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

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.

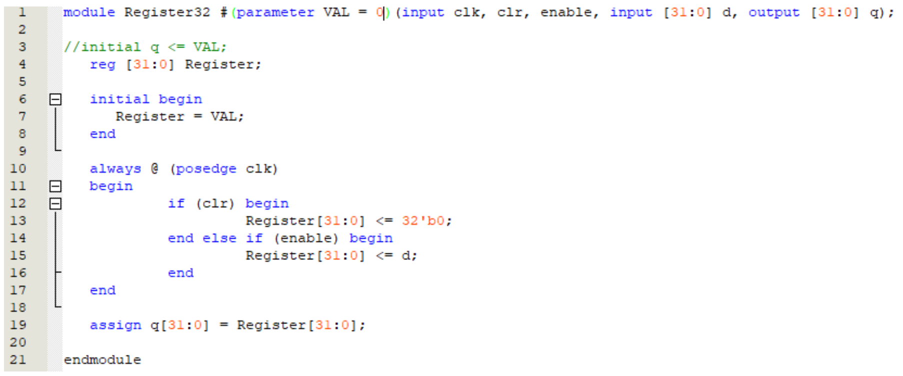
Conclusion

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

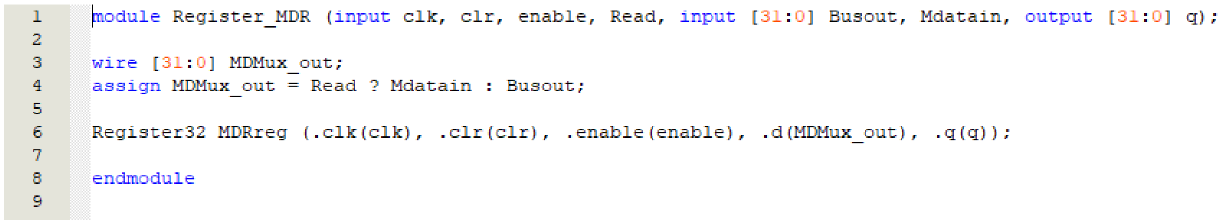
Code

Register_32:



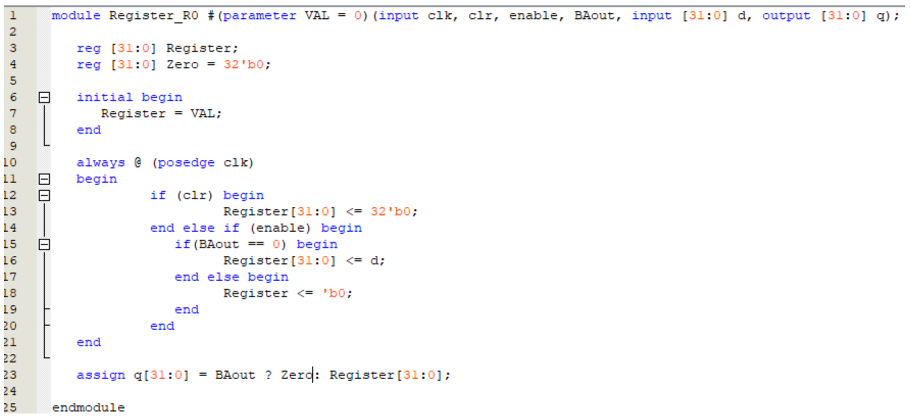
```
module Register32 #(parameter VAL = 0)(input clk, clr, enable, input [31:0] d, output [31:0] q);

//initial q <= VAL;
   reg [31:0] Register;

   initial begin
      Register = VAL;
   end

   always @ (posedge clk)
   begin
         if (clr) begin
               Register[31:0] <= 32'b0;
         end else if (enable) begin
               Register[31:0] <= d;
         end
   end

   assign q[31:0] = Register[31:0];

endmodule
```

Register_MDR:



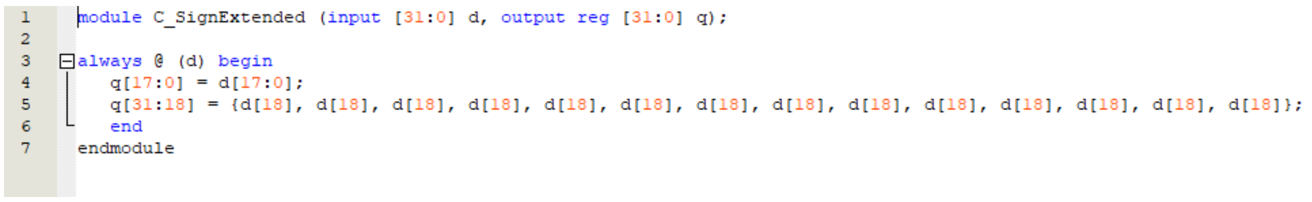
```
module Register_MDR (input clk, clr, enable, Read, input [31:0] Busout, Mdatain, output [31:0] q);

wire [31:0] MDMux_out;
assign MDMux_out = Read ? Mdatain : Busout;

Register32 MDRreg (.clk(clk), .clr(clr), .enable(enable), .d(MDMux_out), .q(q));

endmodule

```

Register_R0:



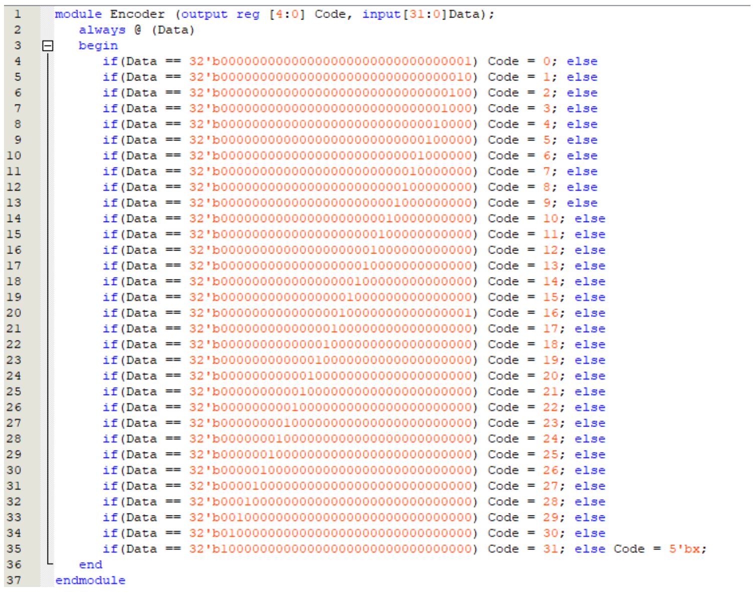
```
module Register_R0 #(parameter VAL = 0)(input clk, clr, enable, BAout, input [31:0] d, output [31:0] q);

   reg [31:0] Register;
   reg [31:0] Zero = 32'b0;

   initial begin
      Register = VAL;
   end

   always @ (posedge clk)
   begin
         if (clr) begin
               Register[31:0] <= 32'b0;
         end else if (enable) begin
            if(BAout == 0) begin
               Register[31:0] <= d;
            end else begin
               Register <= 'b0;
            end
         end
   end

   assign q[31:0] = BAout ? Zero: Register[31:0];

endmodule
```

C Sign Extension:



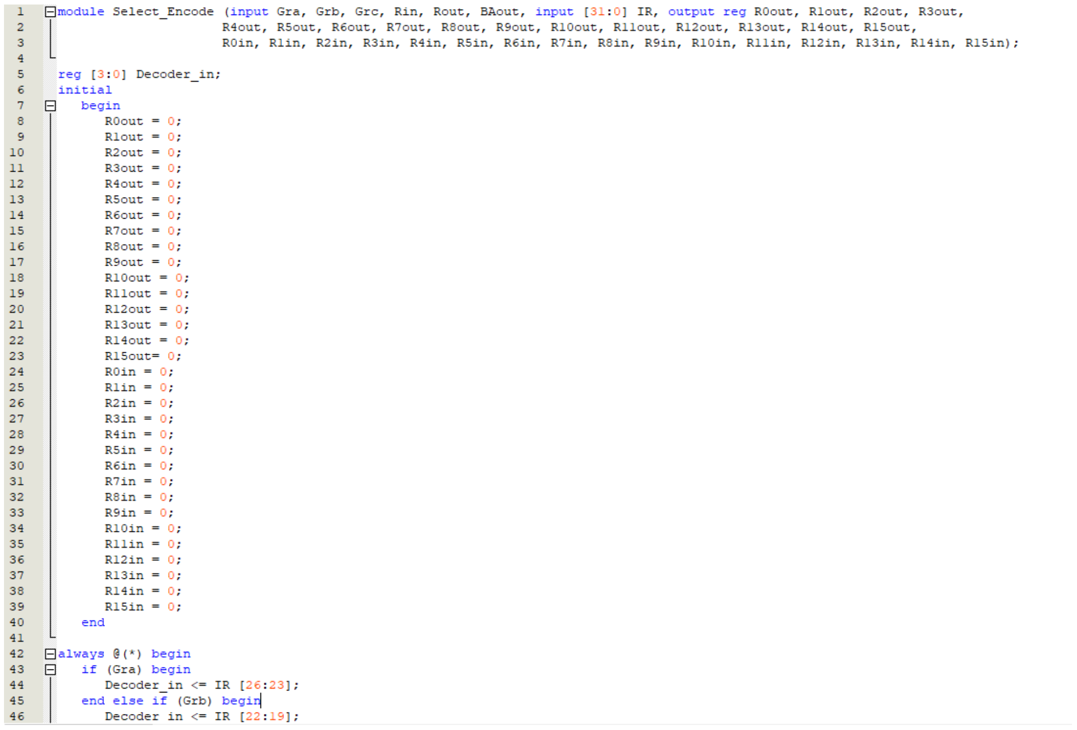
```verilog
module C_SignExtended (input [31:0] d, output reg [31:0] q);

always @ (d) begin
    q[17:0] = d[17:0];
    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]};
    end
endmodule
```

Encoder:

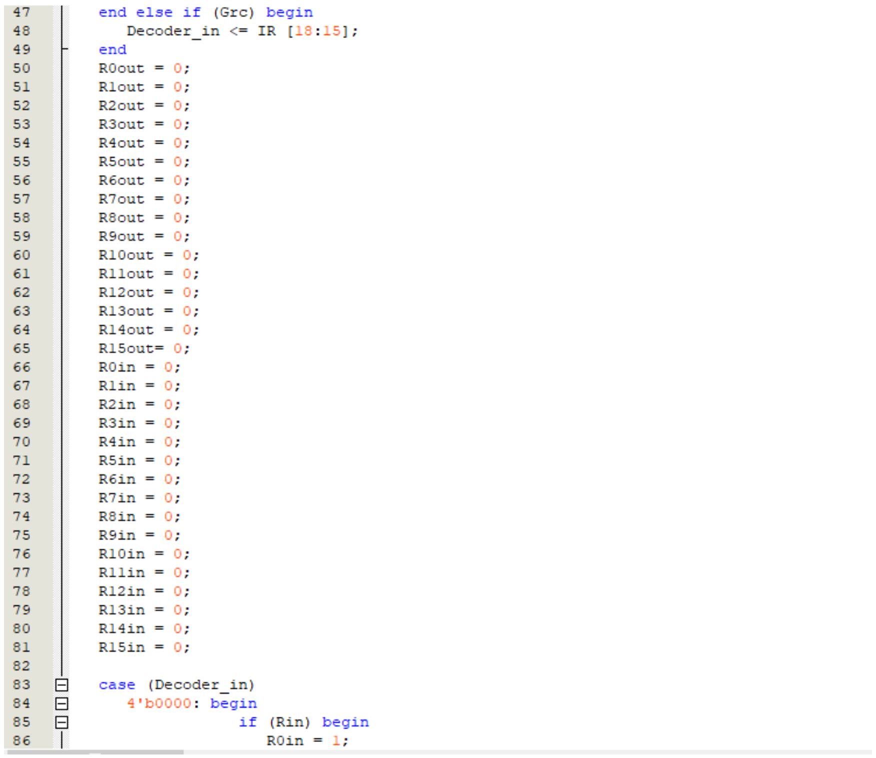


```verilog
module Encoder (output reg [4:0] Code, input[31:0]Data);
   always @ (Data)
   begin
      if(Data == 32'b00000000000000000000000000000001) Code = 0; else
      if(Data == 32'b00000000000000000000000000000010) Code = 1; else
      if(Data == 32'b00000000000000000000000000000100) Code = 2; else
      if(Data == 32'b00000000000000000000000000001000) Code = 3; else
      if(Data == 32'b00000000000000000000000000010000) Code = 4; else
      if(Data == 32'b00000000000000000000000000100000) Code = 5; else
      if(Data == 32'b00000000000000000000000001000000) Code = 6; else
      if(Data == 32'b00000000000000000000000010000000) Code = 7; else
      if(Data == 32'b00000000000000000000000100000000) Code = 8; else
      if(Data == 32'b00000000000000000000001000000000) Code = 9; else
      if(Data == 32'b00000000000000000000010000000000) Code = 10; else
      if(Data == 32'b00000000000000000000100000000000) Code = 11; else
      if(Data == 32'b00000000000000000001000000000000) Code = 12; else
      if(Data == 32'b00000000000000000010000000000000) Code = 13; else
      if(Data == 32'b00000000000000000100000000000000) Code = 14; else
      if(Data == 32'b00000000000000001000000000000000) Code = 15; else
      if(Data == 32'b00000000000000010000000000000001) Code = 16; else
      if(Data == 32'b00000000000000100000000000000000) Code = 17; else
      if(Data == 32'b00000000000001000000000000000000) Code = 18; else
      if(Data == 32'b00000000000010000000000000000000) Code = 19; else
      if(Data == 32'b00000000000100000000000000000000) Code = 20; else
      if(Data == 32'b00000000001000000000000000000000) Code = 21; else
      if(Data == 32'b00000000010000000000000000000000) Code = 22; else
      if(Data == 32'b00000000100000000000000000000000) Code = 23; else
      if(Data == 32'b00000001000000000000000000000000) Code = 24; else
      if(Data == 32'b00000010000000000000000000000000) Code = 25; else
      if(Data == 32'b00000100000000000000000000000000) Code = 26; else
      if(Data == 32'b00001000000000000000000000000000) Code = 27; else
      if(Data == 32'b00010000000000000000000000000000) Code = 28; else
      if(Data == 32'b00100000000000000000000000000000) Code = 29; else
      if(Data == 32'b01000000000000000000000000000000) Code = 30; else
      if(Data == 32'b10000000000000000000000000000000) Code = 31; else Code = 5'bx;
   end
endmodule
```

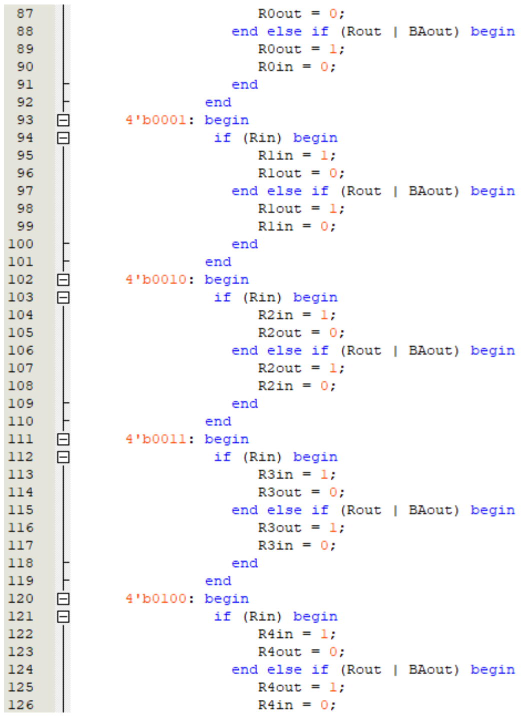
Select and Encode:



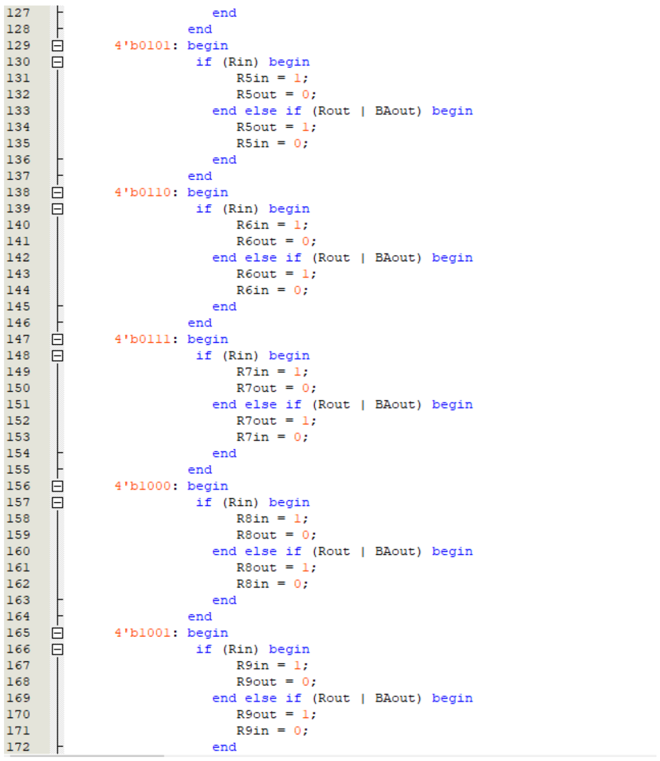
```verilog
module Select_Encode (input Gra, Grb, Grc, Rin, Rout, BAout, input [31:0] IR, output reg R0out, R1out, R2out, R3out,
                      R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out,
                      R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in, R8in, R9in, R10in, R11in, R12in, R13in, R14in, R15in);

reg [3:0] Decoder_in;
initial
   begin
      R0out = 0;
      R1out = 0;
      R2out = 0;
      R3out = 0;
      R4out = 0;
      R5out = 0;
      R6out = 0;
      R7out = 0;
      R8out = 0;
      R9out = 0;
      R10out = 0;
      R11out = 0;
      R12out = 0;
      R13out = 0;
      R14out = 0;
      R15out= 0;
      R0in = 0;
      R1in = 0;
      R2in = 0;
      R3in = 0;
      R4in = 0;
      R5in = 0;
      R6in = 0;
      R7in = 0;
      R8in = 0;
      R9in = 0;
      R10in = 0;
      R11in = 0;
      R12in = 0;
      R13in = 0;
      R14in = 0;
      R15in = 0;
   end

always @(*) begin
   if (Gra) begin
      Decoder_in <= IR [26:23];
   end else if (Grb) begin
      Decoder_in <= IR [22:19];
```



