CSE 2021
Computer Organization

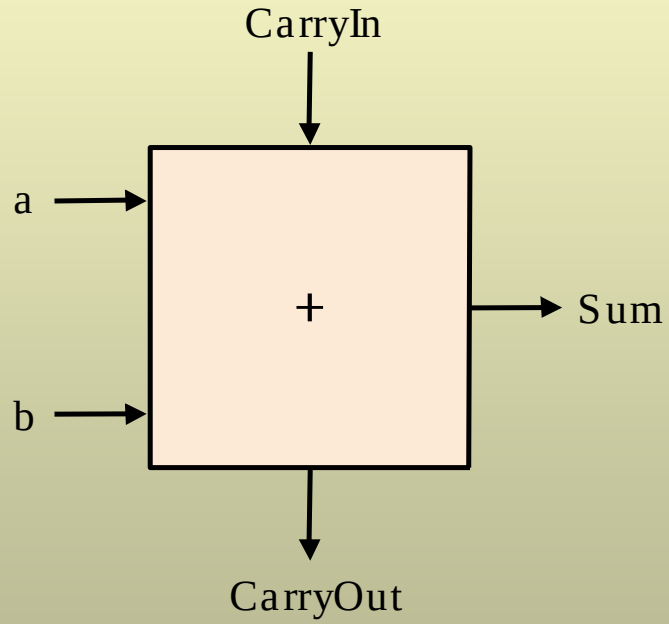
Hugh Chesser, CSEB
1012U

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



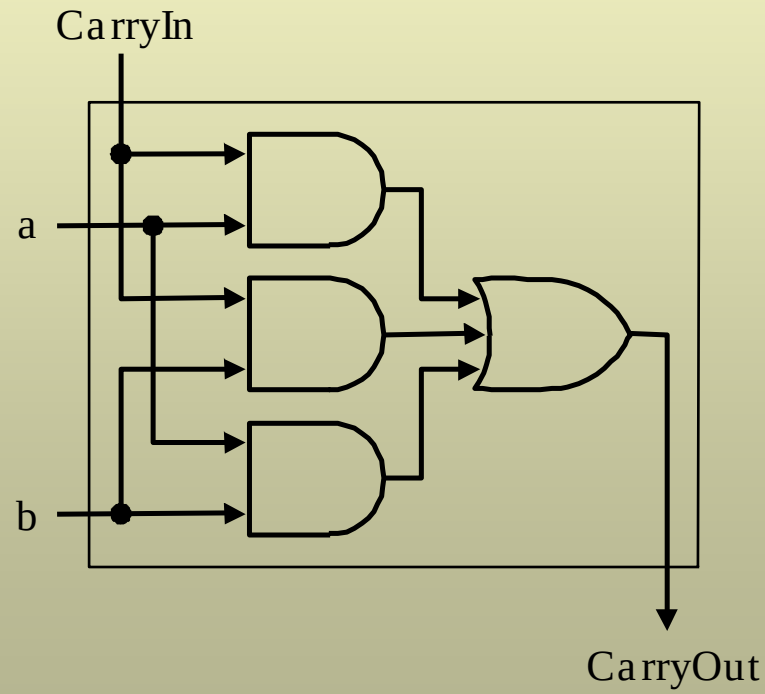
$$\text{Sum} = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc$$
$$\text{Carryout} = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc$$

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

Step 3: Simplify the Boolean expression

$$\text{Carryout} = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc = bc + ac + ab$$

