374 Phase 2 Report

March 24, 2023

Group 4:
Liam Salass (20229595)
Abdellah Ghassel (20230384)

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

Table of Contents

Introduction & Motivation 4

Updated Datapath, Revised modules and New Logic 5

Datapath 5

PC Register 11

Zero Register 12

Updated ALU 13

CON_FF Logic Module 15

Select & Encode Logic 17

RAM Module 19

Testbenches 20

Load Instructions – ld and ldi 20

Case 1: ld R1, $75 20

Case 2: ld R0, $45(R1) 27

Case 3: ldi R1, $75 33

Case 4: ldi R0, $45(R1) 39

Store Instruction 45

Case 1: st $90, R4 45

Case 2: st $90(R4), R4 51

ALU Immediate Instructions – addi, andi, ori 56

Addi R2, R3, -3 56

Andi R2,R3, $25 61

Ori R2, R3, $25 66

Branch Instructions – brzr, brnz, brpl, brim 71

Case 1: brzr R6, 25 71

Case 2: brnz R6, 25 76

Case 3: brpl R6, 25 81

Case 4: brmi R6, 25 86

Jump Instructions – jr, jal 91

Jr R2 91

Jal R2 96

Special Instructions – mfhi and mflo 101

Mfhi 101

Mflo 106
Input/Output Instructions – in, out 111
Out R2 111
In R3 116

Table of Figures

Figure 1: ld R1, $75 (Address 75 stores value 8) 26
Figure 2: ld R0, $45(R1) 32
Figure 3: ldi R1, $75 38
Figure 4: ldi R0, $45(R1) 44
Figure 5: RAM contents. Note at RAM Address 144, value 103 is stored. Explained in initial lines of testbench. 50
Figure 6: st $90, R4. Note, output Q in R4 is in hex for verification purposes. 50
Figure 7: st $90(R4), R4. RAM Address 247, value 103 is stored. 55
Figure 8: RAM contents. Note at RAM Address 247, value 103 is stored. Explained in testbench. 55
Figure 9: addi R2, R3, -3. 60
Figure 10: andi R2, R3, $25 (and operation of R3 with value 5 and $25 returns 5 in R2) 65
Figure 11: ori R2, R3, $25 70
Figure 12: brzr R6, 25 75
Figure 13: brnz R6, 25 80
Figure 14: brpl R6, 25 85
Figure 15: brmi R6, 25 90
Figure 16: jr R2 95
Figure 17: jal R2 100
Figure 18: mfhi R4 105
Figure 19: mflo R6 (moves LO to R6) used value of 10 110
Figure 20: Out R2. Note, R2 holds 10, thus placing the value 10 in OutPort 115
Figure 21: in R3 120

Introduction & Motivation

This project aims to design, simulate, implement, and verify a simple RISC Computer (Mini SRC) which comprises a simple RISC processor, memory, and I/O. The focus of this report is to provide an in-depth analysis of Phase 2 of the project, which involves the design and functional simulation of the Mini SRC datapath in particular “Select and Encode” logic, “Memory Subsystem”, “CON FF” logic, and “Input/Output” ports, as well as load/store instructions, branch and jump instructions, and immediate instructions.

The primary motivation behind this project is to gain a comprehensive understanding of the intricacies involved in designing a RISC-based computing system. Additionally, the project aims to provide valuable insights into the integration of various datapath components, which are essential for the overall functioning of the Mini SRC.

Updated Datapath, Revised modules and New Logic

Datapath

The datapath was modified to add in new specific registers for the MDR, R0, and PC. The ALU was also modified to receive new signals for CONN_FF logic and incPC. CONN_FF checks to see if the branching condition is valid, and if so, adds two values together in the ALU. incPC is a signal that goes to the ALU and increments any value inputted into it from the BusMuxOut wire. The new Select and Encode logic is used for determining which registers to use depending on the IR opcode. It also sign extends the C value to ensure it is a 32 bit number.

```
module datapath (
    input clr, clk,
    input read, write,
    input BAout, Rin, Rout,
    input Gra, Grb, Grc,
    input CONN_in,
    input MARin, MDRin,
    input HIin, LOin,
    input Yin, Zin,
    input PCin, IRin, incPC,
    input InPortIn, OutPortIn,
    input HIout, LOout, ZLowOut, ZHighOut,
    input MDRout, Cout, InPortOut, PCout,
    input [4:0] opcode,
    input [31:0] InPortData
);

wire R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out,
R10out, R11out, R12out, R13out, R14out, R15out;
wire R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in, R8in, R9in, R10in, R11in,
R12in, R13in, R14in, R15in;
wire Control_unit_in;

wire [31:0] BusMuxIn_R0,
            BusMuxIn_R1,
            BusMuxIn_R2,
            BusMuxIn_R3,
            BusMuxIn_R4,
            BusMuxIn_R5,
            BusMuxIn_R6,
            BusMuxIn_R7,
            BusMuxIn_R8,
            BusMuxIn_R9,
            BusMuxIn_R10,
            BusMuxIn_R11,
```

```
            BusMuxIn_R12,
            BusMuxIn_R13,
            BusMuxIn_R14,
            BusMuxIn_R15;

wire [31:0] BusMuxIn_HI,
            BusMuxIn_LO,
            BusMuxIn_Zhigh,
            BusMuxIn_Zlow,
            BusMuxIn_PC,
            BusMuxIn_MDR,
            BusMuxIn_InPort,
            RamDataOut,
            Yout,
            d_pc,
            MARout,
            BusMuxOut,
            C_sign_extended, //Csign extended for branching
            IRdata, //Output of IR reg
            OutPortOut; //Wire for outport. Goes no where

wire [63:0] CRegOut;

regR0 R0 (BAout, clr, clk, R0in, BusMuxOut, BusMuxIn_R0); //input signal is
always 0 for R0 (special reg)
reg32bit #(32'd5) R1 (clr, clk, R1in, BusMuxOut, BusMuxIn_R1);
reg32bit #(32'd10) R2 (clr, clk, R2in, BusMuxOut, BusMuxIn_R2);
reg32bit #(32'd5) R3 (clr, clk, R3in, BusMuxOut, BusMuxIn_R3);
reg32bit #(32'h67) R4 (clr, clk, R4in, BusMuxOut, BusMuxIn_R4);
reg32bit R5 (clr, clk, R5in, BusMuxOut, BusMuxIn_R5);
reg32bit #(32'd1) R6 (clr, clk, R6in, BusMuxOut, BusMuxIn_R6);
reg32bit R7 (clr, clk, R7in, BusMuxOut, BusMuxIn_R7);
reg32bit R8 (clr, clk, R8in, BusMuxOut, BusMuxIn_R8);
reg32bit R9 (clr, clk, R9in, BusMuxOut, BusMuxIn_R9);
reg32bit R10 (clr, clk, R10in, BusMuxOut, BusMuxIn_R10);
reg32bit R11 (clr, clk, R11in, BusMuxOut, BusMuxIn_R11);
reg32bit R12 (clr, clk, R12in, BusMuxOut, BusMuxIn_R12);
reg32bit R13 (clr, clk, R13in, BusMuxOut, BusMuxIn_R13);
reg32bit R14 (clr, clk, R14in, BusMuxOut,  BusMuxIn_R14);
reg32bit R15 (clr, clk, R15in, BusMuxOut,  BusMuxIn_R15);

reg32bit HI (clr, clk, HIin, BusMuxOut, BusMuxIn_HI);
reg32bit #(32'd10) LO (clr, clk, LOin, BusMuxOut, BusMuxIn_LO);
reg32bit #(32'd100) ZHigh (clr, clk, Zin, CRegOut[63:32], BusMuxIn_Zhigh);
reg32bit #(32'd75) ZLow (clr, clk, Zin, CRegOut[31:0], BusMuxIn_Zlow);
```

```
//PC reg initialization, using specific incPC input to increment PC by 1 each
time set to high
regPC PC (clr, clk, PCin, BusMuxOut, BusMuxIn_PC);

//Input and output ports added to datapath (p2)
reg32bit InPort (clr, clk, InPortIn, InPortData, BusMuxIn_InPort);
reg32bit OutPort (clr, clk, OutPortIn, BusMuxOut, OutPortOut);

//MDR reg initialization

MD_reg32 MDR (.clr(clr), .clk(clk), .read(read), .MDRin(MDRin),
.BusMuxOut(BusMuxOut), .Mdatain(RamDataOut), .Q(BusMuxIn_MDR)); //special MDR reg
reg32bit MAR (clr, clk, MARin, BusMuxOut, MARout);

// Goes into ALU A input
reg32bit Y (clr, clk, Yin, BusMuxOut, Yout);

//Instruction register. IRdata doesn't go on the bus, but leads to CON
reg32bit IR (clr, clk, IRin, BusMuxOut, IRdata);

//Memory initialization
//RamDataOut used as MDR input since, using BusMuxIn_MDR would have two registers
writing to the same input.
ram myRam (.clk(clk), .read(read), .write(write), .MARout(MARout[8:0]),
.D(BusMuxIn_MDR), .Q(RamDataOut));

//Control Branch logic
CONN_FF myConn_ff (
    .IRin(IRdata[20:19]),
    .BusMuxOut(BusMuxOut),
    .CONN_in(CONN_in),
    .CONN_out(Control_unit_in)
);

//Select and Encode logic for selecting register functions based on opcode
select_and_encode mySAE (
    .irOut(IRdata),
    .Gra(Gra),
    .Grb(Grb),
    .Grc(Grc),
    .Rin(Rin),
    .Rout(Rout),
    .BAout(BAout),
```

```
    .R0in(R0in),
    .R1in(R1in),
    .R2in(R2in),
    .R3in(R3in),
    .R4in(R4in),
    .R5in(R5in),
    .R6in(R6in),
    .R7in(R7in),
    .R8in(R8in),
    .R9in(R9in),
    .R10in(R10in),
    .R11in(R11in),
    .R12in(R12in),
    .R13in(R13in),
    .R14in(R14in),
    .R15in(R15in),
    .R0out(R0out),
    .R1out(R1out),
    .R2out(R2out),
    .R3out(R3out),
    .R4out(R4out),
    .R5out(R5out),
    .R6out(R6out),
    .R7out(R7out),
    .R8out(R8out),
    .R9out(R9out),
    .R10out(R10out),
    .R11out(R11out),
    .R12out(R12out),
    .R13out(R13out),
    .R14out(R14out),
    .R15out(R15out),
    .C_sign_extended(C_sign_extended)
);

//bus
bus myBus (
    //encoder
    .R0out(R0out),
    .R1out(R1out),
    .R2out(R2out),
    .R3out(R3out),
    .R4out(R4out),
    .R5out(R5out),
    .R6out(R6out),
```

```
    .R7out(R7out),
    .R8out(R8out),
    .R9out(R9out),
    .R10out(R10out),
    .R11out(R11out),
    .R12out(R12out),
    .R13out(R13out),
    .R14out(R14out),
    .R15out(R15out),
    .HIout(HIout),
    .LOout(LOout),
    .ZHighOut(ZHighOut),
    .ZLowOut(ZLowOut),
    .PCout(PCout),
    .MDRout(MDRout),
    .InPortOut(InPortOut),
    .Cout(Cout),
    //multiplexer
    .BusMuxIn_R0(BusMuxIn_R0),
    .BusMuxIn_R1(BusMuxIn_R1),
    .BusMuxIn_R2(BusMuxIn_R2),
    .BusMuxIn_R3(BusMuxIn_R3),
    .BusMuxIn_R4(BusMuxIn_R4),
    .BusMuxIn_R5(BusMuxIn_R5),
    .BusMuxIn_R6(BusMuxIn_R6),
    .BusMuxIn_R7(BusMuxIn_R7),
    .BusMuxIn_R8(BusMuxIn_R8),
    .BusMuxIn_R9(BusMuxIn_R9),
    .BusMuxIn_R10(BusMuxIn_R10),
    .BusMuxIn_R11(BusMuxIn_R11),
    .BusMuxIn_R12(BusMuxIn_R12),
    .BusMuxIn_R13(BusMuxIn_R13),
    .BusMuxIn_R14(BusMuxIn_R14),
    .BusMuxIn_R15(BusMuxIn_R15),
    .BusMuxIn_HI(BusMuxIn_HI),
    .BusMuxIn_LO(BusMuxIn_LO),
    .BusMuxIn_Zhigh(BusMuxIn_Zhigh),
    .BusMuxIn_Zlow(BusMuxIn_Zlow),
    .BusMuxIn_PC(BusMuxIn_PC),
    .BusMuxIn_MDR(BusMuxIn_MDR),
    .BusMuxIn_InPort(BusMuxIn_InPort),
    .C_sign_extended(C_sign_extended),
    .BusMuxOut(BusMuxOut)
);
```

```
//alu
alu_test myAlu(
    .clk(clk),
    .clr(clr),
    .incPC(incPC),
    .CONN_out(Control_unit_in),
    .B(BusMuxOut),
    .A(Yout),
    .opcode(opcode),
    .C(CRegOut)
    );
endmodule
```

PC Register

This is a special register used for the Program Counter (PC).

```
module regPC #(parameter qInitial = 0)(clr, clk, enable, D, Q);
    input wire clr, clk, enable;
    input wire [31:0] D;
    output reg [31:0] Q;

    initial Q = qInitial;

    //Use posedge clk and check for incPC inside the always block
    always @(posedge clk) begin
        if (clr) //If clr is high set to 0
            Q <= 0;
        else if (enable) //If enable is high, read value from bus to Q
            Q <= D;
    end
endmodule
```

Zero Register

Modified R0 to incorporate BAout logic. The register was modified to output a guaranteed 0 if BAout is high. This allows the register to consistently place 0 on the BusMux, but can still hold a value that isn't 0 within it.

```
module regR0(BAout, clr, clk, enable, D, Q);
    input wire BAout, clr, clk, enable;
    input wire [31:0] D;
    output wire [31:0] Q;

    //At positive clock edge begin
    reg [31:0] regout;

    always @(posedge clk) begin
        if (clr) //If clr is high set to 0
            regout <= 0;
        else if (enable) //If enable is high, read value from bus to Q
            regout <= D;
    end

    assign Q = BAout ? 32'b0 : regout;

endmodule
```

