Sean Clifford
11/20/24

CS 354 Project 2

```
module multiplexer_2x1 (X, Y, Z, out);
   input X, Y, Z;
   output out;
   wire c1, c2;
   wire not_z;

   // NOT gate
   not n1 (not_z, Z);

   // AND gates
   and a1 (c1, X, not_z);
   and a2 (c2, Y, Z);

   // OR gate
   or o1 (out, c1, c2);
endmodule

module multiplexer_4x1(X, Y, Z, W, S1, S2, out);
   input X, Y, Z, W;
   input S1, S2;
   output out;
   wire c1, c2;

   // 2x1 mux
   multiplexer_2x1 m1 (X, Y, S1, c1);
   multiplexer_2x1 m2 (Z, W, S1, c2);

   // combine the 2x1 muxes
   multiplexer_2x1 m3 (c1, c2, S2, out);
endmodule

module half_adder (A, B, S, C);
   input A, B;
   output S, C;

   xor g1 (S, A, B);
   and g2 (C, A, B);
endmodule
```

```
module full_adder (X, Y, Z, S, C);
   // X, Y, Carry in, Sum, Carry out
   input X, Y, Z;
   output S, C;
   wire S1, C1, C2;

   half_adder HA1 (X, Y, S1, C1),
              HA2 (S1, Z, S, C2);
   or g1(C, C1, C2);
endmodule

module bit_alu_carry(A, B, L, CI, OP, R, CO);
   // A, B, Less, Carry in, Operation, Result, Carry out
   input A, B, L, CI;
   input [3:0] OP;
   output R, CO;
   // not_a, a_mux_out, not_b, b_mux_out, a_b_and, a_b_or, a_b_add, Nand operation,
Nand result
   wire not_a, a_mux_out, not_b, b_mux_out, a_b_and, a_b_or, a_b_add, NA, N1, R1;

   // A
   not (not_a, A);
   multiplexer_2x1 m1 (A, not_a, OP[3], a_mux_out);

   // B
   not n1 (not_b, B);
   multiplexer_2x1 m2 (B, not_b, OP[2], b_mux_out);

   // Operations
   and a1 (a_b_and, a_mux_out, b_mux_out);
   or o1 (a_b_or, a_mux_out, b_mux_out);
   full_adder fa1 (a_mux_out, b_mux_out, CI, a_b_add, CO);

   // Multiplexer
   multiplexer_4x1 m3 (a_b_and, a_b_or, a_b_add, L, OP[0], OP[1], R1);

   // NAND
   and a2 (NA, OP[0], OP[3]);
   not n3 (N1, a_b_and);
   multiplexer_2x1 m4 (R1, N1, NA, R);
endmodule
```