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

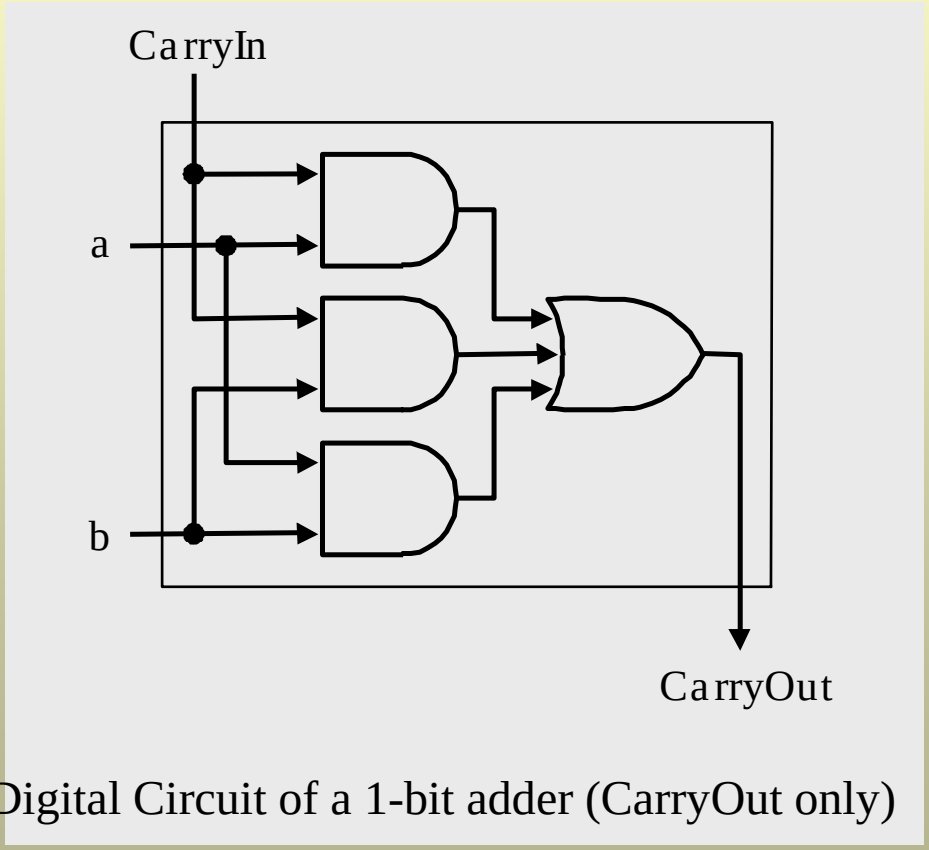
Agenda for Today

- Introduction to Hardware – Logic Design

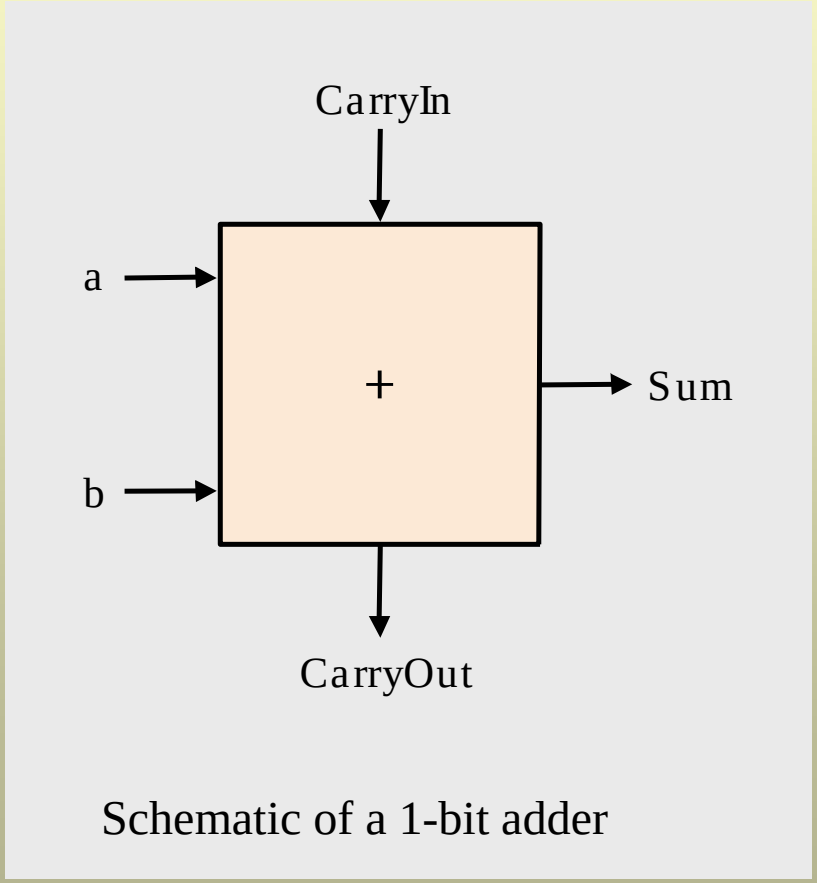
Patterson: Appendix C

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



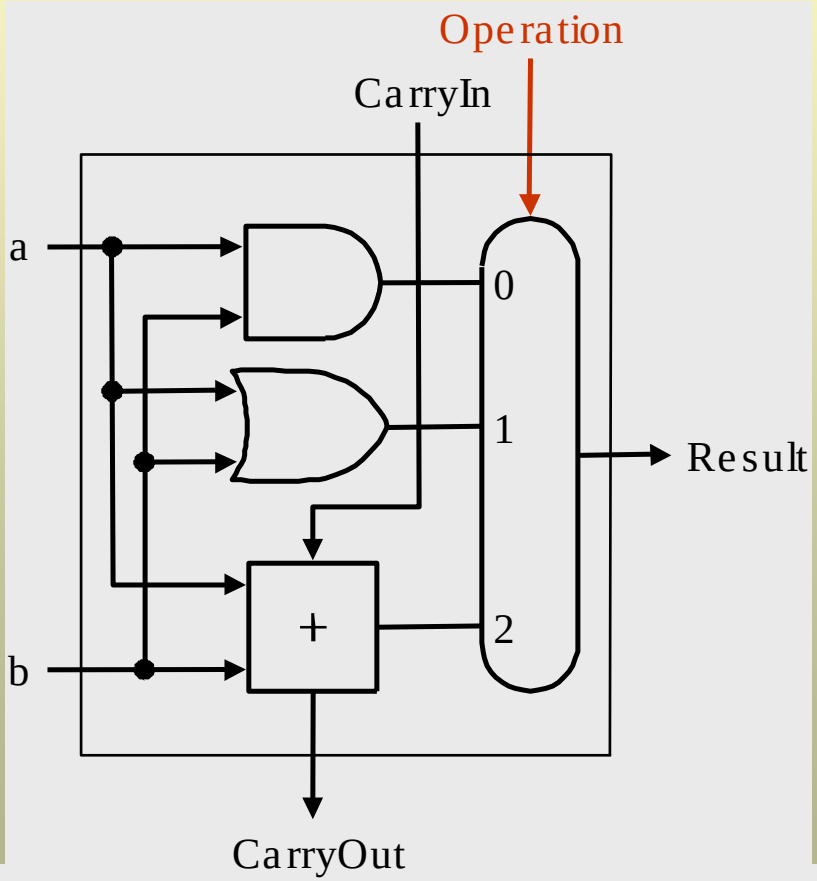
Digital Circuit of a 1-bit adder (CarryOut only)



Schematic of a 1-bit adder

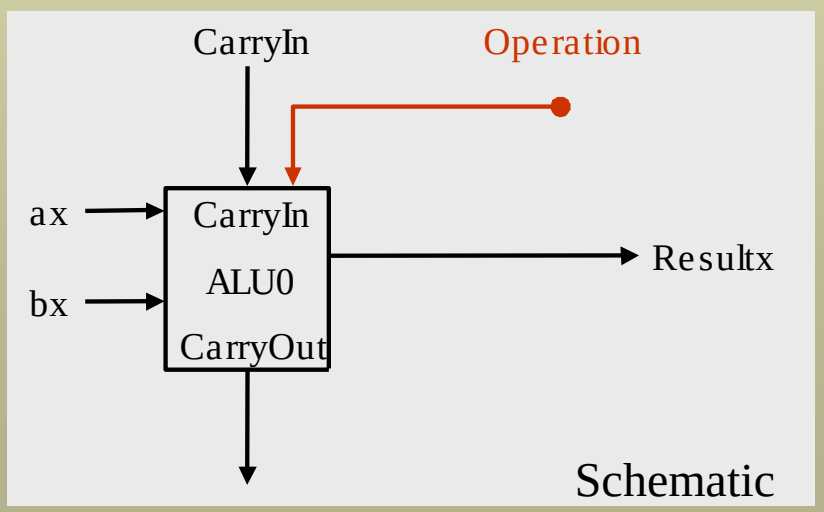
1-bit ALU with AND, OR, and Addition

- The 1-bit adder is supplemented with AND and OR gates
- A multiplexer controls which gate is connected to the output



