         #75 Grb <= 0; r_select <= 0; Y_enable <= 0;
315     end
316
317     sub4: begin
318         #10 Grc <= 1; r_select <= 1; alu_instruction <= sub; Z_enable <= 1;
319         #75 Grc <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
320     end
321
322     sub5: begin
323         #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
324         #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
325     end
326
327     // Or Operation
328     or3: begin
329         #10 Grb <= 1; r_select <= 1; Y_enable <= 1;
330         #75 Grb <= 0; r_select <= 0; Y_enable <= 0;
331     end
332
```

hdl > control_unit > **V** control_unit.v

```
333     or4: begin
334         #10 Grc <= 1; r_select <= 1; alu_instruction <= OR; Z_enable <= 1;
335         #75 Grc <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
336     end
337
338     or5: begin
339         #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
340         #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
341     end
342
343     // And Operation
344     and3: begin
345         #10 Grb <= 1; r_select <= 1; Y_enable <= 1;
346         #75 Grb <= 0; r_select <= 0; Y_enable <= 0;
347     end
348
349     and4: begin
350         #10 Grc <= 1; r_select <= 1; alu_instruction <= AND; Z_enable <= 1;
351         #75 Grc <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
352     end
353
354     and5: begin
355         #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
356         #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
357     end
358
359     // SHR Operation
360     shr3: begin
361         #10 Grb <= 1; r_select <= 1; Y_enable <= 1;
362         #75 Grb <= 0; r_select <= 0; Y_enable <= 0;
363     end
364
365     shr4: begin
366         #10 Grc <= 1; r_select <= 1; alu_instruction <= shr; Z_enable <= 1;
367         #75 Grc <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
368     end
369
370     shr5: begin
371         #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
372         #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
373     end
374
375     // SHL Operation
376     shl3: begin
377         #10 Grb <= 1; r_select <= 1; Y_enable <= 1;
378         #75 Grb <= 0; r_select <= 0; Y_enable <= 0;
379     end
380
```


```

381 shl4: begin
382     #10 Grc <= 1; r_select <= 1; alu_instruction <= shl; Z_enable <= 1;
383     #75 Grc <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
384 end
385
386 shl5: begin
387     #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
388     #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
389 end
390
391 // shra Operation
392 shra3: begin
393     #10 Grb <= 1; r_select <= 1; Y_enable <= 1;
394     #75 Grb <= 0; r_select <= 0; Y_enable <= 0;
395 end
396
397 shra4: begin
398     #10 Grc <= 1; r_select <= 1; alu_instruction <= shra; Z_enable <= 1;
399     #75 Grc <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
400 end
401
402 shra5: begin
403     #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
404     #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
405 end
406
407 // ror Operation
408 ror3: begin
409     #10 Grb <= 1; r_select <= 1; Y_enable <= 1;
410     #75 Grb <= 0; r_select <= 0; Y_enable <= 0;
411 end
412
413 ror4: begin
414     #10 Grc <= 1; r_select <= 1; alu_instruction <= ror; Z_enable <= 1;
415     #75 Grc <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
416 end
417
418 ror5: begin
419     #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
420     #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
421 end
422
423 // rol Operation
424 rol3: begin
425     #10 Grb <= 1; r_select <= 1; Y_enable <= 1;
426     #75 Grb <= 0; r_select <= 0; Y_enable <= 0;
427 end
428

```


hdl > control_unit > V control_unit.v

```
428
429 ~   rol4: begin
430     #10 Grc <= 1; r_select <= 1; alu_instruction <= rol; Z_enable <= 1;
431     #75 Grc <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
432 end
433
434 ~   rol5: begin
435     #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
436     #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
437 end
438
439 ~   // Multiplication Operation
440 ~   mul3: begin
441     #10 Grb <= 1; r_select <= 1; Y_enable <= 1;
442     #75 Grb <= 0; r_select <= 0; Y_enable <= 0;
443 end
444
445 ~   mul4: begin
446     #10 Grc <= 1; r_select <= 1; alu_instruction <= mul; Z_enable <= 1;
447     #75 Grc <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
448 end
449
450 ~   mul5: begin
451     #10 Z_LO_select <= 1; LO_enable <= 1;
452     #75 Z_LO_select <= 0; LO_enable <= 0;
453 end
454
455 ~   mul6: begin
456     #10 Z_HI_select <= 1; HI_enable <= 1;
457     #75 Z_HI_select <= 0; HI_enable <= 0;
458 end
459
460   // Division Operation
461 ~   div3: begin
462     #10 Grb <= 1; r_select <= 1; Y_enable <= 1;
463     #75 Grb <= 0; r_select <= 0; Y_enable <= 0;
464 end
465
466 ~   div4: begin
467     #10 Grc <= 1; r_select <= 1; alu_instruction <= div; Z_enable <= 1;
468     #75 Grc <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
469 end
470
471 ~   div5: begin
472     #10 Z_LO_select <= 1; LO_enable <= 1;
473     #75 Z_LO_select <= 0; LO_enable <= 0;
474 end
475
```

hdl > control_unit >  control_unit.v

```
476     div6: begin
477         #10 Z_HI_select <= 1; HI_enable <= 1;
478         #75 Z_HI_select <= 0; HI_enable <= 0;
479     end
480
481     // Not Operation
482     not3: begin
483         #10 Grb <= 1; r_select <= 1; alu_instruction <= NOT; Y_enable <= 1;
484         #75 Grb <= 0; r_select <= 0; alu_instruction <= 0; Y_enable <= 0;
485     end
486
487     not4: begin
488         #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
489         #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
490     end
491
492     // Negate Operation
493     neg3: begin
494         #10 Grb <= 1; r_select <= 1; alu_instruction <= neg; Y_enable <= 1;
495         #75 Grb <= 0; r_select <= 0; alu_instruction <= 0; Y_enable <= 0;
496     end
497
498     neg4: begin
499         #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
500         #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
501     end
502
503     // And Immediate Operation
504     andi3: begin
505         #10 c_select <= 1; Y_enable <= 1;
506         #75 c_select <= 0; Y_enable <= 0;
507     end
508
509     andi4: begin
510         #10 Grb <= 1; r_select <= 1; alu_instruction <= AND; Z_enable <= 1;
511         #75 Grb <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
512     end
513
514     andi5: begin
515         #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
516         #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
517     end
518
519     // Add Immediate Operation
520     addi3: begin
521         #10 c_select <= 1; Y_enable <= 1;
522         #75 c_select <= 0; Y_enable <= 0;
523     end
```

hdl > control_unit > V control_unit.v