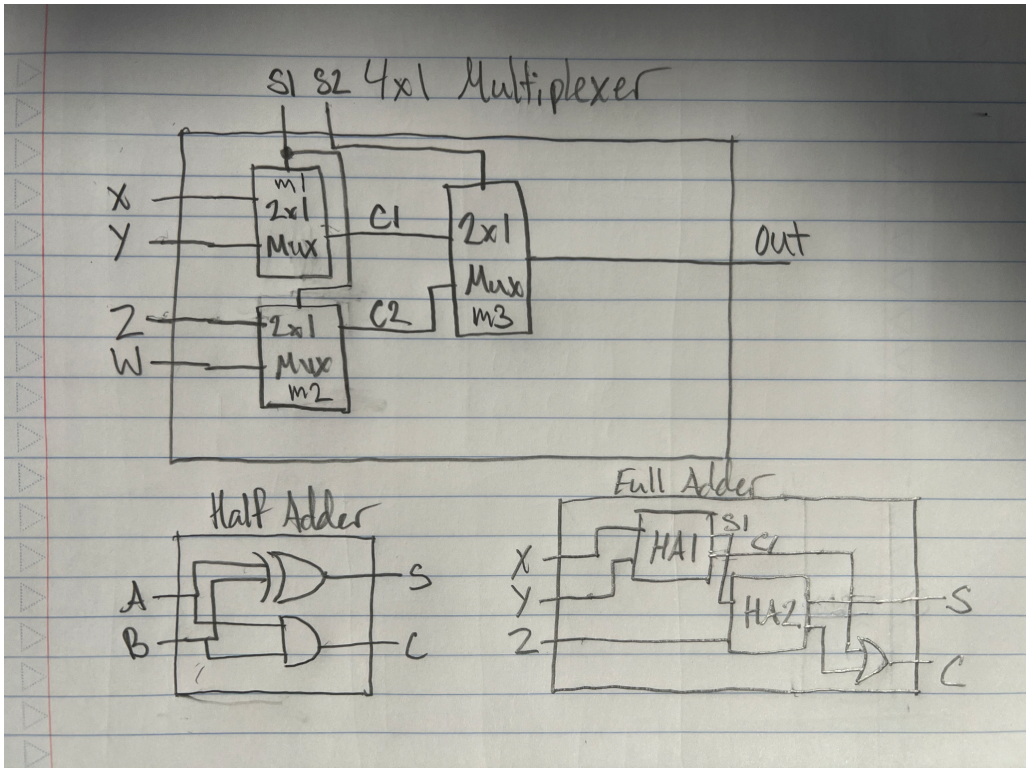
```
module bit_alu_set(A, B, L, CI, OP, R, S, O);
   // A, B, Less, Carry in, Operation, Result, Set, Overflow
   input A, B, L, CI;
   input [3:0] OP;
   // result, set, overflow
   output R, S, O;
   // not_a, a_mux_out, not_b, b_mux_out, a_b_and, a_b_or, a_b_add, Carry Out,
Overflow xor out, Nand operation, Nand result
   wire not_a, a_mux_out, not_b, b_mux_out, a_b_and, a_b_or, a_b_add, CO, O1, NA, N1,
R1;

   // A
   not n1 (not_a, A);
   multiplexer_2x1 m1 (A, not_a, OP[3], a_mux_out);

   // B
   not n2 (not_b, B);
   multiplexer_2x1 m2 (B, not_b, OP[2], b_mux_out);

   // Operations
   and a1 (a_b_and, a_mux_out, b_mux_out);
   or o1 (a_b_or, a_mux_out, b_mux_out);
   full_adder fa1 (a_mux_out, b_mux_out, CI, a_b_add, CO);
   assign S = a_b_add;

   // Multiplexer
   multiplexer_4x1 m3 (a_b_and, a_b_or, a_b_add, L, OP[0], OP[1], R1);

   // NAND
   and a2 (NA, OP[0], OP[3]);
   not n3 (N1, a_b_and);
   multiplexer_2x1 m4 (R1, N1, NA, R);

   // Overflow
   xor x1 (O1, CI, CO);
   and a3 (O, O1, OP[1]);
endmodule

module ALU(OP, A, B, R, Z, O);
   input [3:0] A, B;
   input [3:0] OP;
```