Updated ALU

Alu was changed to have specific functionality if the incPC or CONN_out signals were high. Those were that if incPC was high, the value on the bus would be incremented. Similarly, if CONN_out was high, then the current PC value and the offset are added together in order to branch.

```
`timescale 1ns/10ps

module alu (A, B, opcode, clk, clr, incPC, CONN_out, control, C);
    input wire [31:0] A, B, control;
    input wire clk, clr, incPC, CONN_out;
    input wire [4:0] opcode;
    output reg [63:0] C;


    wire[31:0] neg_reg, nor_reg, or_reg, not_reg, xor_reg, and_reg, shl_reg,
shr_reg, shra_reg, ror_reg, rol_reg, add_reg, sub_reg,
    add_cout_reg, sub_cout_reg;
    wire [63:0] mul_reg, div_reg;


    parameter add = 5'b00001, sub = 5'b00010, mul = 5'b00011, div = 5'b00100, shr
= 5'b00101, shl = 5'b00110, shra = 5'b00111,
                    ror = 5'b01000, rol = 5'b01001, log_and = 5'b01010, log_or =
5'b01011, log_neg = 5'b01100, log_xor = 5'b01101,
                    log_nor = 5'b01110, log_not = 5'b01111;

    cla32bit addop (.a(A), .b(B), .cin(1'd0), .s(add_reg),
.cout(add_cout_reg[0]));
    sub32 subop (.a(A), .b(B), .cin(1'd0), .s(sub_reg), .cout(sub_cout_reg));
    mul32 mulop (.multiplicand(A), .multiplier(B), .product(mul_reg));
    div32 divop (.D(A), .O(B), .Q(div_reg[63:32]), .R(div_reg[31:0]));
    and32 andop (.a(A), .b(B), .c(and_reg));
    or32 orop (.a(A), .b(B), .c(or_reg));
    xor32 xorop (.a(A), .b(B), .c(xor_reg));
    neg32 negop (.in(B), .out(neg_reg));
    nor32 norop (.a(A), .b(B), .c(nor_reg));
    not32 notop (.in(B), .out(not_reg));
    shl32 shlop (B, A, shl_reg);
    shr32 shrop (B, A, shr_reg);
    shra32 shraop (B, A, shra_reg);
    ror32 rorop (B, A, ror_reg);
    rol32 rolop (B, A, rol_reg);
```

```
    always @(posedge clk) begin
        //if (opcode) begin -- don't need this since ur using cases
            case(opcode)
                add: begin
                    C [31:0] <= add_reg[31:0];
                    C [63:32] <= 32'd0;
                end
                sub: begin
                    C [31:0] <= sub_reg [31:0];
                    C [63:32] <= 32'd0;
                end
                mul: begin
                    C [63:0] <= mul_reg [63:0];
                end
                div: begin
                    C [63:0] <= div_reg[63:0];
                end
                shr: begin
                    C [31:0] <= shr_reg[31:0];
                    C [63:32] <= 32'd0;
                end
                shl: begin
                    C [31:0] <= shl_reg[31:0];
                    C [63:32] <= 32'd0;
                end
                shra: begin
                    C [31:0] <= shra_reg[31:0];
                    C [63:32] <= 32'd0;
                end
                ror: begin
                    C [31:0] <= ror_reg[31:0];
                    C [63:32] <= 32'd0;
                end
                rol: begin
                    C [31:0] <= rol_reg[31:0];
                    C [63:32] <= 32'd0;
                end
                log_and: begin
                    C [31:0] <= and_reg[31:0];
                    C [63:32] <= 32'd0;
                end
                log_or: begin
                    C [31:0] <= or_reg[31:0];
                    C [63:32] <= 32'd0;
                end
```

```
                log_neg: begin
                    C [31:0] <= neg_reg[31:0];
                    C [63:32] <= 32'd0;
                end
                log_xor: begin
                    C [31:0] <= xor_reg[31:0];
                    C [63:32] <= 32'd0;
                end
                log_nor: begin
                    C [31:0] <= nor_reg[31:0];
                    C [63:32] <= 32'd0;
                end
                log_not: begin
                    C [31:0] <= not_reg[31:0];
                    C [63:32] <= 32'd0;
                end
                default: begin
                    C <= 64'd0;
                end
            endcase
            if (incPC) begin
                C <= B + 1;
            end
            if (CONN_out) begin
                C <= A + B;
            end
        end
    //end
endmodule
```

CON_FF Logic Module

The code below checks the branching conditions in the IR register and compares the Ra register value to the branching condition. If the value meets the branching criteria, it then sends a high signal to the ALU to add the offset to the PC.

```
module CONN_FF (
    input [1:0] IRin,
    input [31:0] BusMuxOut,
    input wire CONN_in,
    output reg CONN_out
);
    wire nor_bus, msb_bus;
     reg D;

    //Nor gate all bus inputs
```

```
    assign nor_bus = BusMuxOut == 0 ? 1'b1 : 1'b0;

    //Get MSB of bus
    assign msb_bus = BusMuxOut[31];

    //2 to 4 decoder and value a
    always @(*) begin
        case(IRin)
            2'b00: D = nor_bus;
            2'b01: D = ~nor_bus;
            2'b10: D = ~msb_bus;
            2'b11: D = msb_bus;
        endcase

        if(CONN_in)
            CONN_out = D;
        else
            CONN_out = 1'b0;
    end
endmodule
```

Select & Encode Logic

The code below partitions the register opcodes from the IR register. It can then be used along with Gra, Grb, and Grc to select if we are reading into or outputting from a specific register. The select and encode logic also sign extends the C value in the IR out. The Select and Encode uses a 4 to 15 decoder for selecting the register for Ra, Rb, and Rc. Rout and Rin signals determine if the register is outputting or inputting a value. If outputting, the corresponding R#out signal is sent to the BusMux. The corresponding R#in signal goes to the corresponding R# registers input signal.

```
module select_and_encode (
    input [31:0] irOut,
    input Gra, Grb, Grc,
    input Rin, Rout, BAout,
    output R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in, R8in, R9in, R10in,
R11in, R12in, R13in, R14in, R15in,
    output R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out,
R10out, R11out, R12out, R13out,
    R14out, R15out,
    output [31:0] C_sign_extended
);
    wire Rout_or;
    wire [3:0] Ra, Rb, Rc, gra_and, grb_and, grc_and, dec_in;
    wire [15:0] dec_out;

    //Decoder initialization
    decoder4to16 myDecoder (.in(dec_in), .out(dec_out));

    //Partition IR reg output
    assign Ra = irOut[26:23];
    assign Rb = irOut[22:19];
    assign Rc = irOut[18:15];

    //If Gr is high, create 4 bit high, else low for anding later
    assign gra_and = Gra == 1 ? 4'b1111 : 4'b0000;
    assign grb_and = Grb == 1 ? 4'b1111 : 4'b0000;
    assign grc_and = Grc == 1 ? 4'b1111 : 4'b0000;

    //And Ra, Rb, Rc and w Gr input, then or values and input to decoder
    assign dec_in = (gra_and & Ra) | (grb_and & Rb) | (grc_and & Rc);

    //Or Rout and BAout, then covert to 16 bit for and'ing
    assign Rout_or = Rout | BAout;

    //Select wich output signal to send.
    assign R0in = dec_out[0] & Rin;
    assign R1in = dec_out[1] & Rin;
```

```
    assign R2in = dec_out[2] & Rin;
    assign R3in = dec_out[3] & Rin;
    assign R4in = dec_out[4] & Rin;
    assign R5in = dec_out[5] & Rin;
    assign R6in = dec_out[6] & Rin;
    assign R7in = dec_out[7] & Rin;
    assign R8in = dec_out[8] & Rin;
    assign R9in = dec_out[9] & Rin;
    assign R10in = dec_out[10] & Rin;
    assign R11in = dec_out[11] & Rin;
    assign R12in = dec_out[12] & Rin;
    assign R13in = dec_out[13] & Rin;
    assign R14in = dec_out[14] & Rin;
    assign R15in = dec_out[15] & Rin;

    assign R0out = dec_out[0] & Rout_or;
    assign R1out = dec_out[1] & Rout_or;
    assign R2out = dec_out[2] & Rout_or;
    assign R3out = dec_out[3] & Rout_or;
    assign R4out = dec_out[4] & Rout_or;
    assign R5out = dec_out[5] & Rout_or;
    assign R6out = dec_out[6] & Rout_or;
    assign R7out = dec_out[7] & Rout_or;
    assign R8out = dec_out[8] & Rout_or;
    assign R9out = dec_out[9] & Rout_or;
    assign R10out = dec_out[10] & Rout_or;
    assign R11out = dec_out[11] & Rout_or;
    assign R12out = dec_out[12] & Rout_or;
    assign R13out = dec_out[13] & Rout_or;
    assign R14out = dec_out[14] & Rout_or;
    assign R15out = dec_out[15] & Rout_or;

    assign C_sign_extended = irOut[17] == 1 ? {14'b11111111111111, irOut[17:0]} :
{14'b0, irOut[17:0]};
endmodule
```

RAM Module

The below architecture for the uses synchronous functionality. The MARout is used to point to locations in the memory, while D is used to write to the memory, and Q is the memory output. The ram.hex file was used for initializing values within the RAM. We used the `mem[0] = opcode` was used to make the first value in the memory have the required opcode for each testbench, and for simplicity sake.

```
//512 x 32 RAM
module ram(
    input clk, read, write,
    input [8:0] MARout,     //address
    input [31:0] D,
    output [31:0] Q
);
    reg [31:0] mem [511:0];

    initial $readmemh("ram.hex", mem);

    assign Q = (write || !read) ?  32'bZZZZZZZZ : mem[MARout];

   always @(posedge clk) begin
        mem[0] = 32'h9B100019; //opcode for branch case 3 testbench
        if(write) mem[MARout] = D;
    end
endmodule
```

Testbenches

The following are the testbenches requested for the lab organized by their respective categories.

All test benches use the same 3 cycles at the beginning for incrementing the PC and fetching the instruction and placing it in the IR:

1. Place PC on bus, Place PC value in MAR register to point to PC location in memory. Zin high to read PC value into ALU for incrementing. incPC set high to give ALU PC increment signal.
2. ALU increments PC value. ZLowOut set high to place incremented PC value on bus. PCin set high to place new PC value in PC register. Read and MDRin set to high so we fetch the instruction from memory from the location specified by the MAR register. This value is placed in the MDR register.
3. MDRout set high to place instruction on the bus. IRin high to place value into IR register.

Load Instructions – ld and ldi

Our load instruction test benches functioned as followed:

4. Grb and About are set high and placed in Y register to be added as the offset to the address we are loading from
5. ALU given ADD opcode, Cout set high to place specified address in Bus for computing address offset with ALU.
6. ZLowOut set high and MARin set high to place ALU output into MAR register to point the memory to the location we wish to load from. Read is set high so that the value is placed into the MDR register.
7. MDRout, Gra and Rin set high so that the value from the memory is placed in Ra

If the instruction is a load immediate value operation, two less cycles are needed since we don't fetch the value from the memory. Rather the immediate value and the offset are added in the ALU and then placed in Ra.

Case 1: ld R1, $75

```
`timescale 1ns/1ps

module tb_load_case1;

    reg clr, clk;
   reg read, write;
   reg BAout, Rin, Rout;
   reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
   reg HIin, LOin;
```

```
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
   reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
```

```
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
              T5         = 4'b1100,
              T6         = 4'b1101,
              T7         = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = Reg_load1a;
            Reg_load1a : #40 present_state = Reg_load1b;
            Reg_load1b : #40 present_state = Reg_load2a;
            Reg_load2a : #40 present_state = Reg_load2b;
```

```
            Reg_load2b : #40 present_state = Reg_load3a;
            Reg_load3a : #40 present_state = Reg_load3b;
            Reg_load3b : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
            T3         : #40 present_state = T4;
            T4         : #40 present_state = T5;
            T5         : #40 present_state = T6;
            T6         : #40 present_state = T7;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
```

```
                opcode = 0;
                clr = 0;
            end

            Reg_load1a: begin
                // Load data into inport
                #10 InPortData = 32'h75; InPortIn = 1;
                #15 InPortData = 32'hx; InPortIn = 0;
            end

            Reg_load1b: begin
                // Load data from inport into MAR register for address 75
                #10 InPortOut = 1; MARin = 1;
                #15 InPortOut = 0; MARin = 0;
            end

            Reg_load2a: begin
                // Load data into inport
                #10 InPortData = 32'h8; InPortIn = 1;
                #15 InPortData = 32'hx; InPortIn = 0;
            end

            Reg_load2b: begin
                // Load data from inport into MDR register
                #10 InPortOut = 1; MDRin = 1;
                #15 InPortOut = 0; MDRin = 0;
            end

            Reg_load3a: begin
                // Write 0x00000008 to address 0x75 in memory and load 0 into
INPORT
                #10 write = 1; InPortData = 32'h0; InPortIn = 1;
                #15 write = 0; InPortIn = 0;
            end

            Reg_load3b: begin
                // Write 0 to r0 and pc reg
                #10 InPortOut = 1; Rin = 1; PCin = 1;
                #15 InPortOut = 0; Rin = 0; PCin = 0;

            end

            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
```

```
            end

            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end

            T3: begin
                #10 Grb = 1; BAout = 1; Yin = 1; // R0
                #15 Grb = 0; BAout = 0; Yin = 0;
            end

            T4: begin
                #10 Cout = 1; Zin = 1; opcode = add; // ADD
                #15 Cout = 0; Zin = 0; opcode = nop;
            end

            T5: begin
                #10 ZLowOut = 1; MARin = 1; // Adding 0 to mem offset so 0x75
                #15 ZLowOut = 0; MARin = 0;
            end

            T6: begin
                #10 read = 1; MDRin = 1;
                #15 read = 0; MDRin = 0;
            end

            T7: begin
                #10 MDRout = 1; Gra = 1; Rin = 1; // Write result to R1
                #15 MDRout = 0; Gra = 0; Rin = 0;
            end
        endcase

    end


endmodule
```

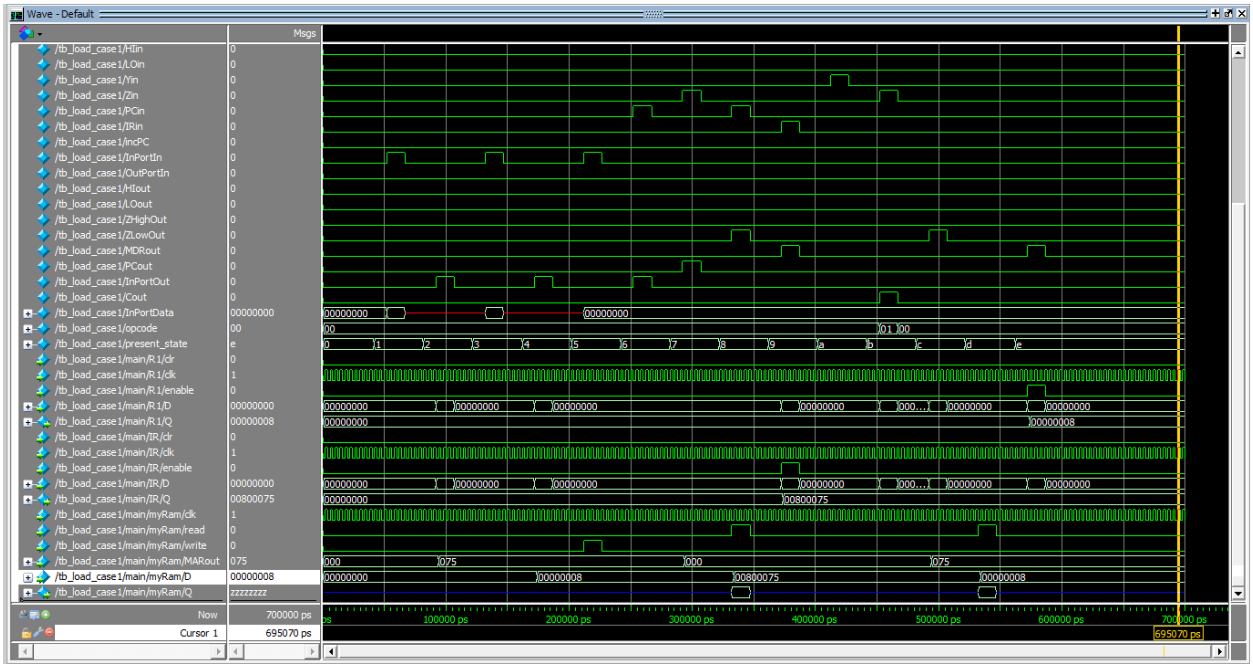


Figure 1: ld R1, $75 (Address 75 stores value 8)

