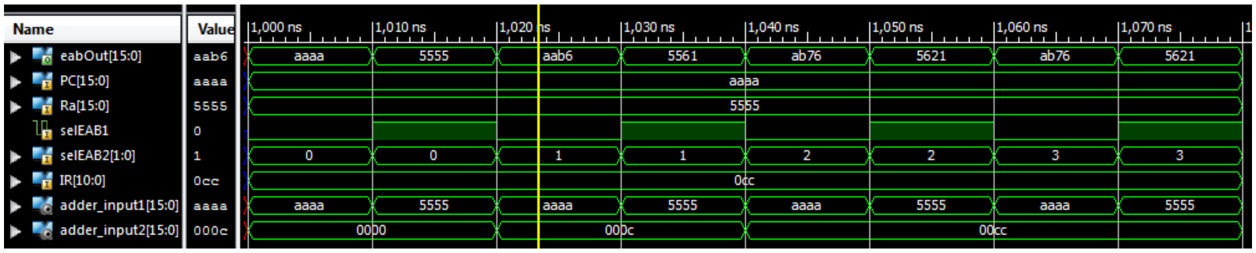
Taylor Cowley
Lab 10: Designing the LC3 Datapath (LC3 Functional Units)
Why do these labs have 2 separate titles?
June 13 2016

EAB Verilog file

```
// The EAB. Which includes
// The ADDR2MUX
// ADDR1MUX
// ADDER
// Sign extension before the addr2mux

module EAB(eabOut, PC, Ra, selEAB1, selEAB2, IR);
    output[15:0] eabOut;
    input[15:0] PC;
    input[15:0] Ra;
    input selEAB1;
    input[1:0] selEAB2;
    input[10:0] IR;

    wire[15:0] adder_input1, adder_input2;

    // ADDR1MUX - If 1, Ra. If 2, PC
    // No sign extension needed because Ra and PC are both 16 bits
    assign adder_input1 = selEAB1? Ra : PC ;

    // ADDR2MUX - If 3, IR[10:0]
    //      If 2, IR[8:0]
    //      If 1, IR[5:0]
    //      If 0, 0
    //      All need to be sign-extended to 16 bits
    assign adder_input2 = (selEAB2 == 2'b11)? {{5{IR[10]}} , IR[10:0]} :
                          (selEAB2 == 2'b10)? {{7{IR[8] }} , IR[8:0] } :
                          (selEAB2 == 2'b01)? {{10{IR[5] }} , IR[5:0] } :
                                              16'h0000;
    assign eabOut = adder_input1 + adder_input2;
endmodule
```

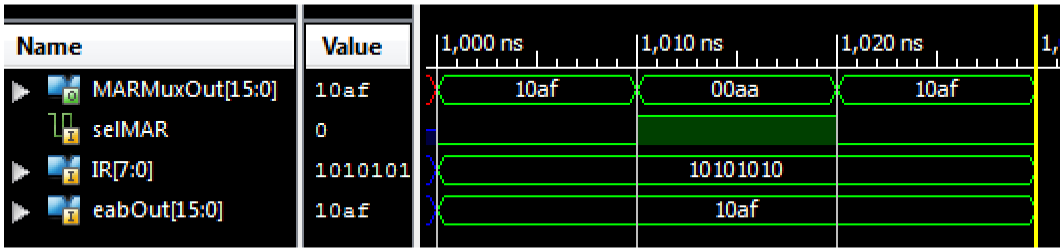
EAB Simulation waveform


Name
Value
1,000 ns
1,010 ns
1,020 ns
1,030 ns
1,040 ns
1,050 ns
1,060 ns
1,070 ns
eabOut[15:0] aab6 aaaa 5555 aab6 5561 ab76 5621 ab76 5621
PC[15:0] aaaa aaaa
Ra[15:0] 5555 5555
selEAB1 0
selEAB2[1:0] 1 0 0 1 1 2 2 3 3
IR[10:0] 0cc 0cc
adder_input1[15:0] aaaa aaaa 5555 aaaa 5555 aaaa 5555 aaaa 5555
adder_input2[15:0] 000c 0000 000c 00cc

