ELEC 374 Final Lab Report

Jessica Landon

20002031

Nick Dal Farra

20010466

April 5th, 2020

Abstract

The successful design process of a simple 32-bit RISC Computer (Mini SRC) equipped with 2 kB of RAM and 8 general-purpose registers is outlined in this report. The design implements an Arithmetic Logic Unit (ALU) of 12 logical operations and a 32-bit instruction format. The solution is built using the Verilog Hardware Description language (Verilog HDL).

Improvements upon the requested design include the implementation of carry-lookahead adders, bit-pair encodings of the Booth algorithm used for multiplication, and a non-restoring shift register divider. Metrics of Mini SRC performance are not tested but are discussed.

Next steps for the Mini SRC include automating control signals via the creation of a control unit and implementing the design on an evaluation board to enable more thorough testing.

Table of Contents

Project Specifications

The objective of this project was to design and verify a Simple RISC Computer (Mini SRC), however, due to COVID-19, this objective was modified to the design and verification of the Mini SRC's data path.

The data path operates using a 32-bit bus and contains 24 registers total. There are sixteen general purpose 32-bit registers R0 to R15 and two 32-bit registers HI and LO that are dedicated to storing the high and low bits of multiplication and division command outputs. There are also 32-bit instruction and program counter registers, IR and PC. The memory data register (MDR) and memory address register (MAR) are a part of the data path's memory subsystem, acting as inputs and outputs to a 512-word RAM. The data path also contains an I/O state consisting of 32-bit output and input ports, as well as the ALU which is responsible for completing all arithmetic calculations. The ALU uses a 32-bit register Y to hold one of its inputs and a 64-bit Z register to store its output.

The Mini SRC supports 32-bit instructions. Implemented in this lab project are load/store instructions, the arithmetic instructions *addi, andi* and *ori,* branch instructions, jump instructions, special instructions *mfhi* and *mflo,* and input/output instructions. These instructions were implemented in direct, indirect and immediate format.

Project Design and Implementation

Phase 1

Registers

All general-purpose registers, as well as the MAR, PC, IR, HI and LO registers, are of the same design, where a 32-bit input is captured according to a *Rin* enable signal. The exceptions are R0, the MDR and the Z register, which have additional features. R0 takes the control signal *BAout* as an input, which gates 0's onto the bus if R0 is selected or the contents of the register if R1-R15 is selected. The MDR is contained in its own wrapper module, which also contains a multiplexer that selects either the bus contents or the memory contents as an input to the MDR depending on whether a *read* signal is low or high, respectively. The Z register is 64 bits in width and sends its high and lower bits to the HI and LO registers. The decision was made to have Z be one 64-bit register instead of having two separate 32-bit registers to avoid additional logic requirements within the ALU to split its output into two registers.

32-Bit Bus

Our Mini SRC design implements a single bus model. The bus connects nearly every element of the data path and enables the transfer of information between components. If data is the blood of a computer, then the bus is its circulatory system.

The device currently speaking on the bus is controlled by 32 control signals which feed into encode and select multiplexers. While 24 separate registers and devices are always connected to the bus, only the device whose control signal is enabled by the control unit will have its output broadcasted to all other devices on the bus.

Design considerations for the future include increasing the number of busses in the data path. This would allow for multiple simultaneous communication channels to exist, which would decrease the number of clock cycles spent by devices waiting to access an available bus.

ALU

The ALU implements two 32-bit Carry-Lookahead Adders (CLA): one to execute addition operations and one to execute subtraction operations. These adders cascade two sixteen-bit CLA's, which cascade four 4-bit CLA's. The decision to use CLA's instead of Ripple-Carry Adders was to increase the speed of the addition operation. The ALU uses a 32x32 Booth algorithm with bit-pair encoding to execute multiplication operations and a non-restoring division algorithm for division operations. The decisions to implement these algorithms were to increase speed and efficiency.

Phase 2

RAM

The RAM was built using a 512-entry vector of 32-bit Verilog register structures. An address value is supplied to RAM by the Memory Address Register (MAR). Although the MAR is a 32-bit register, because the RAM consists of 512 = 2^8 registers, only the bottom 8 bits of the MAR are used in addressing data.

The decision to implement RAM manually as opposed to using pre-built modules was made to simplify understanding and troubleshooting the RAM. There was no guarantee that pre-built modules would provide access to their code, and the implementation of RAM proved to be relatively simple regardless. However, this design choice appears to prevent the synthetization of the project onto a chip. When trying to synthesize RAM on the chip, bidirectional RAM signals were not supported. Given more time, RAM design would have been adjusted to enable synthetization on a chip.

Memory used in instruction testing is written directly into the RAM module definition and is changed manually for different instruction tests. In the future, memory initialization files (.mif files) should be used to reduce the overhead time required to alter the RAM definition between tests.

Select and Encode Logic

The select and encode logic within the data path generates all of the general-purpose registers' enable control signals. It decodes the current instruction contained in IR to obtain the registers used in the operation. Input signals *Gra, Grb* and *Grc* are then used to generate a 16-bit control vector with a single high bit denoting the register to be enabled. *Rin* and *Rout* input signals indicate whether the specified register should be enabled to capture input data or send its data on the bus.

"CON FF" Logic

CON FF logic produces a control signal whenever a conditional branch instruction is called from memory. The CON FF logic checks the branching condition in the information register (IR) and validates whether the contents of a specified register meet this branching condition. Register contents are provided by the bus. The four possible branching conditions are "==0", "!=0", ">=0", and "<0".

The results of the condition check are stored in a D-flip flop. Input to the flip-flop is enabled by a CONin signal, which is supplied the control unit when a condition check must be made. The final output of the CON FF logic is sent to the control unit, which interprets the condition check appropriately.

I/O Unit

The I/O unit consists of two 32-bit registers. The out-port register accepts data from the bus and sends it to the output unit and the in-port register accepts data from the input unit and sends it on the bus. The in-port is designed to accept a *strobe* signal from the input device. This design decision was made so that the same 32-bit register design could be used for the in-port register as for the other registers in the data path, with the *strobe* signal acting as an enable to indicate that data is available. Both the in-port and out-port registers are synchronous.

Challenges

During the design of the data path module, one of the challenges that was faced was how to connect the bidirectional port for RAM data as both an input and an output for the MDR. This challenge was addressed using the following code:

```
assign ram_data = (read == 1)? 32'bz : MDR;

assign Mdatain = ram_data;
```

This ensures that `ram_data` acts as an input to the MDR unless `read` is asserted, in which case it accepts output from the MDR. The ternary statement that conditionally assigns a high impedance value to `ram_data` prevents the implementation of circular logic.

Evaluation Results

Percentage Chip Area Used

During Phase 1, there were many control signals which were not managed by any control unit and had to be manipulated directly through testbenches. As such, neither the DE0 Cyclone III, DE0-CG Cyclone V, or any of the other available virtual chips could support the number of I/O ports required to synthesize the data path.

During Phase 2, chip synthesis was prevented by an undetermined feature of the RAM. In the future, this functionality would be fixed by following the recommended design practices in Chapter 7 of the Quartus II handbook, as outlined by this Intel Quartus help documentation.

Max Frequency of Operation in Simulation

For reasons previously mentioned, synthesis on the chip was not possible, so maximum frequency of operation was not measurable.

Conclusions

The team was disappointed by not having been able to synthesize their SRC design virtually or on a chip. However, the team was happy to demonstrate that all Phase 1 and 2 components were functionally successful during testing, as was shown in the reports.

It is hypothesized that our solution would have had a below-average maximum frequency of operation during simulations. Even though certain operations of the SRC were optimized for fast performance (namely addition and multiplication), the decision to implement a single-bus design and a non-restoring shift division module in the SRC would severely limit the speed of the division operation and the communication channel capacity between components. Communication between components happens

so frequently that a small improvement in communication speed between components would likely lead to significant improvements in the computational speed (flops) of the Mini SRC.

Conclusions and Next Steps

This project accomplished the objective of implementing a data path for a Mini SRC. If this project were to continue, the next steps would consist of designing the control unit for the SRC and testing it with the SRC data path. The SRC would also be implemented on a DE0 evaluation board for testing purposes. This would allow for further evaluation of the design, including timing analysis, average CPI and percentage of chip area used.

Appendix

Design Modules

ALU

```
module Alu (input clk, input AND, OR, ADD, SUB, MUL, DIV, SHR, SHL, ROR, ROL, NEG, NOT, IncPC, input [31:0] A, B, output [63:0] C);

  // Different wires to hold outputs of units

  wire [63:0] C_mult;

  wire [31:0] C_add;

  wire [31:0] C_sub;

  wire [63:0] C_div;


  wire       diverror;


  // Wires to hold G, P, and C_32 from addition op

  // Not ever used? may get rid of them from 32-bit adder

  wire       G;

  wire       P;

  wire       c32;


  // For use in subtraction

  reg [31:0]  neg_B;


  reg [63:0]  ALU_result;


  Multiplier mult(A, B, MUL, C_mult);

  Cla_32 adder(A, B, 1'b0, G, P, c32, C_add);

  Cla_32 subtractor(A,~B, 1'b1, G, P, c32, C_sub);

  Divider32bit divider(A, B, C_div, diverror);
```

```
assign C = ALU_result;


always @(posedge clk)
  begin
        if (AND == 1) begin
          ALU_result <= {32'b0, A & B};
        end
        else if (OR == 1) begin
          ALU_result <= {32'b0, A | B};
        end
        else if (ADD == 1) begin
          ALU_result <= (C_add[31] == 1'b0)? {32'b0, C_add} : {32'hFFFFFFFF, C_add};
        end
        else if (SUB == 1) begin
          ALU_result <= (C_sub[31] == 1'b0)? {32'b0, C_sub} : {32'hFFFFFFFF, C_sub};
        end
        else if (MUL == 1) begin
          #180
          ALU_result <= C_mult;
        end
        else if (DIV == 1) begin
          ALU_result <= C_div;
        end
        else if (SHR == 1) begin
          ALU_result <= {32'b0, A >> B};
        end
        else if (SHL == 1) begin
          ALU_result <= {32'b0, A << B};
        end
```

```
        else if (ROR == 1) begin
          ALU_result <= {32'b0, (A >> B) | (A << (32 - B))};
        end
        else if (ROL == 1) begin
          ALU_result <= {32'b0, (A << B) | (A >> (32 - B))};
        end
        else if (NEG == 1) begin
          ALU_result <= {32'b0, ~B + 1'b1};
        end
        else if (NOT == 1) begin
          ALU_result <= {32'b0, ~B};
        end
        else if (IncPC == 1) begin
          ALU_result <= {32'b0, B + 32'h00000004};
        end
    end

endmodule
```

And4Bit

```
module And4Bit(input [3:0] x, input y, output [3:0] z);
  and(z[0], x[0], y);
  and(z[1], x[1], y);
  and(z[2], x[2], y);
  and(z[3], x[3], y);
endmodule // 4And
module Bit_stage (input x, y, c0, output G, P, s);

        assign G = x & y;
        assign P = x ^ y;
```

```
        assign s = P ^ c0;


endmodule
```

Booth_encoder

```
module Booth_encoder(input [2:0] bits, input [31:0] M,  output [63:0] P);
  reg [63:0] partial;
  reg [63:0] M1;

  // Output partial according to chart on slide 38 of Computer Arithmetic
  always @(bits) begin
    if (bits == 3'b000 || bits == 3'b111)
        partial = 64'b0;
    else if (bits == 3'b001 || bits == 3'b010)
        partial = (M[31] == 1'b0)? {32'b0, M} : {32'hFFFFFFFF, M};
    else if (bits == 3'b011)
        partial = (M[31] == 1'b0)? {32'b0, M << 1} : {32'hFFFFFFFF, M << 1};
    else if (bits == 3'b100) begin
         M1 = (M[31] == 1'b0)? {32'b0, M << 1} : {32'hFFFFFFFF, M << 1};
         partial = ~M1 + 1;
    end
    else if (bits == 3'b101 || bits == 3'b110) begin
        M1 = (M[31] == 1'b0)? {32'b0, M} : {32'hFFFFFFFF, M};
        partial = ~M1 + 1;
    end
  end // always @ (bits)

  assign P = partial;
```

```
endmodule // Booth_encoder
```

Bus32Bit

```
module Bus32bit(

input i0out, i1out, i2out, i3out, i4out, i5out, i6out, i7out, i8out, i9out, i10out, i11out, i12out, i13out, i14out, i15out, i16out, i17out, i18out, i19out, i20out, i21out, i22out, i23out, i24out, i25out, i26out, i27out, i28out, i29out, i30out, i31out,

input [31:0] i0, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31,

output [31:0] busmux_out);


        wire [4:0] enc_out;


        Encoder32to5 enc (i0out, i1out, i2out, i3out, i4out, i5out, i6out, i7out, i8out, i9out, i10out, i11out, i12out, i13out, i14out, i15out, i16out, i17out, i18out, i19out, i20out, i21out, i22out, i23out, i24out, i25out, i26out, i27out, i28out, i29out, i30out, i31out, enc_out);

        Mux32bit32to1 busmux (enc_out, i0, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, busmux_out);


Endmodule
```

Cla_16

```
module Cla_16(input [15:0] x, y, input c0, output G, P, c16, output [15:0] s);

  wire g0;

  wire p0;

  wire g1;

  wire p1;

  wire g2;

  wire p2;

  wire g3;

  wire p3;

  wire c4;

  wire c8;
```

```
  wire c12;

  // Send the operands down to the four 4-bit adders in four 4-bit chunks
  Cla_4 cla1(x[3:0], y[3:0], c0, g0, p0, c4, s[3:0]);
  Cla_4 cla2(x[7:4], y[7:4], c4, g1, p1, c8, s[7:4]);
  Cla_4 cla3(x[11:8], y[11:8], c8, g2, p2, c12, s[11:8]);
  Cla_4 cla4(x[15:12], y[15:12], c12, g3, p3, c16, s[15:12]);

  assign P = p0 & p1 & p2 & p3;
  assign G = g3 | (p3 & g2) | (p3 & p2 & g1) | (p3 & p2 & p1 & g0);

endmodule // Cla_16
```

Cla_32

```
module Cla_32(input [31:0] x, y, input c0, output G, P, c32, output [31:0] s);
  wire g0;
  wire p0;
  wire g1;
  wire p1;
  wire c16;

  // Send the operands down to the 16-bit adder in 16 bit halves
  Cla_16 cla1(x[15:0], y[15:0], c0, g0, p0, c16, s[15:0]);
  Cla_16 cla2(x[31:16], y[31:16], c16, g1, p1, c32, s[31:16]);

  assign P = p0 & p1;
  assign G = g1 | (p1 & g0);

endmodule
```

Cla_4

```
module Cla_4(input [3:0] x, y, input c0, output G, P, c4, output [3:0] s);
  wire g0;
  wire p0;
  wire g1;
  wire p1;
  wire g2;
  wire p2;
  wire g3;
  wire p3;
  wire c1;
  wire c2;
  wire c3;

  // Send each bit of x and y down to the bit stage cell
  Bit_stage bit_stage1(x[0], y[0], c0, g0, p0, s[0]);
  Bit_stage bit_stage2(x[1], y[1], c1, g1, p1, s[1]);
  Bit_stage bit_stage3(x[2], y[2], c2, g2, p2, s[2]);
  Bit_stage bit_stage4(x[3], y[3], c3, g3, p3, s[3]);

  // Just the equations for the CLA logic from course reader
  //and(P, p0, p1, p2, p3);

        assign P = p0 & p1 & p2 & p3;
  assign G = g3 | (p3 & g2) | (p3 & p2 & g1) | (p3 & p2 & p1 & g0);
  assign c1 = g0 | (p0 & c0);
  assign c2 = g1 | (p1 & g0) | (p1 & p0 & c0);
  assign c3 = g2 | (p2 & g1) | (p2 & p1 & g0) | (p2 & p1 & p0 & c0);
  assign c4 = g3 | (p3 & g2) | (p3 & p2 & g1) | (p3 & p2 & p1 & g0) | (p3 & p2 & p1 & p0 & c0);
```

```
endmodule
```

Clocked_dff

```
module clocked_dff(output reg q, output qbar, input D, CONin);


	assign qbar = ~q;


	initial begin
		q <= 0;
	end


	always @( * ) begin
		if (CONin == 1)
			if (D == 0)
				q <= 0;
			else if (D == 1)
				q <= 1;
	end


endmodule
```

Conff_logic

```
module conff_logic(output out, input [31:0] busin, ir, input CONin
	,output w1, w2, w3, w4 // REMOVE
);


	wire brmi, brpl, brnz, brzr; // C2 = 3, 2, 1, or  0.
	wire qbar;


	// automtically handles decoding
	decoder_2to4 decoder (brzr, brnz, brpl, brmi, ir[20:19]);
```

```
        and(w1, (busin == 0), brzr); // if zero
        and(w2, (busin != 0), brnz); // if nonzero
        and(w3, (busin[31] == 0), brpl); // if >= 0
        and(w4, (busin[31] == 1), brmi); // if < 0
        or(w5, w1, w2, w3, w4); // branching condition met

        clocked_dff dff (out, qbar, w5, CONin);

endmodule
```

Datapath

```
module Datapath(input Cout, HIout, LOout, Zhighout, Zlowout, PCout, MDRout, BAout, Inportout, input
clear, clk, read, write, PCin, IRin, MARin,
Yin, HIin, LOin, Zin, MDRin, Gra, Grb, Grc, Rin, Rout, strobe, OutPort, input AND, OR, ADD,
SUB, MUL, DIV, SHR, SHL, ROR, ROL, NEG, NOT, IncPC, input [31:0] InPortIn,
output [31:0] R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, Hi, Lo, PC,
bus_mux_out, IR, MAR, C_sign_ext, InPortOutput, OutPortOut, Mdatain, MDR, output [63:0] Z, ALUout,
output [15:0] Rins, Routs, inout [31:0] ram_data);

  wire [31:0] Yout;

  wire      conff_out; // not sure where this is supposed to go

  // General registers
  R0 r0 (clear, clk, Rins[0], BAout, bus_mux_out, R0);
  Register32 r1 (clear, clk, Rins[1], bus_mux_out, R1);
  Register32 r2 (clear, clk, Rins[2], bus_mux_out, R2);
  Register32 r3 (clear, clk, Rins[3], bus_mux_out, R3);
  Register32 r4 (clear, clk, Rins[4], bus_mux_out, R4);
```

```
Register32 r5 (clear, clk, Rins[5], bus_mux_out, R5);
Register32 r6 (clear, clk, Rins[6], bus_mux_out, R6);
Register32 r7 (clear, clk, Rins[7], bus_mux_out, R7);
Register32 r8 (clear, clk, Rins[8], bus_mux_out, R8);
Register32 r9 (clear, clk, Rins[9], bus_mux_out, R9);
Register32 r10 (clear, clk, Rins[10], bus_mux_out, R10);
Register32 r11 (clear, clk, Rins[11], bus_mux_out, R11);
Register32 r12 (clear, clk, Rins[12], bus_mux_out, R12);
Register32 r13 (clear, clk, Rins[13], bus_mux_out, R13);
Register32 r14 (clear, clk, Rins[14], bus_mux_out, R14);
Register32 r15 (clear, clk, Rins[15], bus_mux_out, R15);

// Special registers
Register32 pc (clear, clk, PCin, bus_mux_out, PC);
Register32 ir (clear, clk, IRin, bus_mux_out, IR);
Register32 mar (clear, clk, MARin, bus_mux_out, MAR);
Register32 y (clear, clk, Yin, bus_mux_out, Yout);
Register32 hi (clear, clk, HIin, bus_mux_out, Hi);
Register32 lo (clear, clk, LOin, bus_mux_out, Lo);

// 64 bit reg for ALU results
Register64 z (clear, clk, Zin, ALUout, Z);

MDR_unit mdr (read, clk, MDRin, clear, bus_mux_out, Mdatain, MDR);

Bus32bit bus (.i0out(Routs[0]), .i1out(Routs[1]), .i2out(Routs[2]), .i3out(Routs[3]), .i4out(Routs[4]),
        .i5out(Routs[5]), .i6out(Routs[6]), .i7out(Routs[7]), .i8out(Routs[8]), .i9out(Routs[9]),
        .i10out(Routs[10]), .i11out(Routs[11]), .i12out(Routs[12]), .i13out(Routs[13]),
        .i14out(Routs[14]), .i15out(Routs[15]), .i16out(HIout), .i17out(LOout), .i18out(Zhighout),
```

```
        .i19out(Zlowout), .i20out(PCout), .i21out(MDRout), .i22out(Inportout), .i23out(Cout),
        .i24out(), .i25out(), .i26out(), .i27out(), .i28out(), .i29out(), .i30out(), .i31out(), .i0(R0),
        .i1(R1), .i2(R2), .i3(R3), .i4(R4), .i5(R5), .i6(R6), .i7(R7), .i8(R8), .i9(R9), .i10(R10), .i11(R11),
        .i12(R12), .i13(R13), .i14(R14), .i15(R15), .i16(Hi), .i17(Lo), .i18(Z[63:32]), .i19(Z[31:0]),
        .i20(PC), .i21(MDR), .i22(InPortOutput), .i23(C_sign_ext), .i24(), .i25(), .i26(),
        .i27(), .i28(), .i29(), .i30(), .i31(), .busmux_out(bus_mux_out));

  Alu alu (clk, AND, OR, ADD, SUB, MUL, DIV, SHR, SHL, ROR, ROL, NEG, NOT, IncPC, Yout, bus_mux_out,
ALUout);

  Sel_Enc sel_enc(Gra, Grb, Grc, Rin, Rout, BAout, IR, Rins, Routs, C_sign_ext);

        conff_logic conff(.out(conff_out), .busin(bus_mux_out), .ir(IR), .clk(clk));

  Inport inport(clear, clk, strobe, InPortIn, InPortOutput);

  Outport outport(clear, clk, OutPort, bus_mux_out, OutPortOut);

  RAM ram(read, write, ram_data, MAR);

        assign ram_data = (read == 1)? 32'bz : MDR;

        assign Mdatain = ram_data;

endmodule // Datapath
```

Decoder_2to4

```
module decoder_2to4 (output brzr, brnz, brpl, brmi, input [1:0] in);

        assign brzr = (!in[1]) && (!in[0]);
```

```
    assign brnz = (!in[1]) && (in[0]);
    assign brpl = (in[1]) && (!in[0]);
    assign brmi = (in[1]) && (in[0]);

endmodule
```

Divider32Bit

```
module Divider32bit (input [31:0] Qin, Min, output [63:0] result, output diverror);

    // REMOVE TEST

    /*Implements non-restoring division on 32-bit 2's complement values. Q is the dividend,
    M is the divisor (so we're calculating Q/M). Upper 32 result bits are the remainder, and
    the lower 32 hold the value of the quotient. */

    reg [31:0] Q; // Dividend
    reg [31:0] M; // Divisor
    reg [32:0] A; // Holds remainder when finished. Contains extra sign bit.

    reg Qisneg;
    reg Misneg;
    reg [31:0] Mneg; // needs to be same size as A??

    reg divbyzero; // bonus points???

    integer i;

    always @(*) begin

        // TODO: add divby0 output
```

```
if (Min == 0)
        divbyzero = 1;
else
        divbyzero = 0;

// Make Q positive Qin, remember Qin sign
Qisneg = Qin[31];

if (Qin[31] == 1) // condition on input to reduce gate delays
        Q = Qin*(-1);
else
        Q = Qin;

// Make M positive Min, remember Min sign
Misneg = Min[31];

if (Min[31] == 1) begin // consider generating both and using MUX select to reduce delay
        Mneg = Min;
        M = Min*(-1);
end else begin
        M = Min;
        Mneg = Min*(-1);
end


// Perform non-restoring division
A = 0;

for (i=0; i<32; i = i + 1) begin
```

```
                A = A*2;            A[0] = Q[31]; Q = Q*2; // Left shifts

                if (A[32] == 0)
                        A = A + {1'b1,Mneg}; // Perform sign extension
                else
                        A = A + {1'b0,M}; // Late restore

                // Set q0
                if (A[32] == 1)
                        Q[0] = 0;
                        //A = A + M; // leave proper positive remainder // YOU DO NOT BELONG
HERE
                else
                        Q[0] = 1;
        end

        if (A[32] == 1)
                A = A + {1'b0,M}; // leave positive remainder

        if (Qisneg ^ Misneg == 1) begin
                Q = Q*(-1);
                A = A*(-1);
        end


end

assign result[63:0] = {A[31:0], Q[31:0]};
```

```
        assign diverror = divbyzero;
endmodule
```

Encoder32to5

```
module Encoder32to5 (input i0, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19,
i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, output [4:0] out);


  reg [4:0] Enc_out = 5'b00000;


  always @(*) begin
                if (i0 == 1)
                        Enc_out = 0;
                else if (i1 == 1)
                        Enc_out = 1;
                else if (i2 == 1)
                        Enc_out = 2;
                else if (i3 == 1)
                        Enc_out = 3;
                else if (i4 == 1)
                        Enc_out = 4;
                else if (i5 == 1)
                        Enc_out = 5;
                else if (i6 == 1)
                        Enc_out = 6;
                else if (i7 == 1)
                        Enc_out = 7;
                else if (i8 == 1)
                        Enc_out = 8;
                else if (i9 == 1)
                        Enc_out = 9;
```

```
else if (i10 == 1)
        Enc_out = 10;
else if (i11 == 1)
        Enc_out = 11;
else if (i12 == 1)
        Enc_out = 12;
else if (i13 == 1)
        Enc_out = 13;
else if (i14 == 1)
        Enc_out = 14;
else if (i15 == 1)
        Enc_out = 15;
else if (i16 == 1)
        Enc_out = 16;
else if (i17 == 1)
        Enc_out = 17;
else if (i18 == 1)
        Enc_out = 18;
else if (i19 == 1)
        Enc_out = 19;
else if (i20 == 1)
        Enc_out = 20;
else if (i21 == 1)
        Enc_out = 21;
else if (i22 == 1)
        Enc_out = 22;
else if (i23 == 1)
        Enc_out = 23;
else if (i24 == 1)
```

```
                Enc_out = 24;
            else if (i25 == 1)
                Enc_out = 25;
            else if (i26 == 1)
                Enc_out = 26;
            else if (i27 == 1)
                Enc_out = 27;
            else if (i28 == 1)
                Enc_out = 28;
            else if (i29 == 1)
                Enc_out = 29;
            else if (i30 == 1)
                Enc_out = 30;
            else if (i31 == 1)
                Enc_out = 31;
            else
                Enc_out = 0;

    end


  assign out = Enc_out;


endmodule
```

Inport

```
module Inport(input clear, clock, strobe, input [31:0] D, output [31:0] Q);

  reg [31:0] Q_reg;
```

```
  always @(posedge clock)
    begin
          if (clear == 1)
            Q_reg = 0;
          else if (strobe == 1)
            Q_reg = D;
    end

  assign Q = Q_reg;

endmodule // Inport
```

MDR

```
module MDR (input clr, clk, mdrin, input [31:0] D, output [31:0] Q);
  reg [31:0] Q_r;

  always @(posedge clk) begin
    if (clr == 1)
          Q_r = 0;
    else if (mdrin == 1)
          Q_r = D;
  end

  assign Q = Q_r;

endmodule // MDR
```

MDR_unit

```
module MDR_unit(input read, clk, MDRin, clr, input [31:0] BusMuxIn, Mdatain, output [31:0] Q);
  wire [31:0] D;
```

```
        // MDRin is a 1 or 0 control signal telling the MDR that its being given an update to accept


  MuxMD md_mux(read, BusMuxIn, Mdatain, D);

  MDR mdr(clr, clk, MDRin, D, Q);


endmodule // MDR_unit
```

Multiplier

```
module Multiplier(input [31:0] M, Q, input MUL,  output [63:0] P);

  // Used to hold current bits of Q being encoded

  // Honestly not rlly sure if we need both but it won't let me use a wire in the always block so I think I need the reg

  reg [2:0] bits_r;

  wire [2:0] bits_w;


  // Used to hold M and Q so they can be used in operations

  reg [31:0] M1;

  reg [31:0] Q1;


  // booth_enc output, holds the current version of M to be added to partial

  // Wire is directly from encoder, reg is used for operations in the always

  wire [63:0] current_M_w;

  reg [63:0]  current_M_r = 64'b0;


  // Current partial product

  reg [63:0]  current_prod = 64'b0;

  // Final product that is connected to output of multiplier, set after all iterations

  reg [63:0]  final_prod;


  // An index for use in the for loop
```

```
  reg [4:0]   i;


  Booth_encoder booth_enc(bits_w, M1, current_M_w);


  always @(*) begin
    if (MUL == 1'b1) begin
          M1 = M;
          Q1 = Q;


          // This block couldn't go in the for loop because on the first iteration you only shift Q right once and you have to feed the encoder a zero for the first bit
          bits_r = {Q1[1], Q1[0], 1'b0};
          // Had to add this delay, otherwise it goes to fast to capture any values
          #10
            Q1 = Q1 >> 1;
          current_M_r = current_M_w;


          current_prod = current_prod + current_M_r;


          // Now run the loop for the remaining 15 times to get the rest of the product
          for (i = 1; i < 16; i = i + 1) begin
            bits_r = {Q1[2], Q1[1], Q1[0]};
            #10
              Q1 = Q1 >> 2;
            current_M_r = current_M_w;
            // At every iteration, the partial product gets shifted 2*i, but can't use multiplication in the multiplier, so it's just a nice left shift
            current_prod = current_prod + (current_M_r << (i << 1));
          end
```

```
            final_prod = current_prod;
        end
    end

    assign P = final_prod;
    assign bits_w = bits_r;

endmodule
```

Mux32bit32to1

```
module Mux32bit32to1(input [4:0] select, input [31:0] i0, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13,
i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, output [31:0] out);

        Mux8bit32to1 mux1 (select, i0[31:24], i1[31:24], i2[31:24], i3[31:24], i4[31:24], i5[31:24],
i6[31:24], i7[31:24], i8[31:24], i9[31:24], i10[31:24], i11[31:24], i12[31:24], i13[31:24], i14[31:24],
i15[31:24], i16[31:24], i17[31:24], i18[31:24], i19[31:24], i20[31:24], i21[31:24], i22[31:24], i23[31:24],
i24[31:24], i25[31:24], i26[31:24], i27[31:24], i28[31:24], i29[31:24], i30[31:24], i31[31:24], out[31:24]);

        Mux8bit32to1 mux2 (select, i0[23:16], i1[23:16], i2[23:16], i3[23:16], i4[23:16], i5[23:16],
i6[23:16], i7[23:16], i8[23:16], i9[23:16], i10[23:16], i11[23:16], i12[23:16], i13[23:16], i14[23:16],
i15[23:16], i16[23:16], i17[23:16], i18[23:16], i19[23:16], i20[23:16], i21[23:16], i22[23:16], i23[23:16],
i24[23:16], i25[23:16], i26[23:16], i27[23:16], i28[23:16], i29[23:16], i30[23:16], i31[23:16], out[23:16]);

        Mux8bit32to1 mux3 (select, i0[15:8], i1[15:8], i2[15:8], i3[15:8], i4[15:8], i5[15:8], i6[15:8],
i7[15:8], i8[15:8], i9[15:8], i10[15:8], i11[15:8], i12[15:8], i13[15:8], i14[15:8], i15[15:8], i16[15:8],
i17[15:8], i18[15:8], i19[15:8], i20[15:8], i21[15:8], i22[15:8], i23[15:8], i24[15:8], i25[15:8], i26[15:8],
i27[15:8], i28[15:8], i29[15:8], i30[15:8], i31[15:8], out[15:8]);

        Mux8bit32to1 mux4 (select, i0[7:0], i1[7:0], i2[7:0], i3[7:0], i4[7:0], i5[7:0], i6[7:0], i7[7:0],
i8[7:0], i9[7:0], i10[7:0], i11[7:0], i12[7:0], i13[7:0], i14[7:0], i15[7:0], i16[7:0], i17[7:0], i18[7:0], i19[7:0],
i20[7:0], i21[7:0], i22[7:0], i23[7:0], i24[7:0], i25[7:0], i26[7:0], i27[7:0], i28[7:0], i29[7:0], i30[7:0],
i31[7:0], out[7:0]);


Endmodule

module Mux8bit32to1 (input [4:0] select, input [7:0] i0, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14,
i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, output [7:0] out);


        reg [7:0] to_out;
```

```
always @(*) begin
        if (select == 5'b00000)
                to_out = i0;
        else if (select == 5'b00001)
                to_out = i1;
        else if (select == 5'b00010)
                to_out = i2;
        else if (select == 5'b00011)
                to_out = i3;
        else if (select == 5'b00100)
                to_out = i4;
        else if (select == 5'b00101)
                to_out = i5;
        else if (select == 5'b00110)
                to_out = i6;
        else if (select == 5'b00111)
                to_out = i7;
        else if (select == 5'b01000)
                to_out = i8;
        else if (select == 5'b01001)
                to_out = i9;
        else if (select == 5'b01010)
                to_out = i10;
        else if (select == 5'b01011)
                to_out = i11;
        else if (select == 5'b01100)
                to_out = i12;
        else if (select == 5'b01101)
```

```
        to_out = i13;
else if (select == 5'b01110)
        to_out = i14;
else if (select == 5'b01111)
        to_out = i15;
else if (select == 5'b10000)
        to_out = i16;
else if (select == 5'b10001)
        to_out = i17;
else if (select == 5'b10010)
        to_out = i18;
else if (select == 5'b10011)
        to_out = i19;
else if (select == 5'b10100)
        to_out = i20;
else if (select == 5'b10101)
        to_out = i21;
else if (select == 5'b10110)
        to_out = i22;
else if (select == 5'b10111)
        to_out = i23;
else if (select == 5'b11000)
        to_out = i24;
else if (select == 5'b11001)
        to_out = i25;
else if (select == 5'b11010)
        to_out = i26;
else if (select == 5'b11011)
        to_out = i27;
```

```
            else if (select == 5'b11100)
                to_out = i28;
            else if (select == 5'b11101)
                to_out = i29;
            else if (select == 5'b11110)
                to_out = i30;
            else if (select == 5'b11111)
                to_out = i31;
        end

        assign out = to_out;

endmodule
```

MuxMD

```
module MuxMD (input read, input [31:0] BusMuxOut, Mdatain, output [31:0] D);

  reg [31:0] D_r;

  always @(*) begin
    if (read == 1)
        D_r = Mdatain;
    else
        D_r = BusMuxOut;
  end

  assign D = D_r;

endmodule // MuxMD
```

Or12Bit

```
module Or12Bit(input [3:0] w, x, y, output [3:0] z);

  or(z[0], w[0], x[0], y[0]);

  or(z[1], w[1], x[1], y[1]);

  or(z[2], w[2], x[2], y[2]);

  or(z[3], w[3], x[3], y[3]);


endmodule // Or12Bit
```

Outport

```
module Outport(input clr, clock, enable, input [31:0] D, output [31:0] Q);

  reg [31:0] Q_reg;


  always @(posedge clock)

    begin

          if (clr == 1)

            Q_reg = 0;

          else if (enable == 1)

            Q_reg = D;

    end


  assign Q = Q_reg;


endmodule // Outport
```

R0

```
module R0 (clear, clk, R0in, BAout, D, Q);


  input clear;

  input clk;

  input R0in;

  input BAout;
```

```
  input [31:0] D;
  output [31:0] Q;
  reg [31:0]        register;

  genvar            gi;

  always @(posedge clk)
   begin
          if (clear == 1)
           begin
             register = 0;
           end else if (R0in == 1)
            begin
              register = D;
            end
   end

  // Produce output
  generate
    for (gi = 0; gi < 32; gi = gi + 1) begin : gen_out
          and(Q[gi], ~BAout, register[gi]);
    end
  endgenerate

endmodule
```

RAM

```
module RAM (input read, write, inout [31:0] data, input [8:0] addr);

          reg [31:0] mem [0:(1<<8)-1];
```

```
assign data = (read == 1) ? mem[addr] : 32'bz;

// Assign values in ram here, in further stages, would use .mif file
initial begin

        //mem[0] <= 'h00800055; // for ld instruction
        // mem[85] <= 'h00000002;

        //mem[0] <= 'h08800055; // for ldi instruction Case 3
        //mem[0] <= 'h08080023; // ldi Case 4

        //mem[0] <= 'h91000023; // brzr
        //mem[0] <= 'h91080023; // brnz
        //mem[0] <= 'h91100023; // brpl
        mem[0] <= 'h91180023; // brmi

  // For ld R1, $85
         //mem[0] <= 'h00800055;
         //mem[85] <= 'h00000002;
        // For st $90, R1
        //mem[0] <= 'h1080005A;
        // For st $90(R1), R1
        //mem[0] <= 'h1088005A;
        // For addi R2, R1, -5
        //mem[0] <= 32'h590FFFFB;
        // For andi/ori R2, R1, $26
        //mem[0] <= 32'h5908001A;
        // For jr R1
```

```
            //mem[0] <= 32'h98800000;
            // For jal R1
            // mem[0] <= 32'hA0800000;
            // For mfhi R1
             //mem[0] <= 32'hB8800000;
            // For mflo R1
            // mem[0] <= 32'hC0800000;
            // For out R1
            // mem[0] <= 32'hB0800000;
            // For in R1
            // mem[0] <= 32'hA8800000;
        end

        always @(*) begin
            if (read && write) begin $display("Error: both read and write enabled in RAM."); end
            else if (write == 1) mem[addr] <= data;
        end

endmoduleemacs c
```

Register32

```
module Register32 (clear, clk, Rin, D, Q);

  input clear;
  input clk;
  input Rin;
  input [31:0] D;
  output [31:0] Q;
  reg [31:0]        register;
```

```
  always @(posedge clk)
   begin
          if (clear == 1)
            begin
              register = 0;
            end else if (Rin == 1)
              begin
                register = D;
              end
   end

  assign Q = register;

endmodule
```

Register64

```
module Register64 (clear, clk, Rin, D, Q);

  input clear;
  input clk;
  input Rin;
  input [63:0] D;
  output [63:0] Q;
  reg [63:0]        register;

  always @(posedge clk)
   begin
          if (clear == 1)
            register <= 0;
          else if (Rin == 1)
```

```
            register <= D;
    end

  assign Q = register;

endmodule
```

Sel_Enc

```
module Sel_Enc(input Gra, Grb, Grc, Rin, Rout, BAout, input [31:0] IR, output [15:0] Rins, Routs, output [31:0] C_sign_extended);
  wire [3:0] z1, z2, z3, z;
  reg [15:0] dec_out;
  wire      out;

  genvar gi;

  // Sign-extend C
  assign C_sign_extended = (IR[18] == 0)? {13'b0000000000000, IR[18:0]} : {13'b1111111111111, IR[18:0]};


  // AND and OR gates to choose Ra, Rb, or Rc
  And4Bit and1(IR[26:23], Gra, z1);
  And4Bit and2(IR[22:19], Grb, z2);
  And4Bit and3(IR[18:15], Grc, z3);
  Or12Bit or12(z1, z2, z3, z);

  // 4-to-16 decoder
  always @(z)
    begin
          case (z)
```

```
            4'b0000 : dec_out = 16'h0001;
            4'b0001 : dec_out = 16'h0002;
            4'b0010 : dec_out = 16'h0004;
            4'b0011 : dec_out = 16'h0008;
            4'b0100 : dec_out = 16'h0010;
            4'b0101 : dec_out = 16'h0020;
            4'b0110 : dec_out = 16'h0040;
            4'b0111 : dec_out = 16'h0080;
            4'b1000 : dec_out = 16'h0100;
            4'b1001 : dec_out = 16'h0200;
            4'b1010 : dec_out = 16'h0400;
            4'b1011 : dec_out = 16'h0800;
            4'b1100 : dec_out = 16'h1000;
            4'b1101 : dec_out = 16'h2000;
            4'b1110 : dec_out = 16'h4000;
            4'b1111 : dec_out = 16'h8000;
          endcase // case (z)
  end // always @ (z)

// Generate bit to release the Routs
or(out, Rout, BAout);

// Generate Rins
generate
   for (gi = 0; gi < 16; gi = gi + 1) begin : gen_rins
           and(Rins[gi], Rin, dec_out[gi]);
   end
endgenerate
```

```
  // Generate Routs
  generate
    for (gi = 0; gi < 16; gi = gi + 1) begin : gen_routs
          and(Routs[gi], out, dec_out[gi]);
    end
  endgenerate



endmodule // Sel_Enc
```