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

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



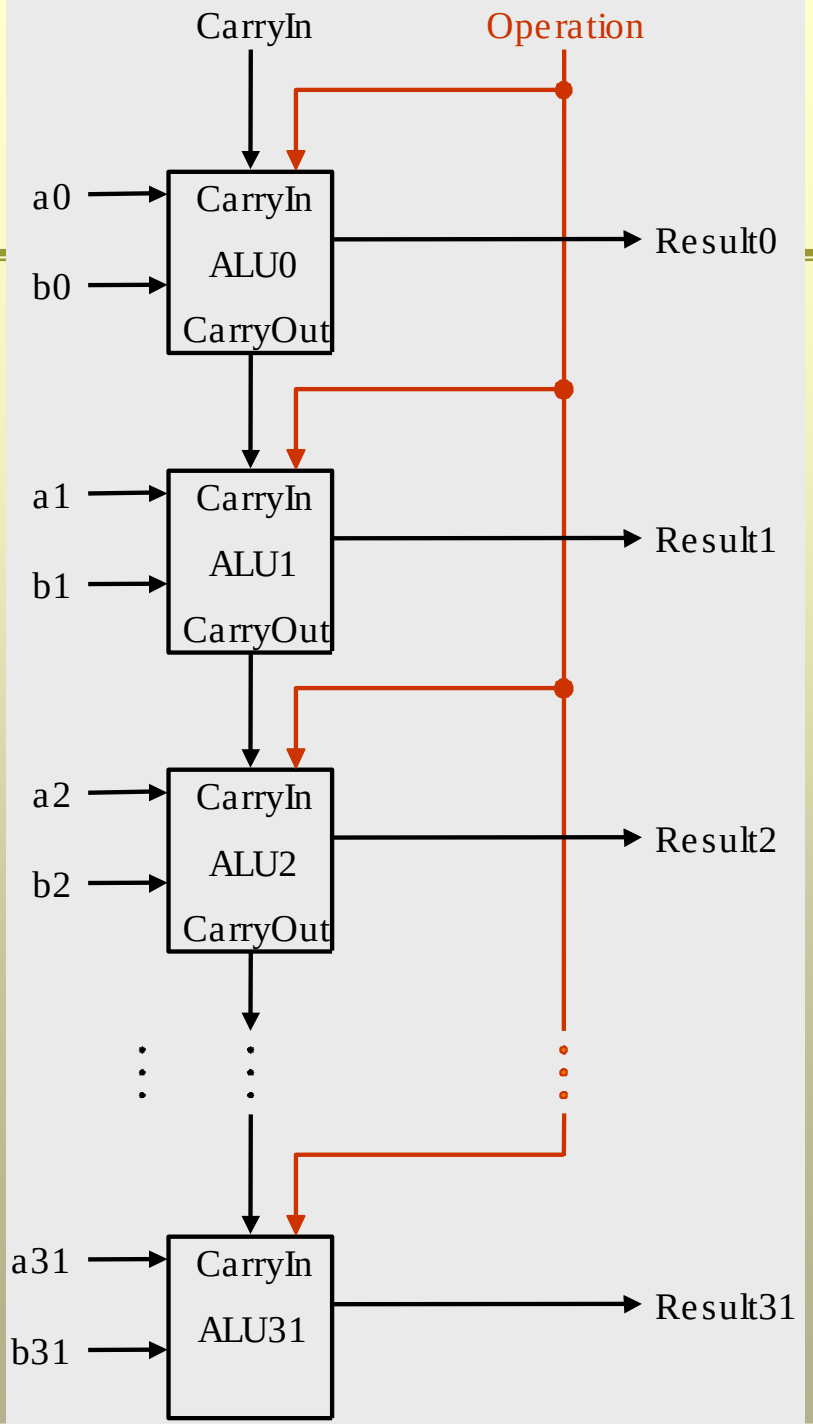
Schematic

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

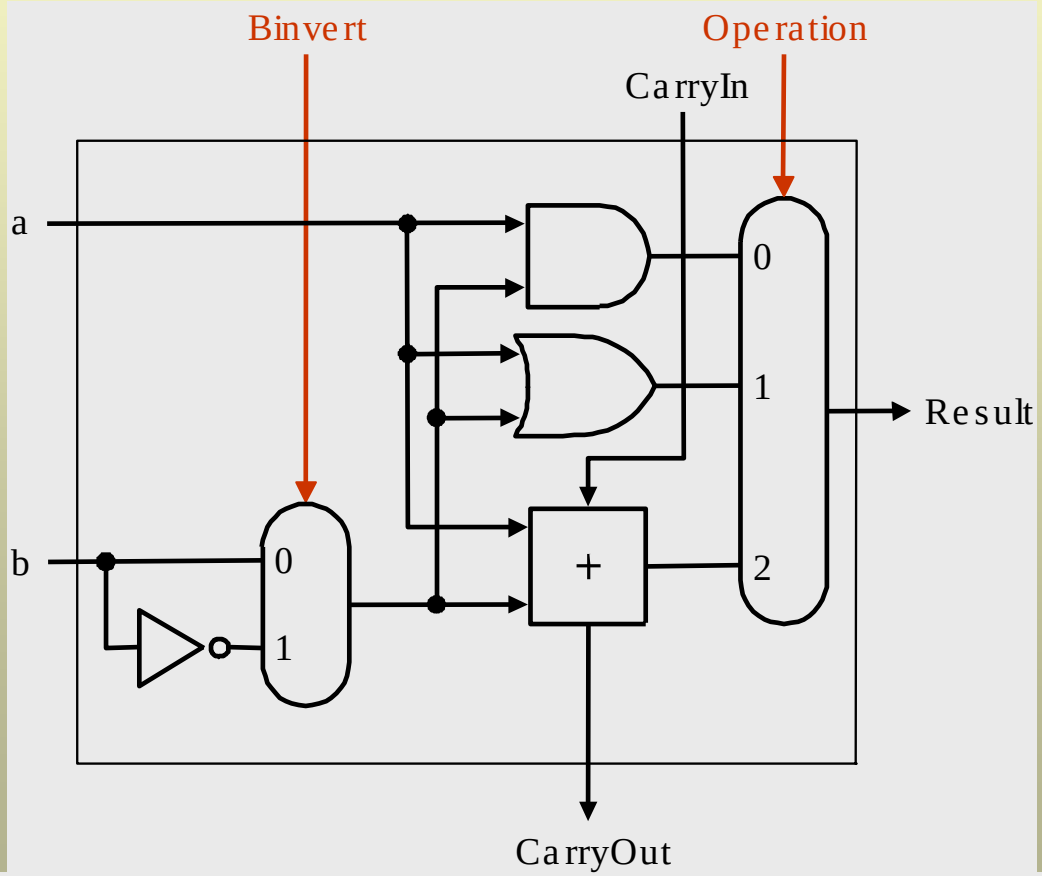
ALU Control Lines		Result
Carry In	**Operation**	