MARMux Verilog file

```verilog
// This is the MARMUX.
// it only has the MARMUX in it
module MARMUX(MARMuxOut, selMAR, IR, eabOut);
    output[15:0] MARMuxOut;
    input selMAR;
    input[7:0] IR;
    input[15:0] eabOut;

    // If 1, a zero-extended IR. If 0, eabOut.
    assign MARMuxOut = selMAR ? {8'h00, IR[7:0]} : eabOut;

endmodule

```

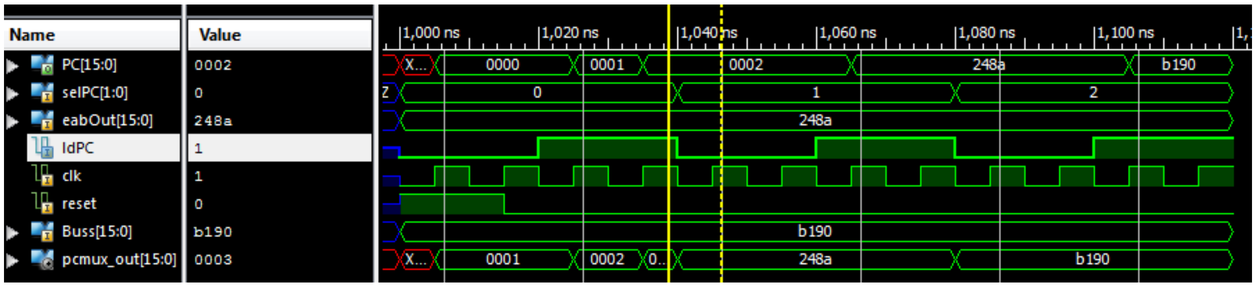
MARMux Simulation waveform



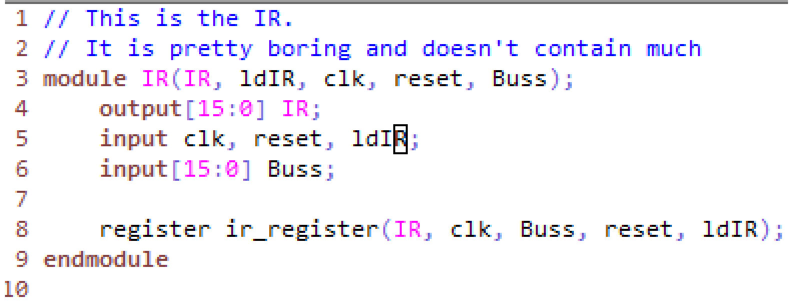
PC Verilog file

```verilog
// This is PC.
// It includes PCMUX, the PC itself, and a +1 adder
module PC(PC, selPC, eabOut, ldPC, clk, reset, Buss);
    output[15:0] PC;
    input[1:0] selPC;
    input[15:0] eabOut;
    input ldPC;
    input clk;
    input reset;
    input[15:0] Buss;


    // This implements PCMUX.
    // 00-PC+1
    // 01-eabOut
    // 10-Buss
    // 11-0xFF just for fun (not authorized)
    wire[15:0] pcmux_out;
    assign pcmux_out =  (selPC==2'b00) ? PC + 1:
                        (selPC==2'b01) ? eabOut:
                        (selPC==2'b10) ? Buss:
                        16'hFF;

    register pc_register(PC, clk, pcmux_out, reset, ldPC);

endmodule
```

