CPU Design Project Phase 1 Submission

DataPath:

```
module DataPath(
    input wire clock, clear,
    input wire [31:0] Mdatain,
    input wire [4:0] ops,

    input RAout, R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out,
    RYout, RZHIout, RZLOout, PCout, IRout, HIout, LOout, MDRout, PORTout,

    input RAin, R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in, R8in, R9in, R10in, R11in, R12in, R13in, R14in, R15in,
    RYin, RZin, PCin, IRin, HIin, LOin, MDRin, PORTin, Read

    );

wire [31:0] BusMuxOut, BusMuxInRA, BusMuxInR0, BusMuxInR1, BusMuxInR2, BusMuxInR3, BusMuxInR4, BusMuxInR5, BusMuxInR6, BusMuxInR7, BusMuxInR8,
    BusMuxInR9, BusMuxInR10, BusMuxInR11,BusMuxInR12, BusMuxInR13, BusMuxInR14, BusMuxInR15, RYdataout, BusMuxInHI, BusMuxInLO, BusMuxInRZHI, BusMuxInRZLO,
    BusMuxInPC, BusMuxInMDR, BusMuxInPort;

wire [63:0] Zregin;

register RA(clear, clock, RAin, RegisterAImmediate, BusMuxInRA);

register R0(clear, clock, R0in, BusMuxOut, BusMuxInR0);
register R1(clear, clock, R1in, BusMuxOut, BusMuxInR1);
register R2(clear, clock, R2in, BusMuxOut, BusMuxInR2);
register R3(clear, clock, R3in, BusMuxOut, BusMuxInR3);
register R4(clear, clock, R4in, BusMuxOut, BusMuxInR4);
register R5(clear, clock, R5in, BusMuxOut, BusMuxInR5);
register R6(clear, clock, R6in, BusMuxOut, BusMuxInR6);
register R7(clear, clock, R7in, BusMuxOut, BusMuxInR7);
register R8(clear, clock, R8in, BusMuxOut, BusMuxInR8);
register R9(clear, clock, R9in, BusMuxOut, BusMuxInR9);
register R10(clear, clock, R10in, BusMuxOut, BusMuxInR10);
register R11(clear, clock, R11in, BusMuxOut, BusMuxInR11);
register R12(clear, clock, R12in, BusMuxOut, BusMuxInR12);
register R13(clear, clock, R13in, BusMuxOut, BusMuxInR13);
register R14(clear, clock, R14in, BusMuxOut, BusMuxInR14);
register R15(clear, clock, R15in, BusMuxOut, BusMuxInR15);
register RY(clear, clock, RYin, BusMuxOut, RYdataout);
register RZHI(clear, clock, RZin, Zregin[63:32], BusMuxInRZHI);
register RZLO(clear, clock, RZin, Zregin[31:0], BusMuxInRZLO);
register PC(clear, clock, PCin, BusMuxOut, BusMuxInPC);
register IR(clear, clock, IRin, BusMuxOut, BusMuxInIR);
register HI(clear, clock, HIin, BusMuxOut, BusMuxInHI);
register LO(clear, clock, LOin, BusMuxOut, BusMuxInLO);

mdrReg MDR(clear, clock, MDRin, Read, Mdatain, BusMuxOut, BusMuxInMDR);

alu alu(ops, RYdataout, BusMuxOut, Zregin);

//ADD add(RYdataout, BusMuxOut, Zregin);

Bus bus(BusMuxInRA, BusMuxInR0, BusMuxInR1, BusMuxInR2, BusMuxInR3, BusMuxInR4, BusMuxInR5, BusMuxInR6, BusMuxInR7, BusMuxInR8,
    BusMuxInR9, BusMuxInR10, BusMuxInR11,BusMuxInR12, BusMuxInR13, BusMuxInR14, BusMuxInR15, BusMuxInHI, BusMuxInLO, BusMuxInRZHI, BusMuxInRZLO,
    BusMuxInPC, BusMuxInMDR, BusMuxInPort, RAout, R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out,
    R13out, R14out, R15out, RYout, RZHIout, RZLOout, PCout, IRout, HIout, LOout, MDRout, PORTout, BusMuxOut);

endmodule
```

Bus:

```
module Bus (
    input [31:0] BusMuxInRA, BusMuxInR0, BusMuxInR1, BusMuxInR2, BusMuxInR3, BusMuxInR4, BusMuxInR5, BusMuxInR6, BusMuxInR7, BusMuxInR8,
    BusMuxInR9, BusMuxInR10, BusMuxInR11,BusMuxInR12, BusMuxInR13, BusMuxInR14, BusMuxInR15, BusMuxInHI, BusMuxInLO, BusMuxInRZHI, BusMuxInRZLO,
    BusMuxInPC, BusMuxInMDR, BusMuxInPort,

    input RAout, R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out,
    RYout, RZHIout, RZLOout, PCout, IRout, HIout, LOout, MDRout, PORTout,

    output wire [31:0] BusMuxOut
);

reg [31:0]q;

always @ (*) begin
    if(RAout) q = BusMuxInRA;

    if(R0out) q = BusMuxInR0;
    if(R1out) q = BusMuxInR1;
    if(R2out) q = BusMuxInR2;
    if(R3out) q = BusMuxInR3;
    if(R4out) q = BusMuxInR4;
    if(R5out) q = BusMuxInR5;
    if(R6out) q = BusMuxInR6;
    if(R7out) q = BusMuxInR7;
    if(R8out) q = BusMuxInR8;
    if(R9out) q = BusMuxInR9;
    if(R10out) q = BusMuxInR10;
    if(R11out) q = BusMuxInR11;
    if(R12out) q = BusMuxInR12;
    if(R13out) q = BusMuxInR13;
    if(R14out) q = BusMuxInR14;
    if(R15out) q = BusMuxInR15;
    if(PCout) q = BusMuxInPC;
    if(HIout) q = BusMuxInHI;
    if(LOout) q = BusMuxInLO;
    if(MDRout) q = BusMuxInMDR;
    if(RZHIout) q = BusMuxInRZHI;
    if(RZLOout) q = BusMuxInRZLO;
    if(PORTout) q = BusMuxInPort;
end
assign BusMuxOut = q;
endmodule
```