0	**0 = $(00)_{two}$**	**AND**
0	**1 = $(01)_{two}$**	**OR**
0	**2 = $(10)_{two}$**	**add**

- 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



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

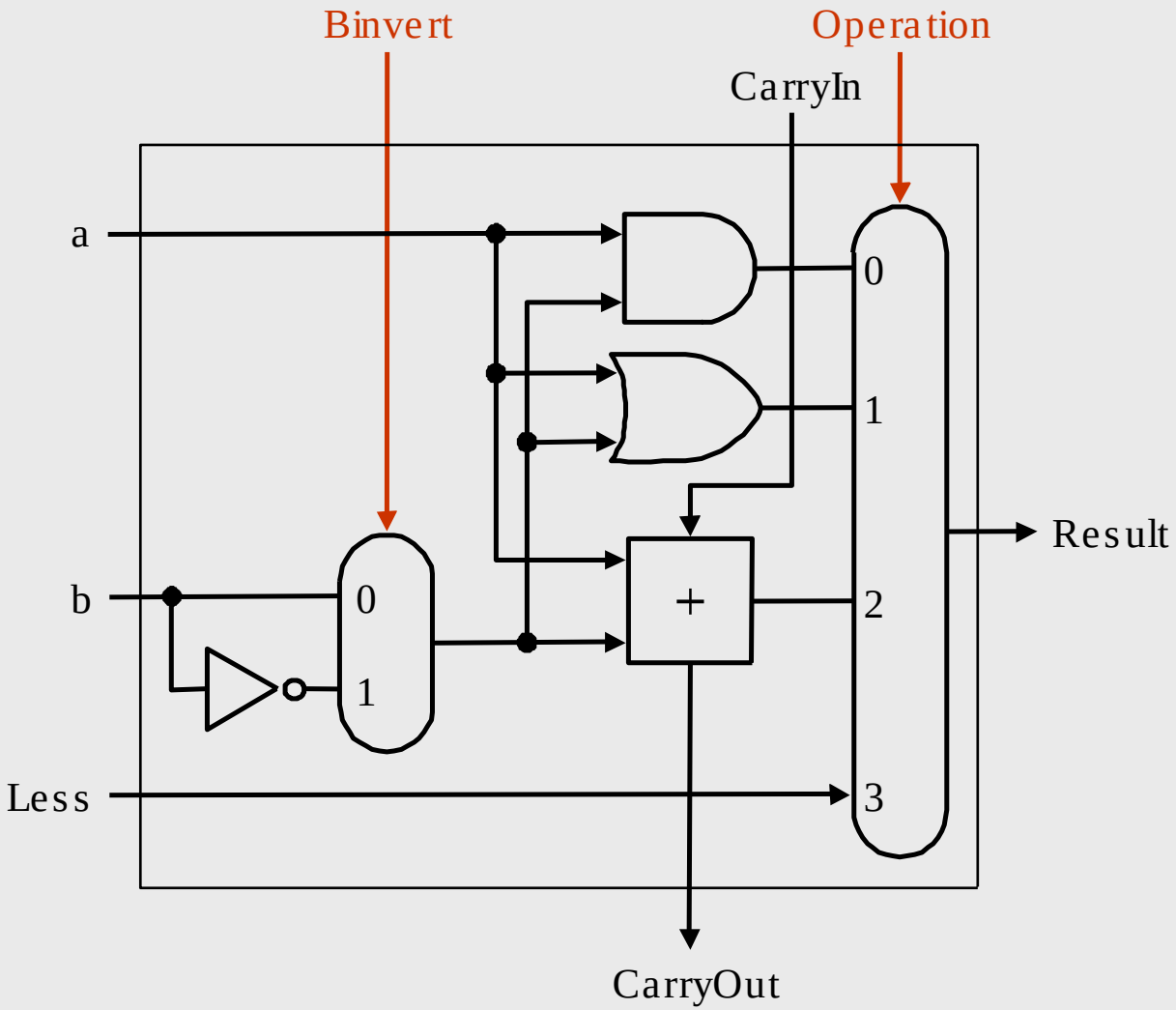
1-bit ALU with AND, OR, Addition, and Subtraction capability

