

ELEC 374: CPU Design

Final Report

John Turnbull 202355

## Abstract

Computer engineering is a diverse field encompassing many fundamental topics, and computer architecture is one of them. This project involves a complex design process that requires creating a 32-bit machine with sixteen 32-bit registers, R0-R15, HI and LO registers, PC, IR, Y, MAR, MDR, and more. Additionally, there are additional registers for input-output capabilities. The design process involved a busmux, encoder, select-encoder, registers, ALU, arithmetic operators (Add, Mul, etc.), the memory system (RAM), CON-FF, the control unit, and a top-level Datapath module that instantiates and connects all other modules. The project's evaluation is based on several parameters, including the SRC's operational frequency and overall accuracy. Therefore, developing a comprehensive understanding of computer architecture and implementing a functional mini SRC can be a challenging but rewarding experience.

## Table of Contents

ELEC 374: CPU Design .....	1
Final Report.....	1
Abstract.....	2
Project Specification .....	4
Project Design and Implementation .....	4
Phase 1 .....	4
Phase 2 .....	4
Phase 3 .....	4
Evaluation Results.....	5
Discussion.....	5
Conclusion.....	6
Appendix .....	6
Code .....	6
Test Benches .....	21
Simulations.....	30

## Project Specification

The project involves specifying a 32-bit machine with sixteen 32-bit registers (R0-R15), program counter (PC), the instruction register (IR), memory address (MAR) and data registers (MDR), two registers for ALU results (HI and LO), and more. Furthermore, the random-access memory system (RAM) is designed to hold 512 32-bit values. The RAM module enables write capabilities with inputs for the address and data, and an output for memory reads. Two additional registers are allocated for input-output capabilities. Additionally, the ALU performs 13 operations, including addition (ADD), subtraction (SUB), multiplication (MUL), division (DIV), shift right (SHR), shift left (SLR), shift right arithmetic (SHRA), rotate right (ROR), rotate left (ROL), logical AND, logical OR, Negate, and NOT.

## Project Design and Implementation

### Phase 1

To begin the design of the SRC, busmux, encoder and ALU operations, were prioritized. The busmux design consisted of a simple mux that connected all the registers along with the C-sign extended value. A simple encoder was designed to dictate which register would be pushed to the Busout. The 32-bit adder was constructed hierarchically, starting with the lowest level design of a simple B-Cell to derive the logic for the generate, propagate, and sum signals. Four B-Cells were then instantiated to create a 4-bit carry look-ahead adder. Four 4-bit carry look-ahead adders were combined to create the 16-bit adder, and two 16-bit adders were combined to create a 32-bit adder with sum and carry out outputs. The multiplier was designed using a booth algorithm with bit-pair recoding to determine the amount that the multiplicand would be shifted at each step. The division module was designed with the non-restoring division algorithm.

### Phase 2

The Select and Encode Logic had a register file of 16 registers, R0-R15, a 16-bit one-hot encoded signal was used as the register enable and bus mux input. The 16-bit value was assigned to the bus mux inputs if Rout or BAout was asserted, or it was assigned to the register enables if Rin was asserted. The design of the CON\_FF module involved an always block that assigned the decoded value for the two instruction register (IR) bits whenever they changed. Furthermore, it also included an always block that assigned the value to be given to the CONout signal when both the Bus input and the CONin signal were asserted. The RAM component was implemented by creating a 32 by 512 bit register and writing the appropriate values to their address locations using a simple enable instruction.

### Phase 3

Phase 3 contains two clocks, one for registers and one for the control unit's finite state machine. To connect the module instantiations, wires were defined within the Datapath, such as the Mdatain wire linking the RAM output wire to one of the Memory Data Register inputs. The Control Unit was developed using the first method described in the Phase 3 document. The first always block altered the finite state machine to the appropriate state based on the input clock. Each instruction involved three fetch cycles to retrieve the instruction from memory, followed by a state change based on the instruction stored in the IR. The second always block defined the signals to be asserted for each state. Also, the hardware instructions were initialized inside the RAM file so that when the testbench file ran, the CPU would cycle through the instructions one by one. Used <https://qu.pgaskin.net/ASM374/> to convert all the architecture instructions into hexadecimal numbers to be stored inside RAM.

## Evaluation Results

### Maximum Frequency of Operation

For Phase 1 and 2 the cycle time was 20ns but for Phase 3 the cycle time is 40ns. For Phase 1 the cycles used for NEG and NOT is 2, but for all the other instructions the cycles used is 4. We have 13 instructions therefore for Phase 1 we have,

$$Avg = 4*11/13 + 2*2/13$$

$$Avg = 3.69$$

And so now we have,

$$Max\ Frequency_{Phase\ 1} = 1/(cycletime) * Avg$$

$$Max\ Frequency_{Phase\ 1} = 1.845 * 10^5$$

Phase 2 has a total of 13 instructions, and an average of 8 cycles for each one. So therefore,

$$Avg = 8/13$$

$$Avg = 0.615$$

$$Max\ Frequency_{Phase\ 2} = 1/(cycle\ time) * Avg$$

$$Max\ Frequency_{Phase\ 2} = 3.07 * 10^4$$

Phase 3 has a total of 45 instructions, and an average of 5 CPU cycles per instruction and an average of 2 CU cycles per instruction

$$Avg = \frac{5 + 2}{2} / 45$$

$$Avg = 0.0777$$

$$Max\ Frequency_{Phase\ 3} = 1/(cycle\ time) * Avg$$

$$Max\ Frequency_{Phase\ 3} = 1.943 * 10^3$$

### Percentage of Chip Area Used:

I cannot give an accurate number for the percentage of chip area used however since this is a considerably small project, we can assume that the majority of the chip area is not being used.

## Discussion

One major challenge I faced was my teammate dropping out of my program at the start of the semester. Having to solo this project was challenging and very time consuming. Probably my greatest challenge was the learning curve of Verilog. Figuring out how to debug was also new to me considering I had to learn how to use Model Sim. I thought that this entire project was tedious and overly long, Phase 4 wasn't even a consideration as I just barely met the final deadline for Phase 3.

## Conclusion

This project of designing a 32-bit mini SRC was undoubtedly challenging due to its complexity. It involved the implementation of various modules, and if even one of those modules had an error this project would undoubtedly not run. There is a possibility that some parameters may not have been accounted for, which could have led to slightly incorrect results. Nonetheless, the project's challenges and potential for inaccuracies provide opportunities for further learning and improvement in future projects.

## Appendix

### Code

#### Register\_32:

```
1 module Register32 #(parameter VAL = 0) (input clk, clr, enable, input [31:0] d, output [31:0] q);
2
3 //initial q <= VAL;
4 reg [31:0] Register;
5
6 initial begin
7     Register = VAL;
8 end
9
10 always @ (posedge clk)
11 begin
12     if (clr) begin
13         Register[31:0] <= 32'b0;
14     end else if (enable) begin
15         Register[31:0] <= d;
16     end
17 end
18
19 assign q[31:0] = Register[31:0];
20
21 endmodule
```

#### Register\_MDR:

```
1 module Register_MDR (input clk, clr, enable, Read, input [31:0] Busout, Mdatain, output [31:0] q);
2
3 wire [31:0] MDMux_out;
4 assign MDMux_out = Read ? Mdatain : Busout;
5
6 Register32 MDRreg (.clk(clk), .clr(clr), .enable(enable), .d(MDMux_out), .q(q));
7
8 endmodule
9
```

#### Register\_R0:

```
1 module Register_R0 #(parameter VAL = 0) (input clk, clr, enable, BAout, input [31:0] d, output [31:0] q);
2
3 reg [31:0] Register;
4 reg [31:0] Zero = 32'b0;
5
6 initial begin
7     Register = VAL;
8 end
9
10 always @ (posedge clk)
11 begin
12     if (clr) begin
13         Register[31:0] <= 32'b0;
14     end else if (enable) begin
15         if (BAout == 0) begin
16             Register[31:0] <= d;
17         end else begin
18             Register <= 'b0;
19         end
20     end
21 end
22
23 assign q[31:0] = BAout ? Zero : Register[31:0];
24
25 endmodule
```

## C Sign Extension:

```
1 module C_SignExtended (input [31:0] d, output reg [31:0] q);
2
3 always @ (d) begin
4     q[17:0] = d[17:0];
5     q[31:18] = {d[18], d[18], d[18], d[18], d[18], d[18], d[18], d[18], d[18], d[18], d[18], d[18], d[18], d[18]};
6 end
7 endmodule
```

## Encoder:

```
1 module Encoder (output reg [4:0] Code, input[31:0]Data);
2     always @ (Data)
3     begin
4         if(Data == 32'b00000000000000000000000000000001) Code = 0; else
5         if(Data == 32'b00000000000000000000000000000010) Code = 1; else
6         if(Data == 32'b000000000000000000000000000000100) Code = 2; else
7         if(Data == 32'b0000000000000000000000000000001000) Code = 3; else
8         if(Data == 32'b00000000000000000000000000000010000) Code = 4; else
9         if(Data == 32'b000000000000000000000000000000100000) Code = 5; else
10        if(Data == 32'b0000000000000000000000000000001000000) Code = 6; else
11        if(Data == 32'b00000000000000000000000000000010000000) Code = 7; else
12        if(Data == 32'b000000000000000000000000000000100000000) Code = 8; else
13        if(Data == 32'b0000000000000000000000000000001000000000) Code = 9; else
14        if(Data == 32'b00000000000000000000000000000010000000000) Code = 10; else
15        if(Data == 32'b000000000000000000000000000000100000000000) Code = 11; else
16        if(Data == 32'b0000000000000000000000000000001000000000000) Code = 12; else
17        if(Data == 32'b00000000000000000000000000000010000000000000) Code = 13; else
18        if(Data == 32'b000000000000000000000000000000100000000000000) Code = 14; else
19        if(Data == 32'b0000000000000000000000000000001000000000000000) Code = 15; else
20        if(Data == 32'b00000000000000000000000000000010000000000000001) Code = 16; else
21        if(Data == 32'b00000000000000000000000000000010000000000000000) Code = 17; else
22        if(Data == 32'b000000000000000000000000000000100000000000000000) Code = 18; else
23        if(Data == 32'b0000000000000000000000000000001000000000000000000) Code = 19; else
24        if(Data == 32'b00000000000000000000000000000010000000000000000000) Code = 20; else
25        if(Data == 32'b000000000000000000000000000000100000000000000000000) Code = 21; else
26        if(Data == 32'b0000000000000000000000000000001000000000000000000000) Code = 22; else
27        if(Data == 32'b00000000000000000000000000000010000000000000000000000) Code = 23; else
28        if(Data == 32'b000000000000000000000000000000100000000000000000000000) Code = 24; else
29        if(Data == 32'b0000000000000000000000000000001000000000000000000000000) Code = 25; else
30        if(Data == 32'b00000000000000000000000000000010000000000000000000000000) Code = 26; else
31        if(Data == 32'b000000000000000000000000000000100000000000000000000000000) Code = 27; else
32        if(Data == 32'b0000000000000000000000000000001000000000000000000000000000) Code = 28; else
33        if(Data == 32'b00000000000000000000000000000010000000000000000000000000000) Code = 29; else
34        if(Data == 32'b0100000000000000000000000000000000000000000000000) Code = 30; else
35        if(Data == 32'b10000000000000000000000000000000000000000000000000) Code = 31; else Code = 5'bx;
36    end
37 endmodule
```