```
525     addi4: begin
526         #10 Grb <= 1; r_select <= 1; alu_instruction <= add; Z_enable <= 1;
527         #75 Grb <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
528     end
529
530     addi5: begin
531         #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
532         #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
533     end
534
535     // Or Immediate Operation
536     ori3: begin
537         #10 c_select <= 1; Y_enable <= 1;
538         #75 c_select <= 0; Y_enable <= 0;
539     end
540
541     ori4: begin
542         #10 Grb <= 1; r_select <= 1; alu_instruction <= OR; Z_enable <= 1;
543         #75 Grb <= 0; r_select <= 0; alu_instruction <= 0; Z_enable <= 0;
544     end
545
546     ori5: begin
547         #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
548         #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
549     end
550
551     // Load Operation
552     ld3: begin
553         #10 Grb <= 1; ba_select <= 1; Y_enable <= 1;
554         #75 Grb <= 0; ba_select <= 0; Y_enable <= 0;
555     end
556
557     ld4: begin
558         #10 c_select <= 1; alu_instruction <= add; Z_enable <= 1;
559         #75 c_select <= 0; alu_instruction <= 0; Z_enable <= 0;
560     end
561
562     ld5: begin
563         #10 Z_LO_select <= 1; MAR_enable <= 1;
564         #75 Z_LO_select <= 0; MAR_enable <= 0;
565     end
566
567     ld6: begin
568         #10 read <= 1; MDR_enable <= 1;
569         #75 read <= 0; MDR_enable <= 0;
570     end
571
```

hdl > control_unit > **V** control_unit.v

```
572     ld7: begin
573         #10 MDR_select <= 1; Gra <= 1; r_enable <= 1;
574         #75 MDR_select <= 0; Gra <= 0; r_enable <= 0;
575     end
576
577     // Load Immediate Operation
578     ldi3: begin
579         #10 Grb <= 1; ba_select <= 1; Y_enable <= 1;
580         #75 Grb <= 0; ba_select <= 0; Y_enable <= 0;
581     end
582
583     ldi4: begin
584         #10 c_select <= 1; alu_instruction <= add; Z_enable <= 1;
585         #75 c_select <= 0; alu_instruction <= 0; Z_enable <= 0;
586     end
587
588     ldi5: begin
589         #10 Z_LO_select <= 1; Gra <= 1; r_enable <= 1;
590         #75 Z_LO_select <= 0; Gra <= 0; r_enable <= 0;
591     end
592
593     //Store Operation
594     st3: begin
595         #10 Grb <= 1; ba_select <= 1; Y_enable <= 1;
596         #75 Grb <= 0; ba_select <= 0; Y_enable <= 0;
597     end
598
599     st4: begin
600         #10 c_select <= 1; alu_instruction <= add; Z_enable <= 1;
601         #75 c_select <= 0; alu_instruction <= 0; Z_enable <= 0;
602     end
603
604     st5: begin
605         #10 Z_LO_select <= 1; MAR_enable <= 1;
606         #75 Z_LO_select <= 0; MAR_enable <= 0;
607     end
608
609     st6: begin
610         #10 write <= 1; MDR_enable <= 1; Gra <= 1; r_select <= 1;
611         #75 write <= 0; MDR_enable <= 0; Gra <= 0; r_select <= 0;
612     end
613
614     // Jump Register Operation
615     jr3: begin
616         #10 Gra <= 1; r_select <= 1; PC_enable <= 1;
617         #75 Gra <= 0; r_select <= 0; PC_enable <= 0;
618     end
619
```

```

620 //Jump and Link Operation
621 jal3: begin
622     #10 manual_R15_enable <= 1; PC_select <= 1;
623     #75 manual_R15_enable <= 0; PC_select <= 0;
624 end
625
626 jal4: begin
627     #10 Gra <= 1; r_select <= 1; PC_enable <= 1;
628     #75 Gra <= 0; r_select <= 0; PC_enable <= 0;
629 end
630
631 // Move from Hi Register Operation
632 mfhi3: begin
633     #10 Gra <= 1; r_enable <= 1; HI_select <= 1;
634     #75 Gra <= 0; r_enable <= 0; HI_select <= 0;
635 end
636
637 // Move from Lo Register Operation
638 mflo3: begin
639     #10 Gra <= 1; r_enable <= 1; LO_select <= 1;
640     #75 Gra <= 0; r_enable <= 0; LO_select <= 0;
641 end
642
643 // Inputting Operation
644 in3: begin
645     #10 Gra <= 1; r_enable <= 1; inport_select <= 1;
646     #75 Gra <= 0; r_enable <= 0; inport_select <= 0;
647 end
648
649 // Outputting Operation
650 out3: begin
651     #10 Gra <= 1; r_select <= 1; outport_enable <= 1;
652     #75 Gra <= 0; r_select <= 0; outport_enable <= 0;
653 end
654
655 // Branch Operation
656 br3: begin
657     #10 Gra <= 1; r_select <= 1; con_enable <= 1;
658     #75 Gra <= 0; r_select <= 0; con_enable <= 0;
659 end
660
661 br4: begin
662     #10 PC_select <= 1; Y_enable <= 1;
663     #75 PC_select <= 0; Y_enable <= 0;
664 end
665

```

