Jacob Sickafoose

Lab # 6

Lab Section - 1B

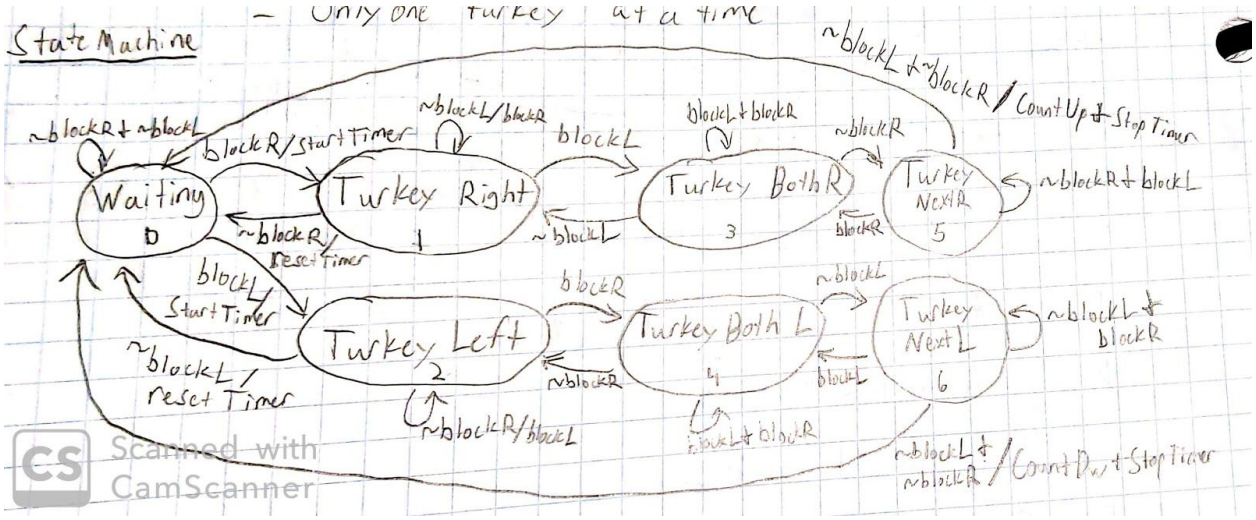
2/28/2020

Description -

For this lab, we were tasked with creating a counter to count turkeys crossing two sensors. It had to increment when they walked one way and decrement the other way. It had to be able to distinguish when a turkey went halfway through, then turned around, as well as if a turkey just stood in the center. It also had to keep track of negative numbers, in the case of the counter being at zero, and a turkey went left to right. For the final part, the left side of the display had to display a second count, while there was a turkey blocking either of the beams.