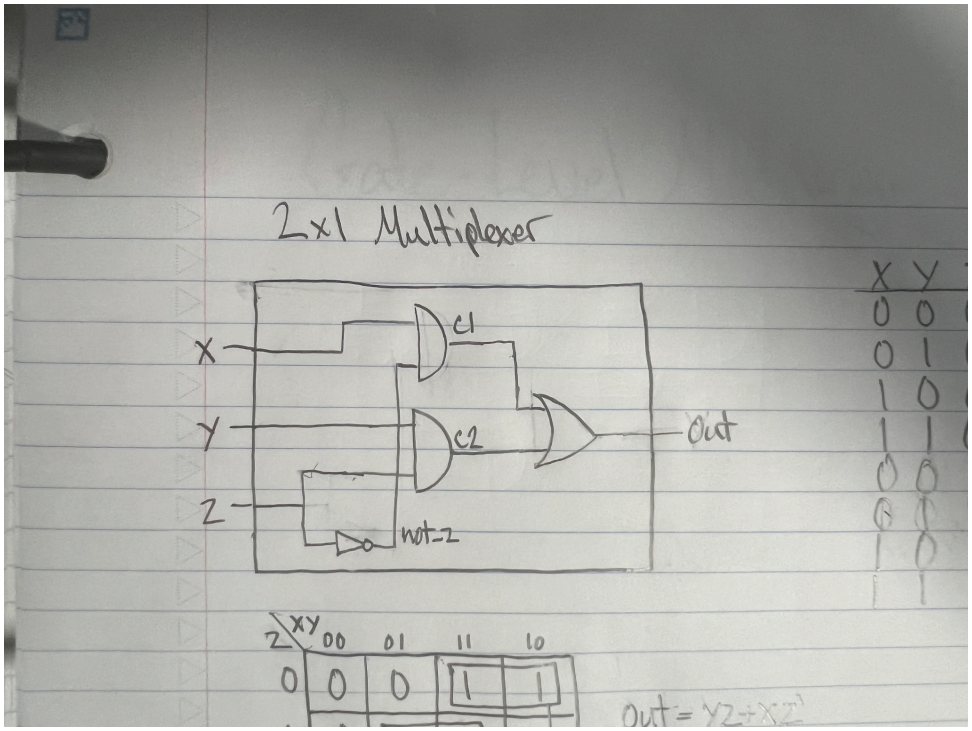
```
   // zero and overflow
   output Z, O;
   // result
   output [3:0] R;
   // carry out and set
   wire CO1, CO2, CO3, S;
   // Bit 0
   bit_alu_carry ba1 (A[0], B[0], S, OP[2], OP, R[0], CO1);
   // Bit 1
   bit_alu_carry ba2 (A[1], B[1], 1'b0, CO1, OP, R[1], CO2);
   // Bit 2
   bit_alu_carry ba3 (A[2], B[2], 1'b0, CO2, OP, R[2], CO3);
   // Bit 3
   bit_alu_set ba4 (A[3], B[3], 1'b0, CO3, OP, R[3], S, O);

   // Zero
   nor o1 (Z, R[0], R[1], R[2], R[3]);
endmodule


module testALU;
 reg signed [3:0] a;
 reg signed [3:0] b;
 reg [3:0] op;
 wire signed [3:0] result;
 wire zero,overflow;
 ALU alu (op,a,b,result,zero,overflow);
 initial
   begin
    $display("op   a        b        result   zero overflow");
    $monitor ("%b %b(%d) %b(%d) %b(%d) %b
%b",op,a,a,b,b,result,result,zero,overflow);
          op = 4'b0000; a = 4'b0111; b = 4'b0010;  // AND
           #1 op = 4'b0001; a = 4'b0101; b = 4'b0010;  // OR
           #1 op = 4'b0010; a = 4'b0101; b = 4'b0001;  // ADD
       #1 op = 4'b0010; a = 4'b0111; b = 4'b0001;  // ADD overflow (8+1=-8)
       #1 op = 4'b0110; a = 4'b0101; b = 4'b0001;  // SUB
       #1 op = 4'b0110; a = 4'b1111; b = 4'b0001;  // SUB
       #1 op = 4'b0110; a = 4'b1111; b = 4'b1000;  // SUB no overflow (-1-(-8)=7)
       #1 op = 4'b0110; a = 4'b1110; b = 4'b0111;  // SUB overflow (-2-7=7)
       #1 op = 4'b0111; a = 4'b0101; b = 4'b0001;  // SLT
       #1 op = 4'b0111; a = 4'b0001; b = 4'b0011;  // SLT
```