Register file:

```
module register(
   input clear, clock, enable,
   input [31:0]BusMuxOut,
   output wire [31:0]BusMuxIn
);
reg [31:0]q;
initial q = 32'b0;
always @ (posedge clock)
      begin
         if (clear) begin
            q <= {32{1'b0}};
         end
         else if (enable) begin
            q <= BusMuxOut;
         end
      end
   assign BusMuxIn = q[31:0];
endmodule
```

MDR Register file:

```
module mdrReg(
   input clear, clock, MDRin, Read,
   input [31:0] Mdatain, BusMuxOut,
   output [31:0] BusMuxIn
);

reg [31:0] mdrregdata;

initial mdrregdata = 8'h0;
always @ (posedge clock)
      begin
         if (clear) begin
            mdrregdata <= {32{1'b0}};
         end
         else if (MDRin) begin
            case (Read)
//             mdrregdata = 32'h10101010;
               1'b0 : mdrregdata = BusMuxOut;
               1'b1 : mdrregdata = Mdatain;
            endcase
         end
      end

assign BusMuxIn = mdrregdata;

endmodule
```

ALU:

```
module alu(
   input [4:0] ops,
   input [31:0] A, B,
   output reg [63:0] Zregin
);

//reg [63:0] x;

wire [31:0] addresult, andresult, orresult, notresult, negresult, rorresult, rolresult, shlresult, shrresult, shraresult, subresult;

wire [63:0] mulresult, divresult;


ADD ADD(A, B, addresult);
AND_mod AND_mod(A, B, andresult);
OR_mod OR_mod(A, B, orresult);
NOT_mod NOT_mod(A, notresult);
NEG NEG(A, negresult);
MUL MUL(A, B, mulresult);
ROR ROR(A, B, rorresult);
ROL ROL(A, B, rolresult);
SHL SHL(A, B, shlresult);
SHR SHR(A, B, shrresult);
SHRA SHRA(A, B, shraresult);
SUB SUB(A, B, subresult);
DIV DIV(A, B, divresult);

//... continue this

//NEED TO IMPLEMENT AND, OR, SUB, MUL, DIV, SHR, SHRA, SHL, ROR, ROL, NEG, NOT

//addresult or andresult or orresult or notresult or negresult or rorresult or rolresult or shlresult or shrresult or shraresult or mulresult

always @(*) begin
   case(ops)
      5'b00000: Zregin = addresult;
      5'b00001: Zregin = andresult;
      5'b00010: Zregin = orresult;
      5'b00011: Zregin = notresult;
      5'b00100: Zregin = negresult;
      5'b00101: Zregin = mulresult;
      5'b00110: Zregin = rorresult;
      5'b00111: Zregin = rolresult;
      5'b01000: Zregin = shlresult;
      5'b01001: Zregin = shrresult;
      5'b01010: Zregin = shraresult;
      5'b01011: Zregin = divresult;
      5'b01100: Zregin = subresult;
//    5'b01101: x = bmi_13;
//    5'b01110: q = bmi_14;
//    5'b01111: q = bmi_15;
//    5'b10000: q = bmi_16;
//    5'b10001: q = bmi_17;
//    5'b10010: q = bmi_18;
//    5'b10011: q = bmi_19;
//    5'b10100: q = bmi_20;
//    5'b10101: q = bmi_21;
//    5'b10110: q = bmi_22;
//    5'b10111: q = bmi_23;
//    5'b11000: q = bmi_24;
//    5'b11001: q = bmi_25;
//    5'b11010: q = bmi_26;
//    5'b11011: q = bmi_27;
//    5'b11100: q = bmi_28;
//    5'b11101: q = bmi_29;
//    5'b11110: q = bmi_30;
//    5'b11111: q = bmi_31;
   endcase
end


//assign Zregin = x;

endmodule
```

ADD module:

```
// Ripple Carry ADD module
module ADD(A, B, addresult);

input [31:0] A, B;
output [31:0] addresult;

reg [31:0] addresult;
reg [32:0] LocalCarry;

integer i;

always@(A or B)
   begin
      LocalCarry = 32'b0;
      for(i = 0; i < 32; i = i + 1)
      begin
            addresult[i] = A[i]^B[i]^  LocalCarry[i];
            LocalCarry[i+1] = (A[i]&B[i])|(LocalCarry[i]&(A[i]|B[i]));
      end
end
endmodule
```