PC Simulation waveform

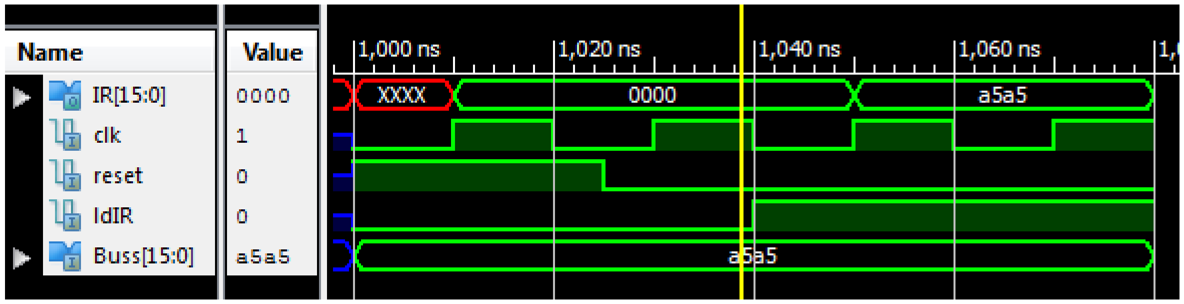


IR Verilog file

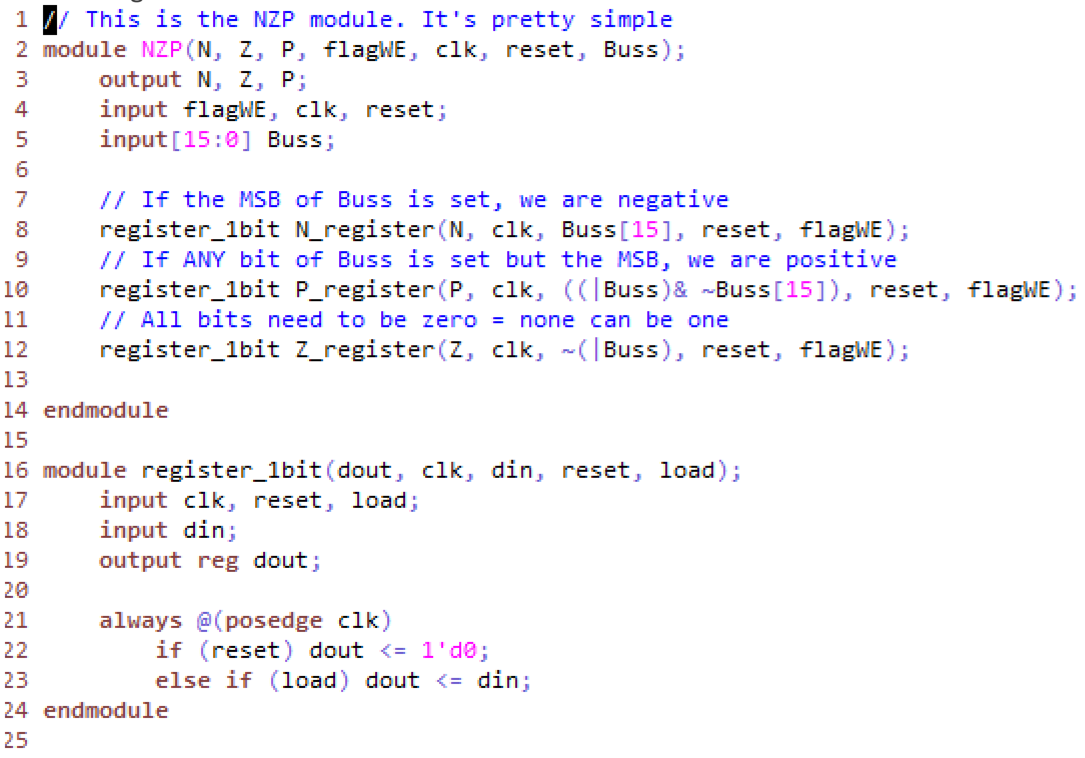


```verilog
// This is the IR.
// It is pretty boring and doesn't contain much
module IR(IR, ldIR, clk, reset, Buss);
    output[15:0] IR;
    input clk, reset, ldIR;
    input[15:0] Buss;

    register ir_register(IR, clk, Buss, reset, ldIR);
endmodule
```