## Select and Encode:

```
1 module Select_Encode (input Gra, Grb, Grc, Rin, Rout, BAout, input [31:0] IR, output reg R0out, R1out, R2out, R3out,
2     R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out,
3     R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in, R8in, R9in, R10in, R11in, R12in, R13in, R14in, R15in);
4
5     reg [3:0] Decoder_in;
6     initial
7     begin
8         R0out = 0;
9         R1out = 0;
10        R2out = 0;
11        R3out = 0;
12        R4out = 0;
13        R5out = 0;
14        R6out = 0;
15        R7out = 0;
16        R8out = 0;
17        R9out = 0;
18        R10out = 0;
19        R11out = 0;
20        R12out = 0;
21        R13out = 0;
22        R14out = 0;
23        R15out = 0;
24        R0in = 0;
25        R1in = 0;
26        R2in = 0;
27        R3in = 0;
28        R4in = 0;
29        R5in = 0;
30        R6in = 0;
31        R7in = 0;
32        R8in = 0;
33        R9in = 0;
34        R10in = 0;
35        R11in = 0;
36        R12in = 0;
37        R13in = 0;
38        R14in = 0;
39        R15in = 0;
40    end
41
42 always @(*) begin
43     if (Gra) begin
44         Decoder_in <= IR [26:23];
45     end else if (Grb) begin
46         Decoder_in <= IR [22:19];
```

```

47     end else if (Grc) begin
48         Decoder_in <= IR [18:15];
49     end
50     R0out = 0;
51     R1out = 0;
52     R2out = 0;
53     R3out = 0;
54     R4out = 0;
55     R5out = 0;
56     R6out = 0;
57     R7out = 0;
58     R8out = 0;
59     R9out = 0;
60     R10out = 0;
61     R11out = 0;
62     R12out = 0;
63     R13out = 0;
64     R14out = 0;
65     R15out = 0;
66     R0in = 0;
67     R1in = 0;
68     R2in = 0;
69     R3in = 0;
70     R4in = 0;
71     R5in = 0;
72     R6in = 0;
73     R7in = 0;
74     R8in = 0;
75     R9in = 0;
76     R10in = 0;
77     R11in = 0;
78     R12in = 0;
79     R13in = 0;
80     R14in = 0;
81     R15in = 0;
82
83     case (Decoder_in)
84     4'b0000: begin
85         if (Rin) begin
86             R0in = 1;
87             R0out = 0;
88             end else if (Rout | BAout) begin
89                 R0out = 1;
90                 R0in = 0;
91             end
92         end
93     4'b0001: begin
94         if (Rin) begin
95             R1in = 1;
96             R1out = 0;
97             end else if (Rout | BAout) begin
98                 R1out = 1;
99                 R1in = 0;
100            end
101        end
102    4'b0010: begin
103        if (Rin) begin
104            R2in = 1;
105            R2out = 0;
106            end else if (Rout | BAout) begin
107                R2out = 1;
108                R2in = 0;
109            end
110        end
111    4'b0011: begin
112        if (Rin) begin
113            R3in = 1;
114            R3out = 0;
115            end else if (Rout | BAout) begin
116                R3out = 1;
117                R3in = 0;
118            end
119        end
120    4'b0100: begin
121        if (Rin) begin
122            R4in = 1;
123            R4out = 0;
124            end else if (Rout | BAout) begin
125                R4out = 1;
126                R4in = 0;

```



```

127         end
128     end
129     4'b0101: begin
130         if (Rin) begin
131             R5in = 1;
132             R5out = 0;
133         end else if (Rout | BAout) begin
134             R5out = 1;
135             R5in = 0;
136         end
137     end
138     4'b0110: begin
139         if (Rin) begin
140             R6in = 1;
141             R6out = 0;
142         end else if (Rout | BAout) begin
143             R6out = 1;
144             R6in = 0;
145         end
146     end
147     4'b0111: begin
148         if (Rin) begin
149             R7in = 1;
150             R7out = 0;
151         end else if (Rout | BAout) begin
152             R7out = 1;
153             R7in = 0;
154         end
155     end
156     4'b1000: begin
157         if (Rin) begin
158             R8in = 1;
159             R8out = 0;
160         end else if (Rout | BAout) begin
161             R8out = 1;
162             R8in = 0;
163         end
164     end
165     4'b1001: begin
166         if (Rin) begin
167             R9in = 1;
168             R9out = 0;
169         end else if (Rout | BAout) begin
170             R9out = 1;
171             R9in = 0;
172         end

```

```

173     end
174     4'b1010: begin
175         if (Rin) begin
176             R10in = 1;
177             R10out = 0;
178         end else if (Rout | BAout) begin
179             R10out = 1;
180             R10in = 0;
181         end
182     end
183     4'b1011: begin
184         if (Rin) begin
185             R11in = 1;
186             R11out = 0;
187         end else if (Rout | BAout) begin
188             R11out = 1;
189             R11in = 0;
190         end
191     end
192     4'b1100: begin
193         if (Rin) begin
194             R12in = 1;
195             R12out = 0;
196         end else if (Rout | BAout) begin
197             R12out = 1;
198             R12in = 0;
199         end
200     end
201     4'b1101: begin
202         if (Rin) begin
203             R13in = 1;
204             R13out = 0;
205         end else if (Rout | BAout) begin
206             R13out = 1;
207             R13in = 0;
208         end
209     end
210     4'b1110: begin
211         if (Rin) begin
212             R14in = 1;
213             R14out = 0;
214         end else if (Rout | BAout) begin
215             R14out = 1;
216             R14in = 0;
217         end
218     end

```

```

219     4'b1111: begin
220         if (Rin) begin
221             R15in = 1;
222             R15out = 0;
223         end else if (Rout | BAout) begin
224             R15out = 1;
225             R15in = 0;
226         end
227     end
228 endcase
229 end
230 endmodule

```

## BusMux:

```

1 module BusMux (input [31:0] BusMuxin_R0, BusMuxin_R1, BusMuxin_R2, BusMuxin_R3, BusMuxin_R4, BusMuxin_R5, BusMuxin_R6, BusMuxin_R7,
2   BusMuxin_R8, BusMuxin_R9, BusMuxin_R10, BusMuxin_R11, BusMuxin_R12, BusMuxin_R13, BusMuxin_R14, BusMuxin_R15,
3   BusMuxin_HI, BusMuxin_LO, BusMuxin_ZHI, BusMuxin_ZLO, BusMuxin_PC, BusMuxin_MDR, BusMuxin_InPort, C_sign_extended,
4   input [4:0] select,
5   output [31:0] BusMuxout);
6   reg [31:0] mux_int;
7   always @ (BusMuxin_R0, BusMuxin_R1, BusMuxin_R2, BusMuxin_R3, BusMuxin_R4, BusMuxin_R5, BusMuxin_R6, BusMuxin_R7,
8     BusMuxin_R8, BusMuxin_R9, BusMuxin_R10, BusMuxin_R11, BusMuxin_R12, BusMuxin_R13, BusMuxin_R14, BusMuxin_R15,
9     BusMuxin_HI, BusMuxin_LO, BusMuxin_ZHI, BusMuxin_ZLO, BusMuxin_PC, BusMuxin_MDR, BusMuxin_InPort, C_sign_extended, select)
10     if(select == 0) mux_int = BusMuxin_R0; else
11     if(select == 1) mux_int = BusMuxin_R1; else
12     if(select == 2) mux_int = BusMuxin_R2; else
13     if(select == 3) mux_int = BusMuxin_R3; else
14     if(select == 4) mux_int = BusMuxin_R4; else
15     if(select == 5) mux_int = BusMuxin_R5; else
16     if(select == 6) mux_int = BusMuxin_R6; else
17     if(select == 7) mux_int = BusMuxin_R7; else
18     if(select == 8) mux_int = BusMuxin_R8; else
19     if(select == 9) mux_int = BusMuxin_R9; else
20     if(select == 10) mux_int = BusMuxin_R10; else
21     if(select == 11) mux_int = BusMuxin_R11; else
22     if(select == 12) mux_int = BusMuxin_R12; else
23     if(select == 13) mux_int = BusMuxin_R13; else
24     if(select == 14) mux_int = BusMuxin_R14; else
25     if(select == 15) mux_int = BusMuxin_HI; else
26     if(select == 16) mux_int = BusMuxin_LO; else
27     if(select == 17) mux_int = BusMuxin_ZHI; else
28     if(select == 18) mux_int = BusMuxin_ZLO; else
29     if(select == 19) mux_int = BusMuxin_PC; else
30     if(select == 20) mux_int = BusMuxin_MDR; else
31     if(select == 21) mux_int = BusMuxin_InPort; else
32     if(select == 22) mux_int = C_sign_extended; else mux_int = 32'bx;
33   assign BusMuxout = mux_int;
34 endmodule

