Simplifying expressions

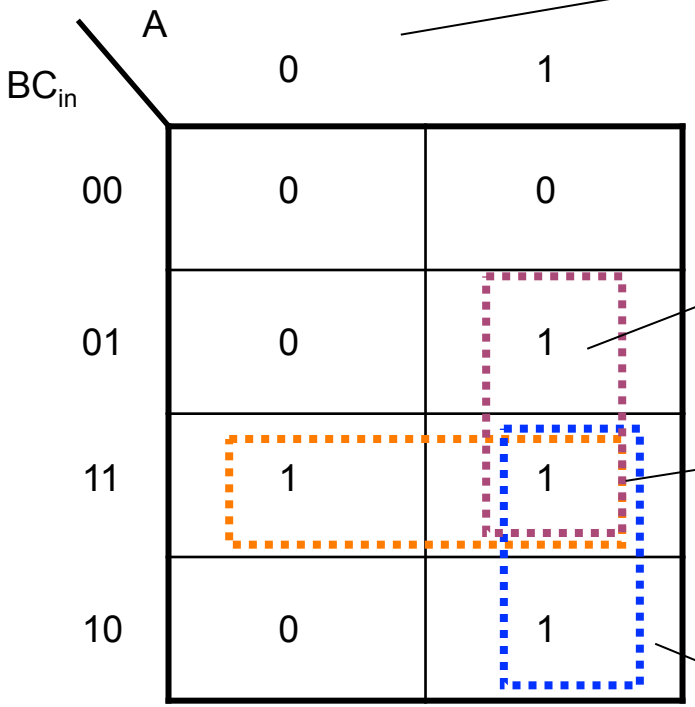
Input			Output	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- $C_{out} = A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in}$
- $C_{out} = BC_{in} + AC_{in} + AB$

- Simplification reduces complexity: faster, smaller circuit!

Karnaugh map

$C_{out} = A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in}$



AC_{in}

BC_{in}

AB

$C_{out} = BC_{in} + AB + AC_{in}$

A "tool" to help simplify boolean expressions
Like a "slide rule": Useful but limited

A table listing "minterms"
Minterms written in Gray code order
One var value changes betw. col/row

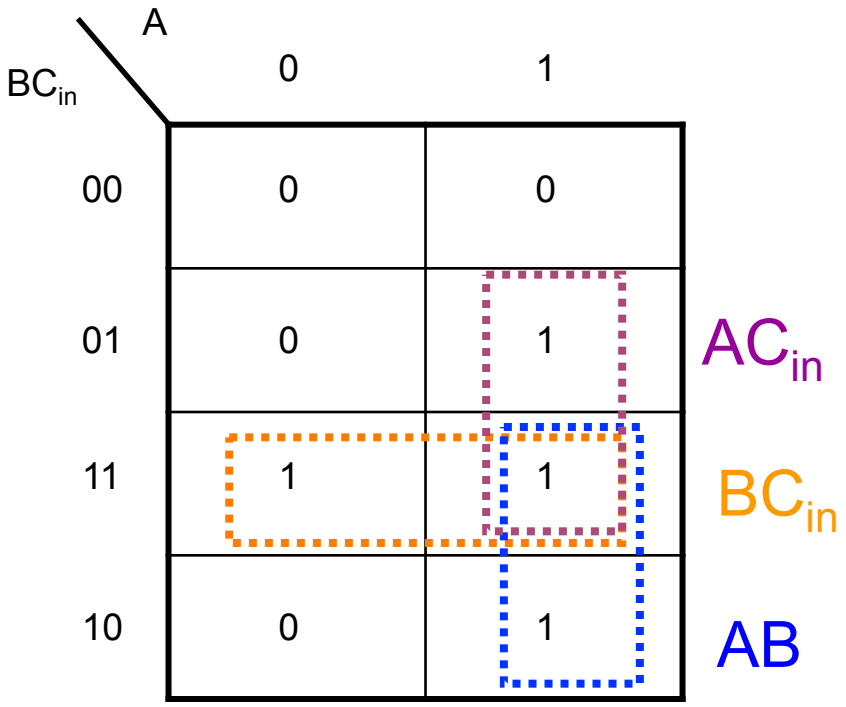
Build from the initial boolean expr.
Put a "1" where a minterm is true