IR Simulation waveform

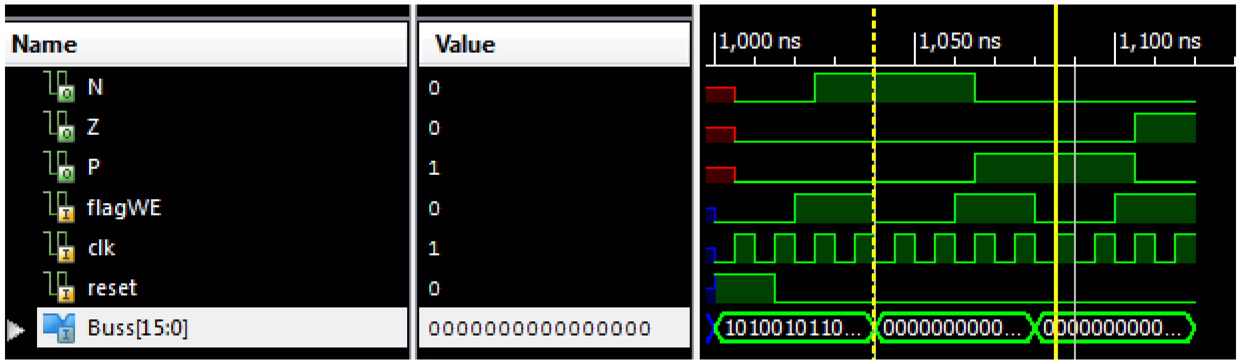


NZP Verilog file



```verilog
// This is the NZP module. It's pretty simple
module NZP(N, Z, P, flagWE, clk, reset, Buss);
    output N, Z, P;
    input flagWE, clk, reset;
    input[15:0] Buss;

    // If the MSB of Buss is set, we are negative
    register_1bit N_register(N, clk, Buss[15], reset, flagWE);
    // If ANY bit of Buss is set but the MSB, we are positive
    register_1bit P_register(P, clk, ((|Buss)& ~Buss[15]), reset, flagWE);
    // All bits need to be zero = none can be one
    register_1bit Z_register(Z, clk, ~(|Buss), reset, flagWE);

endmodule

module register_1bit(dout, clk, din, reset, load);
    input clk, reset, load;
    input din;
    output reg dout;

    always @(posedge clk)
        if (reset) dout <= 1'd0;
        else if (load) dout <= din;
endmodule
```

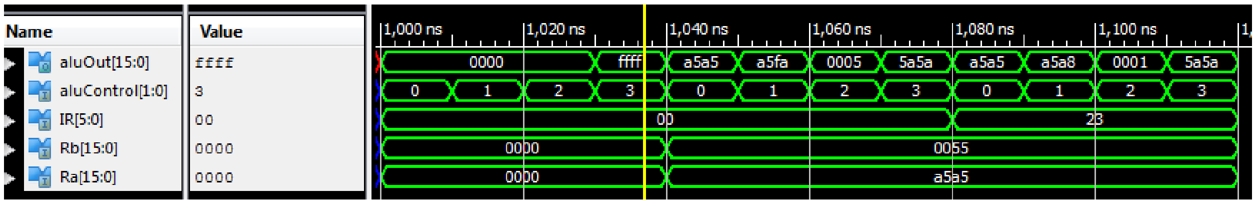
NZP Simulation waveform



ALU Verilog file

```verilog
// An ALU. It says it is the same one we already did
// 00 = A
// 01 = ADD
// 10 = AND
// 11 = NOT A
module ALU(aluOut, aluControl, IR, Rb, Ra);
    output[15:0] aluOut;
    input[1:0] aluControl;
    input[5:0] IR;
    input[15:0] Rb, Ra;

    // First we need to do SR2MUX - 0 means Rb, 1 means IR[4:0]
    wire[15:0] sr2mux;
    assign sr2mux = IR[5] ? {{11{IR[4]}},IR[4:0]} : Rb;

    assign aluOut = (aluControl==2'b00) ? Ra :
                    (aluControl==2'b01) ? Ra + sr2mux :
                    (aluControl==2'b10) ? Ra & sr2mux :
                                          ~Ra;
endmodule