```

## ALU:

```

1  module ALU (input [31:0] A, B, input [4:0] ALU_ctl, input IncPC, output [31:0] Zhigh, Zlow);
2
3  parameter    Load = 5'b00000, Load_imm = 5'b00001, Store = 5'b00010, Addition = 5'b00011, Subtraction = 5'b00100,
4               AND = 5'b00101, OR = 5'b00110, ShiftR = 5'b00111, SHRA = 5'b01000, ShiftL = 5'b01001, ROR = 5'b01010, ROL = 5'b01011,
5               Add_imm = 5'b01100, AND_imm = 5'b01101, OR_imm = 5'b01110, Multiply = 5'b01111, Divide = 5'b10000,
6               Negate = 5'b10001, NOT = 5'b10010, Branch = 5'b10011;
7
8  wire [31:0] Adder_out, Subtractor_out, Shr_out, Shl_out, Ror_out, Rol_out,
9             And_out, Or_out, Neg_out, Not_out, Div_out, Div_rem, IncPC_out, Shra_out;
10
11 wire [63:0] Mult_out;
12 wire Adder_cout, Subtractor_cout, IncPC_cout;
13
14 reg [31:0] Zltemp;
15 reg [31:0] Zhtemp;
16
17 assign Zlow = Zltemp;
18 assign Zhigh = Zhtemp;
19
20 AND_32 Ander (A, B, And_out);
21 OR_32 Orer (A, B, Or_out);
22 Adder_32 Adder (A, B, 1'b0, Adder_out, Adder_cout);
23 Adder_32 PC_inc (32'b1, B, 1'b0, IncPC_out, IncPC_cout);
24 Subtraction Subtractor (A, B, 1'b0, Subtractor_out, Subtractor_cout);
25 Multiplier Mult (A, B, Mult_out);
26 Division Divider (A, B, Div_out, Div_rem);
27 SHR ShifterR (A, B, Shr_out);
28 SHRA ShifterArithmetic (A, B, Shra_out);
29 SHL ShifterL (A, B, Shl_out);
30 ROR RoR (A, B, Ror_out);
31 ROL RoL (A, B, Rol_out);
32 NEG Negater (A, Neg_out);
33 NOT_32 Notter (A, Not_out);
34
35 always @ (*) begin
36     case (ALU_ctl)
37     | Addition, Add_imm, Load, Load_imm, Store, Branch: begin
38         Zltemp <= Adder_out;
39         Zhtemp <= {31'b0, Adder_cout};
40     end
41     | Subtraction: begin
42         Zltemp <= Subtractor_out;
43         Zhtemp <= {31'b0, Subtractor_cout};
44     end
45     | ShiftR: begin
46         Zltemp <= Shr_out;
47         Zhtemp <= 32'b0;
48     end
49     | ShiftL: begin
50         Zltemp <= Shl_out;
51         Zhtemp <= 32'b0;
52     end
53     | ROR: begin
54         Zltemp <= Ror_out;
55         Zhtemp <= 32'b0;
56     end
57     | ROL: begin
58         Zltemp <= Rol_out;
59         Zhtemp <= 32'b0;
60     end
61     | AND, AND_imm: begin
62         Zltemp <= And_out;
63         Zhtemp <= 32'b0;
64     end
65     | OR, OR_imm: begin
66         Zltemp <= Or_out;
67         Zhtemp <= 32'b0;
68     end
69     | Multiply: begin
70         Zltemp <= Mult_out[31:0];
71         Zhtemp <= Mult_out[63:32];
72     end
73     | Divide: begin
74         Zltemp <= Div_out;
75         Zhtemp <= Div_rem;
76     end
77     | Negate: begin
78         Zltemp <= Neg_out;
79         Zhtemp <= 32'b0;
80     end
81     | NOT: begin
82         Zltemp <= Not_out;
83         Zhtemp <= 32'b0;
84     end
85     | default: begin
86         Zltemp <= Adder_out;
87         Zhtemp <= 32'b0;
88     end
89     | SHRA: begin
90         Zltemp <= Shra_out;

```

```

91         Zhtemp <= 32'b0;
92     end
93 endcase
94
95 if (IncPC) begin
96     Zltemp <= IncPC_out;
97     Zhtemp <= 32'b0;
98 end
99 end
100 endmodule

```

## Adder\_32:

```

1  module Adder_32 (input [31:0] a, b, input cin, output [31:0] sum, output cout);
2
3      wire [1:0] carry;
4      assign carry[0] = cin;
5
6      Adder_16 Adder_1 (a[15:0], b[15:0], carry[0], sum[15:0], carry[1]);
7      Adder_16 Adder_2 (a[31:16], b[31:16], carry[1], sum[31:16], cout);
8
9  endmodule
10
11 module Adder_16 (input [15:0] a, b, input cin, output [15:0] sum, output cout);
12
13     wire [3:0] carry;
14     assign carry[0] = cin;
15
16     CLA_4bits Adder_1 (a[3:0], b[3:0], carry[0], sum[3:0], carry[1]);
17     CLA_4bits Adder_2 (a[7:4], b[7:4], carry[1], sum[7:4], carry[2]);
18     CLA_4bits Adder_3 (a[11:8], b[11:8], carry[2], sum[11:8], carry[3]);
19     CLA_4bits Adder_4 (a[15:12], b[15:12], carry[3], sum[15:12], cout);
20
21 endmodule
22
23 module CLA_4bits (input [3:0] a, b, input cin, output [3:0] sum, output cout);
24
25     wire [3:0] carry;
26     wire [3:0] P, G;
27
28     BCell b0(a[0], b[0], carry[0], sum[0], G[0], P[0]);
29     BCell b1(a[1], b[1], carry[1], sum[1], G[1], P[1]);
30     BCell b2(a[2], b[2], carry[2], sum[2], G[2], P[2]);
31     BCell b3(a[3], b[3], carry[3], sum[3], G[3], P[3]);
32
33     assign carry[0] = cin;
34     assign carry[1] = G[0] | (P[0] & carry[0]);
35     assign carry[2] = G[1] | (P[1] & carry[1]);
36     assign carry[3] = G[2] | (P[2] & carry[2]);
37     assign cout = G[3] | (P[3] & carry[3]);
38
39 endmodule
40
41 module BCell (input a, b, cin, output sum, G, P);
42     assign G = a&b;
43     assign P = a ^ b;
44     assign sum = a ^ b ^ cin;
45 endmodule
46
47

```

## 32-bit AND:

```

1  module AND_32 (input [31:0] a, b, output [31:0] result);
2
3      reg [31:0] temp;
4      integer i;
5      always @(*) begin
6          for (i = 0; i < 32; i = i + 1) begin
7              temp[i] = a[i] & b[i];
8          end
9      end
10     assign result = temp;
11 endmodule

```

## Division:

```
1 module Division (input [31:0] Q, M, output [31:0] result, remainder);
2
3     reg [64:0] AQ;
4     reg [32:0] Mtemp, Rem;
5     reg [5:0] i;
6
7     assign result = AQ[31:0];
8     assign remainder = Rem[31:0];
9
10    always @ (Q or M) begin
11        AQ[64:32] = 0;
12        AQ[31:0] = Q;
13        Mtemp[31:0] = M;
14        Mtemp[32] = M[31];
15
16        for (i = 0; i < 32; i = i + 1) begin
17            AQ = AQ << 1;
18            if (AQ[64] == 0) begin
19                AQ[64:32] = AQ[64:32] - Mtemp;
20            end else begin
21                AQ[64:32] = AQ[64:32] + Mtemp;
22            end
23
24            if (AQ[64] == 0) begin
25                AQ[0] = 1;
26            end else begin
27                AQ[0] = 0;
28            end
29        end
30        if (AQ[64]) begin
31            Rem = AQ[64:32] + Mtemp;
32        end else begin
33            Rem = AQ[64:32];
34        end
35    end
36 endmodule
37
```

## Multiplier:

```
1 module Multiplier (input [31:0] M, Q, output [63:0] result);
2     reg [63:0] prod;
3     reg [2:0] bit_pair_recode;
4     assign result = prod;
5     integer i;
6     always @(*) begin
7         prod = 0;
8         for (i = 0; i < 32; i = i + 2) begin
9             if (i == 0) begin
10                 bit_pair_recode = {Q[i+1], Q[i], 1'b0};
11             end else begin
12                 bit_pair_recode = {Q[i+1], Q[i], Q[i-1]};
13             end
14
15             if (bit_pair_recode == 3'b001) begin
16                 prod = prod + (M << i);
17             end else if (bit_pair_recode == 3'b010) begin
18                 prod = prod + (M << i);
19             end else if (bit_pair_recode == 3'b011) begin
20                 prod = prod + (M << i + 1);
21             end else if (bit_pair_recode == 3'b100) begin
22                 prod = prod + ((-M) << i + 1);
23             end else if (bit_pair_recode == 3'b101) begin
24                 prod = prod + ((-M) << i);
25             end else if (bit_pair_recode == 3'b110) begin
26                 prod = prod + ((-M) << i);
27             end else begin
28                 prod = prod;
29             end
30         end
31     end
32 endmodule
33
34
```

### 32-bit NEG:

```
1 module NEG (input [31:0] a, output [31:0] result);
2     //we have to not the whole number then add 1
3     wire [31:0] temp;
4     wire cout;
5
6     NOT_32 logical_NOT (a, temp);
7     Adder_32 add_1 (temp, 32'd1, 1'd0, result, cout);
8
9 endmodule
10
```

### 32-bit NOT:

```
1 module NOT_32 (input [31:0] a, output [31:0] result);
2     reg [31:0] temp;
3     integer i;
4     assign result = temp;
5     always @ (a) begin
6         for (i = 0; i < 32; i = i + 1)
7             begin
8                 temp[i] = !a[i];
9             end
10    end
11 endmodule
12
```

### 32-bit OR:

```
1 module OR_32 (input [31:0] a, b, output [31:0] result);
2
3     reg [31:0] temp;
4     integer i;
5     always @(*) begin
6         for (i = 0; i < 32; i = i + 1) begin
7             temp[i] = a[i] | b[i];
8         end
9     end
10    assign result = temp;
11 endmodule
12
```