Testbenches

Arith_instr_tb

```
module arith_instr_tb();
  reg PCout, Zlowout, MDRout, Cout;
  reg MARin, Zin, PCin, MDRin, IRin, Yin, Gra, Grb, Grc, Rin, Rout, BAout;
  reg IncPC, Read, Write, ADD, AND, OR, SUB, SHR, SHL, ROL, ROR, NEG, NOT;
  reg Clock, clear;
        wire [31:0] Mdatain, ram_data, MDR, MAR, R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12,
        R13, R14, R15, Hi, Lo, PC, bus_mux_out, IR, C_sign_ext;
        wire [63:0] Z, ALUout;
  wire [15:0] Rins, Routs;

  parameter Default = 4'b0000, Reg_load_1a = 4'b0001, Reg_load_1b = 4'b0010, T0 = 4'b0011, T1 =
4'b0100,
        T2 = 4'b0101, T3 = 4'b0110, T4 = 4'b0111, T5 = 4'b1000;
  reg [3:0]  Present_state = Default;

  Datapath DUT (.write(Write), .Mdatain(Mdatain), .clear(clear), .Cout(Cout), .BAout(BAout),
.PCout(PCout), .Zlowout(Zlowout),
        .MDRout(MDRout), .Gra(Gra), .Grb(Grb), .Grc(Grc), .Rin(Rin), .Rout(Rout), .MARin(MARin),
        .Zin(Zin), .PCin(PCin), .MDRin(MDRin), .IRin(IRin), .Yin(Yin), .IncPC(IncPC), .read(Read),
        .ADD(ADD), .clk(Clock), .MAR(MAR), .R0(R0), .R1(R1), .R2(R2), .R3(R3), .R4(R4),
        .R5(R5), .R6(R6), .R7(R7), .R8(R8), .R9(R9), .R10(R10), .R11(R11), .R12(R12), .R13(R13), .R14(R14),
        .R15(R15), .Hi(Hi), .Lo(Lo), .IR(IR),.Z(Z), .AND(AND), .OR(OR), .SUB(SUB),.SHR(SHR), .SHL(SHL),
        .ROR(ROR), .ROL(ROL), .NEG(NEG), .NOT(NOT), .bus_mux_out(bus_mux_out), .ALUout(ALUout),
        .ram_data(ram_data), .MDR(MDR), .PC(PC), .C_sign_ext(C_sign_ext), .Rins(Rins), .Routs(Routs));

        initial begin
    Clock = 0;
    $dumpfile("p2_datapath_tb.vcd");
    $dumpvars(0, DUT);
    forever #10 Clock = ~Clock;
```

```
end

initial begin
  #500 $finish;
end

always @(posedge Clock) begin
  case (Present_state)
                Default : Present_state = Reg_load_1a;
                Reg_load_1a : Present_state = Reg_load_1b;
                Reg_load_1b : Present_state = T0;
                T0 : Present_state = T1;
                T1 : #10 Present_state = T2;
                T2 : #10 Present_state = T3;
                T3 : #10 Present_state = T4;
                T4 : #20 Present_state = T5;
  endcase // case (Present_state)
end // always @ (posedge Clock)

always @(Present_state) begin
  case(Present_state)
        // Assert clear to load zero's into all registers
        Default: begin
          clear <= 1;
          PCout <= 0;
          Zlowout <= 0;
          MDRout <= 0;
          Cout <= 0;
          Rin <= 0;
          Rout <= 0;
          Gra <= 0;
          Grb <= 0;
          Grc <= 0;
          MARin <= 0;
          Zin <= 0;
          PCin <= 0;
          MDRin <= 0;
          IRin <= 0;
          Yin <= 0;
          IncPC <= 0;
          Write <= 0;
                Read <= 0;
          AND <= 0;
                OR <= 0;
                SUB <= 0;
                ADD <= 0;
                SHL <= 0;
                SHR <= 0;
```

```
            ROL <= 0;
            ROR <= 0;
            NEG <= 0;
            NOT <= 0;
            BAout <= 0;
end // case: Default
// Load value of $85 into R1 from RAM
Reg_load_1a: begin
            clear <= 0;
            force ram_data = 'h00000055;
  #10 Read <= 1; MDRin <= 1;
  #10 Read <= 0; MDRin <= 0;
            release ram_data;
end
Reg_load_1b: begin
            force Rins = 'h0002;
  #10 MDRout <= 1;
  #15 MDRout <= 0;
            release Rins;
end
// Get PC in MAR to go to mem and get st instr (@ addr 0), get PC+4 in Z
T0: begin
            PCout <= 1;
  MARin <= 1;
  IncPC <= 1;
  Zin <= 1;
end
// Put PC+4 in PC, load arith instr from memory into MDR
T1: begin
            PCout <= 0; MARin <= 0;
  IncPC <= 0;
            #20 Zlowout <= 1;
  PCin <= 1; Read <= 1;
  MDRin <= 1;
            Zin <= 0;
end
// Load instr into IR
T2: begin
            #10 MDRin <= 0;
  Zlowout <= 0;
            PCin <= 0;
  Read <= 0;
  #10 MDRout <= 1;
  IRin <= 1;
end
// Get contents of Rb on bus and in Y
T3: begin
            #10 MDRout <= 0;
```

```
            IRin <= 0;
            #10 Grb <= 1;
            Rout <= 1;
            Yin <= 1;
          end
          // Put C_sign_ext + Rb in Z
          T4: begin
            #10
                  Rout <= 0;
            Grb <= 0;
            Yin <= 0;
                  Cout <= 1;
            OR <= 1;
            Zin <= 1;
          end
          // Put C_sign_ext + Rb in R1
          T5: begin
            #20 Cout <= 0;
            OR <= 0;
                  Zin <= 0;
                  Zlowout <= 1;
            Gra <= 1;
                  Rin <= 1;
          end
  endcase // case (Present_state)
 end // always @ (Present_state)
endmodule
```

Inout_instr_tb

```
module inout_instr_tb();

  reg PCout, Zlowout, MDRout, Cout, HIout, LOout, InPortOut;

  reg MARin, Zin, PCin, MDRin, IRin, Yin, Gra, Grb, Grc, Rin, Rout, BAout, HIin, LOin, OutPort, strobe;

  reg IncPC, Read, Write, ADD, AND, OR, SUB, SHR, SHL, ROL, ROR, NEG, NOT;

  reg Clock, clear;

        reg [31:0] In_Data;

        wire [31:0] Mdatain, ram_data, MDR, MAR, R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12,

        R13, R14, R15, Hi, Lo, PC, bus_mux_out, IR, C_sign_ext, OutPortOut, InPortOutput;

        wire [63:0] Z, ALUout;

  wire [15:0] Rins, Routs;
```

```
  parameter Default = 4'b0000, Reg_load_1a = 4'b0001, Reg_load_1b = 4'b0010, T0 = 4'b0011, T1 = 4'b0100,
        T2 = 4'b0101, T3 = 4'b0110;
  reg [3:0]  Present_state = Default;

  Datapath DUT (.InPortOutput(InPortOutput), .InPortIn(In_Data), .strobe(strobe),
        .Inportout(InPortOut), .OutPortOut(OutPortOut), .OutPort(OutPort),.HIin(HIin), .LOin(LOin), .HIout(HIout), .LOout(LOout),.write(Write), .Mdatain(Mdatain), .clear(clear), .Cout(Cout), .BAout(BAout), .PCout(PCout), .Zlowout(Zlowout),
        .MDRout(MDRout), .Gra(Gra), .Grb(Grb), .Grc(Grc), .Rin(Rin), .Rout(Rout), .MARin(MARin),
        .Zin(Zin), .PCin(PCin), .MDRin(MDRin), .IRin(IRin), .Yin(Yin), .IncPC(IncPC), .read(Read),
        .ADD(ADD), .clk(Clock), .MAR(MAR), .R0(R0), .R1(R1), .R2(R2), .R3(R3), .R4(R4),
        .R5(R5), .R6(R6), .R7(R7), .R8(R8), .R9(R9), .R10(R10), .R11(R11), .R12(R12), .R13(R13), .R14(R14),
        .R15(R15), .Hi(Hi), .Lo(Lo), .IR(IR),.Z(Z), .AND(AND), .OR(OR), .SUB(SUB),.SHR(SHR), .SHL(SHL),
        .ROR(ROR), .ROL(ROL), .NEG(NEG), .NOT(NOT), .bus_mux_out(bus_mux_out), .ALUout(ALUout),
        .ram_data(ram_data), .MDR(MDR), .PC(PC), .C_sign_ext(C_sign_ext), .Rins(Rins), .Routs(Routs));

        initial begin
    Clock = 0;
    $dumpfile("p2_datapath_tb.vcd");
    $dumpvars(0, DUT);
    forever #10 Clock = ~Clock;
  end

  initial begin
    #500 $finish;
  end

  always @(posedge Clock) begin
    case (Present_state)
```

```
                Default : Present_state = Reg_load_1a;
                Reg_load_1a : Present_state = Reg_load_1b;
                Reg_load_1b : #5 Present_state = T0;
                T0 : Present_state = T1;
                T1 : #15 Present_state = T2;
                T2 : #30 Present_state = T3;
  endcase // case (Present_state)
end // always @ (posedge Clock)

always @(Present_state) begin
  case(Present_state)
        // Assert clear to load zero's into all registers
        Default: begin
          clear <= 1;
          PCout <= 0;
          Zlowout <= 0;
          MDRout <= 0;
          Cout <= 0;
          Rin <= 0;
          Rout <= 0;
          Gra <= 0;
          Grb <= 0;
          Grc <= 0;
          MARin <= 0;
                OutPort <= 0;
          Zin <= 0;
          PCin <= 0;
          MDRin <= 0;
          IRin <= 0;
```

```
  Yin <= 0;
        HIin <= 0;
        LOin <= 0;
        HIout <= 0;
        LOout <= 0;
  IncPC <= 0;
  Write <= 0;
        Read <= 0;
  AND <= 0;
        OR <= 0;
        SUB <= 0;
        ADD <= 0;
        SHL <= 0;
        SHR <= 0;
        ROL <= 0;
        ROR <= 0;
        NEG <= 0;
        NOT <= 0;
        BAout <= 0;
        strobe <= 0;
        InPortOut <= 0;
end // case: Default
// For out R1 instr:
// Load value of $85 into R1 from RAM
/* Reg_load_1a: begin
        clear <= 0;
        force ram_data = 'h00000055;
  #10 Read <= 1; MDRin <= 1;
  #10 Read <= 0; MDRin <= 0;
```

```
            release ram_data;
end
Reg_load_1b: begin
  force Rins = 'h0002;
  #10 MDRout <= 1;
  #15 MDRout <= 0;
            release Rins;
end */
// For in R1 instr:
// Send $85 to InPort
Reg_load_1a: begin
            clear <= 0;
            In_Data <= 32'h00000055;
            #10 strobe <= 1;
            #15 strobe <= 0;
end
Reg_load_1b: begin
 // Do nothing; need so I can use 1 tb for out and in
end
// Get PC in MAR to go to mem and get instr (@ addr 0), get PC+4 in Z
T0: begin
            PCout <= 1;
  MARin <= 1;
  IncPC <= 1;
  Zin <= 1;
end
// Put PC+4 in PC, load mf instr from memory into MDR
T1: begin
            #20 PCout <= 0; MARin <= 0;
```

```
  IncPC <= 0; Zlowout <= 1;
  PCin <= 1; Read <= 1;
  MDRin <= 1;
        Zin <= 0;
end
// Load instr into IR
T2: begin
        #15
        MDRin <= 0;
  Zlowout <= 0;
        PCin <= 0;
  Read <= 0;
  #10 MDRout <= 1;
  IRin <= 1;
end
// For out R1 instr:
// Send R1 contents to Outport
/* T3: begin
        #10 MDRout <= 0;
  IRin <= 0;
  #10 Gra <= 1;
  Rout <= 1;
  OutPort <= 1;
end */
// For in R1 instr:
// Send InPort data to R1
T3: begin
        #10 MDRout <= 0;
  IRin <= 0;
```

```
            #10 Gra <= 1;
            Rin <= 1;
            InPortOut <= 1;
          end
  endcase // case (Present_state)
 end // always @ (Present_state)
endmodule
```

jal_instr_tb

```
module jal_instr_tb();
  reg PCout, Zlowout, MDRout, Cout;
  reg MARin, Zin, PCin, MDRin, IRin, Yin, Gra, Grb, Grc, Rin, Rout, BAout;
  reg IncPC, Read, Write, ADD, AND, OR, SUB, SHR, SHL, ROL, ROR, NEG, NOT;
  reg Clock, clear;
        wire [31:0] Mdatain, ram_data, MDR, MAR, R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12,
        R13, R14, R15, Hi, Lo, PC, bus_mux_out, IR, C_sign_ext;
        wire [63:0] Z, ALUout;
  wire [15:0] Rins, Routs;

  parameter Default = 4'b0000, Reg_load_1a = 4'b0001, Reg_load_1b = 4'b0010, T0 = 4'b0011, T1 =
4'b0100,
        T2 = 4'b0101, T3 = 4'b0110;
  reg [3:0]  Present_state = Default;

  Datapath DUT (.write(Write), .Mdatain(Mdatain), .clear(clear), .Cout(Cout), .BAout(BAout),
.PCout(PCout), .Zlowout(Zlowout),
        .MDRout(MDRout), .Gra(Gra), .Grb(Grb), .Grc(Grc), .Rin(Rin), .Rout(Rout), .MARin(MARin),
        .Zin(Zin), .PCin(PCin), .MDRin(MDRin), .IRin(IRin), .Yin(Yin), .IncPC(IncPC), .read(Read),
        .ADD(ADD), .clk(Clock), .MAR(MAR), .R0(R0), .R1(R1), .R2(R2), .R3(R3), .R4(R4),
        .R5(R5), .R6(R6), .R7(R7), .R8(R8), .R9(R9), .R10(R10), .R11(R11), .R12(R12), .R13(R13), .R14(R14),
        .R15(R15), .Hi(Hi), .Lo(Lo), .IR(IR),.Z(Z), .AND(AND), .OR(OR), .SUB(SUB),.SHR(SHR), .SHL(SHL),
```

```
        .ROR(ROR), .ROL(ROL), .NEG(NEG), .NOT(NOT), .bus_mux_out(bus_mux_out), .ALUout(ALUout),
        .ram_data(ram_data), .MDR(MDR), .PC(PC), .C_sign_ext(C_sign_ext), .Rins(Rins), .Routs(Routs));

        initial begin
  Clock = 0;
  $dumpfile("p2_datapath_tb.vcd");
  $dumpvars(0, DUT);
  forever #10 Clock = ~Clock;
end

initial begin
  #500 $finish;
end

always @(posedge Clock) begin
  case (Present_state)
                Default : Present_state = Reg_load_1a;
                Reg_load_1a : Present_state = Reg_load_1b;
                Reg_load_1b : #5 Present_state = T0;
                T0 : Present_state = T1;
                T1 : #15 Present_state = T2;
                T2 : #30 Present_state = T3;
  endcase // case (Present_state)
end // always @ (posedge Clock)

always @(Present_state) begin
  case(Present_state)
        // Assert clear to load zero's into all registers
        Default: begin
```

```
clear <= 1;
PCout <= 0;
Zlowout <= 0;
MDRout <= 0;
Cout <= 0;
Rin <= 0;
Rout <= 0;
Gra <= 0;
Grb <= 0;
Grc <= 0;
MARin <= 0;
Zin <= 0;
PCin <= 0;
MDRin <= 0;
IRin <= 0;
Yin <= 0;
IncPC <= 0;
Write <= 0;
        Read <= 0;
AND <= 0;
        OR <= 0;
        SUB <= 0;
        ADD <= 0;
        SHL <= 0;
        SHR <= 0;
        ROL <= 0;
        ROR <= 0;
        NEG <= 0;
        NOT <= 0;
```

```
        BAout <= 0;
end // case: Default
// Load value of $85 into R1 from RAM
Reg_load_1a: begin
        clear <= 0;
        force ram_data = 'h00000055;
  #10 Read <= 1; MDRin <= 1;
  #10 Read <= 0; MDRin <= 0;
        release ram_data;
end
Reg_load_1b: begin
        force Rins = 'h0002;
  #10 MDRout <= 1;
  #15 MDRout <= 0;
        release Rins;
end
// Get PC in MAR to go to mem and get instr (@ addr 0), get PC+4 in Z
T0: begin
        PCout <= 1;
  MARin <= 1;
  IncPC <= 1;
  Zin <= 1;
end
// Put PC+4 in R15, load jal instr from memory into MDR
T1: begin
        force Rins = 'h8000;
        #20 PCout <= 0; MARin <= 0;
  IncPC <= 0; Zlowout <= 1;
  Rin <= 1; Read <= 1;
```

```
            MDRin <= 1;
                    Zin <= 0;
            end
            // Load instr into IR
            T2: begin
                    #15 release Rins;
                    MDRin <= 0;
            Zlowout <= 0;
                    Rin <= 0;
            Read <= 0;
            #10 MDRout <= 1;
            IRin <= 1;
            end
            // Store contents of Ra in PC
            T3: begin
                    #10 MDRout <= 0;
            IRin <= 0;
            #10 Gra <= 1;
            Rout <= 1;
            PCin <= 1;
            end
  endcase // case (Present_state)
 end // always @ (Present_state)
endmodule
```

jr_instr_tb

```
module jr_instr_tb();
   reg PCout, Zlowout, MDRout, Cout;
   reg MARin, Zin, PCin, MDRin, IRin, Yin, Gra, Grb, Grc, Rin, Rout, BAout;
   reg IncPC, Read, Write, ADD, AND, OR, SUB, SHR, SHL, ROL, ROR, NEG, NOT;
```

```
  reg Clock, clear;
        wire [31:0] Mdatain, ram_data, MDR, MAR, R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12,
        R13, R14, R15, Hi, Lo, PC, bus_mux_out, IR, C_sign_ext;
        wire [63:0] Z, ALUout;
  wire [15:0] Rins, Routs;

  parameter Default = 4'b0000, Reg_load_1a = 4'b0001, Reg_load_1b = 4'b0010, T0 = 4'b0011, T1 =
4'b0100,
        T2 = 4'b0101, T3 = 4'b0110;
  reg [3:0]  Present_state = Default;

  Datapath DUT (.write(Write), .Mdatain(Mdatain), .clear(clear), .Cout(Cout), .BAout(BAout),
.PCout(PCout), .Zlowout(Zlowout),
        .MDRout(MDRout), .Gra(Gra), .Grb(Grb), .Grc(Grc), .Rin(Rin), .Rout(Rout), .MARin(MARin),
        .Zin(Zin), .PCin(PCin), .MDRin(MDRin), .IRin(IRin), .Yin(Yin), .IncPC(IncPC), .read(Read),
        .ADD(ADD), .clk(Clock), .MAR(MAR), .R0(R0), .R1(R1), .R2(R2), .R3(R3), .R4(R4),
        .R5(R5), .R6(R6), .R7(R7), .R8(R8), .R9(R9), .R10(R10), .R11(R11), .R12(R12), .R13(R13), .R14(R14),
        .R15(R15), .Hi(Hi), .Lo(Lo), .IR(IR),.Z(Z), .AND(AND), .OR(OR), .SUB(SUB),.SHR(SHR), .SHL(SHL),
        .ROR(ROR), .ROL(ROL), .NEG(NEG), .NOT(NOT), .bus_mux_out(bus_mux_out), .ALUout(ALUout),
        .ram_data(ram_data), .MDR(MDR), .PC(PC), .C_sign_ext(C_sign_ext), .Rins(Rins), .Routs(Routs));

        initial begin
    Clock = 0;
    $dumpfile("p2_datapath_tb.vcd");
    $dumpvars(0, DUT);
    forever #10 Clock = ~Clock;
  end

  initial begin
```

```
  #500 $finish;
end

always @(posedge Clock) begin
  case (Present_state)
                Default : Present_state = Reg_load_1a;
                Reg_load_1a : Present_state = Reg_load_1b;
                Reg_load_1b : Present_state = T0;
                T0 : Present_state = T1;
                T1 : #10 Present_state = T2;
                T2 : #10 Present_state = T3;
  endcase // case (Present_state)
end // always @ (posedge Clock)

always @(Present_state) begin
  case(Present_state)
        // Assert clear to load zero's into all registers
        Default: begin
          clear <= 1;
          PCout <= 0;
          Zlowout <= 0;
          MDRout <= 0;
          Cout <= 0;
          Rin <= 0;
          Rout <= 0;
          Gra <= 0;
          Grb <= 0;
          Grc <= 0;
          MARin <= 0;
```

```
  Zin <= 0;
  PCin <= 0;
  MDRin <= 0;
  IRin <= 0;
  Yin <= 0;
  IncPC <= 0;
  Write <= 0;
        Read <= 0;
  AND <= 0;
        OR <= 0;
        SUB <= 0;
        ADD <= 0;
        SHL <= 0;
        SHR <= 0;
        ROL <= 0;
        ROR <= 0;
        NEG <= 0;
        NOT <= 0;
        BAout <= 0;
end // case: Default
// Load value of $85 into R1 from RAM
Reg_load_1a: begin
        clear <= 0;
        force ram_data = 'h00000055;
  #10 Read <= 1; MDRin <= 1;
  #10 Read <= 0; MDRin <= 0;
        release ram_data;
end
Reg_load_1b: begin
```

```
            force Rins = 'h0002;
  #10 MDRout <= 1;
  #15 MDRout <= 0;
            release Rins;
end
// Get PC in MAR to go to mem and get instr (@ addr 0), get PC+4 in Z
T0: begin
            PCout <= 1;
  MARin <= 1;
  IncPC <= 1;
  Zin <= 1;
end
// Put PC+4 in PC, load jr instr from memory into MDR
T1: begin
            PCout <= 0; MARin <= 0;
  IncPC <= 0;
            #20 Zlowout <= 1;
  PCin <= 1; Read <= 1;
  MDRin <= 1;
            Zin <= 0;
end
// Load instr into IR
T2: begin
            #10 MDRin <= 0;
  Zlowout <= 0;
            PCin <= 0;
  Read <= 0;
  #10 MDRout <= 1;
  IRin <= 1;
```

```
            end
            // Store contents of Ra in PC
            T3: begin
                    #10 MDRout <= 0;
              IRin <= 0;
              #10 Gra <= 1;
              Rout <= 1;
              PCin <= 1;
            end
  endcase // case (Present_state)
 end // always @ (Present_state)
endmodule
```

ld_instr_tb

```
module ld_instr_tb();
   reg PCout, Zlowout, MDRout, Cout;
   reg MARin, Zin, PCin, MDRin, IRin, Yin, Gra, Grb, Grc, Rin, Rout, BAout;
   reg IncPC, Read, ADD, AND, OR, SUB, SHR, SHL, ROL, ROR, NEG, NOT;
   reg Clock, clear;
            wire [31:0] Mdatain, ram_data, MDR, MAR, R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12,
            R13, R14, R15, Hi, Lo, PC, bus_mux_out, IR, C_sign_ext;
            wire [63:0] Z, ALUout;
   wire [15:0] Rins, Routs;

   parameter Default = 4'b0000, T0 = 4'b0001, T1 = 4'b0010, T2 = 4'b0011,
            T3 = 4'b0100, T4 = 4'b0101, T5 = 4'b0110, T6 = 4'b0111, T7 = 4'b1000;
   reg [3:0]  Present_state = Default;

   Datapath DUT (.Mdatain(Mdatain), .clear(clear), .Cout(Cout), .BAout(BAout), .PCout(PCout),
.Zlowout(Zlowout),
```

```
        .MDRout(MDRout), .Gra(Gra), .Grb(Grb), .Grc(Grc), .Rin(Rin), .Rout(Rout), .MARin(MARin),
        .Zin(Zin), .PCin(PCin), .MDRin(MDRin), .IRin(IRin), .Yin(Yin), .IncPC(IncPC), .read(Read),
        .ADD(ADD), .clk(Clock), .MAR(MAR), .R0(R0), .R1(R1), .R2(R2), .R3(R3), .R4(R4),
        .R5(R5), .R6(R6), .R7(R7), .R8(R8), .R9(R9), .R10(R10), .R11(R11), .R12(R12), .R13(R13), .R14(R14),
        .R15(R15), .Hi(Hi), .Lo(Lo), .IR(IR),.Z(Z), .AND(AND), .OR(OR), .SUB(SUB),.SHR(SHR), .SHL(SHL),
        .ROR(ROR), .ROL(ROL), .NEG(NEG), .NOT(NOT), .bus_mux_out(bus_mux_out), .ALUout(ALUout),
        .ram_data(ram_data), .MDR(MDR), .PC(PC), .C_sign_ext(C_sign_ext), .Rins(Rins), .Routs(Routs));

initial begin
  Clock = 0;
  $dumpfile("p2_datapath_tb.vcd");
  $dumpvars(0, DUT);
  forever #10 Clock = ~Clock;
end

initial begin
  #500 $finish;
end

always @(posedge Clock) begin
  case (Present_state)
                Default : Present_state = T0;
                T0 : Present_state = T1;
                T1 : #10 Present_state = T2;
                T2 : #10 Present_state = T3;
                T3 : #10 Present_state = T4;
                T4 : #20 Present_state = T5;
                T5:  #40 Present_state = T6;
                T6 : #60 Present_state = T7;
```

```
  endcase // case (Present_state)
end // always @ (posedge Clock)

always @(Present_state) begin
  case(Present_state)
        // Assert clear to load zero's into all registers
        Default: begin
          clear <= 1;
          PCout <= 0;
          Zlowout <= 0;
          MDRout <= 0;
          Cout <= 0;
          Rin <= 0;
          Rout <= 0;
          Gra <= 0;
          Grb <= 0;
          Grc <= 0;
          MARin <= 0;
          Zin <= 0;
          PCin <= 0;
          MDRin <= 0;
          IRin <= 0;
          Yin <= 0;
          IncPC <= 0;
          Read <= 0;
          AND <= 0;
                OR <= 0;
                SUB <= 0;
                ADD <= 0;
```

```
            SHL <= 0;
            SHR <= 0;
            ROL <= 0;
            ROR <= 0;
            NEG <= 0;
            NOT <= 0;
            BAout <= 0;
end // case: Default

// PC loaded with 'h0 due to clear
// Get PC in MAR to go to mem and get ld instr (@ addr 0), get PC+4 in Z
T0: begin
            clear <= 0;
            PCout <= 1;
  MARin <= 1;
  IncPC <= 1;
  Zin <= 1;
end
// Put PC+4 in PC, load instr from memory into MDR
T1: begin
            PCout <= 0; MARin <= 0;
  IncPC <= 0;
            #20 Zlowout <= 1;
  PCin <= 1; Read <= 1;
  MDRin <= 1;
            Zin <= 0;
end
// Load instr into IR
T2: begin
```

```
            #10 MDRin <= 0;
  Zlowout <= 0;
            PCin <= 0;
  Read <= 0;
  #10 MDRout <= 1;
  IRin <= 1;
end
// Put either 0's on bus if R0 selected or contents of Rb
// Put bus contents in Y
T3: begin
            #10 MDRout <= 0;
  IRin <= 0;
  #10 Grb <= 1;
  BAout <= 1;
  Yin <= 1;
end
// Put C_sign_ext in Z
T4: begin
  #10
            BAout <= 0;
  Grb <= 0;
  Yin <= 0;
            Cout <= 1;
  ADD <= 1;
  Zin <= 1;
end
// Put C_sign_ext in MAR
T5: begin
  #20 Cout <= 0;
```

```
            ADD <= 0;
                Zin <= 0;
                Zlowout <= 1;
            MARin <= 1;
            #20 Zlowout <= 0;
            MARin <= 0;
          end
          T6: begin
                // For some reason any assignments i do in this phase don't happen
                // I think it's being skipped due to previous # delays...to fix
          end
          // Read data from RAM into MDR
          // Select R1 as Ra, put data from mem loc in R1
          T7: begin
            MDRin <= 1;
                Read <= 1;
            #20 MDRout <= 1;
            Gra <= 1;
            Rin <= 1;
          end
     endcase // case (Present_state)
  end // always @ (Present_state)
endmodule
```

mf_instr_tb

```
module mf_instr_tb();
  reg PCout, Zlowout, MDRout, Cout, HIout, LOout;
  reg MARin, Zin, PCin, MDRin, IRin, Yin, Gra, Grb, Grc, Rin, Rout, BAout, HIin, LOin;
  reg IncPC, Read, Write, ADD, AND, OR, SUB, SHR, SHL, ROL, ROR, NEG, NOT;
  reg Clock, clear;
```

```
        wire [31:0] Mdatain, ram_data, MDR, MAR, R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12,
        R13, R14, R15, Hi, Lo, PC, bus_mux_out, IR, C_sign_ext;
        wire [63:0] Z, ALUout;
  wire [15:0] Rins, Routs;

  parameter Default = 4'b0000, Reg_load_1a = 4'b0001, Reg_load_1b = 4'b0010, T0 = 4'b0011, T1 =
4'b0100,
        T2 = 4'b0101, T3 = 4'b0110;
  reg [3:0]  Present_state = Default;

  Datapath DUT (.HIin(HIin), .LOin(LOin), .HIout(HIout), .LOout(LOout),.write(Write), .Mdatain(Mdatain),
.clear(clear), .Cout(Cout), .BAout(BAout), .PCout(PCout), .Zlowout(Zlowout),
        .MDRout(MDRout), .Gra(Gra), .Grb(Grb), .Grc(Grc), .Rin(Rin), .Rout(Rout), .MARin(MARin),
        .Zin(Zin), .PCin(PCin), .MDRin(MDRin), .IRin(IRin), .Yin(Yin), .IncPC(IncPC), .read(Read),
        .ADD(ADD), .clk(Clock), .MAR(MAR), .R0(R0), .R1(R1), .R2(R2), .R3(R3), .R4(R4),
        .R5(R5), .R6(R6), .R7(R7), .R8(R8), .R9(R9), .R10(R10), .R11(R11), .R12(R12), .R13(R13), .R14(R14),
        .R15(R15), .Hi(Hi), .Lo(Lo), .IR(IR),.Z(Z), .AND(AND), .OR(OR), .SUB(SUB),.SHR(SHR), .SHL(SHL),
        .ROR(ROR), .ROL(ROL), .NEG(NEG), .NOT(NOT), .bus_mux_out(bus_mux_out), .ALUout(ALUout),
        .ram_data(ram_data), .MDR(MDR), .PC(PC), .C_sign_ext(C_sign_ext), .Rins(Rins), .Routs(Routs));

        initial begin
    Clock = 0;
    $dumpfile("p2_datapath_tb.vcd");
    $dumpvars(0, DUT);
    forever #10 Clock = ~Clock;
  end

  initial begin
    #500 $finish;
```

```
end

always @(posedge Clock) begin
  case (Present_state)
                Default : Present_state = Reg_load_1a;
                Reg_load_1a : Present_state = Reg_load_1b;
                Reg_load_1b : #5 Present_state = T0;
                T0 : Present_state = T1;
                T1 : #15 Present_state = T2;
                T2 : #30 Present_state = T3;
  endcase // case (Present_state)
end // always @ (posedge Clock)

always @(Present_state) begin
  case(Present_state)
        // Assert clear to load zero's into all registers
        Default: begin
          clear <= 1;
          PCout <= 0;
          Zlowout <= 0;
          MDRout <= 0;
          Cout <= 0;
          Rin <= 0;
          Rout <= 0;
          Gra <= 0;
          Grb <= 0;
          Grc <= 0;
          MARin <= 0;
          Zin <= 0;
```

```
  PCin <= 0;
  MDRin <= 0;
  IRin <= 0;
  Yin <= 0;
        HIin <= 0;
        LOin <= 0;
        HIout <= 0;
        LOout <= 0;
  IncPC <= 0;
  Write <= 0;
        Read <= 0;
  AND <= 0;
        OR <= 0;
        SUB <= 0;
        ADD <= 0;
        SHL <= 0;
        SHR <= 0;
        ROL <= 0;
        ROR <= 0;
        NEG <= 0;
        NOT <= 0;
        BAout <= 0;
end // case: Default
// Load value of $85 into LO from RAM
Reg_load_1a: begin
        clear <= 0;
        force ram_data = 'h00000055;
  #10 Read <= 1; MDRin <= 1;
  #10 Read <= 0; MDRin <= 0;
```

```
        release ram_data;
end
Reg_load_1b: begin
  #10 MDRout <= 1;
        LOin <= 1;
  #15 MDRout <= 0;
        LOin <= 0;
end
// Get PC in MAR to go to mem and get instr (@ addr 0), get PC+4 in Z
T0: begin
        PCout <= 1;
  MARin <= 1;
  IncPC <= 1;
  Zin <= 1;
end
// Put PC+4 in PC, load mf instr from memory into MDR
T1: begin
        #20 PCout <= 0; MARin <= 0;
  IncPC <= 0; Zlowout <= 1;
  PCin <= 1; Read <= 1;
  MDRin <= 1;
        Zin <= 0;
end
// Load instr into IR
T2: begin
        #15
        MDRin <= 0;
  Zlowout <= 0;
        PCin <= 0;
```

```
            Read <= 0;
            #10 MDRout <= 1;
            IRin <= 1;
          end
          // Store contents of LO in Ra
          T3: begin
                #10 MDRout <= 0;
            IRin <= 0;
            #10 Gra <= 1;
            LOout <= 1;
            Rin <= 1;
          end
  endcase // case (Present_state)
 end // always @ (Present_state)
endmodule
```

st_instr_tb

```
module st_instr_tb();
  reg PCout, Zlowout, MDRout, Cout;
  reg MARin, Zin, PCin, MDRin, IRin, Yin, Gra, Grb, Grc, Rin, Rout, BAout;
  reg IncPC, Read, Write, ADD, AND, OR, SUB, SHR, SHL, ROL, ROR, NEG, NOT;
  reg Clock, clear;
        wire [31:0] Mdatain, ram_data, MDR, MAR, R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12,
        R13, R14, R15, Hi, Lo, PC, bus_mux_out, IR, C_sign_ext;
        wire [63:0] Z, ALUout;
  wire [15:0] Rins, Routs;

  parameter Default = 4'b0000, Reg_load_1a = 4'b0001, Reg_load_1b = 4'b0010, T0 = 4'b0011, T1 =
4'b0100,
        T2 = 4'b0101, T3 = 4'b0110, T4 = 4'b0111, T5 = 4'b1000, T6 = 4'b1001, T7 = 4'b1010, T8 =
4'b1011;
  reg [3:0]  Present_state = Default;

  Datapath DUT (.write(Write), .Mdatain(Mdatain), .clear(clear), .Cout(Cout), .BAout(BAout),
.PCout(PCout), .Zlowout(Zlowout),
        .MDRout(MDRout), .Gra(Gra), .Grb(Grb), .Grc(Grc), .Rin(Rin), .Rout(Rout), .MARin(MARin),
        .Zin(Zin), .PCin(PCin), .MDRin(MDRin), .IRin(IRin), .Yin(Yin), .IncPC(IncPC), .read(Read),
        .ADD(ADD), .clk(Clock), .MAR(MAR), .R0(R0), .R1(R1), .R2(R2), .R3(R3), .R4(R4),
```

```
        .R5(R5), .R6(R6), .R7(R7), .R8(R8), .R9(R9), .R10(R10), .R11(R11), .R12(R12), .R13(R13), .R14(R14),
        .R15(R15), .Hi(Hi), .Lo(Lo), .IR(IR),.Z(Z), .AND(AND), .OR(OR), .SUB(SUB),.SHR(SHR), .SHL(SHL),
        .ROR(ROR), .ROL(ROL), .NEG(NEG), .NOT(NOT), .bus_mux_out(bus_mux_out), .ALUout(ALUout),
        .ram_data(ram_data), .MDR(MDR), .PC(PC), .C_sign_ext(C_sign_ext), .Rins(Rins), .Routs(Routs));

        initial begin
  Clock = 0;
  $dumpfile("p2_datapath_tb.vcd");
  $dumpvars(0, DUT);
  forever #10 Clock = ~Clock;
end

initial begin
  #500 $finish;
end

always @(posedge Clock) begin
  case (Present_state)
                Default : Present_state = Reg_load_1a;
                Reg_load_1a : Present_state = Reg_load_1b;
                Reg_load_1b : Present_state = T0;
                T0 : Present_state = T1;
                T1 : #10 Present_state = T2;
                T2 : #10 Present_state = T3;
                T3 : #10 Present_state = T4;
                T4 : #20 Present_state = T5;
                T5:  #40 Present_state = T6;
                T6 : #60 Present_state = T7;
                T7 : #20 Present_state = T8;
  endcase // case (Present_state)
end // always @ (posedge Clock)

always @(Present_state) begin
  case(Present_state)
        // Assert clear to load zero's into all registers
        Default: begin
          clear <= 1;
          PCout <= 0;
          Zlowout <= 0;
          MDRout <= 0;
          Cout <= 0;
          Rin <= 0;
          Rout <= 0;
          Gra <= 0;
          Grb <= 0;
          Grc <= 0;
          MARin <= 0;
          Zin <= 0;
```

```
  PCin <= 0;
  MDRin <= 0;
  IRin <= 0;
  Yin <= 0;
  IncPC <= 0;
  Write <= 0;
        Read <= 0;
  AND <= 0;
        OR <= 0;
        SUB <= 0;
        ADD <= 0;
        SHL <= 0;
        SHR <= 0;
        ROL <= 0;
        ROR <= 0;
        NEG <= 0;
        NOT <= 0;
        BAout <= 0;
end // case: Default
// Load value of $85 into R1 from RAM
Reg_load_1a: begin
        clear <= 0;
        force ram_data = 'h00000055;
  #10 Read <= 1; MDRin <= 1;
  #10 Read <= 0; MDRin <= 0;
        release ram_data;
end
Reg_load_1b: begin
        force Rins = 'h0002;
  #10 MDRout <= 1;
  #15 MDRout <= 0;
        release Rins;
end
// Get PC in MAR to go to mem and get st instr (@ addr 0), get PC+4 in Z
T0: begin
        PCout <= 1;
  MARin <= 1;
  IncPC <= 1;
  Zin <= 1;
end
// Put PC+4 in PC, load instr (st $90, R1) from memory into MDR
T1: begin
        PCout <= 0; MARin <= 0;
  IncPC <= 0;
        #20 Zlowout <= 1;
  PCin <= 1; Read <= 1;
  MDRin <= 1;
        Zin <= 0;
```

```
end
// Load instr into IR
T2: begin
        #10 MDRin <= 0;
  Zlowout <= 0;
        PCin <= 0;
  Read <= 0;
  #10 MDRout <= 1;
  IRin <= 1;
end
// Put either 0's on bus if R0 selected or contents of Rb
// Put bus contents in Y
T3: begin
        #10 MDRout <= 0;
  IRin <= 0;
  #10 Grb <= 1;
  BAout <= 1;
  Yin <= 1;
end
// Put C_sign_ext = 32'h5A in Z
T4: begin
  #10
        BAout <= 0;
  Grb <= 0;
  Yin <= 0;
        Cout <= 1;
  ADD <= 1;
  Zin <= 1;
end
// Put C_sign_ext = 32'h5A in MAR
T5: begin
  #20 Cout <= 0;
  ADD <= 0;
        Zin <= 0;
        Zlowout <= 1;
  MARin <= 1;
  #20 Zlowout <= 0;
  MARin <= 0;
end
// Ignore this phase :(
T6: begin
        // For some reason any assignments i do in this phase don't happen
        // I think it's being skipped due to previous # delays...to fix
end
// Take data in R1 ($85 = 'h55) and write to mem
T7: begin
        Rout <= 1;
        Gra <= 1;
```

```
            MDRin <= 1;
                Write <= 1;
        end
        // Read back memory at address $90 ('h5A), should be $85
        // Will be in MDR
        T8 : begin
                #20
                Write <= 0; Read <= 1; MDRin <= 1;
        end
    endcase // case (Present_state)
  end // always @ (Present_state)
endmodule
```