Case 2: ld R0, $45(R1)

```
`timescale 1ns/1ps
//Does operation ld R0, $45(R1)
// R1 holds 5 so it will read from location 74 from memory and store it in R0
// IR opcode: h00080045

module tb_load_case2;

    reg clr, clk;
   reg read, write;
   reg BAout, Rin, Rout;
   reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
   reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
   reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
```

```
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
              T5         = 4'b1100,
              T6         = 4'b1101,
```

```
            T7          = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = Reg_load1a;
            Reg_load1a : #40 present_state = Reg_load1b;
            Reg_load1b : #40 present_state = Reg_load2a;
            Reg_load2a : #40 present_state = Reg_load2b;
            Reg_load2b : #40 present_state = Reg_load3a;
            Reg_load3a : #40 present_state = Reg_load3b;
            Reg_load3b : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
            T3         : #40 present_state = T4;
            T4         : #40 present_state = T5;
            T5         : #40 present_state = T6;
            T6         : #40 present_state = T7;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
```

```
            HIout = 0;
            LOout = 0;
            ZHighOut = 0;
            ZLowOut = 0;
            PCout = 0;
            MDRout = 0;
            InPortOut = 0;
            Cout = 0;
            BAout = 0;
            Gra = 0;
            Grb = 0;
            Grc = 0;
            Rin = 0;
            Rout = 0;
            InPortData = 0;
            opcode = 0;
            clr = 0;
        end

        Reg_load1a: begin
            // Load data into inport
            #10 InPortData = 32'h4A; InPortIn = 1;
            #15 InPortData = 32'hx; InPortIn = 0;
        end

        Reg_load1b: begin
            // Load data from inport into MAR register for address 50
            #10 InPortOut = 1; MARin = 1;
            #15 InPortOut = 0; MARin = 0;
        end

        Reg_load2a: begin
            // Load data into inport
            #10 InPortData = 32'h00000032; InPortIn = 1;
            #15 InPortData = 32'hx; InPortIn = 0;
        end

        Reg_load2b: begin
            // Load data from inport into MDR register
            #10 InPortOut = 1; MDRin = 1;
            #15 InPortOut = 0; MDRin = 0;
        end

        Reg_load3a: begin
```

```
                // Write 0x00000008 to address 0x75 in memory and load 0 into
INPORT
                #10 write = 1; InPortData = 32'h0; InPortIn = 1;
                #15 write = 0; InPortIn = 0;
            end

            Reg_load3b: begin
                // Write 0 to r0 and pc reg
                #10 InPortOut = 1; Rin = 1; PCin = 1;
                #15 InPortOut = 0; Rin = 0; PCin = 0;

            end
            // R5 preinitialized to hold 5
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
            end

            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end

            T3: begin
                #10 Grb = 1; BAout = 1; Yin = 1; // R0
                #15 Grb = 0; BAout = 0; Yin = 0;
            end

            T4: begin
                #10 Cout = 1; Zin = 1; opcode = add; // ADD
                #15 Cout = 0; Zin = 0; opcode = nop;
            end

            T5: begin
                #10 ZLowOut = 1; MARin = 1; // Adding 0 to mem offset so 0x75
                #15 ZLowOut = 0; MARin = 0;
            end

            T6: begin
```

```
                #10 read = 1; MDRin = 1;
                #15 read = 0; MDRin = 0;
            end

            T7: begin
                #10 MDRout = 1; Gra = 1; Rin = 1; // Write result to R1
                #15 MDRout = 0; Gra = 0; Rin = 0;
            end
        endcase

    end


endmodule
```

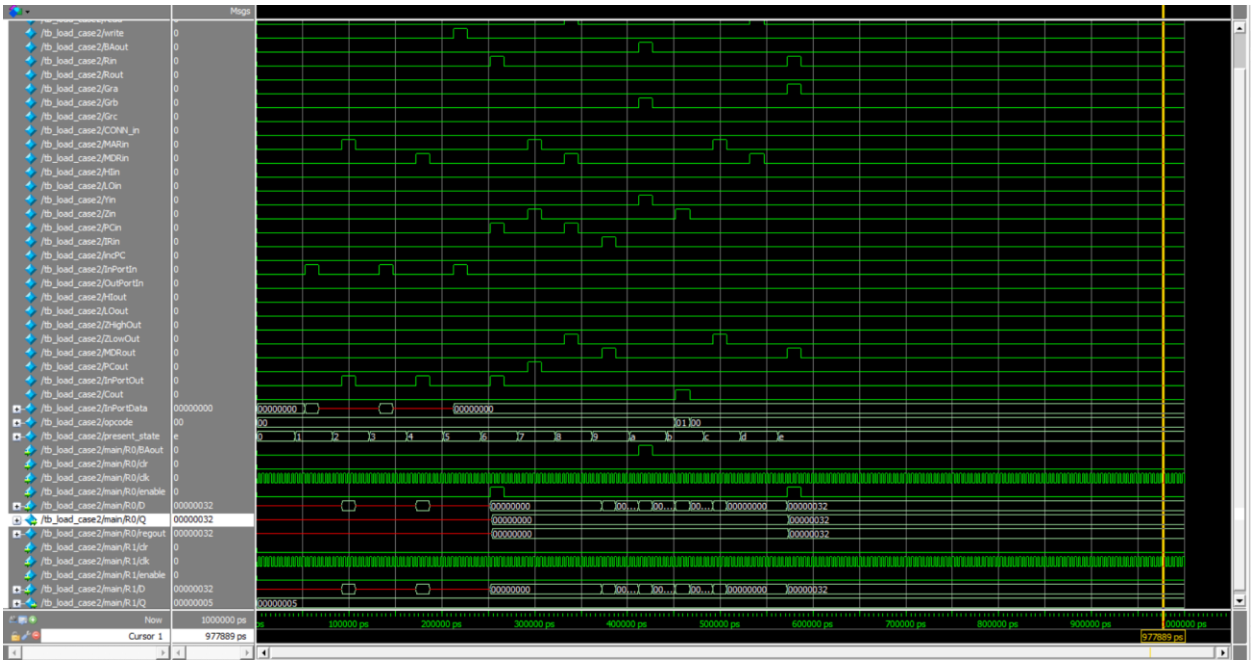


Figure 2: ld R0, $45(R1)

Case 3: ldi R1, $75

```
`timescale 1ns/1ps
//Does operation ldi R1, $75
// R1 will hold the value $75 which is 117 in decimal
// IR opcode: h08800075
module tb_load_case3;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
        .write(write),
```

```
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
              T5         = 4'b1100,
              T6         = 4'b1101,
              T7         = 4'b1110;
```

```
reg [3:0] present_state = Default;

initial begin
    clk = 0;
    forever #2 clk = ~clk;
end

always @(posedge clk) begin
    case (present_state)
        Default    : #40 present_state = Reg_load1a;
        Reg_load1a : #40 present_state = Reg_load1b;
        Reg_load1b : #40 present_state = Reg_load2a;
        Reg_load2a : #40 present_state = Reg_load2b;
        Reg_load2b : #40 present_state = Reg_load3a;
        Reg_load3a : #40 present_state = Reg_load3b;
        Reg_load3b : #40 present_state = T0;
        T0         : #40 present_state = T1;
        T1         : #40 present_state = T2;
        T2         : #40 present_state = T3;
        T3         : #40 present_state = T4;
        T4         : #40 present_state = T5;
    endcase
end

always @(present_state) begin
    case (present_state)
        Default : begin
            CONN_in = 0;
            MDRin = 0;
            MARin = 0;
            read = 0;
            write = 0;
            HIin = 0;
            LOin = 0;
            Yin = 0;
            Zin = 0;
            PCin = 0;
            IRin = 0;
            incPC = 0;
            InPortIn = 0;
            OutPortIn = 0;
            HIout = 0;
            LOout = 0;
            ZHighOut = 0;
```

```
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            Reg_load1a: begin
                // Load data into inport
                #10 InPortData = 32'h50; InPortIn = 1;
                #15 InPortData = 32'hx; InPortIn = 0;
            end

            Reg_load1b: begin
                // Load data from inport into MAR register for address 50
                #10 InPortOut = 1; MARin = 1;
                #15 InPortOut = 0; MARin = 0;
            end

            Reg_load2a: begin
                // Load data into inport
                #10 InPortData = 32'h00000032; InPortIn = 1;
                #15 InPortData = 32'hx; InPortIn = 0;
            end

            Reg_load2b: begin
                // Load data from inport into MDR register
                #10 InPortOut = 1; MDRin = 1;
                #15 InPortOut = 0; MDRin = 0;
            end

            Reg_load3a: begin
                // Write 0x00000008 to address 0x75 in memory and load 0 into
INPORT
                #10 write = 1; InPortData = 32'h0; InPortIn = 1;
                #15 write = 0; InPortIn = 0;
```

```
            end

            Reg_load3b: begin
                // Write 0 to r0 and pc reg
                #10 InPortOut = 1; Rin = 1; PCin = 1;
                #15 InPortOut = 0; Rin = 0; PCin = 0;

            end
            // R5 preinitialized to hold 5
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
            end

            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end

            T3: begin
                #10 Grb = 1; BAout = 1; Yin = 1; // R0
                #15 Grb = 0; BAout = 0; Yin = 0;
            end

            T4: begin
                #10 Cout = 1; Zin = 1; opcode = add; // ADD
                #15 Cout = 0; Zin = 0; opcode = nop;
            end

            T5: begin
                #10 ZLowOut = 1; Gra = 1; Rin = 1;  // Adding 0 to mem offset so
0x75
                #15 ZLowOut = 0; Gra = 0; Rin = 0;
            end
        endcase

    end
```

```
endmodule
```

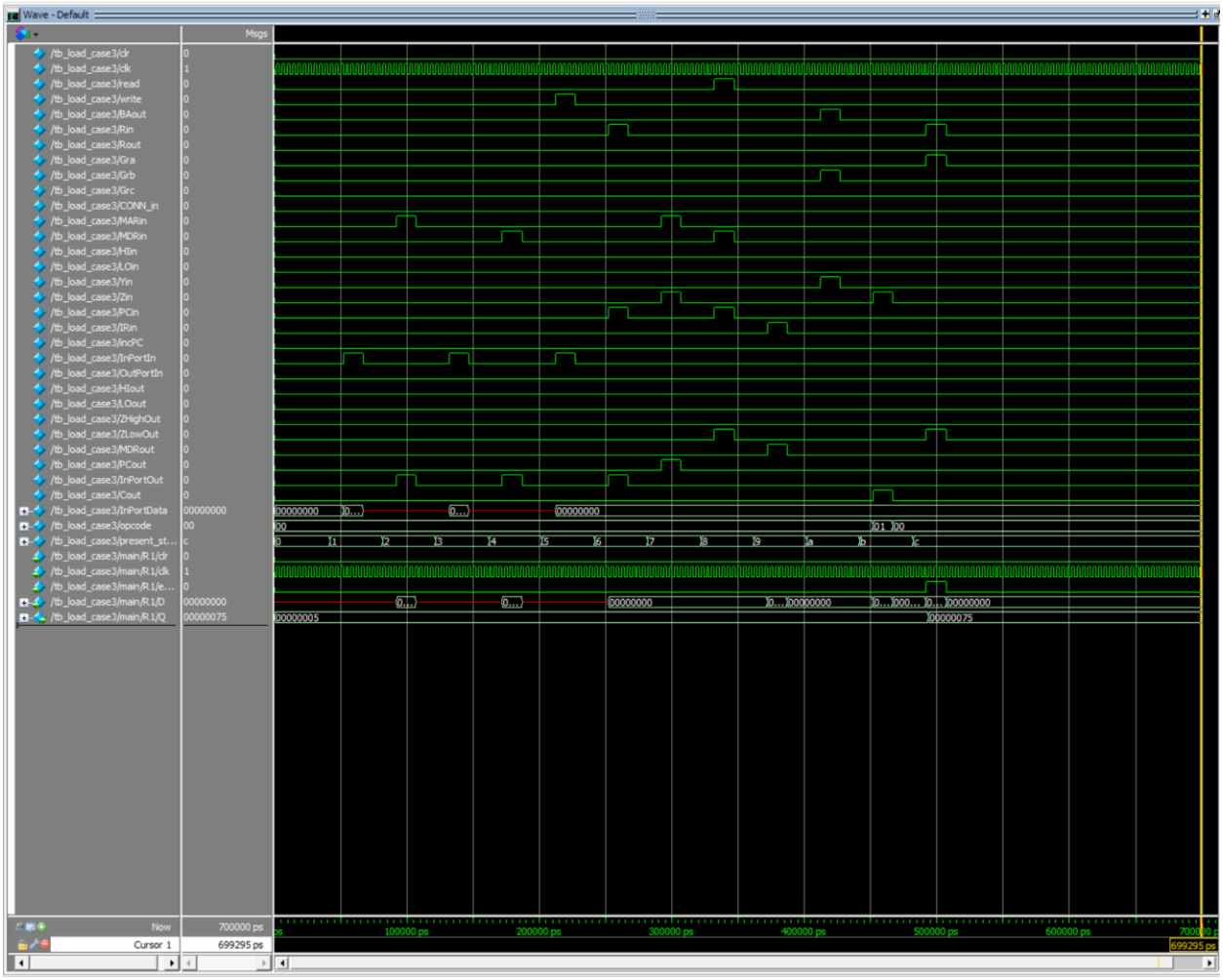


Figure 3: ldi R1, $75

Case 4: ldi R0, $45(R1)

```
`timescale 1ns/1ps
//Does operation ldi R0, $45(R1)
// R1 holds value 5 so the value in R0 should be $45 + 5 = $4A
// IR opcode: h08080045
module tb_load_case4;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
        .write(write),
```

```
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
              T5         = 4'b1100,
              T6         = 4'b1101,
              T7         = 4'b1110;
```

```
reg [3:0] present_state = Default;

initial begin
    clk = 0;
    forever #2 clk = ~clk;
end

always @(posedge clk) begin
    case (present_state)
        Default    : #40 present_state = Reg_load1a;
        Reg_load1a : #40 present_state = Reg_load1b;
        Reg_load1b : #40 present_state = Reg_load2a;
        Reg_load2a : #40 present_state = Reg_load2b;
        Reg_load2b : #40 present_state = Reg_load3a;
        Reg_load3a : #40 present_state = Reg_load3b;
        Reg_load3b : #40 present_state = T0;
        T0         : #40 present_state = T1;
        T1         : #40 present_state = T2;
        T2         : #40 present_state = T3;
        T3         : #40 present_state = T4;
        T4         : #40 present_state = T5;
    endcase
end

always @(present_state) begin
    case (present_state)
        Default : begin
            CONN_in = 0;
            MDRin = 0;
            MARin = 0;
            read = 0;
            write = 0;
            HIin = 0;
            LOin = 0;
            Yin = 0;
            Zin = 0;
            PCin = 0;
            IRin = 0;
            incPC = 0;
            InPortIn = 0;
            OutPortIn = 0;
            HIout = 0;
            LOout = 0;
            ZHighOut = 0;
```

```
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            Reg_load1a: begin
                // Load data into inport
                #10 InPortData = 32'h4a; InPortIn = 1;
                #15 InPortData = 32'hx; InPortIn = 0;
            end

            Reg_load1b: begin
                // Load data from inport into MAR register for address 50
                #10 InPortOut = 1; MARin = 1;
                #15 InPortOut = 0; MARin = 0;
            end

            Reg_load2a: begin
                // Load data into inport
                #10 InPortData = 32'd60; InPortIn = 1;
                #15 InPortData = 32'hx; InPortIn = 0;
            end

            Reg_load2b: begin
                // Load data from inport into MDR register
                #10 InPortOut = 1; MDRin = 1;
                #15 InPortOut = 0; MDRin = 0;
            end

            Reg_load3a: begin
                // Write 0x00000008 to address 0x75 in memory and load 0 into
INPORT
                #10 write = 1; InPortData = 32'h0; InPortIn = 1;
                #15 write = 0; InPortIn = 0;
```

```
            end

            Reg_load3b: begin
                // Write 0 to r0 and pc reg
                #10 InPortOut = 1; Rin = 1; PCin = 1;
                #15 InPortOut = 0; Rin = 0; PCin = 0;

            end
            //
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; incPC = 0;
            end

            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end

            T3: begin
                #10 Grb = 1; BAout = 1; Yin = 1; // R0
                #15 Grb = 0; BAout = 0; Yin = 0;
            end

            T4: begin
                #10 Cout = 1; Zin = 1; opcode = add; // ADD
                #15 Cout = 0; Zin = 0; opcode = nop;
            end

            T5: begin
                #10 ZLowOut = 1; Gra = 1; Rin = 1;
                #15 ZLowOut = 0; Gra = 0; Rin = 0;
            end
        endcase

    end


endmodule
```