### 32-bit ROL:

```
1 module ROL (input [31:0] a, b, output [31:0] result);
2     reg [31:0] res;
3     assign result = res;
4     always @ (a or b) begin
5         case (b)
6             5'd1 : res <= {a[30:0], a[31]};
7             5'd2 : res <= {a[29:0], a[31:30]};
8             5'd3 : res <= {a[28:0], a[31:29]};
9             5'd4 : res <= {a[27:0], a[31:28]};
10            5'd5 : res <= {a[26:0], a[31:27]};
11            5'd6 : res <= {a[25:0], a[31:26]};
12            5'd7 : res <= {a[24:0], a[31:25]};
13            5'd8 : res <= {a[23:0], a[31:24]};
14            5'd9 : res <= {a[22:0], a[31:23]};
15            5'd10 : res <= {a[21:0], a[31:22]};
16            5'd11 : res <= {a[20:0], a[31:21]};
17            5'd12 : res <= {a[19:0], a[31:20]};
18            5'd13 : res <= {a[18:0], a[31:19]};
19            5'd14 : res <= {a[17:0], a[31:18]};
20            5'd15 : res <= {a[16:0], a[31:17]};
21            5'd16 : res <= {a[15:0], a[31:16]};
22            5'd17 : res <= {a[14:0], a[31:15]};
23            5'd18 : res <= {a[13:0], a[31:14]};
24            5'd19 : res <= {a[12:0], a[31:13]};
25            5'd20 : res <= {a[11:0], a[31:12]};
26            5'd21 : res <= {a[10:0], a[31:11]};
27            5'd22 : res <= {a[9:0], a[31:10]};
28            5'd23 : res <= {a[8:0], a[31:9]};
29            5'd24 : res <= {a[7:0], a[31:8]};
30            5'd25 : res <= {a[6:0], a[31:7]};
31            5'd26 : res <= {a[5:0], a[31:6]};
32            5'd27 : res <= {a[4:0], a[31:5]};
33            5'd28 : res <= {a[3:0], a[31:4]};
34            5'd29 : res <= {a[2:0], a[31:3]};
35            5'd30 : res <= {a[1:0], a[31:2]};
36            5'd31 : res <= {a[0], a[31:1]};
37            default: res <= a;
38        endcase
39    end
40 endmodule
41
42
```

### 32-bit ROR:

```
1 module ROR (input [31:0] a, b, output[31:0] result);
2   reg [31:0] res;
3   assign result = res;
4   always @(a or b) begin
5     case (b)
6       5'd1: res <= {a[0], a[31:1]};
7       5'd2: res <= {a[1:0], a[31:2]};
8       5'd3: res <= {a[2:0], a[31:3]};
9       5'd4: res <= {a[3:0], a[31:4]};
10      5'd5: res <= {a[4:0], a[31:5]};
11      5'd6: res <= {a[5:0], a[31:6]};
12      5'd7: res <= {a[6:0], a[31:7]};
13      5'd8: res <= {a[7:0], a[31:8]};
14      5'd9: res <= {a[8:0], a[31:9]};
15      5'd10: res <= {a[9:0], a[31:10]};
16      5'd11: res <= {a[10:0], a[31:11]};
17      5'd12: res <= {a[11:0], a[31:12]};
18      5'd13: res <= {a[12:0], a[31:13]};
19      5'd14: res <= {a[13:0], a[31:14]};
20      5'd15: res <= {a[14:0], a[31:15]};
21      5'd16: res <= {a[15:0], a[31:16]};
22      5'd17: res <= {a[16:0], a[31:17]};
23      5'd18: res <= {a[17:0], a[31:18]};
24      5'd19: res <= {a[18:0], a[31:19]};
25      5'd20: res <= {a[19:0], a[31:20]};
26      5'd21: res <= {a[20:0], a[31:21]};
27      5'd22: res <= {a[21:0], a[31:22]};
28      5'd23: res <= {a[22:0], a[31:23]};
29      5'd24: res <= {a[23:0], a[31:24]};
30      5'd25: res <= {a[24:0], a[31:25]};
31      5'd26: res <= {a[25:0], a[31:26]};
32      5'd27: res <= {a[26:0], a[31:27]};
33      5'd28: res <= {a[27:0], a[31:28]};
34      5'd29: res <= {a[28:0], a[31:29]};
35      5'd30: res <= {a[29:0], a[31:30]};
36      5'd31: res <= {a[30:0], a[31]};
37      default: res <= a;
38    endcase
39  end
40 endmodule
41
42
```

### 32-bit SHL:

```
1 module SHL (input [31:0] a, b, output [31:0] result);
2
3   assign result = a << b;
4
5 endmodule
6
```

### 32-bit SHR:

```
1 module SHR (input [31:0] a, b, output [31:0] result);
2
3   assign result = a >> b;
4
5 endmodule
6
```

### 32-bit SHRA:

```
1 module SHRA (input [31:0] a, b, output [31:0] result);
2
3   assign result = a >> b | {a[31], 31'b0};
4
5 endmodule
6
```

### 32-bit Subtractor:

```
1 module Subtraction (input [31:0] a, b, input cin, output [31:0] result, output cout);
2   //negate the second value and add them
3   wire [31:0] temp;
4   NEG neg_breg (b, temp);
5   Adder_32 subtracting (a, temp, cin, result, cout);
6
7 endmodule
8
```

## CON FF Logic:

```
1 module CON_FF (input CONIn, input [31:0] Busin, input [1:0] IR_bits, output CONOut);
2
3 reg [3:0] decode_out;
4 reg CON_D;
5 reg CON_Q;
6 assign CONOut = CON_Q;
7
8 always @ (*) begin
9     CON_D = 0;
10    case (IR_bits)
11    2'b00: begin
12        decode_out = 4'b0001;
13    end
14    2'b01: begin
15        decode_out = 4'b0010;
16    end
17    2'b10: begin
18        decode_out = 4'b0100;
19    end
20    2'b11: begin
21        decode_out = 4'b1000;
22    end
23    endcase
24
25    if (Busin == 0 && decode_out[0]) begin
26        CON_D = 1;
27    end else if (Busin != 0 && decode_out[1]) begin
28        CON_D = 1;
29    end else if (Busin >= 0 && decode_out[2]) begin
30        CON_D = 1;
31    end else if (Busin[31] && decode_out[3]) begin
32        CON_D = 1;
33    end
34    if (CONIn) begin
35        CON_Q = CON_D;
36    end
37 end
38
39 endmodule
```

## Input Port:

```
1 module In_Port #(parameter VAL = 0) (input clk, clr, InPort_in, input [31:0] d, output reg [31:0] q);
2
3 initial q <= VAL;
4
5 always @ (posedge clk)
6 begin
7     if (clr) begin
8         q <= 32'b0;
9     end else if (InPort_in) begin
10        q <= d;
11    end
12 end
13
14 endmodule
```

## Output Port:

```
1 module Out_Port #(parameter VAL = 0) (input clk, clr, Out_Portin, input [31:0] d, output reg [31:0] q);
2
3 initial q <= VAL;
4
5 always @ (posedge clk)
6 begin
7     if (clr) begin
8         q <= 32'b0;
9     end else if (Out_Portin) begin
10        q <= d;
11    end
12 end
13
14 endmodule
```

## Control Unit:

```

1  `timescale 1ns/1ns
2  module Control (
3      output reg    Gra, Grb, Grc, Rin, Rout, Rl5in,
4      Hiin, Loin, CONin, PCin, IRin, Yin, Zin, MARin, MDRin, OutPortin, Cout, BAout,
5      PCout, MDRout, Zhiout, Zlowout, HIout, LOout, InPortout, IncPC,
6      Read, Write, Clear, Run,
7      input [31:0]  IR,
8      input          Clock, Reset, Stop, CON_out);
9
10     parameter      Load = 5'b00000,
11                     Load_imm = 5'b000001,
12                     Store = 5'b000010,
13                     Addition = 5'b000011,
14                     Subtraction = 5'b000100,
15                     AND = 5'b000101,
16                     OR = 5'b000110,
17                     SHR = 5'b000111,
18                     SHRA = 5'b010000,
19                     SHL = 5'b010001,
20                     ROR = 5'b010010,
21                     ROL = 5'b010011,
22                     ADD_imm = 5'b010100,
23                     AND_imm = 5'b010101,
24                     OR_imm = 5'b010110,
25                     Multiply = 5'b010111,
26                     Divide = 5'b100000,
27                     Negate = 5'b100001,
28                     NOT = 5'b100010,
29                     Branch = 5'b100011,
30                     JR = 5'b101000,
31                     JAL = 5'b101001,
32                     IN = 5'b101010,
33                     OUT = 5'b101011,
34                     MFHI = 5'b110000,
35                     MFLO = 5'b110001,
36                     NOP = 5'b110010,
37                     HALT = 5'b110011;
38
39     parameter      Reset_state = 6'b0000000,
40                     fetch0 = 6'b0000001,
41                     fetch1 = 6'b0000010,
42                     fetch2 = 6'b0000011,
43                     ALUbasic3 = 6'b0000100,
44                     ALUbasic4 = 6'b0000101,
45                     ALUbasic5 = 6'b0000110,
46                     ALUimm3 = 6'b0000111,
47
48                     ALUimm4 = 6'b0001000,
49                     ALUimm5 = 6'b0001001,
50                     muldiv3 = 6'b0001010,
51                     muldiv4 = 6'b0001011,
52                     muldiv5 = 6'b0001100,
53                     muldiv6 = 6'b0001101,
54                     ld3 = 6'b0001110,
55                     ld4 = 6'b0001111,
56                     ld5 = 6'b0100000,
57                     ld6 = 6'b0100001,
58                     ld7 = 6'b0100010,
59                     ldimm3 = 6'b0100011,
60                     ldimm4 = 6'b0100100,
61                     ldimm5 = 6'b0100101,
62                     st3 = 6'b0101010,
63                     st4 = 6'b0101011,
64                     st5 = 6'b0101000,
65                     st6 = 6'b0101001,
66                     st7 = 6'b0101010,
67                     hr3 = 6'b0101011,
68                     hr4 = 6'b0101100,
69                     hr5 = 6'b0101101,
70                     hr6 = 6'b0101110,
71                     jal3 = 6'b0101111,
72                     jal4 = 6'b1000000,
73                     jr3 = 6'b1000001,
74                     mfhi3 = 6'b1000010,
75                     mflo3 = 6'b1000011,
76                     in3 = 6'b1000100,
77                     out3 = 6'b1000101,
78                     halt3 = 6'b1000110;
79
80     reg [5:0]      Present_state = Reset_state;
81
82     always @(posedge Clock, posedge Reset) begin
83         if (Reset == 1'b1) Present_state = Reset_state;
84         else if (Run == 1'b1)
85             case(Present_state)
86                 Reset_state : Present_state = fetch0;
87                 fetch0      : Present_state = fetch1;
88                 fetch1      : Present_state = fetch2;
89                 fetch2      : begin