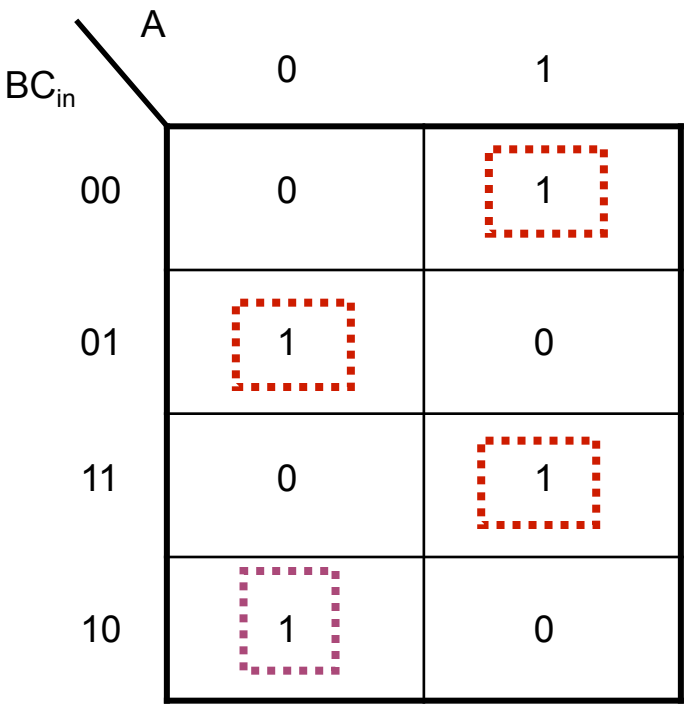
E.g.., $AB'C_{in}$ has a 1

Now, to simplify:
Look for adjacent max rectangular groups with power of 2 elements, with at least one unique variable.
In such a group, some var is {0,1}
Eliminate that variable

Here's another one!
Groups can be vertical too.
They can even "wrap around"
Diagonals aren't allowed

- $C_{out} = A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in}$
- $S = A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in}$





$C_{out} = BC_{in} + AB + AC_{in}$

$S = \mathbf{A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in}}$

Four (or more?) Variables

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	1	0
10	0	1	1	0

Can you minimize this one?

In AB: B is both {0,1}
In CD: C is both {0,1}

Eliminate B, C
Thus, we have just AD

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

Can you minimize this one?

C,D both have {0,1}
A has {0,1}

Eliminate A,C,D
Thus, we have just B

Four (or more?) Variables

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

Can you minimize this one?

Combine on top row
Combine on bottom row

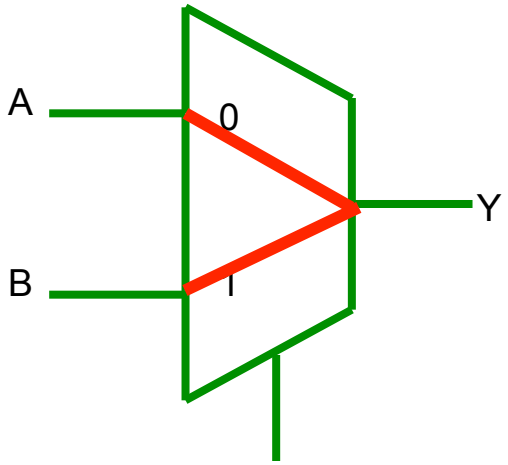
A'B'D'
AB'D'

These terms can now combine
Thus, we have B'D'

Karnaugh Maps (K-Maps) are a simple calculation tool.

In practice, sophisticated logic synthesis algorithms/tools are used.