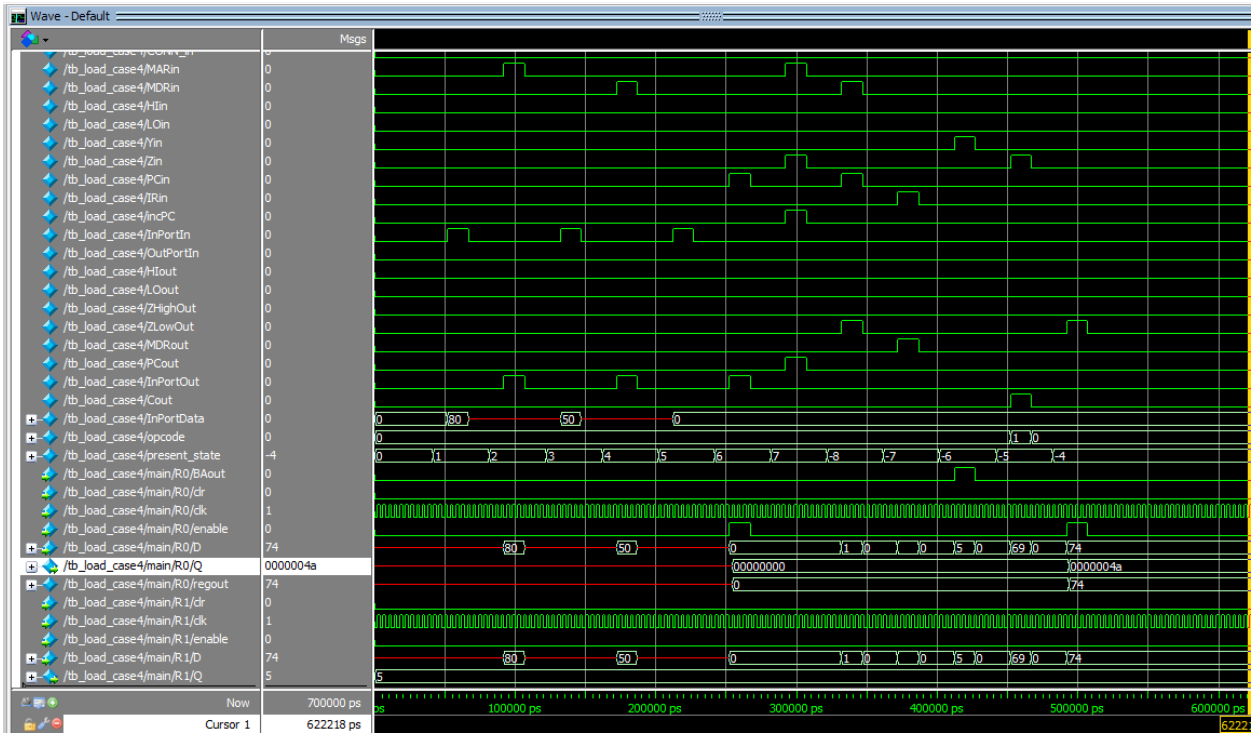


Figure 4: ldi R0, $45(R1)

Store Instruction

Store instruction use the same 3 first initialization cycles as the store instructions. The rest of them differ.

4. Place offset in the Y reg for addition with ALU in next cycle.
5. Add offset with C sign value
6. Place new address in MAR register
7. Write value in Ra to the location pointed to by MAR register in memory.

Case 1: st $90, R4

```
// Case 1: st $90, R4
// R4 = $67
// OPcode : 12000090
// $90 = 144 in decimal, so $67 = 103 in decimal is stored in position 144
`timescale 1ns/1ps
module tb_store_case1;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
```

```
              log_or = 5'b01011,
              log_neg = 5'b01100,
              log_xor = 5'b01101,
              log_nor = 5'b01110,
              log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
```

```
            Reg_load2b = 4'b0100,
            Reg_load3a = 4'b0101,
            Reg_load3b = 4'b0110,
            T0         = 4'b0111,
            T1         = 4'b1000,
            T2         = 4'b1001,
            T3         = 4'b1010,
            T4         = 4'b1011,
            T5         = 4'b1100,
            T6         = 4'b1101,
            T7         = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
            T3         : #40 present_state = T4;
            T4         : #40 present_state = T5;
            T5         : #40 present_state = T6;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
```

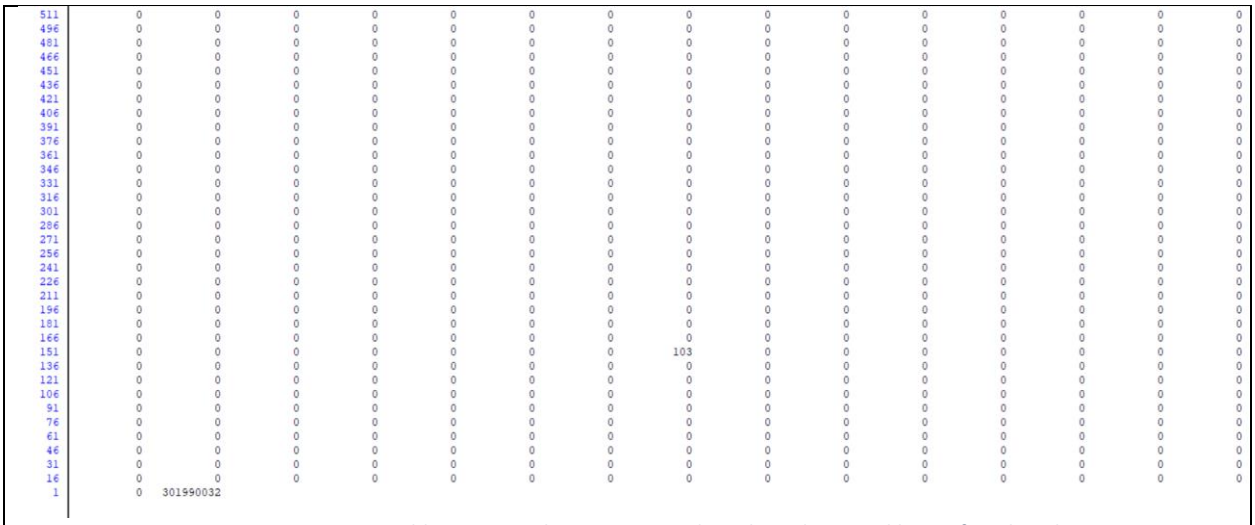
```
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            //Place PC in MAR to point to first instruction
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
            end

            //Read in value from MAR location in mem and place in MDR
            //Increment PC
            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end
            //Place MDR in IR
            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end
            //Place offset in ALU Y reg
            T3: begin
                #10 Grb = 1; BAout = 1; Yin = 1; // R0
                #15 Grb = 0; BAout = 0; Yin = 0;
            end
```

```
            //Add offset to C sign value
            T4: begin
                #10 Cout = 1; Zin = 1; opcode = add; // ADD
                #15 Cout = 0; Zin = 0; opcode = nop;
            end
            //Place new address in MAR
            T5: begin
                #10 ZLowOut = 1; MARin = 1; // Place value in MAR
                #15 ZLowOut = 0; MARin = 0;
            end
            //Write value in Ra to MDR
            T6: begin
                #10 write = 1; MDRin = 1; Gra = 1; Rout = 1; // Write result to
MEM
                #15 write = 0; MDRin = 0; Gra = 0; Rout = 0;
            end
        endcase

    end
endmodule
```



511	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
496	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
481	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
466	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
451	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
436	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
421	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
406	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
391	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
376	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
361	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
346	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
331	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
316	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
301	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
286	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
271	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
256	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
241	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
226	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
211	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
196	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
181	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
166	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
151	0	0	0	0	0	0	0	103	0	0	0	0	0	0	0
136	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
121	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
106	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
91	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
61	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	301990032													

Figure 5: RAM contents. Note at RAM Address 144, value 103 is stored. Explained in initial lines of testbench.

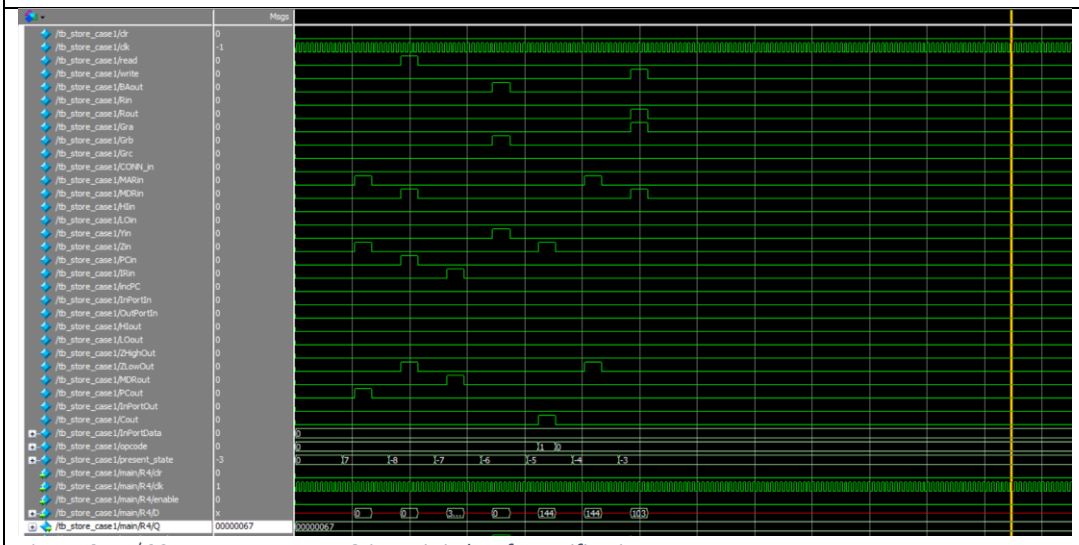


Figure 6: st $90, R4. Note, output Q in R4 is in hex for verification purposes.

Case 2: st $90(R4), R4

```
// Case 1: st $90(R4), R4
// R4 = $67
// OPcode : 12200090
// $90 = 144 in decimal, so $67 = 103 in decimal is stored in position 144 + 103
= 247.
`timescale 1ns/1ps
module tb_store_case2;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
```

```
        .read(read),
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
              T5         = 4'b1100,
```

```
            T6          = 4'b1101,
            T7          = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default     : #40 present_state = T0;
            T0          : #40 present_state = T1;
            T1          : #40 present_state = T2;
            T2          : #40 present_state = T3;
            T3          : #40 present_state = T4;
            T4          : #40 present_state = T5;
            T5          : #40 present_state = T6;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
```

```
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            //Place PC in MAR to point to first instruction
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
            end

            //Read in value from MAR location in mem and place in MDR
            //Increment PC
            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            //Place MDR in IR
            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end
            //Place offset in ALU Y reg
            T3: begin
                #10 Grb = 1; BAout = 1; Yin = 1; // R0
                #15 Grb = 0; BAout = 0; Yin = 0;
            end
            //Add offset to C sign value
            T4: begin
                #10 Cout = 1; Zin = 1; opcode = add; // ADD
                #15 Cout = 0; Zin = 0; opcode = nop;
            end
            //Place new address in MAR
            T5: begin
                #10 ZLowOut = 1; MARin = 1; // Place value in MAR
```

```
                #15 ZLowOut = 0; MARin = 0;
            end
            //Write value in Ra to MDR
            T6: begin
                #10 write = 1; MDRin = 1; Gra = 1; Rout = 1; // Write result to
MEM
                #15 write = 0; MDRin = 0; Gra = 0; Rout = 0;
            end
        endcase

    end
endmodule
```

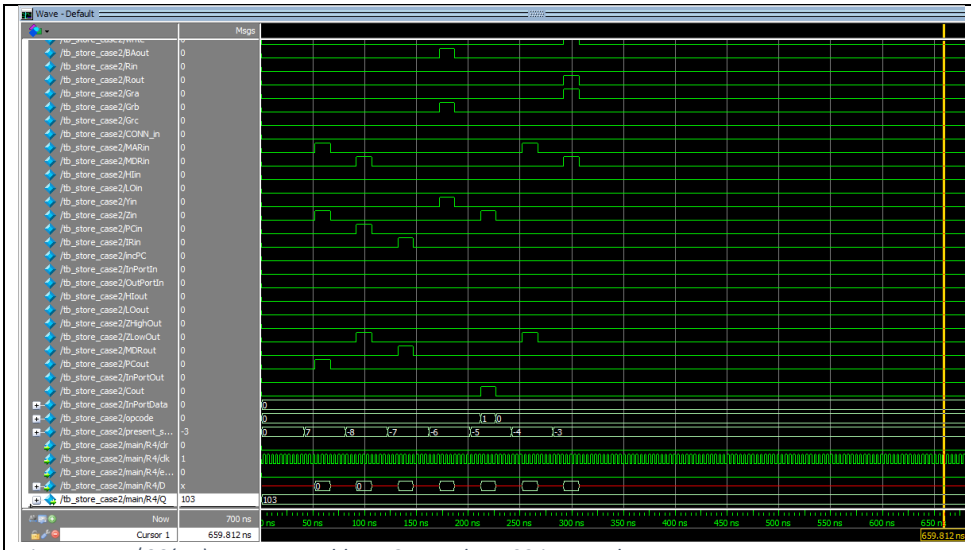


Figure 7: st $90(R4), R4. RAM Address 247, value 103 is stored.

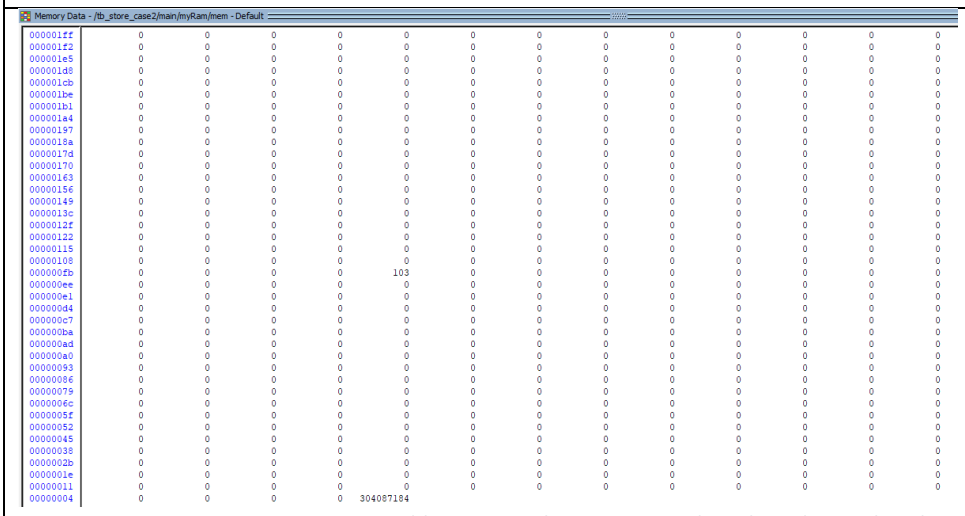


Figure 8: RAM contents. Note at RAM Address 247, value 103 is stored. Explained in testbench.