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

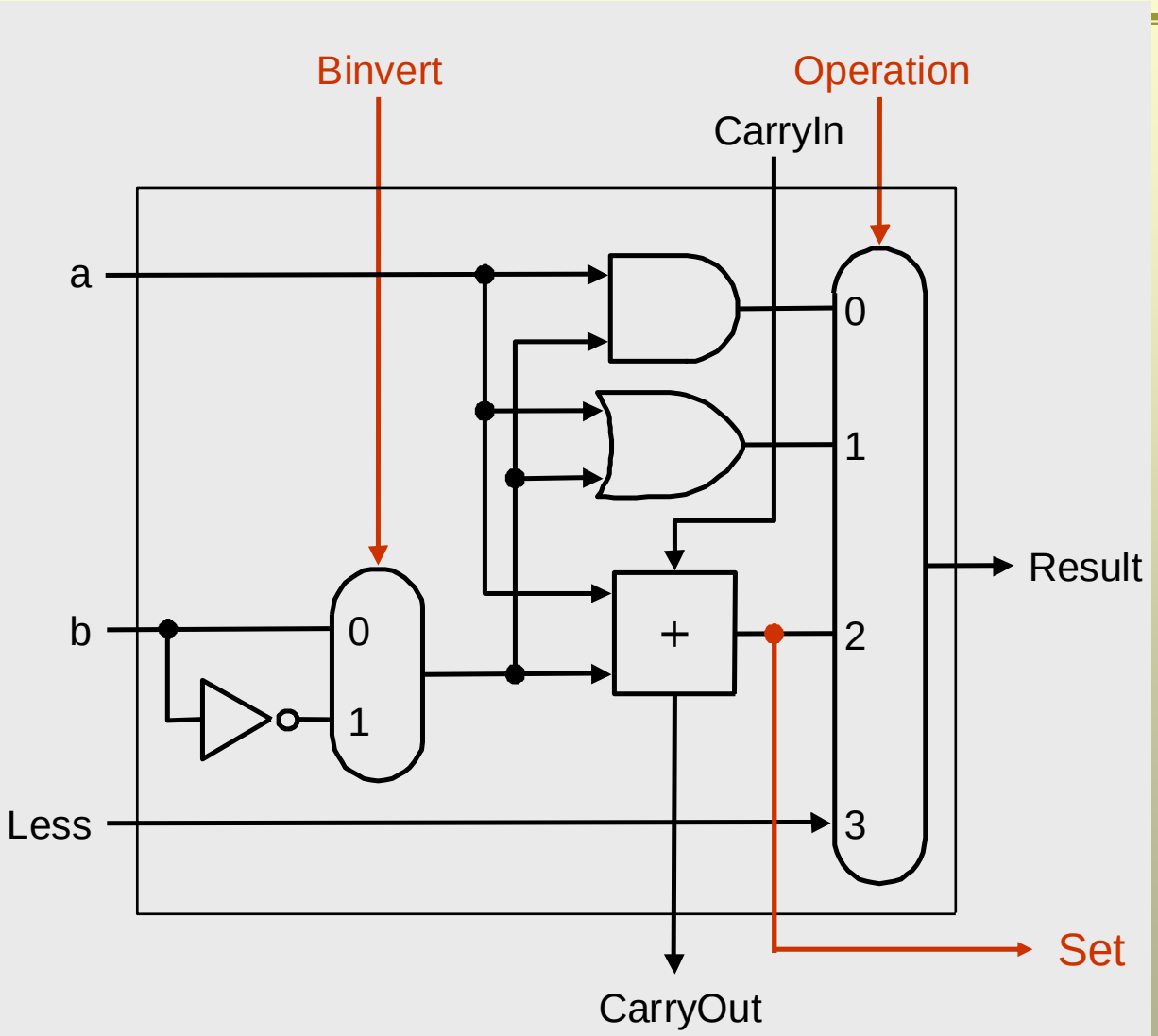
  => if (a – b) < 0, set Less to 1
- SLT operation can therefore be expressed in terms of a subtraction between the two operands.
- If the result of subtraction is negative, set Less to 1.
- How do we determine if the result is negative?



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

1-bit ALU with AND, OR, Add, Sub, and SLT (2)

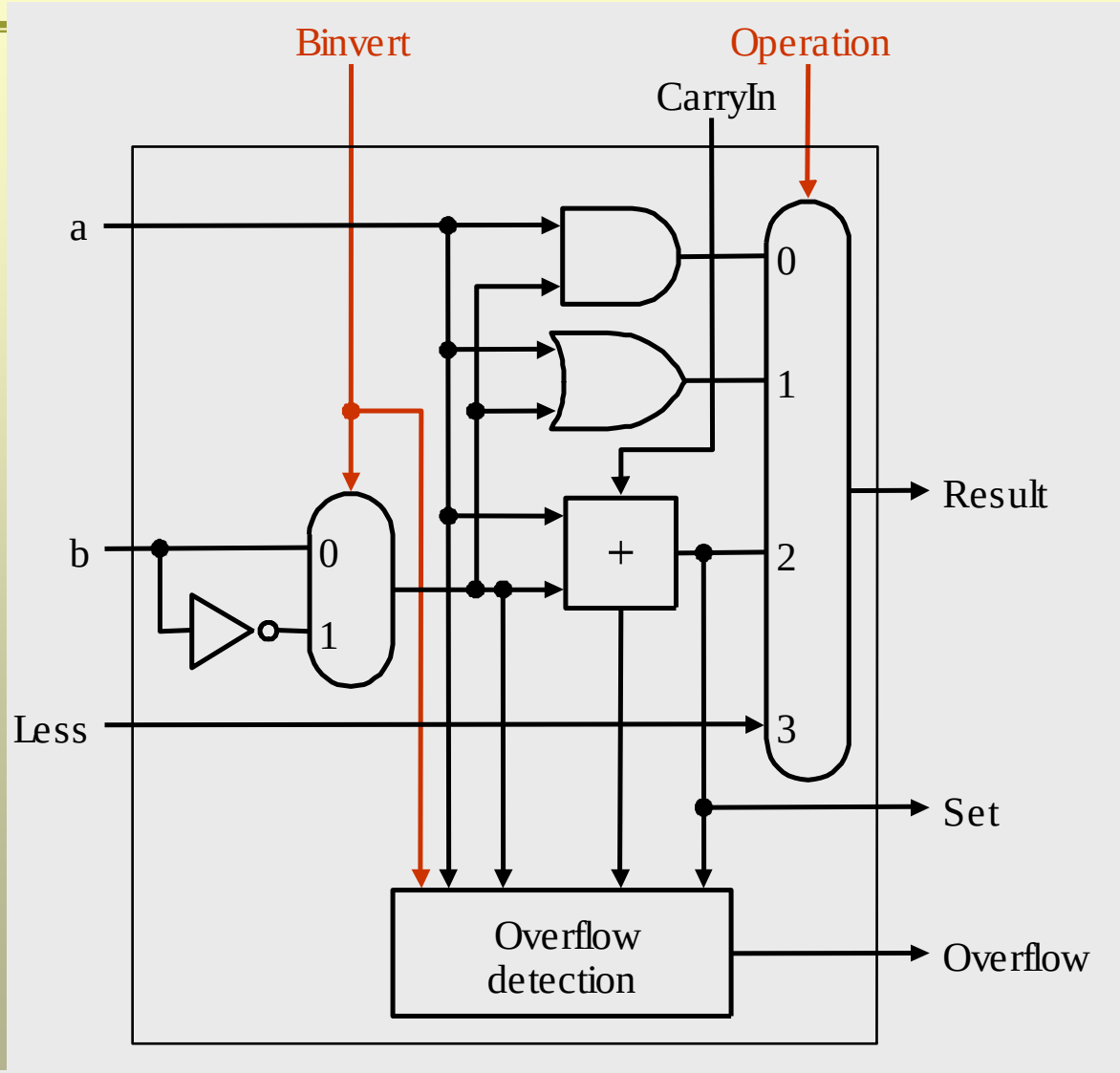
— Use the sign bit obtained from the 1-bit ALU at the MSB position to indicate the result of SLT.