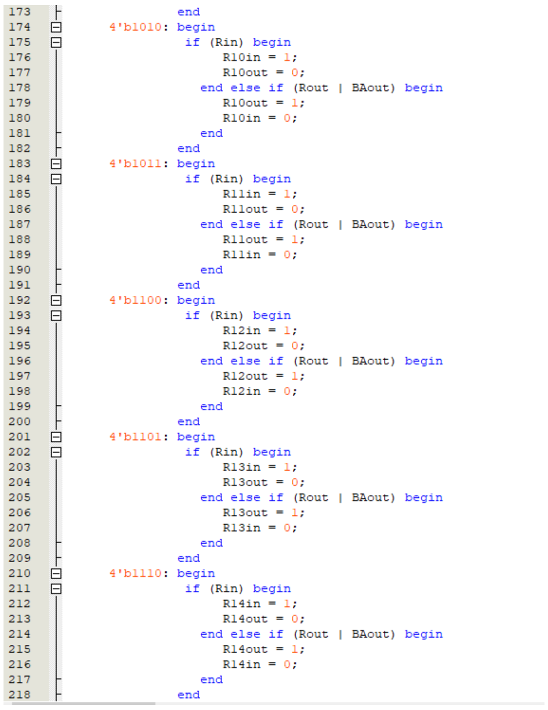
```
end else if (Grc) begin
   Decoder_in <= IR [18:15];
end
R0out = 0;
R1out = 0;
R2out = 0;
R3out = 0;
R4out = 0;
R5out = 0;
R6out = 0;
R7out = 0;
R8out = 0;
R9out = 0;
R10out = 0;
R11out = 0;
R12out = 0;
R13out = 0;
R14out = 0;
R15out= 0;
R0in = 0;
R1in = 0;
R2in = 0;
R3in = 0;
R4in = 0;
R5in = 0;
R6in = 0;
R7in = 0;
R8in = 0;
R9in = 0;
R10in = 0;
R11in = 0;
R12in = 0;
R13in = 0;
R14in = 0;
R15in = 0;

case (Decoder_in)
   4'b0000: begin
               if (Rin) begin
                  R0in = 1;
```



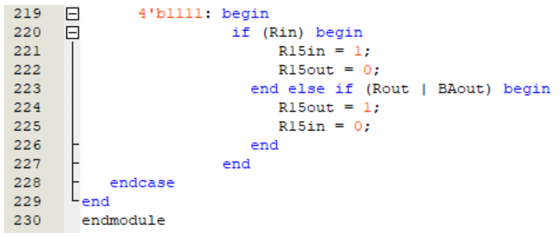
```
                  R0out = 0;
               end else if (Rout | BAout) begin
                  R0out = 1;
                  R0in = 0;
               end
            end
   4'b0001: begin
              if (Rin) begin
                  R1in = 1;
                  R1out = 0;
               end else if (Rout | BAout) begin
                  R1out = 1;
                  R1in = 0;
               end
            end
   4'b0010: begin
              if (Rin) begin
                  R2in = 1;
                  R2out = 0;
               end else if (Rout | BAout) begin
                  R2out = 1;
                  R2in = 0;
               end
            end
   4'b0011: begin
              if (Rin) begin
                  R3in = 1;
                  R3out = 0;
               end else if (Rout | BAout) begin
                  R3out = 1;
                  R3in = 0;
               end
            end
   4'b0100: begin
              if (Rin) begin
                  R4in = 1;
                  R4out = 0;
               end else if (Rout | BAout) begin
                  R4out = 1;
                  R4in = 0;
```



```
               end
            end
      4'b0101: begin
                if (Rin) begin
                     R5in = 1;
                     R5out = 0;
                  end else if (Rout | BAout) begin
                     R5out = 1;
                     R5in = 0;
                  end
            end
      4'b0110: begin
                if (Rin) begin
                     R6in = 1;
                     R6out = 0;
                  end else if (Rout | BAout) begin
                     R6out = 1;
                     R6in = 0;
                  end
            end
      4'b0111: begin
                if (Rin) begin
                     R7in = 1;
                     R7out = 0;
                  end else if (Rout | BAout) begin
                     R7out = 1;
                     R7in = 0;
                  end
            end
      4'b1000: begin
                if (Rin) begin
                     R8in = 1;
                     R8out = 0;
                  end else if (Rout | BAout) begin
                     R8out = 1;
                     R8in = 0;
                  end
            end
      4'b1001: begin
                if (Rin) begin
                     R9in = 1;
                     R9out = 0;
                  end else if (Rout | BAout) begin
                     R9out = 1;
                     R9in = 0;
                  end
```

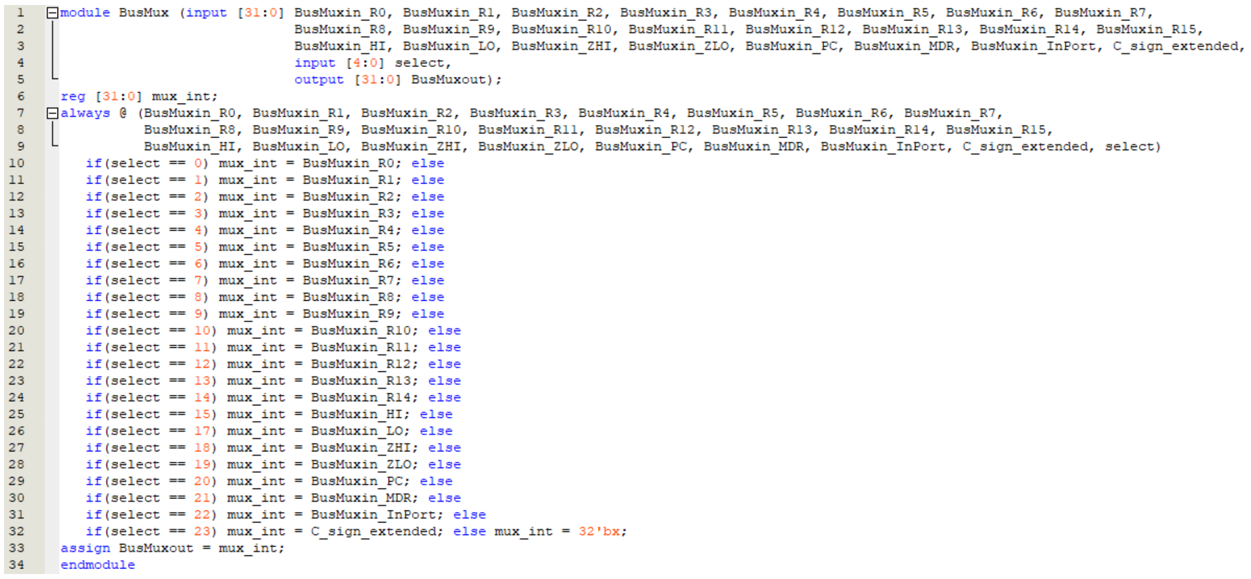


```
               end
      4'b1010: begin
                if (Rin) begin
                     R10in = 1;
                     R10out = 0;
                  end else if (Rout | BAout) begin
                     R10out = 1;
                     R10in = 0;
                  end
               end
      4'b1011: begin
                if (Rin) begin
                     R11in = 1;
                     R11out = 0;
                  end else if (Rout | BAout) begin
                     R11out = 1;
                     R11in = 0;
                  end
               end
      4'b1100: begin
                if (Rin) begin
                     R12in = 1;
                     R12out = 0;
                  end else if (Rout | BAout) begin
                     R12out = 1;
                     R12in = 0;
                  end
               end
      4'b1101: begin
                if (Rin) begin
                     R13in = 1;
                     R13out = 0;
                  end else if (Rout | BAout) begin
                     R13out = 1;
                     R13in = 0;
                  end
               end
      4'b1110: begin
                if (Rin) begin
                     R14in = 1;
                     R14out = 0;
                  end else if (Rout | BAout) begin
                     R14out = 1;
                     R14in = 0;
                  end
               end
```



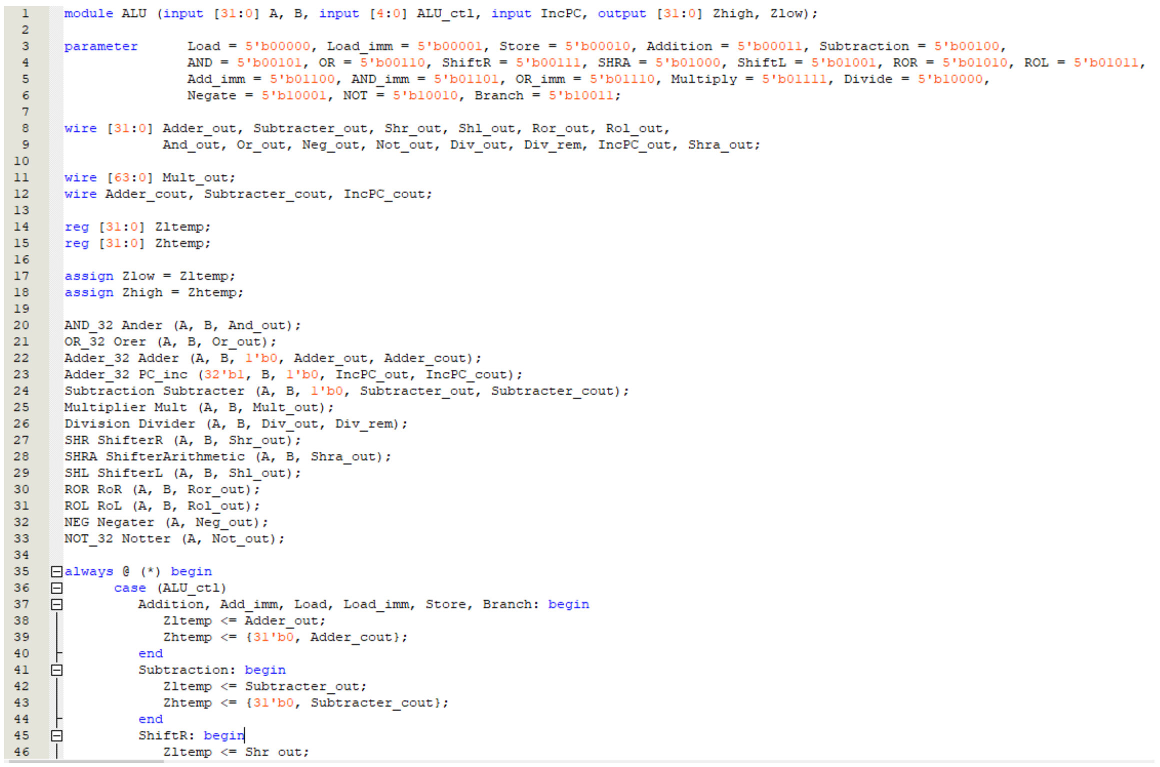
```
      4'b1111: begin
                if (Rin) begin
                     R15in = 1;
                     R15out = 0;
                  end else if (Rout | BAout) begin
                     R15out = 1;
                     R15in = 0;
                  end
               end
   endcase
end
endmodule
```

BusMux:

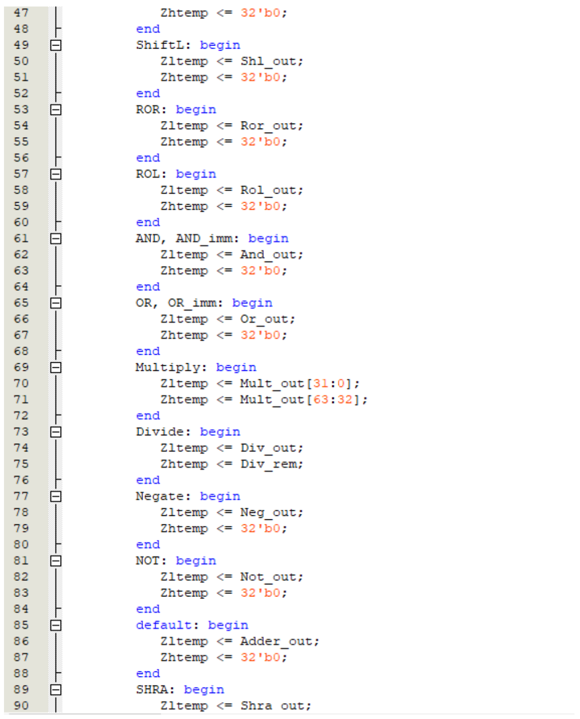


```
module BusMux (input [31:0] BusMuxin_R0, BusMuxin_R1, BusMuxin_R2, BusMuxin_R3, BusMuxin_R4, BusMuxin_R5, BusMuxin_R6, BusMuxin_R7,
                            BusMuxin_R8, BusMuxin_R9, BusMuxin_R10, BusMuxin_R11, BusMuxin_R12, BusMuxin_R13, BusMuxin_R14, BusMuxin_R15,
                          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

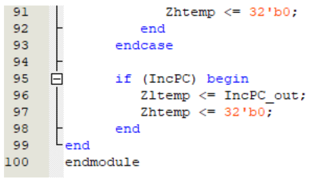
ALU:



```
module ALU (input [31:0] A, B, input [4:0] ALU_ctl, input IncPC, output [31:0] Zhigh, Zlow);

parameter      Load = 5'b00000, Load_imm = 5'b00001, Store = 5'b00010, Addition = 5'b00011, Subtraction = 5'b00100,
               AND = 5'b00101, OR = 5'b00110, ShiftR = 5'b00111, SHRA = 5'b01000, ShiftL = 5'b01001, ROR = 5'b01010, ROL = 5'b01011,
               Add_imm = 5'b01100, AND_imm = 5'b01101, OR_imm = 5'b01110, Multiply = 5'b01111, Divide = 5'b10000,
               Negate = 5'b10001, NOT = 5'b10010, Branch = 5'b10011;

wire [31:0] Adder_out, Subtracter_out, Shr_out, Shl_out, Ror_out, Rol_out,
            And_out, Or_out, Neg_out, Not_out, Div_out, Div_rem, IncPC_out, Shra_out;

wire [63:0] Mult_out;
wire Adder_cout, Subtracter_cout, IncPC_cout;

reg [31:0] Zltemp;
reg [31:0] Zhtemp;

assign Zlow = Zltemp;
assign Zhigh = Zhtemp;

AND_32 Ander (A, B, And_out);
OR_32 Orer (A, B, Or_out);
Adder_32 Adder (A, B, 1'b0, Adder_out, Adder_cout);
Adder_32 PC_inc (32'b1, B, 1'b0, IncPC_out, IncPC_cout);
Subtraction Subtracter (A, B, 1'b0, Subtracter_out, Subtracter_cout);
Multiplier Mult (A, B, Mult_out);
Division Divider (A, B, Div_out, Div_rem);
SHR ShifterR (A, B, Shr_out);
SHRA ShifterArithmetic (A, B, Shra_out);
SHL ShifterL (A, B, Shl_out);
ROR RoR (A, B, Ror_out);
ROL RoL (A, B, Rol_out);
NEG Negater (A, Neg_out);
NOT_32 Notter (A, Not_out);

always @ (*) begin
      case (ALU_ctl)
         Addition, Add_imm, Load, Load_imm, Store, Branch: begin
            Zltemp <= Adder_out;
            Zhtemp <= {31'b0, Adder_cout};
         end
         Subtraction: begin
            Zltemp <= Subtracter_out;
            Zhtemp <= {31'b0, Subtracter_cout};
         end
         ShiftR: begin
            Zltemp <= Shr_out;
```

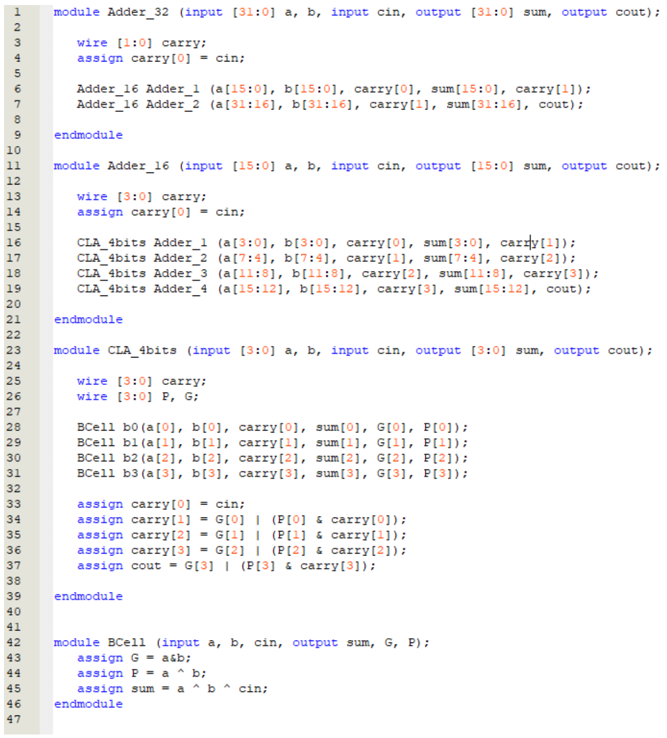


```
            Zhtemp <= 32'b0;
         end
         ShiftL: begin
            Zltemp <= Shl_out;
            Zhtemp <= 32'b0;
         end
         ROR: begin
            Zltemp <= Ror_out;
            Zhtemp <= 32'b0;
         end
         ROL: begin
            Zltemp <= Rol_out;
            Zhtemp <= 32'b0;
         end
         AND, AND_imm: begin
            Zltemp <= And_out;
            Zhtemp <= 32'b0;
         end
         OR, OR_imm: begin
            Zltemp <= Or_out;
            Zhtemp <= 32'b0;
         end
         Multiply: begin
            Zltemp <= Mult_out[31:0];
            Zhtemp <= Mult_out[63:32];
         end
         Divide: begin
            Zltemp <= Div_out;
            Zhtemp <= Div_rem;
         end
         Negate: begin
            Zltemp <= Neg_out;
            Zhtemp <= 32'b0;
         end
         NOT: begin
            Zltemp <= Not_out;
            Zhtemp <= 32'b0;
         end
         default: begin
            Zltemp <= Adder_out;
            Zhtemp <= 32'b0;
         end
         SHRA: begin
            Zltemp <= Shra_out;
```



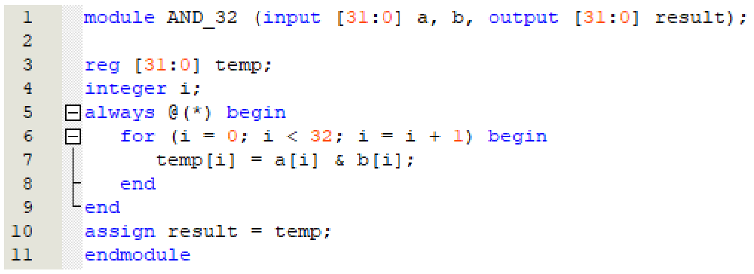
```verilog
            Zhtemp <= 32'b0;
         end
      endcase

      if (IncPC) begin
         Z1temp <= IncPC_out;
         Zhtemp <= 32'b0;
      end
end
endmodule
```

Adder_32:



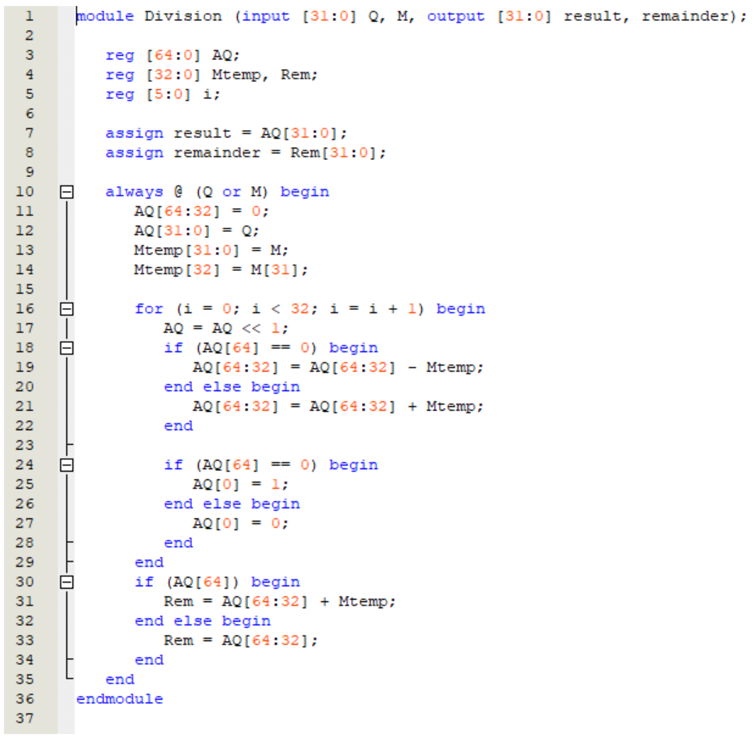
```verilog
module Adder_32 (input [31:0] a, b, input cin, output [31:0] sum, output cout);

   wire [1:0] carry;
   assign carry[0] = cin;