Method -

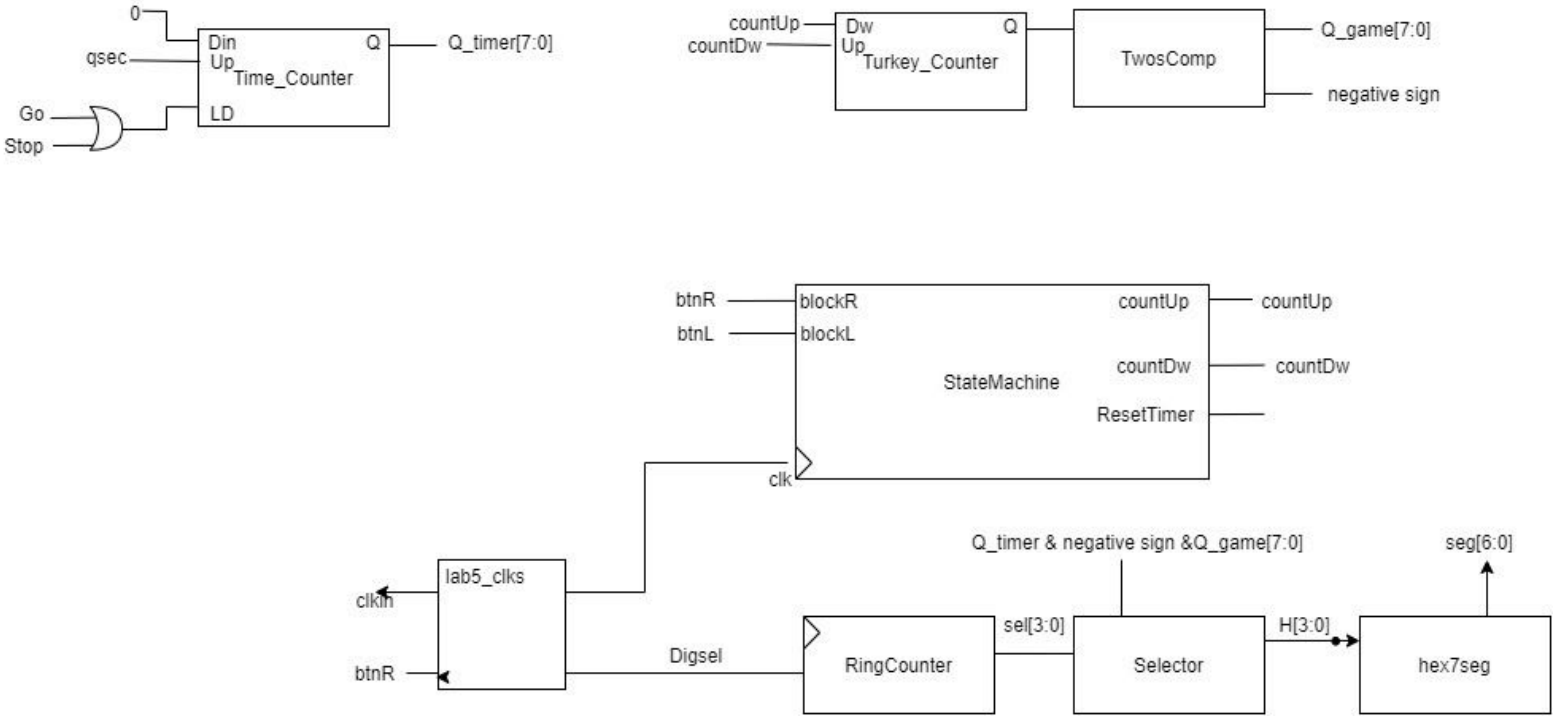
I started this lab by first implementing the logic for the state machine. I started by drawing out the following flow chart with each state and what variables turned on which states, as well as their outputs:



I used the one hot method to implement the states with flip flops. This means that there is a flip flop for each state and only one flip flop is high at a time. The flip flop that is high, is the current state. This made it much easier to implement the logic to switch between states. The counter for counting time was also fairly easy. It was just a standard counter that incremented every 4 times that qsec did. The main counter, and handling the negatives was the trickiest part. I had to create a separate module to handle the negatives. I used a 2 input mux, with each input being an 8-bit number, and the selector being the most significant bit. Once the MSB was high, it chose the regular counter 8-bit out, except every value was inverted and that was sent to a full adder which added the 1 bit, carry. This 1 bit carry was just set to the MSB.

Design -

This is the schematic for my top level:



I had to figure out how to change the states based on the inputs and using a different flip flop for each state. I ended with the following expressions for the Dins on the flip flops:

D_0= (~blockR & ~blockL)

D_1= (Q_0 & blockR & ~blockL) | (Q_1 & blockR & ~blockL) | (Q_3 & blockR & ~blockL)

D_2= (Q_0 & ~blockR & blockL) | (Q_2 & ~blockR & blockL) | (Q_4 & ~blockR & blockL)