ALU Immediate Instructions – addi, andi, ori

ALU operation use the same 3 cycles at the beginning. The following changes are as such:

4. Place Rb in the ALU Y Reg
5. Place Rc or immediate value in the ALU. Pass Opcode to ALU, perform operation and place output in Z
6. Place ALU output into Ra

Addi R2, R3, -3

```
// Add test bench: addi R2, R3, -3
// R3 = 5, R2 = 0
// OPcode : 611BFFFD
// Adding 5 to -3 should return 2 in R2
`timescale 1ns/1ps
module tb_add;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
```

```
              log_xor = 5'b01101,
              log_nor = 5'b01110,
              log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
```

```
            Reg_load3b = 4'b0110,
            T0         = 4'b0111,
            T1         = 4'b1000,
            T2         = 4'b1001,
            T3         = 4'b1010,
            T4         = 4'b1011,
            T5         = 4'b1100,
            T6         = 4'b1101,
            T7         = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
            T3         : #40 present_state = T4;
            T4         : #40 present_state = T5;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
```

```
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            //Place PC in MAR to point to first instruction
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
            end

            //Read in value from MAR location in mem and place in MDR
            //Increment PC
            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            //Place MDR in IR
            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end
            //Place Rb in ALU
            T3: begin
                #10 Grb = 1; Rout = 1; Yin = 1;
                #15 Grb = 0; Rout = 0; Yin = 0;
            end
            //Place Immediate Value in ALU
            T4: begin
```

```
                #10 Cout = 1; Zin = 1; opcode = add; // ADD
                #15 Cout = 0; Zin = 0; opcode = nop;
            end
            //Place new value in Ra
            T5: begin
                #10 ZLowOut = 1; Gra = 1; Rin = 1;
                #15 ZLowOut = 0; Gra = 0; Rin = 0;
            end
        endcase

    end
endmodule
```

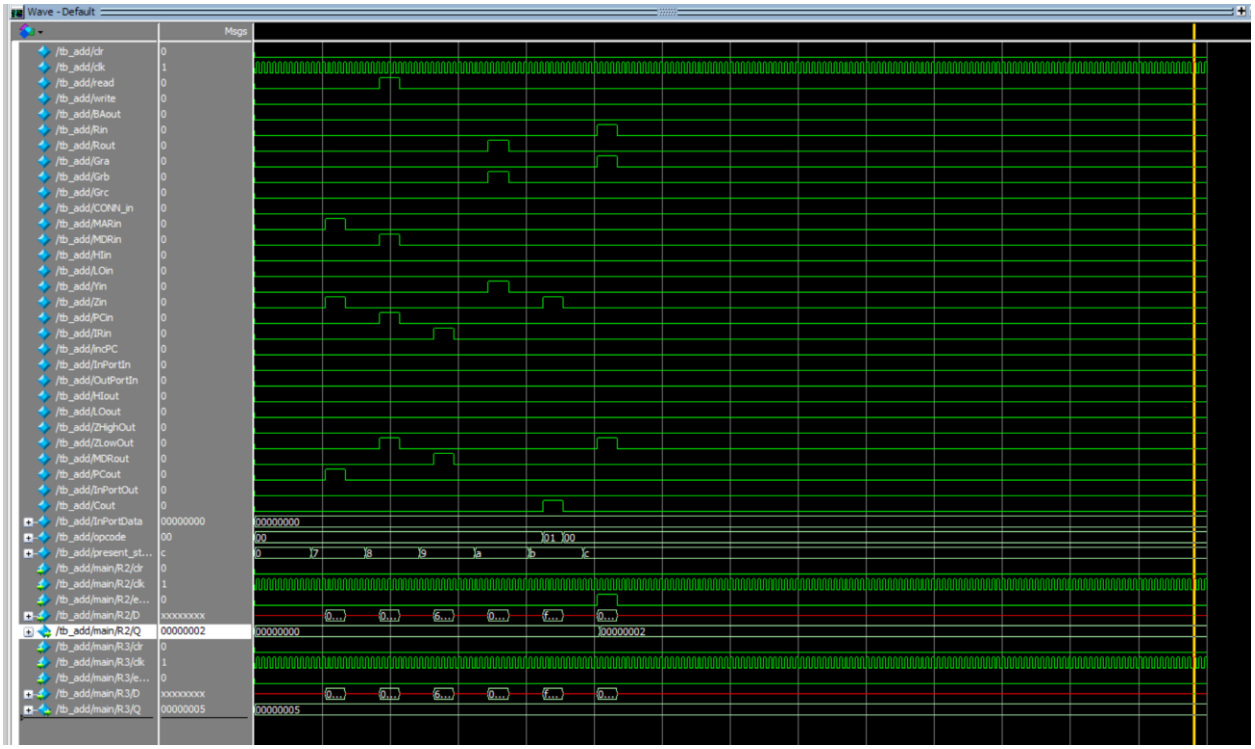


Figure 9: addi R2, R3, -3.

Andi R2,R3, $25

```
// And test bench: andi R2, R3, $25
// R3 = 5, R2 = 0
// OPcode : 69180025
// Anding 5 with $25 will return 5 in R2
`timescale 1ns/1ps
module tb_and;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
```

```
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
              T5         = 4'b1100,
              T6         = 4'b1101,
```

```
            T7          = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
            T3         : #40 present_state = T4;
            T4         : #40 present_state = T5;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
```

```
                BAout = 0;
                Gra = 0;
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            //Place PC in MAR to point to first instruction
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
            end

            //Read in value from MAR location in mem and place in MDR
            //Increment PC
            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            //Place MDR in IR
            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end
            //Place Rb in ALU
            T3: begin
                #10 Grb = 1; Rout = 1; Yin = 1;
                #15 Grb = 0; Rout = 0; Yin = 0;
            end
            //Place Immediate Value in ALU
            T4: begin
                #10 Cout = 1; Zin = 1; opcode = log_and; // and
                #15 Cout = 0; Zin = 0; opcode = nop;
            end
            //Place new value in Ra
            T5: begin
                #10 ZLowOut = 1; Gra = 1; Rin = 1;
                #15 ZLowOut = 0; Gra = 0; Rin = 0;
            end
```

```
        endcase

    end
endmodule
```

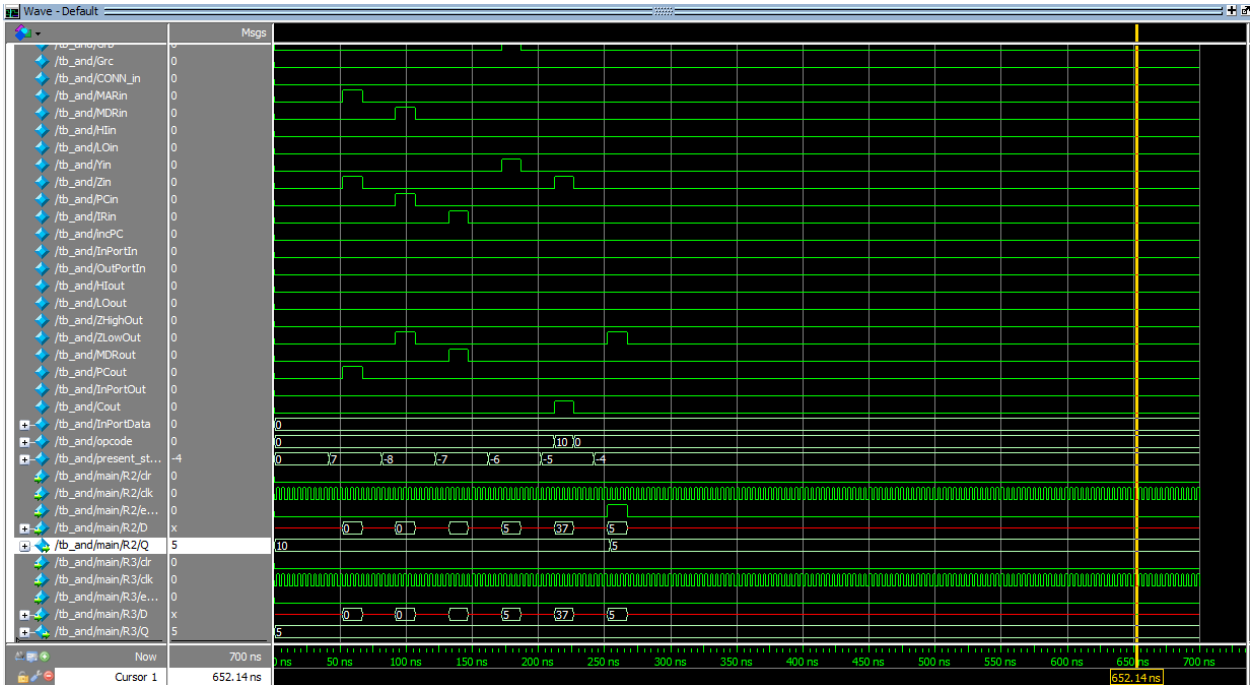


Figure 10: andi R2, R3, $25 (and operation of R3 with value 5 and $25 returns 5 in R2)

Ori R2, R3, $25

```
// Or test bench: ori R2, R3, $25
// R3 = 5, R2 = 0
// OPcode : 71180025
// Oring 5 with $25 will return 37 (in decimal) in R2
`timescale 1ns/1ps
module tb_or;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
```

```
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
              T5         = 4'b1100,
              T6         = 4'b1101,
```

```
            T7          = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
            T3         : #40 present_state = T4;
            T4         : #40 present_state = T5;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
```

```
                BAout = 0;
                Gra = 0;
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            //Place PC in MAR to point to first instruction
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
            end

            //Read in value from MAR location in mem and place in MDR
            //Increment PC
            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            //Place MDR in IR
            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end
            //Place Rb in ALU
            T3: begin
                #10 Grb = 1; Rout = 1; Yin = 1;
                #15 Grb = 0; Rout = 0; Yin = 0;
            end
            //Place Immediate Value in ALU
            T4: begin
                #10 Cout = 1; Zin = 1; opcode = log_or; // and
                #15 Cout = 0; Zin = 0; opcode = nop;
            end
            //Place new value in Ra
            T5: begin
                #10 ZLowOut = 1; Gra = 1; Rin = 1;
                #15 ZLowOut = 0; Gra = 0; Rin = 0;
            end
```

```
        endcase

    end
endmodule
```

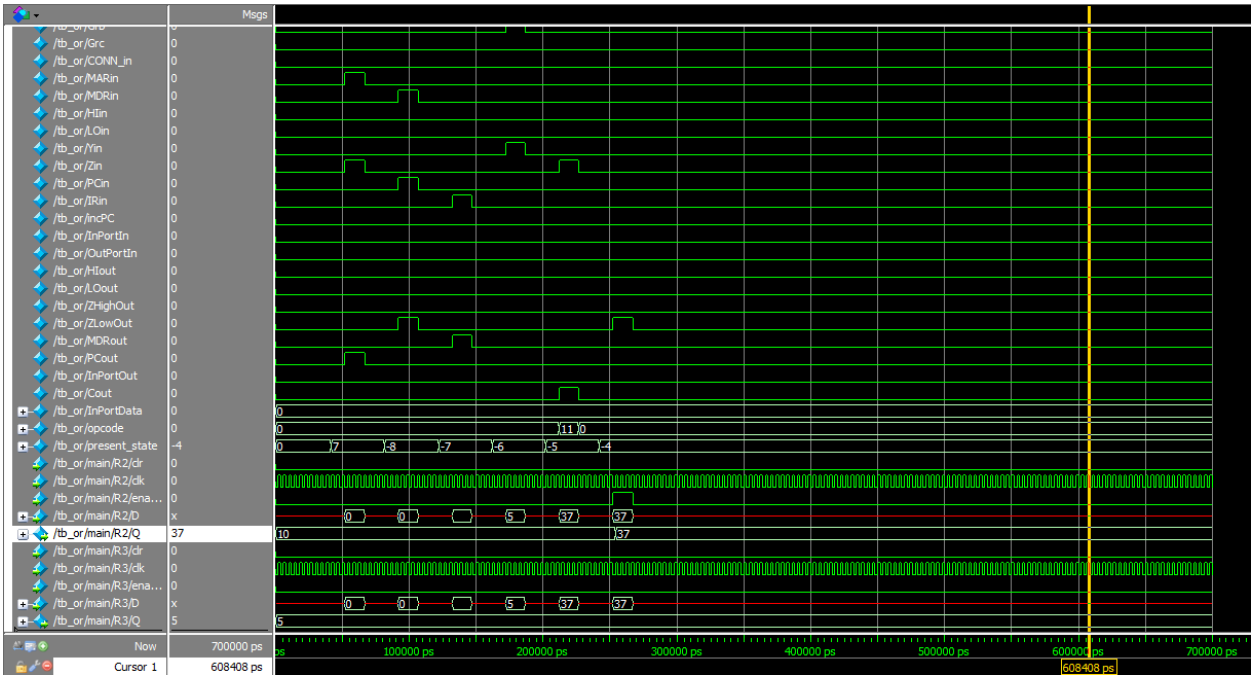


Figure 11: ori R2, R3, $25

Branch Instructions – brzr, brnz, brpl, brim

Branch Instructions have the first 3 cycles the same as all other testbenches. Subsequent cycles are as follows:

4. Place Ra on the bus for CONN logic to compare.
5. Place PC value on register and put into ALU Y register
6. Place offset we are branching too into ALU and comute using COMM_in signal to determine branching condition. If branching condition is met, ALU will add the offset to the PC value, otherwise ALU returns PC value.
7. Place new ALU output into PC

Case 1: brzr R6, 25

```
// Branch Case 1: brzr R6, 25
// Opcode: 9B000019
// R6 holds the value one, so it won't branch
// If R6 held 0, then it would branch to 26

`timescale 1ns/1ps

module tb_branch_case1;

    reg clr, clk;
   reg read, write;
   reg BAout, Rin, Rout;
   reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
   reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
   reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
```

```
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
```

```
            Reg_load1a = 4'b0001,
            Reg_load1b = 4'b0010,
            Reg_load2a = 4'b0011,
            Reg_load2b = 4'b0100,
            Reg_load3a = 4'b0101,
            Reg_load3b = 4'b0110,
            T0         = 4'b0111,
            T1         = 4'b1000,
            T2         = 4'b1001,
            T3         = 4'b1010,
            T4         = 4'b1011,
            T5         = 4'b1100,
            T6         = 4'b1101,
            T7         = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
            T3         : #40 present_state = T4;
            T4         : #40 present_state = T5;
            T5         : #40 present_state = T6;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
```

```
        Zin = 0;
        PCin = 0;
        IRin = 0;
        incPC = 0;
        InPortIn = 0;
        OutPortIn = 0;
        HIout = 0;
        LOout = 0;
        ZHighOut = 0;
        ZLowOut = 0;
        PCout = 0;
        MDRout = 0;
        InPortOut = 0;
        Cout = 0;
        BAout = 0;
        Gra = 0;
        Grb = 0;
        Grc = 0;
        Rin = 0;
        Rout = 0;
        InPortData = 0;
        opcode = 0;
        clr = 0;
    end

    //Instr fetch & increment PC
    T0: begin
        #10 PCout = 1; MARin = 1; Zin = 1; incPC = 1;
        #15 PCout = 0; MARin = 0; Zin = 0; incPC = 0;
    end
    // Load instr from mem and place MDR reg, place new PC val in PC reg
    T1: begin
        #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;
        #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
    end
    // Place instr in IR reg
    T2: begin
        #10 MDRout = 1; IRin = 1;
        #15 MDRout = 0; IRin = 0;
    end
    //Place PC value in ALU
    T3: begin
        #10 Gra = 1; Rout = 1;
        #15 Gra = 0; Rout = 0;
    end
```

```
            T4: begin
                #10 PCout = 1; Yin = 1;
                #15 PCout = 0; Yin = 0;
            end
            T5: begin
                #10 Cout = 1; Zin = 1; CONN_in = 1; opcode = nop;
                #15 Cout = 0; Zin = 0; CONN_in = 0;
            end
            T6: begin
                #10 ZLowOut = 1; PCin = 1;
                #15 ZLowOut = 0; PCin = 0;
            end
        endcase

    end
endmodule
```