```

666     br5: begin
667         #10 c_select <= 1; alu_instruction <= add; Z_enable <= 1;
668         #75 c_select <= 0; alu_instruction <= 0; Z_enable <= 0;
669     end
670
671     br6: begin
672         #10 Z_LO_select <= 1; PC_enable <= con_output;
673         #75 Z_LO_select <= 0; PC_enable <= 0;
674     end
675
676     // No Operation
677     nop3: begin
678         MDR_select <= 0; IR_enable <= 0;
679     end
680
681     // Halt Operation (Stops instruction execution)
682     halt3: begin
683         MDR_select <= 0; IR_enable <= 0;
684     end
685
686     //Any other input would be default to do nothing
687     default: begin
688     end
689 endcase
690 end
691 endmodule
692

```

Memory Subsystem

Ram

```
hdl > memory_subsystem > V ram.v
1  v module ram(
2      input clk,          // RAM enable control signal (active high)
3      input read,         // Read control signal (active high)
4      input write,        // Write control signal (active high)
5      input [31:0] data_in, // Input data
6      input [8:0] address_in, // Address input
7      output wire [31:0] data_out, // Output data
8
9      output [31:0] debug_port_01,
10     output [31:0] debug_port_02
11 );
12
13 reg [31:0] mem [511:0]; // Memory array
14 reg [31:0] tempData;    // Temporary data storage
15
16 v initial begin
17     $readmemh("ram.mif", mem);
18 end
19
20 v always @(posedge clk) begin
21     v if (write) begin
22         mem[address_in] <= data_in; // Write data to memory location
23     end
24     v if (read) begin
25         tempData <= mem[address_in]; // Read data from memory location
26     end
27 end
28
29 assign data_out = tempData;
30
31 assign debug_port_01 = mem [144];
32 assign debug_port_02 = mem [247];
33
34 endmodule
35
```

Ram.mif File

| | | | | | | | |
|----|----------|-----|----------|-----|----------|-----|----------|
| 1 | 08800002 | 43 | ffffffff | 101 | ffffffff | 160 | ffffffff |
| 2 | 08080000 | 44 | ffffffff | 102 | ffffffff | 161 | ffffffff |
| 3 | 01000068 | 45 | ffffffff | 103 | ffffffff | 162 | ffffffff |
| 4 | 0917FFFC | 46 | ffffffff | 104 | ffffffff | 163 | ffffffff |
| 5 | 00900001 | 47 | ffffffff | 105 | 00000055 | 164 | ffffffff |
| 6 | 09800069 | 48 | ffffffff | 106 | ffffffff | 165 | ffffffff |
| 7 | 99980004 | 49 | ffffffff | 107 | ffffffff | 166 | ffffffff |
| 8 | 09980002 | 50 | ffffffff | 108 | ffffffff | 167 | ffffffff |
| 9 | 039FFFFD | 51 | ffffffff | 109 | ffffffff | 168 | ffffffff |
| 10 | D0000000 | 52 | ffffffff | 110 | ffffffff | 169 | ffffffff |
| 11 | 9B900002 | 53 | ffffffff | 111 | ffffffff | 170 | ffffffff |
| 12 | 09000005 | 54 | ffffffff | 112 | ffffffff | 171 | ffffffff |
| 13 | 09880002 | 55 | ffffffff | 113 | ffffffff | 172 | ffffffff |
| 14 | 19918000 | 56 | ffffffff | 114 | ffffffff | 173 | ffffffff |
| 15 | 63B80002 | 57 | ffffffff | 115 | ffffffff | 174 | ffffffff |
| 16 | 8BB80000 | 58 | ffffffff | 116 | ffffffff | 175 | ffffffff |
| 17 | 93B80000 | 59 | ffffffff | 117 | ffffffff | 176 | ffffffff |
| 18 | 6BB8000F | 60 | ffffffff | 118 | ffffffff | 177 | ffffffff |
| 19 | 50880000 | 61 | ffffffff | 119 | ffffffff | 178 | ffffffff |
| 20 | 7388001C | 62 | ffffffff | 120 | ffffffff | 179 | ffffffff |
| 21 | 43B80000 | 63 | ffffffff | 121 | ffffffff | 180 | ffffffff |
| 22 | 39180000 | 64 | ffffffff | 122 | ffffffff | 181 | ffffffff |
| 23 | 11000052 | 65 | ffffffff | 123 | ffffffff | 182 | ffffffff |
| 24 | 59100000 | 66 | ffffffff | 124 | ffffffff | 183 | ffffffff |
| 25 | 31180000 | 67 | ffffffff | 125 | ffffffff | 184 | ffffffff |
| 26 | 28908000 | 68 | ffffffff | 126 | ffffffff | 185 | ffffffff |
| 27 | 11880060 | 69 | ffffffff | 127 | ffffffff | 186 | ffffffff |
| 28 | 21918000 | 70 | ffffffff | 128 | ffffffff | 187 | ffffffff |
| 29 | 48900000 | 71 | ffffffff | 129 | ffffffff | 188 | ffffffff |
| 30 | 0A000006 | 72 | ffffffff | 130 | ffffffff | 189 | ffffffff |
| 31 | 0A800032 | 73 | ffffffff | 131 | ffffffff | 190 | ffffffff |
| 32 | 7AA00000 | 74 | ffffffff | 132 | ffffffff | 191 | ffffffff |
| 33 | C3800000 | 75 | ffffffff | 133 | ffffffff | 192 | ffffffff |
| 34 | CB000000 | 76 | ffffffff | 134 | ffffffff | 193 | ffffffff |
| 35 | 82A00000 | 77 | ffffffff | 135 | ffffffff | 194 | ffffffff |
| 36 | 0C27FFFF | 78 | ffffffff | 136 | ffffffff | 195 | ffffffff |
| 37 | 0CAFFFD | 79 | ffffffff | 137 | ffffffff | 196 | ffffffff |
| 38 | 0D300000 | 80 | ffffffff | 138 | ffffffff | 197 | ffffffff |
| 39 | 0DB80000 | 81 | ffffffff | 139 | ffffffff | 198 | ffffffff |
| 40 | AD000000 | 82 | ffffffff | 140 | ffffffff | 199 | ffffffff |
| 41 | D8000000 | 83 | 00000026 | 141 | ffffffff | 200 | ffffffff |
| 42 | ffffffff | 84 | ffffffff | 142 | ffffffff | 201 | ffffffff |
| | | 85 | ffffffff | 143 | ffffffff | 202 | ffffffff |
| | | 86 | ffffffff | 144 | ffffffff | 203 | ffffffff |
| | | 87 | ffffffff | 145 | ffffffff | 204 | ffffffff |
| | | 88 | ffffffff | 146 | ffffffff | 205 | ffffffff |
| | | 89 | ffffffff | 147 | ffffffff | 206 | ffffffff |
| | | 90 | ffffffff | 148 | ffffffff | 207 | ffffffff |
| | | 91 | ffffffff | 149 | ffffffff | 208 | ffffffff |
| | | 92 | ffffffff | 150 | ffffffff | 209 | ffffffff |
| | | 93 | ffffffff | 151 | ffffffff | 210 | ffffffff |
| | | 94 | ffffffff | 152 | ffffffff | 211 | ffffffff |
| | | 95 | ffffffff | 153 | ffffffff | 212 | ffffffff |
| | | 96 | ffffffff | 154 | ffffffff | 213 | ffffffff |
| | | 97 | ffffffff | 155 | ffffffff | 214 | ffffffff |
| | | 98 | ffffffff | 156 | ffffffff | 215 | ffffffff |
| | | 99 | ffffffff | 157 | ffffffff | 216 | ffffffff |