NEG module:

```
module NEG (A, negresult);

input [31:0] A;
output [31:0] negresult;

reg [31:0] negresult;

integer i;

always @(A)
   begin
      for (i = 0; i < 32; i = i + 1)
      begin
         negresult[i] = ~A[i];
      end
      //two's compliment part
      negresult = negresult + 1;
   end
endmodule
```

Shift Left module:

```
module SHL(

input [31:0] A, B,
output reg [31:0] shlresult

);

always @(A or B)
   begin
      shlresult = A << B;
   end
endmodule
```

Shift Right module:

```
module SHR(

input [31:0] A, B,
output reg [31:0] shrresult

);

always @(A or B)
   begin
      shrresult = A >> B;
   end
endmodule
```

Shift Right Arithmetic Module:

```
module SHRA(

input signed [31:0] A, B,
output reg [31:0] shraresult

);

always @(A or B)
   begin
      shraresult = A >>> B;
   end
endmodule
```

AND module:

```
//AND module
module AND_mod (A, B, andresult);

input [31:0] A, B;
output [31:0] andresult;


reg [31:0] andresult;


integer i;

always @(A or B)
begin
      for (i = 0; i < 32; i = i + 1)
      begin
         andresult[i] <= A[i] & B[i];
      end
end

endmodule
```

OR module:

```
//OR module
module OR_mod (A, B, orresult);

input [31:0] A, B;
output [31:0] orresult;

reg [31:0] orresult;

integer i;

always @(A or B)
   begin
      for (i = 0; i < 32; i = i + 1)
      begin
         orresult[i] = A[i] | B[i];
      end
   end
endmodule
```

Rotate Left module:

```
module ROL(
   input[31:0] A, //value being rotated
   input[31:0] B, //amount of bits A is being rotated
   output reg [31:0] rolresult
);

always @ (A or B)
   begin
      rolresult = ( A << B ) | ( A >> (32-B));
   end
endmodule
```

Rotate Right module:

```
module ROR(
   input[31:0] A, //value being rotated
   input[31:0] B, //amount of bits A is being rotated
   output reg [31:0] rorresult
);

always @ (A or B)
   begin
      rorresult = (A >> B) | (A << (32-B));
   end
endmodule
```

Multiplication module:

```
module MUL(
    input signed [31:0] A, B,
    output [63:0] mulresult
);
    reg[2:0] cc[(32/2)-1: 0]; //carry collection
    reg[31:0] pp[(32/2)-1: 0]; //partial product
    reg [63:0] spp[(32/2)-1: 0]; //signed partial product
    reg [63: 0] product; //final

    wire[32:0] inv_a;
    assign inv_a = {~A[31], ~A} + 1; //2's compiment of A to be used for booth's algo

    integer j,i;

    always @ (A or B or inv_a) begin
       cc[0] = {B[1], B[0], 1'b0}; //concat two first bits, last bit is carry

          for (j=1;j<(32/2);j=j+1)
          begin
             cc[j] = {B[2*j+1], B[2*j], B[2*j-1]};
          end

          //Calculate the partial product for each of the bits depending on the bit-pair encoding found above
          for (j=0; j<(32/2); j=j+1)
          begin
             case(cc[j])

             //+1M
             3'b001, 3'b010 : pp[j] = {A[32-1], A};
             //+2M
             3'b011 : pp[j] = {A, 1'b0};
             //-2M
             3'b100 : pp[j] = {inv_a[32-1:0], 1'b0};
             //-1M
             3'b101, 3'b110 : pp[j] = {inv_a[32-1], inv_a};
             //0
             default : pp[j] = 0;
             endcase
             //Convert them to signed partial products
             spp[j] = $signed(pp[j]) << (j*2);
          end
          product = spp[0];

          //sum up the products to obtain the final result
          for (j=1; j<(32/2); j = j+1)
          begin
             product = product +spp[j];
          end
       end
       assign mulresult = product;
endmodule
```

NOT module:

```
//NOT module
module NOT_mod (A, notresult);

input [31:0] A;
output [31:0] notresult;

reg [31:0] notresult;

integer i;

always @(A)
   begin
      for (i = 0; i < 32; i = i + 1)
      begin
         notresult[i] = ~A[i];
      end
   end
endmodule
```

Subtraction Module:

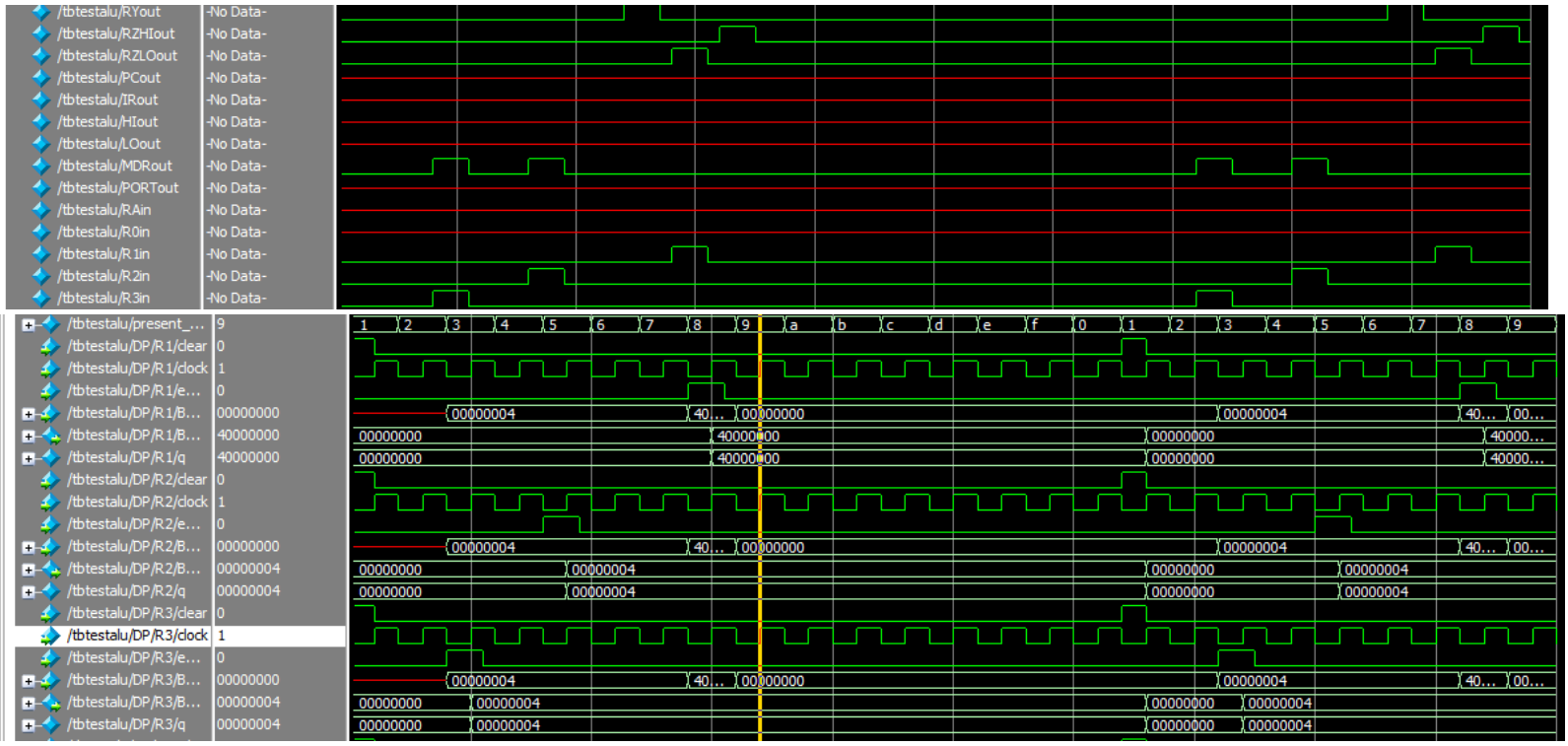
```
module SUB(

input [31:0] A, B,
output reg [31:0] subresult

);

reg [31:0] negB;

reg [32:0] LocalCarry;

integer i;

always@(A or B)
   begin
      negB = ~B + 1;
      LocalCarry = 32'b0;
      for(i = 0; i < 32; i = i + 1)
      begin
            subresult[i] = A[i]^negB[i]^LocalCarry[i];
            LocalCarry[i+1] = (A[i]&negB[i])|(LocalCarry[i]&(A[i]|negB[i]));
      end
end

endmodule
```

Divide Module

```
module DIV(

input [31:0] A, B,
output reg [63:0] Divresult

);

    reg [31:0] Q, Rem, Div;
    reg [31:0] temp_Rem;      //define file specific varibles
    integer y;

    always @(*) begin
        Q = 0;
        Rem = 0;
        Div = B;
        temp_Rem = A; //initilize all veriables starting with A


        for (y = 31; y >= 0; y = y - 1) begin
            //loop through all bits from input register

            //start the non-restorative formula shifting remainder left by one bit
            Rem = Rem << 1;

            Rem[0] = temp_Rem[31]; //prepaire the next bit to be to used from divisor

            temp_Rem = temp_Rem << 1; //set temp remainder after shift, non restorative so no need to reset


            //loop condition to determine if remainder will be positive or negative to determine if 0 or 1 will be added to Q
            if (Rem >= Div) begin
                //calculate new remainder after subtracting the div
                Rem = Rem - Div;
                Q = (Q << 1) | 1; // Set the last bit to 1
            end else begin
                Q = Q << 1; // Set the last bit to 0
            end
        end

        // Adding together quotent with remainder for final
       Divresult[31:0] = Q;
       Divresult[63:32] = Rem;
    end
endmodule
```

Test Bench Used in to create waveforms and test code via manual inputs for 3.1 -3.13:

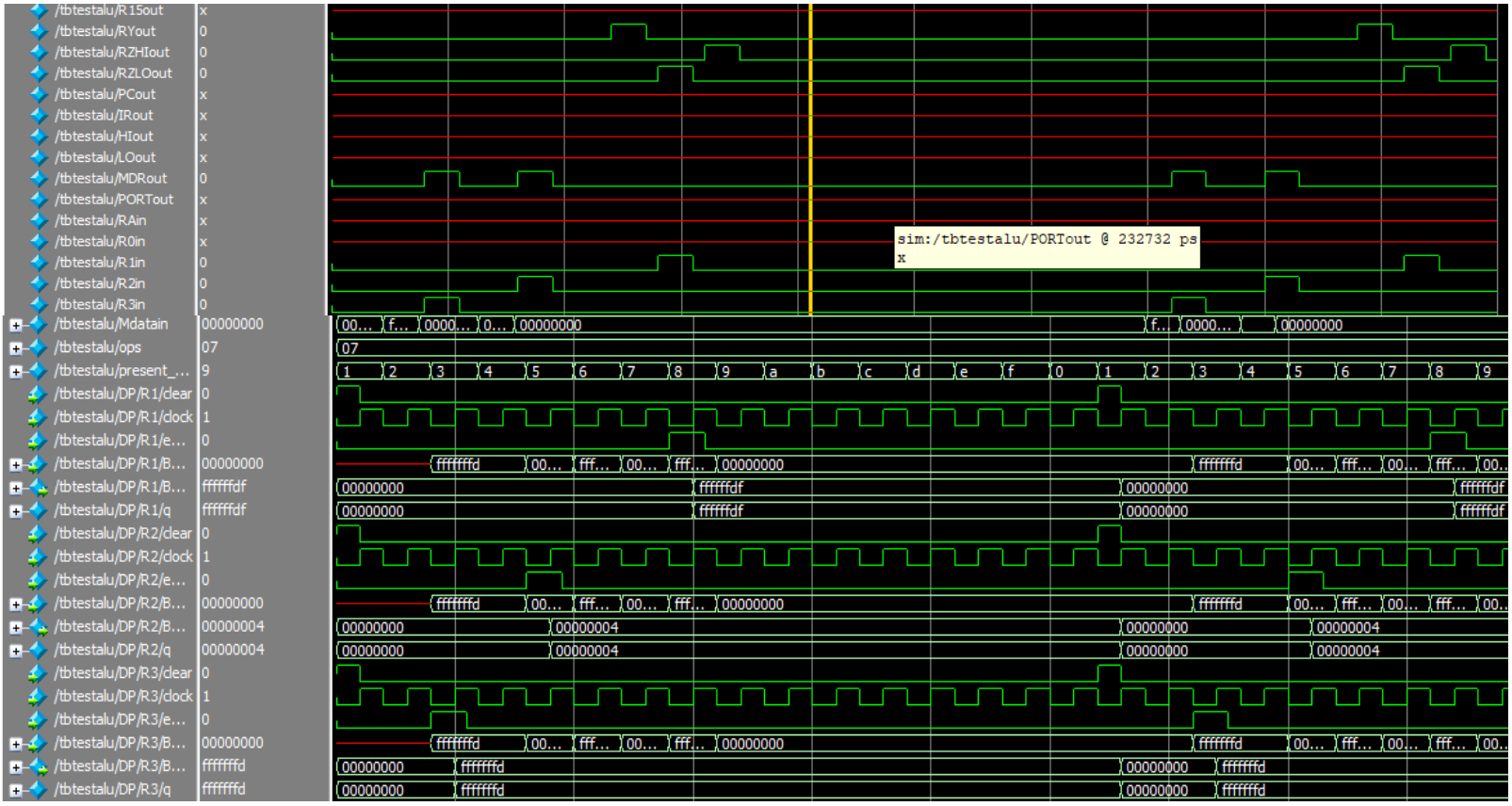
Test Bench and Simulation for 3.10 RoR R1, R2, R3:



Modified testbench control sequence:

```
initial begin clock = 0; present_state = 4'd0; end
always #10 clock = ~clock;
always @ (negedge clock) present_state = present_state + 1;

always @(present_state) begin
   case(present_state)
      init: begin
         clear <= 1; Read <= 1;
         Mdatain <= 8'h00; ops <= 5'b0110;
         RZLOout <= 0; MDRout <= 0; MDRin <= 0; R1out <= 0; R1in <= 0; R2in <= 0; R2out <= 0; R3in <= 0; R3out <=0; RZin <= 0; RYin <= 0; RYout <= 0;
         RZHIout <= 0; HIin <= 0;
         #10 clear <= 0;
      end
      T0: begin
         Mdatain <= 32'b100; MDRin <= 1;
         //Mdatain <= 32'hfffffffd; MDRin <= 1; //for testing SHR AND SHRA
         #15 Mdatain <= 8'h0000; MDRin <= 0;
      end
      T1: begin
         MDRout <= 1; R3in <= 1;
         #15 MDRout <= 0; R3in <= 0;
      end
      T2: begin
         MDRin <= 1; Mdatain <= 32'b100;
         #15 MDRin <= 0; Mdatain <= 8'h0000;
      end
      T3: begin
         MDRout <= 1; R2in <= 1;
         #15 MDRout <= 0; R2in <= 0;
      end
      T4: begin
         R3out <= 1; RYin <=1;
         #15 R3out <= 0; RYin <= 0;
      end
      T5: begin
         RZin <= 1; R2out <= 1; RYout <= 1;
         #15 RZin <= 0; R2out <= 0; RYout <= 0;
      end
      T6: begin
         RZLOout <= 1; R1in <= 1;
         #15 R1in <= 0; RZLOout <= 0;
      end
      T7: begin
         RZHIout <= 1; HIin <= 1;
         #15 HIin <= 0; RZHIout <= 0;
      end
   endcase
end
endmodule
```