p2_datapath_br_tb

```
module p2_datapath_br_tb();

        reg PCout, Zlowout, MDRout, Cout;
  reg MARin, Zin, PCin, MDRin, IRin, Yin, Gra, Grb, Grc, Rin, Rout, BAout;
  reg IncPC, Read, ADD, AND, OR, SUB, SHR, SHL, ROL, ROR, NEG, NOT;
  reg Clock, clear;
        reg CONin;
        wire CON_FF;
        wire [31:0] Mdatain, ram_data, MDR, MAR, R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12,
        R13, R14, R15, Hi, Lo, PC, bus_mux_out, IR, C_sign_ext;
        wire [63:0] Z, ALUout;
  wire [15:0] Rins, Routs;

        wire t1, t2,t3,t4;

  parameter Default = 4'b0000, Reg_load_1a = 4'b0001, Reg_load_1b = 4'b0010, T0 = 4'b0011, T1 =
4'b0100,
        T2 = 4'b0101, T3 = 4'b0110, T4 = 4'b0111, T5 = 4'b1000, T6 = 4'b1001;
  reg [3:0]  Present_state = Default;

  Datapath DUT (.Mdatain(Mdatain), .clear(clear), .Cout(Cout), .BAout(BAout), .PCout(PCout),
.Zlowout(Zlowout),
        .MDRout(MDRout), .Gra(Gra), .Grb(Grb), .Grc(Grc), .Rin(Rin), .Rout(Rout), .MARin(MARin),
        .Zin(Zin), .PCin(PCin), .MDRin(MDRin), .IRin(IRin), .Yin(Yin), .IncPC(IncPC), .read(Read),
        .ADD(ADD), .CON_FF(CON_FF), .clk(Clock), .MAR(MAR),  .CONin(CONin), .R0(R0), .R1(R1),
.R2(R2), .R3(R3), .R4(R4),
        .R5(R5), .R6(R6), .R7(R7), .R8(R8), .R9(R9), .R10(R10), .R11(R11), .R12(R12), .R13(R13), .R14(R14),
        .R15(R15), .Hi(Hi), .Lo(Lo), .IR(IR),.Z(Z), .AND(AND), .OR(OR), .SUB(SUB),.SHR(SHR), .SHL(SHL),
        .ROR(ROR), .ROL(ROL), .NEG(NEG), .NOT(NOT), .bus_mux_out(bus_mux_out), .ALUout(ALUout),
        .ram_data(ram_data), .MDR(MDR), .PC(PC), .C_sign_ext(C_sign_ext), .Rins(Rins), .Routs(Routs)
        , .test1(t1), .test2(t2), .test3(t3), .test4(t4)
        );

  initial begin
```

```
  Clock = 0;
  $dumpfile("p2_datapath_tb.vcd");
  $dumpvars(0, DUT);
  forever #10 Clock = ~Clock;
end

initial begin
  #500 $finish;
end

always @(posedge Clock) begin
  case (Present_state)
                Default : Present_state = Reg_load_1a;
                Reg_load_1a : Present_state = Reg_load_1b;
                Reg_load_1b : #10 Present_state = T0;
                T0 : #10 Present_state = T1;
                T1 : #10 Present_state = T2;
                T2 : #10 Present_state = T3;
                T3 : #10 Present_state = T4;
                T4 : #20 Present_state = T5;
                T5 : #40 Present_state = T6;
  endcase // case (Present_state)
end // always @ (posedge Clock)

always @(Present_state) begin
  case(Present_state)
        // Assert clear to load zero's into all registers
        Default: begin
          clear <= 1;
          PCout <= 0;
          Zlowout <= 0;
          MDRout <= 0;
          Cout <= 0;
          Rin <= 0;
          Rout <= 0;
          Gra <= 0;
          Grb <= 0;
          Grc <= 0;
          MARin <= 0;
          Zin <= 0;
          PCin <= 0;
          MDRin <= 0;
          IRin <= 0;
          Yin <= 0;
          IncPC <= 0;
          Read <= 0;
          AND <= 0;
                OR <= 0;
```

```
        SUB <= 0;
        ADD <= 0;
        SHL <= 0;
        SHR <= 0;
        ROL <= 0;
        ROR <= 0;
        NEG <= 0;
        NOT <= 0;
        BAout <= 0;
        CONin <= 0;
end // case: Default
Reg_load_1a: begin
        clear <= 0;
        force ram_data = 'hFFFFFFF5; // preload with -10
  #10 Read <= 1; MDRin <= 1;
  #10 Read <= 0; MDRin <= 0;
        release ram_data;
end
Reg_load_1b: begin
        force Rins = 'h0004;
  #10 MDRout <= 1;
  #15 MDRout <= 0;
        release Rins;
end
// PC loaded with 'h0 due to clear
// Get PC in MAR to go to mem and get ldi instr (@ addr 0), get PC+4 in Z
T0: begin
        PCout <= 1;
  MARin <= 1;
  IncPC <= 1;
  Zin <= 1;
end
// Put PC+4 in PC, load instr (ldi R1, 85) from memory into MDR
T1: begin
        PCout <= 0;
        MARin <= 0;
  IncPC <= 0;

        #20 Zlowout <= 1;
  PCin <= 1;
        Read <= 1;
  MDRin <= 1;
        Zin <= 0;
end
// Load instr into IR
T2: begin
        #10 MDRin <= 0;
  Zlowout <= 0;
```

```
                PCin <= 0;
          Read <= 0;

          #10 MDRout <= 1;
          IRin <= 1;
        end
        // Put 0's on bus and in Y
        T3: begin
                #10 MDRout <= 0;
          IRin <= 0;

          #10 Gra <= 1;
          Rout <= 1;
          CONin <= 1;
        end

        T4: begin
          #10 Gra <= 0;
          Rout <= 0;
          CONin <= 0;

                PCout <= 1;
                Yin <= 1;
        end

        T5: begin
          PCout <= 0;
                Yin <= 0;

                Cout <= 1;
                ADD <= 1;
                Zin <= 1;
        end
        T6: begin
                Cout <= 0;
                ADD <= 0;
                Zin <= 0;

                Zlowout <= 1;
                PCin <= CON_FF;
        end

    endcase // case (Present_state)
  end // always @ (Present_state)

endmodule
```

p2_datapath_ldi_tb

```
module p2_datapath_ldi_tb();


        reg PCout, Zlowout, MDRout, Cout;
  reg MARin, Zin, PCin, MDRin, IRin, Yin, Gra, Grb, Grc, Rin, Rout, BAout;
  reg IncPC, Read, ADD, AND, OR, SUB, SHR, SHL, ROL, ROR, NEG, NOT;
  reg Clock, clear;
        reg [31:0] preload;
        wire [31:0] Mdatain, ram_data, MDR, MAR, R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12,
        R13, R14, R15, Hi, Lo, PC, bus_mux_out, IR, C_sign_ext;
        wire [63:0] Z, ALUout;
  wire [15:0] Rins, Routs;


  parameter Default = 4'b0000, T0 = 4'b0001, T1 = 4'b0010, T2 = 4'b0011,
        T3 = 4'b0100, T4 = 4'b0101, T5 = 4'b0110, T6 = 4'b0111, T7 = 4'b1000;
  reg [3:0]  Present_state = Default;


  Datapath DUT (.Mdatain(Mdatain), .preload(preload), .clear(clear), .Cout(Cout), .BAout(BAout),
.PCout(PCout), .Zlowout(Zlowout),
        .MDRout(MDRout), .Gra(Gra), .Grb(Grb), .Grc(Grc), .Rin(Rin), .Rout(Rout), .MARin(MARin),
        .Zin(Zin), .PCin(PCin), .MDRin(MDRin), .IRin(IRin), .Yin(Yin), .IncPC(IncPC), .read(Read),
        .ADD(ADD), .clk(Clock), .MAR(MAR), .R0(R0), .R1(R1), .R2(R2), .R3(R3), .R4(R4),
        .R5(R5), .R6(R6), .R7(R7), .R8(R8), .R9(R9), .R10(R10), .R11(R11), .R12(R12), .R13(R13), .R14(R14),
        .R15(R15), .Hi(Hi), .Lo(Lo), .IR(IR),.Z(Z), .AND(AND), .OR(OR), .SUB(SUB),.SHR(SHR), .SHL(SHL),
        .ROR(ROR), .ROL(ROL), .NEG(NEG), .NOT(NOT), .bus_mux_out(bus_mux_out), .ALUout(ALUout),
        .ram_data(ram_data), .MDR(MDR), .PC(PC), .C_sign_ext(C_sign_ext), .Rins(Rins), .Routs(Routs));


  initial begin
    Clock = 0;
```

```
  $dumpfile("p2_datapath_tb.vcd");
  $dumpvars(0, DUT);
  forever #10 Clock = ~Clock;
end

initial begin
  #500 $finish;
end

always @(posedge Clock) begin
  case (Present_state)
                Default : Present_state = T0;
                T0 : Present_state = T1;
                T1 : #10 Present_state = T2;
                T2 : #10 Present_state = T3;
                T3 : #10 Present_state = T4;
                T4 : #20 Present_state = T5;
                T5 : #40 Present_state = T6;
                T6 : #60 Present_state = T7;
  endcase // case (Present_state)
end // always @ (posedge Clock)

always @(Present_state) begin
  case(Present_state)
        // Assert clear to load zero's into all registers
        Default: begin
          clear <= 1;
          PCout <= 0;
          Zlowout <= 0;
```

```
MDRout <= 0;
Cout <= 0;
Rin <= 0;
Rout <= 0;
Gra <= 0;
Grb <= 0;
Grc <= 0;
MARin <= 0;
Zin <= 0;
PCin <= 0;
MDRin <= 0;
IRin <= 0;
Yin <= 0;
IncPC <= 0;
Read <= 0;
AND <= 0;
        OR <= 0;
        SUB <= 0;
        ADD <= 0;
        SHL <= 0;
        SHR <= 0;
        ROL <= 0;
        ROR <= 0;
        NEG <= 0;
        NOT <= 0;
        BAout <= 0;

end // case: Default
```

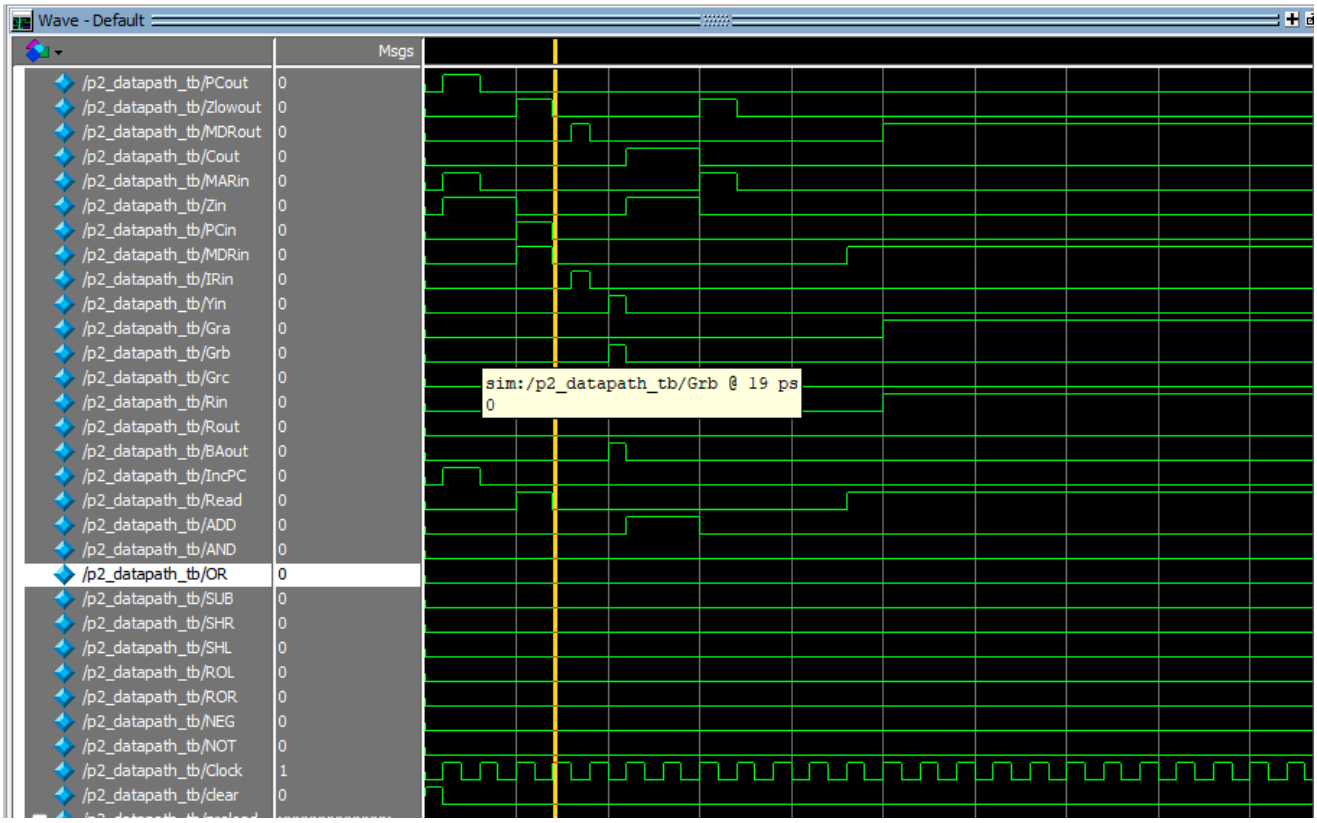
```
// PC loaded with 'h0 due to clear
// Get PC in MAR to go to mem and get ldi instr (@ addr 0), get PC+4 in Z
T0: begin
        clear <= 0;
        PCout <= 1;
  MARin <= 1;
  IncPC <= 1;
  Zin <= 1;
end
// Put PC+4 in PC, load instr (ldi R1, 85) from memory into MDR
T1: begin
        PCout <= 0; MARin <= 0;
  IncPC <= 0;
        #20 Zlowout <= 1;
  PCin <= 1; Read <= 1;
  MDRin <= 1;
        Zin <= 0;
end
// Load instr into IR
T2: begin
        #10 MDRin <= 0;
  Zlowout <= 0;
        PCin <= 0;
  Read <= 0;

  #10 MDRout <= 1;
  IRin <= 1;
end
// Put 0's on bus and in Y
```

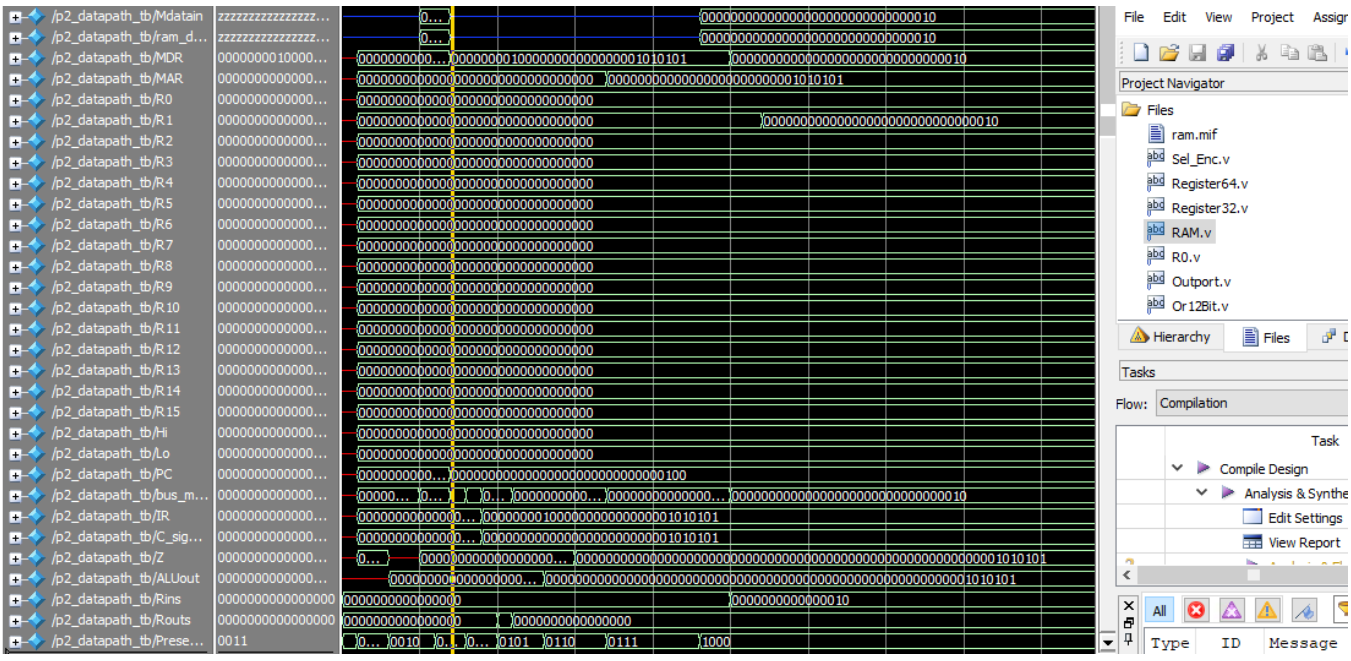
```
T3: begin
        #10 MDRout <= 0;
  IRin <= 0;

  #10 Grb <= 1;
  BAout <= 1;
  Yin <= 1;
end
// Put C_sign_ext = 32'h85 in Z
T4: begin
  #10
        BAout <= 0;
  Grb <= 0;
  Yin <= 0;

        Cout <= 1;
  ADD <= 1;
  Zin <= 1;
end
// Put C_sign_ext = 32'h85 in MAR
T5: begin
  #20 Cout <= 0;
  ADD <= 0;
        Zin <= 0;

        Zlowout <= 1;
        Gra <= 1;
  Rin <= 1;
end
```

```
            T6: begin
                    // For some reason any assignments i do in this phase don't happen
                    // I think it's being skipped due to previous # delays...to fix
            end
            // Read data from $85 in RAM into MDR (shoud be 'h2)
            // Select R1 as Ra, put data from mem loc $85 in R1 ('h2)
            T7: begin

            end
      endcase // case (Present_state)
   end // always @ (Present_state)


endmodule
```

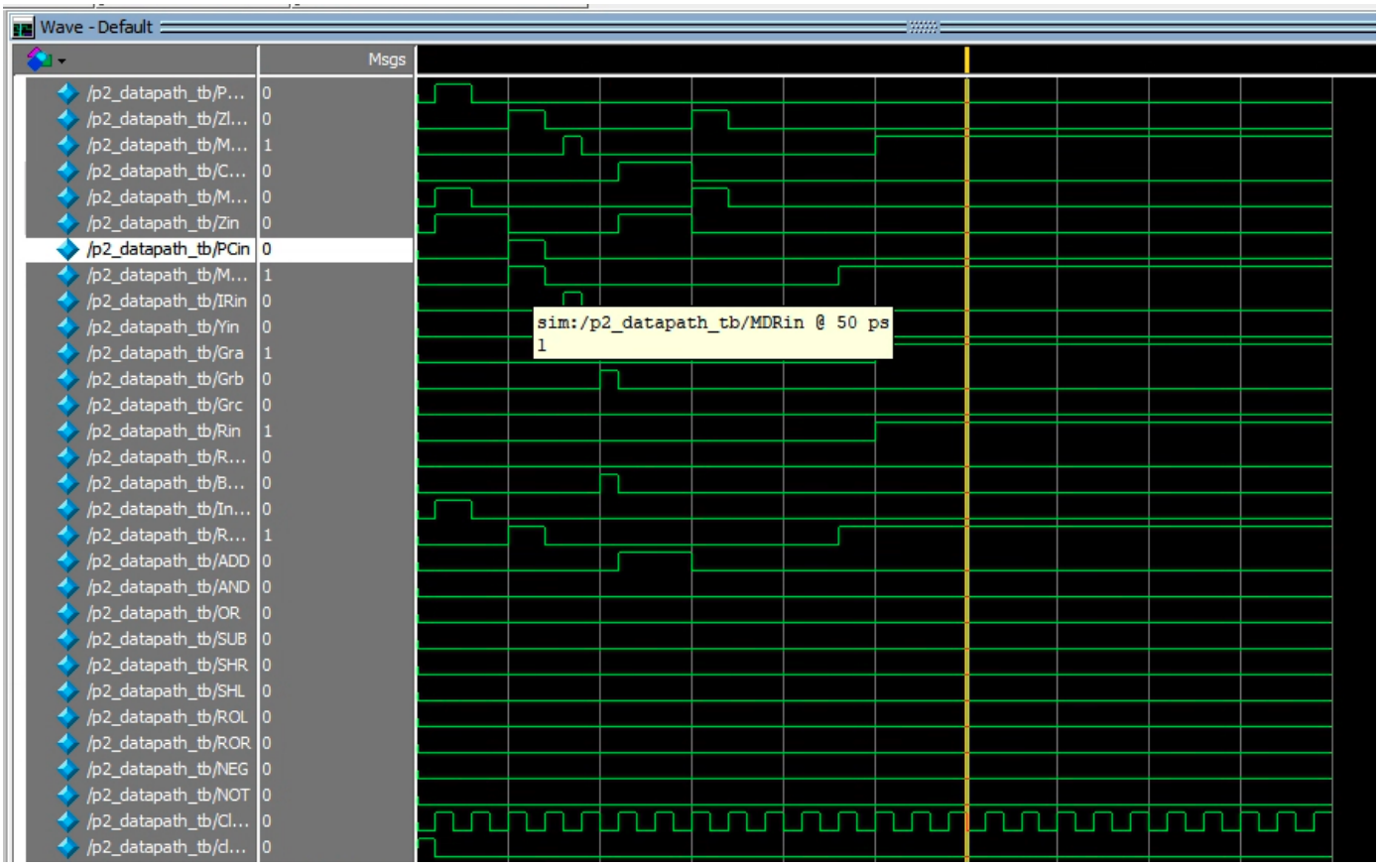
Functional Simulations

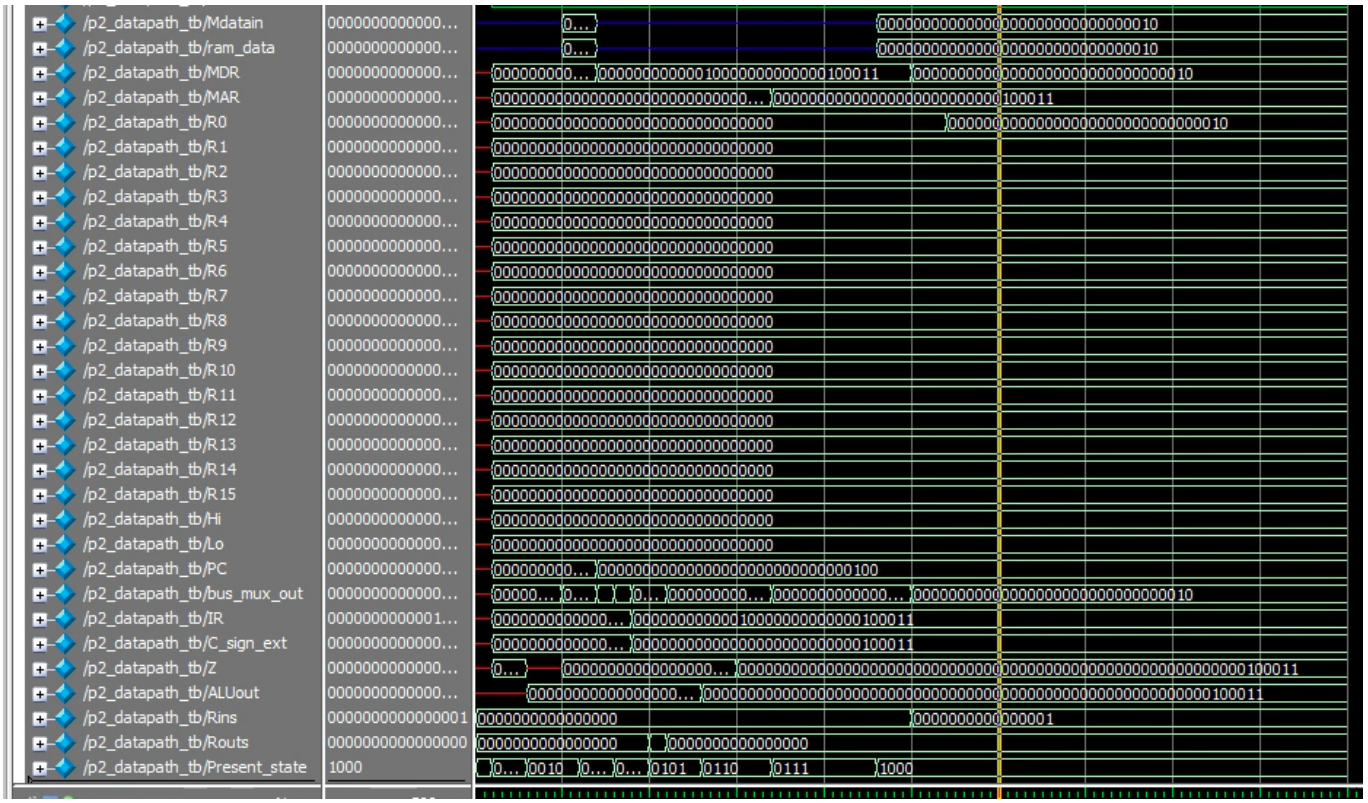
ld R1, $85



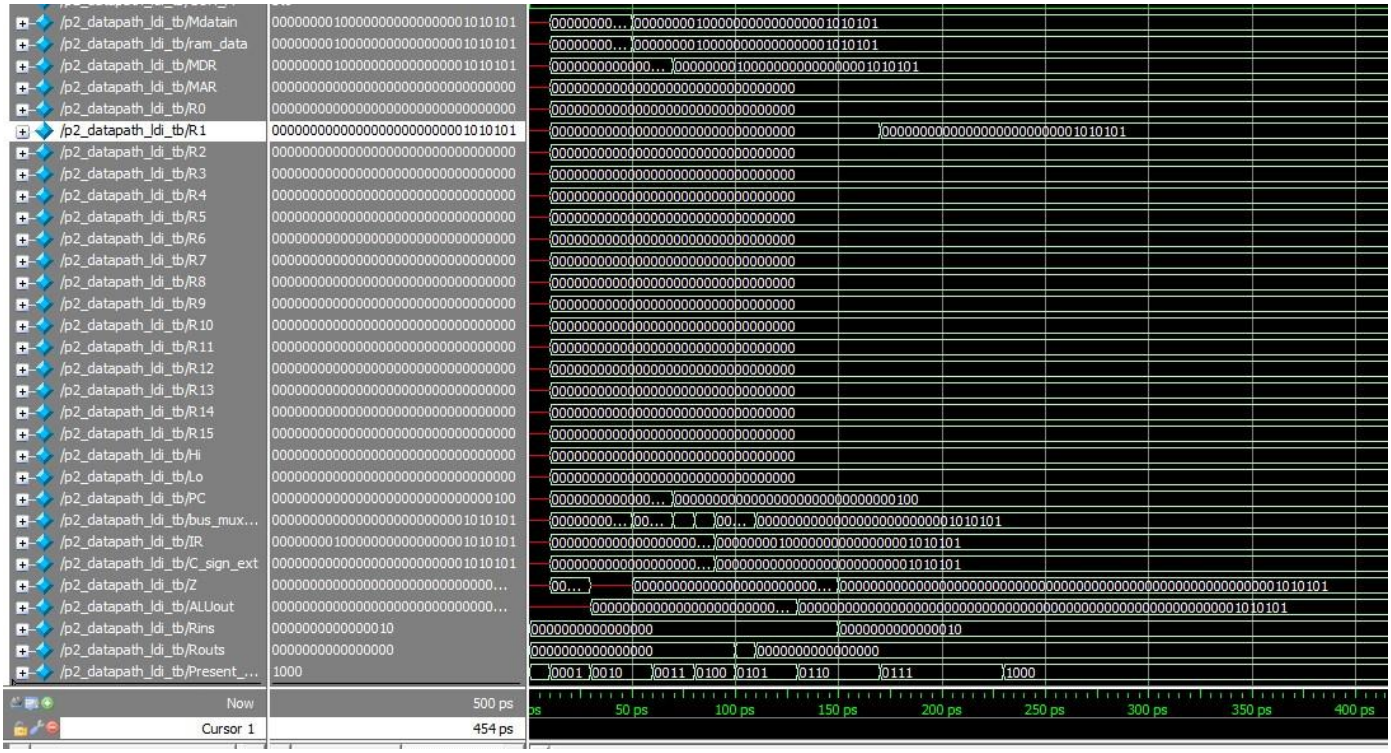


ld R0, $35(R1)

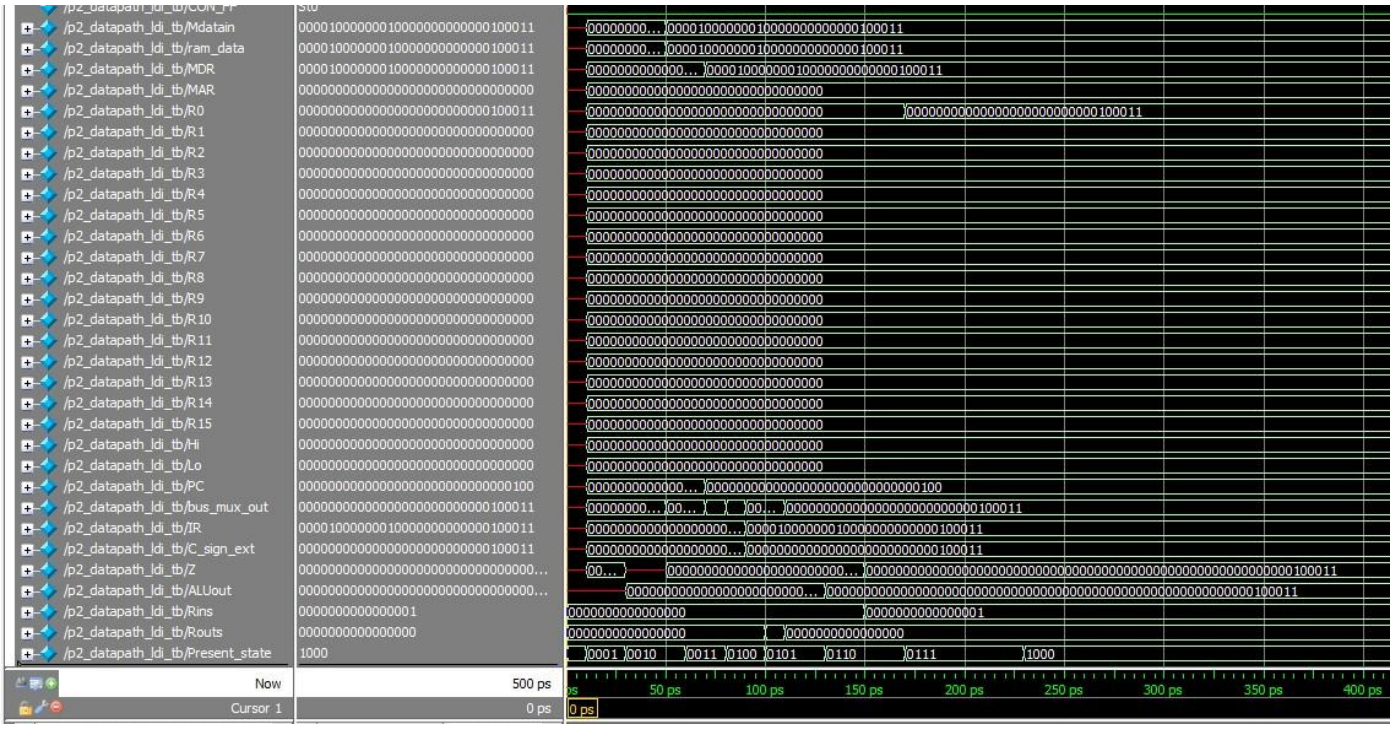

Wave - Default
Msgs
/p2_datapath_tb/P... 0
/p2_datapath_tb/Zl... 0
/p2_datapath_tb/M... 1
/p2_datapath_tb/C... 0
/p2_datapath_tb/M... 0
/p2_datapath_tb/Zin 0
/p2_datapath_tb/PCin 0
/p2_datapath_tb/M... 1
/p2_datapath_tb/IRin 0
/p2_datapath_tb/Yin 0
/p2_datapath_tb/Gra 1
/p2_datapath_tb/Grb 0
/p2_datapath_tb/Grc 0
/p2_datapath_tb/Rin 1
/p2_datapath_tb/R... 0
/p2_datapath_tb/B... 0
/p2_datapath_tb/In... 0
/p2_datapath_tb/R... 1
/p2_datapath_tb/ADD 0
/p2_datapath_tb/AND 0
/p2_datapath_tb/OR 0
/p2_datapath_tb/SUB 0
/p2_datapath_tb/SHR 0
/p2_datapath_tb/SHL 0
/p2_datapath_tb/ROL 0
/p2_datapath_tb/ROR 0
/p2_datapath_tb/NEG 0
/p2_datapath_tb/NOT 0
/p2_datapath_tb/Cl... 0
/p2_datapath_tb/cl... 0
sim:/p2_datapath_tb/MDRin @ 50 ps
1


/p2_datapath_tb/Mdatain
/p2_datapath_tb/ram_data
/p2_datapath_tb/MDR
/p2_datapath_tb/MAR
/p2_datapath_tb/R0
/p2_datapath_tb/R1
/p2_datapath_tb/R2
/p2_datapath_tb/R3
/p2_datapath_tb/R4
/p2_datapath_tb/R5
/p2_datapath_tb/R6
/p2_datapath_tb/R7
/p2_datapath_tb/R8
/p2_datapath_tb/R9
/p2_datapath_tb/R10
/p2_datapath_tb/R11
/p2_datapath_tb/R12
/p2_datapath_tb/R13
/p2_datapath_tb/R14
/p2_datapath_tb/R15
/p2_datapath_tb/Hi
/p2_datapath_tb/Lo
/p2_datapath_tb/PC
/p2_datapath_tb/bus_mux_out
/p2_datapath_tb/IR
/p2_datapath_tb/C_sign_ext
/p2_datapath_tb/Z
/p2_datapath_tb/ALUout
/p2_datapath_tb/Rins
/p2_datapath_tb/Routs
/p2_datapath_tb/Present_state
1000
0010
0101
0110
0111
1000

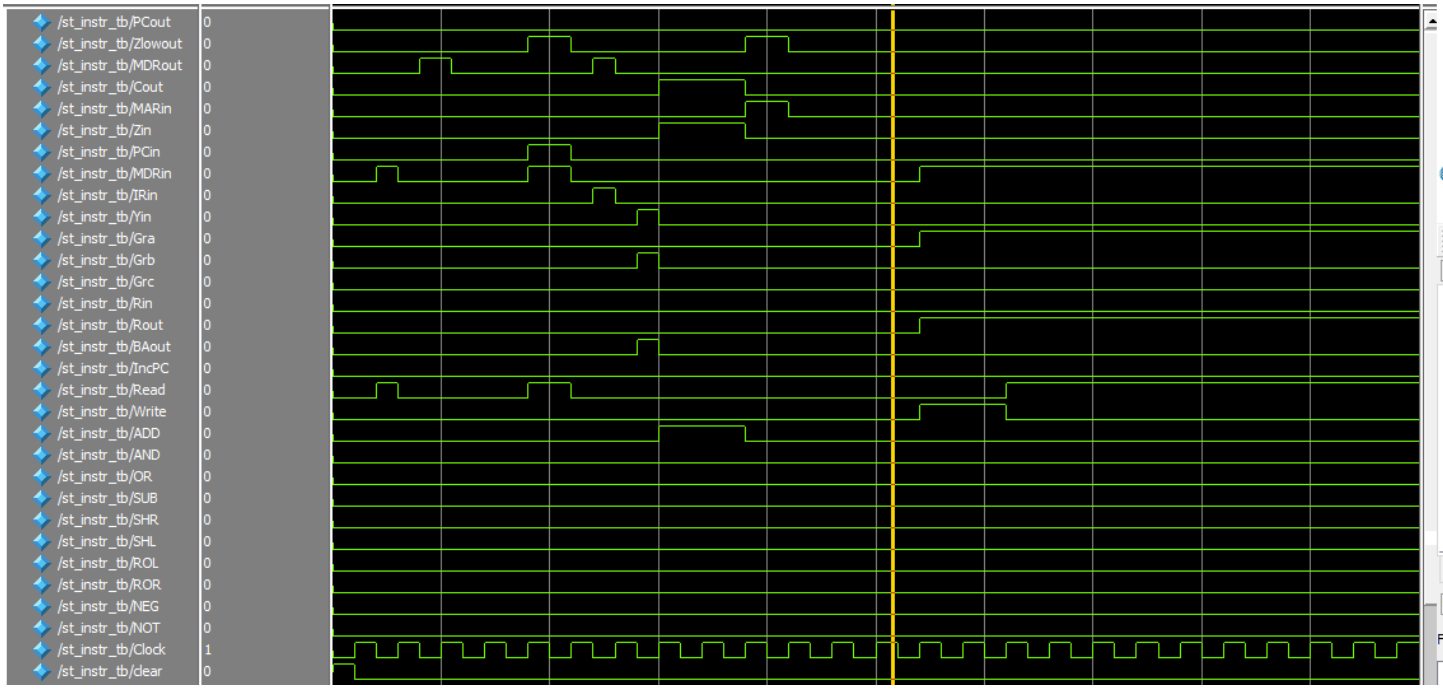
ldi R1, $85

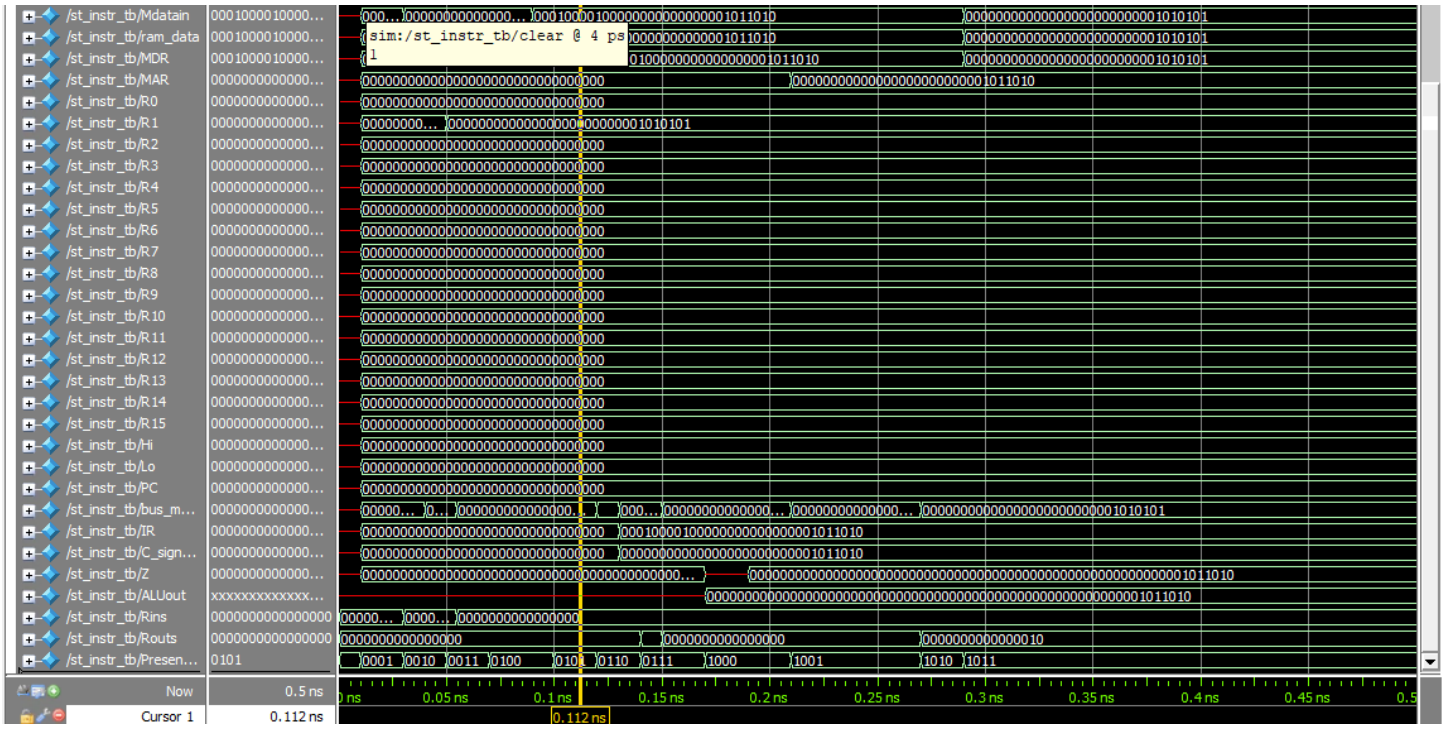


ldi R0, $35(R1)