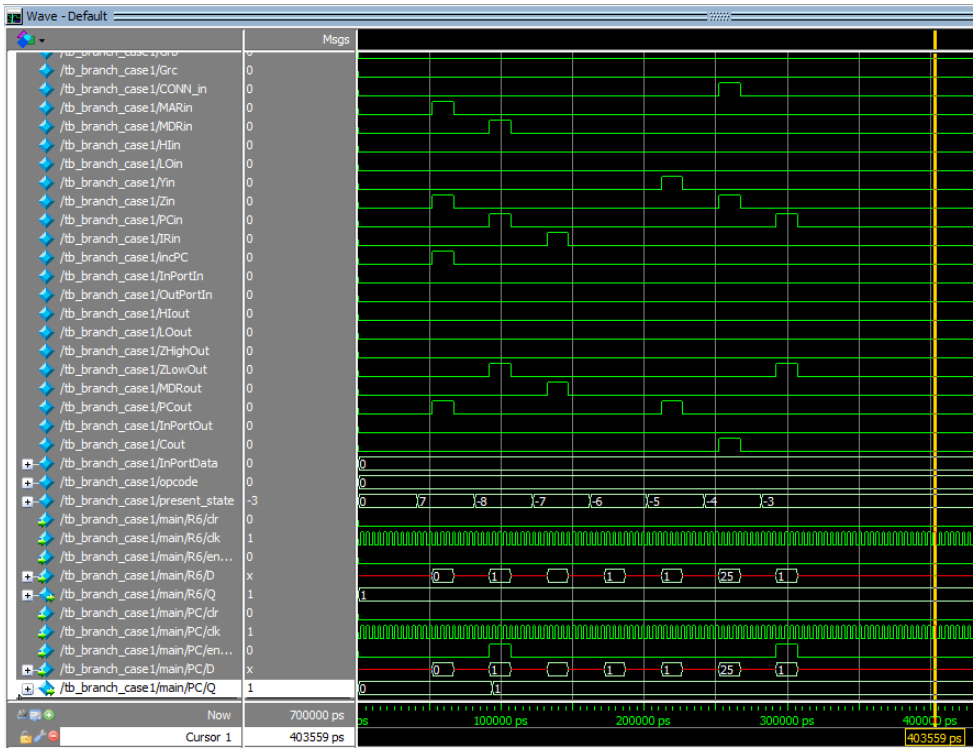


Figure 12: brzr R6, 25

Case 2: brnz R6, 25

```
// Branch Case 2: brnz R6, 25
// Opcde: 9B080019
// R6 holds the value one, so it will branch
// It should branch to value 26

`timescale 1ns/1ps

module tb_branch_case2;

    reg clr, clk;
   reg read, write;
   reg BAout, Rin, Rout;
   reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
   reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
   reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
```

```
        .clk(clk),
        .read(read),
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
```

```
            T5          = 4'b1100,
            T6          = 4'b1101,
            T7          = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
            T3         : #40 present_state = T4;
            T4         : #40 present_state = T5;
            T5         : #40 present_state = T6;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
```

```
            MDRout = 0;
            InPortOut = 0;
            Cout = 0;
            BAout = 0;
            Gra = 0;
            Grb = 0;
            Grc = 0;
            Rin = 0;
            Rout = 0;
            InPortData = 0;
            opcode = 0;
            clr = 0;
        end

        //Instr fetch & increment PC
        T0: begin
            #10 PCout = 1; MARin = 1; Zin = 1; incPC = 1;
            #15 PCout = 0; MARin = 0; Zin = 0; incPC = 0;
        end
        // Load instr from mem and place MDR reg, place new PC val in PC reg
        T1: begin
            #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;
            #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
        end
        // Place instr in IR reg
        T2: begin
            #10 MDRout = 1; IRin = 1;
            #15 MDRout = 0; IRin = 0;
        end
        //Place PC value in ALU
        T3: begin
            #10 Gra = 1; Rout = 1;
            #15 Gra = 0; Rout = 0;
        end
        T4: begin
            #10 PCout = 1; Yin = 1;
            #15 PCout = 0; Yin = 0;
        end
        T5: begin
            #10 Cout = 1; Zin = 1; CONN_in = 1; opcode = nop;
            #15 Cout = 0; Zin = 0; CONN_in = 0;
        end
        T6: begin
            #10 ZLowOut = 1; PCin = 1;
            #15 ZLowOut = 0; PCin = 0;
```

```
            end
        endcase

    end
endmodule
```

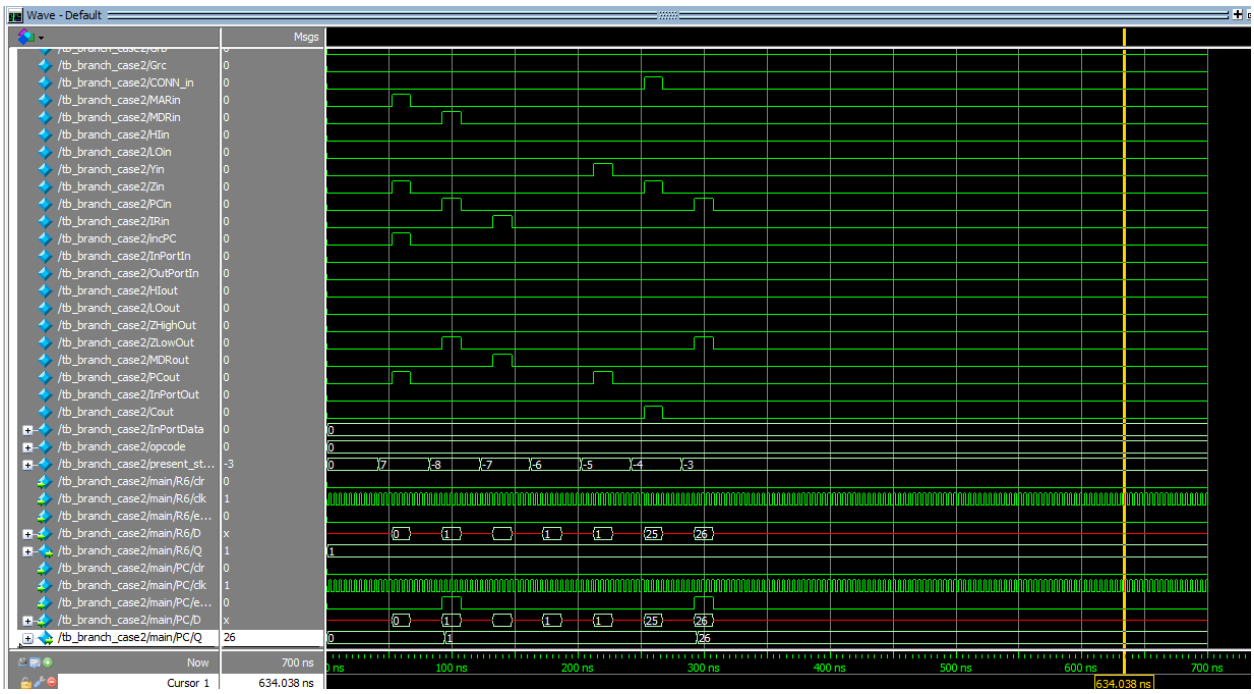


Figure 13: brnz R6, 25

Case 3: brpl R6, 25

```
// Branch Case 3: brpl R6, 25
// Opcde: 9B100019
// R6 holds the value one, so it will branch
// It should branch to value 26

`timescale 1ns/1ps

module tb_branch_case3;

    reg clr, clk;
   reg read, write;
   reg BAout, Rin, Rout;
   reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
   reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
   reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
```

```
        .clk(clk),
        .read(read),
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
```

```
            T5          = 4'b1100,
            T6          = 4'b1101,
            T7          = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
            T3         : #40 present_state = T4;
            T4         : #40 present_state = T5;
            T5         : #40 present_state = T6;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
```

```
            MDRout = 0;
            InPortOut = 0;
            Cout = 0;
            BAout = 0;
            Gra = 0;
            Grb = 0;
            Grc = 0;
            Rin = 0;
            Rout = 0;
            InPortData = 0;
            opcode = 0;
            clr = 0;
        end

        //Instr fetch & increment PC
        T0: begin
            #10 PCout = 1; MARin = 1; Zin = 1; incPC = 1;
            #15 PCout = 0; MARin = 0; Zin = 0; incPC = 0;
        end
        // Load instr from mem and place MDR reg, place new PC val in PC reg
        T1: begin
            #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;
            #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
        end
        // Place instr in IR reg
        T2: begin
            #10 MDRout = 1; IRin = 1;
            #15 MDRout = 0; IRin = 0;
        end
        //Place PC value in ALU
        T3: begin
            #10 Gra = 1; Rout = 1;
            #15 Gra = 0; Rout = 0;
        end
        T4: begin
            #10 PCout = 1; Yin = 1;
            #15 PCout = 0; Yin = 0;
        end
        T5: begin
            #10 Cout = 1; Zin = 1; CONN_in = 1; opcode = nop;
            #15 Cout = 0; Zin = 0; CONN_in = 0;
        end
        T6: begin
            #10 ZLowOut = 1; PCin = 1;
            #15 ZLowOut = 0; PCin = 0;
```

```
            end
        endcase

    end
endmodule
```

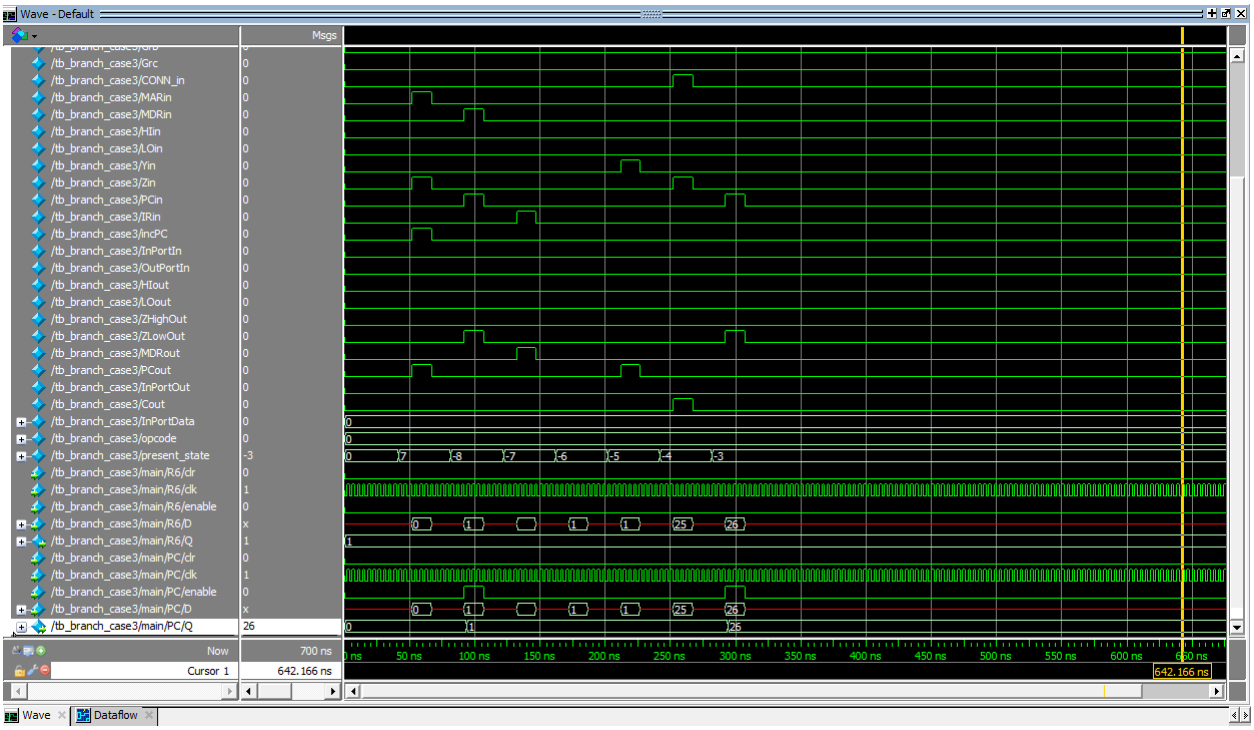


Figure 14: brpl R6, 25

Case 4: brmi R6, 25

```
// Branch Case 4: brmi R6, 25
// Opcde: 9B180019
// R6 holds the value one, so it will not branch
// It shouldn't branch to value 26

`timescale 1ns/1ps

module tb_branch_case4;

    reg clr, clk;
   reg read, write;
   reg BAout, Rin, Rout;
   reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
   reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
   reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
```

```
        .clk(clk),
        .read(read),
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
```

```
            T5          = 4'b1100,
            T6          = 4'b1101,
            T7          = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
            T3         : #40 present_state = T4;
            T4         : #40 present_state = T5;
            T5         : #40 present_state = T6;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
```

```
            MDRout = 0;
            InPortOut = 0;
            Cout = 0;
            BAout = 0;
            Gra = 0;
            Grb = 0;
            Grc = 0;
            Rin = 0;
            Rout = 0;
            InPortData = 0;
            opcode = 0;
            clr = 0;
        end

        //Instr fetch & increment PC
        T0: begin
            #10 PCout = 1; MARin = 1; Zin = 1; incPC = 1;
            #15 PCout = 0; MARin = 0; Zin = 0; incPC = 0;
        end
        // Load instr from mem and place MDR reg, place new PC val in PC reg
        T1: begin
            #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;
            #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
        end
        // Place instr in IR reg
        T2: begin
            #10 MDRout = 1; IRin = 1;
            #15 MDRout = 0; IRin = 0;
        end
        //Place PC value in ALU
        T3: begin
            #10 Gra = 1; Rout = 1;
            #15 Gra = 0; Rout = 0;
        end
        T4: begin
            #10 PCout = 1; Yin = 1;
            #15 PCout = 0; Yin = 0;
        end
        T5: begin
            #10 Cout = 1; Zin = 1; CONN_in = 1; opcode = nop;
            #15 Cout = 0; Zin = 0; CONN_in = 0;
        end
        T6: begin
            #10 ZLowOut = 1; PCin = 1;
            #15 ZLowOut = 0; PCin = 0;
```

```
            end
        endcase

    end
endmodule
```