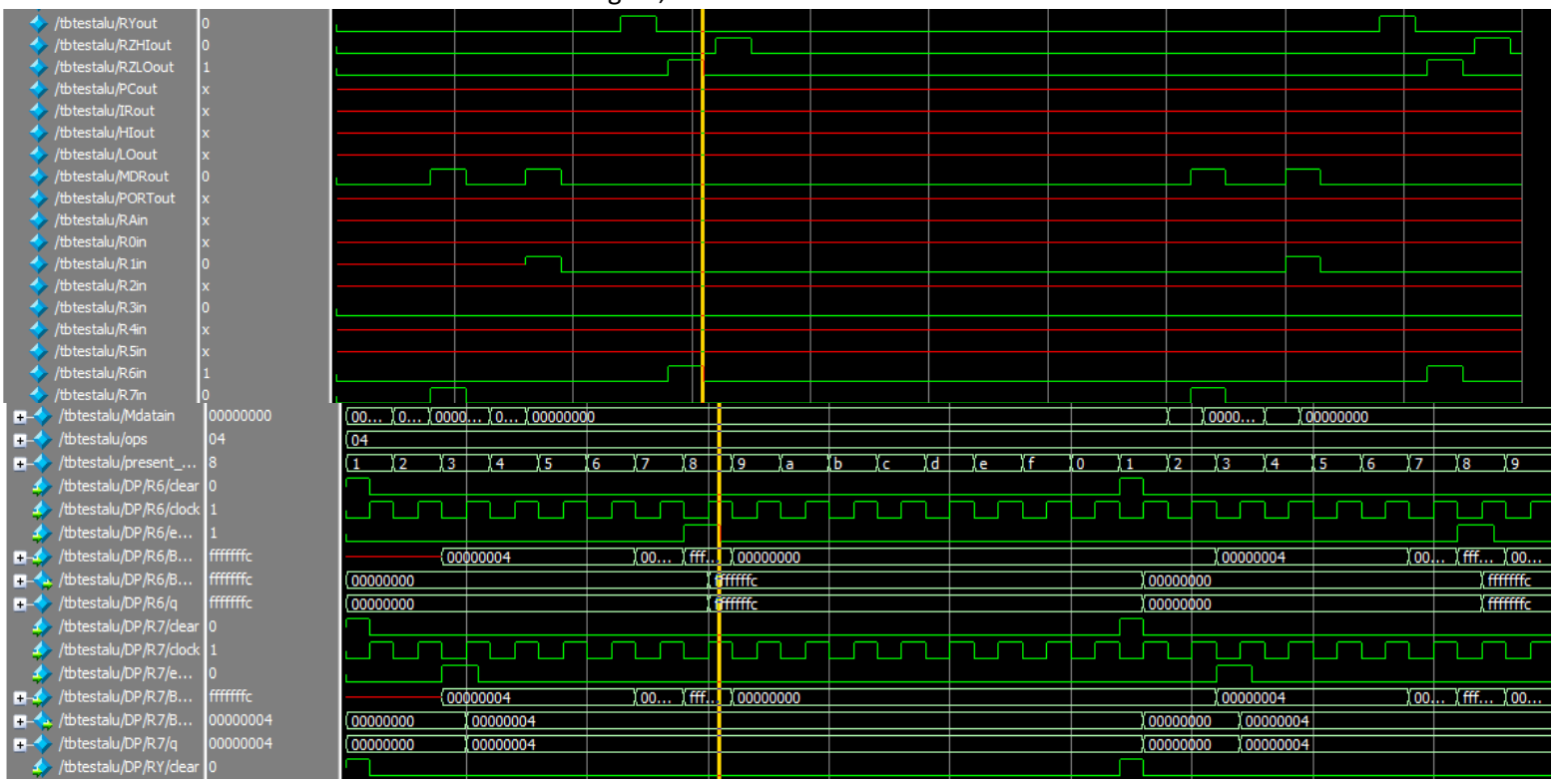
Test Bench and Simulation for 3.11 Rol R1, R2, R3:




```
module tbtestalu();
reg clock, clear;

reg RAout, R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out,
   RYout, RZHIout, RZLOout, PCout, IRout, HIout, LOout, MDRout, PORTout;

reg RAin, R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in, R8in, R9in, R10in, R11in, R12in, R13in, R14in, R15in,
   RYin, RZin, PCin, IRin, HIin, LOin, MDRin, PORTin, Read;

reg [31:0] Mdatain;

reg [4:0] ops;

reg [3:0] present_state;


DataPath DP(
   clock, clear,
   Mdatain, ops,

   RAout, R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out,
   RYout, RZHIout, RZLOout, PCout, IRout, HIout, LOout, MDRout, PORTout,

   RAin, R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in, R8in, R9in, R10in, R11in, R12in, R13in, R14in, R15in,
   RYin, RZin, PCin, IRin, HIin, LOin, MDRin, PORTin, Read
);
```

```
parameter init = 4'd1, T0 = 4'd2, T1 = 4'd3, T2 = 4'd4, T3 = 4'd5, T4 = 4'd6, T5 = 4'd7, T6 = 4'd8, T7 = 4'd9;

initial begin clock = 0; present_state = 4'd0; end
always #10 clock = ~clock;
always @ (negedge clock) present_state = present_state + 1;

always @(present_state) begin
   case(present_state)
      init: begin
         clear <= 1; Read <= 1;
         Mdatain <= 8'h00; ops <= 5'b00111;
         RZLOout <= 0; MDRout <= 0; R1out <= 0; MDRin <= 0; R1in <= 0; R2in <= 0; R2out <= 0; R3in <= 0; R3out <=0; RZin <= 0; RYin <= 0; RYout <= 0;
         RZHIout <= 0; HIin <= 0;
         #10 clear <= 0;
      end
      T0: begin
         //Mdatain <= 32'b1001; MDRin <= 1;
         Mdatain <= 32'hfffffffd; MDRin <= 1; //for testing SHR AND SHRA
         #15 Mdatain <= 8'h0000; MDRin <= 0;
      end
      T1: begin
         MDRout <= 1; R3in <= 1;
         #15 MDRout <= 0; R3in <= 0;
      end
      T2: begin
         MDRin <= 1; Mdatain <= 32'b100;
         #15 MDRin <= 0; Mdatain <= 8'h0000;
      end
      T3: begin
         MDRout <= 1; R2in <= 1;
         #15 MDRout <= 0; R2in <= 0;
      end
      T4: begin
         R3out <= 1; RYin <=1;
         #15 R3out <= 0; RYin <= 0;
      end
      T5: begin
         RZin <= 1; R2out <= 1; RYout <= 1;
         #15 RZin <= 0; R2out <= 0; RYout <= 0;
      end
      T6: begin
         RZLOout <= 1; R1in <= 1;
         #15 R1in <= 0; RZLOout <= 0;
      end
      T7: begin
         RZHIout <= 1; HIin <= 1;
         #15 HIin <= 0; RZHIout <= 0;
      end
   endcase
end
endmodule
```