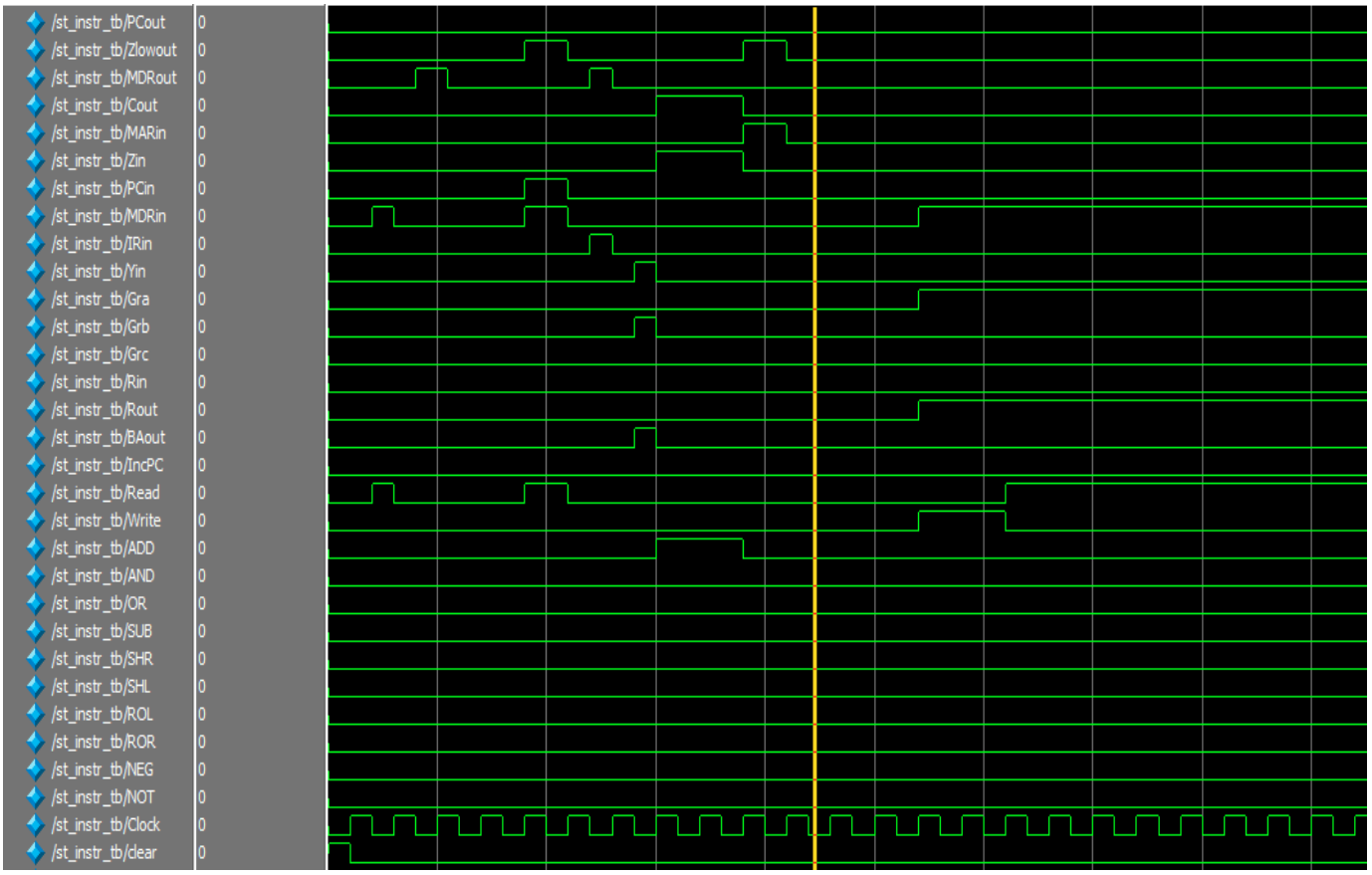


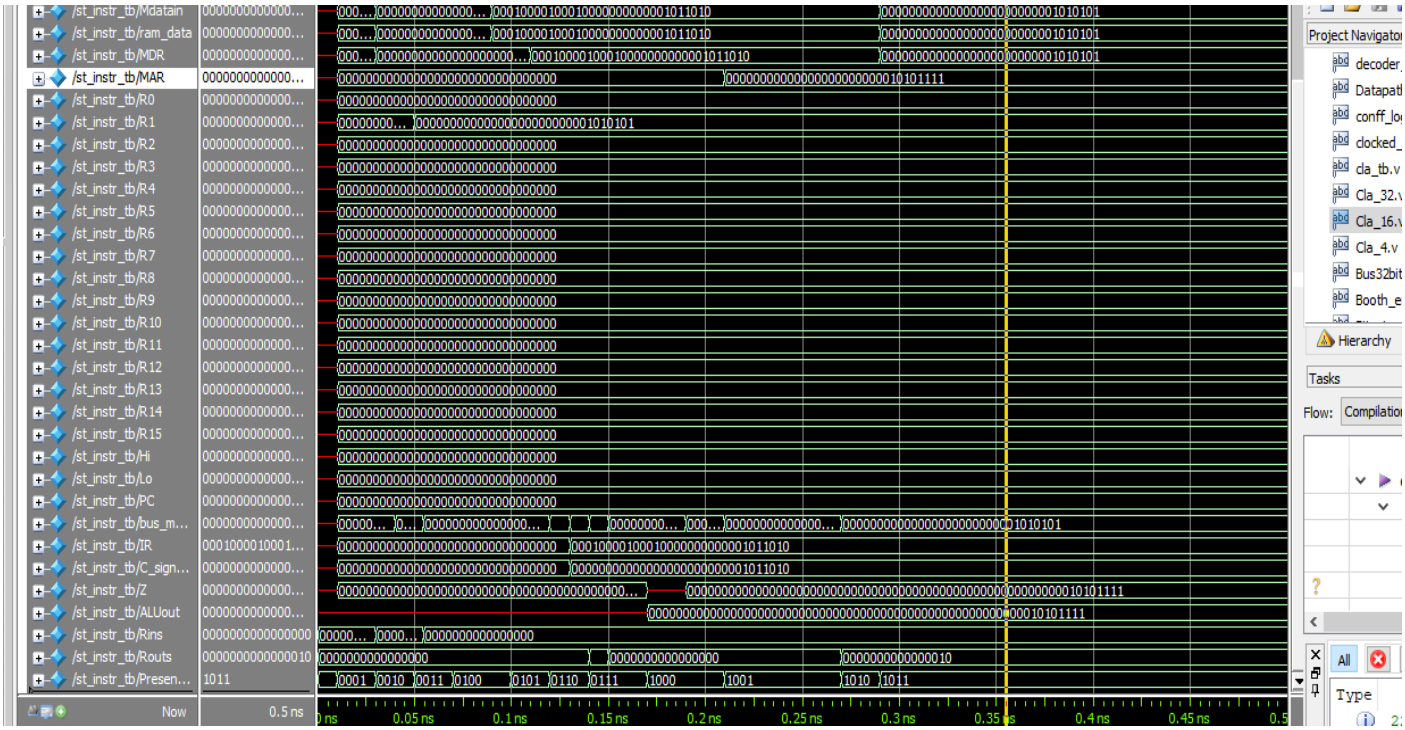
st $90, R1


/st_instr_tb/PCout
/st_instr_tb/Zlowout
/st_instr_tb/MDRout
/st_instr_tb/Cout
/st_instr_tb/MARin
/st_instr_tb/Zin
/st_instr_tb/PCin
/st_instr_tb/MDRin
/st_instr_tb/IRin
/st_instr_tb/Yin
/st_instr_tb/Gra
/st_instr_tb/Grb
/st_instr_tb/Grc
/st_instr_tb/Rin
/st_instr_tb/Rout
/st_instr_tb/BAout
/st_instr_tb/IncPC
/st_instr_tb/Read
/st_instr_tb/Write
/st_instr_tb/ADD
/st_instr_tb/AND
/st_instr_tb/OR
/st_instr_tb/SUB
/st_instr_tb/SHR
/st_instr_tb/SHL
/st_instr_tb/ROL
/st_instr_tb/ROR
/st_instr_tb/NEG
/st_instr_tb/NOT
/st_instr_tb/Clock
/st_instr_tb/clear

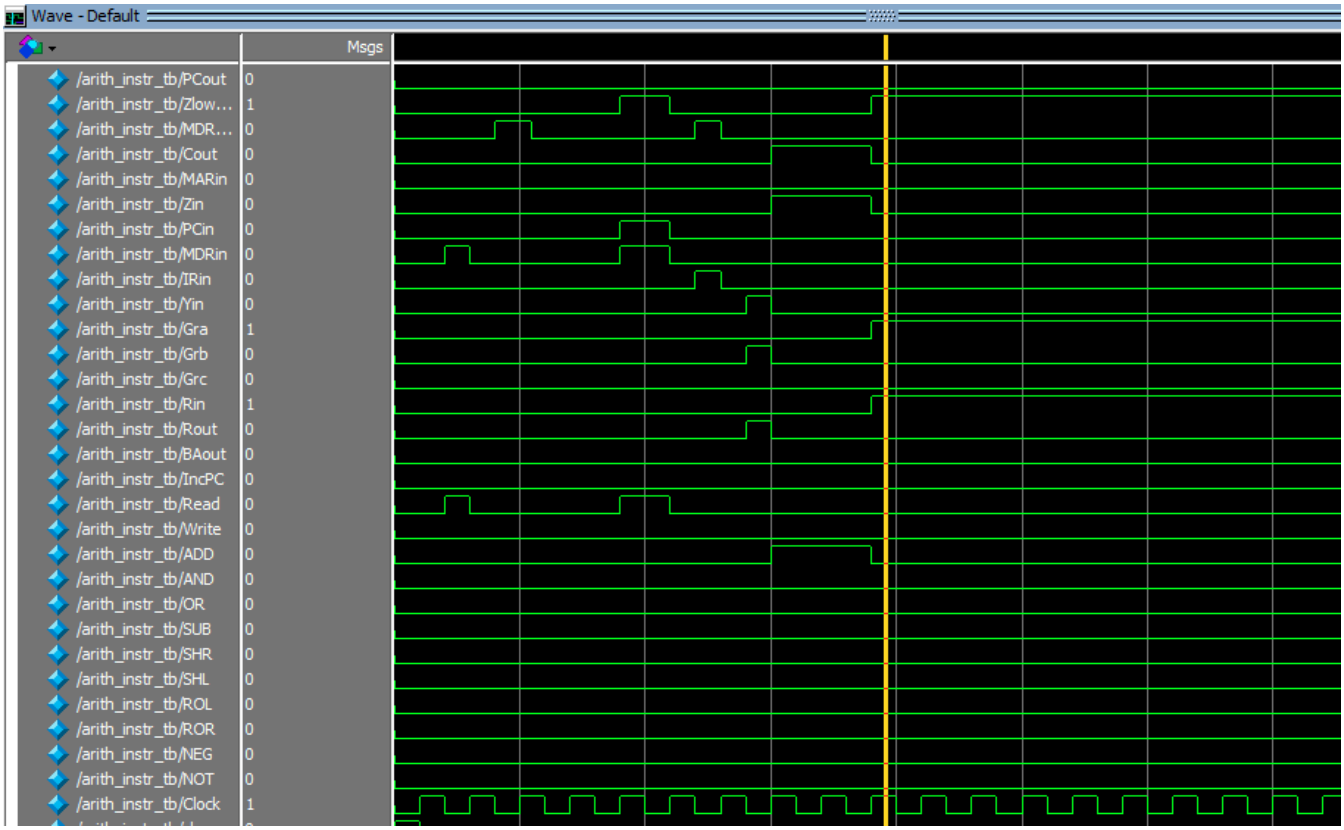

/st_instr_tb/Mdatain
/st_instr_tb/ram_data
/st_instr_tb/MDR
/st_instr_tb/MAR
/st_instr_tb/R0
/st_instr_tb/R1
/st_instr_tb/R2
/st_instr_tb/R3
/st_instr_tb/R4
/st_instr_tb/R5
/st_instr_tb/R6
/st_instr_tb/R7
/st_instr_tb/R8
/st_instr_tb/R9
/st_instr_tb/R10
/st_instr_tb/R11
/st_instr_tb/R12
/st_instr_tb/R13
/st_instr_tb/R14
/st_instr_tb/R15
/st_instr_tb/Hi
/st_instr_tb/Lo
/st_instr_tb/PC
/st_instr_tb/bus_m...
/st_instr_tb/IR
/st_instr_tb/C_sign...
/st_instr_tb/Z
/st_instr_tb/ALUout
/st_instr_tb/Rins
/st_instr_tb/Routs
/st_instr_tb/Presen...
sim:/st_instr_tb/clear @ 4 ps
1
Now
0.5 ns
Cursor 1
0.112 ns
0.05 ns
0.1 ns
0.15 ns
0.2 ns
0.25 ns
0.3 ns
0.35 ns
0.4 ns
0.45 ns

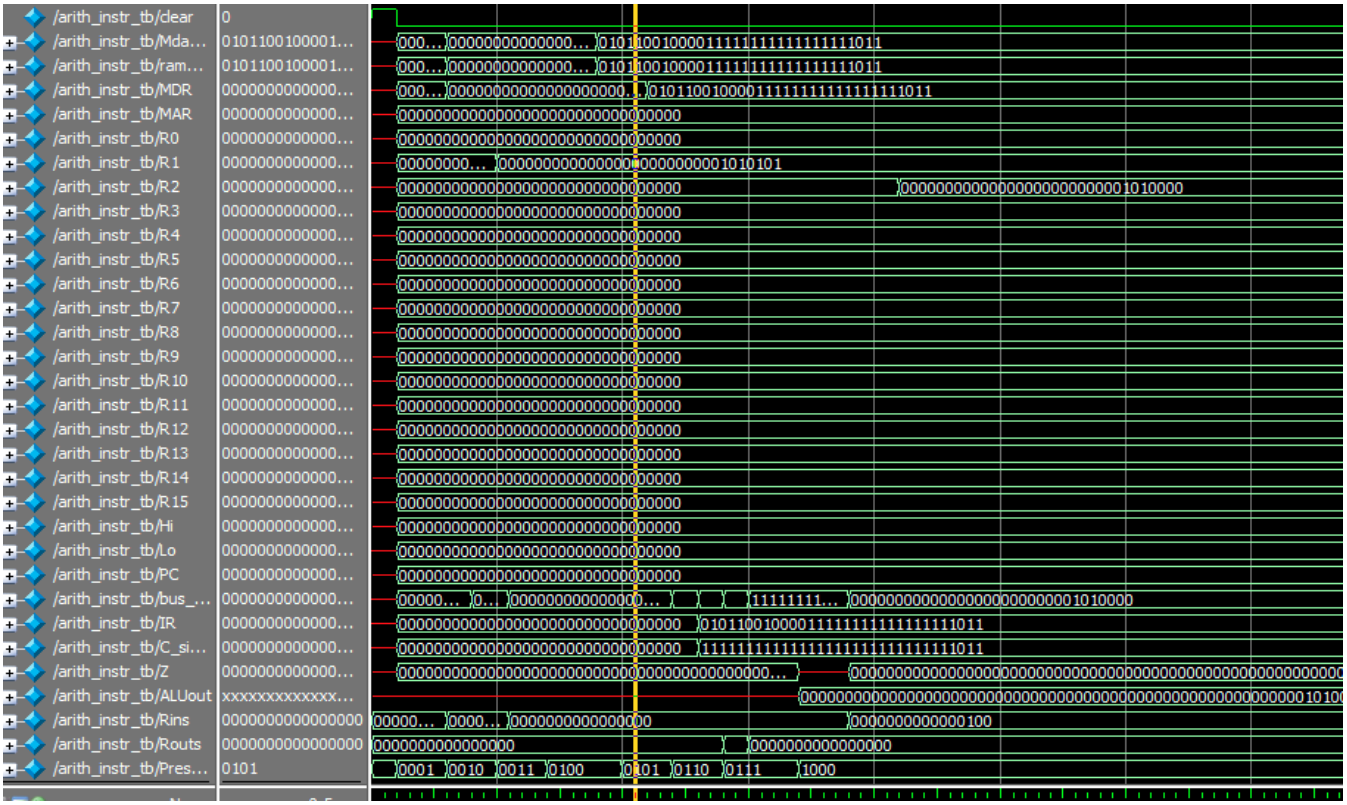
st $90(R1), R1


/st_instr_tb/PCout
/st_instr_tb/Zlowout
/st_instr_tb/MDRout
/st_instr_tb/Cout
/st_instr_tb/MARin
/st_instr_tb/Zin
/st_instr_tb/PCin
/st_instr_tb/MDRin
/st_instr_tb/IRin
/st_instr_tb/Yin
/st_instr_tb/Gra
/st_instr_tb/Grb
/st_instr_tb/Grc
/st_instr_tb/Rin
/st_instr_tb/Rout
/st_instr_tb/BAout
/st_instr_tb/IncPC
/st_instr_tb/Read
/st_instr_tb/Write
/st_instr_tb/ADD
/st_instr_tb/AND
/st_instr_tb/OR
/st_instr_tb/SUB
/st_instr_tb/SHR
/st_instr_tb/SHL
/st_instr_tb/ROL
/st_instr_tb/ROR
/st_instr_tb/NEG
/st_instr_tb/NOT
/st_instr_tb/Clock
/st_instr_tb/clear

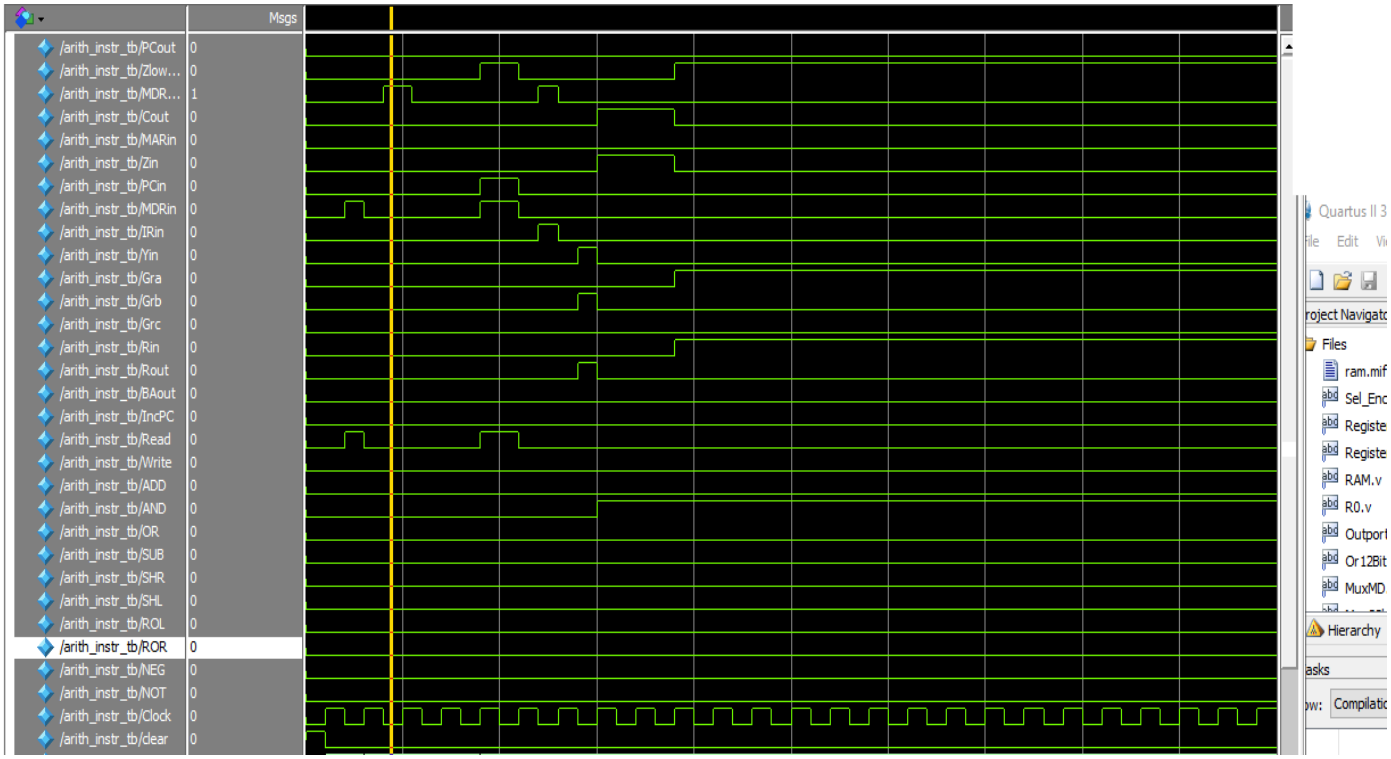

/st_instr_tb/Mdatain
/st_instr_tb/ram_data
/st_instr_tb/MDR
/st_instr_tb/MAR
/st_instr_tb/R0
/st_instr_tb/R1
/st_instr_tb/R2
/st_instr_tb/R3
/st_instr_tb/R4
/st_instr_tb/R5
/st_instr_tb/R6
/st_instr_tb/R7
/st_instr_tb/R8
/st_instr_tb/R9
/st_instr_tb/R10
/st_instr_tb/R11
/st_instr_tb/R12
/st_instr_tb/R13
/st_instr_tb/R14
/st_instr_tb/R15
/st_instr_tb/Hi
/st_instr_tb/Lo
/st_instr_tb/PC
/st_instr_tb/bus_m...
/st_instr_tb/IR
/st_instr_tb/C_sign...
/st_instr_tb/Z
/st_instr_tb/ALUout
/st_instr_tb/Rins
/st_instr_tb/Routs
/st_instr_tb/Presen...
Now
0.5 ns
0 ns
0.05 ns
0.1 ns
0.15 ns
0.2 ns
0.25 ns
0.3 ns
0.35 ns
0.4 ns
0.45 ns
Project Navigator
Hierarchy
Tasks
Flow:
Type

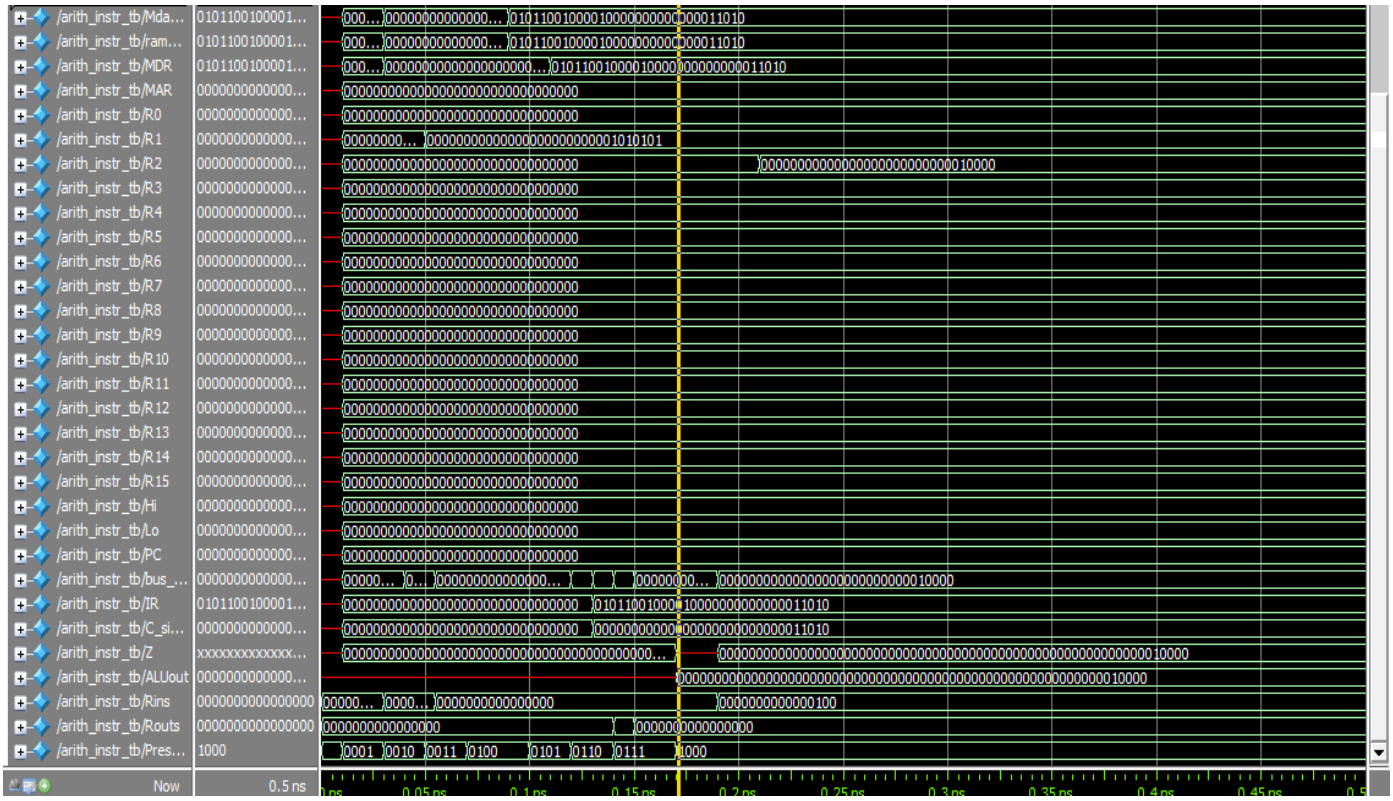
addi R2, R1, -5


Wave - Default
Msgs
/arith_instr_tb/PCout
/arith_instr_tb/Zlow...
/arith_instr_tb/MDR...
/arith_instr_tb/Cout
/arith_instr_tb/MARin
/arith_instr_tb/Zin
/arith_instr_tb/PCin
/arith_instr_tb/MDRin
/arith_instr_tb/IRin
/arith_instr_tb/Yin
/arith_instr_tb/Gra
/arith_instr_tb/Grb
/arith_instr_tb/Grc
/arith_instr_tb/Rin
/arith_instr_tb/Rout
/arith_instr_tb/BAout
/arith_instr_tb/IncPC
/arith_instr_tb/Read
/arith_instr_tb/Write
/arith_instr_tb/ADD
/arith_instr_tb/AND
/arith_instr_tb/OR
/arith_instr_tb/SUB
/arith_instr_tb/SHR
/arith_instr_tb/SHL
/arith_instr_tb/ROL
/arith_instr_tb/ROR
/arith_instr_tb/NEG
/arith_instr_tb/NOT
/arith_instr_tb/Clock

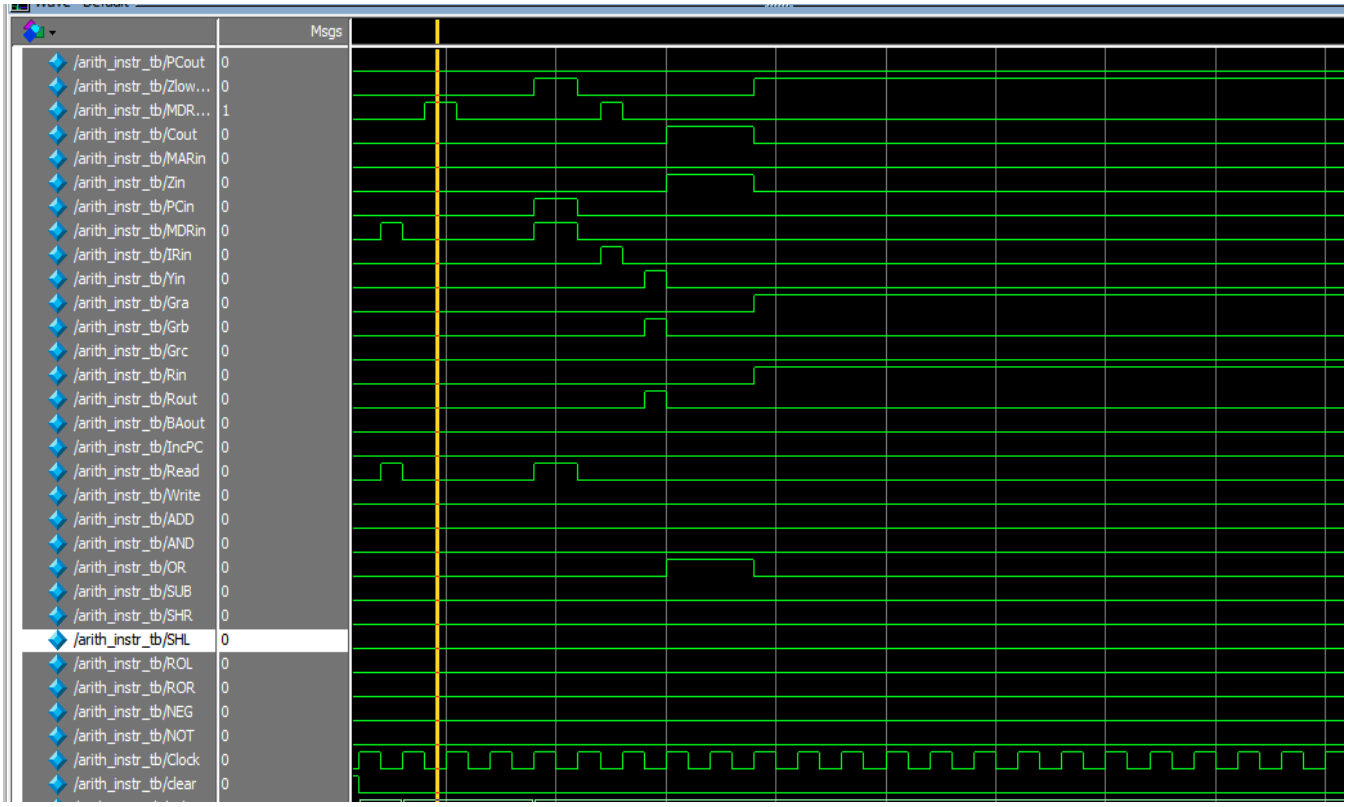

/arith_instr_tb/clear
/arith_instr_tb/Mda...
/arith_instr_tb/ram...
/arith_instr_tb/MDR
/arith_instr_tb/MAR
/arith_instr_tb/R0
/arith_instr_tb/R1
/arith_instr_tb/R2
/arith_instr_tb/R3
/arith_instr_tb/R4
/arith_instr_tb/R5
/arith_instr_tb/R6
/arith_instr_tb/R7
/arith_instr_tb/R8
/arith_instr_tb/R9
/arith_instr_tb/R10
/arith_instr_tb/R11
/arith_instr_tb/R12
/arith_instr_tb/R13
/arith_instr_tb/R14
/arith_instr_tb/R15
/arith_instr_tb/Hi
/arith_instr_tb/Lo
/arith_instr_tb/PC
/arith_instr_tb/bus_...
/arith_instr_tb/IR
/arith_instr_tb/C_si...
/arith_instr_tb/Z
/arith_instr_tb/ALUout
/arith_instr_tb/Rins
/arith_instr_tb/Routs
/arith_instr_tb/Pres...
0101
0001
0010
0011
0100
0101
0110
0111
1000

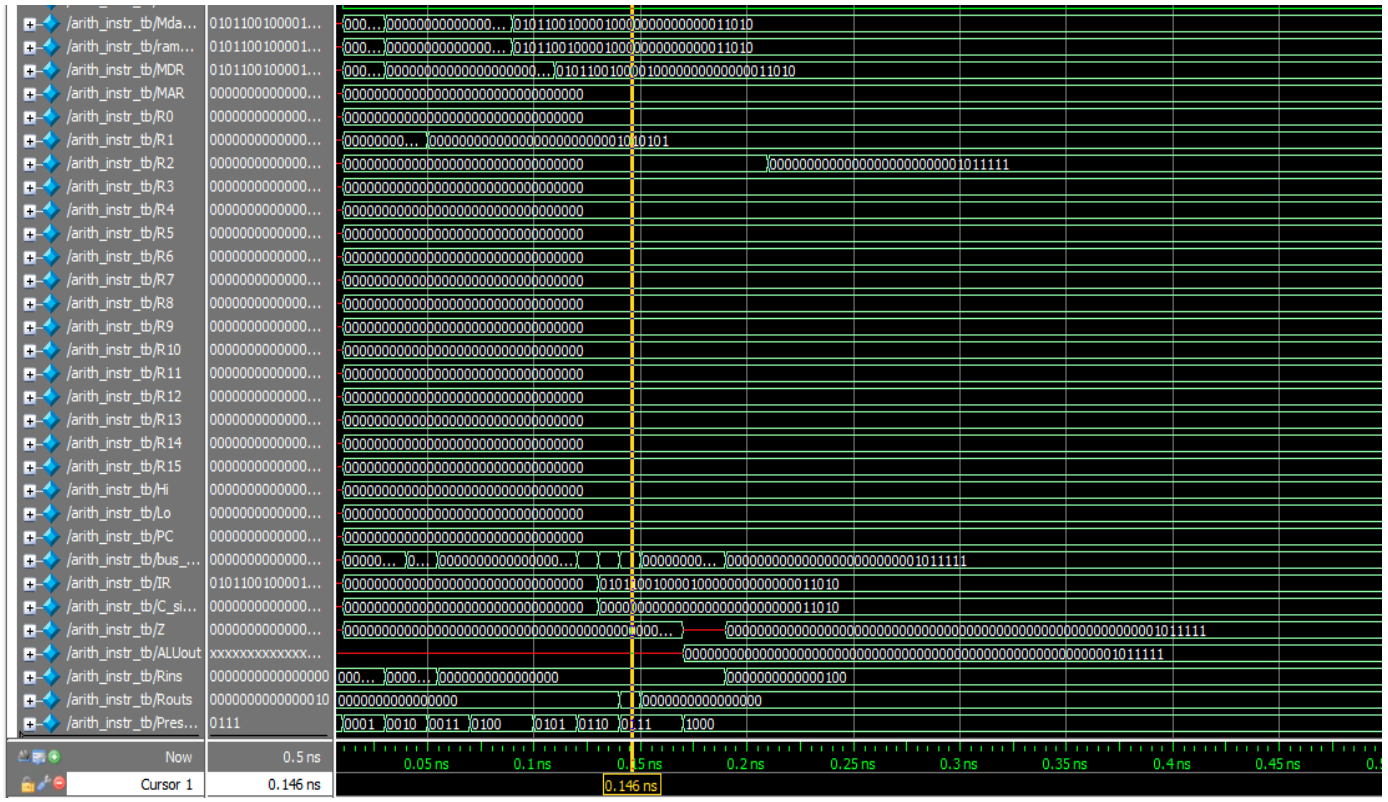
andi R2, R2, $26


Msgs
/arith_instr_tb/PCout
/arith_instr_tb/Zlow...
/arith_instr_tb/MDR...
/arith_instr_tb/Cout
/arith_instr_tb/MARin
/arith_instr_tb/Zin
/arith_instr_tb/PCin
/arith_instr_tb/MDRin
/arith_instr_tb/IRin
/arith_instr_tb/Yin
/arith_instr_tb/Gra
/arith_instr_tb/Grb
/arith_instr_tb/Grc
/arith_instr_tb/Rin
/arith_instr_tb/Rout
/arith_instr_tb/BAout
/arith_instr_tb/IncPC
/arith_instr_tb/Read
/arith_instr_tb/Write
/arith_instr_tb/ADD
/arith_instr_tb/AND
/arith_instr_tb/OR
/arith_instr_tb/SUB
/arith_instr_tb/SHR
/arith_instr_tb/SHL
/arith_instr_tb/ROL
/arith_instr_tb/ROR
/arith_instr_tb/NEG
/arith_instr_tb/NOT
/arith_instr_tb/Clock
/arith_instr_tb/clear
Quartus II 3
File Edit
Project Navigato
Files
ram.mif
Sel_Enc
Registe
Registe
RAM.v
R0.v
Outport
Or12Bit
MuxMD
Hierarchy
asks
ow: Compilatio


/arith_instr_tb/Mda...
/arith_instr_tb/ram...
/arith_instr_tb/MDR
/arith_instr_tb/MAR
/arith_instr_tb/R0
/arith_instr_tb/R1
/arith_instr_tb/R2
/arith_instr_tb/R3
/arith_instr_tb/R4
/arith_instr_tb/R5
/arith_instr_tb/R6
/arith_instr_tb/R7
/arith_instr_tb/R8
/arith_instr_tb/R9
/arith_instr_tb/R10
/arith_instr_tb/R11
/arith_instr_tb/R12
/arith_instr_tb/R13
/arith_instr_tb/R14
/arith_instr_tb/R15
/arith_instr_tb/Hi
/arith_instr_tb/Lo
/arith_instr_tb/PC
/arith_instr_tb/bus_...
/arith_instr_tb/IR
/arith_instr_tb/C_si...
/arith_instr_tb/Z
/arith_instr_tb/ALUout
/arith_instr_tb/Rins
/arith_instr_tb/Routs
/arith_instr_tb/Pres...
Now
0.5 ns