Multiplexor (aka MUX)

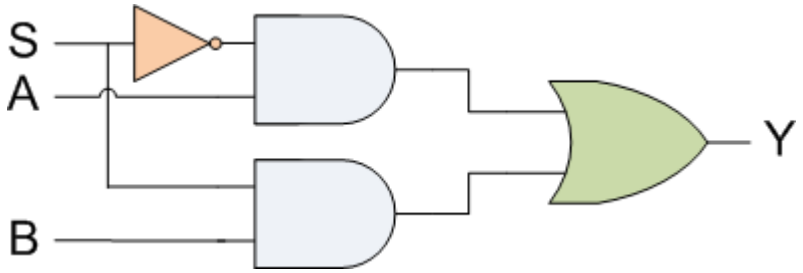


Y = (S) ? B:A;

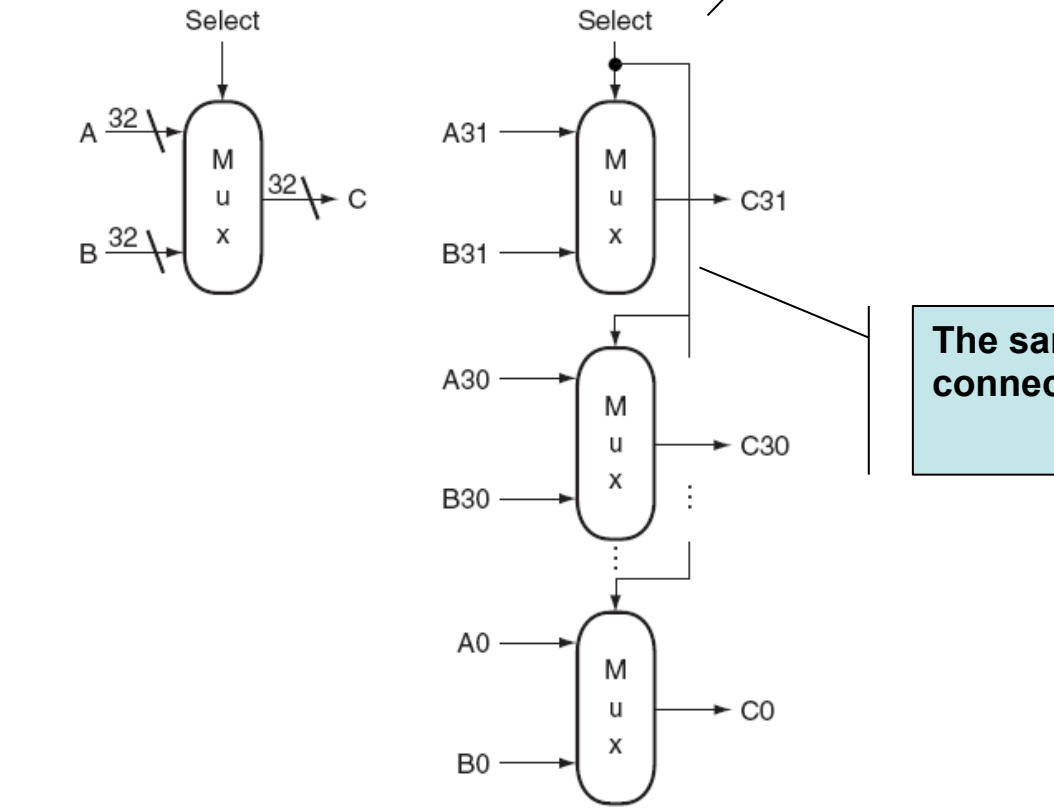
when S =
0: output A
1: output B

S	A	B	Y
0	0	x	0
0	1	x	1
1	x	0	0
1	x	1	1

Y=S'A+SB



A 32-bit MUX



a. A 32-bit wide 2-to-1 multiplexor

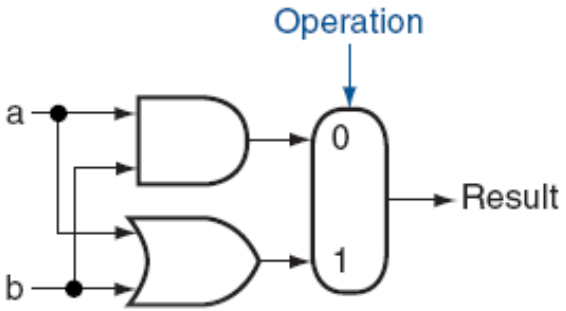
b. The 32-bit wide multiplexor is actually an array of 32 1-bit multiplexors

