Jack Kapino

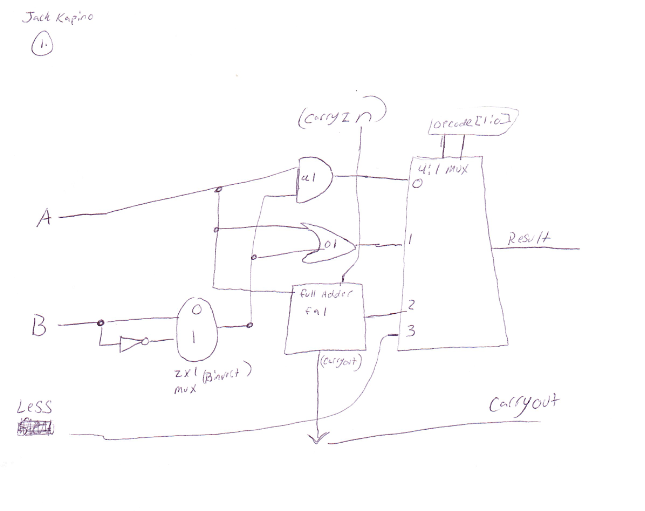
Project 2:

**Part 1: 1 bit alu for bits (0 to 2)**

**Description:**

One bit alu circuit receives two 1-bit inputs A,B. Opcode input tells the mux which operation will be connected to the output. This 1-bit alu can perform and, or, addition, subtraction operations. For the subtraction operation the input of b must be inverted this is done with binvert 2:1 multiplexer. The inverted result is sent to the adder where an additional 1 from the carryin is added. (a + binverted)+1. The output depends on the opcode given to the multiplexer. Opcodes { 00 = And, 01 = Or, 010 = addition, 110 = Subtraction}. Depending on the opcode given, output will be connected to one of the three gates producing result wire.

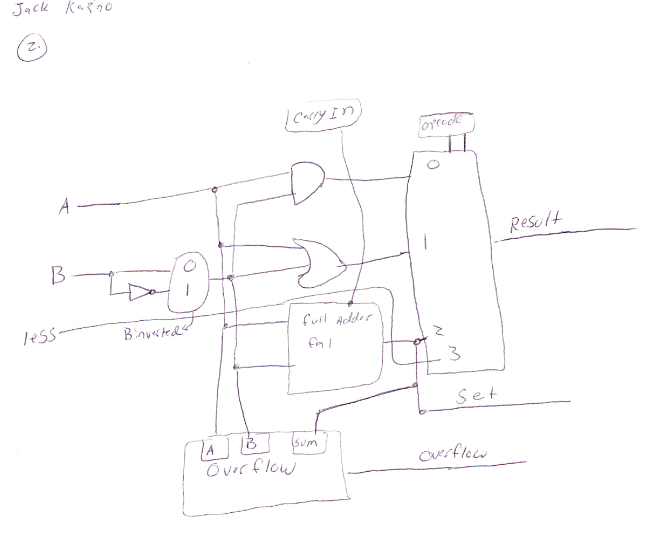
**Diagram:**

****

**Part 2: 1-Bit Alu Most Significant bit:**

**Description:** This circuit has inputs (a,b, less ). Outputs are result, overflow, set. This circuit is similar to part 1 but includes overflow detection and a fifth operation SLT “Set on Less than” is included in the multiplexer. If A<B slt results in a 1 otherwise value = 0.

