CSE 2021 COMPUTER ORGANIZATION

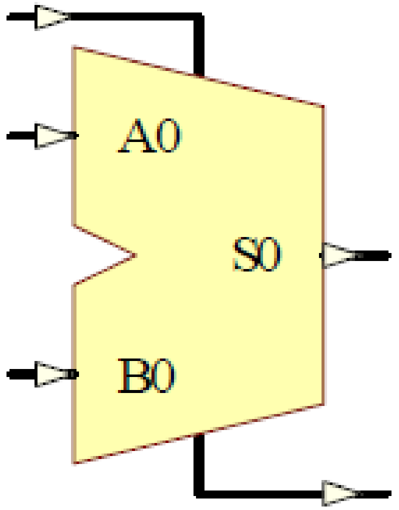
HUGH CHESSER
CSEB 1012U

W5-W

Combinational Logic: Design of a 1-bit adder (2)

Step 2: Derive the Boolean expression for each output from the truth table

INPUTS			**OUTPUTS**	
a	**b**	**c (CarryIn)**	**CarryOut**	**Sum**
0	**0**	**0**	**0**	**0**
0	**0**	**1**	**0**	**1**
0	**1**	**0**	**0**	**1**
0	**1**	**1**	**1**	**0**
1	**0**	**0**	**0**	**1**
1	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**0**
1	**1**	**1**	**1**	**1**



$$\text{Sum} = \bar{a}\,\bar{b}\,c + \bar{a}\,b\,\bar{c} + a\,\bar{b}\,\bar{c} + abc$$
$$\text{Carry-Out} = \bar{a}\,bc + a\,\bar{b}\,c + ab\,\bar{c} + abc$$

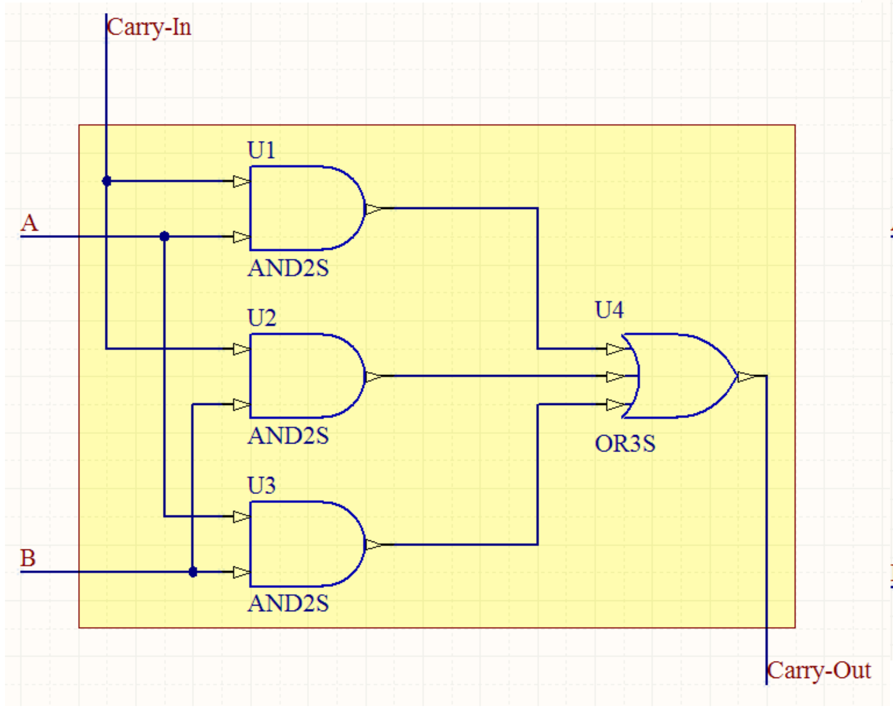
Combinational Logic: Design of a 1-bit adder (3)