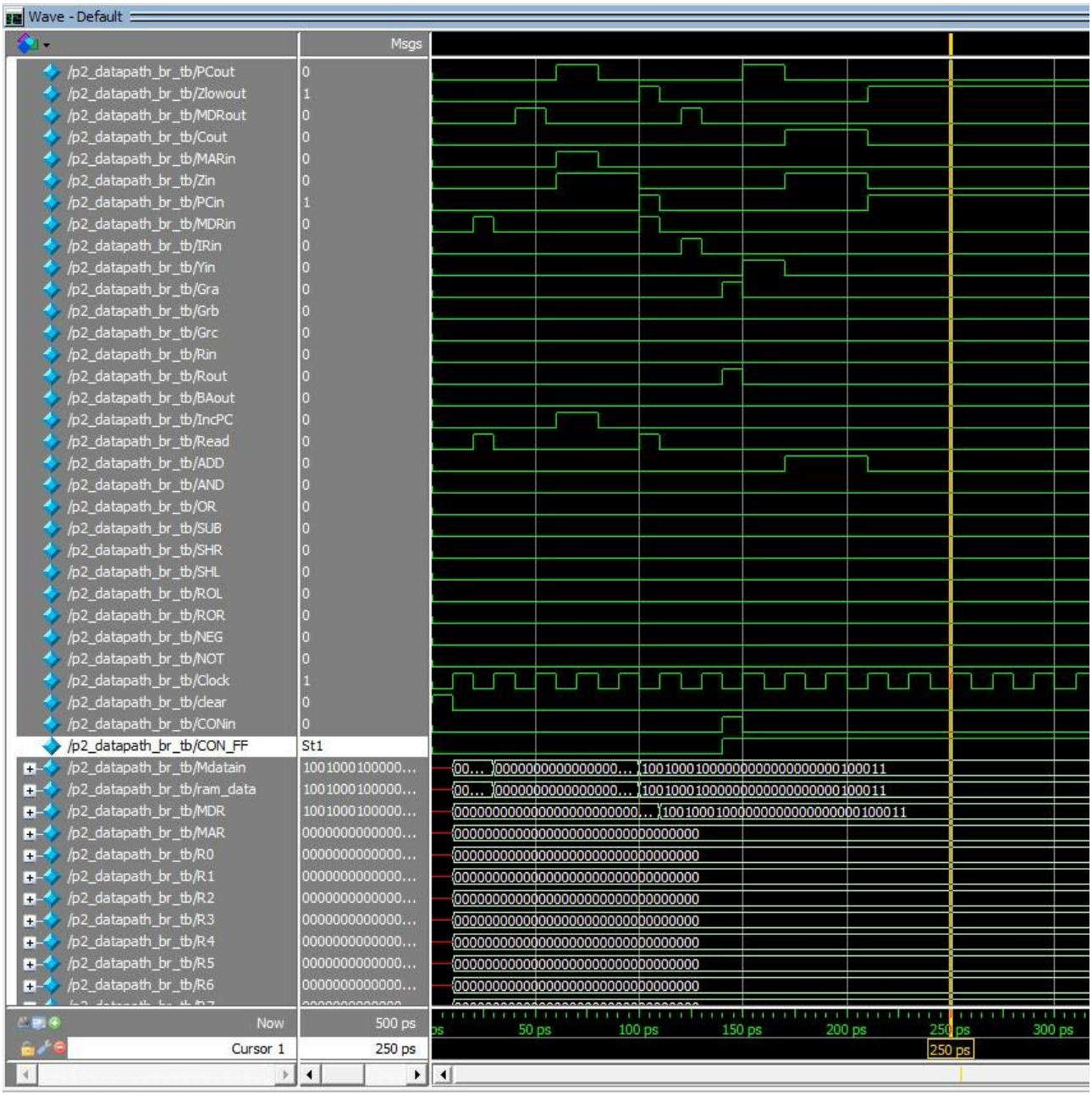
ori R2, R1, $26

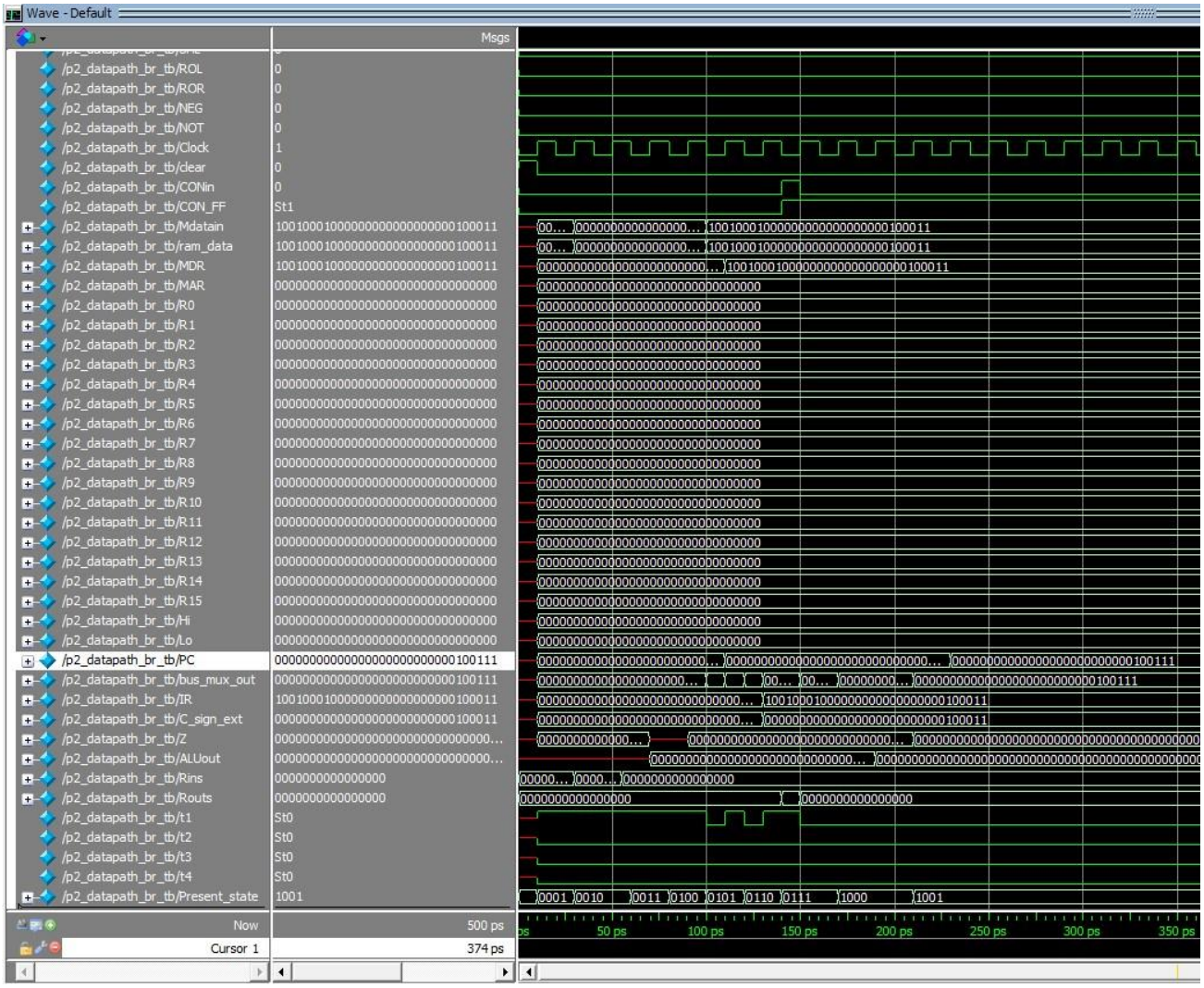

Msgs
/arith_instr_tb/PCout
/arith_instr_tb/Zlow...
/arith_instr_tb/MDR...
/arith_instr_tb/Cout
/arith_instr_tb/MARin
/arith_instr_tb/Zin
/arith_instr_tb/PCin
/arith_instr_tb/MDRin
/arith_instr_tb/IRin
/arith_instr_tb/Yin
/arith_instr_tb/Gra
/arith_instr_tb/Grb
/arith_instr_tb/Grc
/arith_instr_tb/Rin
/arith_instr_tb/Rout
/arith_instr_tb/BAout
/arith_instr_tb/IncPC
/arith_instr_tb/Read
/arith_instr_tb/Write
/arith_instr_tb/ADD
/arith_instr_tb/AND
/arith_instr_tb/OR
/arith_instr_tb/SUB
/arith_instr_tb/SHR
/arith_instr_tb/SHL
/arith_instr_tb/ROL
/arith_instr_tb/ROR
/arith_instr_tb/NEG
/arith_instr_tb/NOT
/arith_instr_tb/Clock
/arith_instr_tb/clear


/arith_instr_tb/Mda...
/arith_instr_tb/ram...
/arith_instr_tb/MDR
/arith_instr_tb/MAR
/arith_instr_tb/R0
/arith_instr_tb/R1
/arith_instr_tb/R2
/arith_instr_tb/R3
/arith_instr_tb/R4
/arith_instr_tb/R5
/arith_instr_tb/R6
/arith_instr_tb/R7
/arith_instr_tb/R8
/arith_instr_tb/R9
/arith_instr_tb/R10
/arith_instr_tb/R11
/arith_instr_tb/R12
/arith_instr_tb/R13
/arith_instr_tb/R14
/arith_instr_tb/R15
/arith_instr_tb/Hi
/arith_instr_tb/Lo
/arith_instr_tb/PC
/arith_instr_tb/bus_...
/arith_instr_tb/IR
/arith_instr_tb/C_si...
/arith_instr_tb/Z
/arith_instr_tb/ALUout
/arith_instr_tb/Rins
/arith_instr_tb/Routs
/arith_instr_tb/Pres...
Now
Cursor 1
0.5 ns
0.146 ns
0.05 ns
0.1 ns
0.15 ns
0.2 ns
0.25 ns
0.3 ns
0.35 ns
0.4 ns
0.45 ns

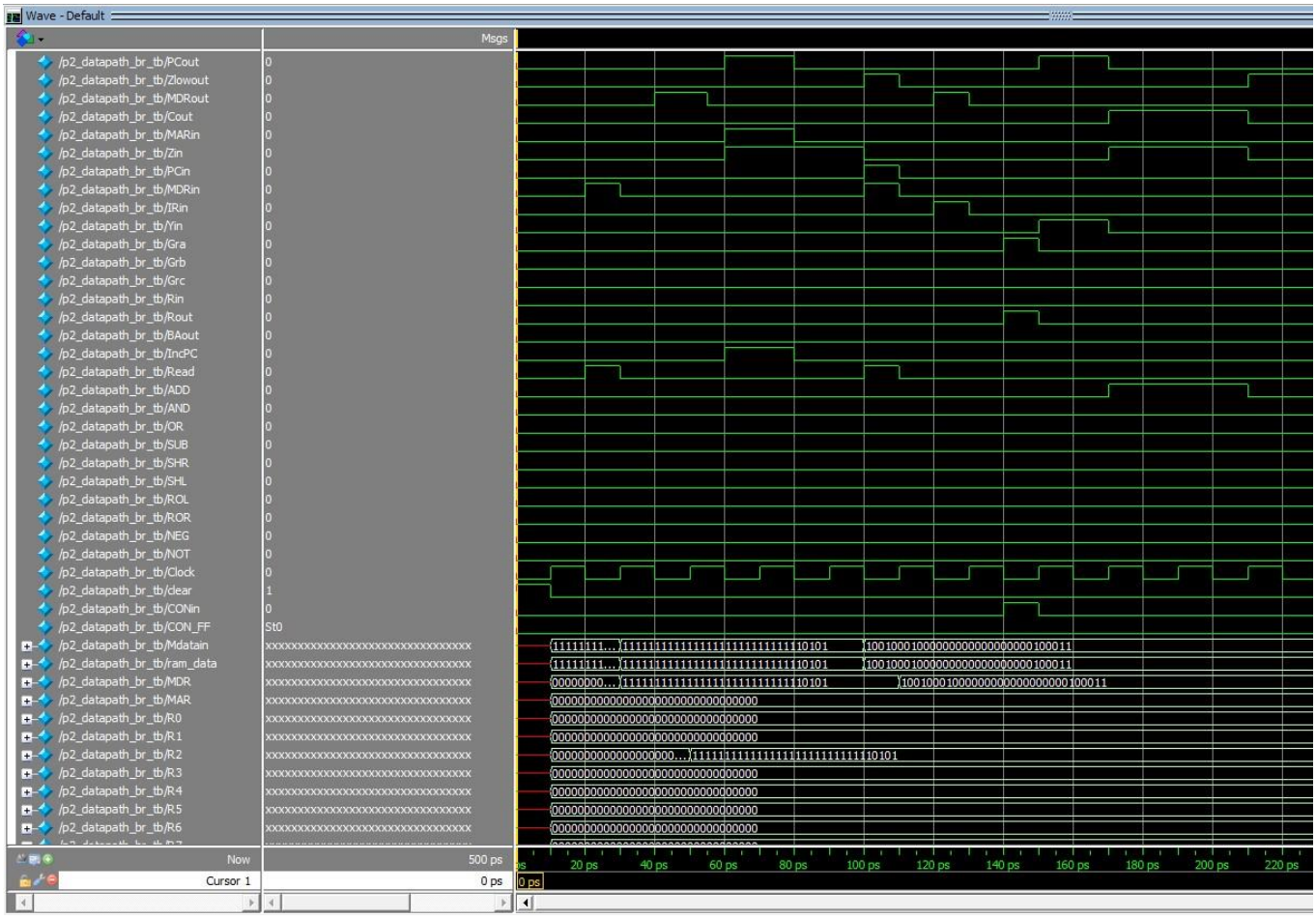
brzr R2, 35

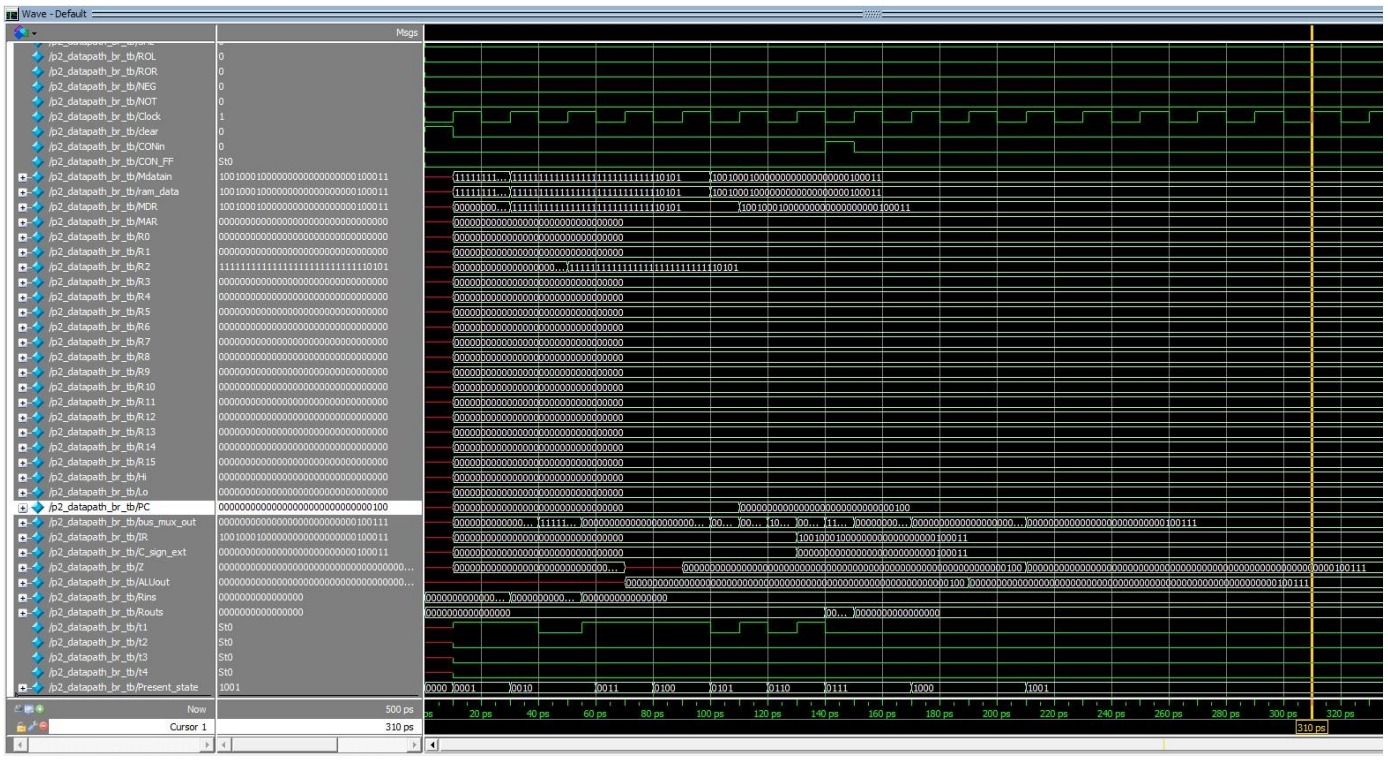
R2 = 0, so branching is expected:





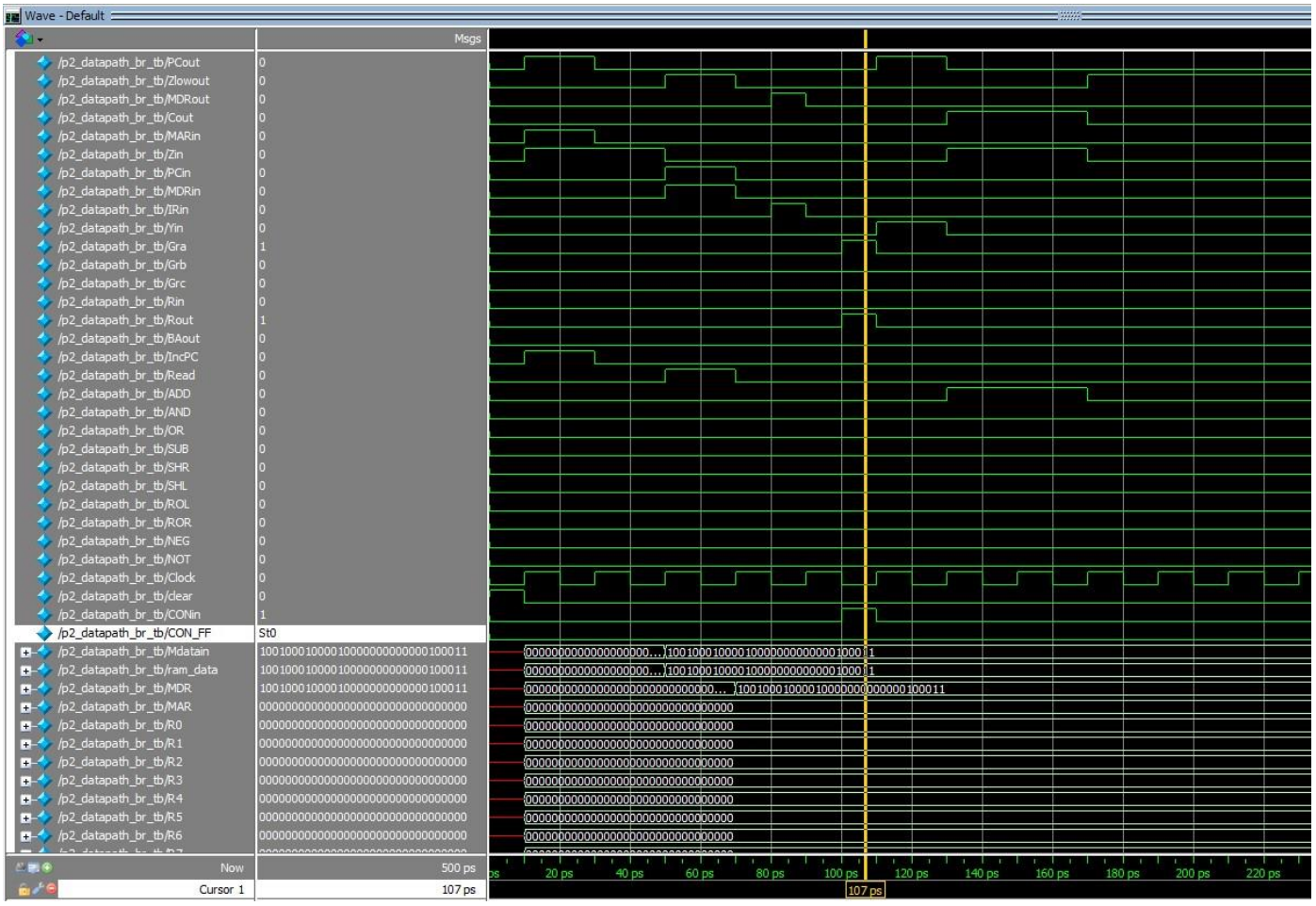
R2 = -10, so branching not expected:

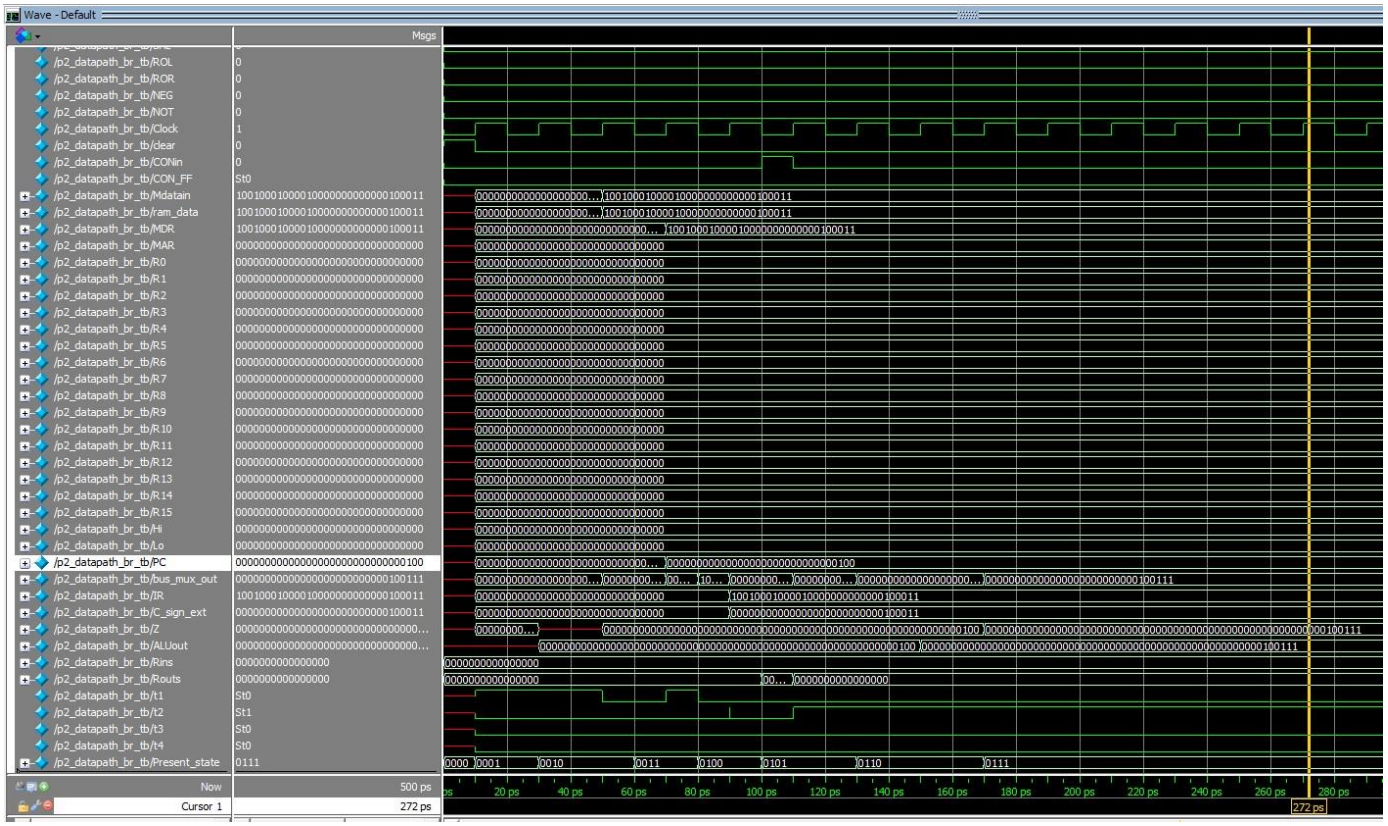

Wave - Default
Msgs
/p2_datapath_br_tb/PCout
/p2_datapath_br_tb/Zlowout
/p2_datapath_br_tb/MDRout
/p2_datapath_br_tb/Cout
/p2_datapath_br_tb/MARin
/p2_datapath_br_tb/Zin
/p2_datapath_br_tb/PCin
/p2_datapath_br_tb/MDRin
/p2_datapath_br_tb/IRin
/p2_datapath_br_tb/Yin
/p2_datapath_br_tb/Gra
/p2_datapath_br_tb/Grb
/p2_datapath_br_tb/Grc
/p2_datapath_br_tb/Rin
/p2_datapath_br_tb/Rout
/p2_datapath_br_tb/BAout
/p2_datapath_br_tb/IncPC
/p2_datapath_br_tb/Read
/p2_datapath_br_tb/ADD
/p2_datapath_br_tb/AND
/p2_datapath_br_tb/OR
/p2_datapath_br_tb/SUB
/p2_datapath_br_tb/SHR
/p2_datapath_br_tb/SHL
/p2_datapath_br_tb/ROL
/p2_datapath_br_tb/ROR
/p2_datapath_br_tb/NEG
/p2_datapath_br_tb/NOT
/p2_datapath_br_tb/Clock
/p2_datapath_br_tb/clear
/p2_datapath_br_tb/CONin
/p2_datapath_br_tb/CON_FF
/p2_datapath_br_tb/Mdatain
/p2_datapath_br_tb/ram_data
/p2_datapath_br_tb/MDR
/p2_datapath_br_tb/MAR
/p2_datapath_br_tb/R0
/p2_datapath_br_tb/R1
/p2_datapath_br_tb/R2
/p2_datapath_br_tb/R3
/p2_datapath_br_tb/R4
/p2_datapath_br_tb/R5
/p2_datapath_br_tb/R6
Now
500 ps
Cursor 1
0 ps
20 ps
40 ps
60 ps
80 ps
100 ps
120 ps
140 ps
160 ps
180 ps
200 ps
220 ps


Wave - Default
Msgs
/p2_datapath_br_tb/ROL
/p2_datapath_br_tb/ROR
/p2_datapath_br_tb/NEG
/p2_datapath_br_tb/NOT
/p2_datapath_br_tb/Clock
/p2_datapath_br_tb/clear
/p2_datapath_br_tb/CONin
/p2_datapath_br_tb/CON_FF
/p2_datapath_br_tb/Mdatain
/p2_datapath_br_tb/ram_data
/p2_datapath_br_tb/MDR
/p2_datapath_br_tb/MAR
/p2_datapath_br_tb/R0
/p2_datapath_br_tb/R1
/p2_datapath_br_tb/R2
/p2_datapath_br_tb/R3
/p2_datapath_br_tb/R4
/p2_datapath_br_tb/R5
/p2_datapath_br_tb/R6
/p2_datapath_br_tb/R7
/p2_datapath_br_tb/R8
/p2_datapath_br_tb/R9
/p2_datapath_br_tb/R10
/p2_datapath_br_tb/R11
/p2_datapath_br_tb/R12
/p2_datapath_br_tb/R13
/p2_datapath_br_tb/R14
/p2_datapath_br_tb/R15
/p2_datapath_br_tb/Hi
/p2_datapath_br_tb/Lo
/p2_datapath_br_tb/PC
/p2_datapath_br_tb/bus_mux_out
/p2_datapath_br_tb/IR
/p2_datapath_br_tb/C_sign_ext
/p2_datapath_br_tb/Z
/p2_datapath_br_tb/ALUout
/p2_datapath_br_tb/Rins
/p2_datapath_br_tb/Routs
/p2_datapath_br_tb/t1
/p2_datapath_br_tb/t2
/p2_datapath_br_tb/t3
/p2_datapath_br_tb/t4
/p2_datapath_br_tb/Present_state
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
Now
500 ps
Cursor 1
310 ps
20 ps
40 ps
60 ps
80 ps
100 ps
120 ps
140 ps
160 ps
180 ps
200 ps
220 ps
240 ps
260 ps
280 ps
300 ps
320 ps

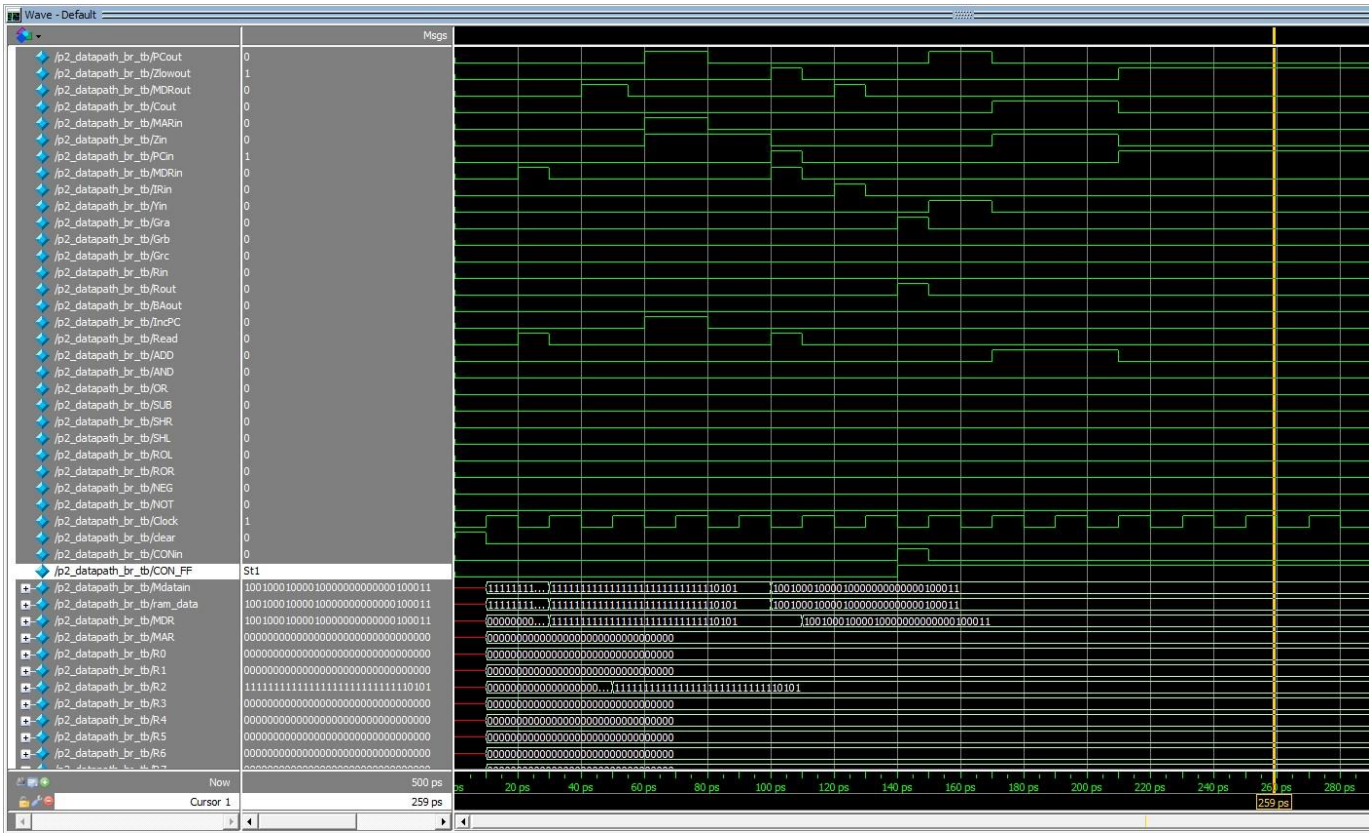
brnz R2, 35

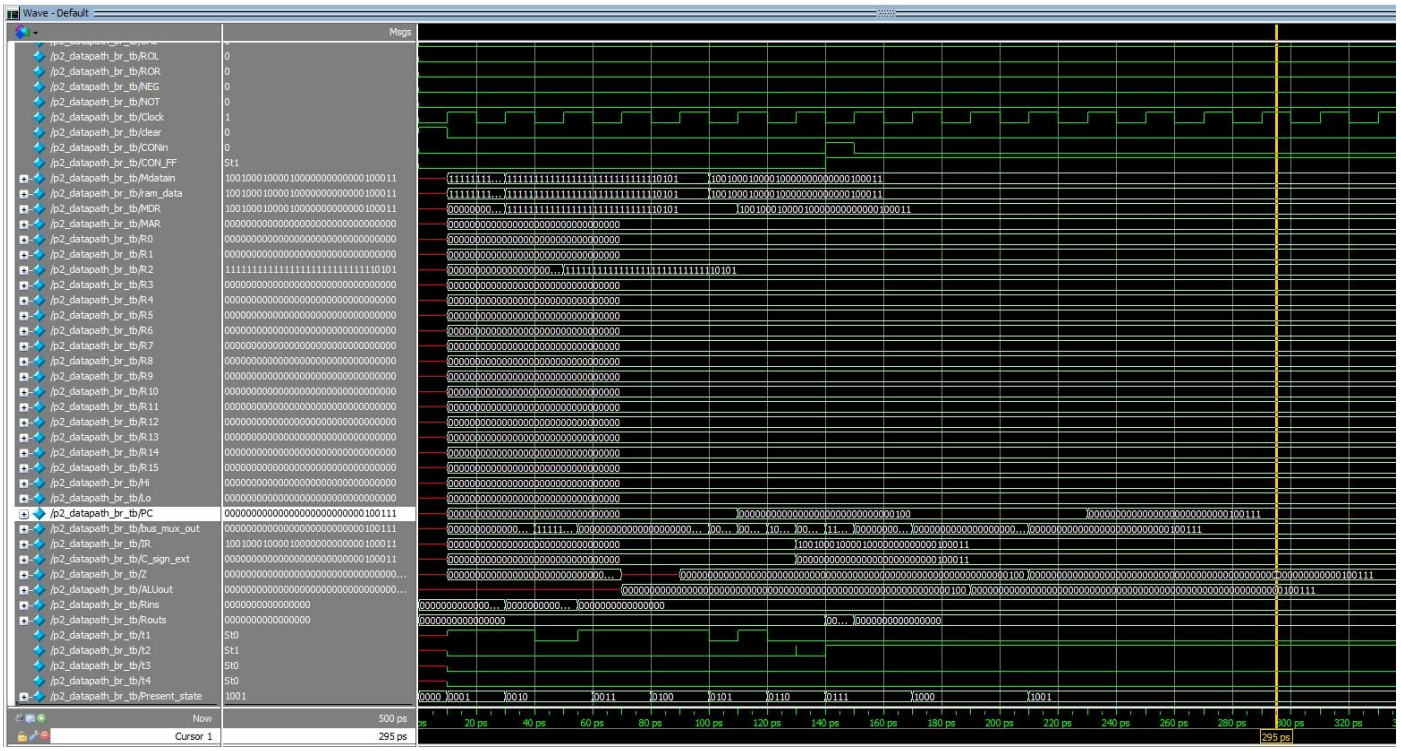
R2 = 0, so branching not expected:





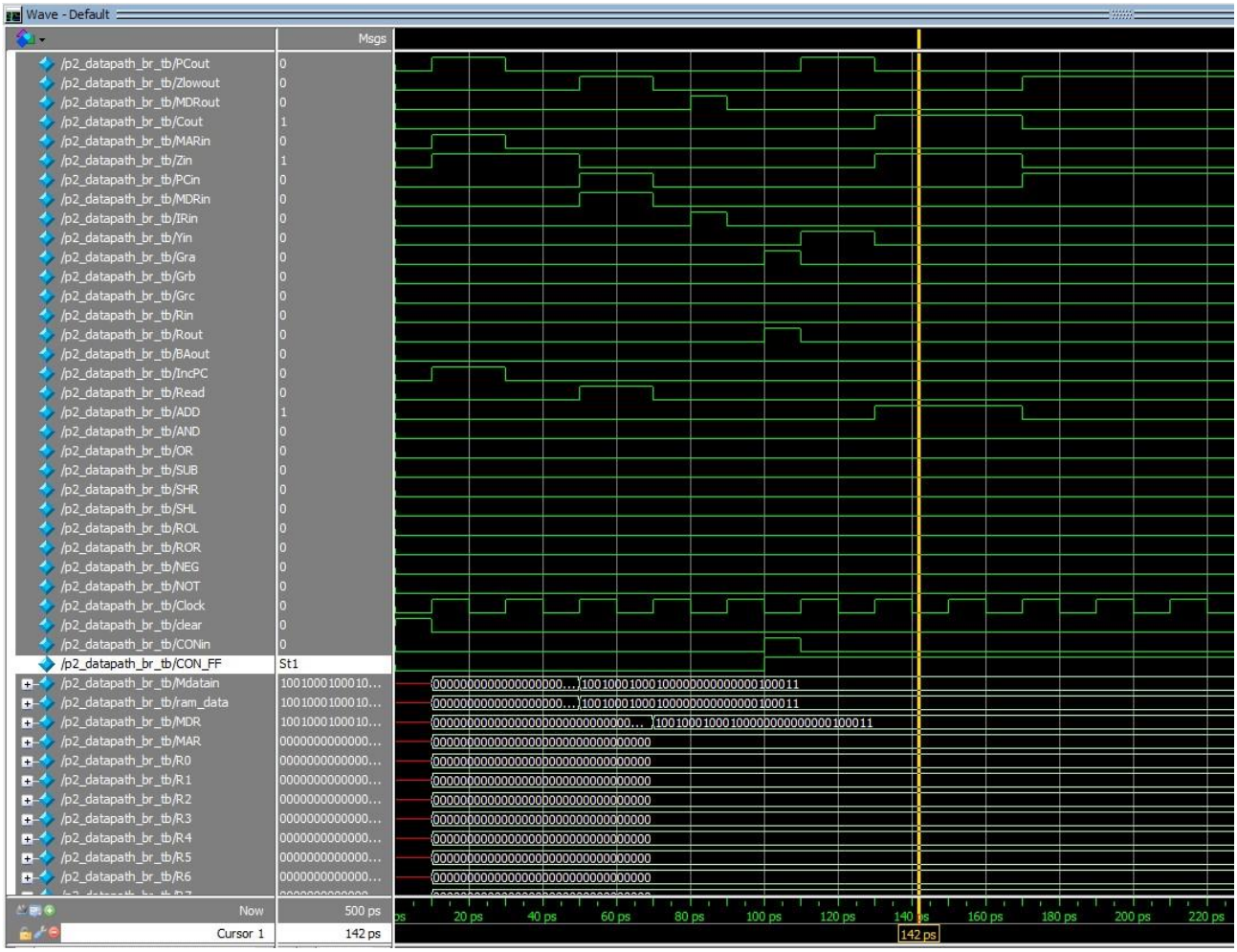
R2 = -10, so branching expected:

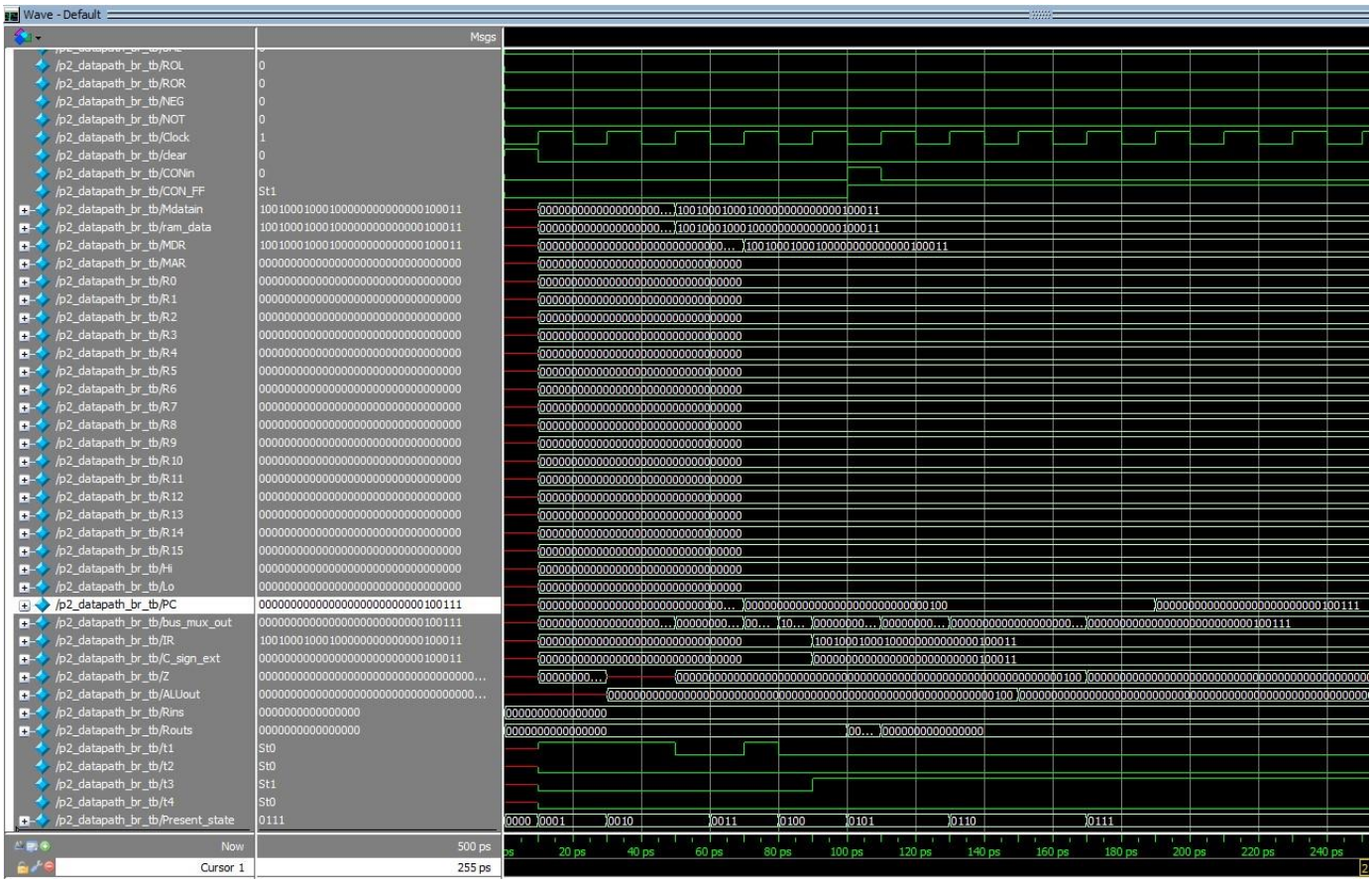

Wave - Default
Msgs
/p2_datapath_br_tb/PCout
/p2_datapath_br_tb/Zlowout
/p2_datapath_br_tb/MDRout
/p2_datapath_br_tb/Cout
/p2_datapath_br_tb/MARin
/p2_datapath_br_tb/Zin
/p2_datapath_br_tb/PCin
/p2_datapath_br_tb/MDRin
/p2_datapath_br_tb/IRin
/p2_datapath_br_tb/Yin
/p2_datapath_br_tb/Gra
/p2_datapath_br_tb/Grb
/p2_datapath_br_tb/Grc
/p2_datapath_br_tb/Rin
/p2_datapath_br_tb/Rout
/p2_datapath_br_tb/BAout
/p2_datapath_br_tb/IncPC
/p2_datapath_br_tb/Read
/p2_datapath_br_tb/ADD
/p2_datapath_br_tb/AND
/p2_datapath_br_tb/OR
/p2_datapath_br_tb/SUB
/p2_datapath_br_tb/SHR
/p2_datapath_br_tb/SHL
/p2_datapath_br_tb/ROL
/p2_datapath_br_tb/ROR
/p2_datapath_br_tb/NEG
/p2_datapath_br_tb/NOT
/p2_datapath_br_tb/Clock
/p2_datapath_br_tb/clear
/p2_datapath_br_tb/CONin
/p2_datapath_br_tb/CON_FF
St1
/p2_datapath_br_tb/Mdatain
/p2_datapath_br_tb/ram_data
/p2_datapath_br_tb/MDR
/p2_datapath_br_tb/MAR
/p2_datapath_br_tb/R0
/p2_datapath_br_tb/R1
/p2_datapath_br_tb/R2
/p2_datapath_br_tb/R3
/p2_datapath_br_tb/R4
/p2_datapath_br_tb/R5
/p2_datapath_br_tb/R6
Now
500 ps
Cursor 1
259 ps
20 ps
40 ps
60 ps
80 ps
100 ps
120 ps
140 ps
160 ps
180 ps
200 ps
220 ps
240 ps
260 ps
280 ps