   Adder_16 Adder_1 (a[15:0], b[15:0], carry[0], sum[15:0], carry[1]);
   Adder_16 Adder_2 (a[31:16], b[31:16], carry[1], sum[31:16], cout);

endmodule

module Adder_16 (input [15:0] a, b, input cin, output [15:0] sum, output cout);

   wire [3:0] carry;
   assign carry[0] = cin;

   CLA_4bits Adder_1 (a[3:0], b[3:0], carry[0], sum[3:0], carry[1]);
   CLA_4bits Adder_2 (a[7:4], b[7:4], carry[1], sum[7:4], carry[2]);
   CLA_4bits Adder_3 (a[11:8], b[11:8], carry[2], sum[11:8], carry[3]);
   CLA_4bits Adder_4 (a[15:12], b[15:12], carry[3], sum[15:12], cout);

endmodule

module CLA_4bits (input [3:0] a, b, input cin, output [3:0] sum, output cout);

   wire [3:0] carry;
   wire [3:0] P, G;

   BCell b0(a[0], b[0], carry[0], sum[0], G[0], P[0]);
   BCell b1(a[1], b[1], carry[1], sum[1], G[1], P[1]);
   BCell b2(a[2], b[2], carry[2], sum[2], G[2], P[2]);
   BCell b3(a[3], b[3], carry[3], sum[3], G[3], P[3]);

   assign carry[0] = cin;
   assign carry[1] = G[0] | (P[0] & carry[0]);
   assign carry[2] = G[1] | (P[1] & carry[1]);
   assign carry[3] = G[2] | (P[2] & carry[2]);
   assign cout = G[3] | (P[3] & carry[3]);

endmodule


module BCell (input a, b, cin, output sum, G, P);
   assign G = a&b;
   assign P = a ^ b;
   assign sum = a ^ b ^ cin;
endmodule

```

32-bit AND:



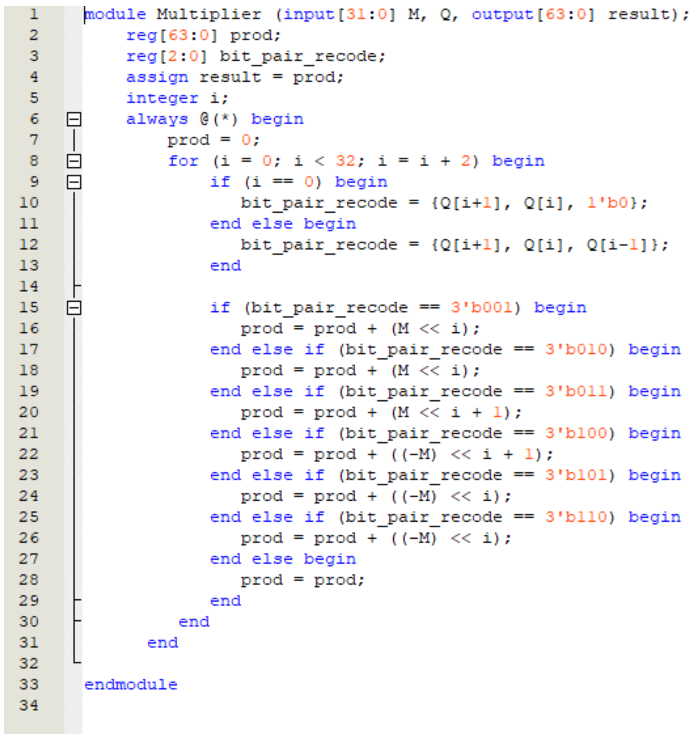
```verilog
module AND_32 (input [31:0] a, b, output [31:0] result);

reg [31:0] temp;
integer i;
always @(*) begin
    for (i = 0; i < 32; i = i + 1) begin
        temp[i] = a[i] & b[i];
    end
end
assign result = temp;
endmodule
```

Division:



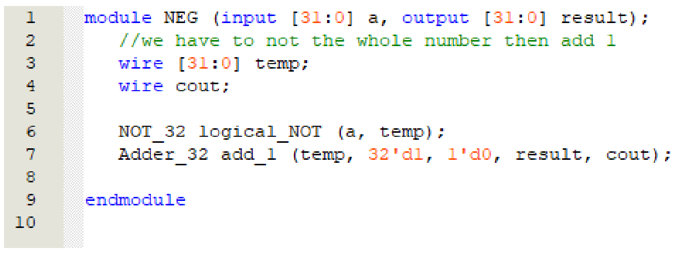
```verilog
module Division (input [31:0] Q, M, output [31:0] result, remainder);

   reg [64:0] AQ;
   reg [32:0] Mtemp, Rem;
   reg [5:0] i;

   assign result = AQ[31:0];
   assign remainder = Rem[31:0];

   always @ (Q or M) begin
      AQ[64:32] = 0;
      AQ[31:0] = Q;
      Mtemp[31:0] = M;
      Mtemp[32] = M[31];

      for (i = 0; i < 32; i = i + 1) begin
         AQ = AQ << 1;
         if (AQ[64] == 0) begin
            AQ[64:32] = AQ[64:32] - Mtemp;
         end else begin
            AQ[64:32] = AQ[64:32] + Mtemp;
         end

         if (AQ[64] == 0) begin
            AQ[0] = 1;
         end else begin
            AQ[0] = 0;
         end
      end
      if (AQ[64]) begin
         Rem = AQ[64:32] + Mtemp;
      end else begin
         Rem = AQ[64:32];
      end
   end
endmodule
```

Multiplier:



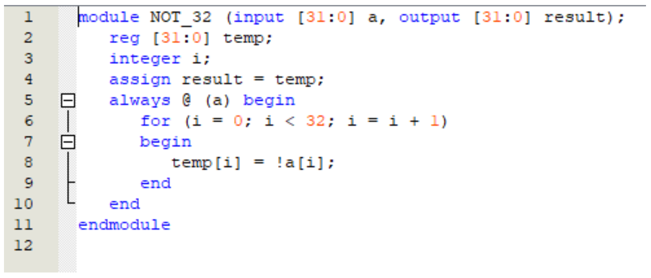
```verilog
module Multiplier (input[31:0] M, Q, output[63:0] result);
    reg[63:0] prod;
    reg[2:0] bit_pair_recode;
    assign result = prod;
    integer i;
    always @(*) begin
        prod = 0;
        for (i = 0; i < 32; i = i + 2) begin
            if (i == 0) begin
                bit_pair_recode = {Q[i+1], Q[i], 1'b0};
            end else begin
                bit_pair_recode = {Q[i+1], Q[i], Q[i-1]};
            end

            if (bit_pair_recode == 3'b001) begin
                prod = prod + (M << i);
            end else if (bit_pair_recode == 3'b010) begin
                prod = prod + (M << i);
            end else if (bit_pair_recode == 3'b011) begin
                prod = prod + (M << i + 1);
            end else if (bit_pair_recode == 3'b100) begin
                prod = prod + ((-M) << i + 1);
            end else if (bit_pair_recode == 3'b101) begin
                prod = prod + ((-M) << i);
            end else if (bit_pair_recode == 3'b110) begin
                prod = prod + ((-M) << i);
            end else begin
                prod = prod;
            end
        end
    end

endmodule
```

32-bit NEG:



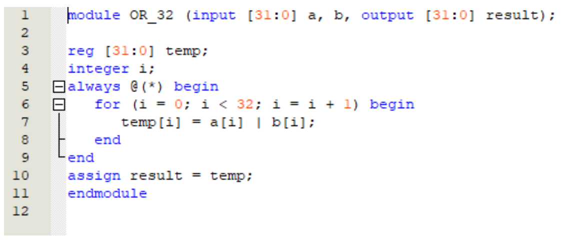
```
module NEG (input [31:0] a, output [31:0] result);
   //we have to not the whole number then add 1
   wire [31:0] temp;
   wire cout;

   NOT_32 logical_NOT (a, temp);
   Adder_32 add_1 (temp, 32'd1, 1'd0, result, cout);

endmodule
```

32-bit NOT:



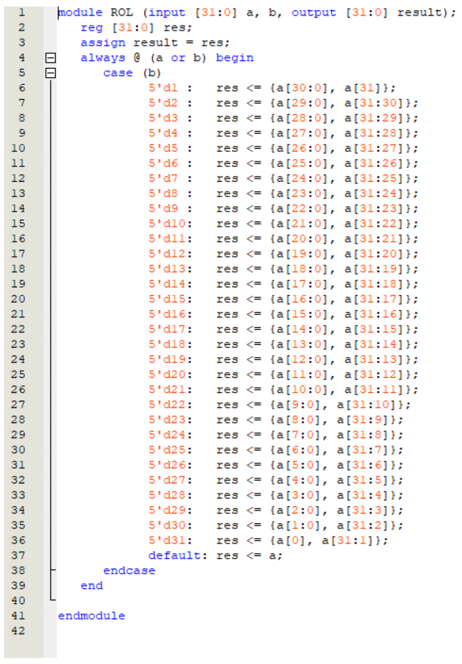
```
module NOT_32 (input [31:0] a, output [31:0] result);
   reg [31:0] temp;
   integer i;
   assign result = temp;
   always @ (a) begin
      for (i = 0; i < 32; i = i + 1)
      begin
         temp[i] = !a[i];
      end
   end
endmodule
```

32-bit OR:



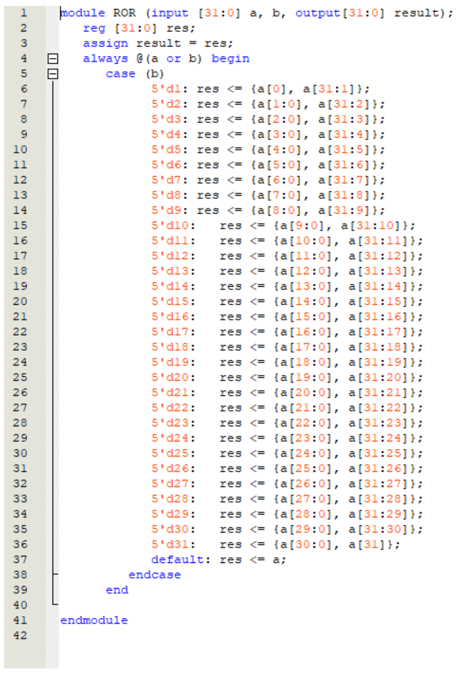
```
module OR_32 (input [31:0] a, b, output [31:0] result);

reg [31:0] temp;
integer i;
always @(*) begin
   for (i = 0; i < 32; i = i + 1) begin
      temp[i] = a[i] | b[i];
   end
end
assign result = temp;
endmodule
```

32-bit ROL:



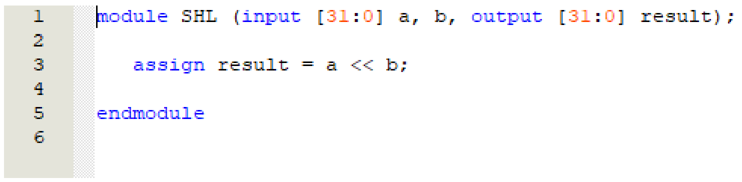
```
module ROL (input [31:0] a, b, output [31:0] result);
   reg [31:0] res;
   assign result = res;
   always @ (a or b) begin
      case (b)
            5'd1 :   res <= {a[30:0], a[31]};
            5'd2 :   res <= {a[29:0], a[31:30]};
            5'd3 :   res <= {a[28:0], a[31:29]};
            5'd4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

endmodule
```

32-bit ROR:



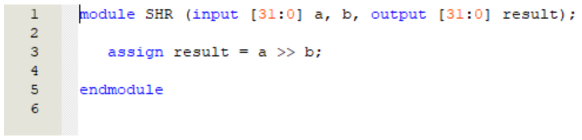
```verilog
module ROR (input [31:0] a, b, output[31:0] result);
   reg [31:0] res;
   assign result = res;
   always @(a or b) begin
      case (b)
            5'd1: res <= {a[0], a[31:1]};
            5'd2: res <= {a[1:0], a[31:2]};
            5'd3: res <= {a[2:0], a[31:3]};
            5'd4: res 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

endmodule

```

32-bit SHL:



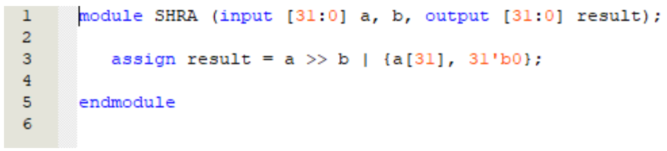
```verilog
module SHL (input [31:0] a, b, output [31:0] result);

   assign result = a << b;

endmodule

```

32-bit SHR:



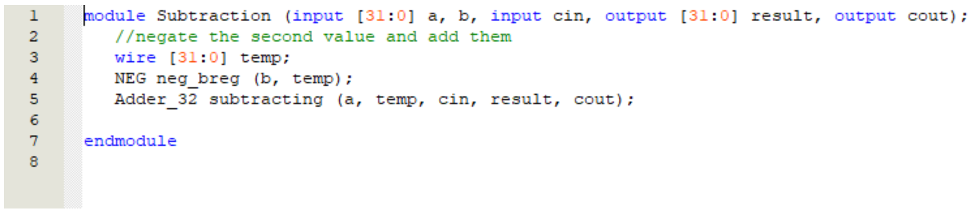
```verilog
module SHR (input [31:0] a, b, output [31:0] result);

   assign result = a >> b;

endmodule

```

32-bit SHRA:



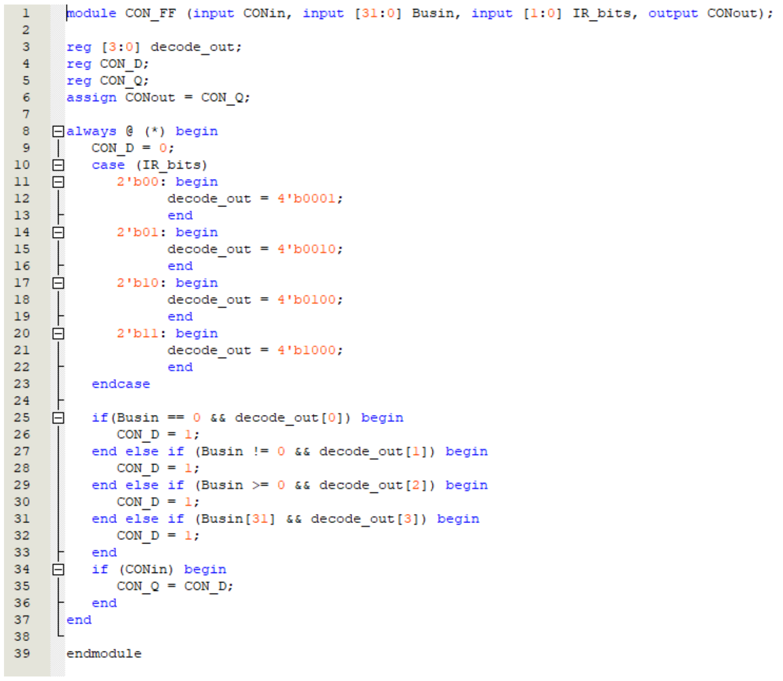
```verilog
module SHRA (input [31:0] a, b, output [31:0] result);

   assign result = a >> b | {a[31], 31'b0};

endmodule

```

32-bit Subtractor:



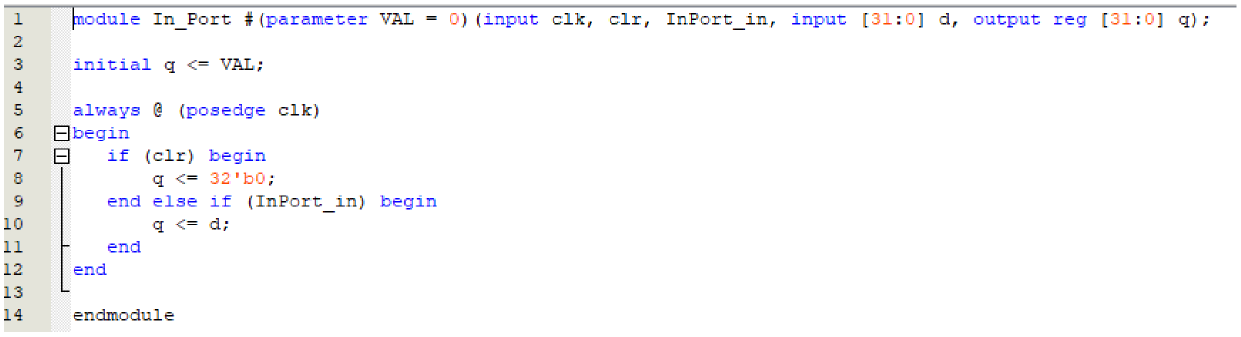
```verilog
module Subtraction (input [31:0] a, b, input cin, output [31:0] result, output cout);
   //negate the second value and add them
   wire [31:0] temp;
   NEG neg_breg (b, temp);
   Adder_32 subtracting (a, temp, cin, result, cout);