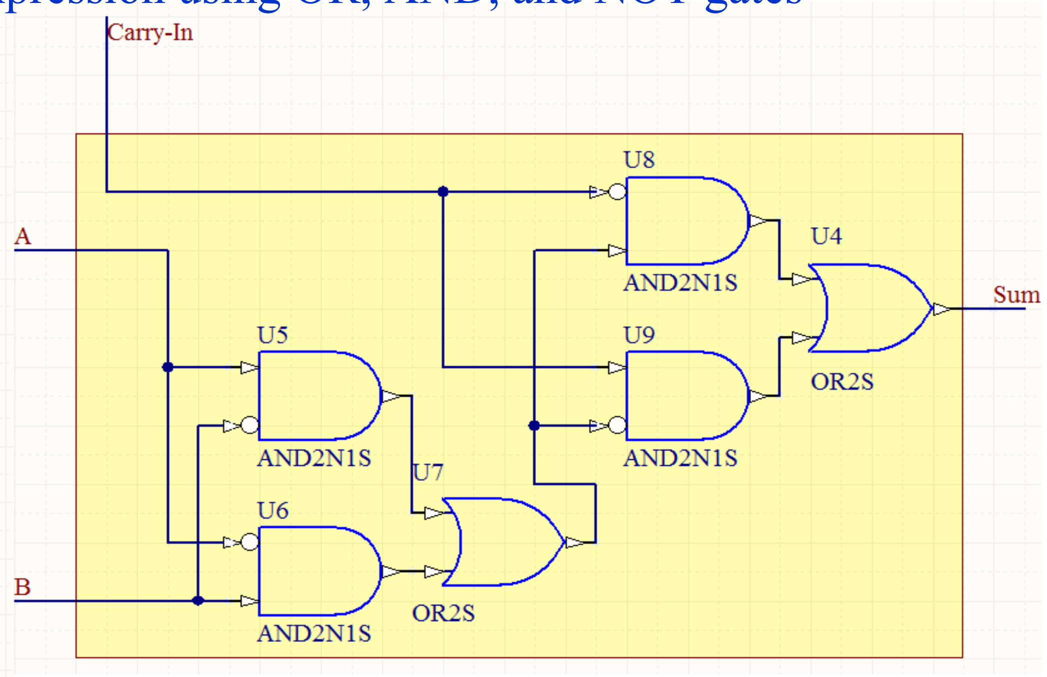
Step 3: Simplify the Boolean expression

$$\text{Carry-Out} = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc = bc + ac + ab$$

$$\text{Sum} = (\bar{a}\bar{b} + ab)c + (a\bar{b} + \bar{a}b)\bar{c} = \overline{(a\bar{b} + \bar{a}b)}c + (a\bar{b} + \bar{a}b)\bar{c}$$

Step 4: Implement the simplified Boolean expression using OR, AND, and NOT gates





Activity: Implement the hardware for the Sum output of the 1-bit adder

Agenda for Today

- Concerns from Prof. Roumani
- 1-bit ALU – Logic Design

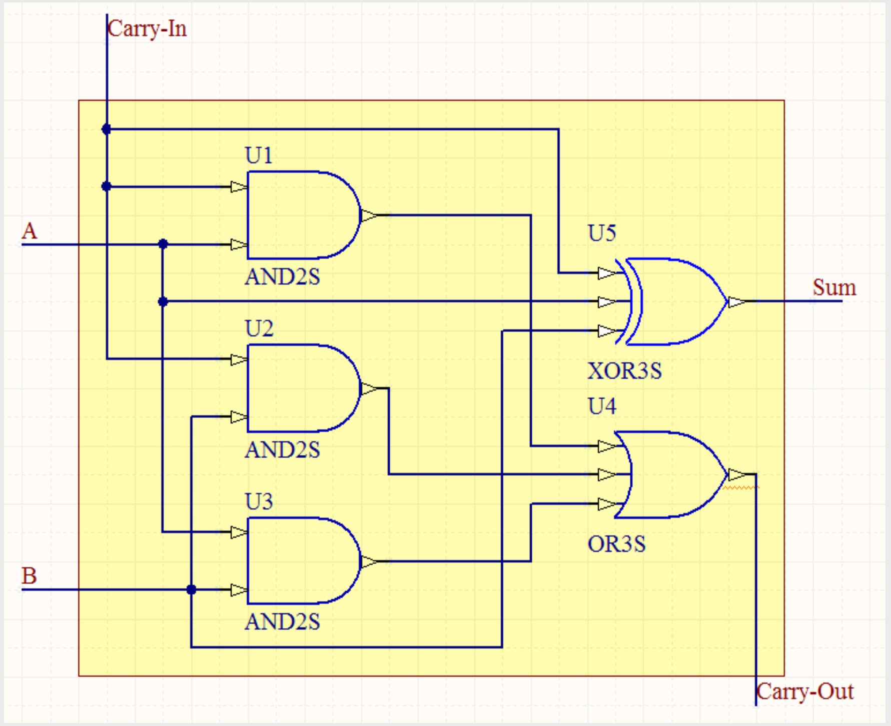
Patterson: Appendix C

Prof. Roumani's Concerns

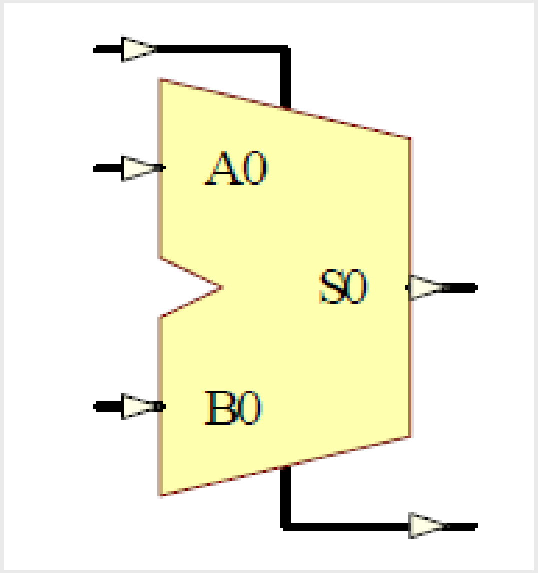
- Not enough students are doing the pre-lab activities at home and as a consequence not very many students are completing the Lab exercises on time
- If you labs have been manually marked, you can pick them up from the TA either at the next lab session or during his office hours (W 16:00 – 17:00) – NO ONE has done this
- All labs have been posted on ePost for Labs A and B. Manually marked Lab C's are to be posted in a few days