Wave - Default
Msgs
/p2_datapath_br_tb/ROL
/p2_datapath_br_tb/ROR
/p2_datapath_br_tb/NEG
/p2_datapath_br_tb/NOT
/p2_datapath_br_tb/Clock
/p2_datapath_br_tb/clear
/p2_datapath_br_tb/CONin
/p2_datapath_br_tb/CON_FF
St1
/p2_datapath_br_tb/Mdatain
/p2_datapath_br_tb/ram_data
/p2_datapath_br_tb/MDR
/p2_datapath_br_tb/MAR
/p2_datapath_br_tb/R0
/p2_datapath_br_tb/R1
/p2_datapath_br_tb/R2
/p2_datapath_br_tb/R3
/p2_datapath_br_tb/R4
/p2_datapath_br_tb/R5
/p2_datapath_br_tb/R6
/p2_datapath_br_tb/R7
/p2_datapath_br_tb/R8
/p2_datapath_br_tb/R9
/p2_datapath_br_tb/R10
/p2_datapath_br_tb/R11
/p2_datapath_br_tb/R12
/p2_datapath_br_tb/R13
/p2_datapath_br_tb/R14
/p2_datapath_br_tb/R15
/p2_datapath_br_tb/Hi
/p2_datapath_br_tb/Lo
/p2_datapath_br_tb/PC
/p2_datapath_br_tb/bus_mux_out
/p2_datapath_br_tb/IR
/p2_datapath_br_tb/C_sign_ext
/p2_datapath_br_tb/Z
/p2_datapath_br_tb/ALUout
/p2_datapath_br_tb/Rins
/p2_datapath_br_tb/Routs
/p2_datapath_br_tb/t1
St0
/p2_datapath_br_tb/t2
St1
/p2_datapath_br_tb/t3
St0
/p2_datapath_br_tb/t4
St0
/p2_datapath_br_tb/Present_state
1001
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
Now
500 ps
Cursor 1
295 ps
20 ps
40 ps
60 ps
80 ps
100 ps
120 ps
140 ps
160 ps
180 ps
200 ps
220 ps
240 ps
260 ps
280 ps
300 ps
320 ps

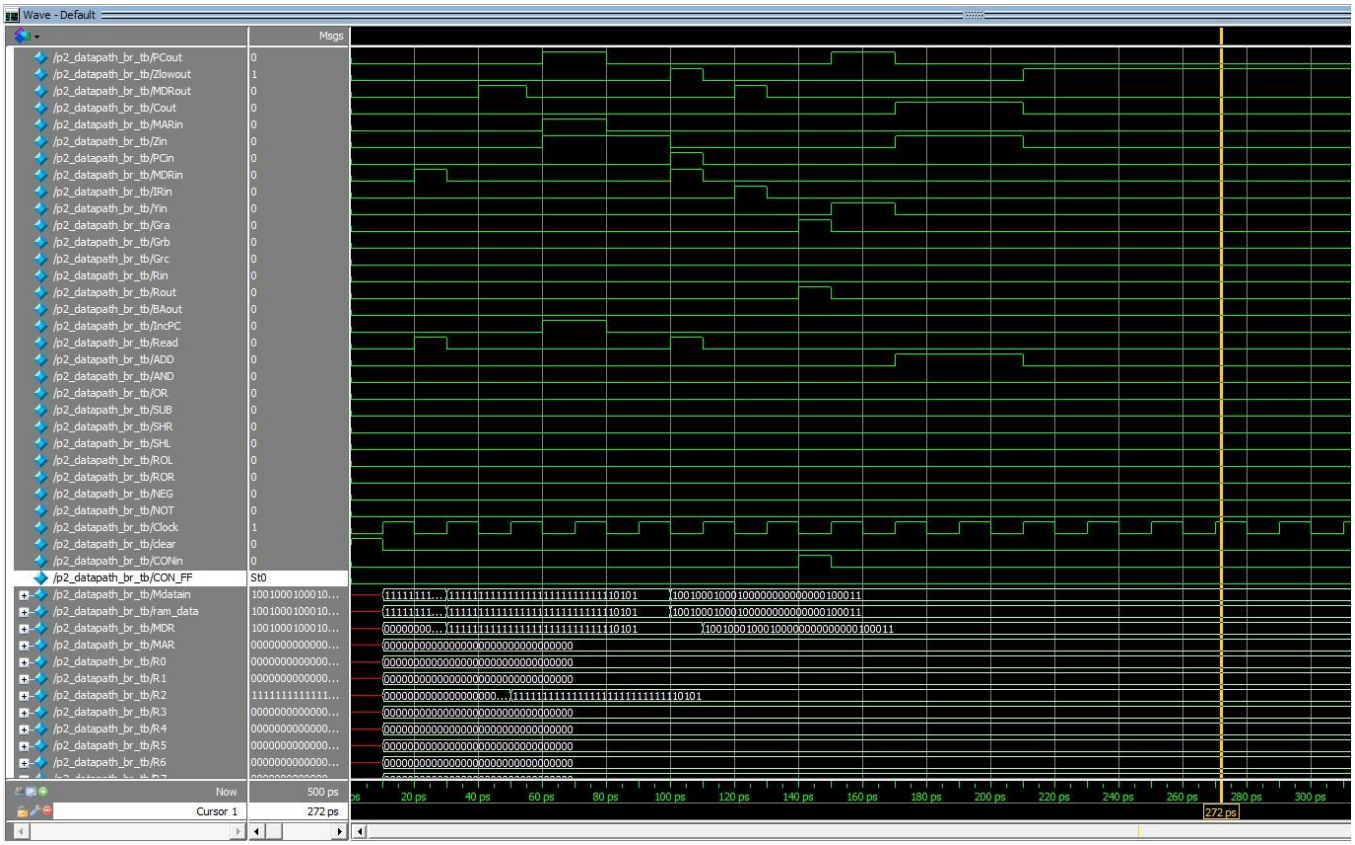
brpl R2, 35

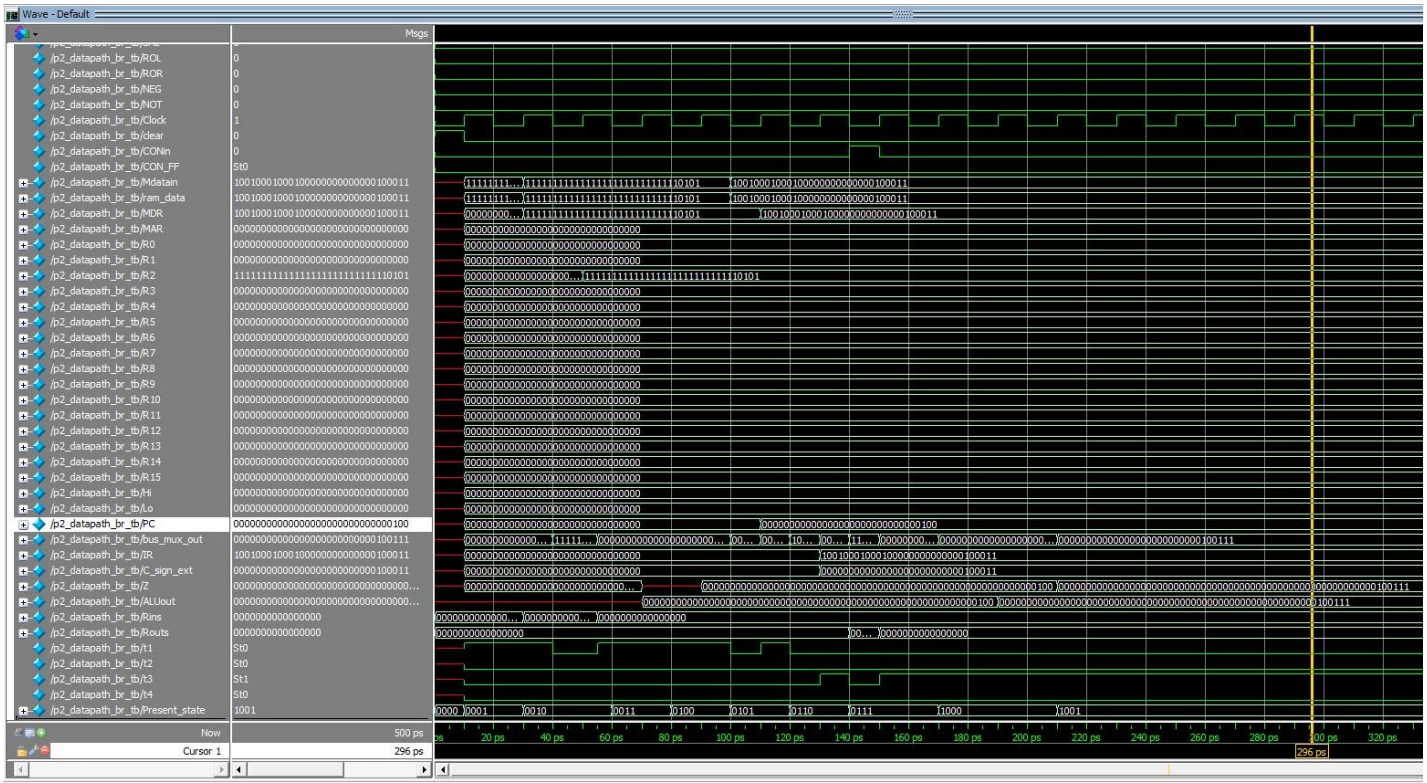
R2 = 0, so branching expected:





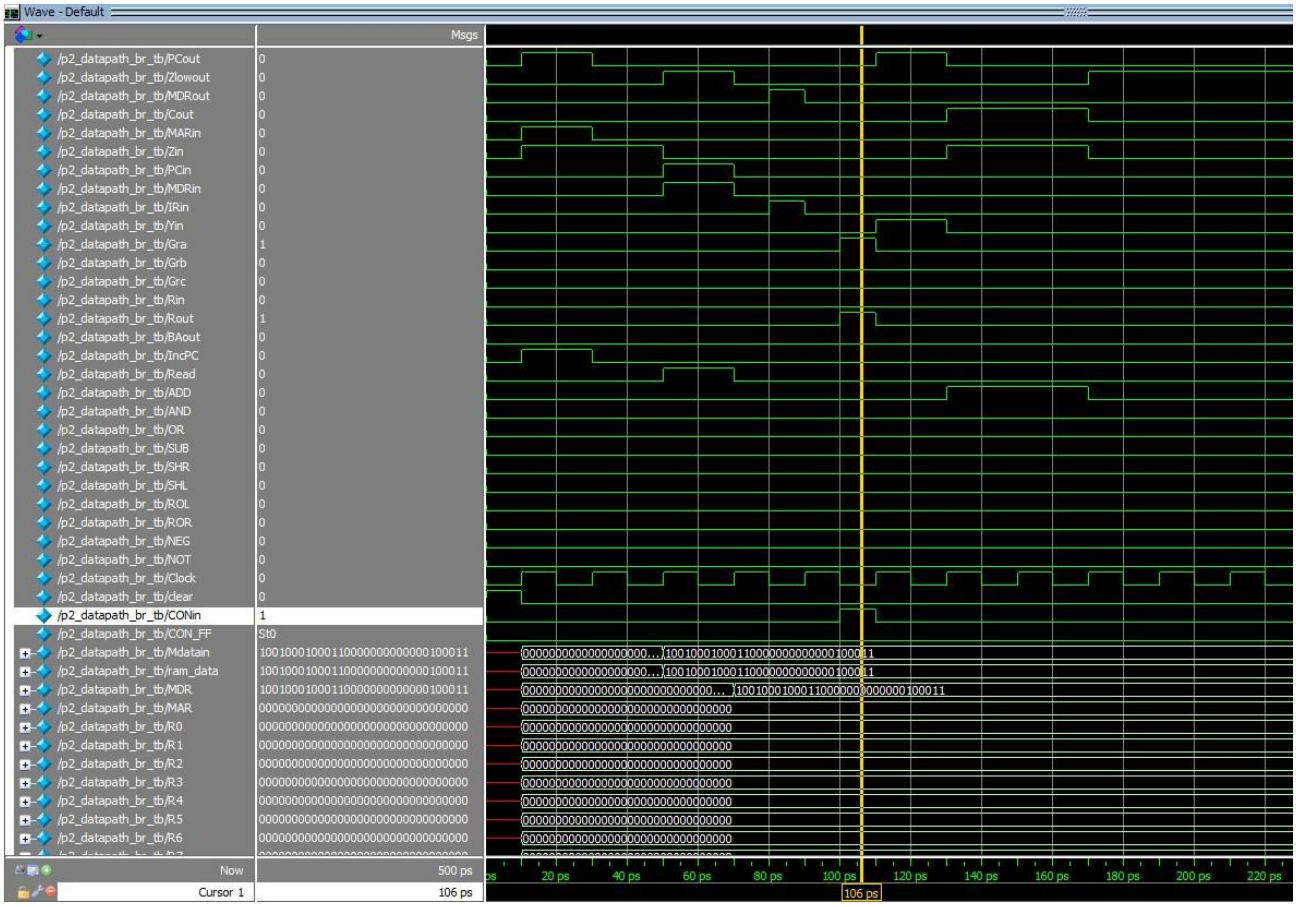
R2 = -10, so branching not expected:

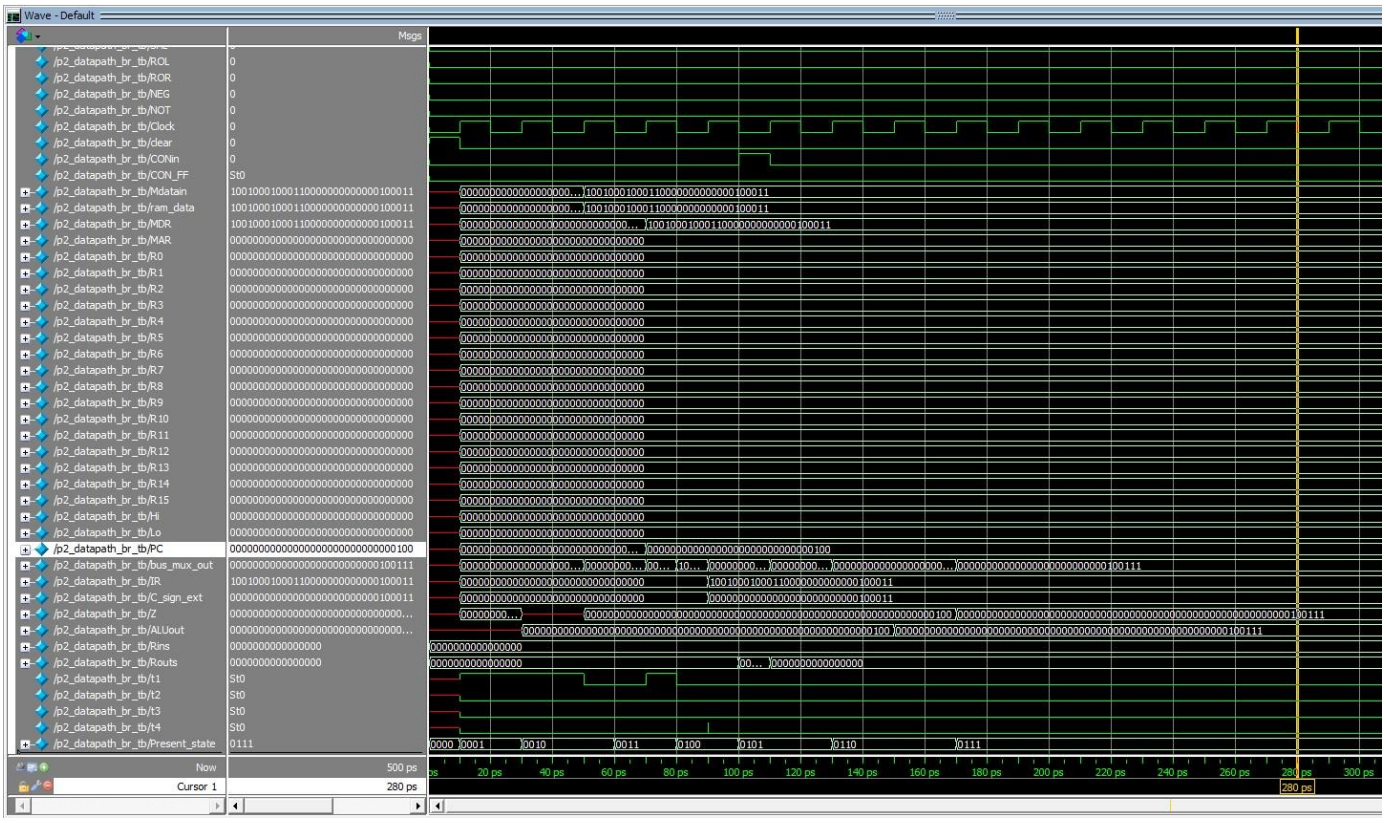

Wave - Default
Msgs
/p2_datapath_br_tb/PCout
/p2_datapath_br_tb/Zlowout
/p2_datapath_br_tb/MDRout
/p2_datapath_br_tb/Cout
/p2_datapath_br_tb/MARin
/p2_datapath_br_tb/Zin
/p2_datapath_br_tb/PCin
/p2_datapath_br_tb/MDRin
/p2_datapath_br_tb/IRin
/p2_datapath_br_tb/Yin
/p2_datapath_br_tb/Gra
/p2_datapath_br_tb/Grb
/p2_datapath_br_tb/Grc
/p2_datapath_br_tb/Rin
/p2_datapath_br_tb/Rout
/p2_datapath_br_tb/BAout
/p2_datapath_br_tb/IncPC
/p2_datapath_br_tb/Read
/p2_datapath_br_tb/ADD
/p2_datapath_br_tb/AND
/p2_datapath_br_tb/OR
/p2_datapath_br_tb/SUB
/p2_datapath_br_tb/SHR
/p2_datapath_br_tb/SHL
/p2_datapath_br_tb/ROL
/p2_datapath_br_tb/ROR
/p2_datapath_br_tb/NEG
/p2_datapath_br_tb/NOT
/p2_datapath_br_tb/Clock
/p2_datapath_br_tb/clear
/p2_datapath_br_tb/CONin
/p2_datapath_br_tb/CON_FF
/p2_datapath_br_tb/Mdatain
/p2_datapath_br_tb/ram_data
/p2_datapath_br_tb/MDR
/p2_datapath_br_tb/MAR
/p2_datapath_br_tb/R0
/p2_datapath_br_tb/R1
/p2_datapath_br_tb/R2
/p2_datapath_br_tb/R3
/p2_datapath_br_tb/R4
/p2_datapath_br_tb/R5
/p2_datapath_br_tb/R6
Now
Cursor 1
272 ps


Wave - Default
Msgs
/p2_datapath_br_tb/ROL
/p2_datapath_br_tb/ROR
/p2_datapath_br_tb/NEG
/p2_datapath_br_tb/NOT
/p2_datapath_br_tb/Clock
/p2_datapath_br_tb/clear
/p2_datapath_br_tb/CONin
/p2_datapath_br_tb/CON_FF
/p2_datapath_br_tb/Mdatain
/p2_datapath_br_tb/ram_data
/p2_datapath_br_tb/MDR
/p2_datapath_br_tb/MAR
/p2_datapath_br_tb/R0
/p2_datapath_br_tb/R1
/p2_datapath_br_tb/R2
/p2_datapath_br_tb/R3
/p2_datapath_br_tb/R4
/p2_datapath_br_tb/R5
/p2_datapath_br_tb/R6
/p2_datapath_br_tb/R7
/p2_datapath_br_tb/R8
/p2_datapath_br_tb/R9
/p2_datapath_br_tb/R10
/p2_datapath_br_tb/R11
/p2_datapath_br_tb/R12
/p2_datapath_br_tb/R13
/p2_datapath_br_tb/R14
/p2_datapath_br_tb/R15
/p2_datapath_br_tb/Hi
/p2_datapath_br_tb/Lo
/p2_datapath_br_tb/PC
/p2_datapath_br_tb/bus_mux_out
/p2_datapath_br_tb/IR
/p2_datapath_br_tb/C_sign_ext
/p2_datapath_br_tb/Z
/p2_datapath_br_tb/ALUout
/p2_datapath_br_tb/Rins
/p2_datapath_br_tb/Routs
/p2_datapath_br_tb/t1
/p2_datapath_br_tb/t2
/p2_datapath_br_tb/t3
/p2_datapath_br_tb/t4
/p2_datapath_br_tb/Present_state
Now
Cursor 1
296 ps

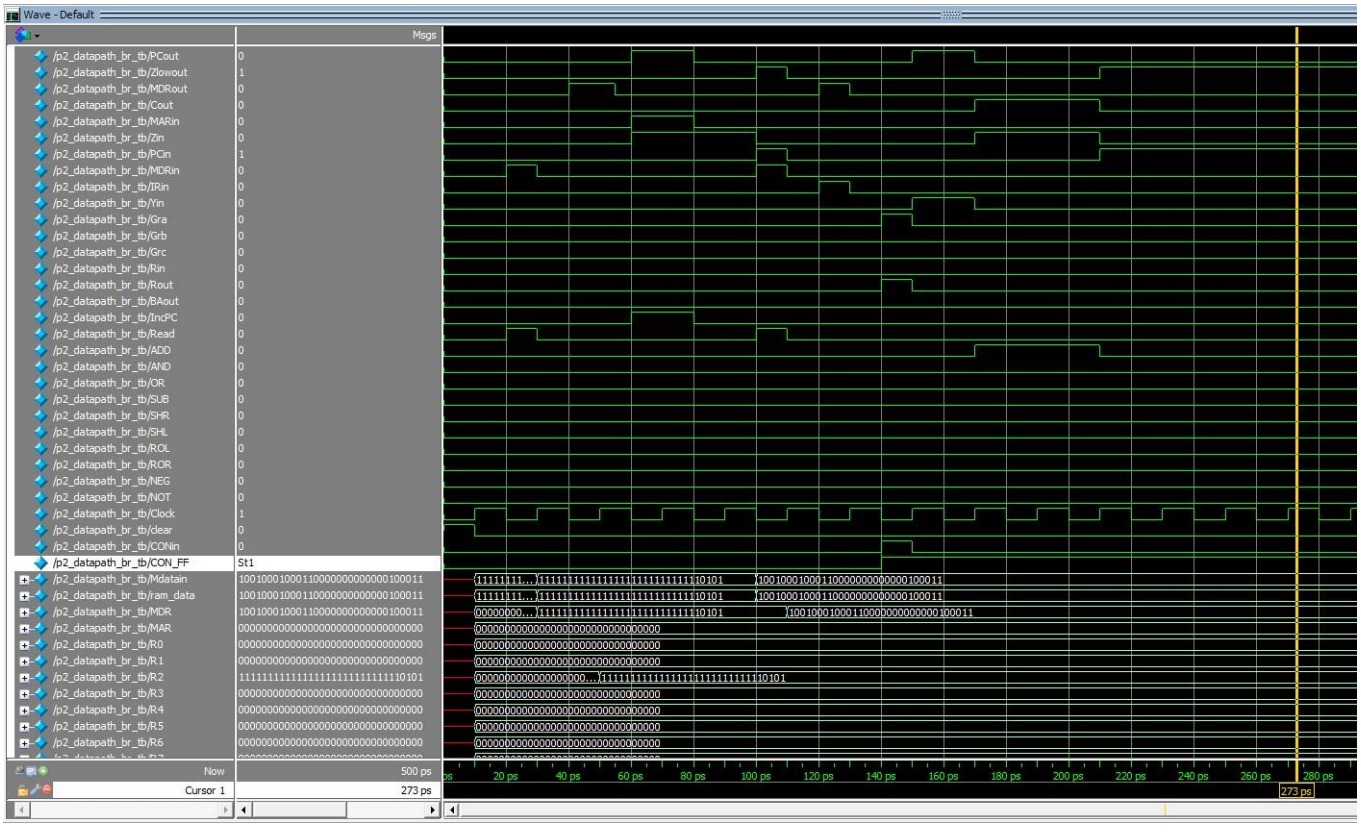
brmi R2, 35

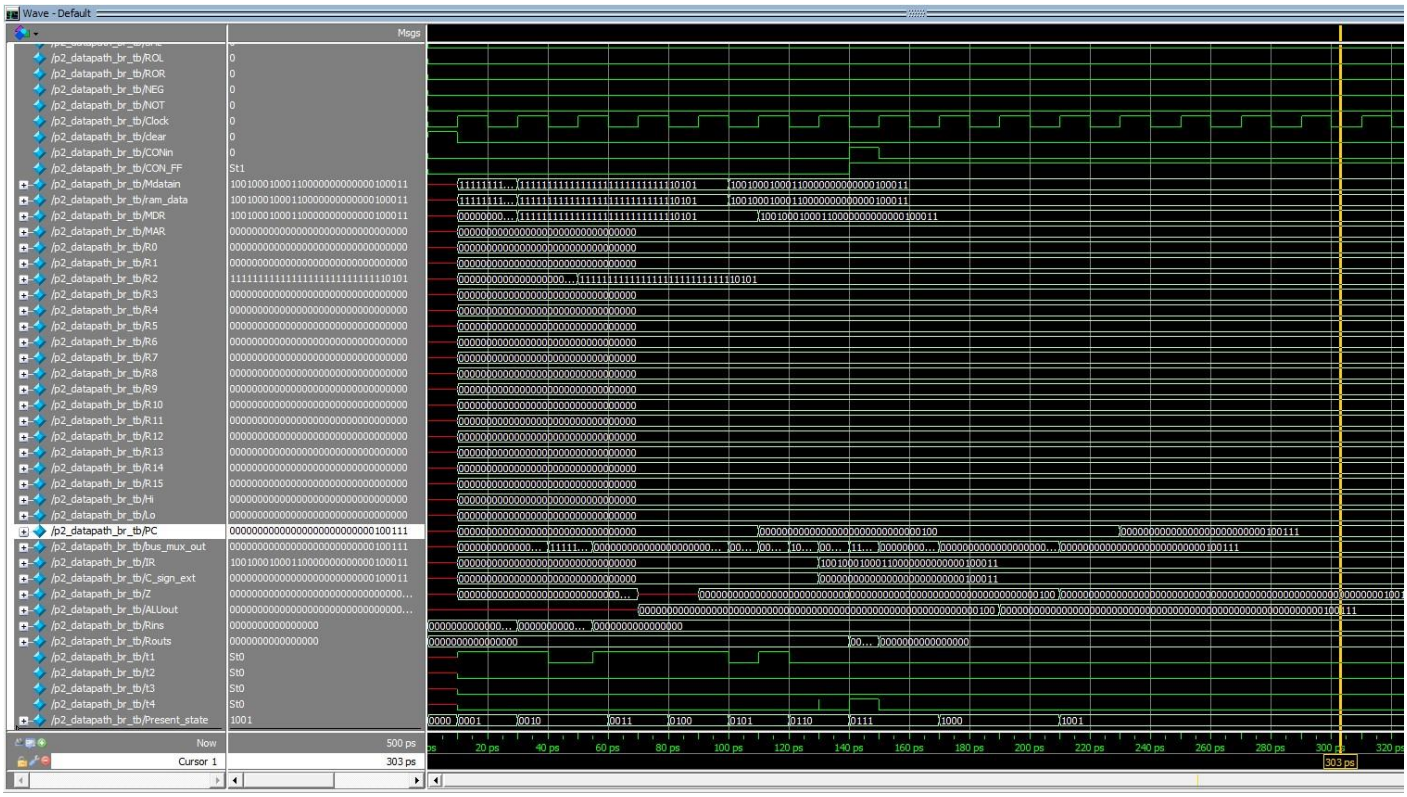
R2 = 0, so branching not expected:



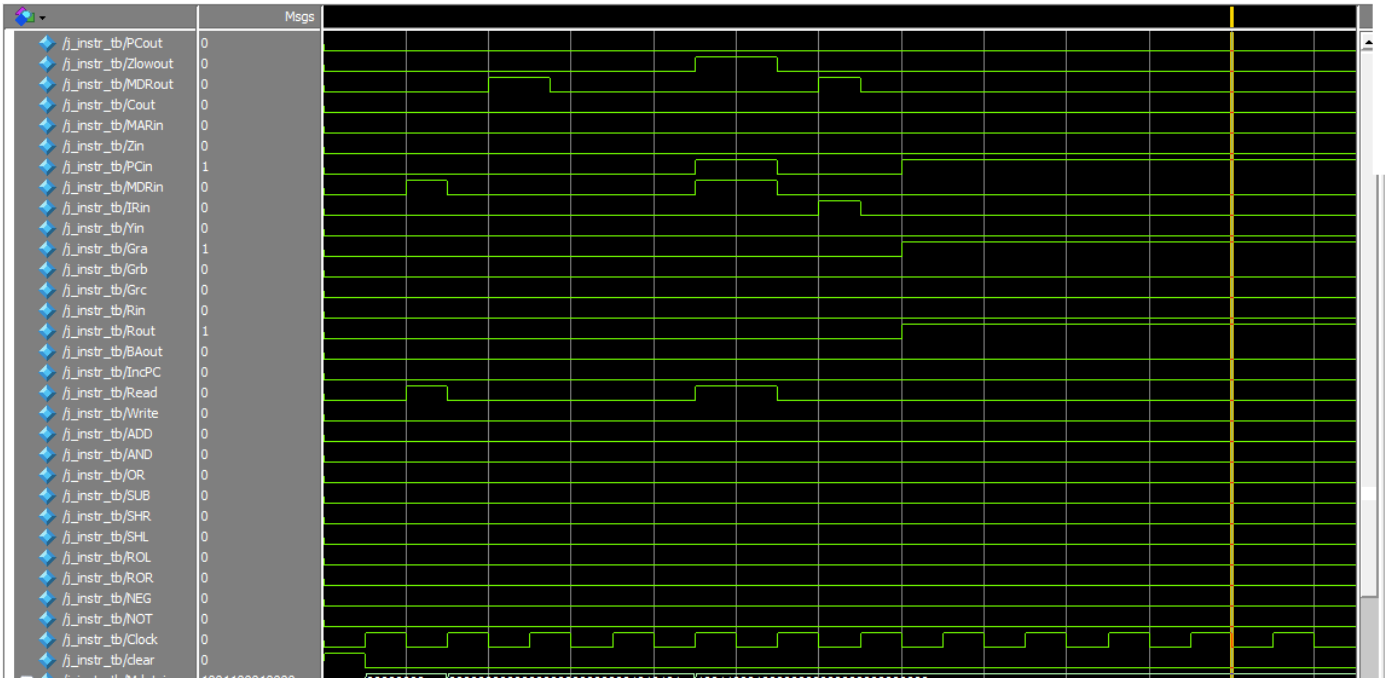


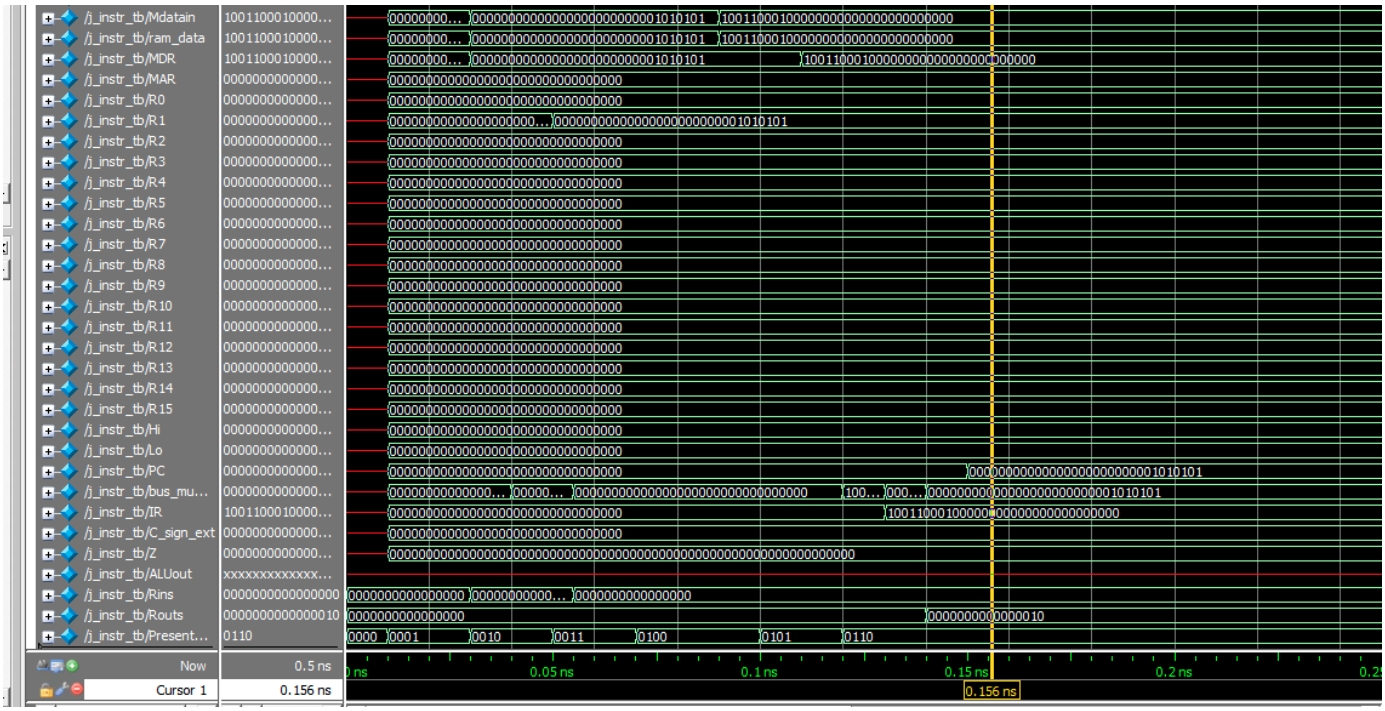
R2 = -10, so branching expected:



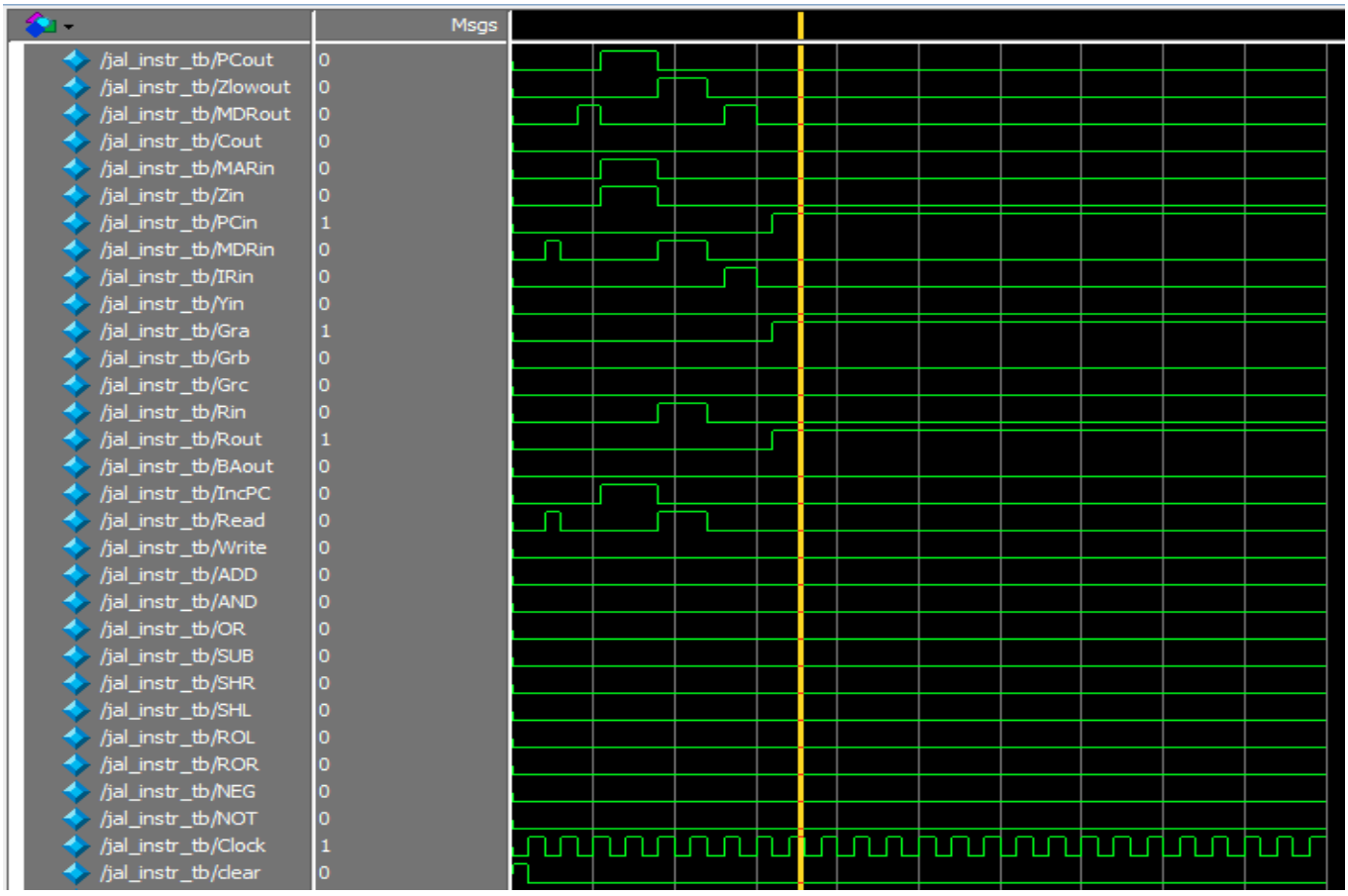

Wave - Default
Msgs
/p2_datapath_br_tb/ROL
/p2_datapath_br_tb/ROR
/p2_datapath_br_tb/NEG
/p2_datapath_br_tb/NOT
/p2_datapath_br_tb/Clock
/p2_datapath_br_tb/clear
/p2_datapath_br_tb/CONin
/p2_datapath_br_tb/CON_FF
/p2_datapath_br_tb/Mdatain
/p2_datapath_br_tb/ram_data
/p2_datapath_br_tb/MDR
/p2_datapath_br_tb/MAR
/p2_datapath_br_tb/R0
/p2_datapath_br_tb/R1
/p2_datapath_br_tb/R2
/p2_datapath_br_tb/R3
/p2_datapath_br_tb/R4
/p2_datapath_br_tb/R5
/p2_datapath_br_tb/R6
/p2_datapath_br_tb/R7
/p2_datapath_br_tb/R8
/p2_datapath_br_tb/R9
/p2_datapath_br_tb/R10
/p2_datapath_br_tb/R11
/p2_datapath_br_tb/R12
/p2_datapath_br_tb/R13
/p2_datapath_br_tb/R14
/p2_datapath_br_tb/R15
/p2_datapath_br_tb/Hi
/p2_datapath_br_tb/Lo
/p2_datapath_br_tb/PC
/p2_datapath_br_tb/bus_mux_out
/p2_datapath_br_tb/IR
/p2_datapath_br_tb/C_sign_ext
/p2_datapath_br_tb/Z
/p2_datapath_br_tb/ALUout
/p2_datapath_br_tb/Rins
/p2_datapath_br_tb/Routs
/p2_datapath_br_tb/t1
/p2_datapath_br_tb/t2
/p2_datapath_br_tb/t3
/p2_datapath_br_tb/t4
/p2_datapath_br_tb/Present_state
Now
Cursor 1
500 ps
303 ps