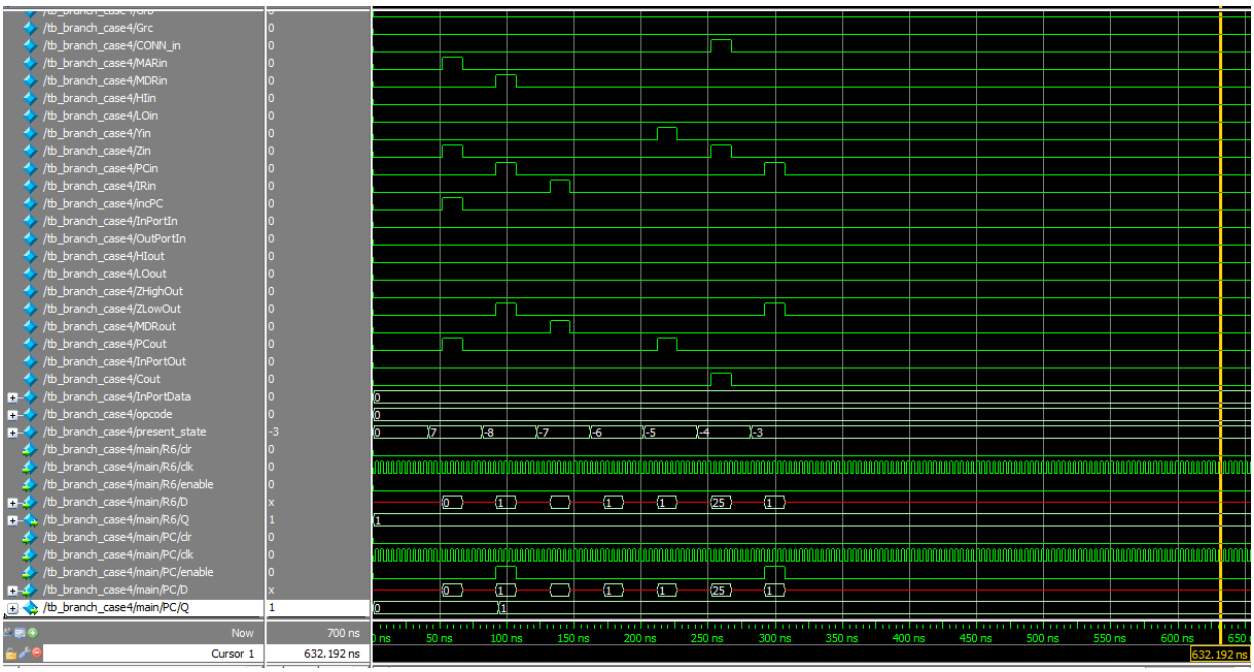


Figure 15: brmi R6, 25

Jump Instructions – jr, jal

Jump Instructions have the first 3 cycles the same as all other testbenches. Subsequent cycles are as follows for jr instructions:

4. Place value in Ra in PC

If the instruction is jal:

4. Place value in PC in Ra

Jr R2

```
// jr test bench: jr R2
// opcode: A1000000
// R2 holds 10, so PC should hold 10 after T2

`timescale 1ns/1ps
module jr_tb;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
```

```
              log_neg = 5'b01100,
              log_xor = 5'b01101,
              log_nor = 5'b01110,
              log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
```

```
            Reg_load3a = 4'b0101,
            Reg_load3b = 4'b0110,
            T0         = 4'b0111,
            T1         = 4'b1000,
            T2         = 4'b1001,
            T3         = 4'b1010,
            T4         = 4'b1011,
            T5         = 4'b1100,
            T6         = 4'b1101,
            T7         = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
```

```
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            //Place PC in MAR to point to first instruction
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; incPC = 0;
            end

            //Read in value from MAR location in mem and place in MDR
            //Increment PC
            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            //Place MDR in IR
            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end
            //Place Rb in ALU
            T3: begin
                #10 Gra = 1; Rout = 1; PCin = 1;
                #15 Gra = 0; Rout = 0; PCin = 0;
            end
        endcase

    end
```

```
endmodule
```

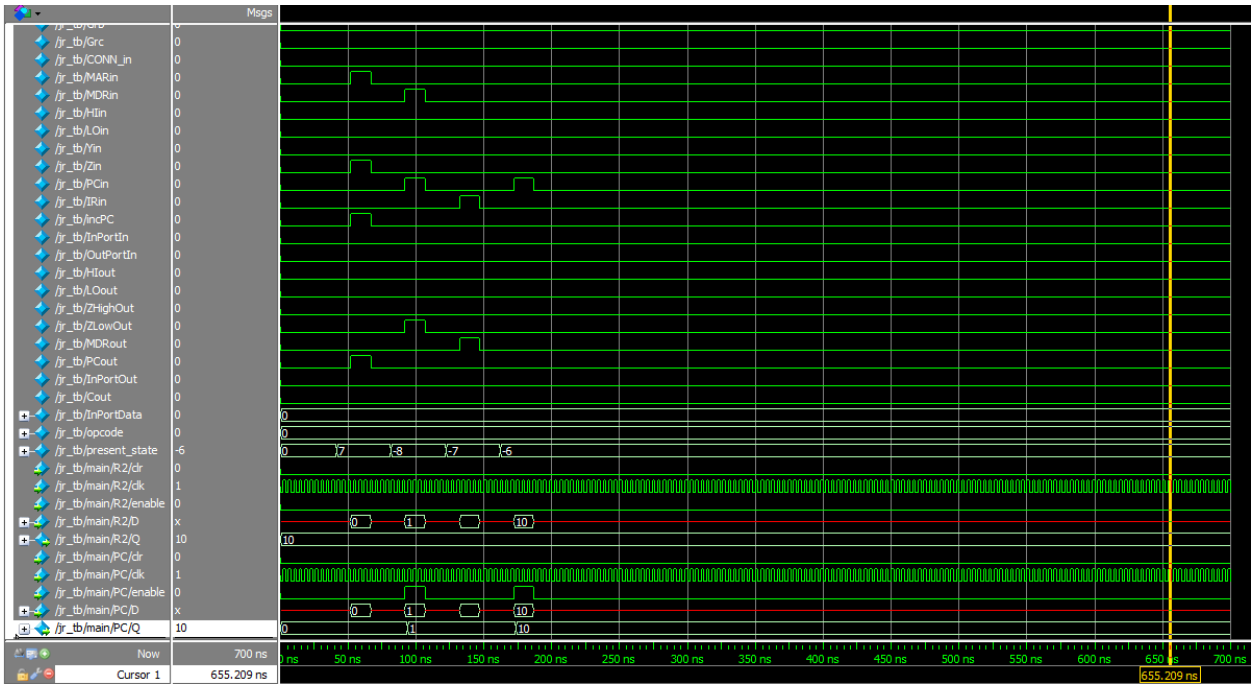


Figure 16: jr R2

Jal R2

```
// jal test bench: jal R2
// opcode: A9000000
// R2 holds 10, so PC should hold 10 after T2

`timescale 1ns/1ps
module jal_tb;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
```

```
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
              T5         = 4'b1100,
              T6         = 4'b1101,
```

```
            T7          = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
```

```
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            //Place PC in MAR to point to first instruction
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; incPC = 0;
            end

            //Read in value from MAR location in mem and place in MDR
            //Increment PC
            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            //Place MDR in IR
            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end
            //Place PC in Ra
            T3: begin
                #10 Gra = 1; Rin = 1; PCout = 1;
                #15 Gra = 0; Rin = 0; PCout = 0;
            end
        endcase

    end
endmodule
```

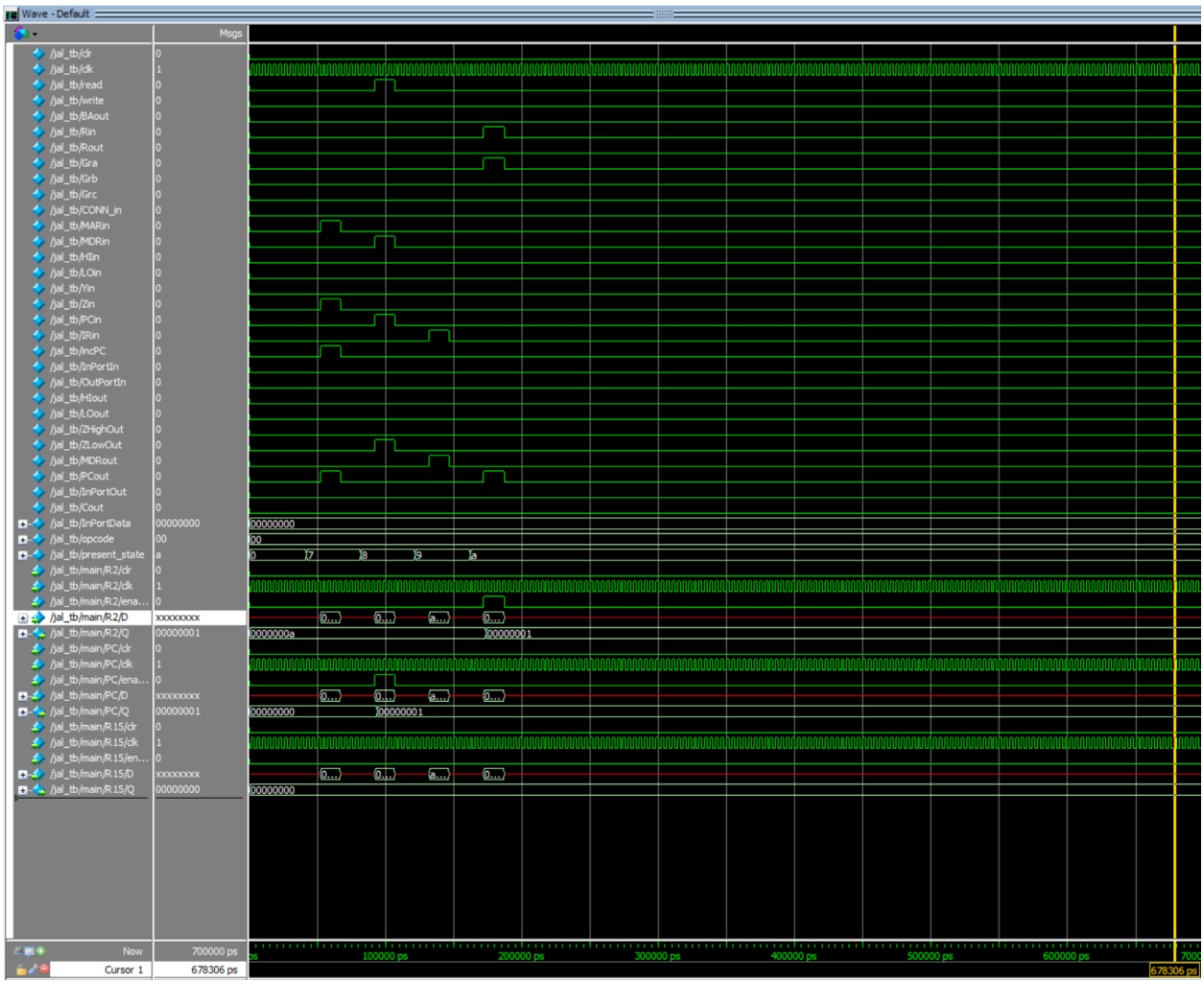


Figure 17: jal R2

Special Instructions – mfhi and mflo

Mfhi and mflo Instructions have the first 3 cycles the same as all other testbenches. Subsequent cycles are as follows for the instructions:

4. Place value in mfhi or mflo in Ra

Mfhi

```
// mfhi test bench: mfhi R4
// opcode: C2000000
// High holds 100, so R4 should hold 100

`timescale 1ns/1ps
module tb_mfhi;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
```

```
            log_nor = 5'b01110,
            log_not = 5'b01111;

datapath main (
    .clr(clr),
    .clk(clk),
    .read(read),
    .write(write),
    .BAout(BAout),
    .Rin(Rin),
    .Rout(Rout),
    .Gra(Gra),
    .Grb(Grb),
    .Grc(Grc),
    .CONN_in(CONN_in),
    .MARin(MARin),
    .MDRin(MDRin),
    .HIin(HIin),
    .LOin(LOin),
    .Yin(Yin),
    .Zin(Zin),
    .PCin(PCin),
    .IRin(IRin),
    .incPC(incPC),
    .InPortIn(InPortIn),
    .OutPortIn(OutPortIn),
    .HIout(HIout),
    .LOout(LOout),
    .ZLowOut(ZLowOut),
    .ZHighOut(ZHighOut),
    .MDRout(MDRout),
    .Cout(Cout),
    .InPortOut(InPortOut),
    .PCout(PCout),
    .opcode(opcode),
    .InPortData(InPortData)
);

parameter Default    = 4'b0000,
          Reg_load1a = 4'b0001,
          Reg_load1b = 4'b0010,
          Reg_load2a = 4'b0011,
          Reg_load2b = 4'b0100,
          Reg_load3a = 4'b0101,
          Reg_load3b = 4'b0110,
```

```
            T0          = 4'b0111,
            T1          = 4'b1000,
            T2          = 4'b1001,
            T3          = 4'b1010,
            T4          = 4'b1011,
            T5          = 4'b1100,
            T6          = 4'b1101,
            T7          = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default     : #40 present_state = T0;
            T0          : #40 present_state = T1;
            T1          : #40 present_state = T2;
            T2          : #40 present_state = T3;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
```

```
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            //Place PC in MAR to point to first instruction
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
            end

            //Read in value from MAR location in mem and place in MDR
            //Increment PC
            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            //Place MDR in IR
            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end
            //Place Rb in ALU
            T3: begin
                #10 HIout = 1; Gra = 1; Rin = 1;
                #15 HIout = 1; Gra = 1; Rin = 1;
            end
        endcase

    end
endmodule
```

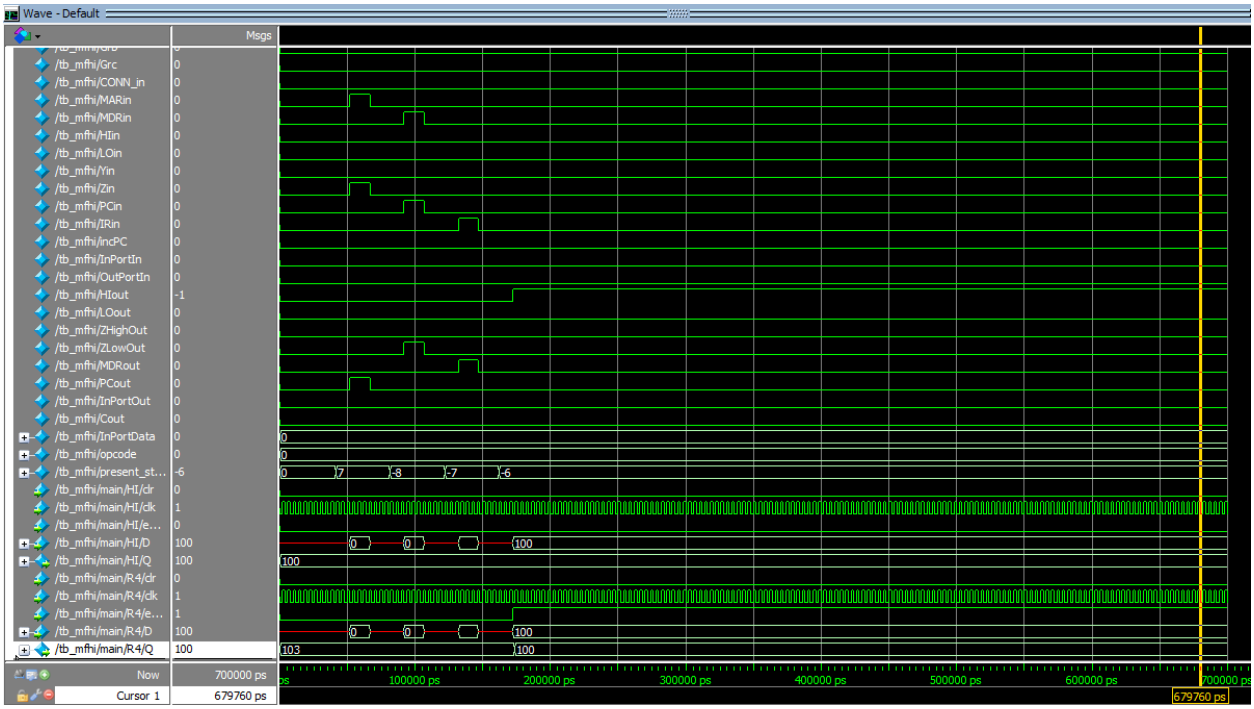


Figure 18: mfhi R4

Mflo

```
// mflo test bench: mflo R6
// opcode: CB000000
// LOW holds 10, so R6 should hold 10

`timescale 1ns/1ps
module tb_mflo;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
```

```
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
              T5         = 4'b1100,
              T6         = 4'b1101,
```

```
            T7          = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
```

```
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            //Place PC in MAR to point to first instruction
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
            end

            //Read in value from MAR location in mem and place in MDR
            //Increment PC
            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            //Place MDR in IR
            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end
            //Place Rb in ALU
            T3: begin
                #10 LOout = 1; Gra = 1; Rin = 1;
                #15 LOout = 1; Gra = 1; Rin = 1;
            end
        endcase

    end
endmodule
```