| | | 100 | ffffffff | 158 | ffffffff | 217 | ffffffff |
| | | | | 159 | ffffffff | | |

| | | | | | | | |
|-----|----------|-----|----------|-----|----------|-----|----------|
| 218 | ffffffff | 277 | ffffffff | 335 | ffffffff | 392 | ffffffff |
| 219 | ffffffff | 278 | ffffffff | 336 | ffffffff | 393 | ffffffff |
| 220 | ffffffff | 279 | ffffffff | 337 | ffffffff | 394 | ffffffff |
| 221 | ffffffff | 280 | ffffffff | 338 | ffffffff | 395 | ffffffff |
| 222 | ffffffff | 281 | ffffffff | 339 | ffffffff | 396 | ffffffff |
| 223 | ffffffff | 282 | ffffffff | 340 | ffffffff | 397 | ffffffff |
| 224 | ffffffff | 283 | ffffffff | 341 | ffffffff | 398 | ffffffff |
| 225 | ffffffff | 284 | ffffffff | 342 | ffffffff | 399 | ffffffff |
| 226 | ffffffff | 285 | ffffffff | 343 | ffffffff | 400 | ffffffff |
| 227 | ffffffff | 286 | ffffffff | 344 | ffffffff | 401 | ffffffff |
| 228 | ffffffff | 287 | ffffffff | 345 | ffffffff | 402 | ffffffff |
| 229 | ffffffff | 288 | ffffffff | 346 | ffffffff | 403 | ffffffff |
| 230 | ffffffff | 289 | ffffffff | 347 | ffffffff | 404 | ffffffff |
| 231 | ffffffff | 290 | ffffffff | 348 | ffffffff | 405 | ffffffff |
| 232 | ffffffff | 291 | ffffffff | 349 | ffffffff | 406 | ffffffff |
| 233 | ffffffff | 292 | ffffffff | 350 | ffffffff | 407 | ffffffff |
| 234 | ffffffff | 293 | ffffffff | 351 | ffffffff | 408 | ffffffff |
| 235 | ffffffff | 294 | ffffffff | 352 | ffffffff | 409 | ffffffff |
| 236 | ffffffff | 295 | ffffffff | 353 | ffffffff | 410 | ffffffff |
| 237 | ffffffff | 296 | ffffffff | 354 | ffffffff | 411 | ffffffff |
| 238 | ffffffff | 297 | ffffffff | 355 | ffffffff | 412 | ffffffff |
| 239 | ffffffff | 298 | ffffffff | 356 | ffffffff | 413 | ffffffff |
| 240 | ffffffff | 299 | ffffffff | 357 | ffffffff | 414 | ffffffff |
| 241 | ffffffff | 300 | ffffffff | 358 | ffffffff | 415 | ffffffff |
| 242 | ffffffff | 301 | 1EC50000 | 359 | ffffffff | 416 | ffffffff |
| 243 | ffffffff | 302 | 264D8000 | 360 | ffffffff | 417 | ffffffff |
| 244 | ffffffff | 303 | 26EE0000 | 361 | ffffffff | 418 | ffffffff |
| 245 | ffffffff | 304 | A7800000 | 362 | ffffffff | 419 | ffffffff |
| 246 | ffffffff | 305 | ffffffff | 363 | ffffffff | 420 | ffffffff |
| 247 | ffffffff | 306 | ffffffff | 364 | ffffffff | 421 | ffffffff |
| 248 | ffffffff | 307 | ffffffff | 365 | ffffffff | 422 | ffffffff |
| 249 | ffffffff | 308 | ffffffff | 366 | ffffffff | 423 | ffffffff |
| 250 | ffffffff | 309 | ffffffff | 367 | ffffffff | 424 | ffffffff |
| 251 | ffffffff | 310 | ffffffff | 368 | ffffffff | 425 | ffffffff |
| 252 | ffffffff | 311 | ffffffff | 369 | ffffffff | 426 | ffffffff |
| 253 | ffffffff | 312 | ffffffff | 370 | ffffffff | 427 | ffffffff |
| 254 | ffffffff | 313 | ffffffff | 371 | ffffffff | 428 | ffffffff |
| 255 | ffffffff | 314 | ffffffff | 372 | ffffffff | 429 | ffffffff |
| 256 | ffffffff | 315 | ffffffff | 373 | ffffffff | 430 | ffffffff |
| 257 | ffffffff | 316 | ffffffff | 374 | ffffffff | 431 | ffffffff |
| 258 | ffffffff | 317 | ffffffff | 375 | ffffffff | 432 | ffffffff |
| 259 | ffffffff | 318 | ffffffff | 376 | ffffffff | 433 | ffffffff |
| 260 | ffffffff | 319 | ffffffff | 377 | ffffffff | 434 | ffffffff |
| 261 | ffffffff | 320 | ffffffff | 378 | ffffffff | 435 | ffffffff |
| 262 | ffffffff | 321 | ffffffff | 379 | ffffffff | 436 | ffffffff |
| 263 | ffffffff | 322 | ffffffff | 380 | ffffffff | 437 | ffffffff |
| 264 | ffffffff | 323 | ffffffff | 381 | ffffffff | 438 | ffffffff |
| 265 | ffffffff | 324 | ffffffff | 382 | ffffffff | 439 | ffffffff |
| 266 | ffffffff | 325 | ffffffff | 383 | ffffffff | 440 | ffffffff |
| 267 | ffffffff | 326 | ffffffff | 384 | ffffffff | 441 | ffffffff |
| 268 | ffffffff | 327 | ffffffff | 385 | ffffffff | 442 | ffffffff |
| 269 | ffffffff | 328 | ffffffff | 386 | ffffffff | 443 | ffffffff |
| 270 | ffffffff | 329 | ffffffff | 387 | ffffffff | 444 | ffffffff |
| 271 | ffffffff | 330 | ffffffff | 388 | ffffffff | 445 | ffffffff |
| 272 | ffffffff | 331 | ffffffff | 389 | ffffffff | 446 | ffffffff |
| 273 | ffffffff | 332 | ffffffff | 390 | ffffffff | 447 | ffffffff |
| 274 | ffffffff | 333 | ffffffff | 391 | ffffffff | 448 | ffffffff |
| 275 | ffffffff | 334 | ffffffff | | | 449 | ffffffff |
| 276 | ffffffff | | | | | | |

| | |
|-----|----------|
| 450 | ffffffff |
| 451 | ffffffff |
| 452 | ffffffff |
| 453 | ffffffff |
| 454 | ffffffff |
| 455 | ffffffff |
| 456 | ffffffff |
| 457 | ffffffff |
| 458 | ffffffff |
| 459 | ffffffff |
| 460 | ffffffff |
| 461 | ffffffff |
| 462 | ffffffff |
| 463 | ffffffff |
| 464 | ffffffff |
| 465 | ffffffff |
| 466 | ffffffff |
| 467 | ffffffff |
| 468 | ffffffff |
| 469 | ffffffff |
| 470 | ffffffff |
| 471 | ffffffff |
| 472 | ffffffff |
| 473 | ffffffff |
| 474 | ffffffff |
| 475 | ffffffff |
| 476 | ffffffff |
| 477 | ffffffff |
| 478 | ffffffff |
| 479 | ffffffff |
| 480 | ffffffff |
| 481 | ffffffff |
| 482 | ffffffff |
| 483 | ffffffff |
| 484 | ffffffff |
| 485 | ffffffff |
| 486 | ffffffff |
| 487 | ffffffff |
| 488 | ffffffff |
| 489 | ffffffff |
| 490 | ffffffff |
| 491 | ffffffff |
| 492 | ffffffff |
| 493 | ffffffff |
| 494 | ffffffff |
| 495 | ffffffff |
| 496 | ffffffff |
| 497 | ffffffff |
| 498 | ffffffff |
| 499 | ffffffff |
| 500 | ffffffff |
| 501 | ffffffff |
| 502 | ffffffff |
| 503 | ffffffff |
| 504 | ffffffff |
| 505 | ffffffff |
| 506 | ffffffff |
| 507 | ffffffff |
| 508 | ffffffff |
| 509 | ffffffff |
| 510 | ffffffff |
| 511 | ffffffff |
| 512 | ffffffff |

Ports

Inport