1-bit ALU of MSB (bit 31) with AND, OR, Add, Sub, and SLT capability

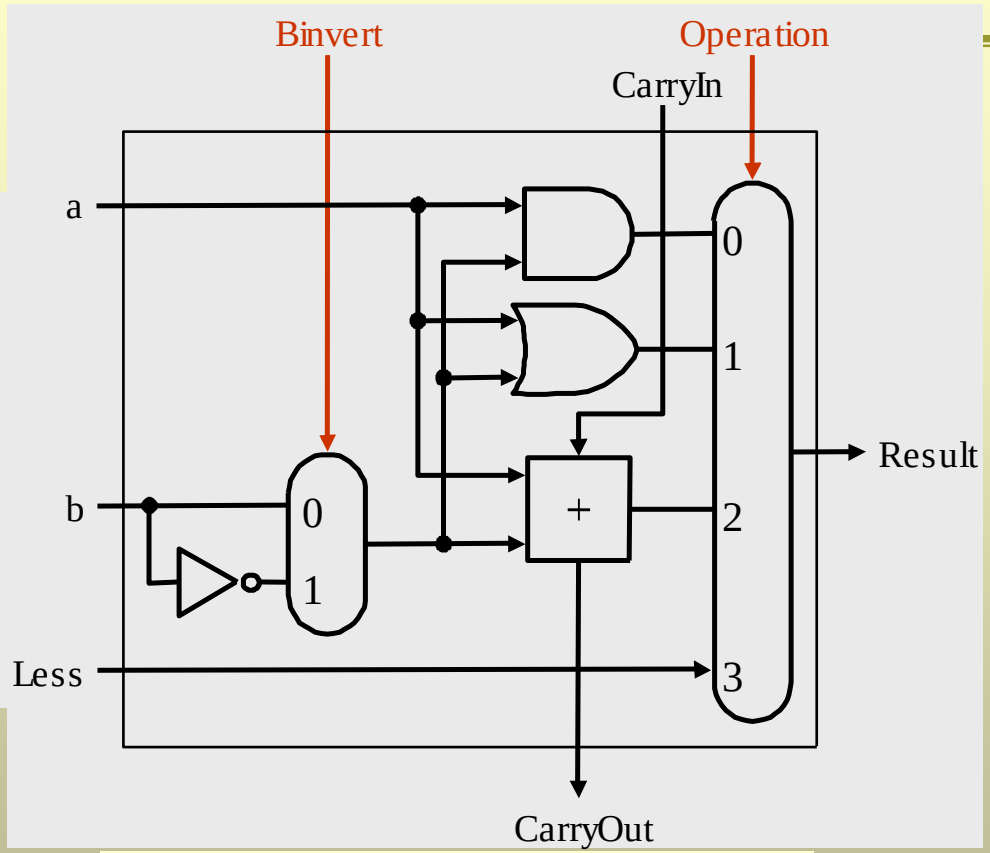
1-bit ALU with AND, OR, Add, Sub, and SLT (3)