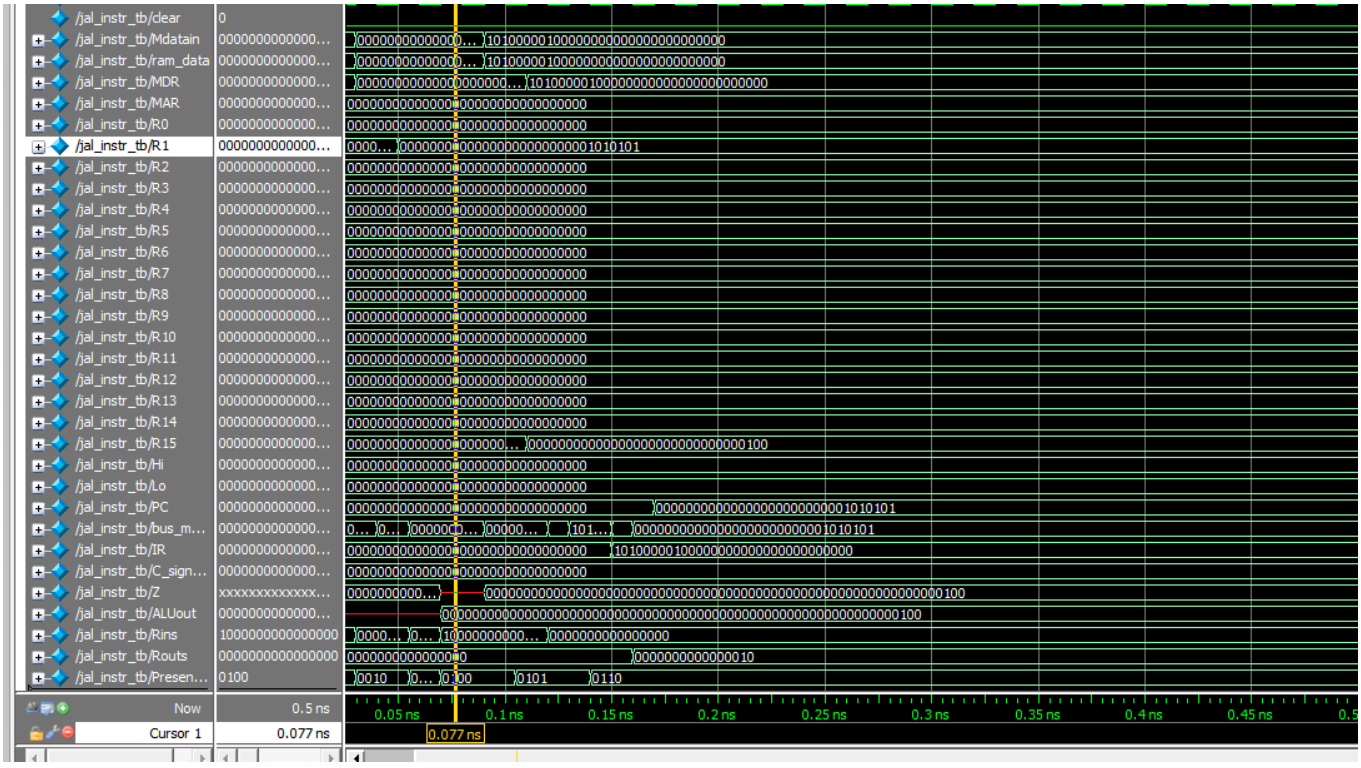
jr R1


Msgs
/j_instr_tb/PCout
/j_instr_tb/Zlowout
/j_instr_tb/MDRout
/j_instr_tb/Cout
/j_instr_tb/MARin
/j_instr_tb/Zin
/j_instr_tb/PCin
/j_instr_tb/MDRin
/j_instr_tb/IRin
/j_instr_tb/Yin
/j_instr_tb/Gra
/j_instr_tb/Grb
/j_instr_tb/Grc
/j_instr_tb/Rin
/j_instr_tb/Rout
/j_instr_tb/BAout
/j_instr_tb/IncPC
/j_instr_tb/Read
/j_instr_tb/Write
/j_instr_tb/ADD
/j_instr_tb/AND
/j_instr_tb/OR
/j_instr_tb/SUB
/j_instr_tb/SHR
/j_instr_tb/SHL
/j_instr_tb/ROL
/j_instr_tb/ROR
/j_instr_tb/NEG
/j_instr_tb/NOT
/j_instr_tb/Clock
/j_instr_tb/clear

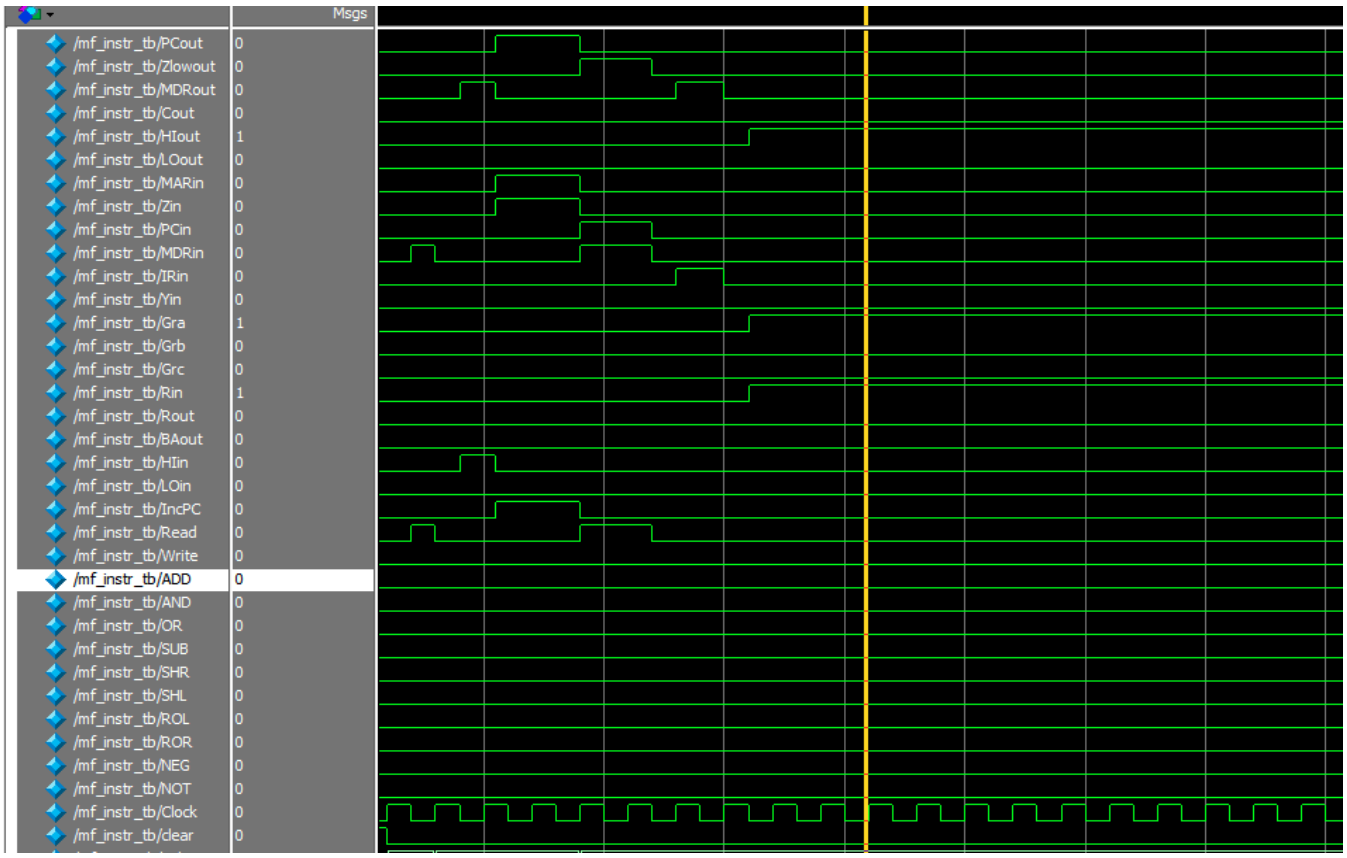

/j_instr_tb/Mdatain
/j_instr_tb/ram_data
/j_instr_tb/MDR
/j_instr_tb/MAR
/j_instr_tb/R0
/j_instr_tb/R1
/j_instr_tb/R2
/j_instr_tb/R3
/j_instr_tb/R4
/j_instr_tb/R5
/j_instr_tb/R6
/j_instr_tb/R7
/j_instr_tb/R8
/j_instr_tb/R9
/j_instr_tb/R10
/j_instr_tb/R11
/j_instr_tb/R12
/j_instr_tb/R13
/j_instr_tb/R14
/j_instr_tb/R15
/j_instr_tb/Hi
/j_instr_tb/Lo
/j_instr_tb/PC
/j_instr_tb/bus_mu...
/j_instr_tb/IR
/j_instr_tb/C_sign_ext
/j_instr_tb/Z
/j_instr_tb/ALUout
/j_instr_tb/Rins
/j_instr_tb/Routs
/j_instr_tb/Present...
Now
Cursor 1
0.5 ns
0.156 ns
0.05 ns
0.1 ns
0.15 ns
0.2 ns

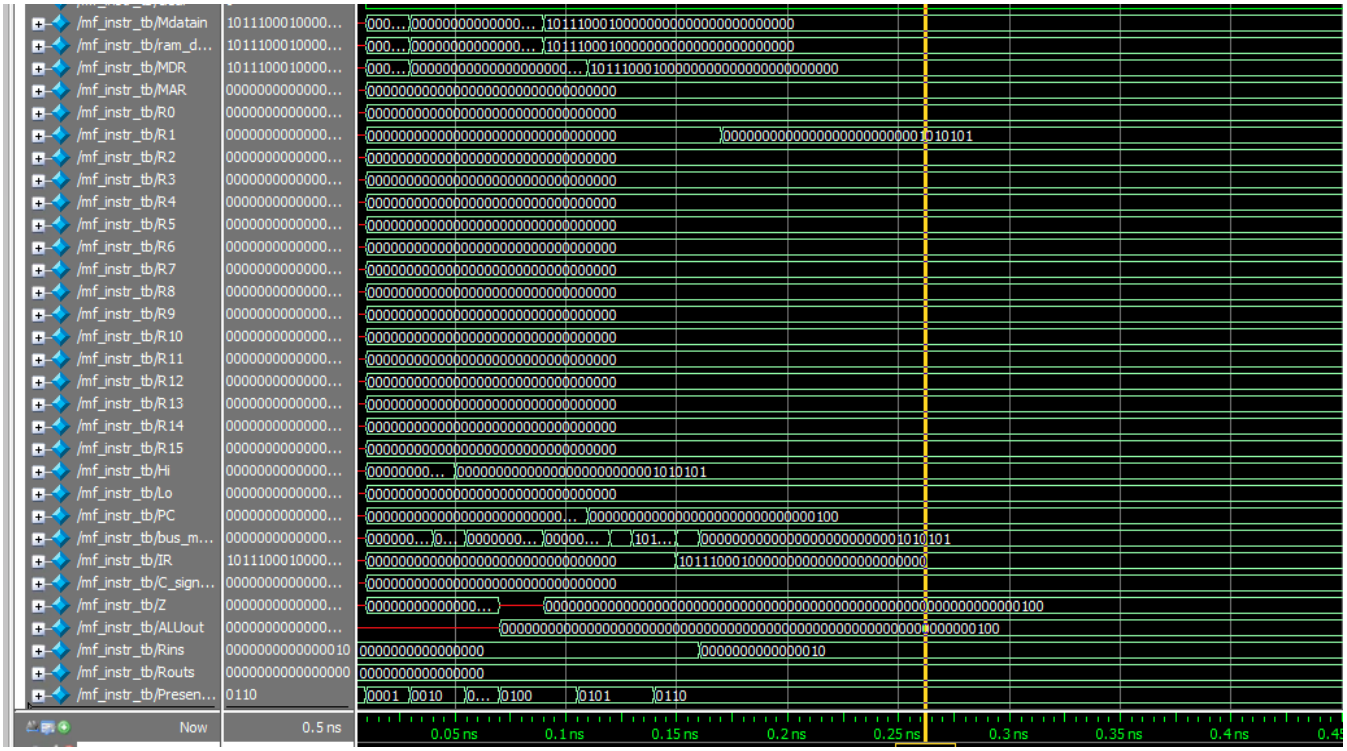
jal R1


Msgs
/jal_instr_tb/PCout
/jal_instr_tb/Zlowout
/jal_instr_tb/MDRout
/jal_instr_tb/Cout
/jal_instr_tb/MARin
/jal_instr_tb/Zin
/jal_instr_tb/PCin
/jal_instr_tb/MDRin
/jal_instr_tb/IRin
/jal_instr_tb/Yin
/jal_instr_tb/Gra
/jal_instr_tb/Grb
/jal_instr_tb/Grc
/jal_instr_tb/Rin
/jal_instr_tb/Rout
/jal_instr_tb/BAout
/jal_instr_tb/IncPC
/jal_instr_tb/Read
/jal_instr_tb/Write
/jal_instr_tb/ADD
/jal_instr_tb/AND
/jal_instr_tb/OR
/jal_instr_tb/SUB
/jal_instr_tb/SHR
/jal_instr_tb/SHL
/jal_instr_tb/ROL
/jal_instr_tb/ROR
/jal_instr_tb/NEG
/jal_instr_tb/NOT
/jal_instr_tb/Clock
/jal_instr_tb/clear

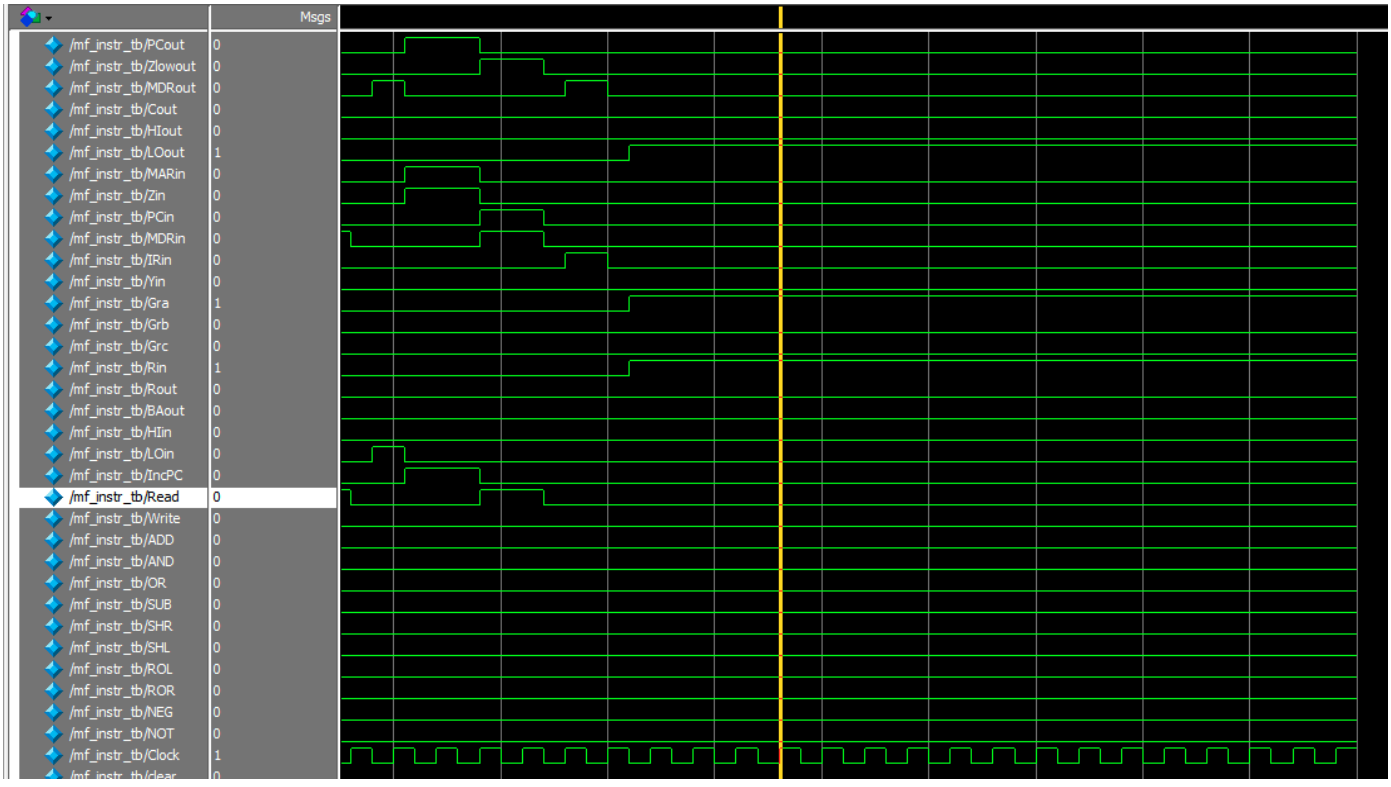

/jal_instr_tb/clear
/jal_instr_tb/Mdatain
/jal_instr_tb/ram_data
/jal_instr_tb/MDR
/jal_instr_tb/MAR
/jal_instr_tb/R0
/jal_instr_tb/R1
/jal_instr_tb/R2
/jal_instr_tb/R3
/jal_instr_tb/R4
/jal_instr_tb/R5
/jal_instr_tb/R6
/jal_instr_tb/R7
/jal_instr_tb/R8
/jal_instr_tb/R9
/jal_instr_tb/R10
/jal_instr_tb/R11
/jal_instr_tb/R12
/jal_instr_tb/R13
/jal_instr_tb/R14
/jal_instr_tb/R15
/jal_instr_tb/Hi
/jal_instr_tb/Lo
/jal_instr_tb/PC
/jal_instr_tb/bus_m...
/jal_instr_tb/IR
/jal_instr_tb/C_sign...
/jal_instr_tb/Z
/jal_instr_tb/ALUout
/jal_instr_tb/Rins
/jal_instr_tb/Routs
/jal_instr_tb/Presen...
Now
Cursor 1
0.077 ns

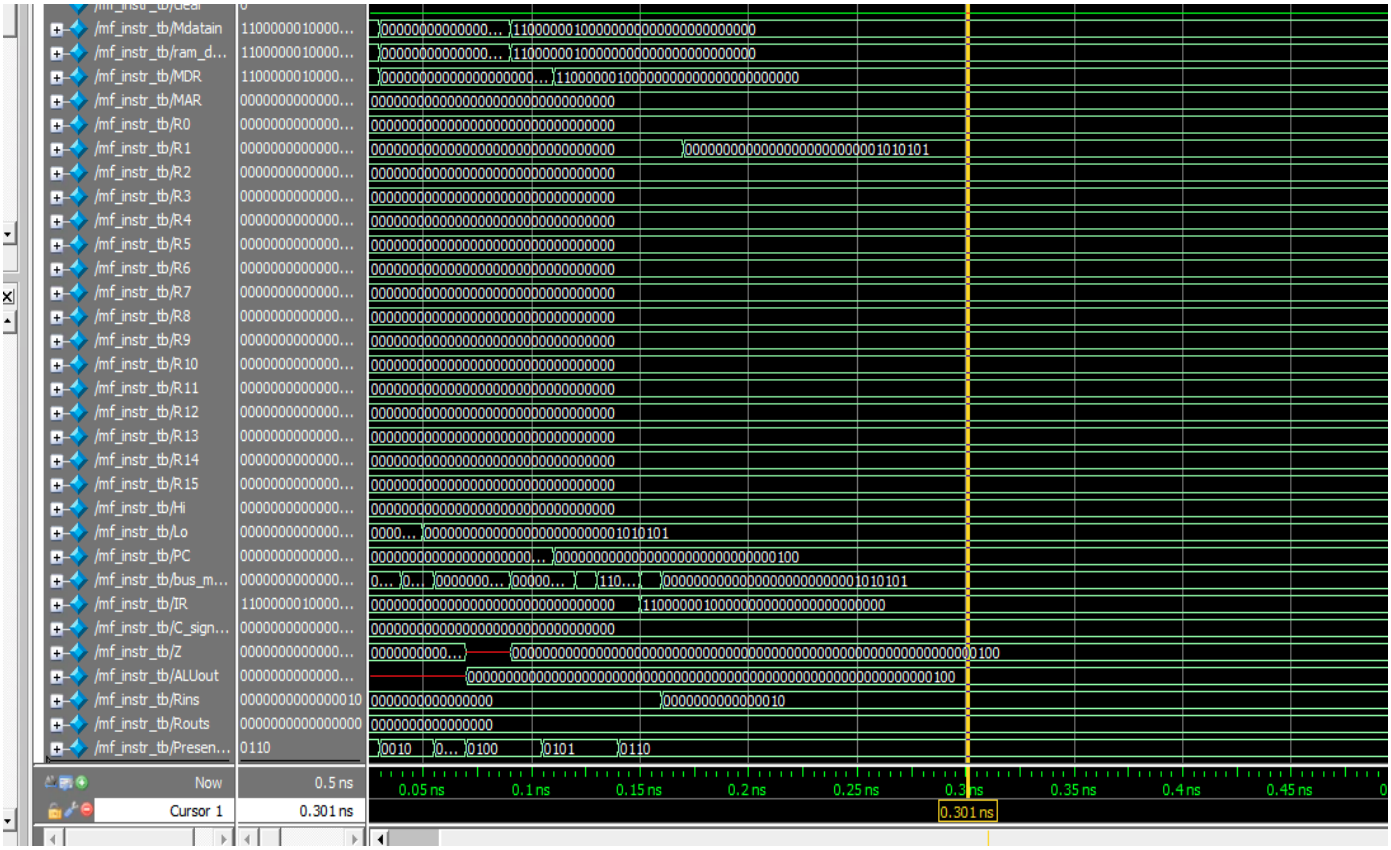
mfhi R1


Msgs
/mf_instr_tb/PCout
/mf_instr_tb/Zlowout
/mf_instr_tb/MDRout
/mf_instr_tb/Cout
/mf_instr_tb/HIout
/mf_instr_tb/LOout
/mf_instr_tb/MARin
/mf_instr_tb/Zin
/mf_instr_tb/PCin
/mf_instr_tb/MDRin
/mf_instr_tb/IRin
/mf_instr_tb/Yin
/mf_instr_tb/Gra
/mf_instr_tb/Grb
/mf_instr_tb/Grc
/mf_instr_tb/Rin
/mf_instr_tb/Rout
/mf_instr_tb/BAout
/mf_instr_tb/HIin
/mf_instr_tb/LOin
/mf_instr_tb/IncPC
/mf_instr_tb/Read
/mf_instr_tb/Write
/mf_instr_tb/ADD
/mf_instr_tb/AND
/mf_instr_tb/OR
/mf_instr_tb/SUB
/mf_instr_tb/SHR
/mf_instr_tb/SHL
/mf_instr_tb/ROL
/mf_instr_tb/ROR
/mf_instr_tb/NEG
/mf_instr_tb/NOT
/mf_instr_tb/Clock
/mf_instr_tb/clear

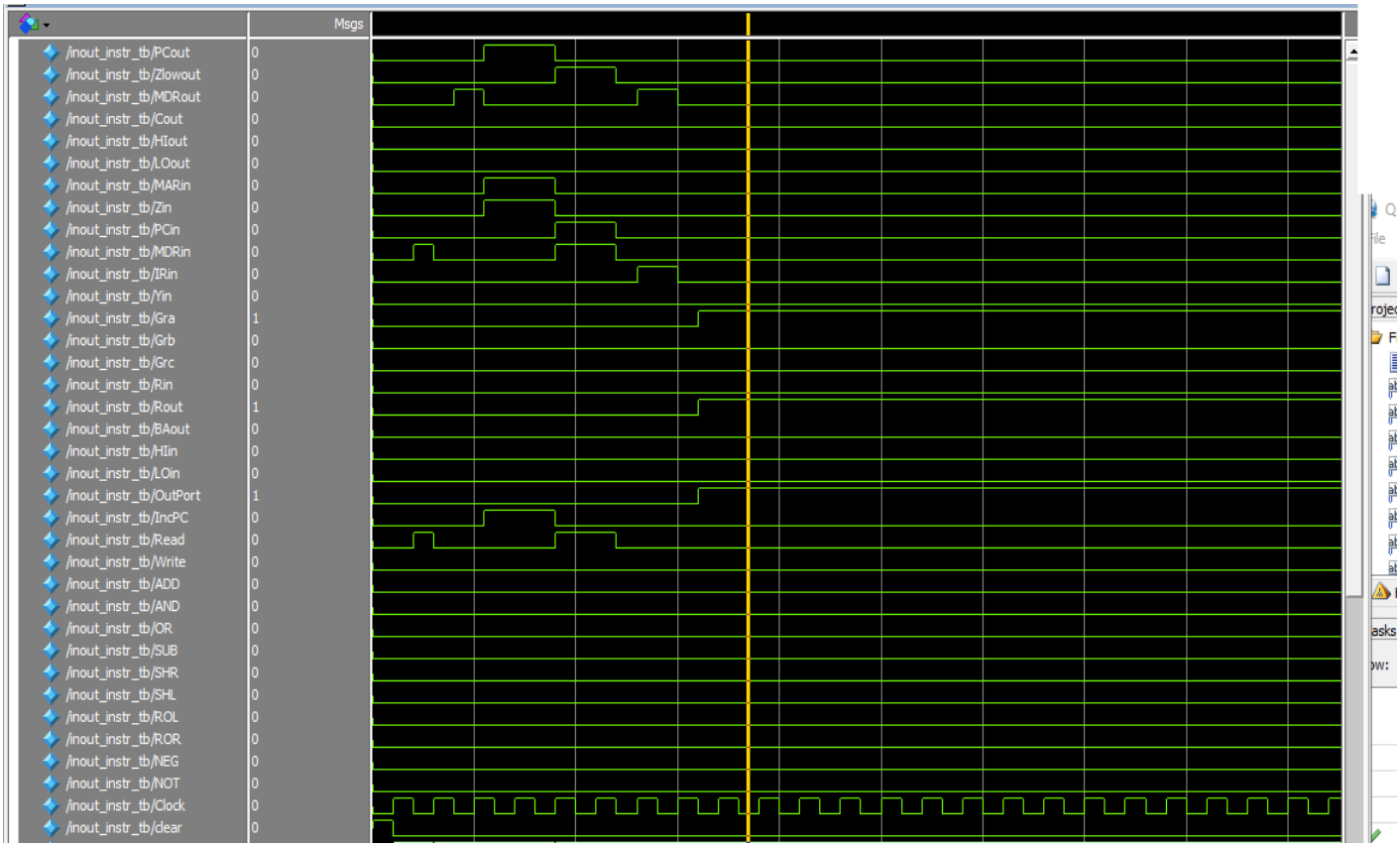

/mf_instr_tb/Mdatain
/mf_instr_tb/ram_d...
/mf_instr_tb/MDR
/mf_instr_tb/MAR
/mf_instr_tb/R0
/mf_instr_tb/R1
/mf_instr_tb/R2
/mf_instr_tb/R3
/mf_instr_tb/R4
/mf_instr_tb/R5
/mf_instr_tb/R6
/mf_instr_tb/R7
/mf_instr_tb/R8
/mf_instr_tb/R9
/mf_instr_tb/R10
/mf_instr_tb/R11
/mf_instr_tb/R12
/mf_instr_tb/R13
/mf_instr_tb/R14
/mf_instr_tb/R15
/mf_instr_tb/Hi
/mf_instr_tb/Lo
/mf_instr_tb/PC
/mf_instr_tb/bus_m...
/mf_instr_tb/IR
/mf_instr_tb/C_sign...
/mf_instr_tb/Z
/mf_instr_tb/ALUout
/mf_instr_tb/Rins
/mf_instr_tb/Routs
/mf_instr_tb/Presen...
Now
0.5 ns
0.05 ns
0.1 ns
0.15 ns
0.2 ns
0.25 ns
0.3 ns
0.35 ns
0.4 ns

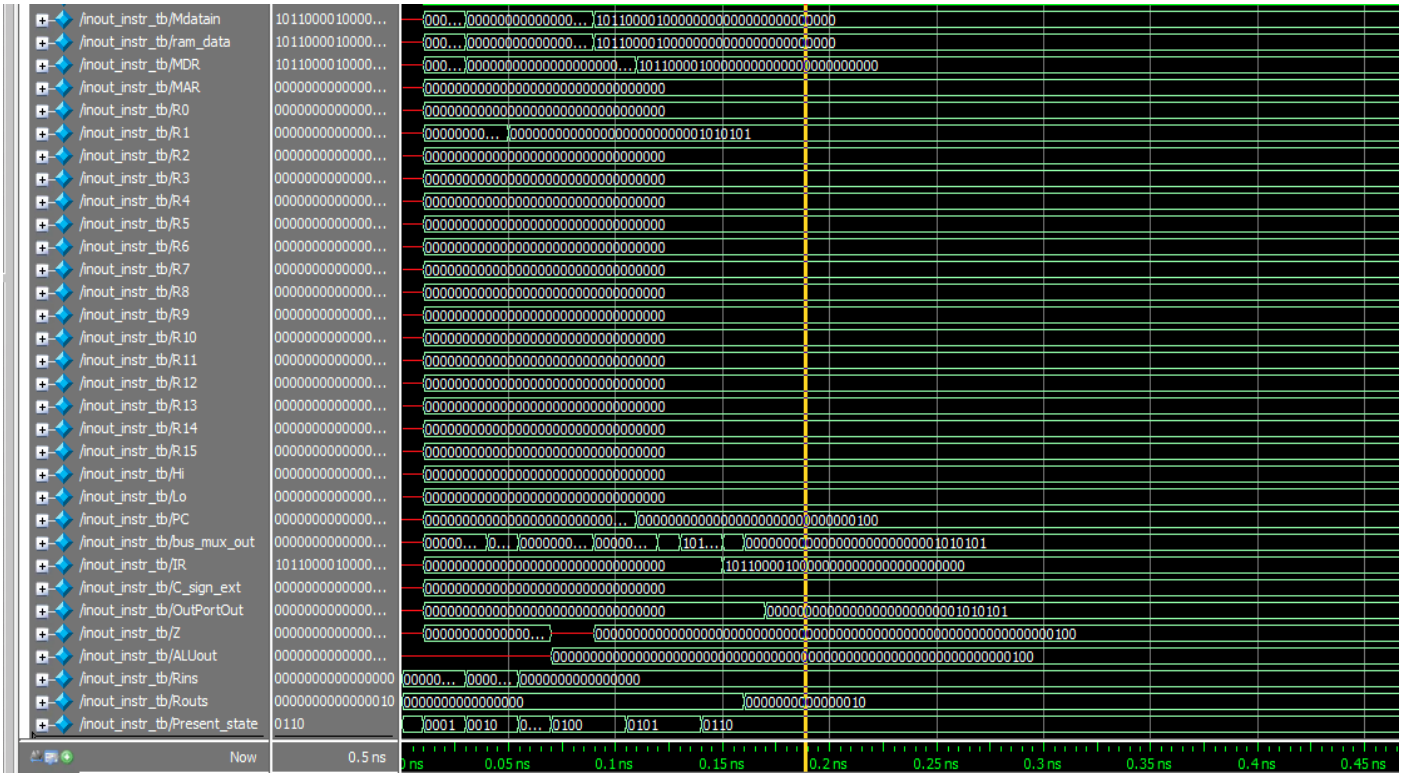
mflo R1


Msgs
/mf_instr_tb/PCout
/mf_instr_tb/Zlowout
/mf_instr_tb/MDRout
/mf_instr_tb/Cout
/mf_instr_tb/HIout
/mf_instr_tb/LOout
/mf_instr_tb/MARin
/mf_instr_tb/Zin
/mf_instr_tb/PCin
/mf_instr_tb/MDRin
/mf_instr_tb/IRin
/mf_instr_tb/Yin
/mf_instr_tb/Gra
/mf_instr_tb/Grb
/mf_instr_tb/Grc
/mf_instr_tb/Rin
/mf_instr_tb/Rout
/mf_instr_tb/BAout
/mf_instr_tb/HIin
/mf_instr_tb/LOin
/mf_instr_tb/IncPC
/mf_instr_tb/Read
/mf_instr_tb/Write
/mf_instr_tb/ADD
/mf_instr_tb/AND
/mf_instr_tb/OR
/mf_instr_tb/SUB
/mf_instr_tb/SHR
/mf_instr_tb/SHL
/mf_instr_tb/ROL
/mf_instr_tb/ROR
/mf_instr_tb/NEG
/mf_instr_tb/NOT
/mf_instr_tb/Clock

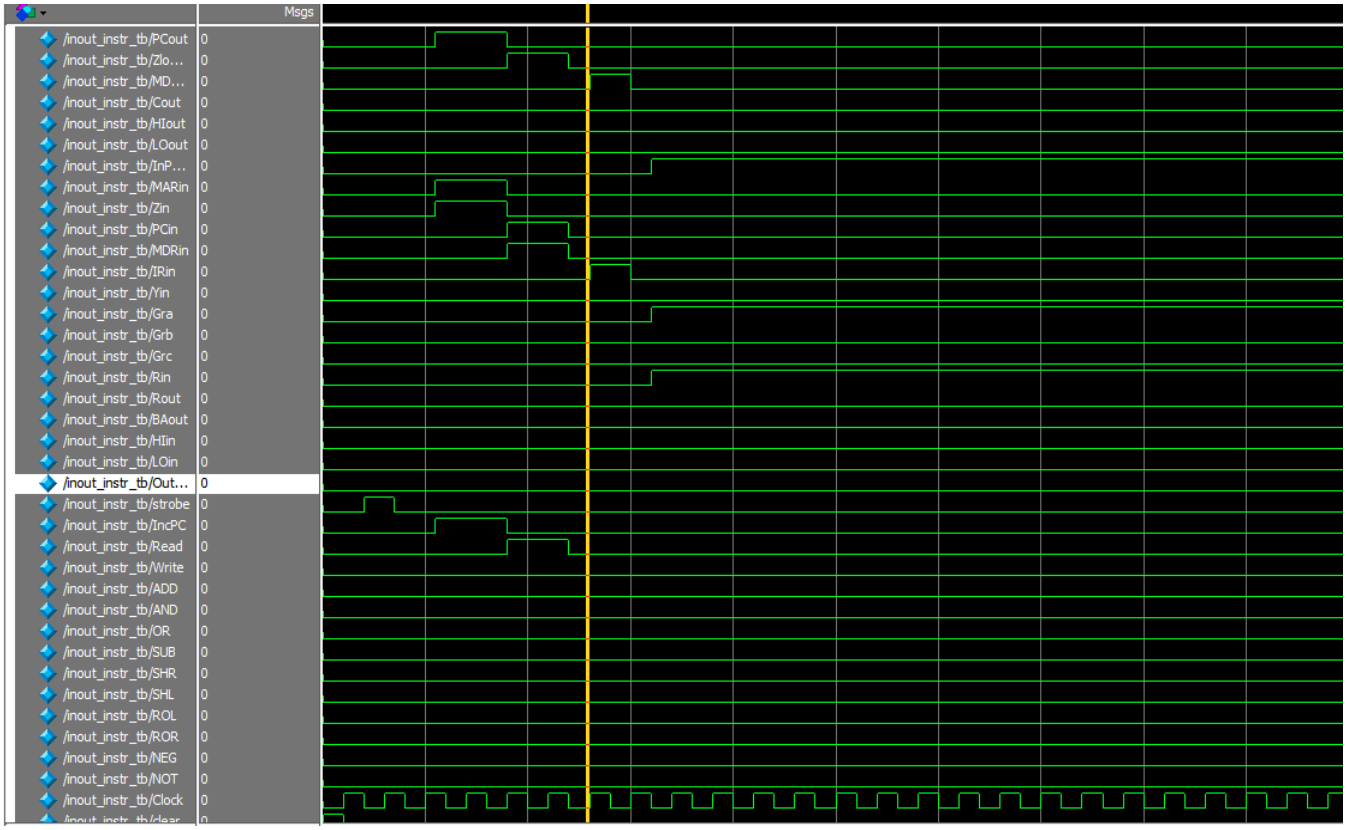

/mf_instr_tb/Mdatain
/mf_instr_tb/ram_d...
/mf_instr_tb/MDR
/mf_instr_tb/MAR
/mf_instr_tb/R0
/mf_instr_tb/R1
/mf_instr_tb/R2
/mf_instr_tb/R3
/mf_instr_tb/R4
/mf_instr_tb/R5
/mf_instr_tb/R6
/mf_instr_tb/R7
/mf_instr_tb/R8
/mf_instr_tb/R9
/mf_instr_tb/R10
/mf_instr_tb/R11
/mf_instr_tb/R12
/mf_instr_tb/R13
/mf_instr_tb/R14
/mf_instr_tb/R15
/mf_instr_tb/Hi
/mf_instr_tb/Lo
/mf_instr_tb/PC
/mf_instr_tb/bus_m...
/mf_instr_tb/IR
/mf_instr_tb/C_sign...
/mf_instr_tb/Z
/mf_instr_tb/ALUout
/mf_instr_tb/Rins
/mf_instr_tb/Routs
/mf_instr_tb/Presen...
Now
Cursor 1
0.5 ns
0.301 ns
0.05 ns
0.1 ns
0.15 ns
0.2 ns
0.25 ns
0.3 ns
0.35 ns
0.4 ns
0.45 ns