endmodule

```

CON FF Logic:



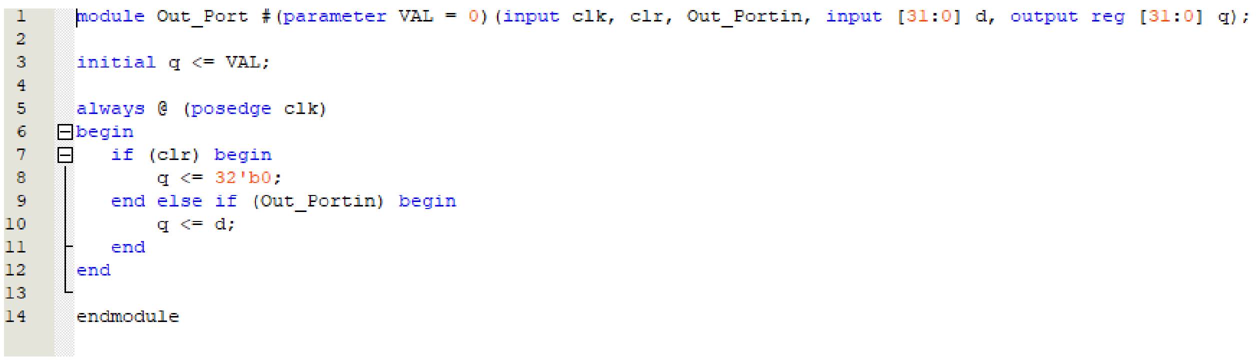
```verilog
module CON_FF (input CONin, input [31:0] Busin, input [1:0] IR_bits, output CONout);

reg [3:0] decode_out;
reg CON_D;
reg CON_Q;
assign CONout = CON_Q;

always @ (*) begin
   CON_D = 0;
   case (IR_bits)
      2'b00: begin
            decode_out = 4'b0001;
            end
      2'b01: begin
            decode_out = 4'b0010;
            end
      2'b10: begin
            decode_out = 4'b0100;
            end
      2'b11: begin
            decode_out = 4'b1000;
            end
   endcase

   if(Busin == 0 && decode_out[0]) begin
      CON_D = 1;
   end else if (Busin != 0 && decode_out[1]) begin
      CON_D = 1;
   end else if (Busin >= 0 && decode_out[2]) begin
      CON_D = 1;
   end else if (Busin[31] && decode_out[3]) begin
      CON_D = 1;
   end
   if (CONin) begin
      CON_Q = CON_D;
   end
end

endmodule
```

Input Port:



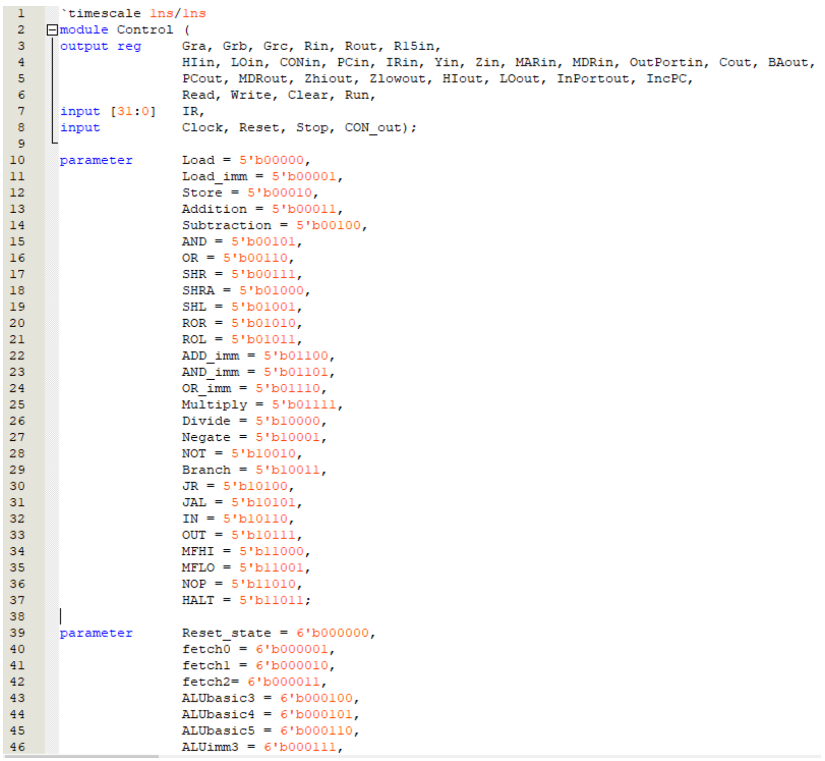
```verilog
module In_Port #(parameter VAL = 0)(input clk, clr, InPort_in, input [31:0] d, output reg [31:0] q);

initial q <= VAL;

always @ (posedge clk)
begin
   if (clr) begin
      q <= 32'b0;
   end else if (InPort_in) begin
      q <= d;
   end
end

endmodule
```

Output Port:

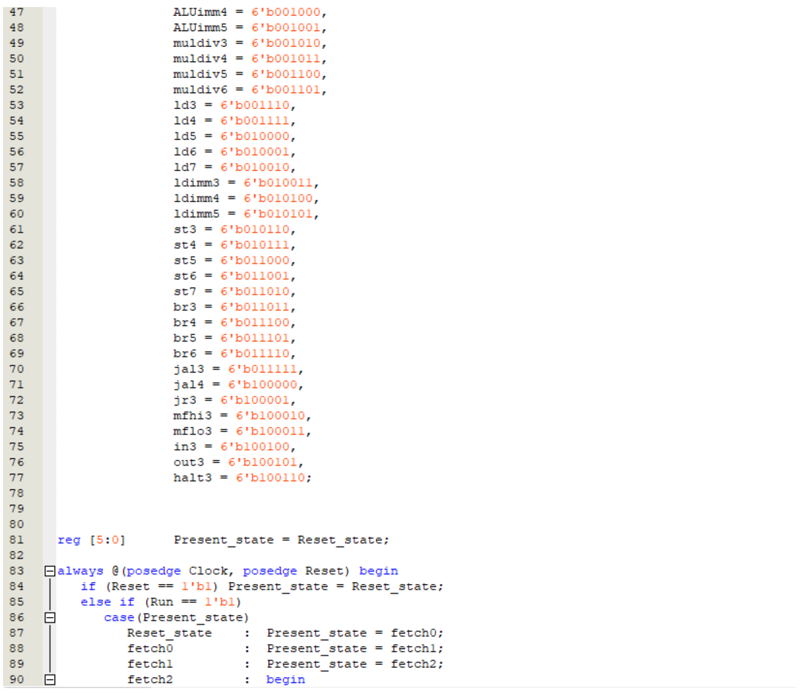


```verilog
module Out_Port #(parameter VAL = 0)(input clk, clr, Out_Portin, input [31:0] d, output reg [31:0] q);

initial q <= VAL;

always @ (posedge clk)
begin
   if (clr) begin
      q <= 32'b0;
   end else if (Out_Portin) begin
      q <= d;
   end
end

endmodule
```

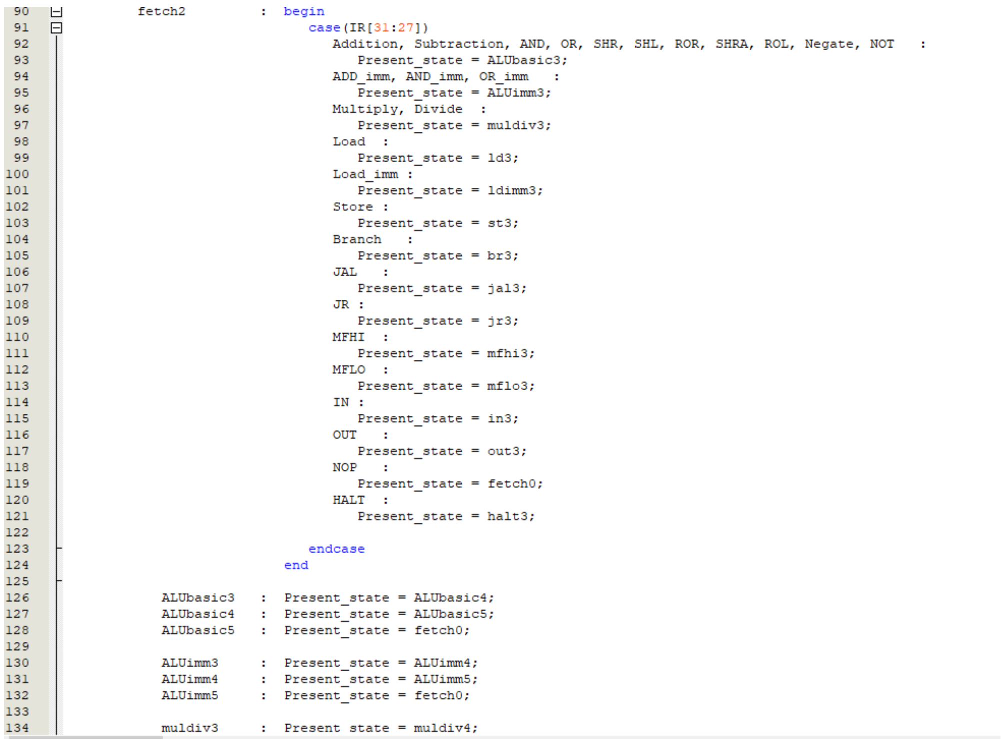
Control Unit:



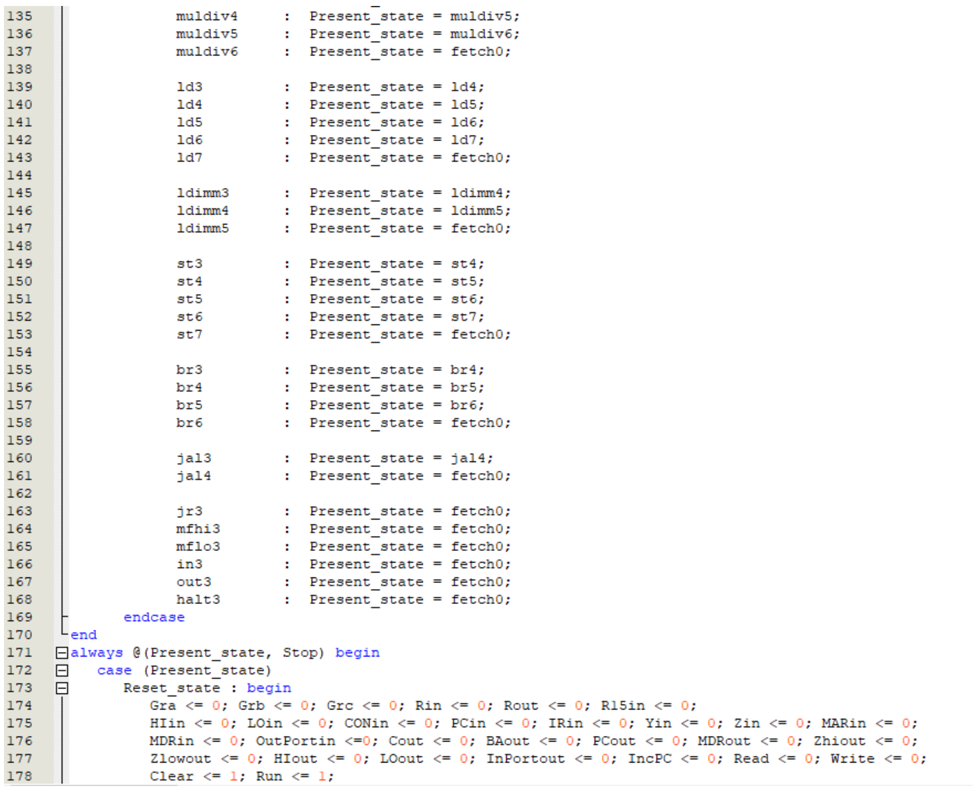
```verilog
`timescale 1ns/1ns
module Control (
output reg      Gra, Grb, Grc, Rin, Rout, R15in,
                HIin, LOin, CONin, PCin, IRin, Yin, Zin, MARin, MDRin, OutPortin, Cout, BAout,
                PCout, MDRout, Zhiout, Zlowout, HIout, LOout, InPortout, IncPC,
                Read, Write, Clear, Run,
input [31:0]    IR,
input           Clock, Reset, Stop, CON_out);

parameter       Load = 5'b00000,
                Load_imm = 5'b00001,
                Store = 5'b00010,
                Addition = 5'b00011,
                Subtraction = 5'b00100,
                AND = 5'b00101,
                OR = 5'b00110,
                SHR = 5'b00111,
                SHRA = 5'b01000,
                SHL = 5'b01001,
                ROR = 5'b01010,
                ROL = 5'b01011,
                ADD_imm = 5'b01100,
                AND_imm = 5'b01101,
                OR_imm = 5'b01110,
                Multiply = 5'b01111,
                Divide = 5'b10000,
                Negate = 5'b10001,
                NOT = 5'b10010,
                Branch = 5'b10011,
                JR = 5'b10100,
                JAL = 5'b10101,
                IN = 5'b10110,
                OUT = 5'b10111,
                MFHI = 5'b11000,
                MFLO = 5'b11001,
                NOP = 5'b11010,
                HALT = 5'b11011;

parameter       Reset_state = 6'b000000,
                fetch0 = 6'b000001,
                fetch1 = 6'b000010,
                fetch2= 6'b000011,
                ALUbasic3 = 6'b000100,
                ALUbasic4 = 6'b000101,
                ALUbasic5 = 6'b000110,
                ALUimm3 = 6'b000111,
```



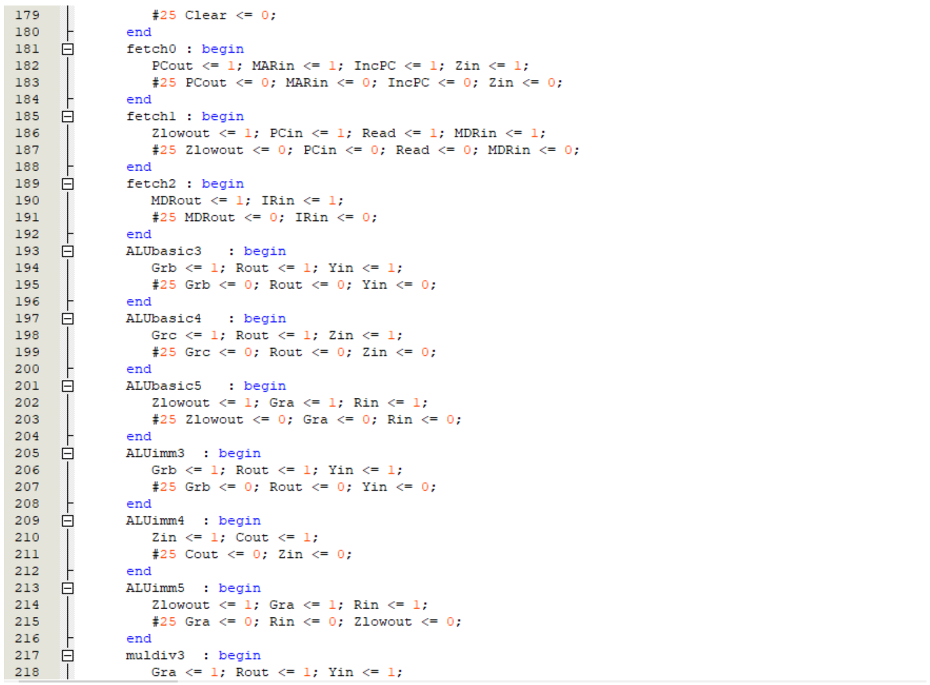
```verilog
                ALUimm4 = 6'b001000,
                ALUimm5 = 6'b001001,
                muldiv3 = 6'b001010,
                muldiv4 = 6'b001011,
                muldiv5 = 6'b001100,
                muldiv6 = 6'b001101,
                ld3 = 6'b001110,
                ld4 = 6'b001111,
                ld5 = 6'b010000,
                ld6 = 6'b010001,
                ld7 = 6'b010010,
                ldimm3 = 6'b010011,
                ldimm4 = 6'b010100,
                ldimm5 = 6'b010101,
                st3 = 6'b010110,
                st4 = 6'b010111,
                st5 = 6'b011000,
                st6 = 6'b011001,
                st7 = 6'b011010,
                br3 = 6'b011011,
                br4 = 6'b011100,
                br5 = 6'b011101,
                br6 = 6'b011110,
                jal3 = 6'b011111,
                jal4 = 6'b100000,
                jr3 = 6'b100001,
                mfhi3 = 6'b100010,
                mflo3 = 6'b100011,
                in3 = 6'b100100,
                out3 = 6'b100101,
                halt3 = 6'b100110;



reg [5:0]       Present_state = Reset_state;

always @(posedge Clock, posedge Reset) begin
    if (Reset == 1'b1) Present_state = Reset_state;
    else if (Run == 1'b1)
        case(Present_state)
            Reset_state     :   Present_state = fetch0;
            fetch0          :   Present_state = fetch1;
            fetch1          :   Present_state = fetch2;
            fetch2          :   begin
```



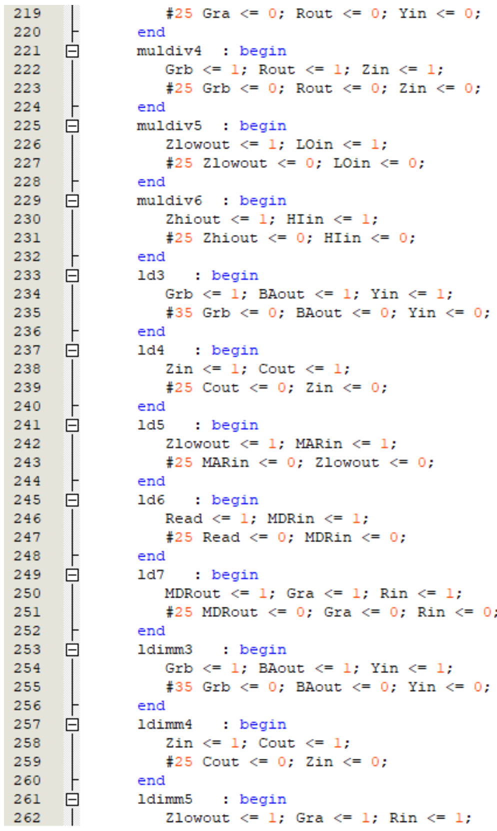
```verilog
         fetch2          :  begin
                              case(IR[31:27])
                                 Addition, Subtraction, AND, OR, SHR, SHL, ROR, SHRA, ROL, Negate, NOT   :
                                    Present_state = ALUbasic3;
                                 ADD_imm, AND_imm, OR_imm   :
                                    Present_state = ALUimm3;
                                 Multiply, Divide   :
                                    Present_state = muldiv3;
                                 Load   :
                                    Present_state = ld3;
                                 Load_imm :
                                    Present_state = ldimm3;
                                 Store :
                                    Present_state = st3;
                                 Branch    :
                                    Present_state = br3;
                                 JAL    :
                                    Present_state = jal3;
                                 JR :
                                    Present_state = jr3;
                                 MFHI   :
                                    Present_state = mfhi3;
                                 MFLO   :
                                    Present_state = mflo3;
                                 IN :
                                    Present_state = in3;
                                 OUT    :
                                    Present_state = out3;
                                 NOP    :
                                    Present_state = fetch0;
                                 HALT   :
                                    Present_state = halt3;