1-bit adder

- Recall the digital circuit of a 1-bit adder
- We will enhance the 1-bit adder to develop a prototype ALU for MIPS



Digital Circuit of a 1-bit adder



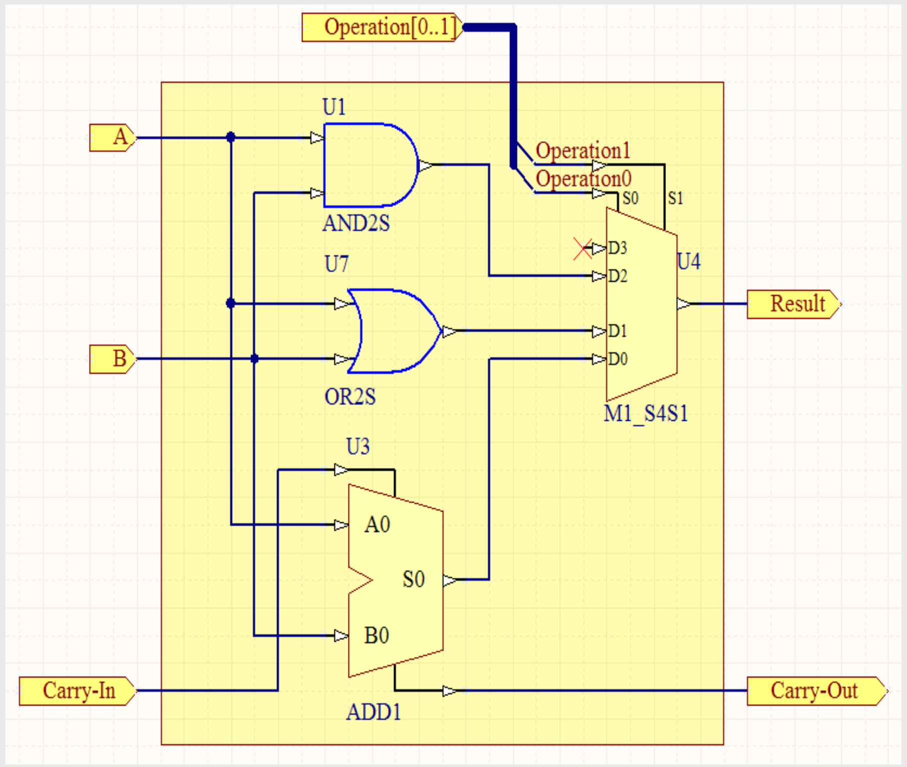
Schematic of a 1-bit adder

1-bit ALU with AND, OR, and Addition

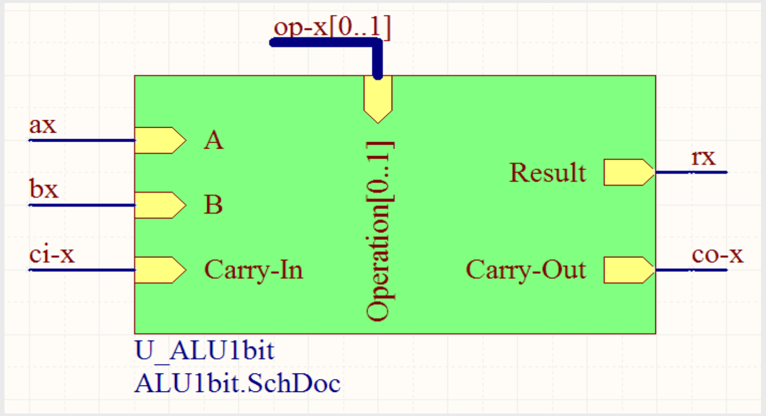
— The 1-bit adder is supplemented with AND and OR gates

— A multiplexer controls which gate is connected to the output

ALU Control Lines		Result
Carry In	**Operation**	
0	**0 = $(00)_{two}$**	**add**
0	**1 = $(01)_{two}$**	**OR**
0	**2 = $(10)_{two}$**	**AND**