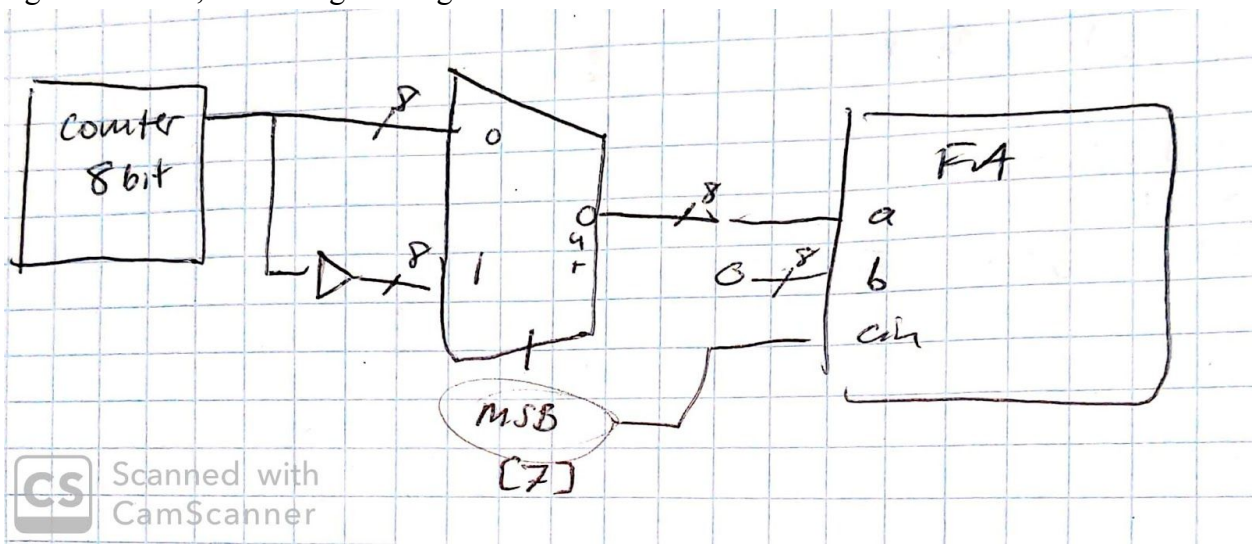
D_3= (Q_1 & blockL & blockR) | (Q_3 & blockL & blockR) | (Q_5 & blockL& blockR)

D_4= (Q_2 & blockR & blockL) | (Q_4 & blockR & blockL) | (Q_6 & blockR & blockL)

D_5 = (Q_3 & ~blockR & blockL) | (Q_5 & ~blockR & blockL)

D_6 = (Q_4 & blockR & ~blockL) | (Q_6 & blockR & ~blockL)

I also ended up with the following design for converting the negative Two's Complement into a regular number, with a negative sign:



Conclusion -

In order to finish this lab, the only new thing I had to do was convert the number I get when I count down from zero, into a negative number using two's complement. The longest part was figuring out how to plug everything together. The state machine took a little bit of thinking, but thankfully it worked from the get go and didn't take any changing. It just took me a long time

to figure out how to do the two's complement. With some help, I figured out I needed my own module to translate from two's complement, and to use a mux and an adder.. If I had to do this same lab again, it would probably be much easier now that I have it all figured out.