                              endcase
                           end

            ALUbasic3   :  Present_state = ALUbasic4;
            ALUbasic4   :  Present_state = ALUbasic5;
            ALUbasic5   :  Present_state = fetch0;

            ALUimm3     :  Present_state = ALUimm4;
            ALUimm4     :  Present_state = ALUimm5;
            ALUimm5     :  Present_state = fetch0;

            muldiv3     :  Present_state = muldiv4;
```



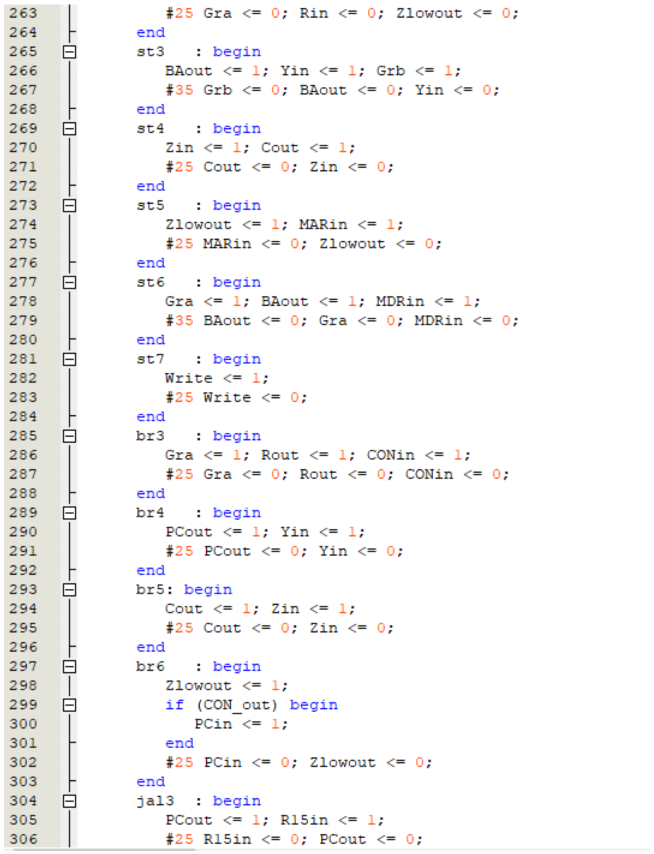
```verilog
            muldiv4      :  Present_state = muldiv5;
            muldiv5      :  Present_state = muldiv6;
            muldiv6      :  Present_state = fetch0;

            ld3          :  Present_state = ld4;
            ld4          :  Present_state = ld5;
            ld5          :  Present_state = ld6;
            ld6          :  Present_state = ld7;
            ld7          :  Present_state = fetch0;

            ldimm3       :  Present_state = ldimm4;
            ldimm4       :  Present_state = ldimm5;
            ldimm5       :  Present_state = fetch0;

            st3          :  Present_state = st4;
            st4          :  Present_state = st5;
            st5          :  Present_state = st6;
            st6          :  Present_state = st7;
            st7          :  Present_state = fetch0;

            br3          :  Present_state = br4;
            br4          :  Present_state = br5;
            br5          :  Present_state = br6;
            br6          :  Present_state = fetch0;

            jal3         :  Present_state = jal4;
            jal4         :  Present_state = fetch0;

            jr3          :  Present_state = fetch0;
            mfhi3        :  Present_state = fetch0;
            mflo3        :  Present_state = fetch0;
            in3          :  Present_state = fetch0;
            out3         :  Present_state = fetch0;
            halt3        :  Present_state = fetch0;
      endcase
end
always @(Present_state, Stop) begin
   case (Present_state)
      Reset_state : begin
         Gra <= 0; Grb <= 0; Grc <= 0; Rin <= 0; Rout <= 0; R15in <= 0;
         HIin <= 0; LOin <= 0; CONin <= 0; PCin <= 0; IRin <= 0; Yin <= 0; Zin <= 0; MARin <= 0;
         MDRin <= 0; OutPortin <=0; Cout <= 0; BAout <= 0; PCout <= 0; MDRout <= 0; Zhiout <= 0;
         Zlowout <= 0; HIout <= 0; LOout <= 0; InPortout <= 0; IncPC <= 0; Read <= 0; Write <= 0;
         Clear <= 1; Run <= 1;
```



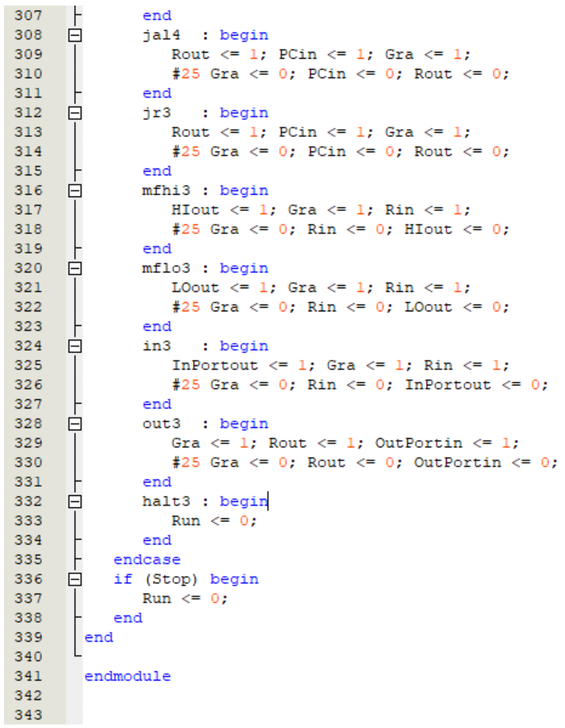
```
         #25 Clear <= 0;
      end
      fetch0 : begin
         PCout <= 1; MARin <= 1; IncPC <= 1; Zin <= 1;
         #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
      end
      fetch1 : begin
         Zlowout <= 1; PCin <= 1; Read <= 1; MDRin <= 1;
         #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
      end
      fetch2 : begin
         MDRout <= 1; IRin <= 1;
         #25 MDRout <= 0; IRin <= 0;
      end
      ALUbasic3   : begin
         Grb <= 1; Rout <= 1; Yin <= 1;
         #25 Grb <= 0; Rout <= 0; Yin <= 0;
      end
      ALUbasic4   : begin
         Grc <= 1; Rout <= 1; Zin <= 1;
         #25 Grc <= 0; Rout <= 0; Zin <= 0;
      end
      ALUbasic5   : begin
         Zlowout <= 1; Gra <= 1; Rin <= 1;
         #25 Zlowout <= 0; Gra <= 0; Rin <= 0;
      end
      ALUimm3  : begin
         Grb <= 1; Rout <= 1; Yin <= 1;
         #25 Grb <= 0; Rout <= 0; Yin <= 0;
      end
      ALUimm4  : begin
         Zin <= 1; Cout <= 1;
         #25 Cout <= 0; Zin <= 0;
      end
      ALUimm5  : begin
         Zlowout <= 1; Gra <= 1; Rin <= 1;
         #25 Gra <= 0; Rin <= 0; Zlowout <= 0;
      end
      muldiv3  : begin
         Gra <= 1; Rout <= 1; Yin <= 1;
```



```
         #25 Gra <= 0; Rout <= 0; Yin <= 0;
      end
      muldiv4  : begin
         Grb <= 1; Rout <= 1; Zin <= 1;
         #25 Grb <= 0; Rout <= 0; Zin <= 0;
      end
      muldiv5  : begin
         Zlowout <= 1; LOin <= 1;
         #25 Zlowout <= 0; LOin <= 0;
      end
      muldiv6  : begin
         Zhiout <= 1; HIin <= 1;
         #25 Zhiout <= 0; HIin <= 0;
      end
      ld3    : begin
         Grb <= 1; BAout <= 1; Yin <= 1;
         #35 Grb <= 0; BAout <= 0; Yin <= 0;
      end
      ld4    : begin
         Zin <= 1; Cout <= 1;
         #25 Cout <= 0; Zin <= 0;
      end
      ld5    : begin
         Zlowout <= 1; MARin <= 1;
         #25 MARin <= 0; Zlowout <= 0;
      end
      ld6    : begin
         Read <= 1; MDRin <= 1;
         #25 Read <= 0; MDRin <= 0;
      end
      ld7    : begin
         MDRout <= 1; Gra <= 1; Rin <= 1;
         #25 MDRout <= 0; Gra <= 0; Rin <= 0;
      end
      ldimm3   : begin
         Grb <= 1; BAout <= 1; Yin <= 1;
         #35 Grb <= 0; BAout <= 0; Yin <= 0;
      end
      ldimm4   : begin
         Zin <= 1; Cout <= 1;
         #25 Cout <= 0; Zin <= 0;
      end
      ldimm5   : begin
         Zlowout <= 1; Gra <= 1; Rin <= 1;
```

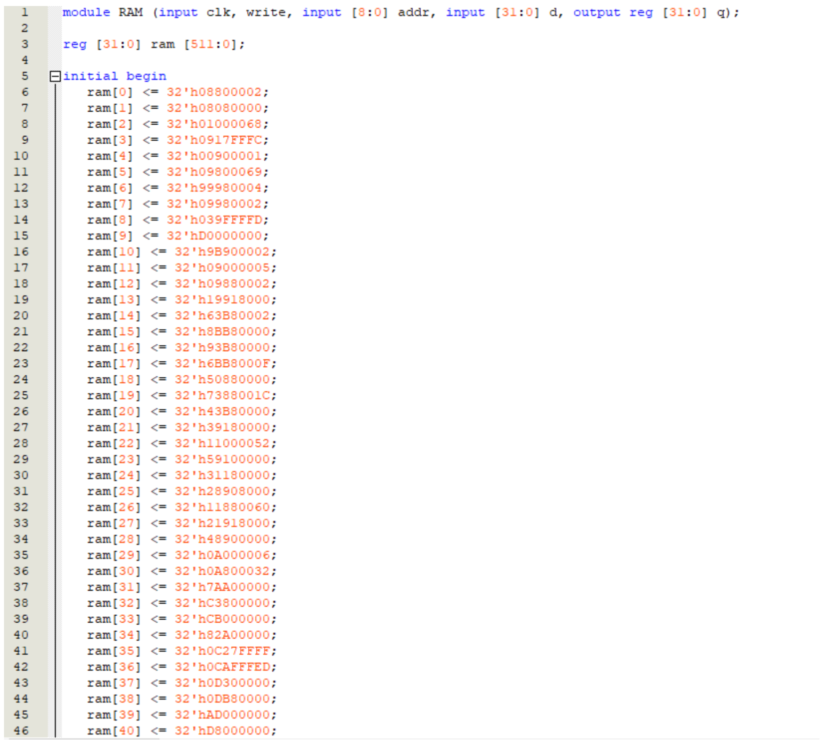


```verilog
         #25 Gra <= 0; Rin <= 0; Zlowout <= 0;
      end
      st3    : begin
         BAout <= 1; Yin <= 1; Grb <= 1;
         #35 Grb <= 0; BAout <= 0; Yin <= 0;
      end
      st4    : begin
         Zin <= 1; Cout <= 1;
         #25 Cout <= 0; Zin <= 0;
      end
      st5    : begin
         Zlowout <= 1; MARin <= 1;
         #25 MARin <= 0; Zlowout <= 0;
      end
      st6    : begin
         Gra <= 1; BAout <= 1; MDRin <= 1;
         #35 BAout <= 0; Gra <= 0; MDRin <= 0;
      end
      st7    : begin
         Write <= 1;
         #25 Write <= 0;
      end
      br3    : begin
         Gra <= 1; Rout <= 1; CONin <= 1;
         #25 Gra <= 0; Rout <= 0; CONin <= 0;
      end
      br4    : begin
         PCout <= 1; Yin <= 1;
         #25 PCout <= 0; Yin <= 0;
      end
      br5: begin
         Cout <= 1; Zin <= 1;
         #25 Cout <= 0; Zin <= 0;
      end
      br6    : begin
         Zlowout <= 1;
         if (CON_out) begin
            PCin <= 1;
         end
         #25 PCin <= 0; Zlowout <= 0;
      end
      jal3   : begin
         PCout <= 1; R15in <= 1;
         #25 R15in <= 0; PCout <= 0;
```

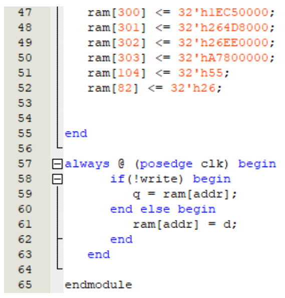


```verilog
      end
      jal4   : begin
         Rout <= 1; PCin <= 1; Gra <= 1;
         #25 Gra <= 0; PCin <= 0; Rout <= 0;
      end
      jr3    : begin
         Rout <= 1; PCin <= 1; Gra <= 1;
         #25 Gra <= 0; PCin <= 0; Rout <= 0;
      end
      mfhi3 : begin
         HIout <= 1; Gra <= 1; Rin <= 1;
         #25 Gra <= 0; Rin <= 0; HIout <= 0;
      end
      mflo3 : begin
         LOout <= 1; Gra <= 1; Rin <= 1;
         #25 Gra <= 0; Rin <= 0; LOout <= 0;
      end
      in3    : begin
         InPortout <= 1; Gra <= 1; Rin <= 1;
         #25 Gra <= 0; Rin <= 0; InPortout <= 0;
      end
      out3   : begin
         Gra <= 1; Rout <= 1; OutPortin <= 1;
         #25 Gra <= 0; Rout <= 0; OutPortin <= 0;
      end
      halt3 : begin
         Run <= 0;
      end
   endcase
   if (Stop) begin
      Run <= 0;
   end
end

endmodule
```

Ram:



```
module RAM (input clk, write, input [8:0] addr, input [31:0] d, output reg [31:0] q);

reg [31:0] ram [511:0];

initial begin
   ram[0] <= 32'h08800002;
   ram[1] <= 32'h08080000;
   ram[2] <= 32'h01000068;
   ram[3] <= 32'h0917FFFC;
   ram[4] <= 32'h00900001;
   ram[5] <= 32'h09800069;
   ram[6] <= 32'h99980004;
   ram[7] <= 32'h09980002;
   ram[8] <= 32'h039FFFFD;
   ram[9] <= 32'hD0000000;
   ram[10] <= 32'h9B900002;
   ram[11] <= 32'h09000005;
   ram[12] <= 32'h09880002;
   ram[13] <= 32'h19918000;
   ram[14] <= 32'h63B80002;
   ram[15] <= 32'h8BB80000;
   ram[16] <= 32'h93B80000;
   ram[17] <= 32'h6BB8000F;
   ram[18] <= 32'h50880000;
   ram[19] <= 32'h7388001C;
   ram[20] <= 32'h43B80000;
   ram[21] <= 32'h39180000;
   ram[22] <= 32'h11000052;
   ram[23] <= 32'h59100000;
   ram[24] <= 32'h31180000;
   ram[25] <= 32'h28908000;
   ram[26] <= 32'h11880060;
   ram[27] <= 32'h21918000;
   ram[28] <= 32'h48900000;
   ram[29] <= 32'h0A000006;
   ram[30] <= 32'h0A800032;
   ram[31] <= 32'h7AA00000;
   ram[32] <= 32'hC3800000;
   ram[33] <= 32'hCB000000;
   ram[34] <= 32'h82A00000;
   ram[35] <= 32'h0C27FFFF;
   ram[36] <= 32'h0CAFFFED;
   ram[37] <= 32'h0D300000;
   ram[38] <= 32'h0DB80000;
   ram[39] <= 32'hAD000000;
   ram[40] <= 32'hD8000000;
```



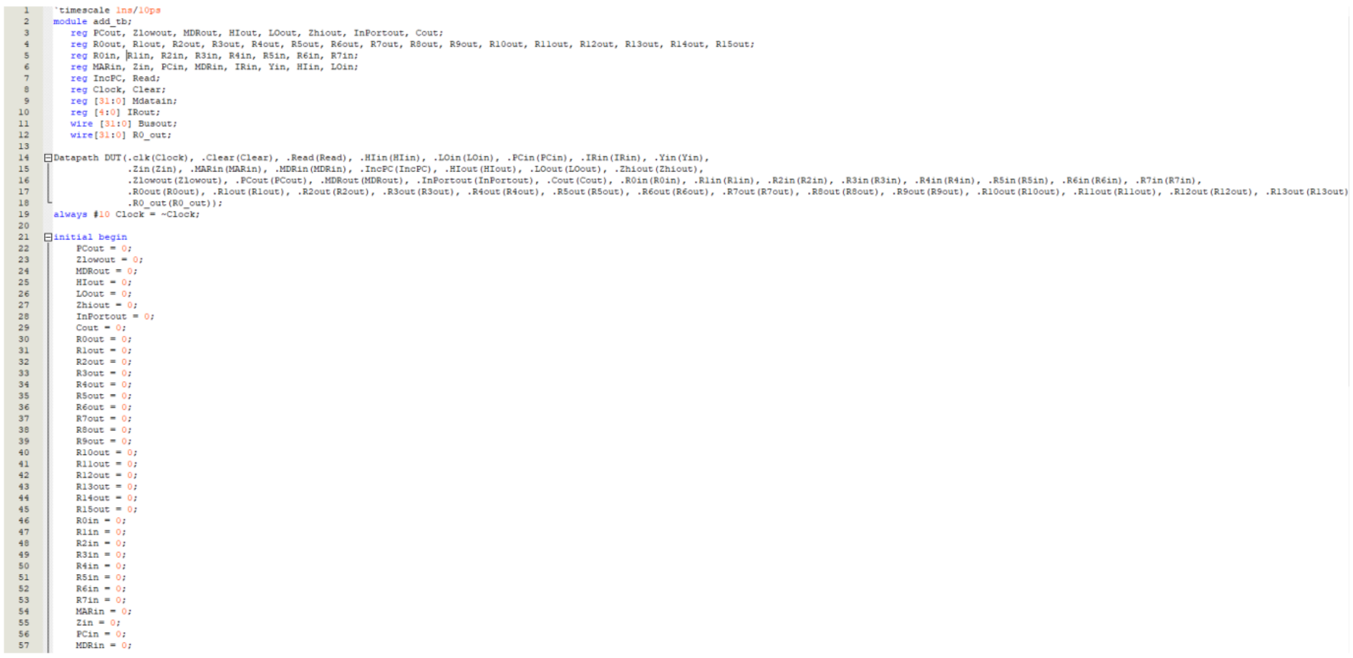
```
   ram[300] <= 32'h1EC50000;
   ram[301] <= 32'h264D8000;
   ram[302] <= 32'h26EE0000;
   ram[303] <= 32'hA7800000;
   ram[104] <= 32'h55;
   ram[82] <= 32'h26;


end

always @ (posedge clk) begin
      if(!write) begin
         q = ram[addr];
      end else begin
         ram[addr] = d;
      end
   end

endmodule
```

Test Benches

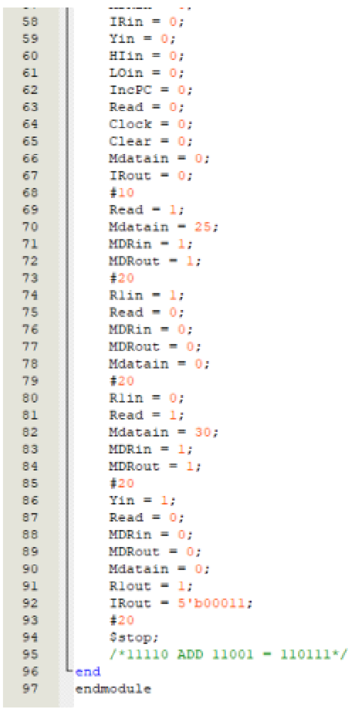
Phase 1:

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:



```verilog
`timescale 1ns/10ps
module add_tb;
   reg PCout, Zlowout, MDRout, HIout, LOout, Zhiout, InPortout, Cout;
   reg R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out;
   reg R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in;
   reg MARin, Zin, PCin, MDRin, IRin, Yin, HIin, LOin;
   reg IncPC, Read;
   reg Clock, Clear;
   reg [31:0] Mdatain;
   reg [4:0] IRout;
   wire [31:0] Busout;
   wire[31:0] R0_out;

Datapath DUT(.clk(Clock), .Clear(Clear), .Read(Read), .HIin(HIin), .LOin(LOin), .PCin(PCin), .IRin(IRin), .Yin(Yin),
             .Zin(Zin), .MARin(MARin), .MDRin(MDRin), .IncPC(IncPC), .HIout(HIout), .LOout(LOout), .Zhiout(Zhiout),
             .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),
             .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)
             .R0_out(R0_out));
always #10 Clock = ~Clock;

initial begin
    PCout = 0;
    Zlowout = 0;
    MDRout = 0;
    HIout = 0;
    LOout = 0;
    Zhiout = 0;
    InPortout = 0;
    Cout = 0;
    R0out = 0;
    R1out = 0;
    R2out = 0;
    R3out = 0;
    R4out = 0;
    R5out = 0;
    R6out = 0;
    R7out = 0;
    R8out = 0;
    R9out = 0;
    R10out = 0;
    R11out = 0;
    R12out = 0;
    R13out = 0;
    R14out = 0;
    R15out = 0;
    R0in = 0;
    R1in = 0;
    R2in = 0;
    R3in = 0;
    R4in = 0;
    R5in = 0;
    R6in = 0;
    R7in = 0;
    MARin = 0;
    Zin = 0;
    PCin = 0;
    MDRin = 0;
```

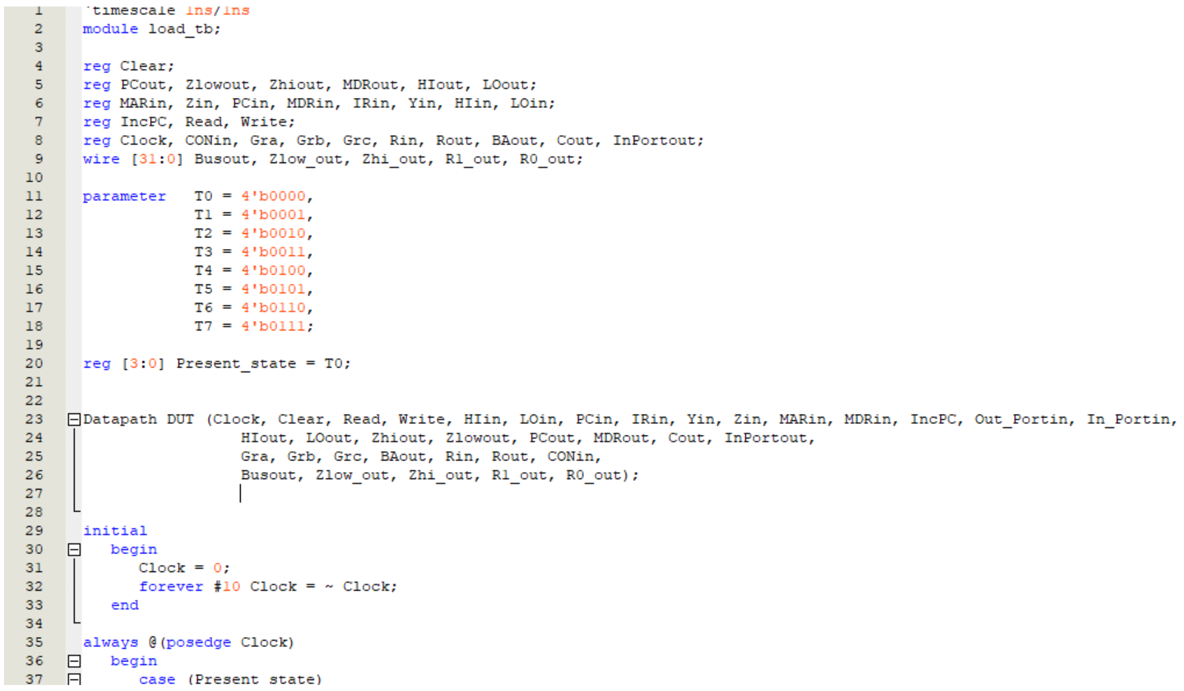


```verilog
    IRin = 0;
    Yin = 0;
    HIin = 0;
    LOin = 0;
    IncPC = 0;
    Read = 0;
    Clock = 0;
    Clear = 0;
    Mdatain = 0;
    IRout = 0;
    #10
    Read = 1;
    Mdatain = 25;
    MDRin = 1;
    MDRout = 1;
    #20
    R1in = 1;
    Read = 0;
    MDRin = 0;
    MDRout = 0;
    Mdatain = 0;
    #20
    R1in = 0;
    Read = 1;
    Mdatain = 30;
    MDRin = 1;
    MDRout = 1;
    #20
    Yin = 1;
    Read = 0;
    MDRin = 0;
    MDRout = 0;
    Mdatain = 0;
    R1out = 1;
    IRout = 5'b00011;
    #20
    $stop;
    /*11110 ADD 11001 = 110111*/
end
endmodule
```

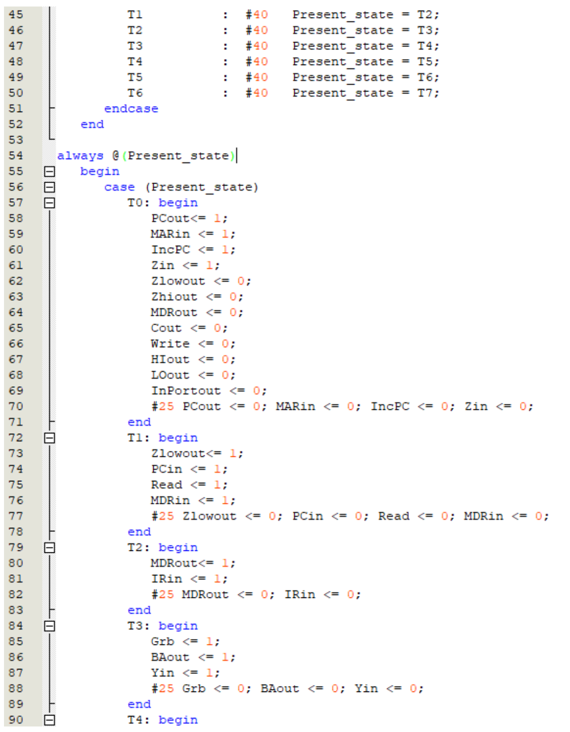
Phase 2:

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.

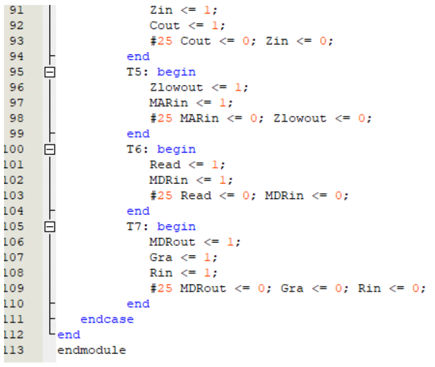
Load_TB:



```
`timescale 1ns/1ns
module load_tb;

reg Clear;
reg PCout, Zlowout, Zhiout, MDRout, HIout, LOout;
reg MARin, Zin, PCin, MDRin, IRin, Yin, HIin, LOin;
reg IncPC, Read, Write;
reg Clock, CONin, Gra, Grb, Grc, Rin, Rout, BAout, Cout, InPortout;
wire [31:0] Busout, Zlow_out, Zhi_out, R1_out, R0_out;

parameter    T0 = 4'b0000,
             T1 = 4'b0001,
             T2 = 4'b0010,
             T3 = 4'b0011,
             T4 = 4'b0100,
             T5 = 4'b0101,
             T6 = 4'b0110,
             T7 = 4'b0111;

reg [3:0] Present_state = T0;


Datapath DUT (Clock, Clear, Read, Write, HIin, LOin, PCin, IRin, Yin, Zin, MARin, MDRin, IncPC, Out_Portin, In_Portin,
                     HIout, LOout, Zhiout, Zlowout, PCout, MDRout, Cout, InPortout,
                     Gra, Grb, Grc, BAout, Rin, Rout, CONin,
                     Busout, Zlow_out, Zhi_out, R1_out, R0_out);


initial
   begin
      Clock = 0;
      forever #10 Clock = ~ Clock;
   end

always @(posedge Clock)
   begin
      case (Present state)
```

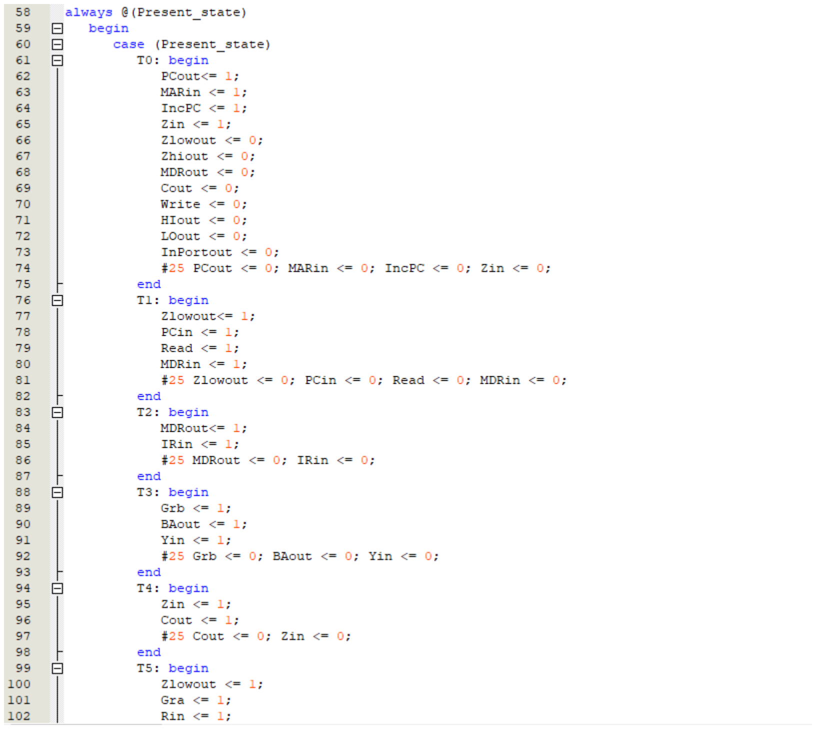


```
         T1          :  #40   Present_state = T2;
         T2          :  #40   Present_state = T3;
         T3          :  #40   Present_state = T4;
         T4          :  #40   Present_state = T5;
         T5          :  #40   Present_state = T6;
         T6          :  #40   Present_state = T7;
      endcase
   end

always @(Present_state)
   begin
      case (Present_state)
         T0: begin
            PCout<= 1;
            MARin <= 1;
            IncPC <= 1;
            Zin <= 1;
            Zlowout <= 0;
            Zhiout <= 0;
            MDRout <= 0;
            Cout <= 0;
            Write <= 0;
            HIout <= 0;
            LOout <= 0;
            InPortout <= 0;
            #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
         end
         T1: begin
            Zlowout<= 1;
            PCin <= 1;
            Read <= 1;
            MDRin <= 1;
            #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
         end
         T2: begin
            MDRout<= 1;
            IRin <= 1;
            #25 MDRout <= 0; IRin <= 0;
         end
         T3: begin
            Grb <= 1;
            BAout <= 1;
            Yin <= 1;
            #25 Grb <= 0; BAout <= 0; Yin <= 0;
         end
         T4: begin
```



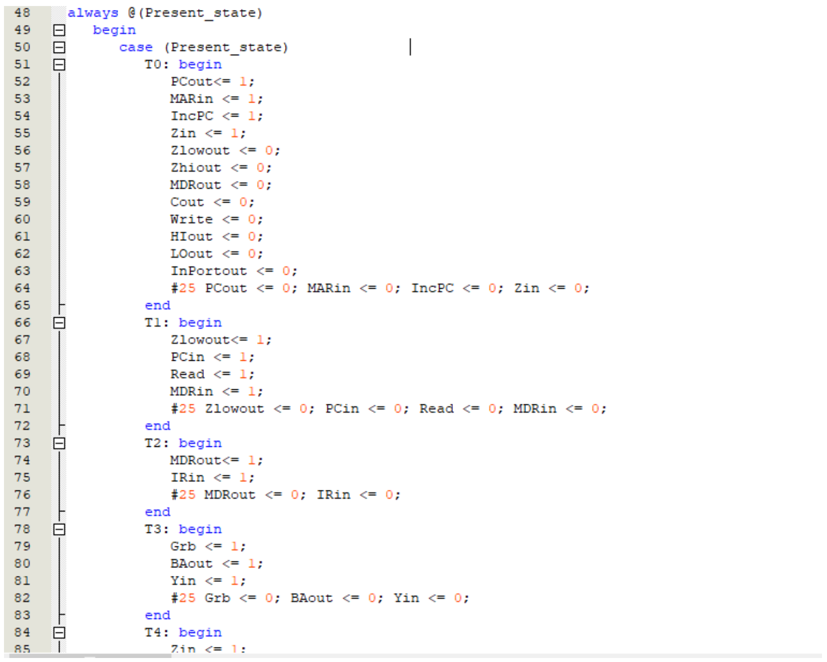
```
            Zin <= 1;
            Cout <= 1;
            #25 Cout <= 0; Zin <= 0;
         end
         T5: begin
            Zlowout <= 1;
            MARin <= 1;
            #25 MARin <= 0; Zlowout <= 0;
         end
         T6: begin
            Read <= 1;
            MDRin <= 1;
            #25 Read <= 0; MDRin <= 0;
         end
         T7: begin
            MDRout <= 1;
            Gra <= 1;
            Rin <= 1;
            #25 MDRout <= 0; Gra <= 0; Rin <= 0;
         end
   endcase
end
endmodule
```

Loadi_TB:

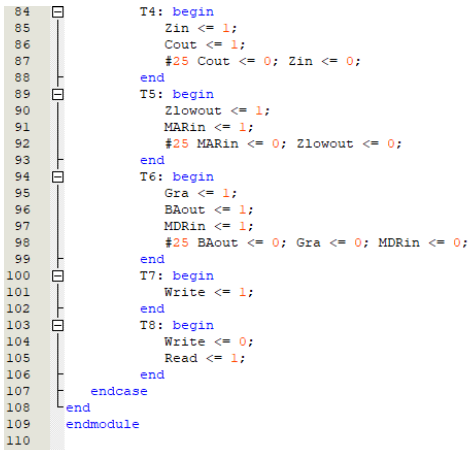


```
always @(Present_state)
   begin
      case (Present_state)
         T0: begin
            PCout<= 1;
            MARin <= 1;
            IncPC <= 1;
            Zin <= 1;
            Zlowout <= 0;
            Zhiout <= 0;
            MDRout <= 0;
            Cout <= 0;
            Write <= 0;
            HIout <= 0;
            LOout <= 0;
            InPortout <= 0;
            #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
         end
         T1: begin
            Zlowout<= 1;
            PCin <= 1;
            Read <= 1;
            MDRin <= 1;
            #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
         end
         T2: begin
            MDRout<= 1;
            IRin <= 1;
            #25 MDRout <= 0; IRin <= 0;
         end
         T3: begin
            Grb <= 1;
            BAout <= 1;
            Yin <= 1;
            #25 Grb <= 0; BAout <= 0; Yin <= 0;
         end
         T4: begin
            Zin <= 1;
            Cout <= 1;
            #25 Cout <= 0; Zin <= 0;
         end
         T5: begin
            Zlowout <= 1;
            Gra <= 1;
            Rin <= 1;
```

Store_TB:

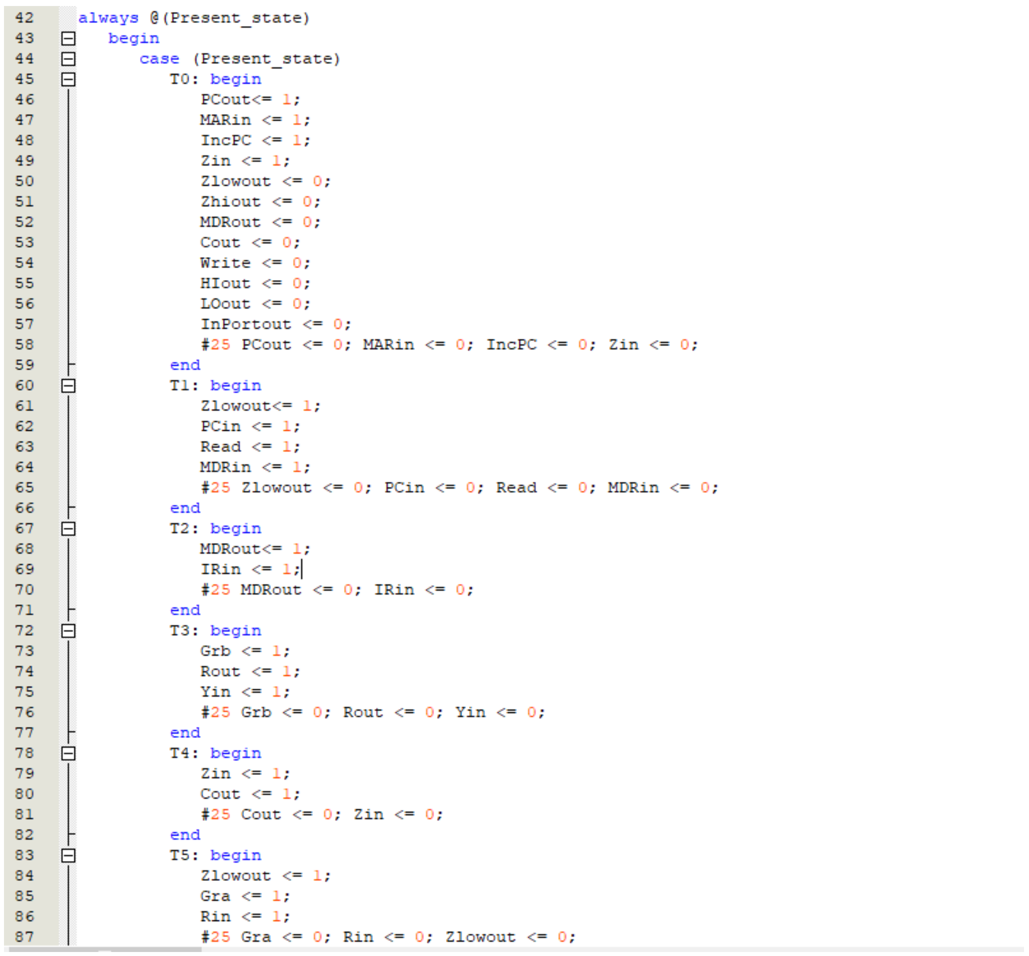


