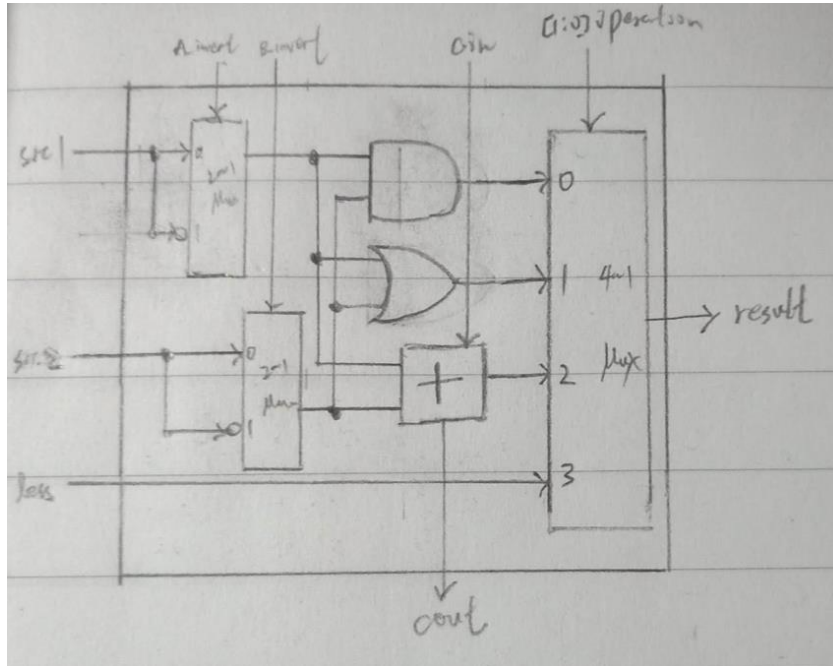


# Computer Organization Lab 1: 32-bit ALU

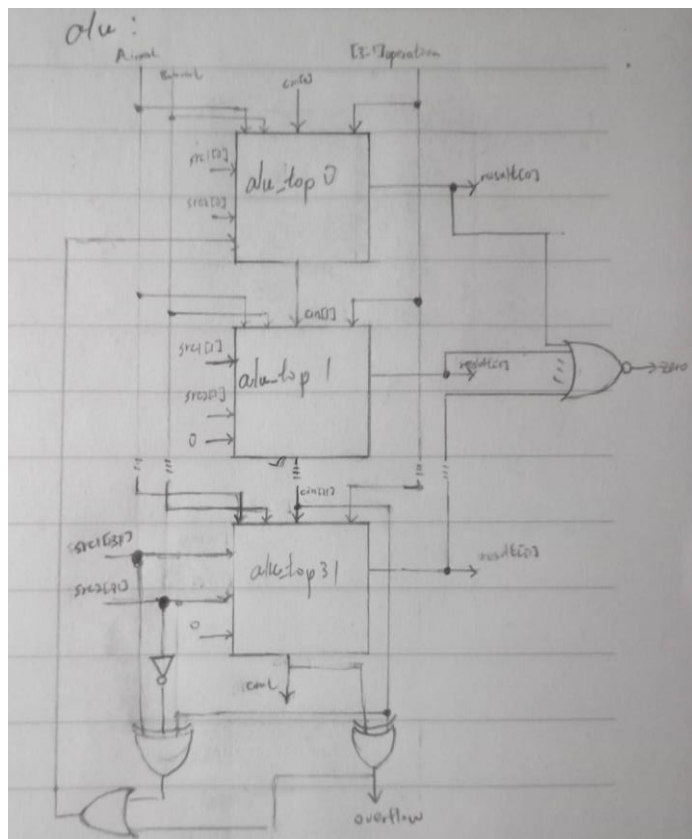
Student ID:                      Name:

(1) Architecture diagram

module alu\_top:



module alu:



## (2) Detailed description of the implementation

In this alu, we need to implement following operations.