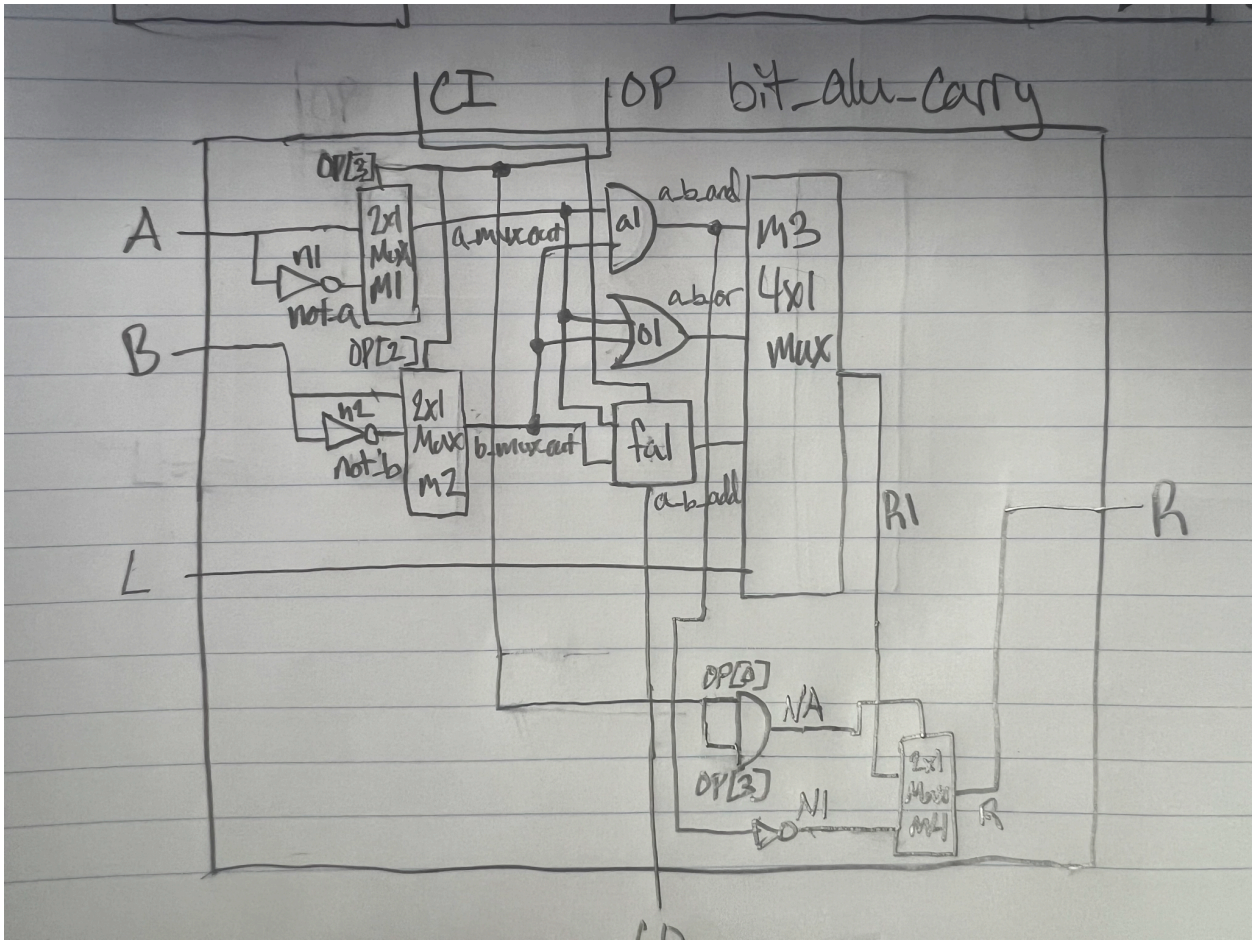
```
        #1 op = 4'b0111; a = 4'b1101; b = 4'b0110;  // SLT overflow (-3-6=7 => SLT=0)
        #1 op = 4'b1000; a = 4'b0101; b = 4'b0001;  // NOR
        #1 op = 4'b1101; a = 4'b0101; b = 4'b0001;  // NAND
    end
endmodule
```