1-bit ALU with AND, OR, and Addition capability



Schematic

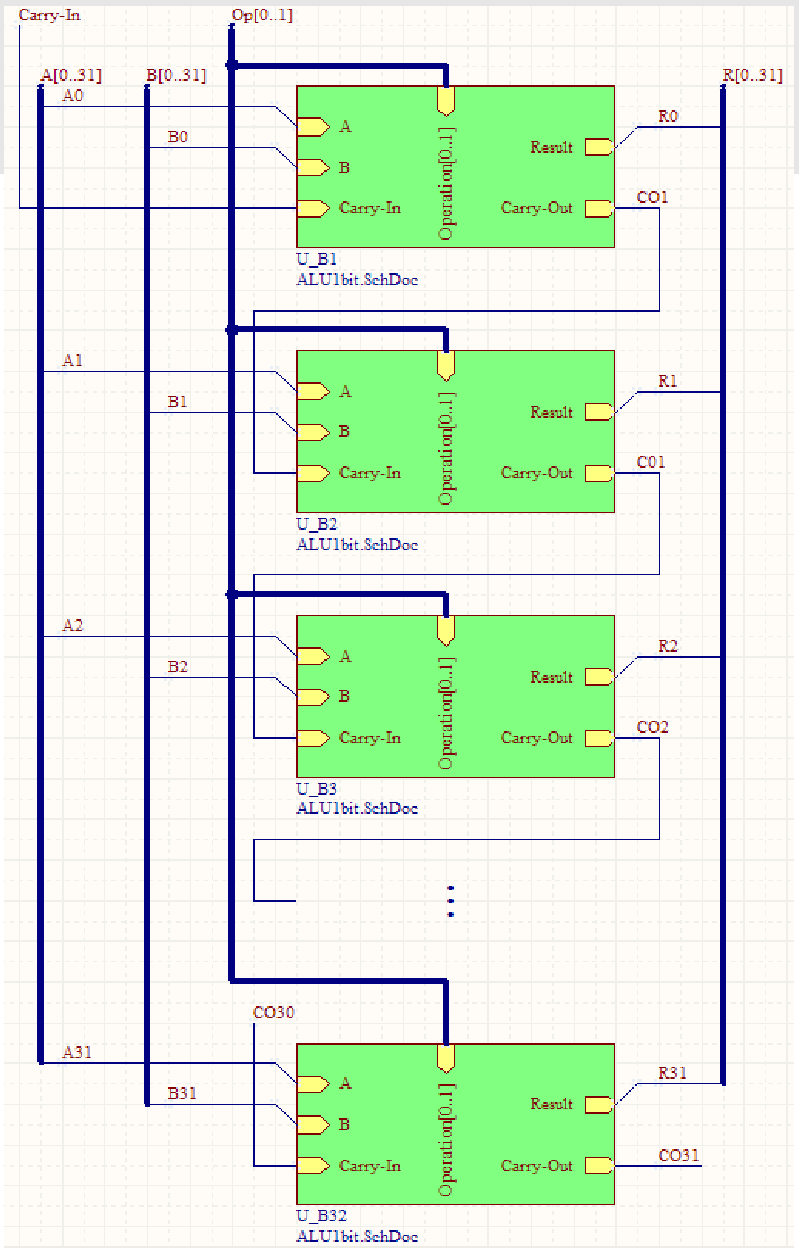
32-bit ALU w/ AND, OR, and ADD

— The 1-bit ALU can be cascaded together to form a 32 bit ALU

— Which operation is performed is controlled by the Operation bus

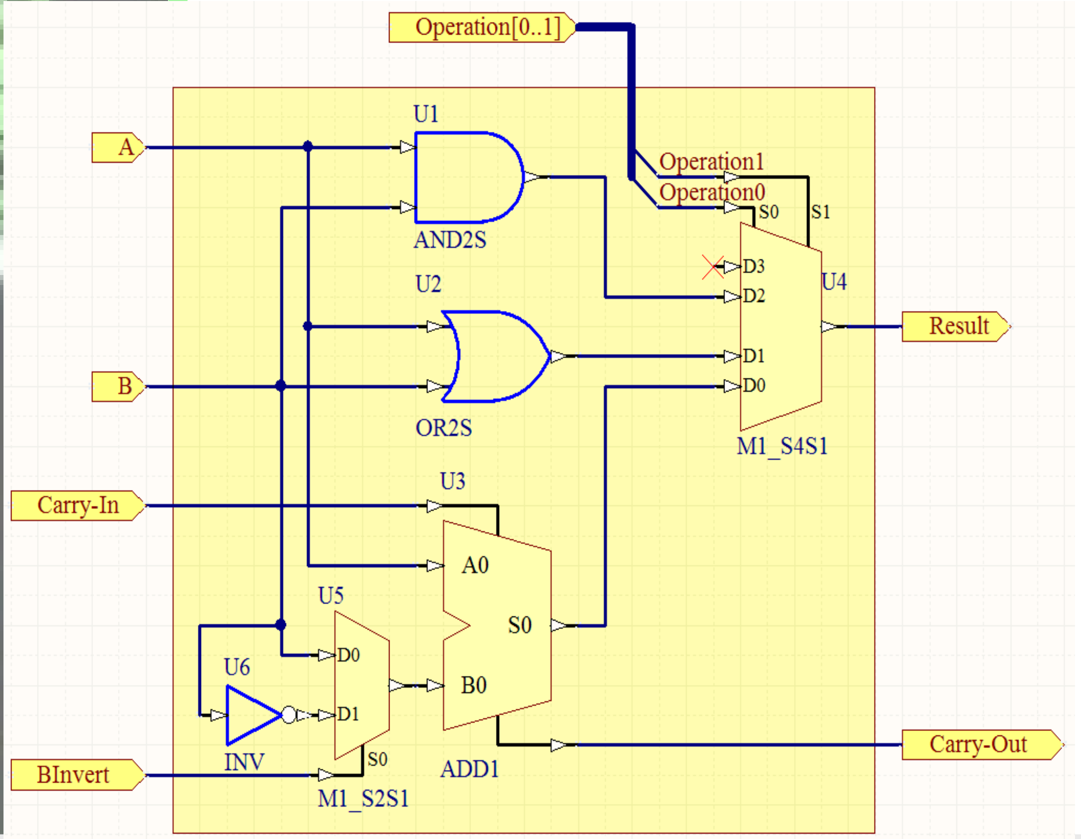
ALU Control Lines		Result
Carry In	**Operation**	
0	**$0 = (00)_{two}$**	**add**
0	**$1 = (01)_{two}$**	**OR**
0	**$2 = (10)_{two}$**	**AND**