Appendix -

```
module top_lab6(
    input clkin,
    input btnU,
    input btnR,
    input btnL,
    output [15:0] led,
    output [3:0] an,
    output dp,
    output [6:0] seg
    );
    /*Clk           */wire digsel, qsec, clk;
    /*StateMachine  */wire countUp, countDw, resetTimer/*, startTimer*/;
    /*Time_Counter  */wire [7:0] Q_timer;

    lab6_clks lab6_clk (.clkin(clkin), .greset(btnU), .clk(clk), .digsel(digsel), .qsec(qsec));
    stateMachine StateMachine (.clk(clk), .blockR(btnR), .blockL(btnL), .countUp(countUp),
        .countDw(countDw), .resetTimer(resetTimer)/*, .startTimer(startTimer)*/);
    // Have edge detector just in case, for the resetTimer because it stays high
    wire reset;
    edgeDetector EdgeDetector (.btn(resetTimer), .clk(clk), .btnOut(reset));
    counterUD8L Time_Counter (.clk(clk), .Din(8'b00000000), .Up(qsec & ~resetTimer & ~(&Q_timer)), .Dw(1'b0),.LD(reset), .Q(Q_timer));

    counterUD8L Turkey_Counter(); // need to modify for negatives I think
endmodule
```

```
module stateMachine(
    input clk,
    input blockR,
    input blockL,
    output countUp,
    output countDw,
    output resetTimer/*,
    output startTimer*/
    );
    wire [6:0] D;
    wire [6:0] Q;

    assign D[0] = (~blockR&~blockL);
    assign D[1] = (Q[0]&blockR&~blockL) | (Q[1]&blockR&~blockL) | (Q[3]&blockR&~blockL);
    assign D[2] = (Q[0]&~blockR&blockL) | (Q[2]&~blockR&blockL) | (Q[4]&~blockR&blockL);
    assign D[3] = (Q[1]&blockL&blockR) | (Q[3]&blockL&blockR) | (Q[5]&blockL&blockR);
    assign D[4] = (Q[2]&blockR&blockL) | (Q[4]&blockR&blockL) | (Q[6]&blockR&blockL);
    assign D[5] = (Q[3]&~blockR&blockL) | (Q[5]&~blockR&blockL);
    assign D[6] = (Q[4]&blockR&~blockL) | (Q[6]&blockR&~blockL);

    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[3]), .Q(Q[3]));
    FDRE #(.INIT(1'b0)) Q4_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[4]), .Q(Q[4]));
    FDRE #(.INIT(1'b0)) Q5_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[5]), .Q(Q[5]));
    FDRE #(.INIT(1'b0)) Q6_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[6]), .Q(Q[6]));

    assign countUp = Q[5]&~blockL&~blockR;
    assign countDw = Q[6]&~blockL&~blockR;
    assign resetTimer = Q[0];
//    assign startTimer = (Q[0]&blockR) | (Q[0]&blockL);
endmodule
```