Use 32 1-bit muxes
Each mux selects 1 bit

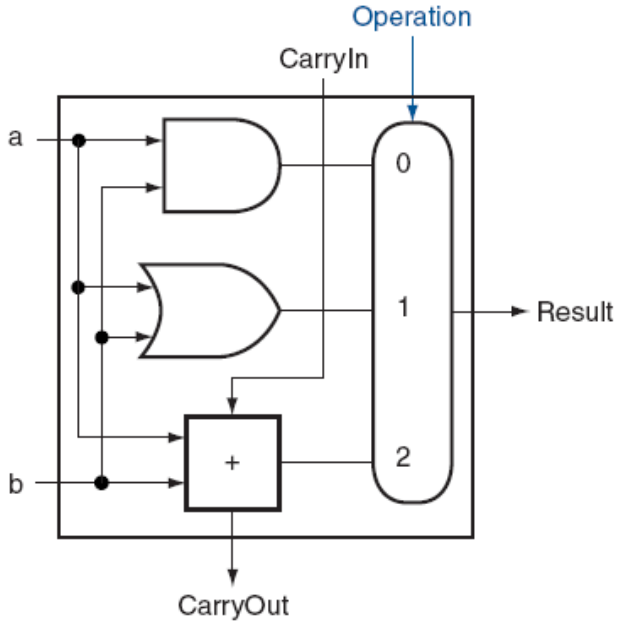
The same 1-bit Select (S) is connected to each mux