— The designed 32-bit ALU is still missing the subtraction, slt (set if less than), and conditional branch operations



1-bit ALU with AND, OR, Addition, and Subtraction

- Recall that subtraction is performed using 2's complement arithmetic
- We calculate the 2's compliment of the sub-operand and add to the first operand



1-bit ALU with AND, OR, Addition, and Subtraction capability

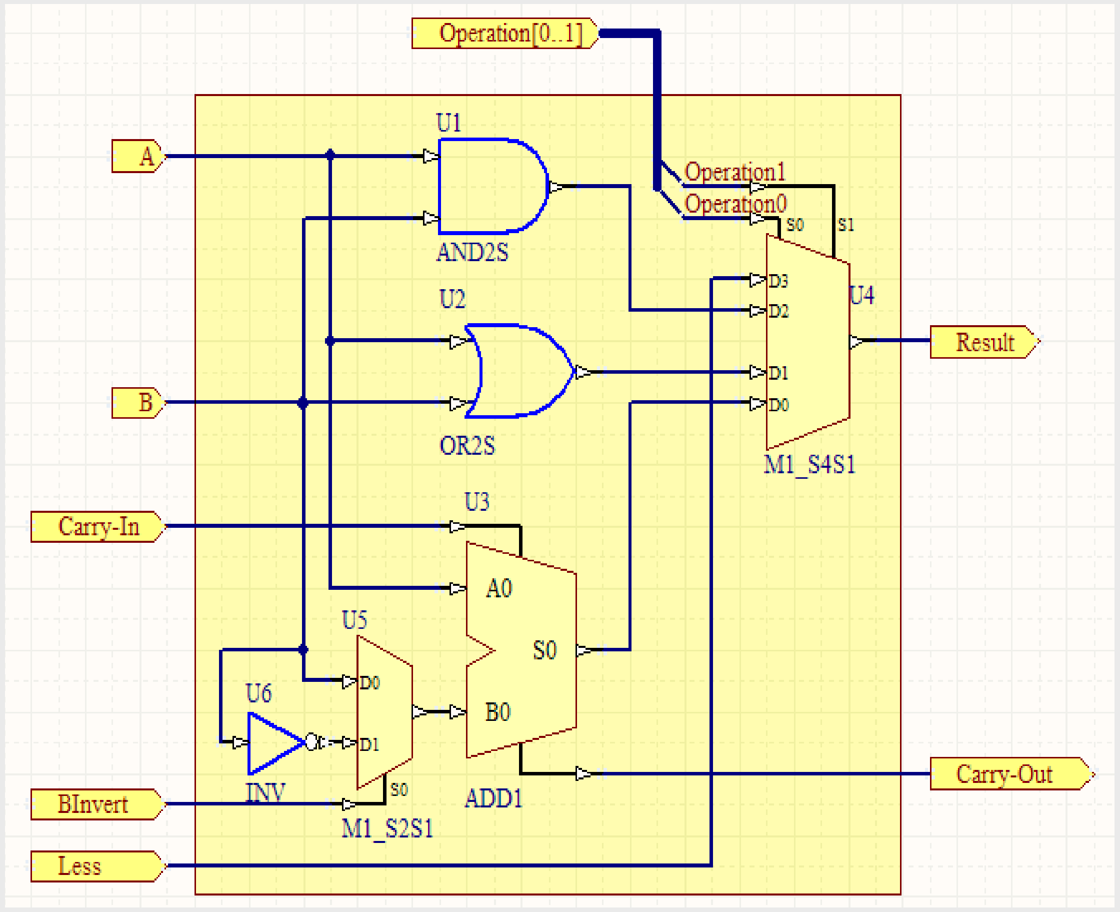
ALU Control Lines			Result
Binvert	**Carry In**	**Operation**	
0	**0**	**2 = $(10)_{two}$**	**AND**
0	**0**	**1 = $(01)_{two}$**	**OR**
0	**0**	**0 = $(00)_{two}$**	**add**
1	**1**	**0 = $(00)_{two}$**	**sub**

1-bit ALU with AND, OR, Add, Sub, and SLT (1)

- Since we need to perform one more operation, we increase the number of inputs at the multiplexer by 1 and label the new input as Less
- SLT operation:

  if (a < b), set Less to 1

  => if (a – b) < 0, set Less to 1
- SLT operation can therefore be expressed in terms of a subtraction between the two operands.
- If the result of subtraction is negative, set Less to 1.
- How do we determine if the result is negative?