**Diagram:**



**Part 3: 4-bit alu:**

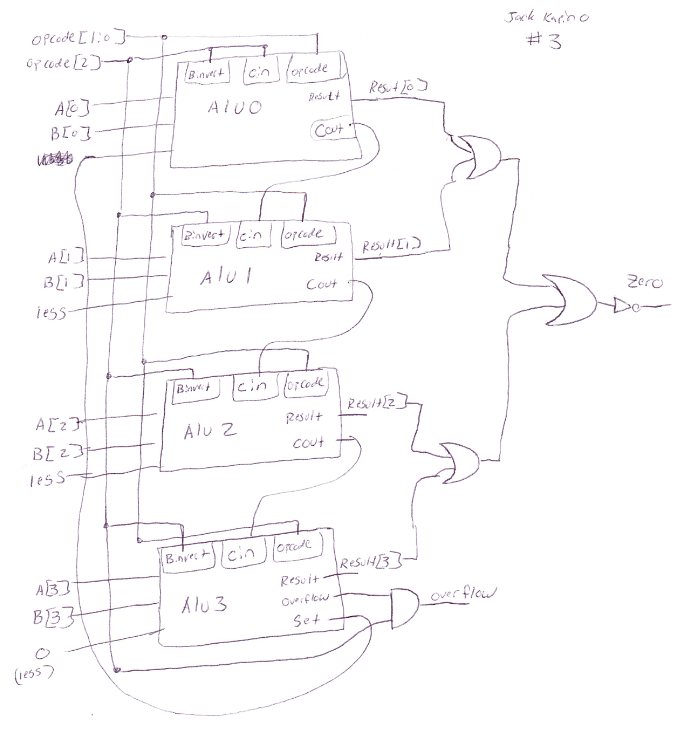
**Description:**

4 bit Alu inputs include A,B and an opcode to select the corresponding operation. The result output gives you the result of the selected 4:1 mux operation. Zero output gives 1 when arithmetic operation is zero, and gives 0 for non-zero results and logic operations. Overflow output determines if overflow has occurred. Overflow is detected when the sign of the operands doesn’t match that of the output. Overflow detection only for arithmetic operations can be accomplished by using an And Gate with the second bit of the opcode, which will always be 1 for an arithmetic operation.

4 bit Alu opcodes:

000 =And, 001 = or, 010 = Addition, 110 = Subtraction, 111 = SLT

**Diagram:**

****

**Part 4: Test Module + Verilog Code:**

**Verilog Code:**

**//Half adder**

**module half\_adder(x,y,S,C);**

**input x,y;**

**output S, C;**

**xor x1(S,x,y);**

**and a1(C,x,y);**

**endmodule**

**//Full adder**

**module full\_adder(x,y,z,S,C);**

**input x,y,z;**

**output S, C;**

**half\_adder ha1(x,y,u,v);**

**half\_adder ha2(u,z,S,w);**

**or o1(C,v,w);**

**endmodule**

**//2:1 Mux**

**module two\_oneMux(sel,op,out);**

**input sel;**

**input [1:0] op;**

**output out;**

**not n1(inverted, sel);**

**and a1(and1, op[0], inverted);**

**and a2(and2, op[1], sel);**

**or o1(out, and1, and2);**

**endmodule**

**//4:1 Mux**

**module muxFour\_One(sel,operation,out);**

**input [1:0] sel;**

**input [3:0] operation;**

**output out;**

**two\_oneMux muxOne(sel[0], {operation[0], operation[1]}, one);**

**two\_oneMux muxTwo(sel[0], {operation[2], operation[3]}, two);**

**two\_oneMux muxThree(sel[1], {one, two}, out);**

**endmodule**

**//Part 1**

**module One\_Bit\_Alu(a,b,cin,opcode,binvert,less,result,cout);**

**input a,b,cin;**

**input [1:0] opcode;**

**input binvert,less;**

**output result,cout;**

**not notb(bbar, b);**

**two\_oneMux mux2\_1(binvert, {bbar, b}, binverted);**

**muxFour\_One mux(opcode, {and0, or1, sum2, less}, result);**

**full\_adder fa1( a,binverted, cin, sum2, cout);**

**and a1( and0, a, binverted);**

**or o1( or1, a, binverted);**

**endmodule**

**module Overflow(a,b,total,overflow);**

**input a, b, total;**

**output overflow;**

**not n1(aprime, a);**

**not n2(bprime, b);**

**not n3(totalprime, total);**

**and a1(and1, aprime, bprime);**

**and a2(negative\_sum, and1, total);**

**and a3(and2, a,b);**

**and a4(positive\_sum, and2, totalprime);**

**or o1(overflow, negative\_sum, positive\_sum);**

**endmodule**

**//Part 2**

**module Msb\_Alu(a,b,cin,opcode,binvert,less,set,result,overflow);**

**input a,b, cin;**

**input [1:0] opcode;**

**input binvert,less;**

**output set,result,overflow;**

**not n1(bbar, b);**

**two\_oneMux inverter(binvert, {bbar, b}, binverted);**

**muxFour\_One mux(opcode, {out1, out2, sum3, less}, result);**

**full\_adder fa1(a, binverted, cin, sum3, cout);**

**or or\_msb(out2, a, binverted);**

**and and\_msb(out1, a, binverted);**

**not n2(outpt,sum3);**

**not n3(set,outpt);**

**Overflow over1(a, binverted, sum3, overflow);**

**endmodule**

**//Part 3**

**module Four\_bitAlu(a,b,opcode,result,overflow,zero);**

**input [3:0] a,b;**

**input [2:0] opcode;**

**output [3:0] result;**

**output overflow,zero;**

**One\_Bit\_Alu alu0(a[0],b[0],opcode[2],opcode[1:0],opcode[2],set,result[0],carry1);**

**One\_Bit\_Alu alu1(a[1],b[1],carry1,opcode[1:0],opcode[2],1'b0,result[1],carry2);**

**One\_Bit\_Alu alu2(a[2],b[2],carry2,opcode[1:0],opcode[2],1'b0,result[2],carry3);**

**Msb\_Alu alu3(a[3],b[3],carry3, opcode[1:0],opcode[2], 1'b0 , set, result[3], overflow\_out);**

**and a1(overflow, overflow\_out, opcode[1]);**

**or o1(left, result[0], result[1]);**

**or o2(right, result[2], result[3]);**

**or o3(notzero, left, right);**

**not n1(zero, notzero);**

**endmodule**

**//Part 4 Testing ALu and output**

**module test\_Alu;**

**reg signed [3:0] a,b;**

**reg [2:0] opcode;**

**wire signed [3:0] result;**

**wire overflow,zero;**

**Four\_bitAlu fba1( a, b,opcode, result, overflow, zero);**

**initial**

**begin**

**$display(" Opcode A B Result Overflow Zero ");**

**$monitor(" %b %d %b %d %b %d %b %d %d", opcode, a, a, b, b, result, result, overflow, zero);**

**opcode = 0; a = 1; b = 2;**

**#10; opcode = 0; a = -3; b = 7;**

**#10; opcode = 0; a = -1; b = 3;**

**#10; opcode = 1; a = 2; b = 5;**

**#10; opcode = 1; a = -1; b = -1;**

**#10; opcode = 1; a = -4; b = 3;**

**#10; opcode = 2; a = 2; b = 1;**

**#10; opcode = 2; a = 5; b = 5;**

**#10; opcode = 2; a = 4; b = 4;**

**#10; opcode = 2; a = 7; b = -7;**

**#10; opcode = 6; a = 6; b = 2;**

**#10; opcode = 6; a = 5; b = 7;**

**#10; opcode = 6; a = -2; b = -2;**

**#10; opcode = 6; a = -3; b = 3;**

**#10; opcode = 6; a = 1; b = -1;**

**#10; opcode = 7; a = 2; b = 3;**

**#10; opcode = 7; a = 3; b = 2;**

**#10; opcode = 7; a = 1; b = 2;**

**//#10; opcode = 7; a = 1; b = 9;**

**#10; opcode = 7; a = -2; b =5;**

**end**

**endmodule**

**Program output:**