Building a 1-bit ALU

- ALU = arithmetic logic unit = arithmetic unit + logic unit



1 Let's start small: Choose (select) betw. AND and OR

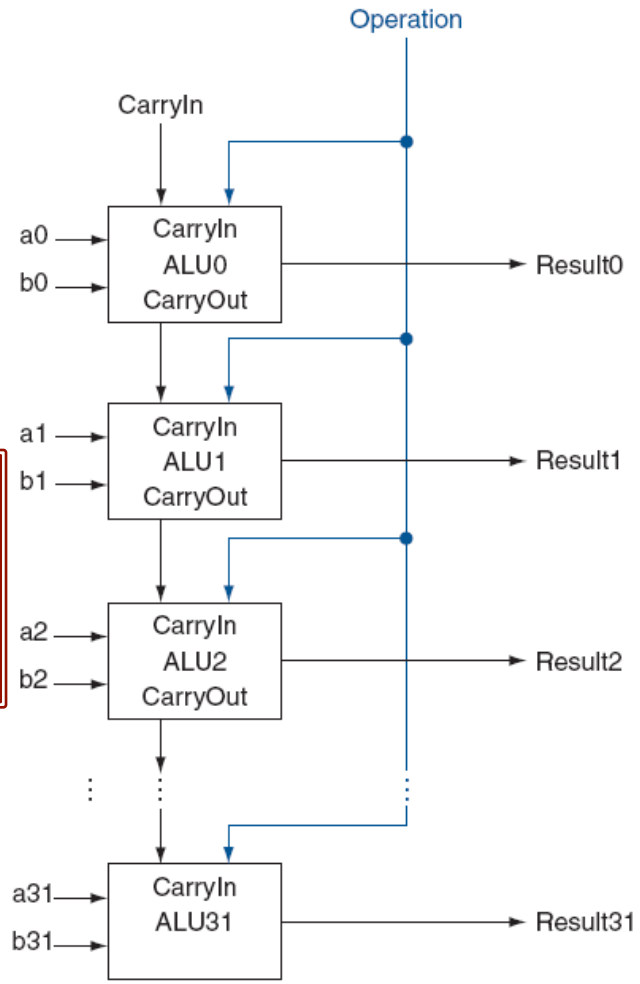


2 Now, introduce addition to ALU

Building a 32-bit ALU

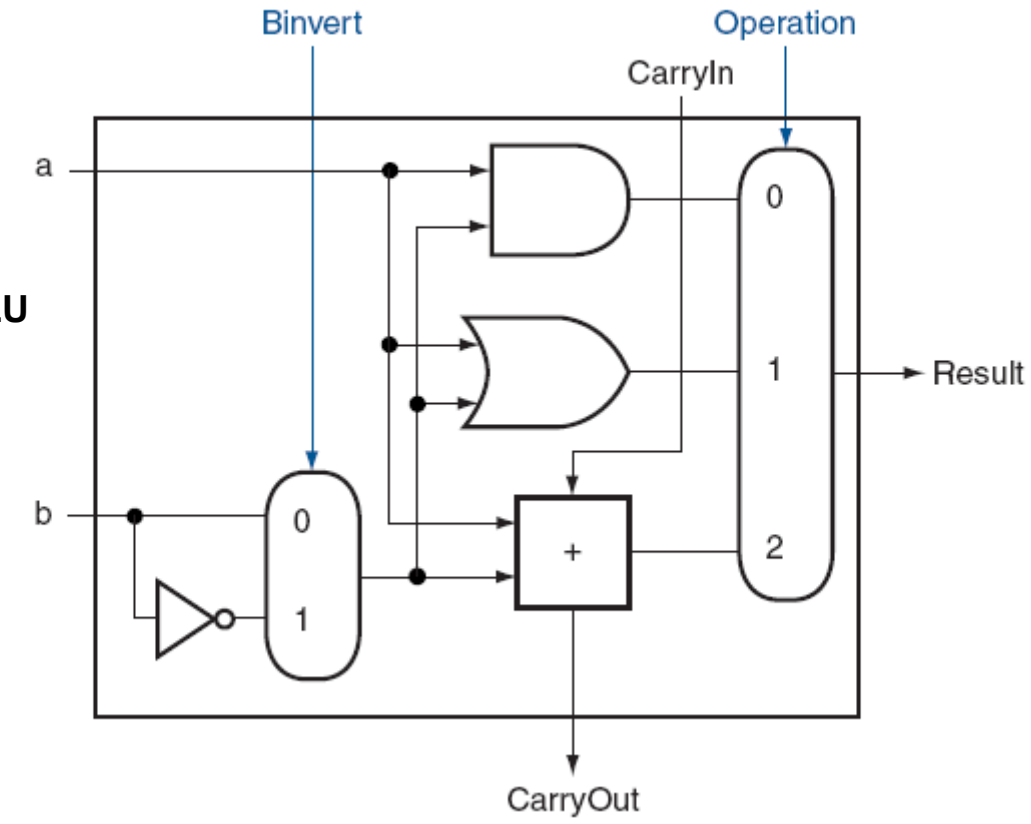
Can you build a 2-bit ALU that does +, AND, OR?

Let's try it!



Implementing “sub”