1-bit ALU with AND, OR, Add, Sub, and SLT capability

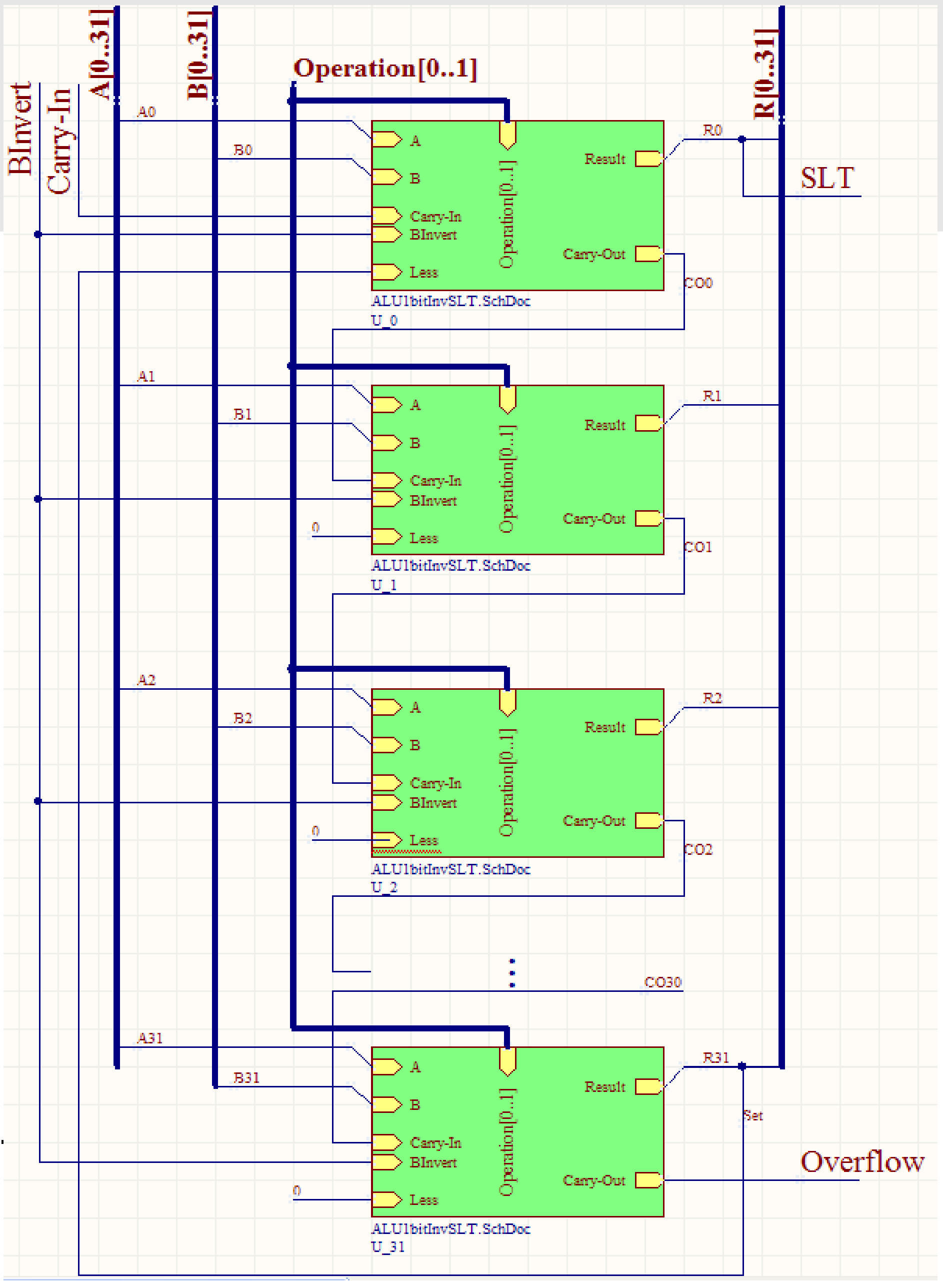
32-bit ALU

w/ And, OR, Add, Subtract, and SLT

- The 1-bit ALU's can be cascaded together to form a 32 bit ALU
- Operations are controlled by the Operation bus

ALU Control Lines			Result
Binvert	**Carry In**	**Operation**	
0	0	0 = $(00)_{two}$	**Add** sum(a,b)
0	0	1 = $(01)_{two}$	**OR** (a+b)
0	0	2 = $(10)_{two}$	**AND** (a·b)
1	1	0 = $(00)_{two}$	**Subtract** (a – b)
1	1	3 = $(11)_{two}$	**SLT** Set Result0 if (a < b)