```

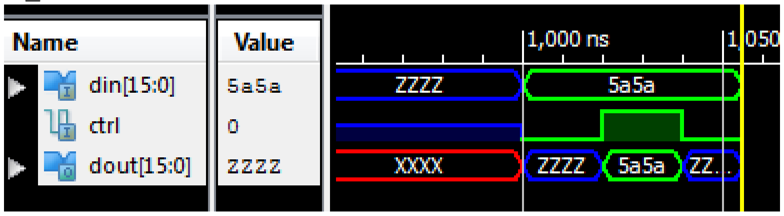
ALU Simulation waveform



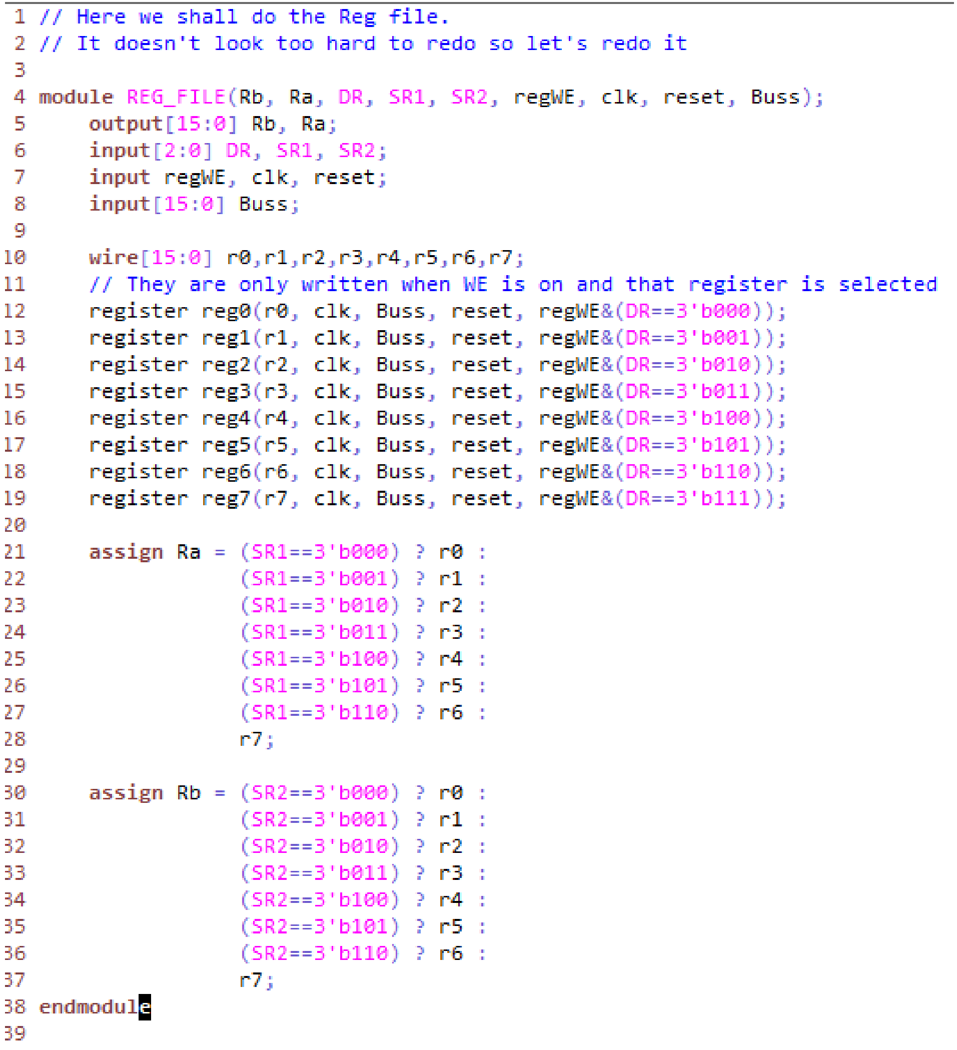
ts_driver Verilog file

```verilog
// Copy-pasted from learning suite?
module ts_driver ( din, dout, ctrl );
    input [15:0] din;
    input ctrl;
    output [15:0] dout;

    assign dout = (ctrl)? din:(16'bZZZZZZZZZZZZZZZZ);
endmodule