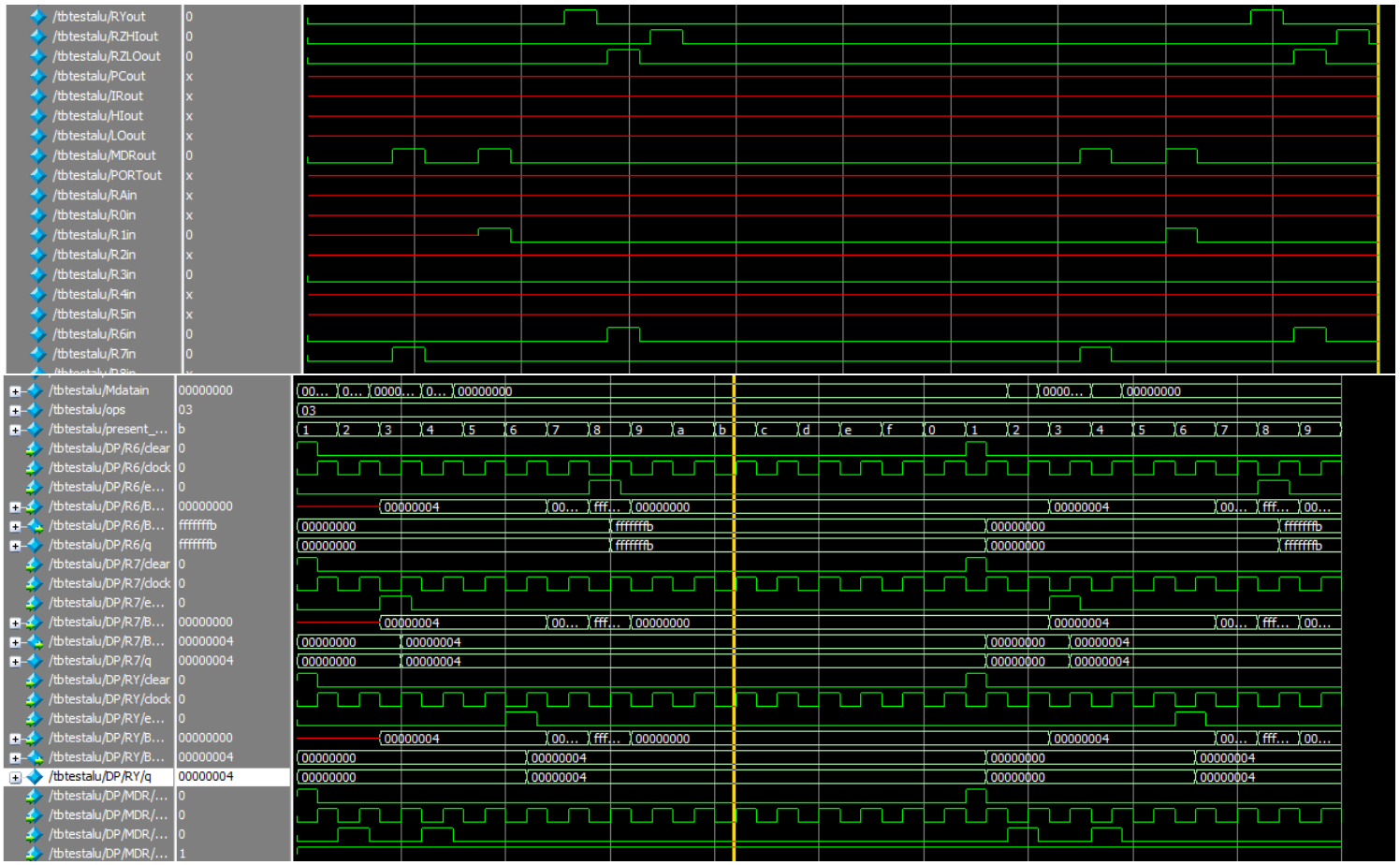
Test Bench and Simulation for 3.12 neg R6, R7:



Changed Control Sequence:

```
parameter init = 4'd1, T0 = 4'd2, T1 = 4'd3, T2 = 4'd4, T3 = 4'd5, T4 = 4'd6, T5 = 4'd7, T6 = 4'd8, T7 = 4'd9;

initial begin clock = 0; present_state = 4'd0; end
always #10 clock = ~clock;
always @ (negedge clock) present_state = present_state + 1;

always @(present_state) begin
   case(present_state)
      init: begin
         clear <= 1; Read <= 1;
         Mdatain <= 8'h00; ops <= 5'b00100;
         RZLOout <= 0; MDRout <= 0; MDRin <= 0; R6out <= 0; R6in <= 0; R7in <= 0; R7out <= 0; R3in <= 0; R3out <=0; RZin <= 0; RYin <= 0; RYout <= 0;
         RZHIout <= 0; HIin <= 0;
         #10 clear <= 0;
      end
      T0: begin
         Mdatain <= 32'b100; MDRin <= 1;
         //Mdatain <= 32'hfffffffd; MDRin <= 1; //for testing SHR AND SHRA
         #15 Mdatain <= 8'h0000; MDRin <= 0;
      end
      T1: begin
         MDRout <= 1; R7in <= 1;
         #15 MDRout <= 0; R7in <= 0;
      end
      T2: begin
         MDRin <= 1; Mdatain <= 32'b100;
         #15 MDRin <= 0; Mdatain <= 8'h0000;
      end
      T3: begin
         MDRout <= 1; R1in <= 1;
         #15 MDRout <= 0; R1in <= 0;
      end
      T4: begin
         R7out <= 1; RYin <=1;
         #15 R7out <= 0; RYin <= 0;
      end
      T5: begin
         RZin <= 1; R2out <= 1; RYout <= 1;
         #15 RZin <= 0; R2out <= 0; RYout <= 0;
      end
      T6: begin
         RZLOout <= 1; R6in <= 1;
         #15 R6in <= 0; RZLOout <= 0;
      end
      T7: begin
         RZHIout <= 1; HIin <= 1;
         #15 HIin <= 0; RZHIout <= 0;
      end
   endcase
end
endmodule
```

Test Bench and Simulation for 3.13 not R6, R7



Modified Testbench control sequence:

```
parameter init = 4'd1, T0 = 4'd2, T1 = 4'd3, T2 = 4'd4, T3 = 4'd5, T4 = 4'd6, T5 = 4'd7, T6 = 4'd8, T7 = 4'd9;

initial begin clock = 0; present_state = 4'd0; end
always #10 clock = ~clock;
always @ (negedge clock) present_state = present_state + 1;

always @(present_state) begin
   case(present_state)
      init: begin
         clear <= 1; Read <= 1;
         Mdatain <= 8'h00; ops <= 5'b00011;
         RZLOout <= 0; MDRout <= 0; MDRin <= 0; R6out <= 0; R6in <= 0; R7in <= 0; R7out <= 0; R3in <= 0; R3out <=0; RZin <= 0; RYin <= 0; RYout <= 0;
         RZHIout <= 0; HIin <= 0;
         #10 clear <= 0;
      end
      T0: begin
         Mdatain <= 32'b100; MDRin <= 1;
         //Mdatain <= 32'hfffffffd; MDRin <= 1; //for testing SHR AND SHRA
         #15 Mdatain <= 8'h0000; MDRin <= 0;
      end
      T1: begin
         MDRout <= 1; R7in <= 1;
         #15 MDRout <= 0; R7in <= 0;
      end
      T2: begin
         MDRin <= 1; Mdatain <= 32'b100;
         #15 MDRin <= 0; Mdatain <= 8'h0000;
      end
      T3: begin
         MDRout <= 1; R1in <= 1;
         #15 MDRout <= 0; R1in <= 0;
      end
      T4: begin
         R7out <= 1; RYin <=1;
         #15 R7out <= 0; RYin <= 0;
      end
      T5: begin
         RZin <= 1; R2out <= 1; RYout <= 1;
         #15 RZin <= 0; R2out <= 0; RYout <= 0;
      end
      T6: begin
         RZLOout <= 1; R6in <= 1;
         #15 R6in <= 0; RZLOout <= 0;
      end
      T7: begin
         RZHIout <= 1; HIin <= 1;
         #15 HIin <= 0; RZHIout <= 0;
      end
   endcase
end
endmodule
```