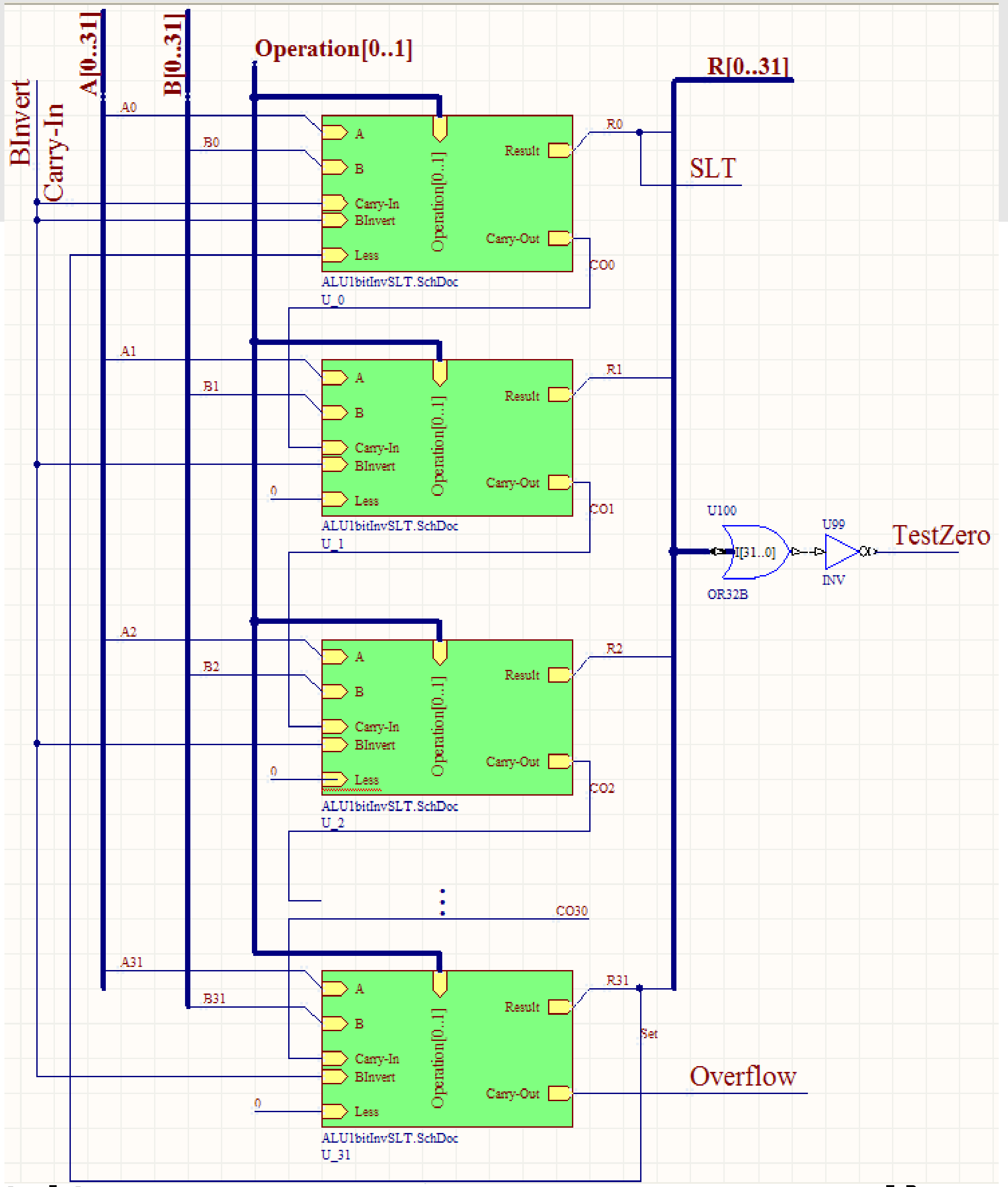
- Note that Binvert is always the same as Carry-In
- To test equality between *a* and *b*, subtract *b* from a and check if the result is 0.