```
hdl > ports > V inport.v
1  ∨ module inport(
2      input clk, clr,
3      input wire [31:0] input_Data,
4      output wire [31:0] inport_Data);
5
6      reg [31:0] tempData;
7      initial tempData = 32'h0;
8
9      always @(posedge clk)
10 ∨ begin
11     if (clr) tempData <= {32{1'b0}};
12     else tempData <= input_Data;
13 end
14
15 assign inport_Data = tempData[31:0];
16
17 endmodule
18
```

Outport

```
hdl > ports > V output.v
1  v module outport(
2      input clk, clr, enable,
3      input wire [31:0] bus_Data,
4      output wire [31:0] outport_Data);
5
6      reg [31:0] tempData;
7      initial tempData = 32'h0;
8
9      always @(posedge clk)
10     v begin
11         if (clr) tempData <= {32{1'b0}};
12         else if (enable) tempData <= bus_Data;
13     end
14
15     assign outport_Data = tempData[31:0];
16
17     endmodule
```

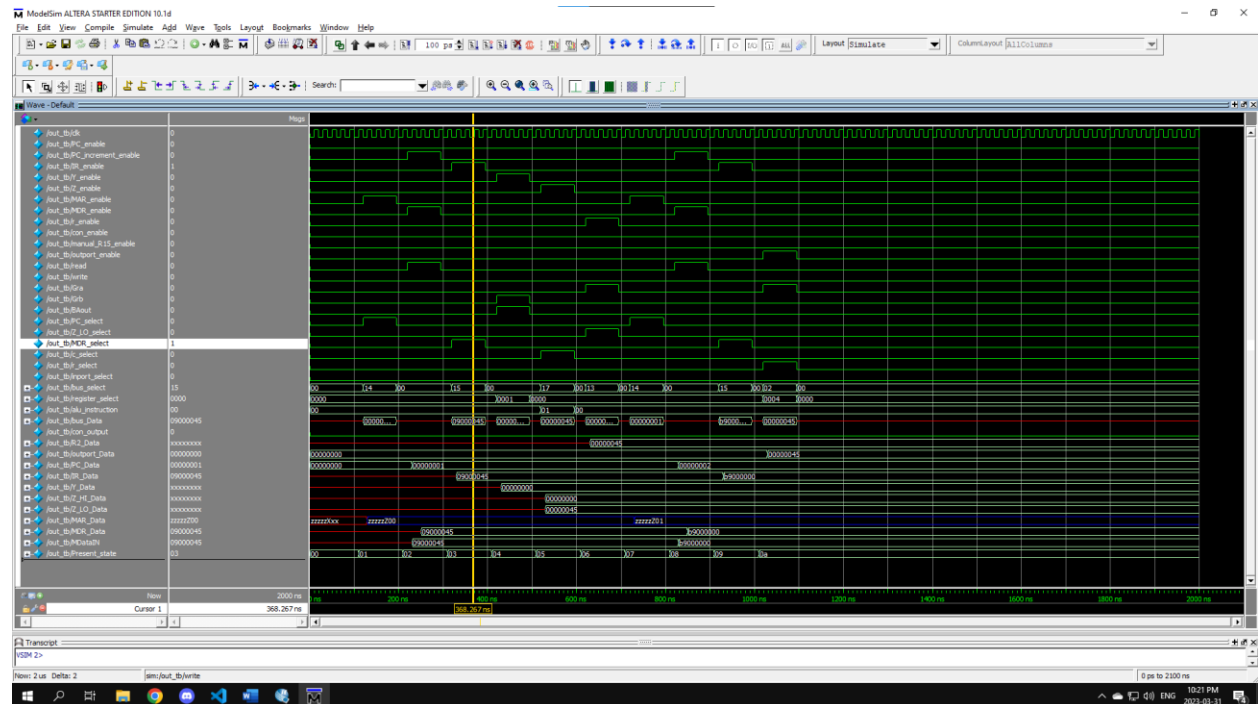
Control Unit Testbench

Our group finished the code for the control unit but since there were no specified instructions, we were not sure how to implement a testbench for this phase. In this section, we used our phase 2 simulations to show that most features required still work.

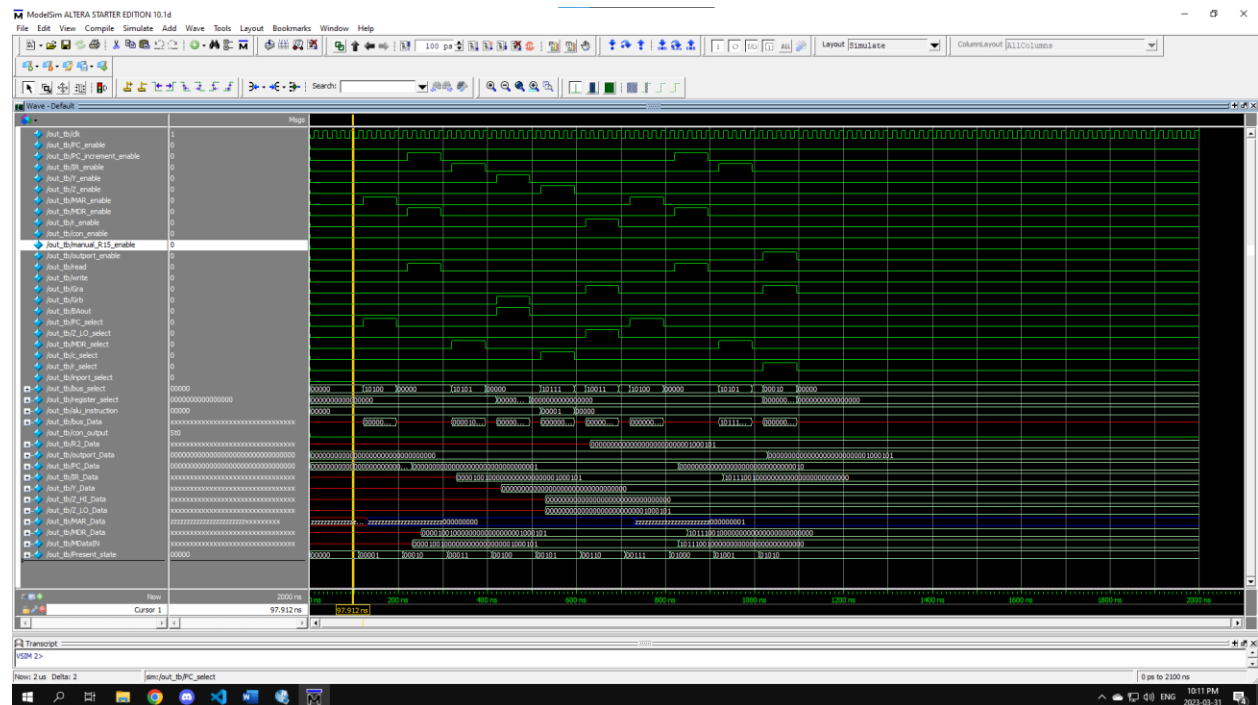
What features actually work:

- IR holds the instruction from the .MIF file
- PC is a dedicated module and it will only increment once every time the increment PC signal is enabled
- PC works with all branch instructions
- All registers can hold data from the bus and the select and encode logic works