```
always @(Present_state)
   begin
      case (Present_state)
         T0: begin
            PCout<= 1;
            MARin <= 1;
            IncPC <= 1;
            Zin <= 1;
            Zlowout <= 0;
            Zhiout <= 0;
            MDRout <= 0;
            Cout <= 0;
            Write <= 0;
            HIout <= 0;
            LOout <= 0;
            InPortout <= 0;
            #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
         end
         T1: begin
            Zlowout<= 1;
            PCin <= 1;
            Read <= 1;
            MDRin <= 1;
            #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
         end
         T2: begin
            MDRout<= 1;
            IRin <= 1;
            #25 MDRout <= 0; IRin <= 0;
         end
         T3: begin
            Grb <= 1;
            BAout <= 1;
            Yin <= 1;
            #25 Grb <= 0; BAout <= 0; Yin <= 0;
         end
         T4: begin
            Zin <= 1;
```



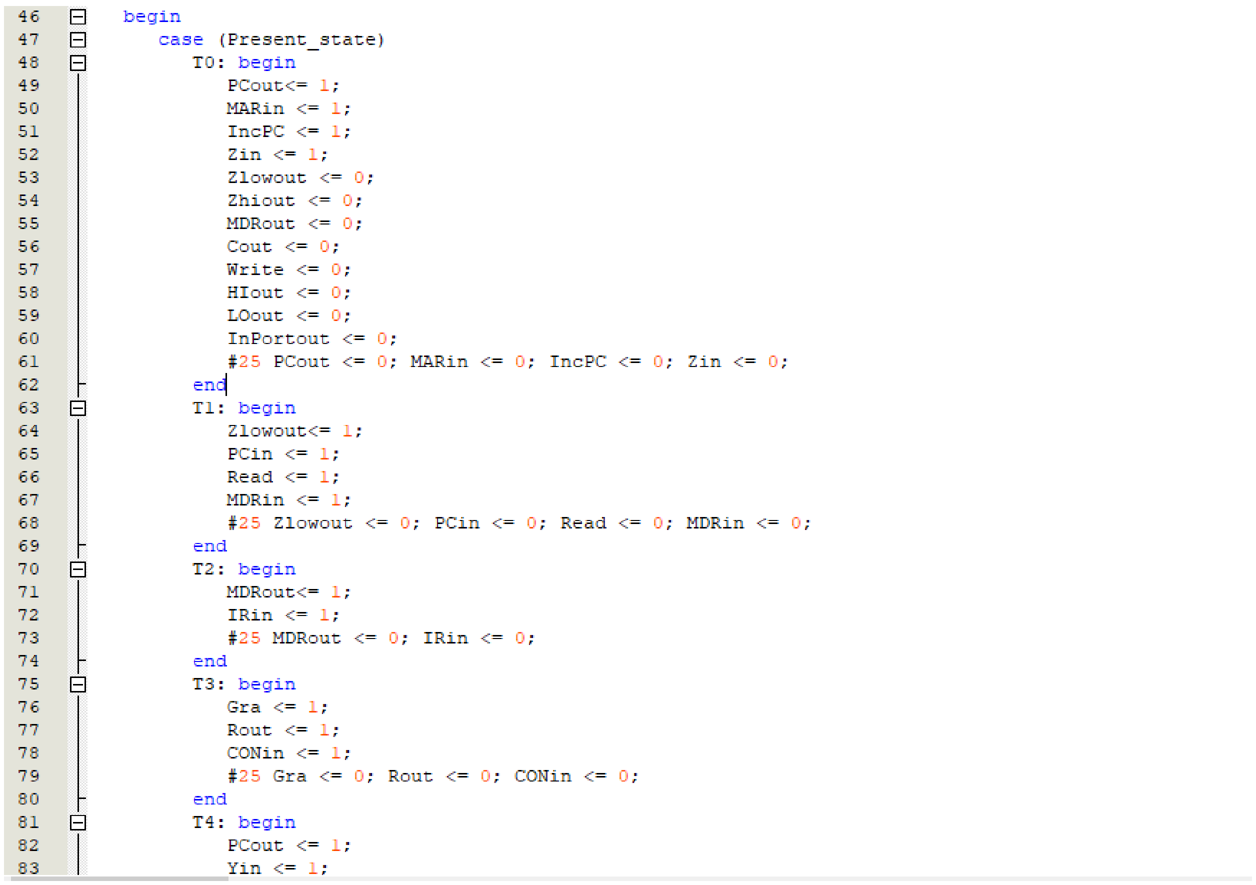
```
         T4: begin
            Zin <= 1;
            Cout <= 1;
            #25 Cout <= 0; Zin <= 0;
         end
         T5: begin
            Zlowout <= 1;
            MARin <= 1;
            #25 MARin <= 0; Zlowout <= 0;
         end
         T6: begin
            Gra <= 1;
            BAout <= 1;
            MDRin <= 1;
            #25 BAout <= 0; Gra <= 0; MDRin <= 0;
         end
         T7: begin
            Write <= 1;
         end
         T8: begin
            Write <= 0;
            Read <= 1;
         end
      endcase
end
endmodule
```

ADDI/ANDI/ORI:

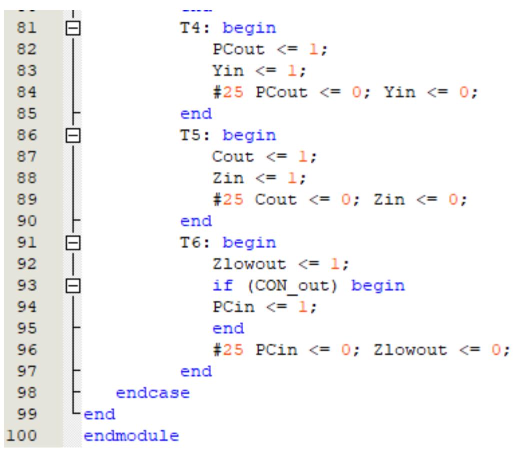


```
always @(Present_state)
   begin
      case (Present_state)
         T0: begin
            PCout<= 1;
            MARin <= 1;
            IncPC <= 1;
            Zin <= 1;
            Zlowout <= 0;
            Zhiout <= 0;
            MDRout <= 0;
            Cout <= 0;
            Write <= 0;
            HIout <= 0;
            LOout <= 0;
            InPortout <= 0;
            #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
         end
         T1: begin
            Zlowout<= 1;
            PCin <= 1;
            Read <= 1;
            MDRin <= 1;
            #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
         end
         T2: begin
            MDRout<= 1;
            IRin <= 1;
            #25 MDRout <= 0; IRin <= 0;
         end
         T3: begin
            Grb <= 1;
            Rout <= 1;
            Yin <= 1;
            #25 Grb <= 0; Rout <= 0; Yin <= 0;
         end
         T4: begin
            Zin <= 1;
            Cout <= 1;
            #25 Cout <= 0; Zin <= 0;
         end
         T5: begin
            Zlowout <= 1;
            Gra <= 1;
            Rin <= 1;
            #25 Gra <= 0; Rin <= 0; Zlowout <= 0;
```

BRZR/BRNZ/BRPL/BRMI:



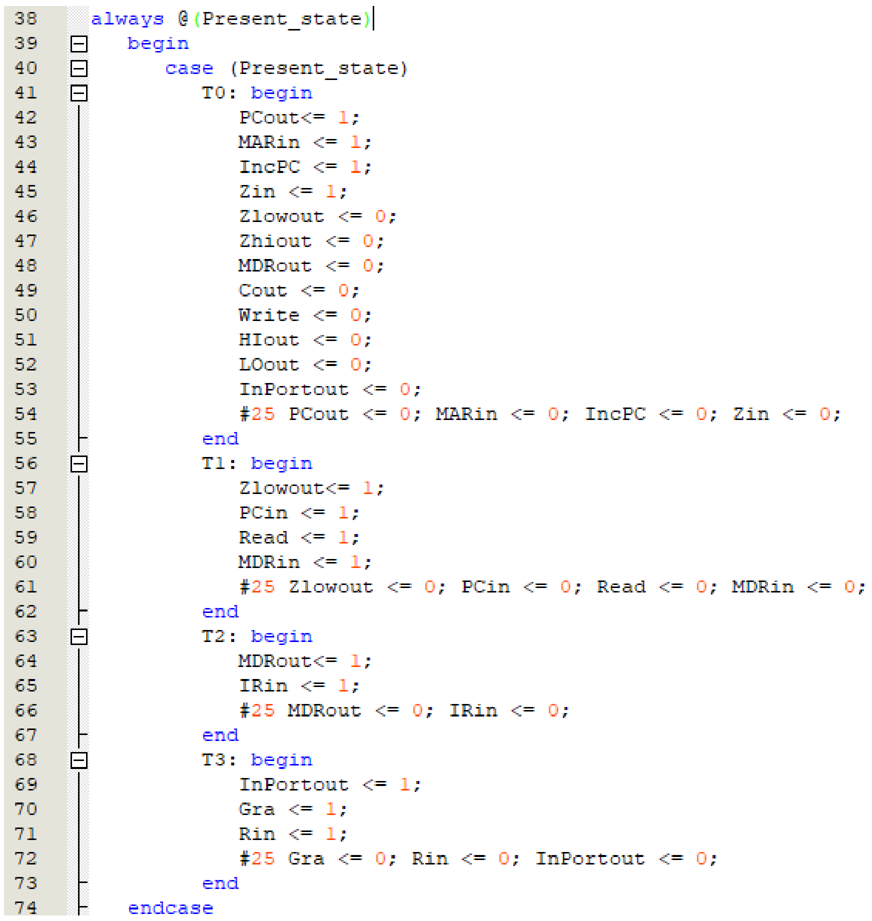
```
begin
   case (Present_state)
      T0: begin
         PCout<= 1;
         MARin <= 1;
         IncPC <= 1;
         Zin <= 1;
         Zlowout <= 0;
         Zhiout <= 0;
         MDRout <= 0;
         Cout <= 0;
         Write <= 0;
         HIout <= 0;
         LOout <= 0;
         InPortout <= 0;
         #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
      end
      T1: begin
         Zlowout<= 1;
         PCin <= 1;
         Read <= 1;
         MDRin <= 1;
         #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
      end
      T2: begin
         MDRout<= 1;
         IRin <= 1;
         #25 MDRout <= 0; IRin <= 0;
      end
      T3: begin
         Gra <= 1;
         Rout <= 1;
         CONin <= 1;
         #25 Gra <= 0; Rout <= 0; CONin <= 0;
      end
      T4: begin
         PCout <= 1;
         Yin <= 1;
```



```
      T4: begin
         PCout <= 1;
         Yin <= 1;
```

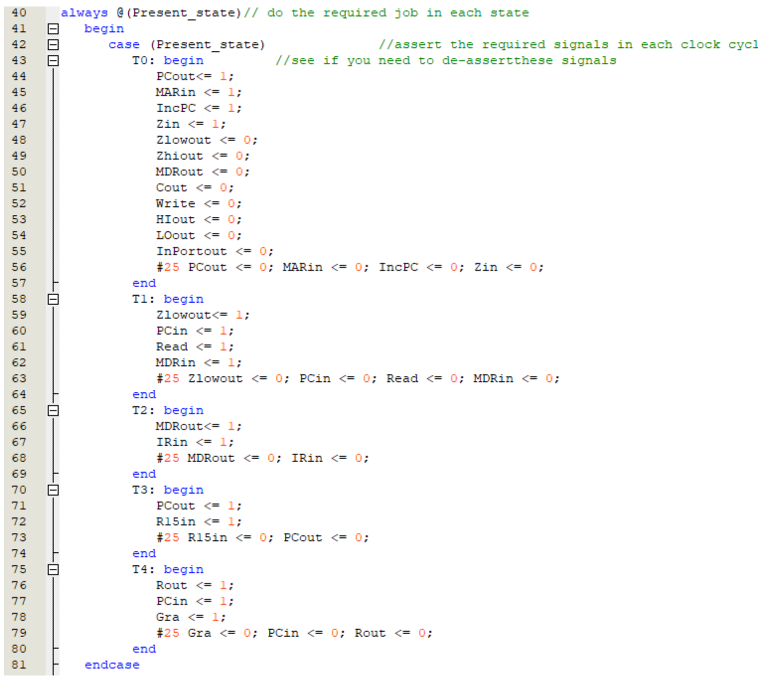
```
         #25 PCout <= 0; Yin <= 0;
      end
      T5: begin
         Cout <= 1;
         Zin <= 1;
         #25 Cout <= 0; Zin <= 0;
      end
      T6: begin
         Zlowout <= 1;
         if (CON_out) begin
         PCin <= 1;
         end
         #25 PCin <= 0; Zlowout <= 0;
      end
   endcase
end
endmodule
```

IN/OUT:



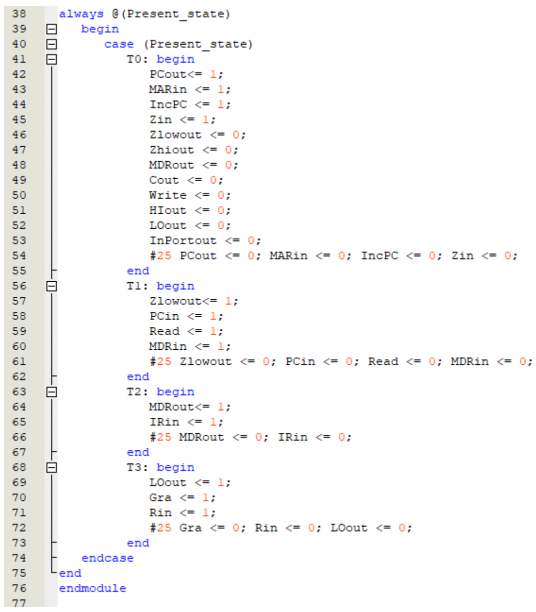
```
always @(Present_state)
   begin
      case (Present_state)
         T0: begin
            PCout<= 1;
            MARin <= 1;
            IncPC <= 1;
            Zin <= 1;
            Zlowout <= 0;
            Zhiout <= 0;
            MDRout <= 0;
            Cout <= 0;
            Write <= 0;
            HIout <= 0;
            LOout <= 0;
            InPortout <= 0;
            #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
         end
         T1: begin
            Zlowout<= 1;
            PCin <= 1;
            Read <= 1;
            MDRin <= 1;
            #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
         end
         T2: begin
            MDRout<= 1;
            IRin <= 1;
            #25 MDRout <= 0; IRin <= 0;
         end
         T3: begin
            InPortout <= 1;
            Gra <= 1;
            Rin <= 1;
            #25 Gra <= 0; Rin <= 0; InPortout <= 0;
         end
   endcase
```

JR/JAL:



```
always @(Present_state)// do the required job in each state
   begin
      case (Present_state)               //assert the required signals in each clock cycl
         T0: begin          //see if you need to de-assertthese signals
            PCout<= 1;
            MARin <= 1;
            IncPC <= 1;
            Zin <= 1;
            Zlowout <= 0;
            Zhiout <= 0;
            MDRout <= 0;
            Cout <= 0;
            Write <= 0;
            HIout <= 0;
            LOout <= 0;
            InPortout <= 0;
            #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
         end
         T1: begin
            Zlowout<= 1;
            PCin <= 1;
            Read <= 1;
            MDRin <= 1;
            #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
         end
         T2: begin
            MDRout<= 1;
            IRin <= 1;
            #25 MDRout <= 0; IRin <= 0;
         end
         T3: begin
            PCout <= 1;
            R15in <= 1;
            #25 R15in <= 0; PCout <= 0;
         end
         T4: begin
            Rout <= 1;
            PCin <= 1;
            Gra <= 1;
            #25 Gra <= 0; PCin <= 0; Rout <= 0;
         end
   endcase
```

MFHI/MFLO:

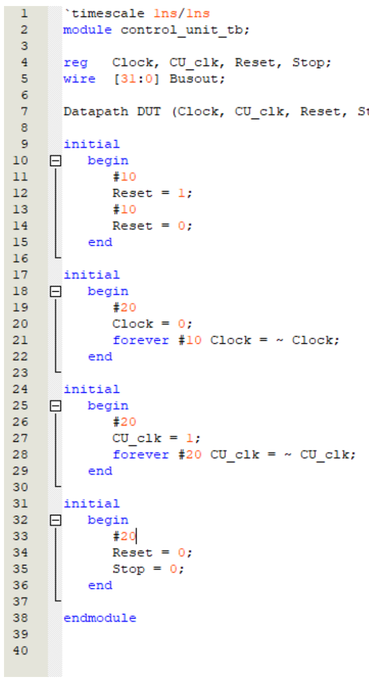