```



```

90 fetch2 : begin
91     case(IR[31:27])
92         Addition, Subtraction, AND, OR, SHR, SHL, ROR, SHRA, ROL, Negate, NOT :
93             Present_state = ALUbasic3;
94             ADD_imm, AND_imm, OR_imm :
95                 Present_state = ALUimm3;
96             Multiply, Divide :
97                 Present_state = muldiv3;
98             Load :
99                 Present_state = ld3;
100             Load_imm :
101                 Present_state = ldimm3;
102             Store :
103                 Present_state = st3;
104             Branch :
105                 Present_state = br3;
106             JAL :
107                 Present_state = jal3;
108             JR :
109                 Present_state = jr3;
110             MFHI :
111                 Present_state = mfhi3;
112             MFLO :
113                 Present_state = mflo3;
114             IN :
115                 Present_state = in3;
116             OUT :
117                 Present_state = out3;
118             NOP :
119                 Present_state = fetch0;
120             HALT :
121                 Present_state = halt3;
122     endcase
123 end
124
125 ALUbasic3 : Present_state = ALUbasic4;
126 ALUbasic4 : Present_state = ALUbasic5;
127 ALUbasic5 : Present_state = fetch0;
128
129 ALUimm3 : Present_state = ALUimm4;
130 ALUimm4 : Present_state = ALUimm5;
131 ALUimm5 : Present_state = fetch0;
132
133 muldiv3 : Present_state = muldiv4;
134
135 muldiv4 : Present_state = muldiv5;
136 muldiv5 : Present_state = muldiv6;
137 muldiv6 : Present_state = fetch0;
138
139 ld3 : Present_state = ld4;
140 ld4 : Present_state = ld5;
141 ld5 : Present_state = ld6;
142 ld6 : Present_state = ld7;
143 ld7 : Present_state = fetch0;
144
145 ldimm3 : Present_state = ldimm4;
146 ldimm4 : Present_state = ldimm5;
147 ldimm5 : Present_state = fetch0;
148
149 st3 : Present_state = st4;
150 st4 : Present_state = st5;
151 st5 : Present_state = st6;
152 st6 : Present_state = st7;
153 st7 : Present_state = fetch0;
154
155 br3 : Present_state = br4;
156 br4 : Present_state = br5;
157 br5 : Present_state = br6;
158 br6 : Present_state = fetch0;
159
160 jal3 : Present_state = jal4;
161 jal4 : Present_state = fetch0;
162
163 jr3 : Present_state = fetch0;
164 mfhi3 : Present_state = fetch0;
165 mflo3 : Present_state = fetch0;
166 in3 : Present_state = fetch0;
167 out3 : Present_state = fetch0;
168 halt3 : Present_state = fetch0;
169 endcase
170 end
171 always @(Present_state, Stop) begin
172     case (Present_state)
173     Reset_state : begin
174         Gra <= 0; Grb <= 0; Grc <= 0; Rin <= 0; Rout <= 0; Rl5in <= 0;
175         Hl1in <= 0; LOin <= 0; CONin <= 0; PCin <= 0; IRin <= 0; Yin <= 0; Zin <= 0; MARin <= 0;
176         MDRin <= 0; OutPortin <= 0; Cout <= 0; BAout <= 0; PCout <= 0; MDRout <= 0; Zhiout <= 0;
177         Zlowout <= 0; HIout <= 0; LOout <= 0; InPortout <= 0; IncPC <= 0; Read <= 0; Write <= 0;
178         Clear <= 1; Run <= 1;

```

```

179      #25 Clear <= 0;
180  end
181  fetch0 : begin
182      PCout <= 1; MARin <= 1; IncPC <= 1; Zin <= 1;
183      #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
184  end
185  fetch1 : begin
186      Zlowout <= 1; PCin <= 1; Read <= 1; MDRin <= 1;
187      #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
188  end
189  fetch2 : begin
190      MDRout <= 1; IRin <= 1;
191      #25 MDRout <= 0; IRin <= 0;
192  end
193  ALUbasic3 : begin
194      Grb <= 1; Rout <= 1; Yin <= 1;
195      #25 Grb <= 0; Rout <= 0; Yin <= 0;
196  end
197  ALUbasic4 : begin
198      Gro <= 1; Rout <= 1; Zin <= 1;
199      #25 Gro <= 0; Rout <= 0; Zin <= 0;
200  end
201  ALUbasic5 : begin
202      Zlowout <= 1; Gra <= 1; Rin <= 1;
203      #25 Zlowout <= 0; Gra <= 0; Rin <= 0;
204  end
205  ALUimm3 : begin
206      Grb <= 1; Rout <= 1; Yin <= 1;
207      #25 Grb <= 0; Rout <= 0; Yin <= 0;
208  end
209  ALUimm4 : begin
210      Zin <= 1; Cout <= 1;
211      #25 Cout <= 0; Zin <= 0;
212  end
213  ALUimm5 : begin
214      Zlowout <= 1; Gra <= 1; Rin <= 1;
215      #25 Gra <= 0; Rin <= 0; Zlowout <= 0;
216  end
217  muldiv3 : begin
218      Gra <= 1; Rout <= 1; Yin <= 1;

```

```

219      #25 Gra <= 0; Rout <= 0; Yin <= 0;
220  end
221  muldiv4 : begin
222      Grb <= 1; Rout <= 1; Zin <= 1;
223      #25 Grb <= 0; Rout <= 0; Zin <= 0;
224  end
225  muldiv5 : begin
226      Zlowout <= 1; LOin <= 1;
227      #25 Zlowout <= 0; LOin <= 0;
228  end
229  muldiv6 : begin
230      Zhiout <= 1; HIin <= 1;
231      #25 Zhiout <= 0; HIin <= 0;
232  end
233  ld3 : begin
234      Grb <= 1; BAout <= 1; Yin <= 1;
235      #35 Grb <= 0; BAout <= 0; Yin <= 0;
236  end
237  ld4 : begin
238      Zin <= 1; Cout <= 1;
239      #25 Cout <= 0; Zin <= 0;
240  end
241  ld5 : begin
242      Zlowout <= 1; MARin <= 1;
243      #25 MARin <= 0; Zlowout <= 0;
244  end
245  ld6 : begin
246      Read <= 1; MDRin <= 1;
247      #25 Read <= 0; MDRin <= 0;
248  end
249  ld7 : begin
250      MDRout <= 1; Gra <= 1; Rin <= 1;
251      #25 MDRout <= 0; Gra <= 0; Rin <= 0;
252  end
253  ldimm3 : begin
254      Grb <= 1; BAout <= 1; Yin <= 1;
255      #35 Grb <= 0; BAout <= 0; Yin <= 0;
256  end
257  ldimm4 : begin
258      Zin <= 1; Cout <= 1;
259      #25 Cout <= 0; Zin <= 0;
260  end
261  ldimm5 : begin
262      Zlowout <= 1; Gra <= 1; Rin <= 1;

```

```

263      #25 Gra <= 0; Rin <= 0; Zlowout <= 0;
264  end
265  st3 : begin
266      BAout <= 1; Yin <= 1; Grb <= 1;
267      #35 Grb <= 0; BAout <= 0; Yin <= 0;
268  end
269  st4 : begin
270      Zin <= 1; Cout <= 1;
271      #25 Cout <= 0; Zin <= 0;
272  end
273  st5 : begin
274      Zlowout <= 1; MARin <= 1;
275      #25 MARin <= 0; Zlowout <= 0;
276  end
277  st6 : begin
278      Gra <= 1; BAout <= 1; MDRin <= 1;
279      #35 BAout <= 0; Gra <= 0; MDRin <= 0;
280  end
281  st7 : begin
282      Write <= 1;
283      #25 Write <= 0;
284  end
285  br3 : begin
286      Gra <= 1; Rout <= 1; CONin <= 1;
287      #25 Gra <= 0; Rout <= 0; CONin <= 0;
288  end
289  br4 : begin
290      PCout <= 1; Yin <= 1;
291      #25 PCout <= 0; Yin <= 0;
292  end
293  br5: begin
294      Cout <= 1; Zin <= 1;
295      #25 Cout <= 0; Zin <= 0;
296  end
297  br6 : begin
298      Zlowout <= 1;
299      if (CON_out) begin
300          PCin <= 1;
301      end
302      #25 PCin <= 0; Zlowout <= 0;
303  end
304  jal3 : begin
305      PCout <= 1; Rl5in <= 1;
306      #25 Rl5in <= 0; PCout <= 0;

```

```

307  end
308  jal4 : begin
309      Rout <= 1; PCin <= 1; Gra <= 1;
310      #25 Gra <= 0; PCin <= 0; Rout <= 0;
311  end
312  jrr3 : begin
313      Rout <= 1; PCin <= 1; Gra <= 1;
314      #25 Gra <= 0; PCin <= 0; Rout <= 0;
315  end
316  mphi3 : begin
317      HIout <= 1; Gra <= 1; Rin <= 1;
318      #25 Gra <= 0; Rin <= 0; HIout <= 0;
319  end
320  mflo3 : begin
321      LOout <= 1; Gra <= 1; Rin <= 1;
322      #25 Gra <= 0; Rin <= 0; LOout <= 0;
323  end
324  in3 : begin
325      InPortout <= 1; Gra <= 1; Rin <= 1;
326      #25 Gra <= 0; Rin <= 0; InPortout <= 0;
327  end
328  out3 : begin
329      Gra <= 1; Rout <= 1; OutPortin <= 1;
330      #25 Gra <= 0; Rout <= 0; OutPortin <= 0;
331  end
332  halt3 : begin
333      Run <= 0;
334  end
335  endcase
336  if (Stop) begin
337      Run <= 0;
338  end
339  end
340  endmodule
341
342
343