— In fact we also include the overflow circuit within the 1-bit ALU at the MSB

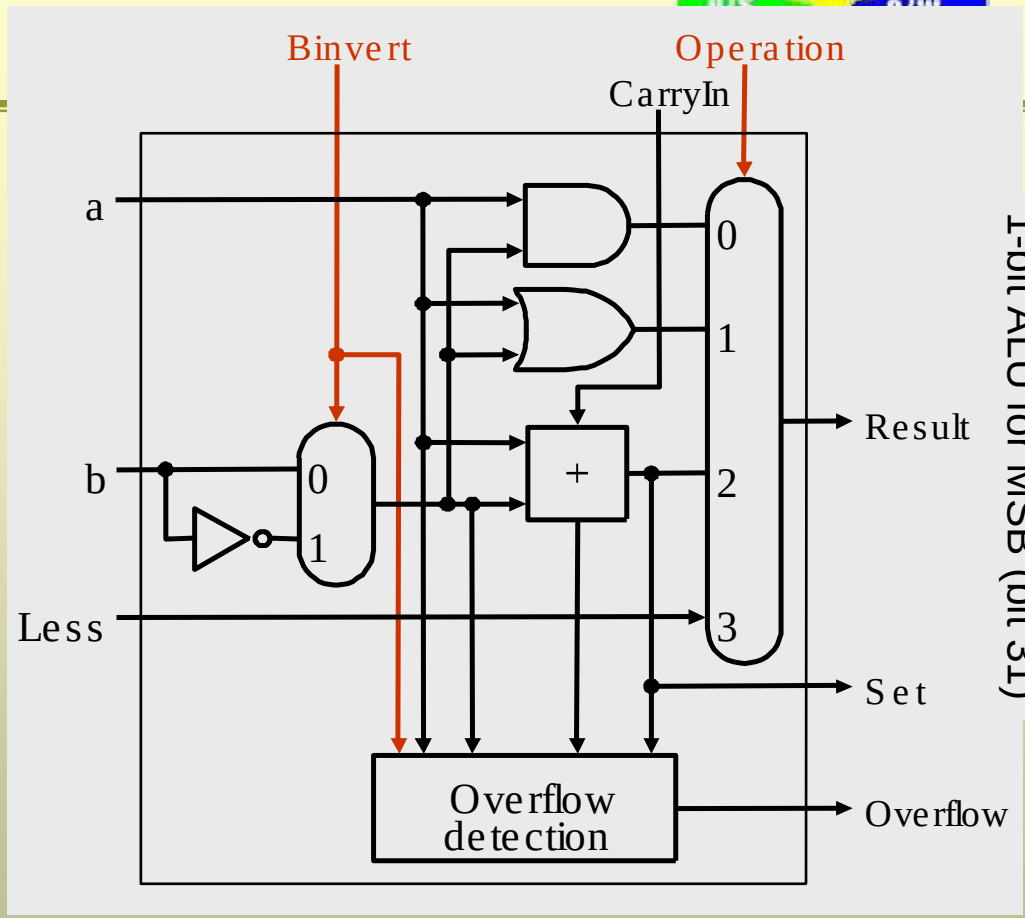


1-bit ALU of MSB (bit 31) with AND, OR, Add, Sub, SLT, and overflow

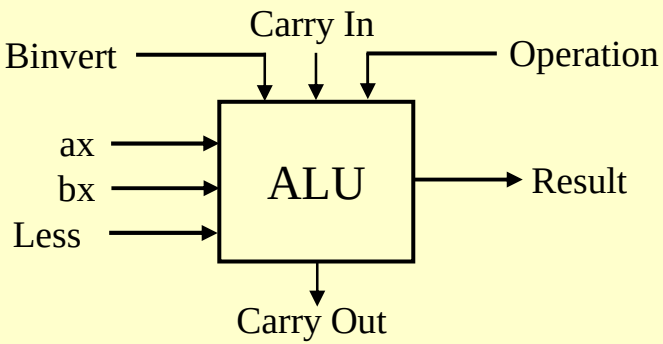
1-bit ALU (4)

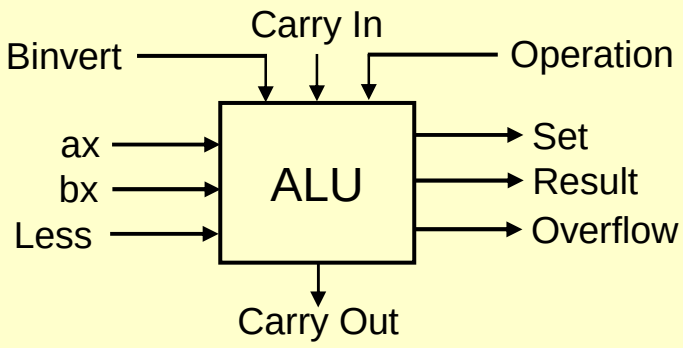


1-bit ALU for bits (0 – 30)



1-bit ALU for MSB (bit 31)