Modules used from previous labs :

```
module edgeDetector(
    input btn,
    input clk,
    output btnOut
    );
    wire [1:0] D;
    wire [1:0] Q;

    assign D[0] = btn;
    assign D[1] = Q[0];

    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[1]), .Q(Q[1]));

    assign btnOut = ~Q[1]&Q[0];
endmodule
```

```
module counterUD8L(
    input clk,
    input [7:0] Din,
    input Dw,
    input Up,
    input LD,
    output UTC,
    output DTC,
    output [7:0] Q
    );
    wire [1:0] utc;
    wire [1:0] dtc;

    countUD4L count0 (.Up(Up), .Dw(Dw), .LD(LD), .Din(Din[3:0]), .clk(clk), .UTC(utc[0]), .DTC(dtc[0]), .Q(Q[3:0]));
    countUD4L count1 (.Up(utc[0]&Up | Up&UTC), .Dw(dtc[0]&Dw | Dw&DTC), .LD(LD), .Din(Din[7:4]), .clk(clk), .UTC(utc[1]), .DTC(dtc[1]), .Q(Q[7:4]));

    assign UTC = utc[0]&utc[1];
    assign DTC = dtc[0]&dtc[1];
endmodule
```

```
module countUD4L(
    input Up,
    input Dw,
    input LD,
    input [3:0] Din,
    input clk,
    output UTC,
    output DTC,
    output [3:0] Q
    );
    wire [3:0] D;

    assign D[0] = (Q[0]^Up^Dw)&~LD | Din[0]&LD;
    assign D[1] = (Q[1]^(Q[0]&Up)^(~Q[0]&Dw))&~LD | Din[1]&LD;
    assign D[2] = (Q[2]^(Q[0]&Q[1]&Up)^(~Q[0]&~Q[1]&Dw))&~LD | Din[2]&LD;
    assign D[3] = (Q[3]^(Up&Q[2]&Q[1]&Q[0])^(Dw&~Q[2]&~Q[1]&~Q[0]))&~LD | Din[3]&LD;

    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[3]), .Q(Q[3]));

    assign UTC = Q[0]&Q[1]&Q[2]&Q[3];
    assign DTC = ~Q[0]&~Q[1]&~Q[2]&~Q[3];

endmodule
```

```
module ringCounter(
    input adv,
    input clk,
    output [3:0] sel
    );
//    wire [3:0] D;
    wire [3:0] Q;

//    assign D[0] = (Q[0]&~adv) | (Q[3]&adv);
//    assign D[1] = (Q[1]&~adv) | (Q[0]&adv);
//    assign D[2] = (Q[2]&~adv) | (Q[1]&adv);
//    assign D[3] = (Q[3]&~adv) | (Q[2]&adv);

    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .R(1'b0), .CE(adv), .D(Q[3]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(1'b0), .CE(adv), .D(Q[0]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(1'b0), .CE(adv), .D(Q[1]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(1'b0), .CE(adv), .D(Q[2]), .Q(Q[3]));

    assign sel[0] = Q[0];
    assign sel[1] = Q[1];
    assign sel[2] = Q[2];
    assign sel[3] = Q[3];
endmodule
```

```
module selector(
    input [3:0] sel,
    input [15:0] N,
    output [3:0] H
    );

    assign H[0] = (~sel[3]&~sel[2]&~sel[1]&sel[0]&N[0]) | (~sel[3]&~sel[2]&sel[1]&~sel[0]&N[4]) | (~sel[3]&sel[2]&~sel[1]&~sel[0]&N[8]) | (sel[3]&~sel[2]&~sel[1]&~sel[0]&N[12]);
    assign H[1] = (~sel[3]&~sel[2]&~sel[1]&sel[0]&N[1]) | (~sel[3]&~sel[2]&sel[1]&~sel[0]&N[5]) | (~sel[3]&sel[2]&~sel[1]&~sel[0]&N[9]) | (sel[3]&~sel[2]&~sel[1]&~sel[0]&N[13]);
    assign H[2] = (~sel[3]&~sel[2]&~sel[1]&sel[0]&N[2]) | (~sel[3]&~sel[2]&sel[1]&~sel[0]&N[6]) | (~sel[3]&sel[2]&~sel[1]&~sel[0]&N[10]) | (sel[3]&~sel[2]&~sel[1]&~sel[0]&N[14]);
    assign H[3] = (~sel[3]&~sel[2]&~sel[1]&sel[0]&N[3]) | (~sel[3]&~sel[2]&sel[1]&~sel[0]&N[7]) | (~sel[3]&sel[2]&~sel[1]&~sel[0]&N[11]) | (sel[3]&~sel[2]&~sel[1]&~sel[0]&N[15]);

endmodule
```