```

Ram:

```

1  module RAM (input clk, write, input [8:0] addr, input [31:0] d, output reg [31:0] q);
2
3  reg [31:0] ram [511:0];
4
5  initial begin
6      ram[0] <= 32'h08800002;
7      ram[1] <= 32'h08080000;
8      ram[2] <= 32'h01000068;
9      ram[3] <= 32'h0917FFFC;
10     ram[4] <= 32'h00900001;
11     ram[5] <= 32'h09800069;
12     ram[6] <= 32'h99980004;
13     ram[7] <= 32'h09980002;
14     ram[8] <= 32'h039FFFFD;
15     ram[9] <= 32'hD0000000;
16     ram[10] <= 32'h9B900002;
17     ram[11] <= 32'h09000005;
18     ram[12] <= 32'h09880002;
19     ram[13] <= 32'h19918000;
20     ram[14] <= 32'h63B80002;
21     ram[15] <= 32'h8BB80000;
22     ram[16] <= 32'h93B80000;
23     ram[17] <= 32'h6BB8000F;
24     ram[18] <= 32'h50880000;
25     ram[19] <= 32'h7388001C;
26     ram[20] <= 32'h43B80000;
27     ram[21] <= 32'h39180000;
28     ram[22] <= 32'h11000052;
29     ram[23] <= 32'h59100000;
30     ram[24] <= 32'h31180000;
31     ram[25] <= 32'h28908000;
32     ram[26] <= 32'h11880060;
33     ram[27] <= 32'h21918000;
34     ram[28] <= 32'h48900000;
35     ram[29] <= 32'h0A000006;
36     ram[30] <= 32'h0A800032;
37     ram[31] <= 32'h7AA00000;
38     ram[32] <= 32'hC3800000;
39     ram[33] <= 32'hCB000000;
40     ram[34] <= 32'h82A00000;
41     ram[35] <= 32'h0C27FFFF;
42     ram[36] <= 32'h0CAFFFFD;
43     ram[37] <= 32'h0D300000;
44     ram[38] <= 32'h0DB80000;
45     ram[39] <= 32'hAD000000;
46     ram[40] <= 32'hD8000000;
47
48     ram[300] <= 32'h1EC50000;
49     ram[301] <= 32'h264D8000;
50     ram[302] <= 32'h26EE0000;
51     ram[303] <= 32'hA7800000;
52     ram[104] <= 32'h55;
53     ram[82] <= 32'h26;
54
55 end
56
57 always @ (posedge clk) begin
58     if(!write) begin
59         q = ram[addr];
60     end else begin
61         ram[addr] = d;
62     end
63 end
64
65 endmodule

```

## Test Benches

### Phase 1:

The testbench files for the other arithmetic operations are very similar. The only difference is that the IRout value gets changed to allow the ALU to select the proper operation. Also, for NEG and NOT only one value needs to be pushed to the ALU so those testbench files are also slightly different.

### ADD\_TB:

```
1 timescale 1ns/10ps
2 module add_tb;
3 reg PCout, Zlowout, MDRout, HIout, LOout, Zhiout, InPortout, Cout;
4 reg R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out;
5 reg R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in;
6 reg MDRin, Zin, PCin, MDRin, IRin, Yin, HIn, LIn;
7 reg IncPC, Read;
8 reg Clock, Clear;
9 reg [31:0] Mdatain;
10 reg [4:0] IRout;
11 wire [31:0] Busout;
12 wire [31:0] R0_out;
13
14 Datapath DUT(.clk(Clock), .Clear(Clear), .Read(Read), .HIn(HIn), .LIn(LIn), .PCin(PCin), .IRin(IRin), .Yin(Yin),
15 .Zin(Zin), .MARin(MARin), .MDRin(MDRin), .IncPC(IncPC), .HIout(HIout), .LOout(LOout), .Zhiout(Zhiout),
16 .Zlowout(Zlowout), .PCout(PCout), .MDRout(MDRout), .InPortout(InPortout), .Cout(Cout), .R0in(R0in), .R1in(R1in), .R2in(R2in), .R3in(R3in), .R4in(R4in), .R5in(R5in), .R6in(R6in), .R7in(R7in),
17 .R0out(R0out), .R1out(R1out), .R2out(R2out), .R3out(R3out), .R4out(R4out), .R5out(R5out), .R6out(R6out), .R7out(R7out), .R8out(R8out), .R9out(R9out), .R10out(R10out), .R11out(R11out), .R12out(R12out), .R13out(R13out),
18 .R0_out(R0_out));
19 always #10 Clock = ~Clock;
20
21 initial begin
22 PCout = 0;
23 Zlowout = 0;
24 MDRout = 0;
25 HIout = 0;
26 LOout = 0;
27 Zhiout = 0;
28 InPortout = 0;
29 Cout = 0;
30 R0out = 0;
31 R1out = 0;
32 R2out = 0;
33 R3out = 0;
34 R4out = 0;
35 R5out = 0;
36 R6out = 0;
37 R7out = 0;
38 R8out = 0;
39 R9out = 0;
40 R10out = 0;
41 R11out = 0;
42 R12out = 0;
43 R13out = 0;
44 R14out = 0;
45 R15out = 0;
46 R0in = 0;
47 R1in = 0;
48 R2in = 0;
49 R3in = 0;
50 R4in = 0;
51 R5in = 0;
52 R6in = 0;
53 R7in = 0;
54 MARin = 0;
55 Zin = 0;
56 PCin = 0;
57 MDRin = 0;
58
59 IRin = 0;
60 Yin = 0;
61 HIn = 0;
62 LIn = 0;
63 IncPC = 0;
64 Read = 0;
65 Clock = 0;
66 Clear = 0;
67 Mdatain = 0;
68 IRout = 0;
69 #10
70 Read = 1;
71 Mdatain = 25;
72 MDRin = 1;
73 MDRout = 1;
74 #20
75 R1in = 1;
76 Read = 0;
77 MDRin = 0;
78 MDRout = 0;
79 Mdatain = 0;
80 #20
81 R1in = 0;
82 Read = 1;
83 Mdatain = 30;
84 MDRin = 1;
85 MDRout = 1;
86 #20
87 Yin = 1;
88 Read = 0;
89 MDRin = 0;
90 MDRout = 0;
91 Mdatain = 0;
92 R1out = 1;
93 IRout = 5'b00011;
94 #20
95 $stop;
96 /*11110 ADD 11001 = 110111*/
97 end
endmodule
```

## Phase 2:

The RAM register had each instruction initialized at address zero, had to be changed for each testbench. Also, for certain instructions I had to preload values into specific registers.

## Load\_TB:

```
1  `timescale 1ns/1ns
2  module load_tb;
3
4  reg Clear;
5  reg PCout, Zlowout, Zhiout, MDRout, HIout, LOout;
6  reg MARin, Zin, PCin, MDRin, IRin, Yin, HIin, LOin;
7  reg IncPC, Read, Write;
8  reg Clock, CONin, Gra, Grb, Grc, Rin, Rout, BAout, Cout, InPortout;
9  wire [31:0] Busout, Zlow_out, Zhi_out, RI_out, RO_out;
10
11  parameter T0 = 4'b0000,
12             T1 = 4'b0001,
13             T2 = 4'b0010,
14             T3 = 4'b0011,
15             T4 = 4'b0100,
16             T5 = 4'b0101,
17             T6 = 4'b0110,
18             T7 = 4'b0111;
19
20  reg [3:0] Present_state = T0;
21
22
23  Datapath DUT (Clock, Clear, Read, Write, HIin, LOin, PCin, IRin, Yin, Zin, MARin, MDRin, IncPC, Out_Portin, In_Portin,
24               HIout, LOout, Zhiout, Zlowout, PCout, MDRout, Cout, InPortout,
25               Gra, Grb, Grc, BAout, Rin, Rout, CONin,
26               Busout, Zlow_out, Zhi_out, RI_out, RO_out);
27
28
29  initial
30  begin
31      Clock = 0;
32      forever #10 Clock = ~ Clock;
33  end
34
35  always @(posedge Clock)
36  begin
37      case (Present_state)
38
39
40
41
42
43
44
45          T1      : #40 Present_state = T2;
46          T2      : #40 Present_state = T3;
47          T3      : #40 Present_state = T4;
48          T4      : #40 Present_state = T5;
49          T5      : #40 Present_state = T6;
50          T6      : #40 Present_state = T7;
51      endcase
52  end
53
54  always @(Present_state)
55  begin
56      case (Present_state)
57      T0: begin
58          PCout <= 1;
59          MARin <= 1;
60          IncPC <= 1;
61          Zin <= 1;
62          Zlowout <= 0;
63          Zhiout <= 0;
64          MDRout <= 0;
65          Cout <= 0;
66          Write <= 0;
67          HIout <= 0;
68          LOout <= 0;
69          InPortout <= 0;
70          #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
71      end
72      T1: begin
73          Zlowout <= 1;
74          PCin <= 1;
75          Read <= 1;
76          MDRin <= 1;
77          #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
78      end
79      T2: begin
80          MDRout <= 1;
81          IRin <= 1;
82          #25 MDRout <= 0; IRin <= 0;
83      end
84      T3: begin
85          Grb <= 1;
86          BAout <= 1;
87          Yin <= 1;
88          #25 Grb <= 0; BAout <= 0; Yin <= 0;
89      end
90      T4: begin
```

```

91         Zin <= 1;
92         Cout <= 1;
93         #25 Cout <= 0; Zin <= 0;
94     end
95     T5: begin
96         Zlowout <= 1;
97         MARin <= 1;
98         #25 MARin <= 0; Zlowout <= 0;
99     end
100    T6: begin
101        Read <= 1;
102        MDRin <= 1;
103        #25 Read <= 0; MDRin <= 0;
104    end
105    T7: begin
106        MDRout <= 1;
107        Gra <= 1;
108        Rin <= 1;
109        #25 MDRout <= 0; Gra <= 0; Rin <= 0;
110    end
111 endcase
112 end
113 endmodule

```

## Loadi\_TB:

```

58 always @(Present_state)
59 begin
60     case (Present_state)
61     T0: begin
62         PCout <= 1;
63         MARin <= 1;
64         IncPC <= 1;
65         Zin <= 1;
66         Zlowout <= 0;
67         Zhiout <= 0;
68         MDRout <= 0;
69         Cout <= 0;
70         Write <= 0;
71         HIout <= 0;
72         LOout <= 0;
73         InPortout <= 0;
74         #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
75     end
76     T1: begin
77         Zlowout <= 1;
78         PCin <= 1;
79         Read <= 1;
80         MDRin <= 1;
81         #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
82     end
83     T2: begin
84         MDRout <= 1;
85         IRin <= 1;
86         #25 MDRout <= 0; IRin <= 0;
87     end
88     T3: begin
89         Grb <= 1;
90         BAout <= 1;
91         Yin <= 1;
92         #25 Grb <= 0; BAout <= 0; Yin <= 0;
93     end
94     T4: begin
95         Zin <= 1;
96         Cout <= 1;
97         #25 Cout <= 0; Zin <= 0;
98     end
99     T5: begin
100        Zlowout <= 1;
101        Gra <= 1;
102        Rin <= 1;