Binvert=1
CarryIn=1 for 1st 1-bit ALU
Operation=2

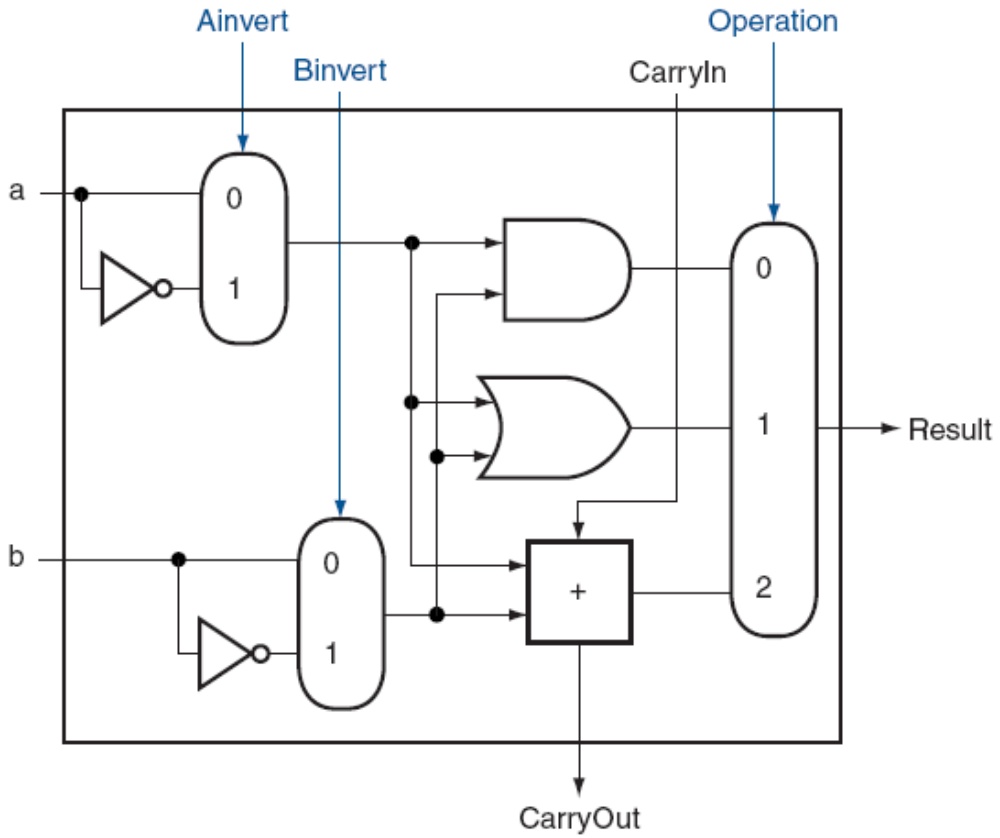


Implementing NAND and NOR

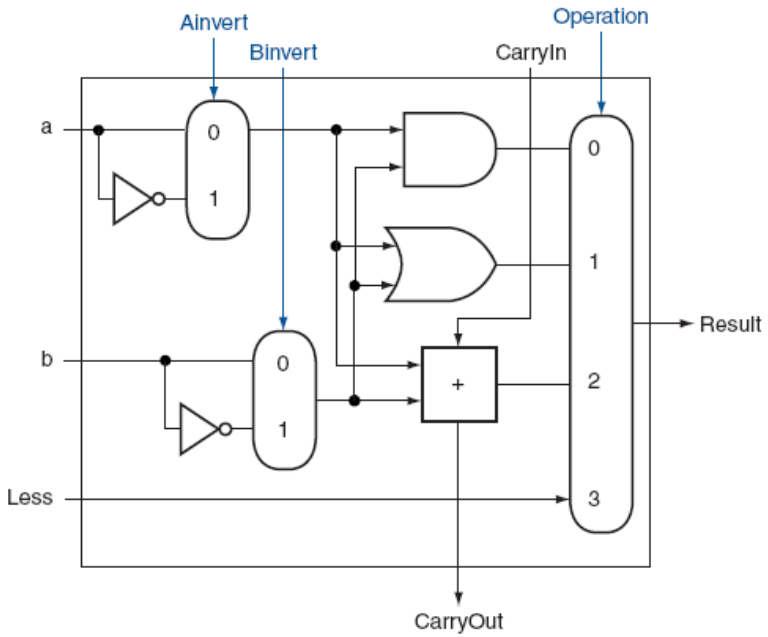
NOR:
NOT (A OR B)
by DeMorgan's Law:
(NOT A) AND (NOT B)