operation	ALU_control	symbol	modify
AND	0000	$a \& b$	
OR	0001	$a   b$	
ADD	0010	$a + b$	$s = a \wedge b \wedge c_i$ $c_{i+1} = (a \& b)   (c \& (a \wedge b))$
SUB	0110	$a - b$	$a + b' + 1$
NOR	1100	$(a   b)'$	$a' \& b'$
NAND	1101	$(a \& b)'$	$a'   b'$
SLT	0111	$a < b?$	$a - b = a + b' + 1 < 0?$

For the inputs 32-bits ALU (module alu),

set cin[0] into 1 if operation is SUB or SLT,

set A\_invert into 1 if operation is NOR or NAND,

set B\_invert into 1 if operation is SUB or SLT or NOR or NAND,

set operation in 1-bit ALU (alu\_top) into 0 if operation is AND or NOR,

set operation in 1-bit ALU (alu\_top) into 1 if operation is OR or NAND,

set operation in 1-bit ALU (alu\_top) into 2 if operation is ADD or SUB,

set operation in 1-bit ALU (alu\_top) into 3 if operation is SLT.

For the outputs 32-bits ALU (module alu),

set zero into 1 if result is 0,

set cout into cin[32] if operation is ADD or SUB,

set overflow into (cin[31] xor cin[32]) if operation is ADD or SUB.

And each 1-bit ALU alu\_top[i] where  $0 \leq i \leq 31$ ,

input src1[i] to port src1,

input src2[i] to port src2,

input A\_invert to port A\_invert,

input B\_invert to port B\_invert,

input cin[i] to port cin, port result will output result[i],

port cout will output cin[i + 1].

The port less in 1-bit ALU alu\_top[i] where  $1 \leq i \leq 31$  are all input with 0.

For  $i=0$ , input ((src1[31] xor complement of src2[31] xor cin[31]) or overflow) to port less.

$$\text{SUB}(\text{src1}[31], \text{src2}[31]) = (\text{src1}[31] \wedge \neg \text{src2}[31] \wedge \text{cin}[31])$$

$$\text{less}[0] = \text{SUB}(\text{src1}[31], \text{src2}[31]) | \text{overflow}$$

Below is Truth table:

overflow \ sub[31]	0	1
	0	1
0	0	1
1	1	1

```

43 assign cin[0] = (ALU_control == 4'b0110 || ALU_control == 4'b0111);
44 assign A_invert = (ALU_control == 4'b1100 || ALU_control == 4'b1101);
45 assign B_invert = (ALU_control == 4'b0110 || ALU_control == 4'b1100 || ALU_control == 4'b1101 || ALU_control == 4'b1110);
46 assign operation = (ALU_control == 4'b0000 || ALU_control == 4'b1100) ? 2'b00
47 : ((ALU_control == 4'b0001 || ALU_control == 4'b1101) ? 2'b01
48 : ((ALU_control == 4'b0010 || ALU_control == 4'b0110) ? 2'b10
49 : 2'b11);
50
51 assign zero = (result == 32'b0);
52 assign cout = (ALU_control == 4'b1100 || ALU_control == 4'b1101 || ALU_control == 4'b1110);
53 assign overflow = (ALU_control == 4'b0110 || ALU_control == 4'b0111 || ALU_control == 4'b1100 || ALU_control == 4'b1101 || ALU_control == 4'b1110);
54 alu_top alu_top0(
55     .src1(src1[0]),
56     .src2(src2[0]),
57     // dest[0] = sum[31] | overflow
58     .less((src1[31] ^ src2[31] & cin[0]) || (src1[31] & src2[31] & cin[0])),
59     .A_invert(A_invert),
60     .B_invert(B_invert),
61     .cin(cin[0]),
62     .operation(operation),
63     .result(result[0]),
64     .cout(cout[0])
65 );
66 generate
67     for (i = 1; i < 32; i = i + 1) begin

```

Successfully pass all data.

### (3) Commands for executing your source codes

iverilog -o basic.vvp testbench.v alu.v alu\_top.v

### (4) Problems encountered and solutions

In original alu.v, the output result, zero, cout, overflow have reg type. When I do the compile, it comes up a reg result error. It seems that reg type can't be assign directly. To solve this problem, I delete the reg type of these outputs. It works successfully and shows the result is true.

### (5) Lesson learnt

How to descript an 32-bits ALU by Verilog.