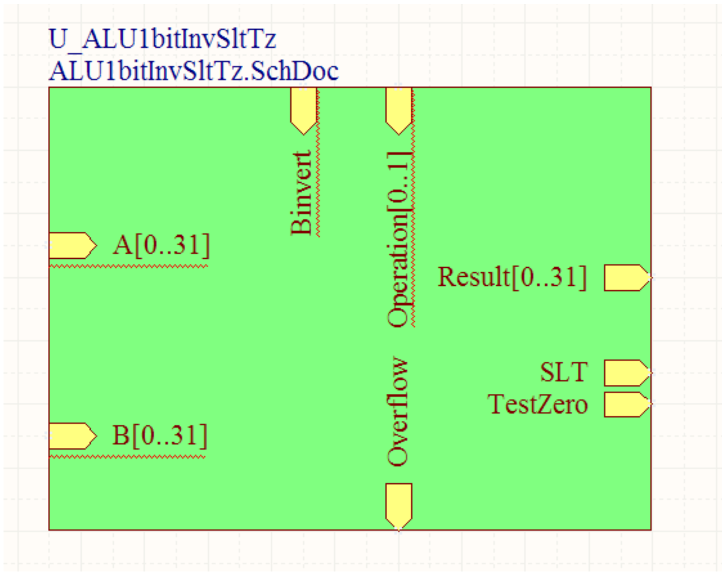
W5-W

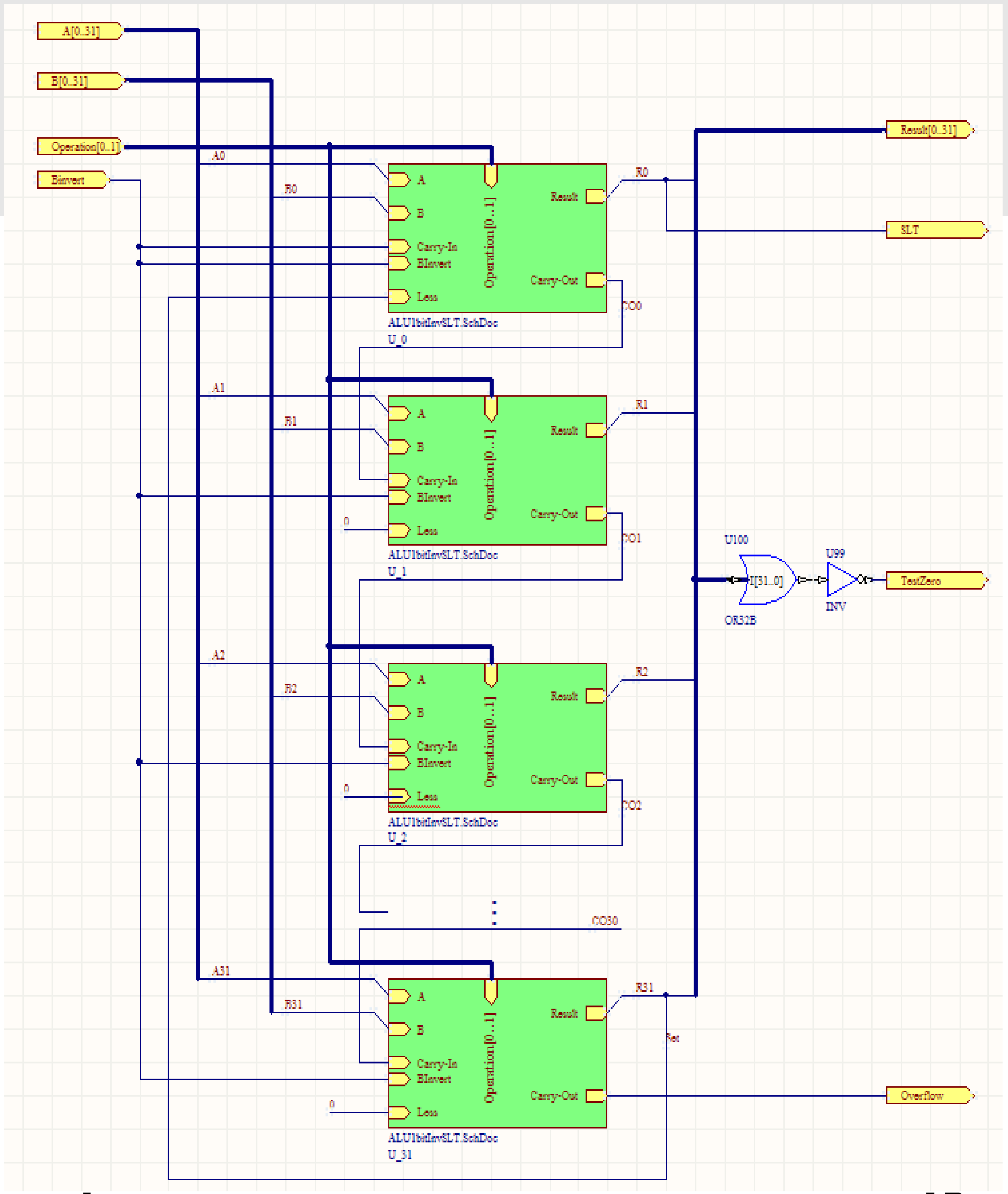
32-bit ALU w/ And, OR, Add, Subtract, SLT, and Equality Test

ALU Control Lines			Result
Binvert	Carry In	Operation	
0	0	$0 = (00)_{two}$	Add sum(a,b)
0	0	$1 = (01)_{two}$	OR (a+b)
0	0	$2 = (10)_{two}$	AND $(a \cdot b)$
1	1	$0 = (00)_{two}$	Subtract (a - b)
1	1	$3 = (11)_{two}$	SLT if $(a < b)$ Result0 = 1
1	1	$0 = (00)_{two}$	Test Equality Zero = 1 if $(a = b)$



32-bit ALU w/ And, OR, Add, Subtract, SLT, and Equality Test


U_ALU1bitInvSltTz
ALU1bitInvSltTz.SchDoc
A[0..31]
B[0..31]
Binvert
Operation[0..1]
Overflow
Result[0..31]
SLT
TestZero


A[0..31]
B[0..31]
Operation[0..1]
Binvert
A0
B0
A
B
Carry-In
Binvert
Less
Operation[0..1]
Result
Carry-Out
R0
CO0
ALU1bitInvSLT.SchDoc
U_0
A1
B1
0
R1
CO1
U_1
A2
B2
R2
CO2
U_2
CO30
A31
B31
R31
Set
U_31
Result[0..31]
SLT
U100
I[31..0]
OR32B
U99
INV
TestZero
Overflow