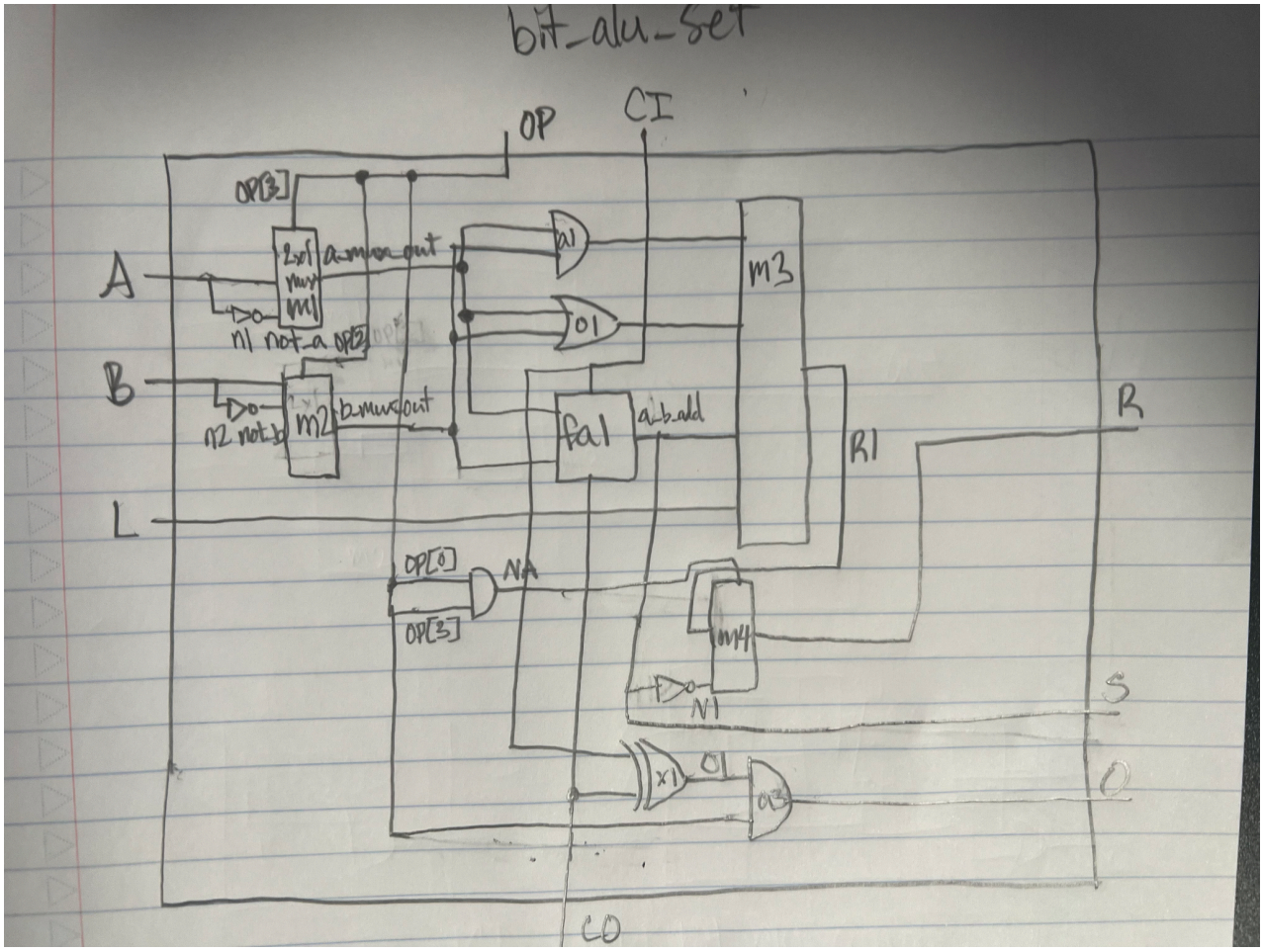
vvp a.out

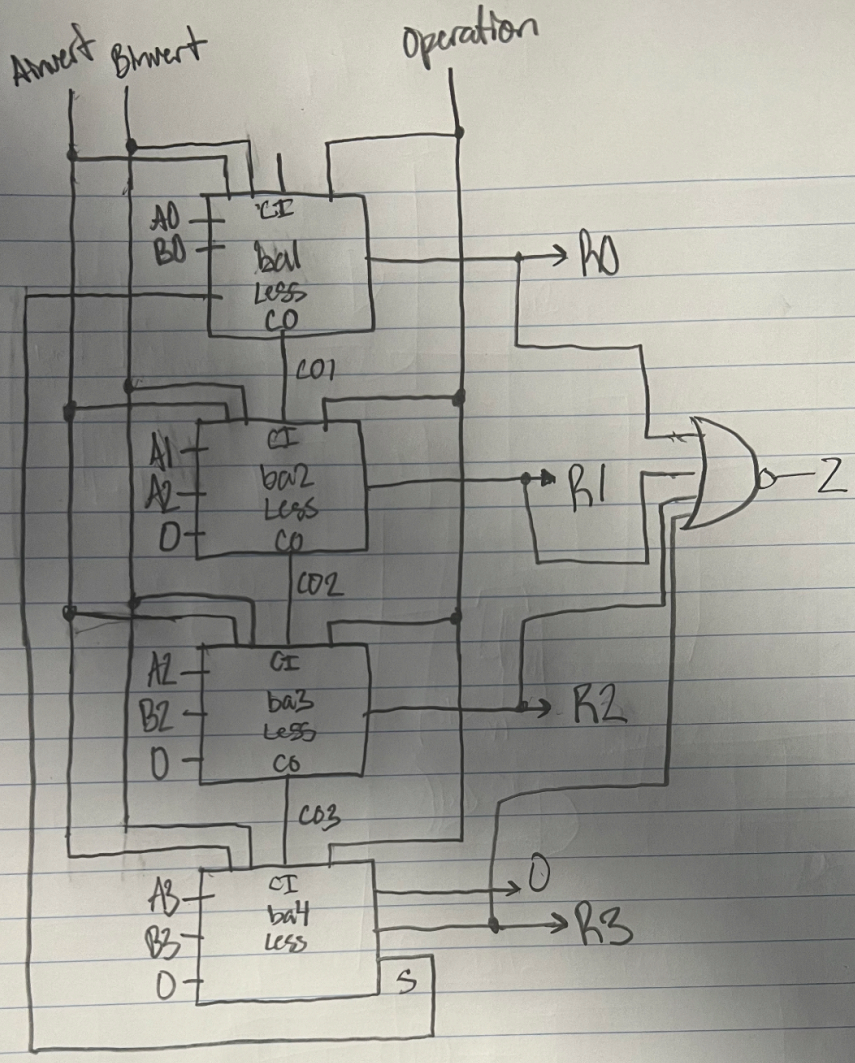
op	a	b	result	zero	overflow
0000	0111(7)	0010(2)	0010(2)	0	0
0001	0101(5)	0010(2)	0111(7)	0	0
0010	0101(5)	0001(1)	0110(6)	0	0
0010	0111(7)	0001(1)	1000(-8)	0	1
0110	0101(5)	0001(1)	0100(4)	0	0
0110	1111(-1)	0001(1)	1110(-2)	0	0
0110	1111(-1)	1000(-8)	0111(7)	0	0
0110	1110(-2)	0111(7)	0111(7)	0	1
0111	0101(5)	0001(1)	0000(0)	1	0
0111	0001(1)	0011(3)	0001(1)	0	0
0111	1101(-3)	0110(6)	0000(0)	1	1
1000	0101(5)	0001(1)	0000(0)	1	0
1101	0101(5)	0001(1)	0101(5)	0	0


S1 S2 4x1 Multiplexer
X
Y
Z
W
m1
2x1
Mux
C1
2x1
Mux
m3
Out
2x1
Mux
m2