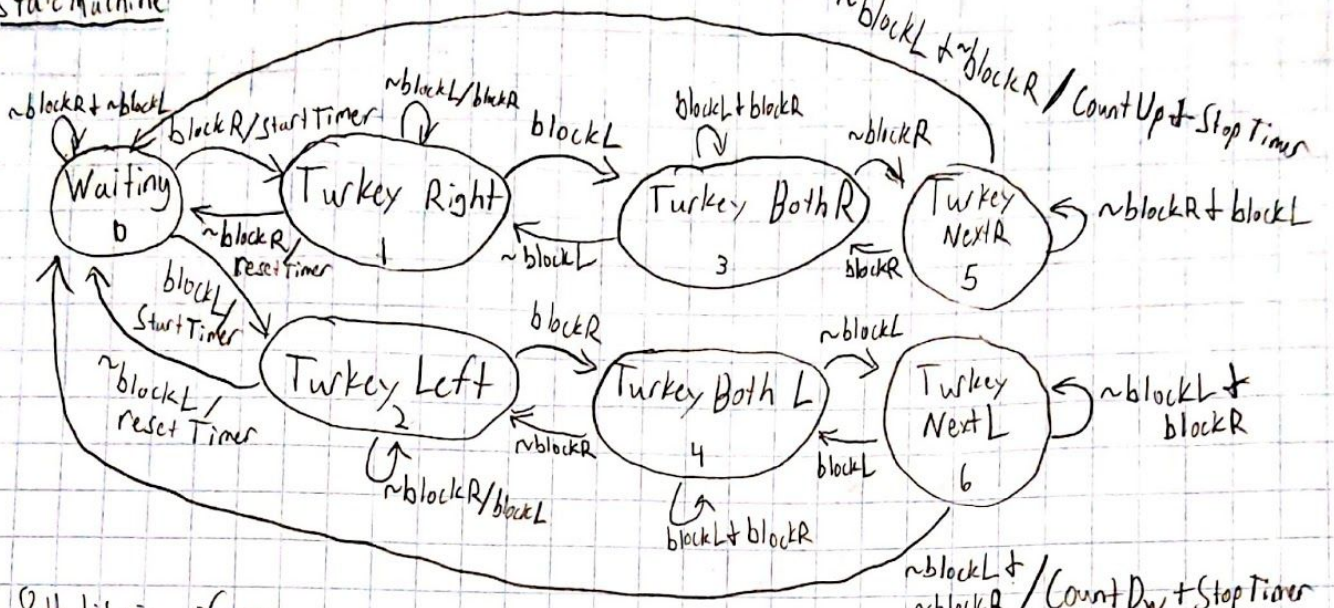
```
module m8_1(
    input [7:0] in,
    input [2:0] sel,
    output o
    );

    assign o = ((~sel[2]&~sel[1]&~sel[0]&in[0]) | (~sel[2]&~sel[1]&sel[0]&in[1]) |
    (~sel[2]&sel[1]&~sel[0]&in[2]) | (~sel[2]&sel[1]&sel[0]&in[3]) | (sel[2]&~sel[1]&~sel[0]&in[4]) |
    (sel[2]&~sel[1]&sel[0]&in[5]) | (sel[2]&sel[1]&~sel[0]&in[6]) | (sel[2]&sel[1]&sel[0]&in[7]));
endmodule

module hex7seg(
    input [3:0] n,
    output [6:0] seg
    );

    m8_1 a(.in({1'b0,n[0],n[0], 1'b0, 1'b0, ~n[0], 1'b0, n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[0]));
    m8_1 b(.in({1'b1,~n[0],n[0], 1'b0, ~n[0], n[0], 1'b0, 1'b0}), .sel({n[3], n[2], n[1]}), .o(seg[1]));
    m8_1 c(.in({1'b1,~n[0],1'b0, 1'b0, 1'b0, 1'b0, ~n[0], 1'b0}), .sel({n[3], n[2], n[1]}), .o(seg[2]));
    m8_1 d(.in({n[0],1'b0,~n[0], n[0], n[0], ~n[0], 1'b0, n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[3]));
    m8_1 e(.in({1'b0,1'b0,1'b0, n[0], n[0], 1'b1, n[0], n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[4]));
    m8_1 f(.in({1'b0,n[0],1'b0, 1'b0, n[0], 1'b0, 1'b1, n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[5]));
    m8_1 g(.in({1'b0,~n[0],1'b0, 1'b0, n[0], 1'b0, 1'b0, 1'b1}), .sel({n[3], n[2], n[1]}), .o(seg[6]));
endmodule
```

Lab Book :

Lab 6 -

- Turkeys cannot be between sensors w/o activation
- Only one turkey at a time

State Machine



8th bit is if neg