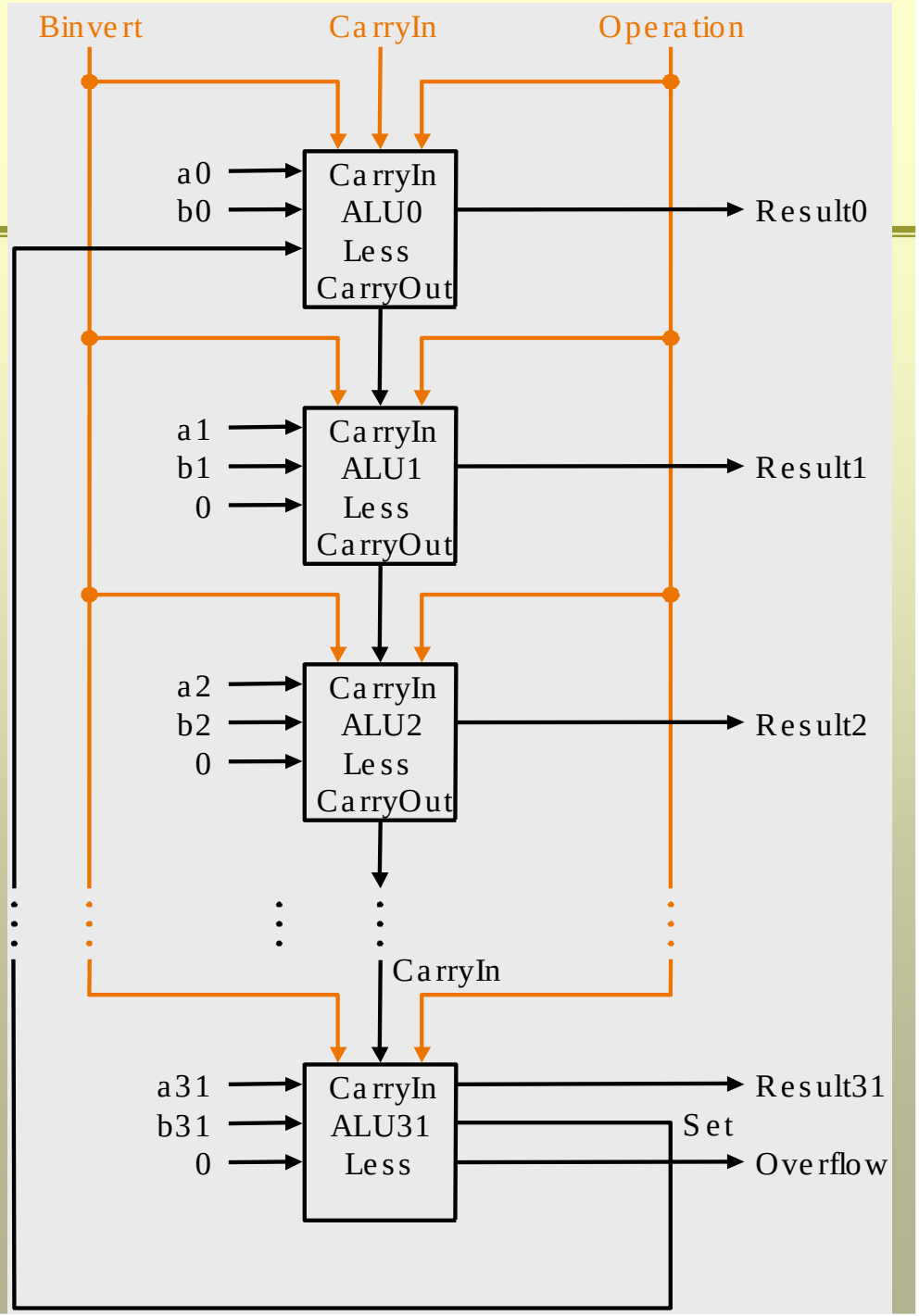
32-bit ALU

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

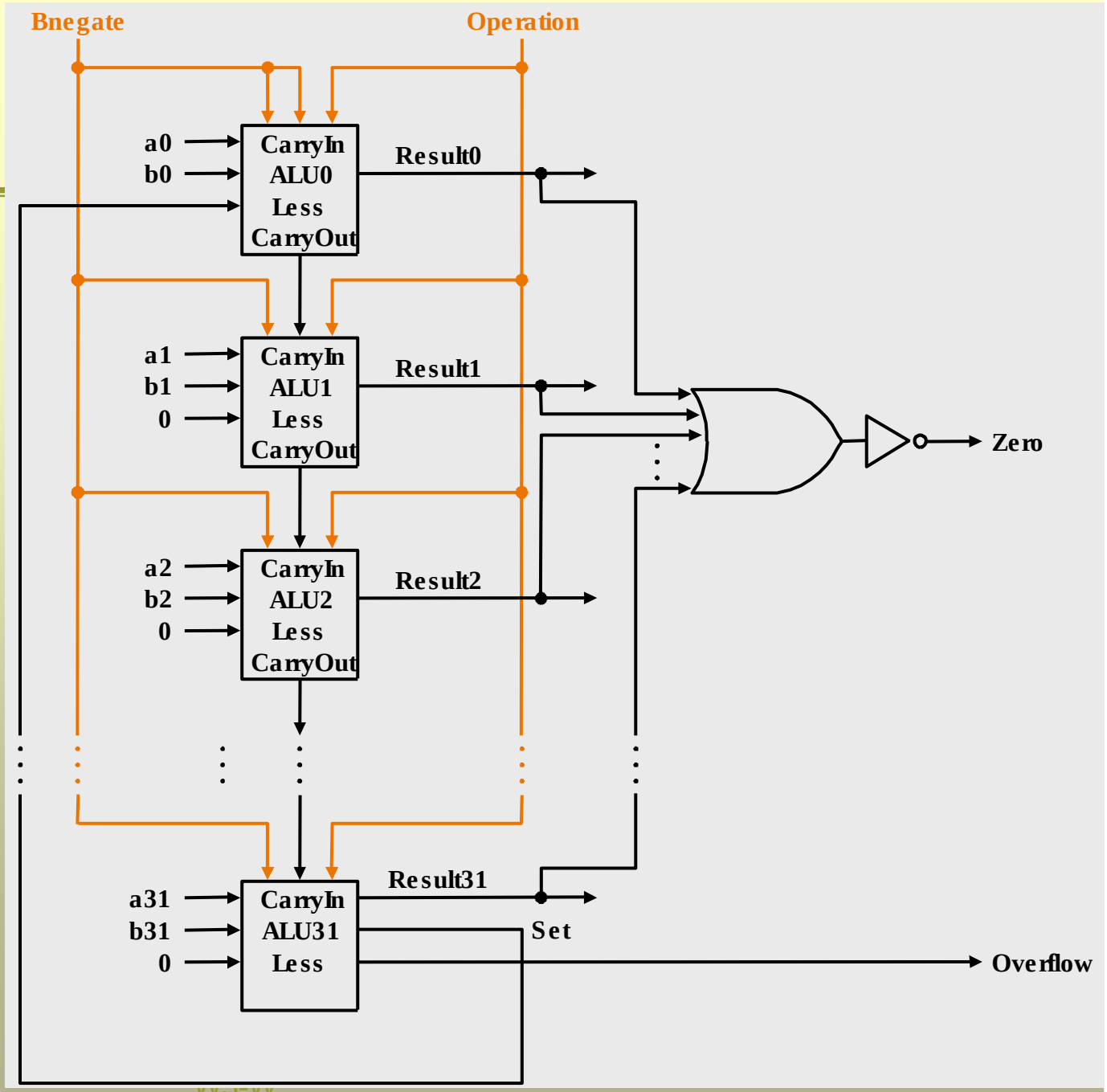
ALU Control Lines			Result
Binvert	**Carry In**	**Operation**	
0	**0**	**0 = $(00)_{two}$**	**AND (a·b)**
0	**0**	**1 = $(01)_{two}$**	**OR (a+b)**
0	**0**	**2 = $(10)_{two}$**	**Add sum(a,b)**
1	**1**	**2 = $(11)_{two}$**	**Subtract (a – b)**
1	**1**	**3 = $(11)_{two}$**	**SLT Set Result0 if (a < b)**

- 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.



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

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



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