C2
Half Adder
A
B
S
C
Full Adder
X
Y
Z
HA1
S1
C1
HA2
S
C


2x1 Multiplexer
X
Y
Z
C1
C2
Out
not_z
XY
Z
00
01
11
10
0
0
0
1
1
Out = YZ + XZ'
X Y
0 0
0 1
1 0
1 1
0 0
0 1
1 0
1 1


CI
OP
bit_alu_carry
A
B
L
OP[3]
2x1 Mux M1
m1
not_a
a_mux_out
a1
a_b_and
o1
a_b_or
M3 4x1 mux
OP[2]
2x1 Mux M2
m2
not_b
b_mux_out
fa1
a_b_add
R1
R
OP[0]
NA
OP[3]
N1
2x1 Mux M4
R


bit_alu_set
OP
CI
OP[3]
A
2x1 mux m1
a_mux_out
a1
m3
B
o1
n1 not_a OP[2]
n2 not_b
m2
b_mux_out
fa1
a_b_add
R1
R
L
OP[0]
NA
OP[3]
m4
N1
S
x1
o1
o3
O
CO



ALU Control Lines

AI	BI	OP	Instruction
0	0	00	and
0	0	01	or
0	0	10	add
0	1	10	sub
0	1	11	slt
1	1	00	nor
1	1	01	nand