```

## Store\_TB:

```

48 always @(Present_state)
49 begin
50 case (Present_state)
51 T0: begin
52     PCout<= 1;
53     MARin <= 1;
54     IncPC <= 1;
55     Zin <= 1;
56     Zlowout <= 0;
57     Zhiout <= 0;
58     MDRout <= 0;
59     Cout <= 0;
60     Write <= 0;
61     HIout <= 0;
62     LOout <= 0;
63     InPortout <= 0;
64     #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
65 end
66 T1: begin
67     Zlowout<= 1;
68     PCin <= 1;
69     Read <= 1;
70     MDRin <= 1;
71     #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
72 end
73 T2: begin
74     MDRout<= 1;
75     IRin <= 1;
76     #25 MDRout <= 0; IRin <= 0;
77 end
78 T3: begin
79     Grb <= 1;
80     BAout <= 1;
81     Yin <= 1;
82     #25 Grb <= 0; BAout <= 0; Yin <= 0;
83 end
84 T4: begin
85     Zin <= 1;
86     Cout <= 1;
87     #25 Cout <= 0; Zin <= 0;
88 end
89 T5: begin
90     Zlowout <= 1;
91     MARin <= 1;
92     #25 MARin <= 0; Zlowout <= 0;
93 end
94 T6: begin
95     Gra <= 1;
96     BAout <= 1;
97     MDRin <= 1;
98     #25 BAout <= 0; Gra <= 0; MDRin <= 0;
99 end
100 T7: begin
101     Write <= 1;
102 end
103 T8: begin
104     Write <= 0;
105     Read <= 1;
106 end
107 endcase
108 end
109 endmodule
110

```



## ADDI/ANDI/ORI:

```
42 always @(Present_state)
43 begin
44 case (Present_state)
45 T0: begin
46     PCout<= 1;
47     MARin <= 1;
48     IncPC <= 1;
49     Zin <= 1;
50     Zlowout <= 0;
51     Zhiout <= 0;
52     MDRout <= 0;
53     Cout <= 0;
54     Write <= 0;
55     HIout <= 0;
56     LOout <= 0;
57     InPortout <= 0;
58     #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
59 end
60 T1: begin
61     Zlowout<= 1;
62     PCin <= 1;
63     Read <= 1;
64     MDRin <= 1;
65     #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
66 end
67 T2: begin
68     MDRout<= 1;
69     IRin <= 1;
70     #25 MDRout <= 0; IRin <= 0;
71 end
72 T3: begin
73     Grb <= 1;
74     Rout <= 1;
75     Yin <= 1;
76     #25 Grb <= 0; Rout <= 0; Yin <= 0;
77 end
78 T4: begin
79     Zin <= 1;
80     Cout <= 1;
81     #25 Cout <= 0; Zin <= 0;
82 end
83 T5: begin
84     Zlowout <= 1;
85     Gra <= 1;
86     Rin <= 1;
87     #25 Gra <= 0; Rin <= 0; Zlowout <= 0;
```

## BRZR/BRNZ/BRPL/BRMI:

```
46 begin
47   case (Present_state)
48   T0: begin
49     PCout<= 1;
50     MARin <= 1;
51     IncPC <= 1;
52     Zin <= 1;
53     Zlowout <= 0;
54     Zhiout <= 0;
55     MDRout <= 0;
56     Cout <= 0;
57     Write <= 0;
58     HIout <= 0;
59     LOout <= 0;
60     InPortout <= 0;
61     #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
62   end
63   T1: begin
64     Zlowout<= 1;
65     PCin <= 1;
66     Read <= 1;
67     MDRin <= 1;
68     #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
69   end
70   T2: begin
71     MDRout<= 1;
72     IRin <= 1;
73     #25 MDRout <= 0; IRin <= 0;
74   end
75   T3: begin
76     Gra <= 1;
77     Rout <= 1;
78     CONin <= 1;
79     #25 Gra <= 0; Rout <= 0; CONin <= 0;
80   end
81   T4: begin
82     PCout <= 1;
83     Yin <= 1;
84     #25 PCout <= 0; Yin <= 0;
85   end
86   T5: begin
87     Cout <= 1;
88     Zin <= 1;
89     #25 Cout <= 0; Zin <= 0;
90   end
91   T6: begin
92     Zlowout <= 1;
93     if (CON_out) begin
94       PCin <= 1;
95     end
96     #25 PCin <= 0; Zlowout <= 0;
97   end
98   endcase
99 end
100 endmodule
```

## IN/OUT:

```
38 always @(Present_state)|
39 begin
40     case (Present_state)
41     T0: begin
42         PCout<= 1;
43         MARin <= 1;
44         IncPC <= 1;
45         Zin <= 1;
46         Zlowout <= 0;
47         Zhiout <= 0;
48         MDRout <= 0;
49         Cout <= 0;
50         Write <= 0;
51         HIout <= 0;
52         LOout <= 0;
53         InPortout <= 0;
54         #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
55     end
56     T1: begin
57         Zlowout<= 1;
58         PCin <= 1;
59         Read <= 1;
60         MDRin <= 1;
61         #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
62     end
63     T2: begin
64         MDRout<= 1;
65         IRin <= 1;
66         #25 MDRout <= 0; IRin <= 0;
67     end
68     T3: begin
69         InPortout <= 1;
70         Gra <= 1;
71         Rin <= 1;
72         #25 Gra <= 0; Rin <= 0; InPortout <= 0;
73     end
74 endcase
```

## JR/JAL:

```
40 always @(Present_state)// do the required job in each state
41 begin
42     case (Present_state) //assert the required signals in each clock cycl
43     T0: begin //see if you need to de-assertthese signals
44         PCout<= 1;
45         MARin <= 1;
46         IncPC <= 1;
47         Zin <= 1;
48         Zlowout <= 0;
49         Zhiout <= 0;
50         MDRout <= 0;
51         Cout <= 0;
52         Write <= 0;
53         HIout <= 0;
54         LOout <= 0;
55         InPortout <= 0;
56         #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
57     end
58     T1: begin
59         Zlowout<= 1;
60         PCin <= 1;
61         Read <= 1;
62         MDRin <= 1;
63         #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
64     end
65     T2: begin
66         MDRout<= 1;
67         IRin <= 1;
68         #25 MDRout <= 0; IRin <= 0;
69     end
70     T3: begin
71         PCout <= 1;
72         RI5in <= 1;
73         #25 RI5in <= 0; PCout <= 0;
74     end
75     T4: begin
76         Rout <= 1;
77         PCin <= 1;
78         Gra <= 1;
79         #25 Gra <= 0; PCin <= 0; Rout <= 0;
80     end
81 endcase
```

## MFHI/MFLO:

```
38 always @(Present_state)
39 begin
40     case (Present_state)
41     T0: begin
42         PCout<= 1;
43         MARin <= 1;
44         IncPC <= 1;
45         Zin <= 1;
46         Zlowout <= 0;
47         Zhiout <= 0;
48         MDRout <= 0;
49         Cout <= 0;
50         Write <= 0;
51         HIout <= 0;
52         LOout <= 0;
53         InPortout <= 0;
54         #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
55     end
56     T1: begin
57         Zlowout<= 1;
58         FCin <= 1;
59         Read <= 1;
60         MDRin <= 1;
61         #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
62     end
63     T2: begin
64         MDRout<= 1;
65         IRin <= 1;
66         #25 MDRout <= 0; IRin <= 0;
67     end
68     T3: begin
69         LOout <= 1;
70         Gra <= 1;
71         Rin <= 1;
72         #25 Gra <= 0; Rin <= 0; LOout <= 0;
73     end
74     endcase
75 end
76 endmodule
77
```

## Phase 3:

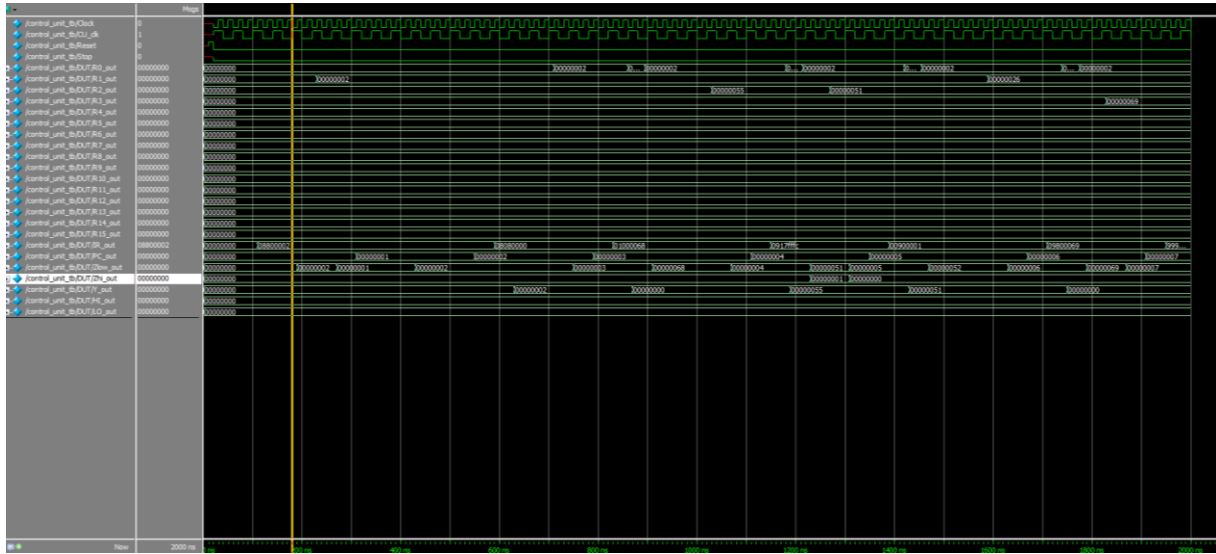
The RAM file has all the instructions preloaded at their specific address locations, the control unit tb will cycle through them.