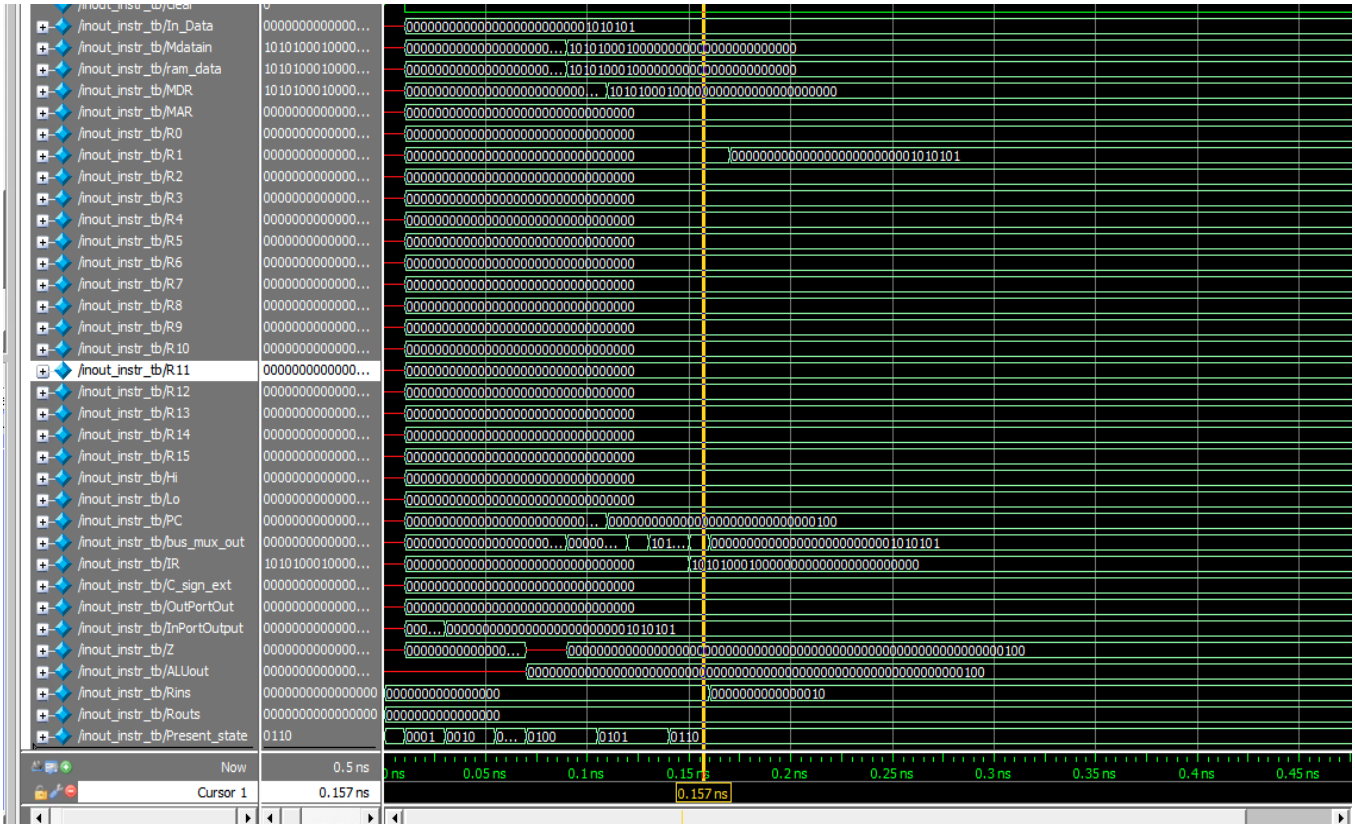
out R1


Msgs
/inout_instr_tb/PCout
/inout_instr_tb/Zlowout
/inout_instr_tb/MDRout
/inout_instr_tb/Cout
/inout_instr_tb/HIout
/inout_instr_tb/LOout
/inout_instr_tb/MARin
/inout_instr_tb/Zin
/inout_instr_tb/PCin
/inout_instr_tb/MDRin
/inout_instr_tb/IRin
/inout_instr_tb/Yin
/inout_instr_tb/Gra
/inout_instr_tb/Grb
/inout_instr_tb/Grc
/inout_instr_tb/Rin
/inout_instr_tb/Rout
/inout_instr_tb/BAout
/inout_instr_tb/HIin
/inout_instr_tb/LOin
/inout_instr_tb/OutPort
/inout_instr_tb/IncPC
/inout_instr_tb/Read
/inout_instr_tb/Write
/inout_instr_tb/ADD
/inout_instr_tb/AND
/inout_instr_tb/OR
/inout_instr_tb/SUB
/inout_instr_tb/SHR
/inout_instr_tb/SHL
/inout_instr_tb/ROL
/inout_instr_tb/ROR
/inout_instr_tb/NEG
/inout_instr_tb/NOT
/inout_instr_tb/Clock
/inout_instr_tb/clear


/inout_instr_tb/Mdatain
/inout_instr_tb/ram_data
/inout_instr_tb/MDR
/inout_instr_tb/MAR
/inout_instr_tb/R0
/inout_instr_tb/R1
/inout_instr_tb/R2
/inout_instr_tb/R3
/inout_instr_tb/R4
/inout_instr_tb/R5
/inout_instr_tb/R6
/inout_instr_tb/R7
/inout_instr_tb/R8
/inout_instr_tb/R9
/inout_instr_tb/R10
/inout_instr_tb/R11
/inout_instr_tb/R12
/inout_instr_tb/R13
/inout_instr_tb/R14
/inout_instr_tb/R15
/inout_instr_tb/Hi
/inout_instr_tb/Lo
/inout_instr_tb/PC
/inout_instr_tb/bus_mux_out
/inout_instr_tb/IR
/inout_instr_tb/C_sign_ext
/inout_instr_tb/OutPortOut
/inout_instr_tb/Z
/inout_instr_tb/ALUout
/inout_instr_tb/Rins
/inout_instr_tb/Routs
/inout_instr_tb/Present_state
Now
0.5 ns
0.05 ns
0.1 ns
0.15 ns
0.2 ns
0.25 ns
0.3 ns
0.35 ns
0.4 ns
0.45 ns

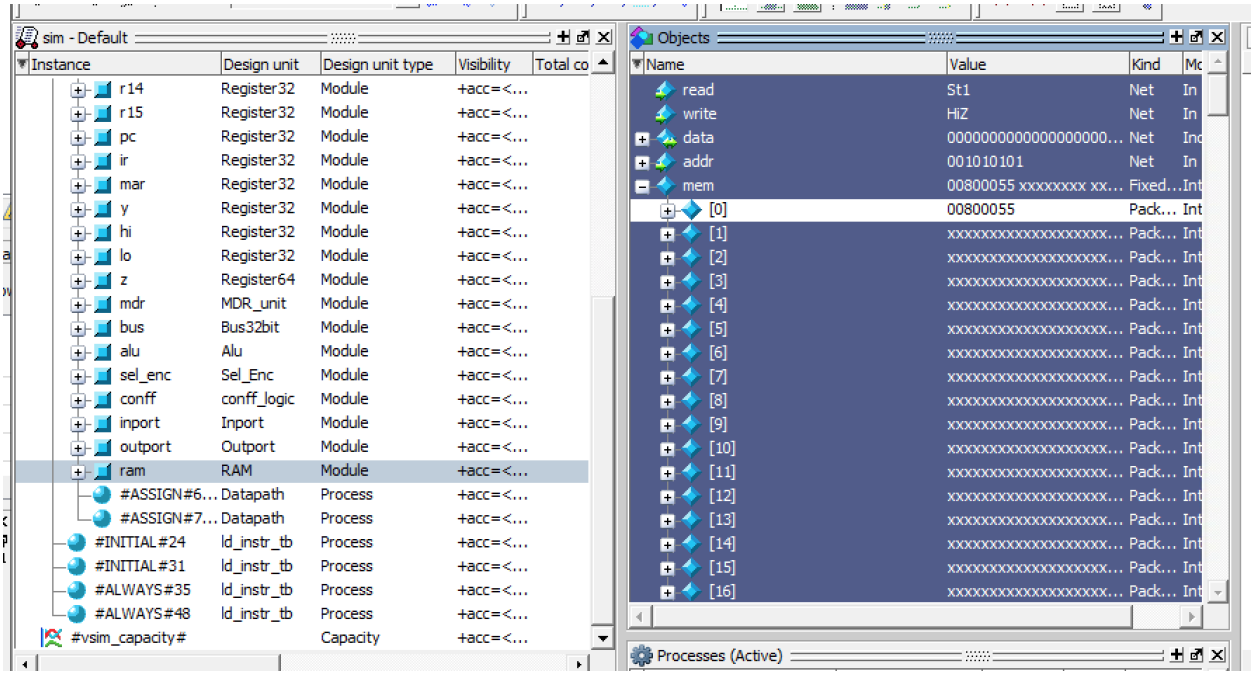
in R1

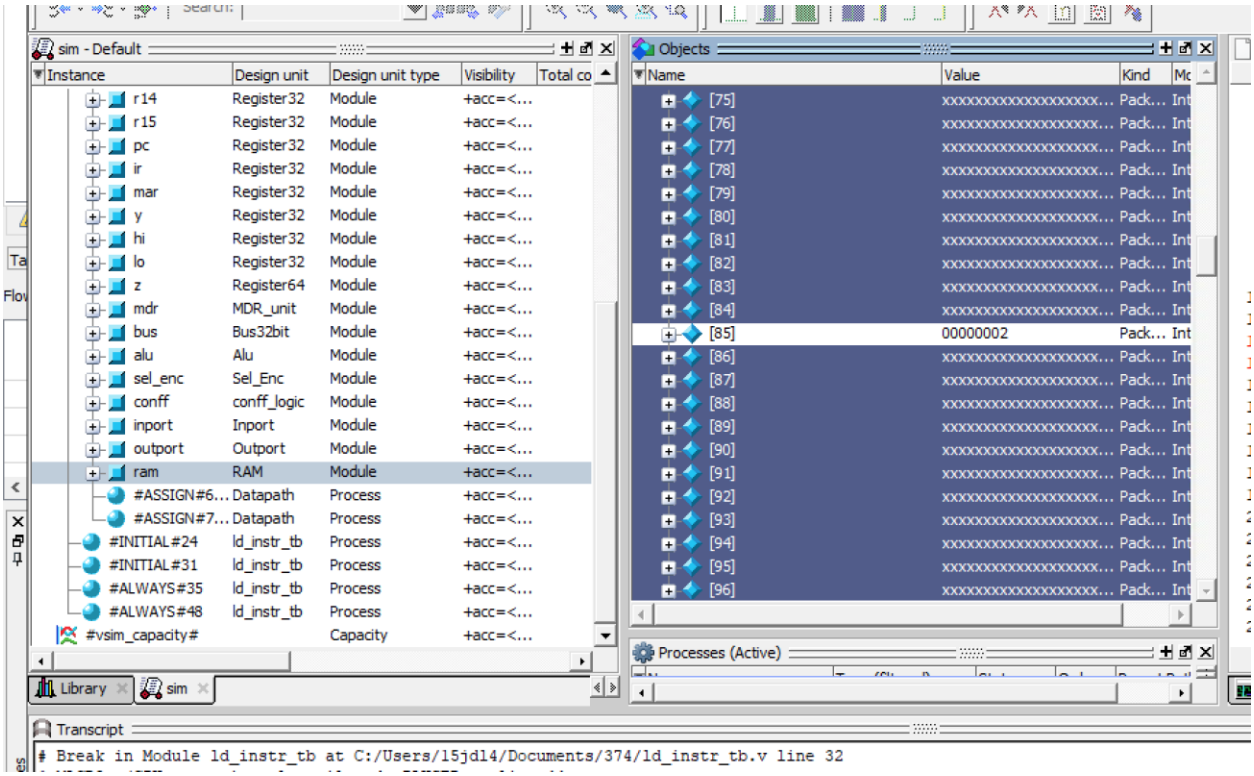

Msgs
/inout_instr_tb/PCout
/inout_instr_tb/Zlo...
/inout_instr_tb/MD...
/inout_instr_tb/Cout
/inout_instr_tb/HIout
/inout_instr_tb/LOout
/inout_instr_tb/InP...
/inout_instr_tb/MARin
/inout_instr_tb/Zin
/inout_instr_tb/PCin
/inout_instr_tb/MDRin
/inout_instr_tb/IRin
/inout_instr_tb/Yin
/inout_instr_tb/Gra
/inout_instr_tb/Grb
/inout_instr_tb/Grc
/inout_instr_tb/Rin
/inout_instr_tb/Rout
/inout_instr_tb/BAout
/inout_instr_tb/HIin
/inout_instr_tb/LOin
/inout_instr_tb/Out...
/inout_instr_tb/strobe
/inout_instr_tb/IncPC
/inout_instr_tb/Read
/inout_instr_tb/Write
/inout_instr_tb/ADD
/inout_instr_tb/AND
/inout_instr_tb/OR
/inout_instr_tb/SUB
/inout_instr_tb/SHR
/inout_instr_tb/SHL
/inout_instr_tb/ROL
/inout_instr_tb/ROR
/inout_instr_tb/NEG
/inout_instr_tb/NOT
/inout_instr_tb/Clock


/inout_instr_tb/In_Data
/inout_instr_tb/Mdatain
/inout_instr_tb/ram_data
/inout_instr_tb/MDR
/inout_instr_tb/MAR
/inout_instr_tb/R0
/inout_instr_tb/R1
/inout_instr_tb/R2
/inout_instr_tb/R3
/inout_instr_tb/R4
/inout_instr_tb/R5
/inout_instr_tb/R6
/inout_instr_tb/R7
/inout_instr_tb/R8
/inout_instr_tb/R9
/inout_instr_tb/R10
/inout_instr_tb/R11
/inout_instr_tb/R12
/inout_instr_tb/R13
/inout_instr_tb/R14
/inout_instr_tb/R15
/inout_instr_tb/Hi
/inout_instr_tb/Lo
/inout_instr_tb/PC
/inout_instr_tb/bus_mux_out
/inout_instr_tb/IR
/inout_instr_tb/C_sign_ext
/inout_instr_tb/OutPortOut
/inout_instr_tb/InPortOutput
/inout_instr_tb/Z
/inout_instr_tb/ALUout
/inout_instr_tb/Rins
/inout_instr_tb/Routs
/inout_instr_tb/Present_state
Now
Cursor 1
0.157 ns

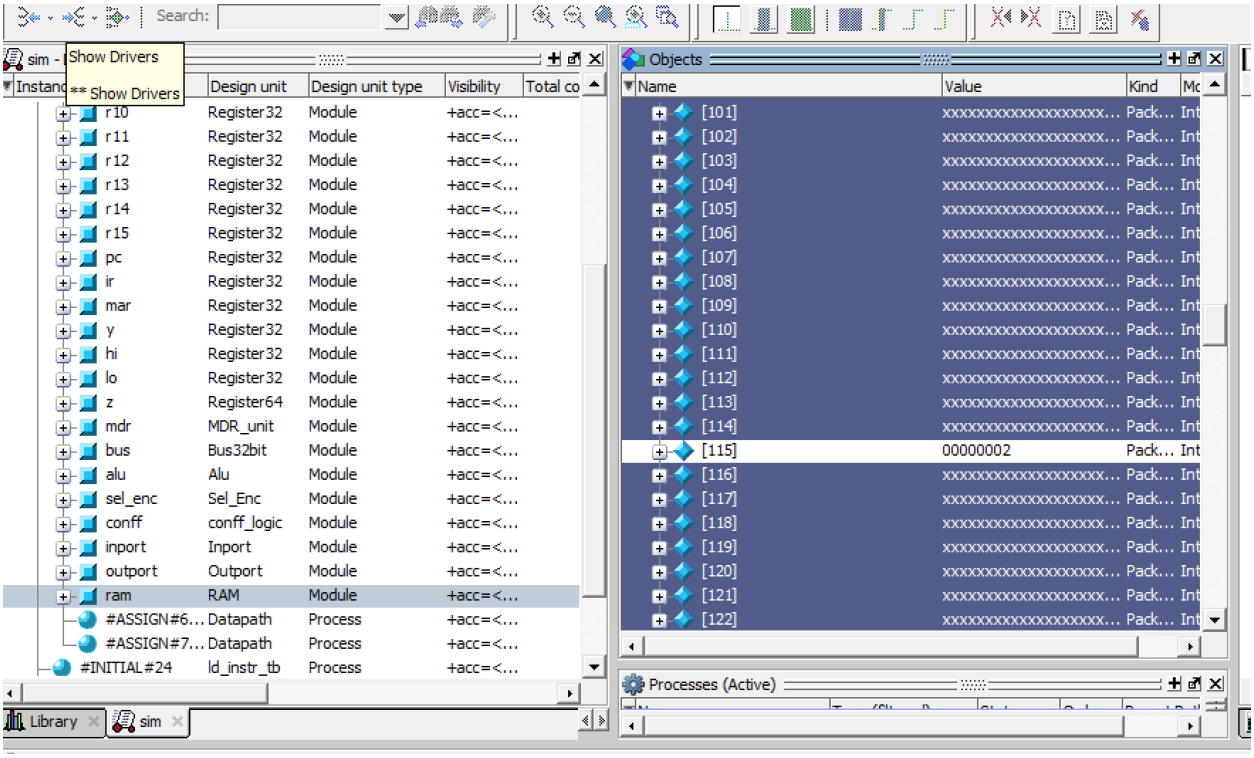
Memory Contents

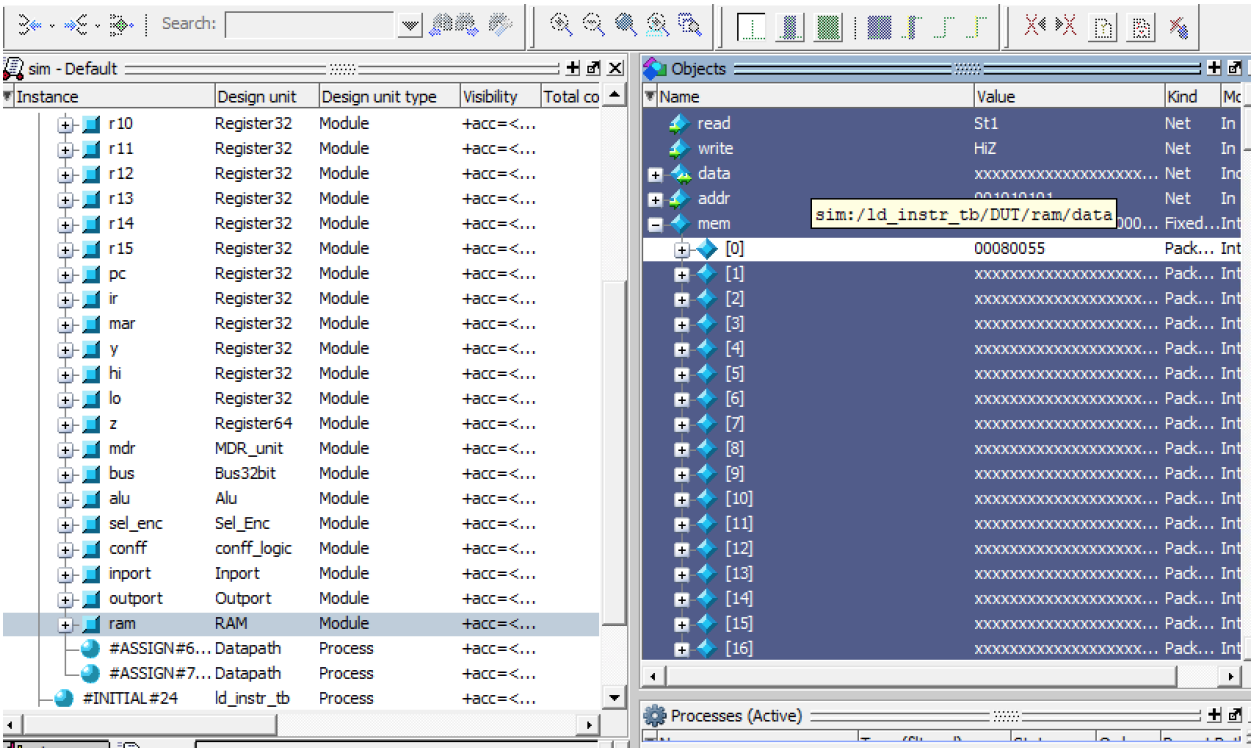
ld R1, $85



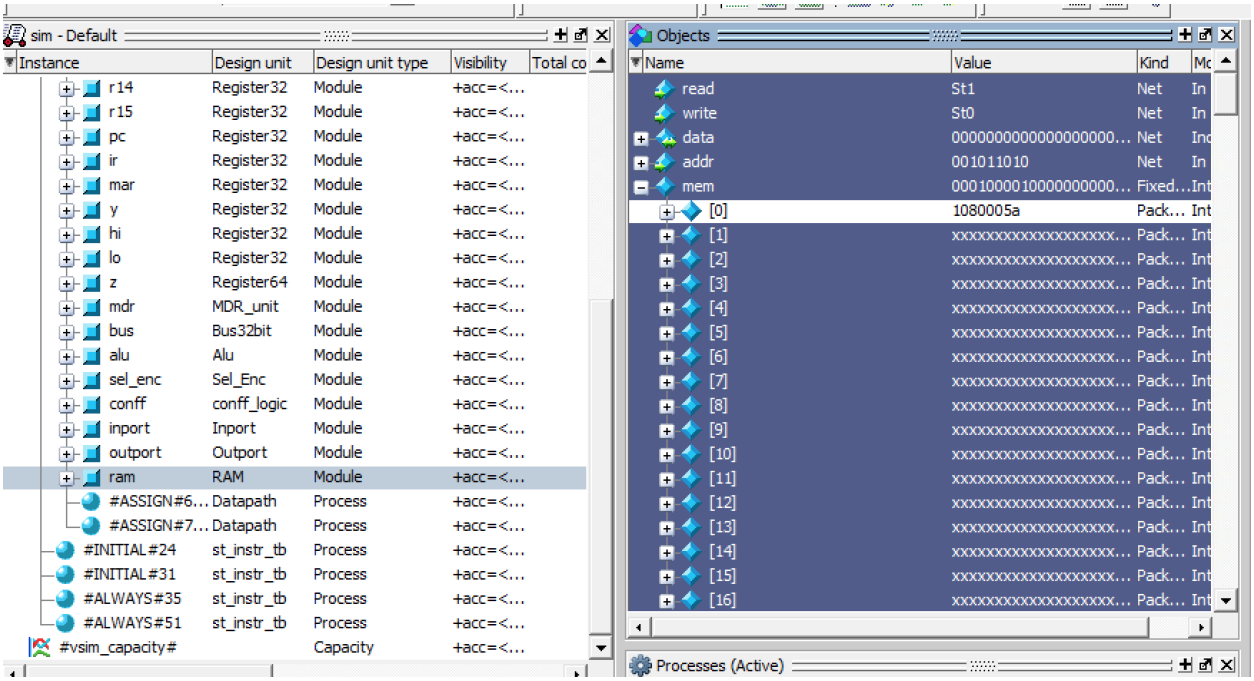


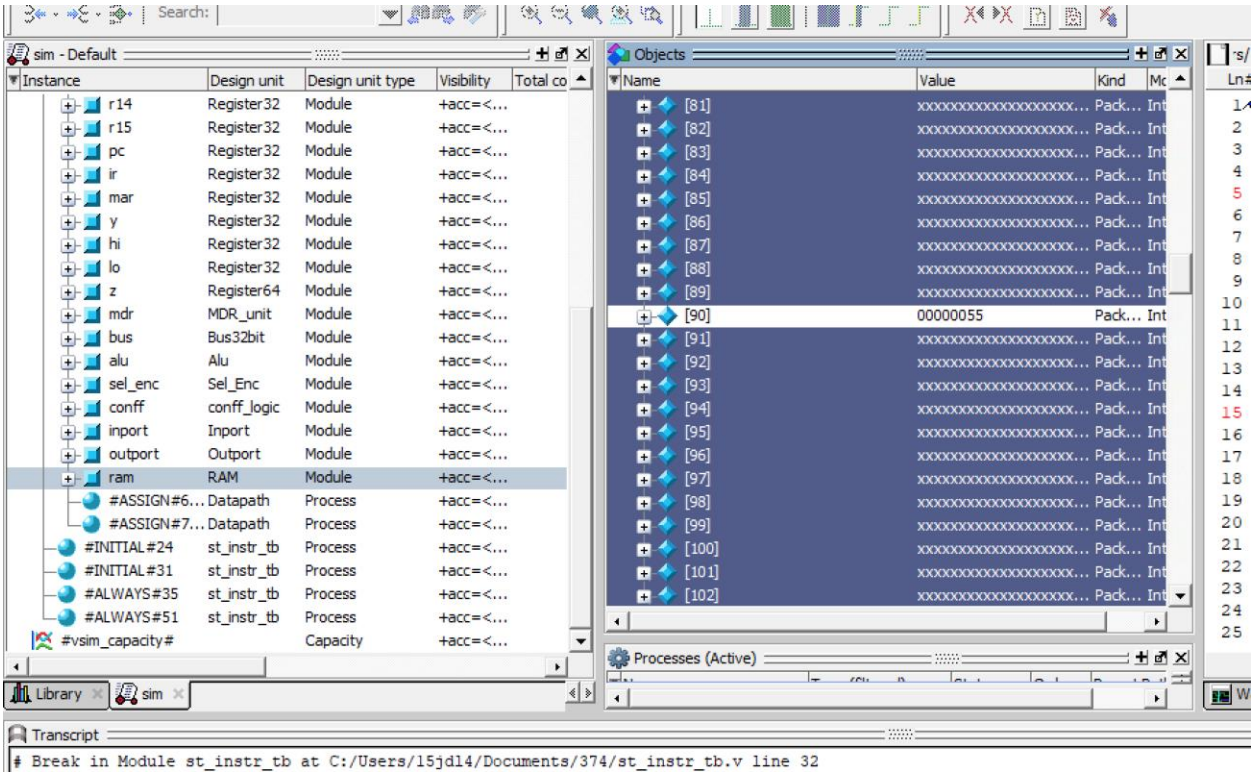
ld R0, $35(R1)


Search:
sim -
Show Drivers
** Show Drivers
Instance
Design unit
Design unit type
Visibility
Total co
r10 Register32 Module +acc=<...
r11 Register32 Module +acc=<...
r12 Register32 Module +acc=<...
r13 Register32 Module +acc=<...
r14 Register32 Module +acc=<...
r15 Register32 Module +acc=<...
pc Register32 Module +acc=<...
ir Register32 Module +acc=<...
mar Register32 Module +acc=<...
y Register32 Module +acc=<...
hi Register32 Module +acc=<...
lo Register32 Module +acc=<...
z Register64 Module +acc=<...
mdr MDR_unit Module +acc=<...
bus Bus32bit Module +acc=<...
alu Alu Module +acc=<...
sel_enc Sel_Enc Module +acc=<...
conff conff_logic Module +acc=<...
inport Inport Module +acc=<...
outport Outport Module +acc=<...
ram RAM Module +acc=<...
#ASSIGN#6... Datapath Process +acc=<...
#ASSIGN#7... Datapath Process +acc=<...
#INITIAL#24 ld_instr_tb Process +acc=<...
Library
sim
Objects
Name
Value
Kind
[101] xxxxxxxxxxxxxxxxxxx... Pack... Int
[102] xxxxxxxxxxxxxxxxxxx... Pack... Int
[103] xxxxxxxxxxxxxxxxxxx... Pack... Int
[104] xxxxxxxxxxxxxxxxxxx... Pack... Int
[105] xxxxxxxxxxxxxxxxxxx... Pack... Int
[106] xxxxxxxxxxxxxxxxxxx... Pack... Int
[107] xxxxxxxxxxxxxxxxxxx... Pack... Int
[108] xxxxxxxxxxxxxxxxxxx... Pack... Int
[109] xxxxxxxxxxxxxxxxxxx... Pack... Int
[110] xxxxxxxxxxxxxxxxxxx... Pack... Int
[111] xxxxxxxxxxxxxxxxxxx... Pack... Int
[112] xxxxxxxxxxxxxxxxxxx... Pack... Int
[113] xxxxxxxxxxxxxxxxxxx... Pack... Int
[114] xxxxxxxxxxxxxxxxxxx... Pack... Int
[115] 00000002 Pack... Int
[116] xxxxxxxxxxxxxxxxxxx... Pack... Int
[117] xxxxxxxxxxxxxxxxxxx... Pack... Int
[118] xxxxxxxxxxxxxxxxxxx... Pack... Int
[119] xxxxxxxxxxxxxxxxxxx... Pack... Int
[120] xxxxxxxxxxxxxxxxxxx... Pack... Int
[121] xxxxxxxxxxxxxxxxxxx... Pack... Int
[122] xxxxxxxxxxxxxxxxxxx... Pack... Int
Processes (Active)

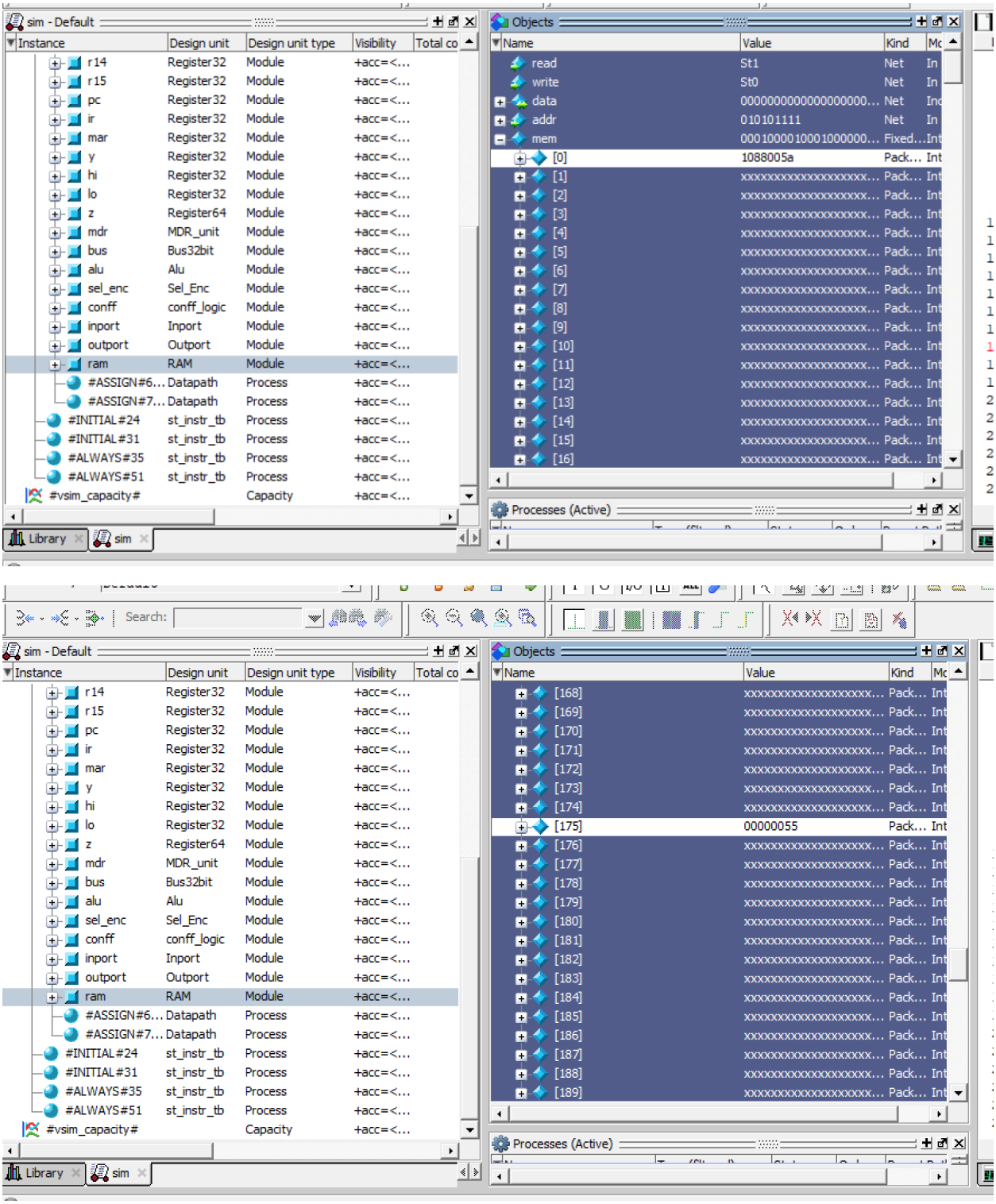

Search:
sim - Default
Instance
Design unit
Design unit type
Visibility
Total co
r10 Register32 Module +acc=<...
r11 Register32 Module +acc=<...
r12 Register32 Module +acc=<...
r13 Register32 Module +acc=<...
r14 Register32 Module +acc=<...
r15 Register32 Module +acc=<...
pc Register32 Module +acc=<...
ir Register32 Module +acc=<...
mar Register32 Module +acc=<...
y Register32 Module +acc=<...
hi Register32 Module +acc=<...
lo Register32 Module +acc=<...
z Register64 Module +acc=<...
mdr MDR_unit Module +acc=<...
bus Bus32bit Module +acc=<...
alu Alu Module +acc=<...
sel_enc Sel_Enc Module +acc=<...
conff conff_logic Module +acc=<...
inport Inport Module +acc=<...
outport Outport Module +acc=<...
ram RAM Module +acc=<...
#ASSIGN#6... Datapath Process +acc=<...
#ASSIGN#7... Datapath Process +acc=<...
#INITIAL#24 ld_instr_tb Process +acc=<...
Objects
Name
Value
Kind
read St1 Net In
write HiZ Net In
data xxxxxxxxxxxxxxxxxxx... Net Inc
addr Net In
mem sim:/ld_instr_tb/DUT/ram/data Fixed...Int
[0] 00080055 Pack... Int
[1] xxxxxxxxxxxxxxxxxxx... Pack... Int
[2] xxxxxxxxxxxxxxxxxxx... Pack... Int
[3] xxxxxxxxxxxxxxxxxxx... Pack... Int
[4] xxxxxxxxxxxxxxxxxxx... Pack... Int
[5] xxxxxxxxxxxxxxxxxxx... Pack... Int
[6] xxxxxxxxxxxxxxxxxxx... Pack... Int
[7] xxxxxxxxxxxxxxxxxxx... Pack... Int
[8] xxxxxxxxxxxxxxxxxxx... Pack... Int
[9] xxxxxxxxxxxxxxxxxxx... Pack... Int
[10] xxxxxxxxxxxxxxxxxxx... Pack... Int
[11] xxxxxxxxxxxxxxxxxxx... Pack... Int
[12] xxxxxxxxxxxxxxxxxxx... Pack... Int
[13] xxxxxxxxxxxxxxxxxxx... Pack... Int
[14] xxxxxxxxxxxxxxxxxxx... Pack... Int
[15] xxxxxxxxxxxxxxxxxxx... Pack... Int
[16] xxxxxxxxxxxxxxxxxxx... Pack... Int
Processes (Active)

st $90, R1


sim - Default
Instance
Design unit
Design unit type
Visibility
Total co
r14 Register32 Module +acc=<...
r15 Register32 Module +acc=<...
pc Register32 Module +acc=<...
ir Register32 Module +acc=<...
mar Register32 Module +acc=<...
y Register32 Module +acc=<...
hi Register32 Module +acc=<...
lo Register32 Module +acc=<...
z Register64 Module +acc=<...
mdr MDR_unit Module +acc=<...
bus Bus32bit Module +acc=<...
alu Alu Module +acc=<...
sel_enc Sel_Enc Module +acc=<...
conff conff_logic Module +acc=<...
inport Inport Module +acc=<...
outport Outport Module +acc=<...
ram RAM Module +acc=<...
#ASSIGN#6... Datapath Process +acc=<...
#ASSIGN#7... Datapath Process +acc=<...
#INITIAL#24 st_instr_tb Process +acc=<...
#INITIAL#31 st_instr_tb Process +acc=<...
#ALWAYS#35 st_instr_tb Process +acc=<...
#ALWAYS#51 st_instr_tb Process +acc=<...
#vsim_capacity# Capacity +acc=<...
Objects
Name
Value
Kind
read St1 Net In
write St0 Net In
data 0000000000000000000... Net Inc
addr 001011010 Net In
mem 0001000010000000000... Fixed... Int
[0] 1080005a Pack... Int
[1] xxxxxxxxxxxxxxxxxxx... Pack... Int
[2] xxxxxxxxxxxxxxxxxxx... Pack... Int
[3] xxxxxxxxxxxxxxxxxxx... Pack... Int
[4] xxxxxxxxxxxxxxxxxxx... Pack... Int
[5] xxxxxxxxxxxxxxxxxxx... Pack... Int
[6] xxxxxxxxxxxxxxxxxxx... Pack... Int
[7] xxxxxxxxxxxxxxxxxxx... Pack... Int
[8] xxxxxxxxxxxxxxxxxxx... Pack... Int
[9] xxxxxxxxxxxxxxxxxxx... Pack... Int
[10] xxxxxxxxxxxxxxxxxxx... Pack... Int
[11] xxxxxxxxxxxxxxxxxxx... Pack... Int
[12] xxxxxxxxxxxxxxxxxxx... Pack... Int
[13] xxxxxxxxxxxxxxxxxxx... Pack... Int
[14] xxxxxxxxxxxxxxxxxxx... Pack... Int
[15] xxxxxxxxxxxxxxxxxxx... Pack... Int
[16] xxxxxxxxxxxxxxxxxxx... Pack... Int
Processes (Active)


Search:
sim - Default
Instance
Design unit
Design unit type
Visibility
Total co
r14 Register32 Module +acc=<...
r15 Register32 Module +acc=<...
pc Register32 Module +acc=<...
ir Register32 Module +acc=<...
mar Register32 Module +acc=<...
y Register32 Module +acc=<...
hi Register32 Module +acc=<...
lo Register32 Module +acc=<...
z Register64 Module +acc=<...
mdr MDR_unit Module +acc=<...
bus Bus32bit Module +acc=<...
alu Alu Module +acc=<...
sel_enc Sel_Enc Module +acc=<...
conff conff_logic Module +acc=<...
inport Inport Module +acc=<...
outport Outport Module +acc=<...
ram RAM Module +acc=<...
#ASSIGN#6... Datapath Process +acc=<...
#ASSIGN#7... Datapath Process +acc=<...
#INITIAL#24 st_instr_tb Process +acc=<...
#INITIAL#31 st_instr_tb Process +acc=<...
#ALWAYS#35 st_instr_tb Process +acc=<...
#ALWAYS#51 st_instr_tb Process +acc=<...
#vsim_capacity# Capacity +acc=<...
Objects
Name
Value
Kind
[81] xxxxxxxxxxxxxxxxxxx... Pack... Int
[82] xxxxxxxxxxxxxxxxxxx... Pack... Int
[83] xxxxxxxxxxxxxxxxxxx... Pack... Int
[84] xxxxxxxxxxxxxxxxxxx... Pack... Int
[85] xxxxxxxxxxxxxxxxxxx... Pack... Int
[86] xxxxxxxxxxxxxxxxxxx... Pack... Int
[87] xxxxxxxxxxxxxxxxxxx... Pack... Int
[88] xxxxxxxxxxxxxxxxxxx... Pack... Int
[89] xxxxxxxxxxxxxxxxxxx... Pack... Int
[90] 00000055 Pack... Int
[91] xxxxxxxxxxxxxxxxxxx... Pack... Int
[92] xxxxxxxxxxxxxxxxxxx... Pack... Int
[93] xxxxxxxxxxxxxxxxxxx... Pack... Int
[94] xxxxxxxxxxxxxxxxxxx... Pack... Int
[95] xxxxxxxxxxxxxxxxxxx... Pack... Int
[96] xxxxxxxxxxxxxxxxxxx... Pack... Int
[97] xxxxxxxxxxxxxxxxxxx... Pack... Int
[98] xxxxxxxxxxxxxxxxxxx... Pack... Int
[99] xxxxxxxxxxxxxxxxxxx... Pack... Int
[100] xxxxxxxxxxxxxxxxxxx... Pack... Int
[101] xxxxxxxxxxxxxxxxxxx... Pack... Int
[102] xxxxxxxxxxxxxxxxxxx... Pack... Int
Processes (Active)
Library
sim
Transcript
Break in Module st_instr_tb at C:/Users/15jdl4/Documents/374/st_instr_tb.v line 32

st $90(R1), R1


sim - Default
Instance
Design unit
Design unit type
Visibility
Total co
r14 Register32 Module +acc=<...
r15 Register32 Module +acc=<...
pc Register32 Module +acc=<...
ir Register32 Module +acc=<...
mar Register32 Module +acc=<...
y Register32 Module +acc=<...
hi Register32 Module +acc=<...
lo Register32 Module +acc=<...
z Register64 Module +acc=<...
mdr MDR_unit Module +acc=<...
bus Bus32bit Module +acc=<...
alu Alu Module +acc=<...
sel_enc Sel_Enc Module +acc=<...
conff conff_logic Module +acc=<...
inport Inport Module +acc=<...
outport Outport Module +acc=<...
ram RAM Module +acc=<...
#ASSIGN#6... Datapath Process +acc=<...
#ASSIGN#7... Datapath Process +acc=<...
#INITIAL#24 st_instr_tb Process +acc=<...
#INITIAL#31 st_instr_tb Process +acc=<...
#ALWAYS#35 st_instr_tb Process +acc=<...
#ALWAYS#51 st_instr_tb Process +acc=<...
#vsim_capacity# Capacity +acc=<...
Library
sim
Objects
Name
Value
Kind
read St1 Net In
write St0 Net In
data 0000000000000000000... Net Inc
addr 010101111 Net In
mem 00010000100010000000... Fixed... Int
[0] 1088005a Pack... Int
[1] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[2] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[3] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[4] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[5] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[6] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[7] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[8] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[9] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[10] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[11] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[12] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[13] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[14] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[15] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[16] xxxxxxxxxxxxxxxxxxxx... Pack... Int
Processes (Active)
Search:
[168] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[169] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[170] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[171] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[172] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[173] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[174] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[175] 00000055 Pack... Int
[176] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[177] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[178] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[179] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[180] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[181] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[182] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[183] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[184] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[185] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[186] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[187] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[188] xxxxxxxxxxxxxxxxxxxx... Pack... Int
[189] xxxxxxxxxxxxxxxxxxxx... Pack... Int