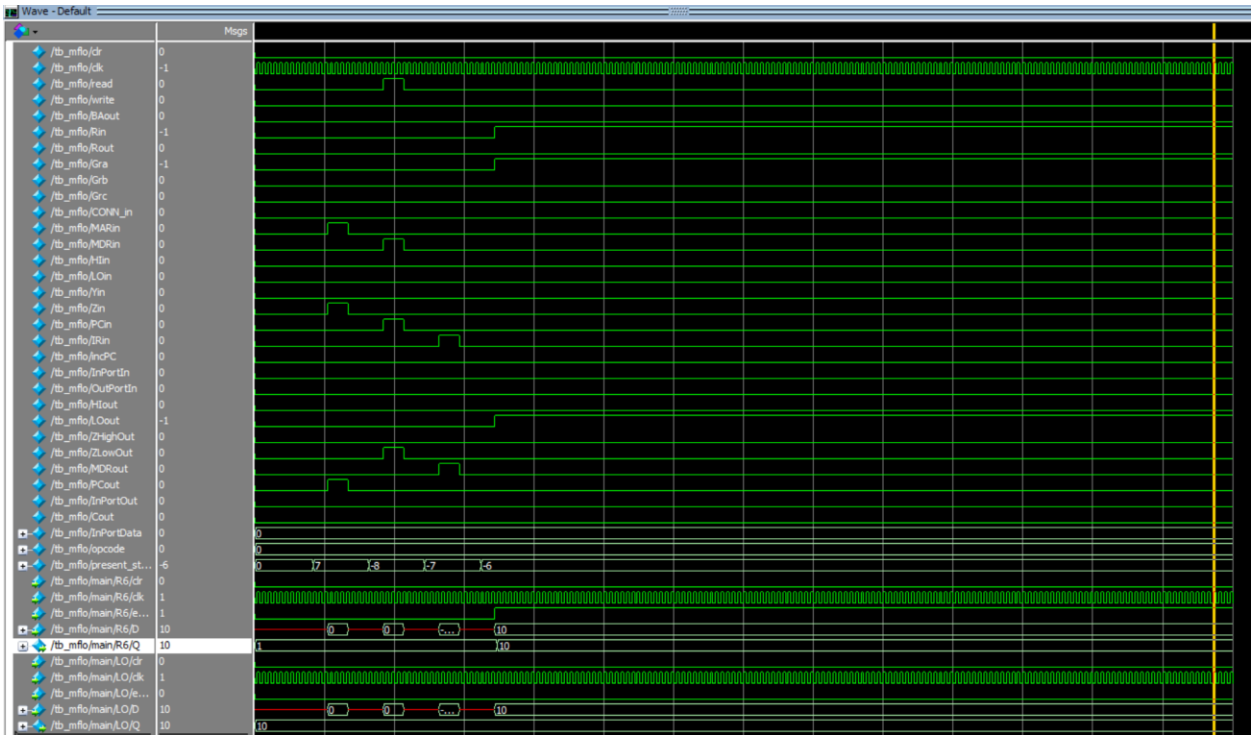


Figure 19: mflo R6 (moves LO to R6) used value of 10

Input/Output Instructions – in, out

IO Instructions have the first 3 cycles the same as all other testbenches. Subsequent cycles are as follows for in instructions:

3. Place value in InPortData into Ra

If the instruction is out:

3. Place value in Ra into OutPortData

Out R2

```
// out test bench: out R2
// opcode: B9000000
// R2 holds 10, so outport should hold 10

`timescale 1ns/1ps
module tb_out;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                    add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
```

```
              log_or = 5'b01011,
              log_neg = 5'b01100,
              log_xor = 5'b01101,
              log_nor = 5'b01110,
              log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
```

```
            Reg_load2b = 4'b0100,
            Reg_load3a = 4'b0101,
            Reg_load3b = 4'b0110,
            T0         = 4'b0111,
            T1         = 4'b1000,
            T2         = 4'b1001,
            T3         = 4'b1010,
            T4         = 4'b1011,
            T5         = 4'b1100,
            T6         = 4'b1101,
            T7         = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
```

```
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            //Place PC in MAR to point to first instruction
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
            end

            //Read in value from MAR location in mem and place in MDR
            //Increment PC
            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            //Place MDR in IR
            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end
            //Place Rb in ALU
            T3: begin
                #10 OutPortIn = 1; Gra = 1; Rout = 1;
                #15 OutPortIn = 0; Gra = 0; Rout = 0;
            end
        endcase
```

```
    end
endmodule
```

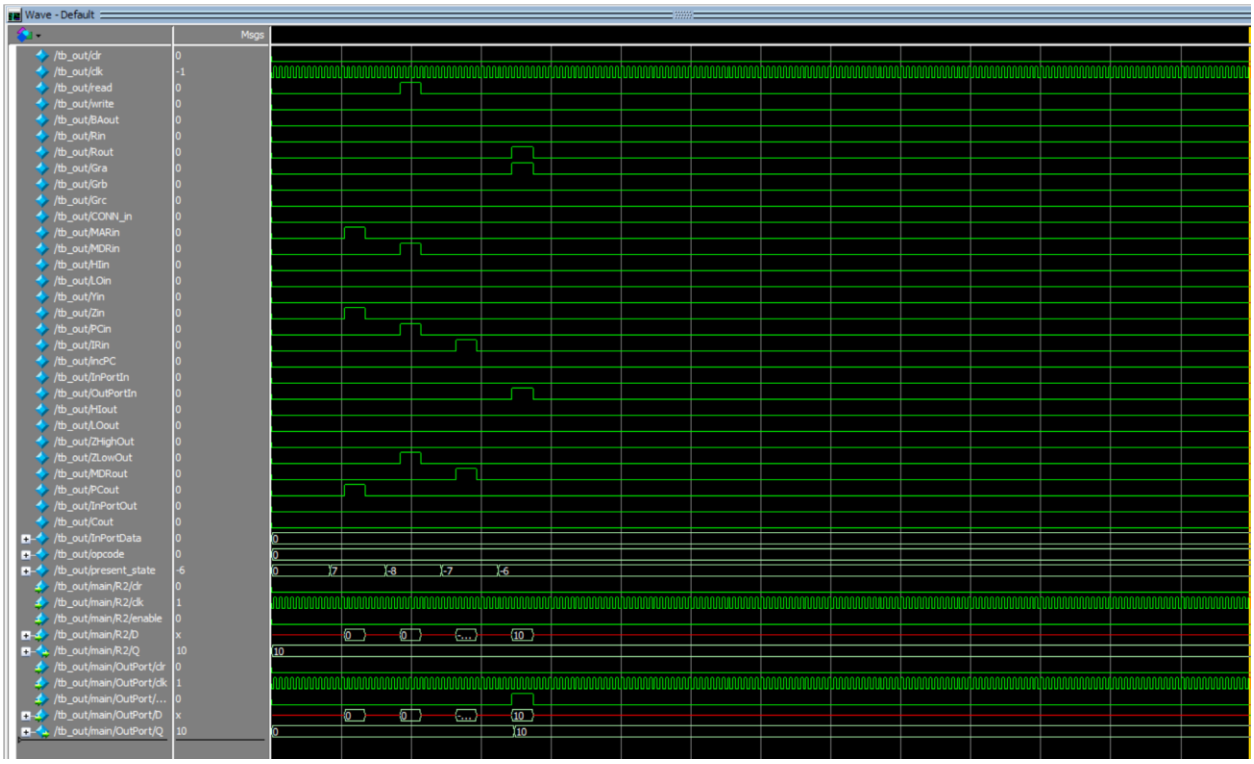


Figure 20: Out R2. Note, R2 holds 10, thus placing the value 10 in OutPort

In R3

```
// out test bench: in R3
// opcode: B1800000
// Places value 50 in R3

`timescale 1ns/1ps
module tb_in;

    reg clr, clk;
    reg read, write;
    reg BAout, Rin, Rout;
    reg Gra, Grb, Grc;
    reg CONN_in;
    reg MARin, MDRin;
    reg HIin, LOin;
    reg Yin, Zin;
    reg PCin, IRin, incPC;
    reg InPortIn, OutPortIn;
    reg HIout, LOout, ZHighOut, ZLowOut;
    reg MDRout, PCout, InPortOut, Cout;
    reg [31:0] InPortData;
    reg [4:0] opcode;


    parameter   nop = 5'b00000,
                    add = 5'b00001,
                sub = 5'b00010,
                mul = 5'b00011,
                div = 5'b00100,
                shr = 5'b00101,
                shl = 5'b00110,
                shra = 5'b00111,
                ror = 5'b01000,
                rol = 5'b01001,
                log_and = 5'b01010,
                log_or = 5'b01011,
                log_neg = 5'b01100,
                log_xor = 5'b01101,
                log_nor = 5'b01110,
                log_not = 5'b01111;

    datapath main (
        .clr(clr),
        .clk(clk),
        .read(read),
```

```
        .write(write),
        .BAout(BAout),
        .Rin(Rin),
        .Rout(Rout),
        .Gra(Gra),
        .Grb(Grb),
        .Grc(Grc),
        .CONN_in(CONN_in),
        .MARin(MARin),
        .MDRin(MDRin),
        .HIin(HIin),
        .LOin(LOin),
        .Yin(Yin),
        .Zin(Zin),
        .PCin(PCin),
        .IRin(IRin),
        .incPC(incPC),
        .InPortIn(InPortIn),
        .OutPortIn(OutPortIn),
        .HIout(HIout),
        .LOout(LOout),
        .ZLowOut(ZLowOut),
        .ZHighOut(ZHighOut),
        .MDRout(MDRout),
        .Cout(Cout),
        .InPortOut(InPortOut),
        .PCout(PCout),
        .opcode(opcode),
        .InPortData(InPortData)
    );

    parameter Default    = 4'b0000,
              Reg_load1a = 4'b0001,
              Reg_load1b = 4'b0010,
              Reg_load2a = 4'b0011,
              Reg_load2b = 4'b0100,
              Reg_load3a = 4'b0101,
              Reg_load3b = 4'b0110,
              T0         = 4'b0111,
              T1         = 4'b1000,
              T2         = 4'b1001,
              T3         = 4'b1010,
              T4         = 4'b1011,
              T5         = 4'b1100,
              T6         = 4'b1101,
```

```
            T7          = 4'b1110;

    reg [3:0] present_state = Default;

    initial begin
        clk = 0;
        forever #2 clk = ~clk;
    end

    always @(posedge clk) begin
        case (present_state)
            Default    : #40 present_state = T0;
            T0         : #40 present_state = T1;
            T1         : #40 present_state = T2;
            T2         : #40 present_state = T3;
        endcase
    end

    always @(present_state) begin
        case (present_state)
            Default : begin
                CONN_in = 0;
                MDRin = 0;
                MARin = 0;
                read = 0;
                write = 0;
                HIin = 0;
                LOin = 0;
                Yin = 0;
                Zin = 0;
                PCin = 0;
                IRin = 0;
                incPC = 0;
                InPortIn = 0;
                OutPortIn = 0;
                HIout = 0;
                LOout = 0;
                ZHighOut = 0;
                ZLowOut = 0;
                PCout = 0;
                MDRout = 0;
                InPortOut = 0;
                Cout = 0;
                BAout = 0;
                Gra = 0;
```

```
                Grb = 0;
                Grc = 0;
                Rin = 0;
                Rout = 0;
                InPortData = 0;
                opcode = 0;
                clr = 0;
            end

            //Place PC in MAR to point to first instruction
            T0: begin
                #10 PCout = 1; MARin = 1; Zin = 1; //incPC = 1;
                #15 PCout = 0; MARin = 0; Zin = 0; //incPC = 0;
            end

            //Read in value from MAR location in mem and place in MDR
            //Increment PC
            T1: begin
                #10 ZLowOut = 1; PCin = 1; read = 1; MDRin = 1;//ZLowOut should
be high but for not using ALU to increment PC
                #15 ZLowOut = 0; PCin = 0; read = 0; MDRin = 0;
            end

            //Place MDR in IR
            T2: begin
                #10 MDRout = 1; IRin = 1;
                #15 MDRout = 0; IRin = 0;
            end
            //Place Inport into R3
            T3: begin
                #10 InPortData = 32'd50; InPortIn = 1;
                #15 InPortIn = 0;
                #10 InPortOut = 1; Gra = 1; Rin = 1;
                #15 InPortOut = 0; Gra = 0; Rin = 0;
            end
        endcase

    end
endmodule
```

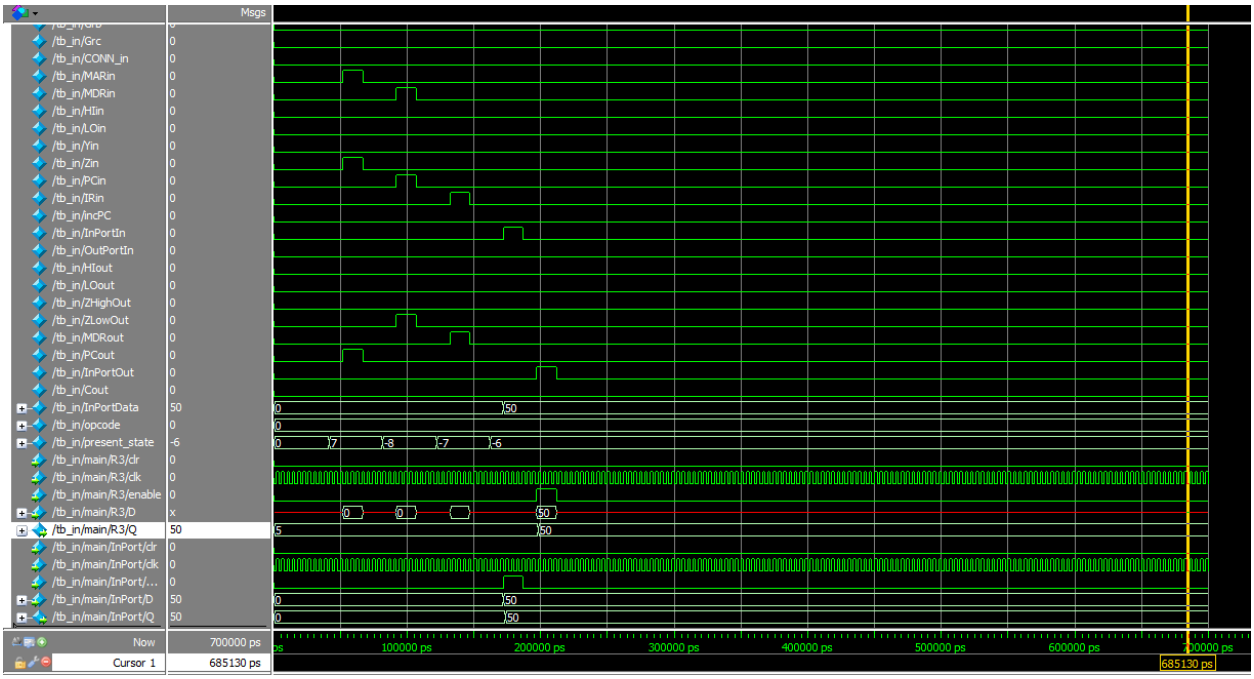


Figure 21: in R3