## Control\_Unit\_TB:

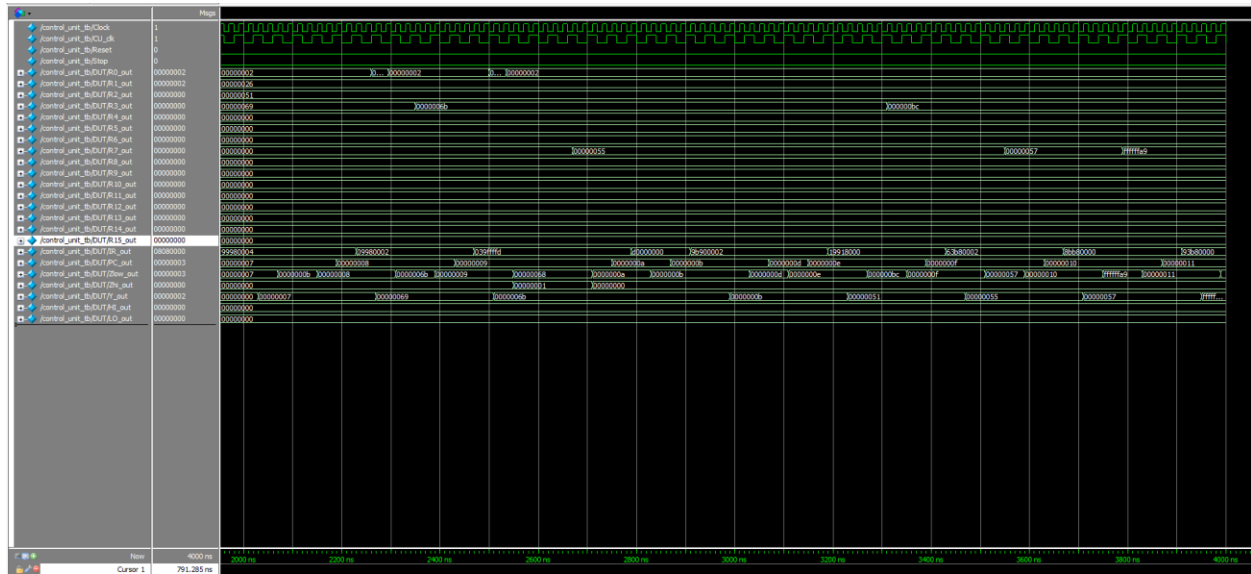
```
1 `timescale 1ns/1ns
2 module control_unit_tb;
3
4     reg Clock, CU_clk, Reset, Stop;
5     wire [31:0] Busout;
6
7     Datapath DUT (Clock, CU_clk, Reset, Stop, Busout);
8
9     initial
10     begin
11         #10
12         Reset = 1;
13         #10
14         Reset = 0;
15     end
16
17     initial
18     begin
19         #20
20         Clock = 0;
21         forever #10 Clock = ~ Clock;
22     end
23
24     initial
25     begin
26         #20
27         CU_clk = 1;
28         forever #20 CU_clk = ~ CU_clk;
29     end
30
31     initial
32     begin
33         #20
34         Reset = 0;
35         Stop = 0;
36     end
37
38 endmodule
39
40
```

Address	ORG	0	
0	ldi	R1, 2	; R1 = 2
	ldi	R0, 0(R1)	; R0 = 2
	ld	R2, \$68	; R2 = (\$68) = \$55
	ldi	R2, -4(R2)	; R2 = \$51
	ld	R1, 1(R2)	; R1 = (\$52) = \$26
	ldi	R3, \$69	; R3 = \$69
	brmi	R3, 4	; continue with the next instruction (will not branch)
	ldi	R3, 2(R3)	; R3 = \$6B
	ld	R7, -3(R3)	; R7 = (\$6B - 3) = \$55
	nop		
	brpl	R7, 2	; continue with the instruction at "target" (will branch)
	ldi	R2, 5(R0)	; this instruction will not execute
	ldi	R3, 2(R1)	; this instruction will not execute
target	add	R3, R2, R3	; R3 = \$BC
	addi	R7, R7, 2	; R7 = \$57
	neg	R7, R7	; R7 = \$FFFFFFA9
	not	R7, R7	; R7 = \$56
	andi	R7, R7, \$0F	; R7 = 6
	ror	R1, R1, R0	; R1 = \$80000009
	ori	R7, R1, \$1C	; R7 = \$8000001D
	shra	R7, R7, R0	; R7 = \$E0000007
	shr	R2, R3, R0	; R2 = \$2F
	st	\$52, R2	; (\$52) = \$2F    new value in memory with address \$52
	rol	R2, R2, R0	; R2 = \$BC
	or	R2, R3, R0	; R2 = \$BE
	and	R1, R2, R1	; R1 = \$8
	st	\$60(R1), R3	; (\$68) = \$BC    new value in memory with address \$68
	sub	R3, R2, R3	; R3 = 2
	shl	R1, R2, R0	; R1 = \$2F8
	ldi	R4, 6	; R4 = 6
	ldi	R5, \$32	; R5 = \$32
	mul	R5, R4	; HI = 0; LO = \$12C
	mfhi	R7	; R7 = 0
	mflo	R6	; R6 = \$12C
	div	R5, R4	; HI = 2, LO = 8
	ldi	R8, -1(R4)	; R8 = 5    setting up argument registers
	ldi	R9, -19(R5)	; R9 = \$1F    R8, R9, R10, and R11
	ldi	R10, 0(R6)	; R10 = \$12C
	ldi	R11, 0(R7)	; R11 = 0
	jal	R10	; address of subroutine subA in R10 - return address in R15
\$28	halt		; upon return, the program halts
\$12C	subA	ORG	\$12C    ; procedure subA
	add	R13, R8, R10	; R12 and R13 are return value registers
	sub	R12, R9, R11	; R13 = \$131, R12 = \$1F
	sub	R13, R13, R12	; R13 = \$112
	jr	R15	; return from procedure

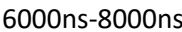
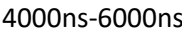
## Simulations

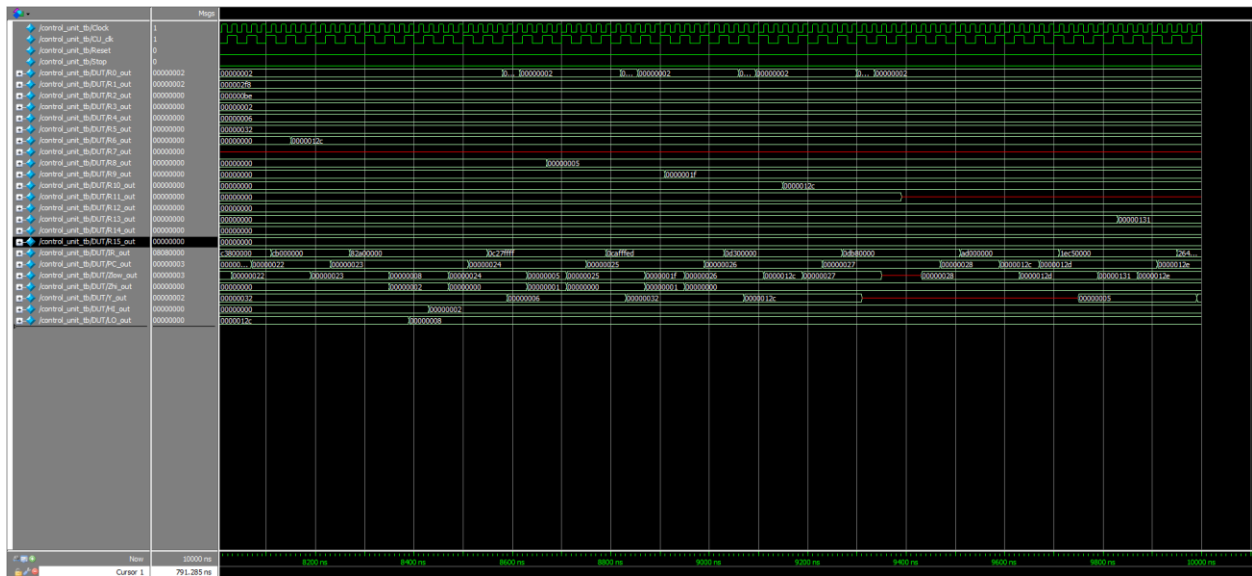


0ns-2000ns

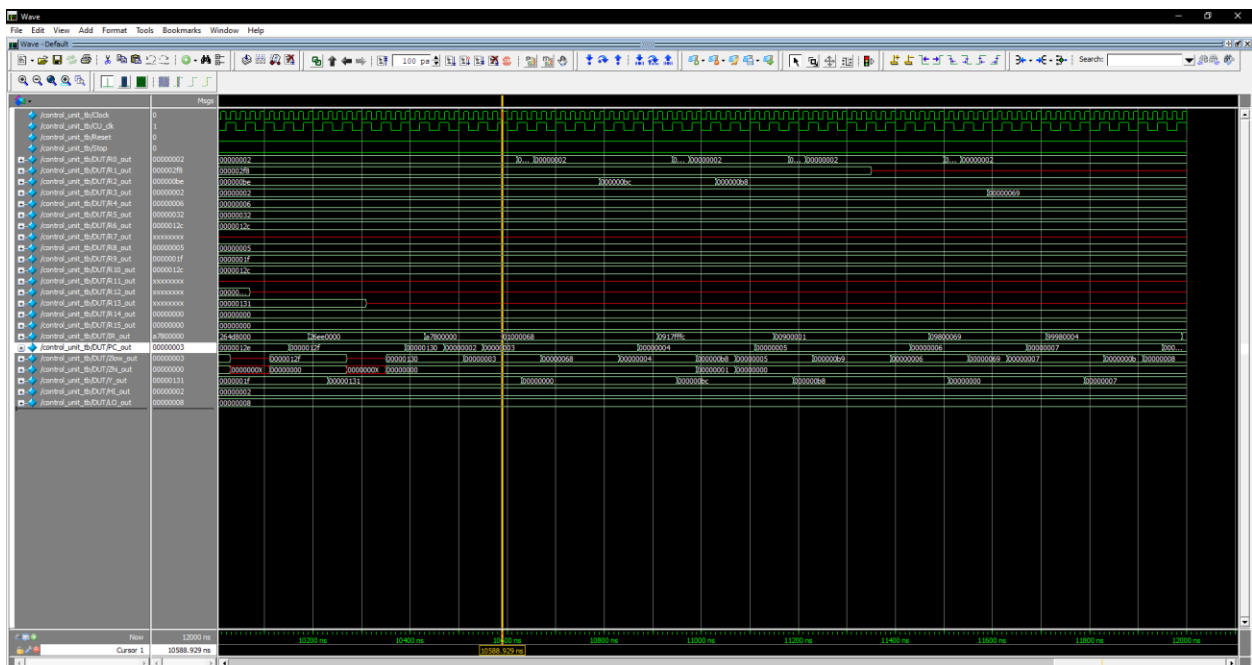


2000ns-4000ns





8000ns-10000ns



10000ns-10588ns