- R15_enable is also modified to enable from the select encode logic and another manual enable signal
- Instruction fetch (shown in control unit) and decode work

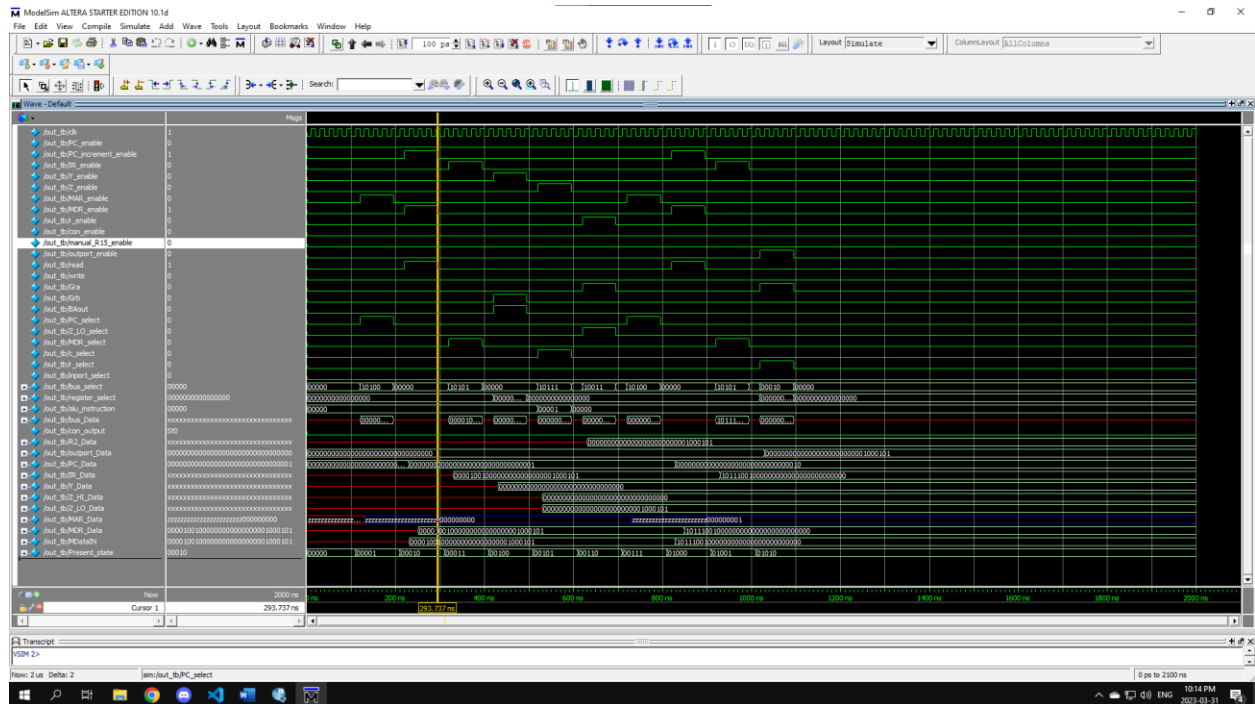
Example of IR Holding the Instruction from Memory:



Example Testbench at the Start:

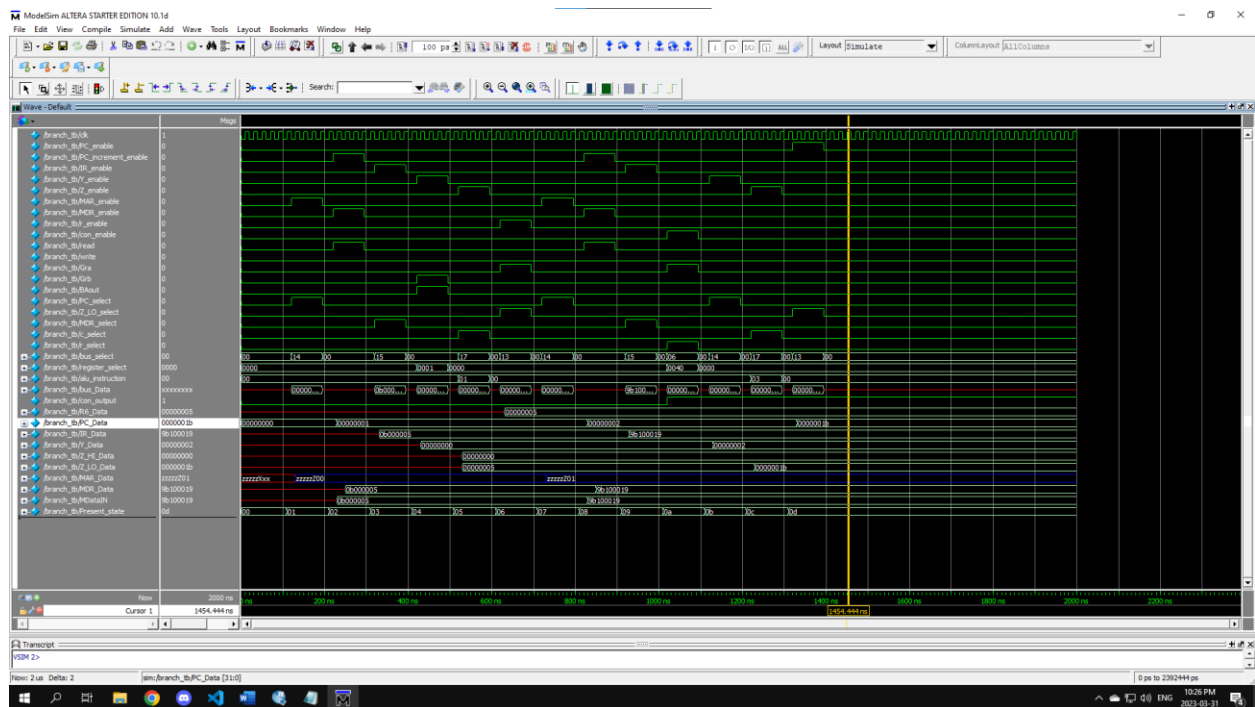


Example Testbench Showing that PC Incremented only 1 time:

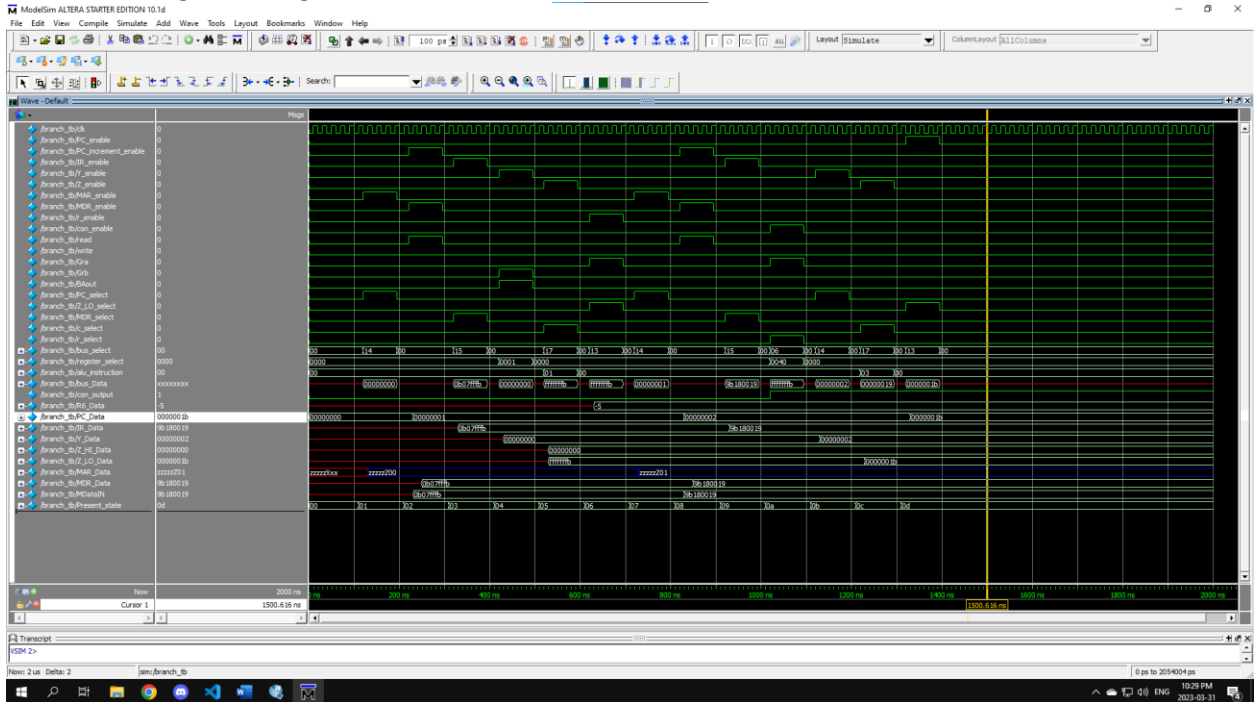


PC Changing for BRMI and BRPL

For BRPL, the general register 6 holds the value 5 and after the instruction, PC holds the value 0x1b.



For BRMI, the general register 6 holds the value -5 and after the instruction, PC holds the value 0x1b.



Instruction Fetch and Decode