Thus,
Operation=0,
Ainvert=1,
Binvert=1

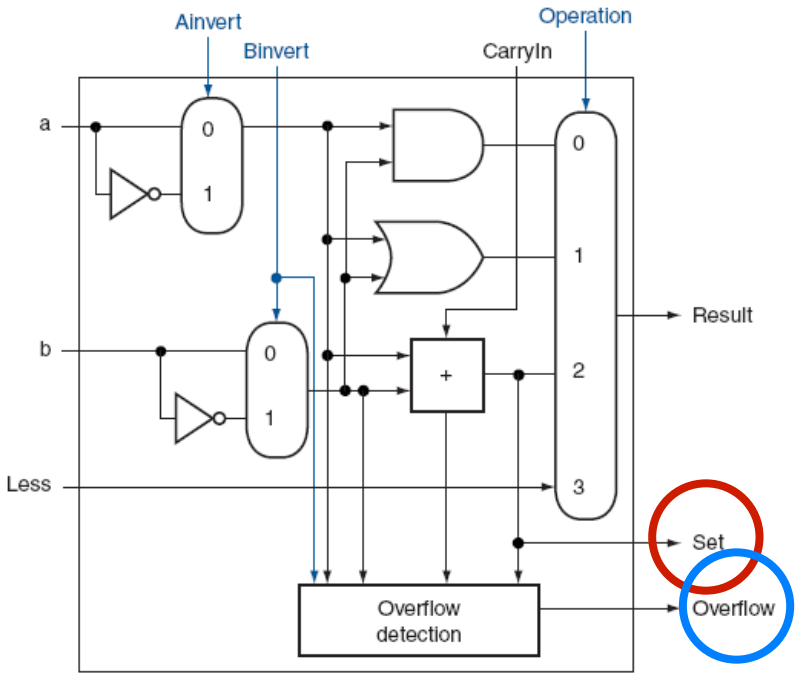
And, NAND???



Implementing SLT (set-less-than)



1-bit ALU for bits 0~30



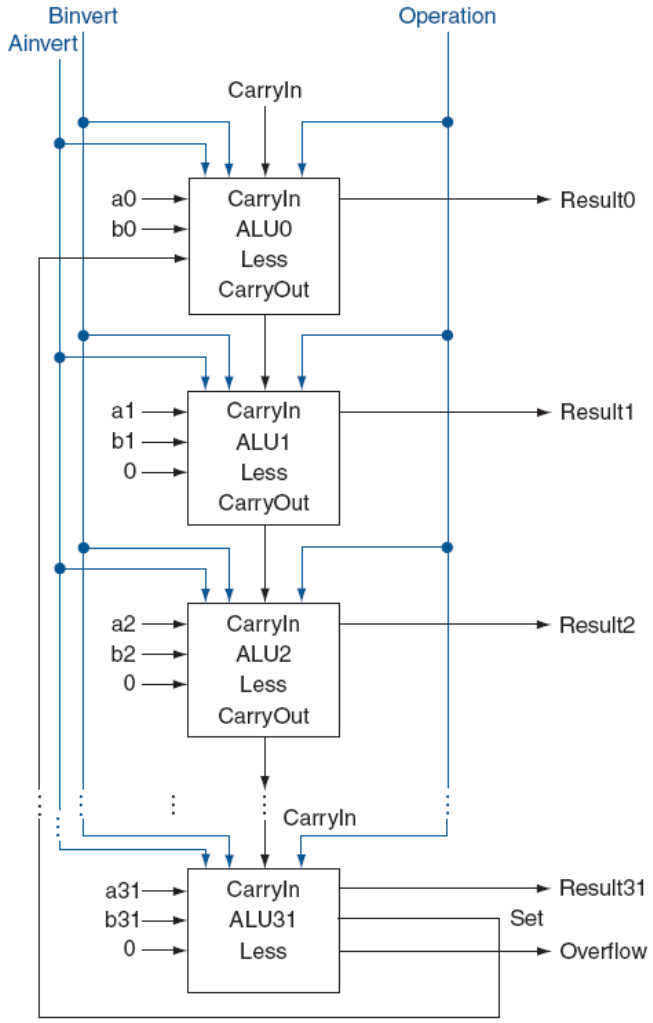
1-bit ALU for bit 31

Implementing SLT (set-less-than)

SLT uses subtraction
slt $t0,$t1,$t2
$t1<$t2: $t1-$t2 gives negative result
set is 1 when negative

Setting the control
perform subtraction (Cin=1,Binvert=1)
select Less as output (Operation=3)
ALU31's Set connected to ALU0 Less

Why do we need Set? Could we use just the Result31?



Supporting BEQ and BNE

BEQ uses subtraction
beq $t0,$t1,LABEL
perform $t0-$t1
result=0 ➔ equality

Setting the control
subtract (Cin=1,Binvert=1)
select result (operation=2)
detect zero result