```

ts_driver Simulation waveform

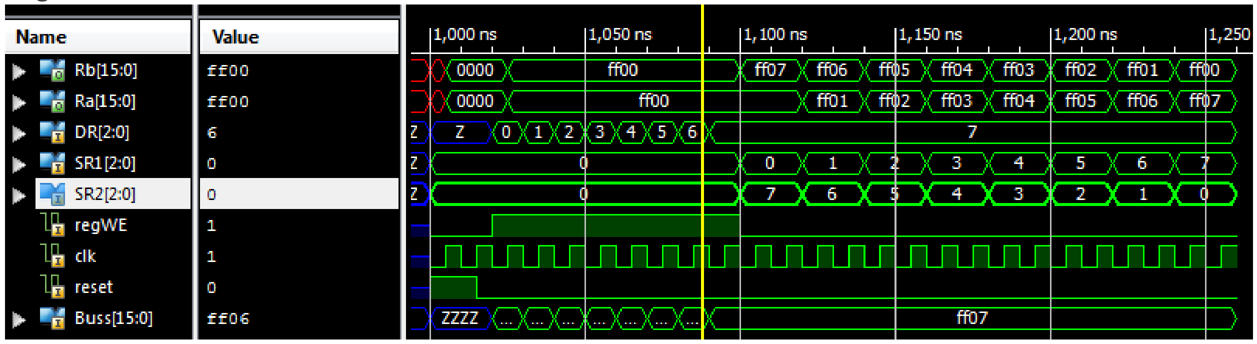


RegFile Verilog file



```
// Here we shall do the Reg file.
// It doesn't look too hard to redo so let's redo it

module REG_FILE(Rb, Ra, DR, SR1, SR2, regWE, clk, reset, Buss);
    output[15:0] Rb, Ra;
    input[2:0] DR, SR1, SR2;
    input regWE, clk, reset;
    input[15:0] Buss;

    wire[15:0] r0,r1,r2,r3,r4,r5,r6,r7;
    // They are only written when WE is on and that register is selected
    register reg0(r0, clk, Buss, reset, regWE&(DR==3'b000));
    register reg1(r1, clk, Buss, reset, regWE&(DR==3'b001));
    register reg2(r2, clk, Buss, reset, regWE&(DR==3'b010));
    register reg3(r3, clk, Buss, reset, regWE&(DR==3'b011));
    register reg4(r4, clk, Buss, reset, regWE&(DR==3'b100));
    register reg5(r5, clk, Buss, reset, regWE&(DR==3'b101));
    register reg6(r6, clk, Buss, reset, regWE&(DR==3'b110));
    register reg7(r7, clk, Buss, reset, regWE&(DR==3'b111));

    assign Ra = (SR1==3'b000) ? r0 :
                (SR1==3'b001) ? r1 :
                (SR1==3'b010) ? r2 :
                (SR1==3'b011) ? r3 :
                (SR1==3'b100) ? r4 :
                (SR1==3'b101) ? r5 :
                (SR1==3'b110) ? r6 :
                r7;

    assign Rb = (SR2==3'b000) ? r0 :
                (SR2==3'b001) ? r1 :
                (SR2==3'b010) ? r2 :
                (SR2==3'b011) ? r3 :
                (SR2==3'b100) ? r4 :
                (SR2==3'b101) ? r5 :
                (SR2==3'b110) ? r6 :
                r7;
endmodule

```

RegFile Simulation waveform



Name	Value
Rb[15:0]	ff00
Ra[15:0]	ff00
DR[2:0]	6
SR1[2:0]	0
SR2[2:0]	0
regWE	1
clk	1
reset	0
Buss[15:0]	ff06

Anomalies (bugs, problems, and suggestions)(5pts possible)

So just realizing that I had to trust the basic modules- they aren't doing anything special! Then it was easy to code up. And VIM is much nicer to code Verilog into than the Xilinx editor. In case you were wondering.