```

fetch0: begin
    #10 PC_select <= 1; MAR_enable <= 1;
    #75 PC_select <= 0; MAR_enable <= 0;
end

fetch1: begin
    #10 PC_increment_enable <= 1; read <= 1; MDR_enable <= 1;
    #75 PC_increment_enable <= 0; read <= 0; MDR_enable <= 0;
end

fetch2: begin
    #10 MDR_select <= 1; IR_enable <= 1;
    #75 MDR_select <= 0; IR_enable <= 0;
end

```

```

always @ (instruction, Gra, Grb, Grc) begin
    // Assigning Values to Instruction Register Content Variables
    Ra = instruction [26:23];      // Bit 23 -> 26 of the instruction register is register a
    Rb = instruction [22:19];      // Bit 19 -> 22 of the instruction register is register b
    Rc = instruction [19:15];      // Bit 15 -> 19 of the instruction register is register c

    if (Gra) decoder_input = Ra;
    else if (Grb) decoder_input = Rb;
    else if (Grc) decoder_input = Rc;

```

```

    case (decoder_input)
        4'd0: decoder_out = 16'd1;
        4'd1: decoder_out = 16'd2;
        4'd2: decoder_out = 16'd4;
        4'd3: decoder_out = 16'd8;
        4'd4: decoder_out = 16'd16;
        4'd5: decoder_out = 16'd32;
        4'd6: decoder_out = 16'd64;
        4'd7: decoder_out = 16'd128;
        4'd8: decoder_out = 16'd256;
        4'd9: decoder_out = 16'd512;
        4'd10: decoder_out = 16'd1024;
        4'd11: decoder_out = 16'd2048;
        4'd12: decoder_out = 16'd4096;
        4'd13: decoder_out = 16'd8192;
        4'd14: decoder_out = 16'd16384;
        4'd15: decoder_out = 16'd32768;
    endcase
end

// Assigning Values to the Outputs
assign register_enable = {16{r_enable}} & decoder_out;
assign register_select = ({16{ba_select}} | {16{r_select}}) & decoder_out;
assign C_sign_ext_Data = {{13{instruction[18]}}, instruction[18:0]};

```

The decoder allows the select and encode logic to determine which register needs to be enabled or selected and then the `r_enable`, `r_select`, and `ba_select` signals allow the select and encode logic to output the proper enable and select signals. The `C_sign_extended_Data` variable holds the first 19 bits which is usually the immediate value. For some instructions bits 15-19 can be `Rc`.