```
always @(Present_state)
   begin
      case (Present_state)
         T0: begin
            PCout<= 1;
            MARin <= 1;
            IncPC <= 1;
            Zin <= 1;
            Zlowout <= 0;
            Zhiout <= 0;
            MDRout <= 0;
            Cout <= 0;
            Write <= 0;
            HIout <= 0;
            LOout <= 0;
            InPortout <= 0;
            #25 PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
         end
         T1: begin
            Zlowout<= 1;
            PCin <= 1;
            Read <= 1;
            MDRin <= 1;
            #25 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
         end
         T2: begin
            MDRout<= 1;
            IRin <= 1;
            #25 MDRout <= 0; IRin <= 0;
         end
         T3: begin
            LOout <= 1;
            Gra <= 1;
            Rin <= 1;
            #25 Gra <= 0; Rin <= 0; LOout <= 0;
         end
   endcase
end
endmodule
```

Phase 3:

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

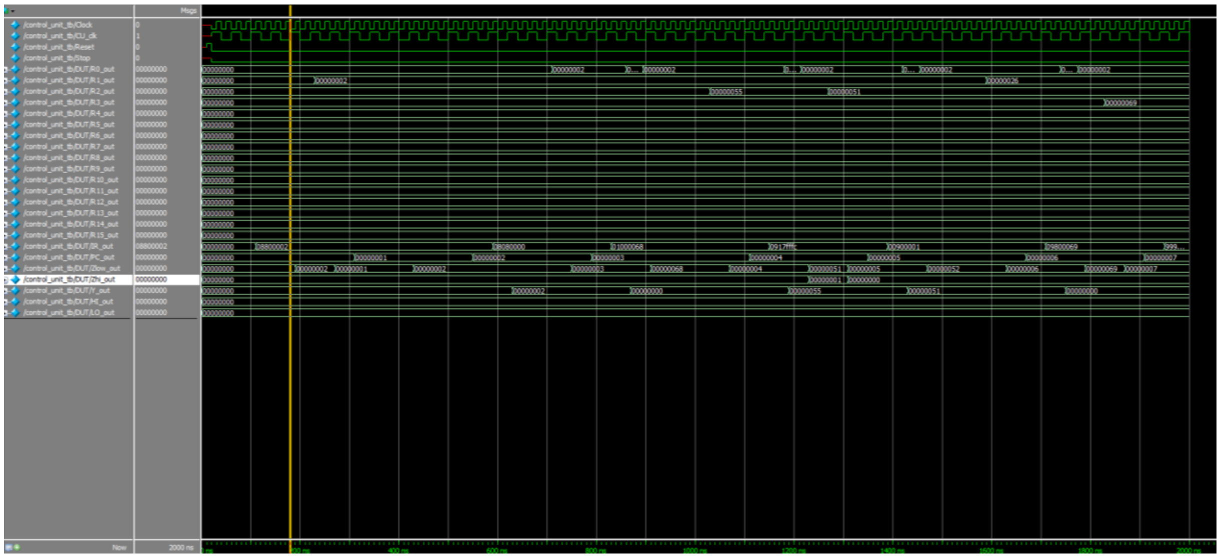
Control_Unit_TB:



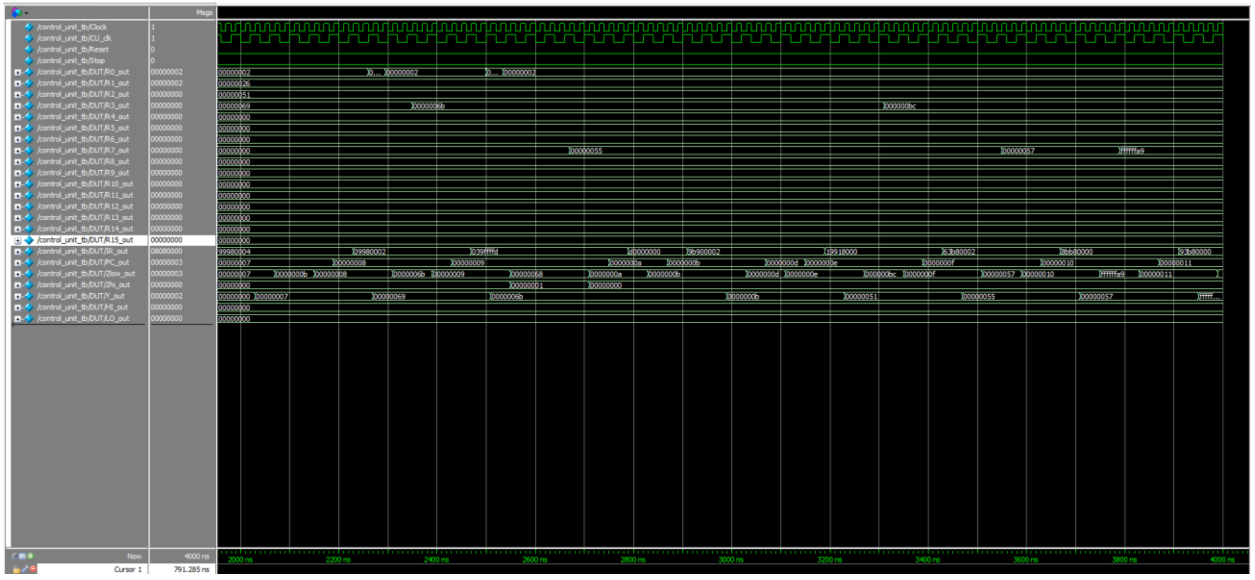
```
`timescale 1ns/1ns
module control_unit_tb;

reg   Clock, CU_clk, Reset, Stop;
wire  [31:0] Busout;

Datapath DUT (Clock, CU_clk, Reset, Stop, Busout);

initial
   begin
      #10
      Reset = 1;
      #10
      Reset = 0;
   end

initial
   begin
      #20
      Clock = 0;
      forever #10 Clock = ~ Clock;
   end

initial
   begin
      #20
      CU_clk = 1;
      forever #20 CU_clk = ~ CU_clk;
   end

initial
   begin
      #20
      Reset = 0;
      Stop = 0;
   end

endmodule
```

```
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
```

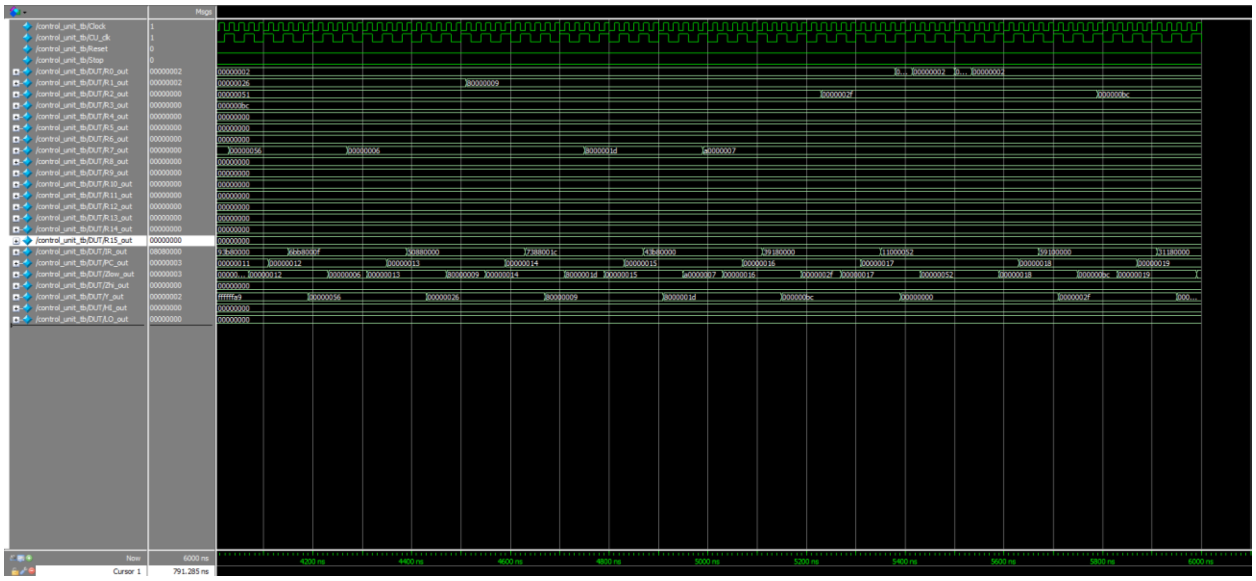
Simulations


/control_unit_tb/Clock
/control_unit_tb/CU_clk
/control_unit_tb/Reset
/control_unit_tb/Stop
/control_unit_tb/DUT/R0_out
/control_unit_tb/DUT/R1_out
/control_unit_tb/DUT/R2_out
/control_unit_tb/DUT/R3_out
/control_unit_tb/DUT/R4_out
/control_unit_tb/DUT/R5_out
/control_unit_tb/DUT/R6_out
/control_unit_tb/DUT/R7_out
/control_unit_tb/DUT/R8_out
/control_unit_tb/DUT/R9_out
/control_unit_tb/DUT/R10_out
/control_unit_tb/DUT/R11_out
/control_unit_tb/DUT/R12_out
/control_unit_tb/DUT/R13_out
/control_unit_tb/DUT/R14_out
/control_unit_tb/DUT/R15_out
/control_unit_tb/DUT/IR_out
/control_unit_tb/DUT/PC_out
/control_unit_tb/DUT/Zlow_out
/control_unit_tb/DUT/Zhi_out
/control_unit_tb/DUT/Y_out
/control_unit_tb/DUT/HI_out
/control_unit_tb/DUT/LO_out
Now
2000 ns

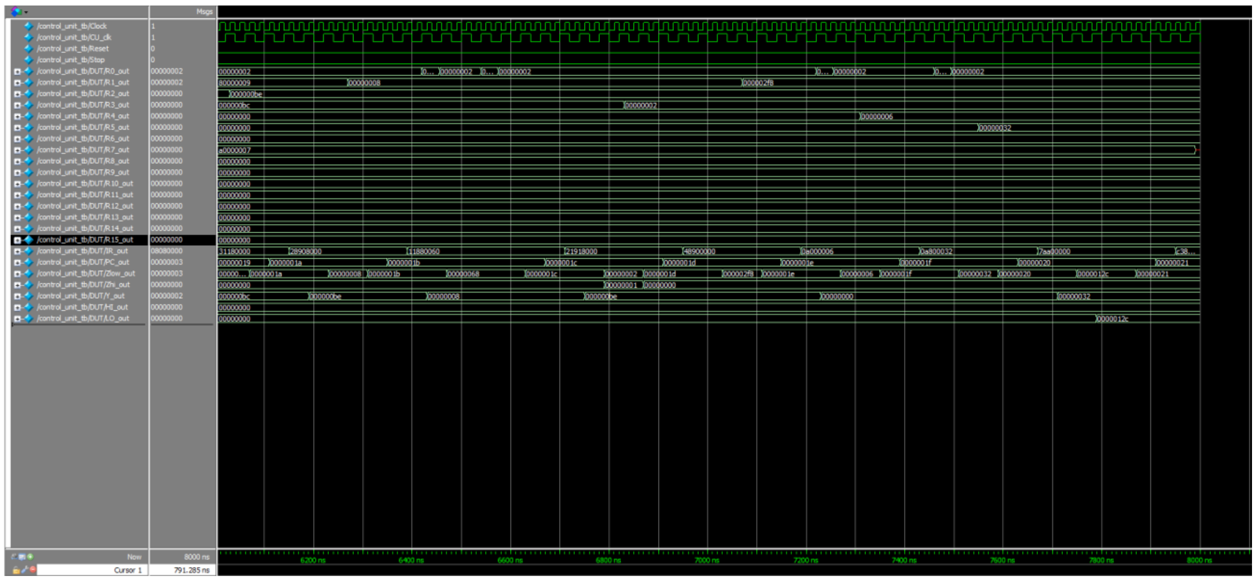
0ns-2000ns


/control_unit_tb/Clock
/control_unit_tb/CU_clk
/control_unit_tb/Reset
/control_unit_tb/Stop
/control_unit_tb/DUT/R0_out
/control_unit_tb/DUT/R1_out
/control_unit_tb/DUT/R2_out
/control_unit_tb/DUT/R3_out
/control_unit_tb/DUT/R4_out
/control_unit_tb/DUT/R5_out
/control_unit_tb/DUT/R6_out
/control_unit_tb/DUT/R7_out
/control_unit_tb/DUT/R8_out
/control_unit_tb/DUT/R9_out
/control_unit_tb/DUT/R10_out
/control_unit_tb/DUT/R11_out
/control_unit_tb/DUT/R12_out
/control_unit_tb/DUT/R13_out
/control_unit_tb/DUT/R14_out
/control_unit_tb/DUT/R15_out
/control_unit_tb/DUT/IR_out
/control_unit_tb/DUT/PC_out
/control_unit_tb/DUT/Zlow_out
/control_unit_tb/DUT/Zhi_out
/control_unit_tb/DUT/Y_out
/control_unit_tb/DUT/HI_out
/control_unit_tb/DUT/LO_out
Now
4000 ns
Cursor 1
791.285 ns

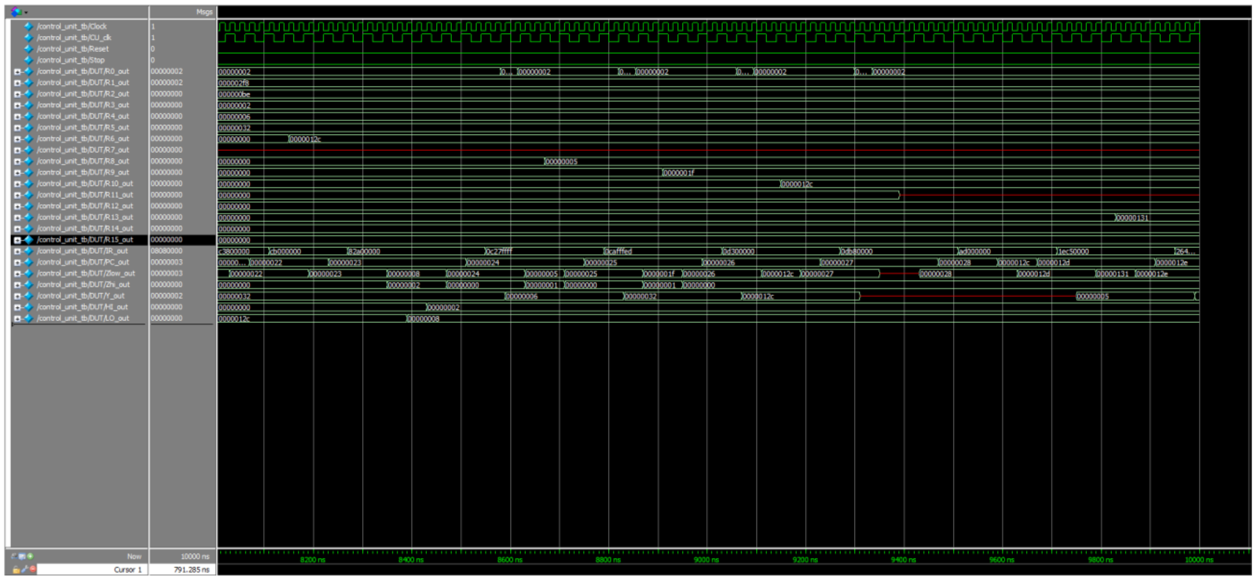
2000ns-4000ns


/control_unit_tb/Clock
/control_unit_tb/CU_clk
/control_unit_tb/Reset
/control_unit_tb/Stop
/control_unit_tb/DUT/R0_out
/control_unit_tb/DUT/R1_out
/control_unit_tb/DUT/R2_out
/control_unit_tb/DUT/R3_out
/control_unit_tb/DUT/R4_out
/control_unit_tb/DUT/R5_out
/control_unit_tb/DUT/R6_out
/control_unit_tb/DUT/R7_out
/control_unit_tb/DUT/R8_out
/control_unit_tb/DUT/R9_out
/control_unit_tb/DUT/R10_out
/control_unit_tb/DUT/R11_out
/control_unit_tb/DUT/R12_out
/control_unit_tb/DUT/R13_out
/control_unit_tb/DUT/R14_out
/control_unit_tb/DUT/R15_out
/control_unit_tb/DUT/IR_out
/control_unit_tb/DUT/PC_out
/control_unit_tb/DUT/Zlow_out
/control_unit_tb/DUT/Zhi_out
/control_unit_tb/DUT/Y_out
/control_unit_tb/DUT/HI_out
/control_unit_tb/DUT/LO_out
Now 6000 ns
Cursor 1 791.285 ns

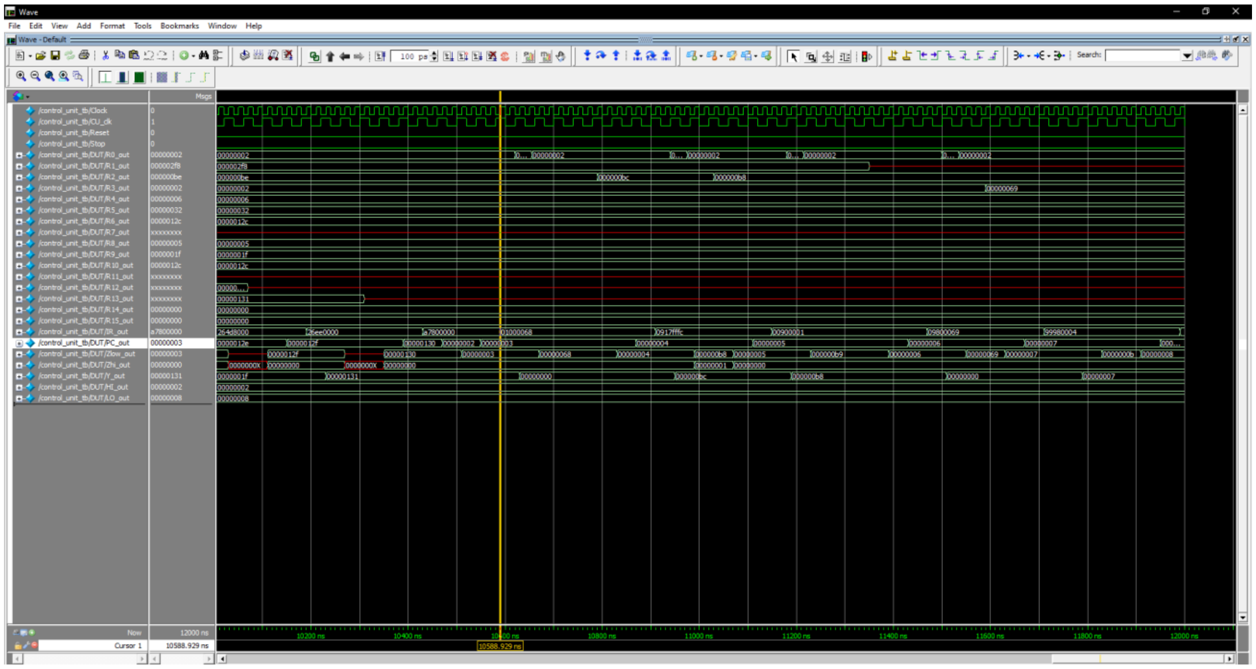
4000ns-6000ns


/control_unit_tb/Clock
/control_unit_tb/CU_clk
/control_unit_tb/Reset
/control_unit_tb/Stop
/control_unit_tb/DUT/R0_out
/control_unit_tb/DUT/R1_out
/control_unit_tb/DUT/R2_out
/control_unit_tb/DUT/R3_out
/control_unit_tb/DUT/R4_out
/control_unit_tb/DUT/R5_out
/control_unit_tb/DUT/R6_out
/control_unit_tb/DUT/R7_out
/control_unit_tb/DUT/R8_out
/control_unit_tb/DUT/R9_out
/control_unit_tb/DUT/R10_out
/control_unit_tb/DUT/R11_out
/control_unit_tb/DUT/R12_out
/control_unit_tb/DUT/R13_out
/control_unit_tb/DUT/R14_out
/control_unit_tb/DUT/R15_out
/control_unit_tb/DUT/IR_out
/control_unit_tb/DUT/PC_out
/control_unit_tb/DUT/Zlow_out
/control_unit_tb/DUT/Zhi_out
/control_unit_tb/DUT/Y_out
/control_unit_tb/DUT/HI_out
/control_unit_tb/DUT/LO_out
Now 8000 ns
Cursor 1 791.285 ns

6000ns-8000ns



8000ns-10000ns



10000ns-10588ns