ARITHMETIC HARDWARE

Introduction

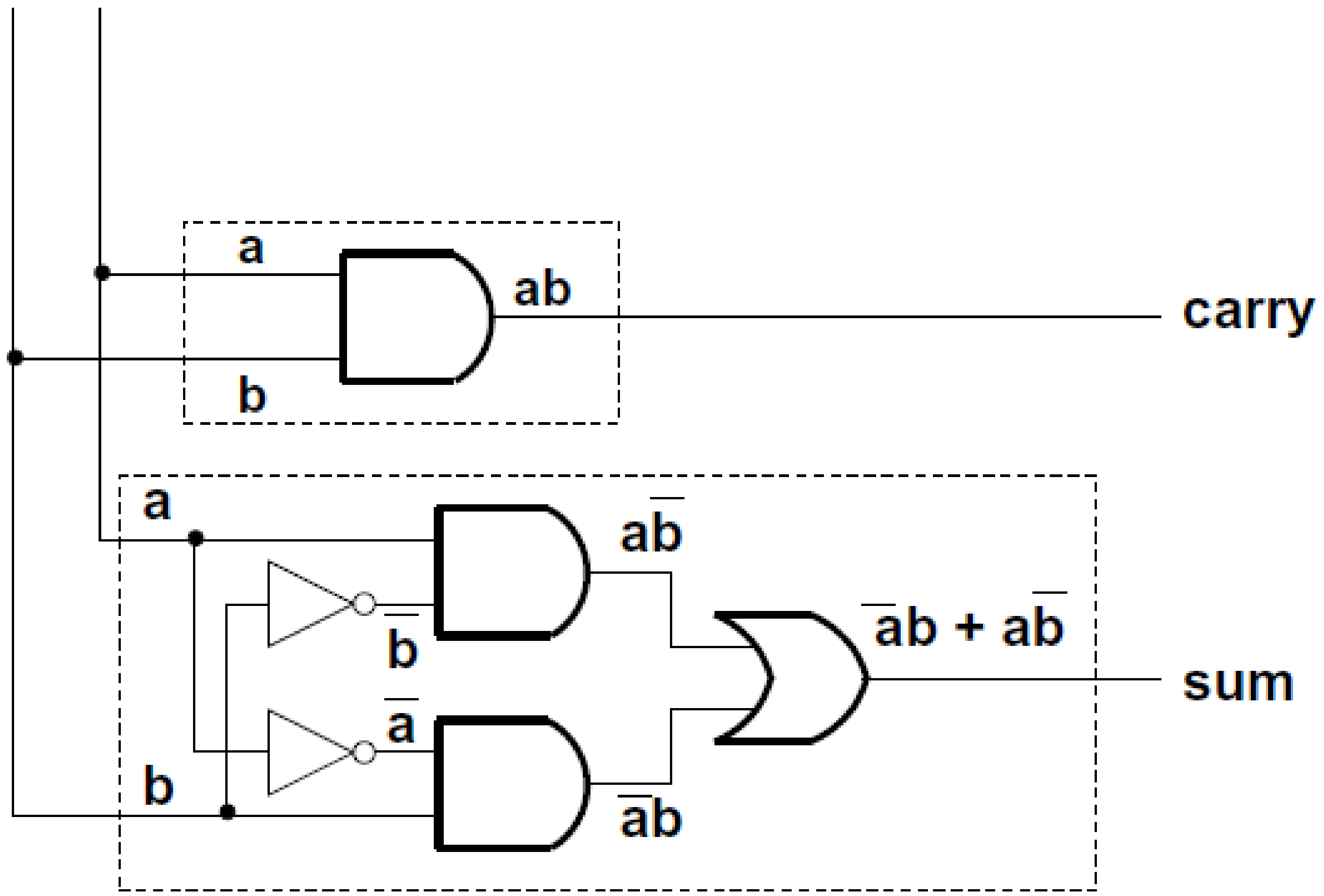
- Adders
  - Half
  - Full
  - Ripple-Carry
  - Carry-Lookahead
  - Carry-Select
- Arithmetic Logic Unit

1-Bit Half Adder

A	B	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Sum = A’B + AB’
- Cout = AB

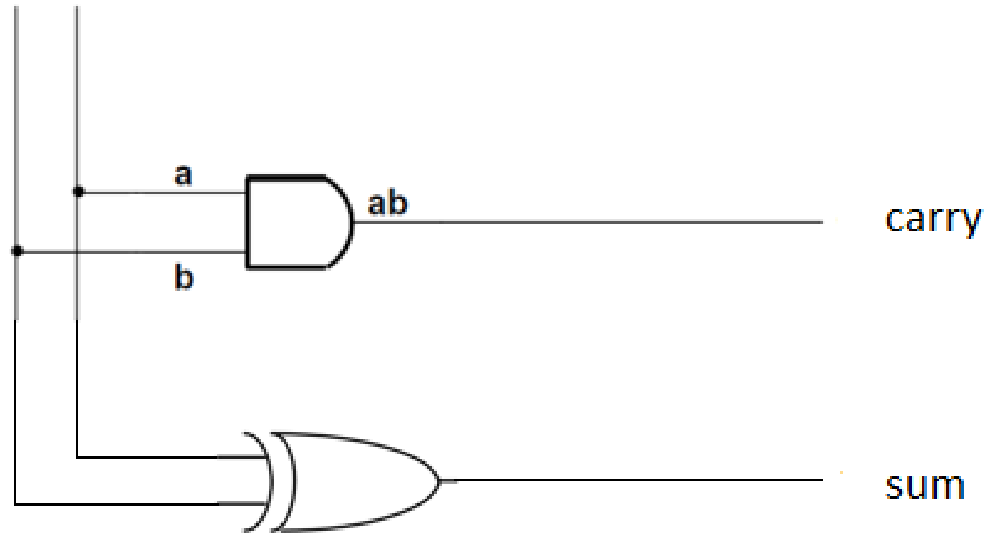
1-Bit Half Adder



1-Bit Half Adder

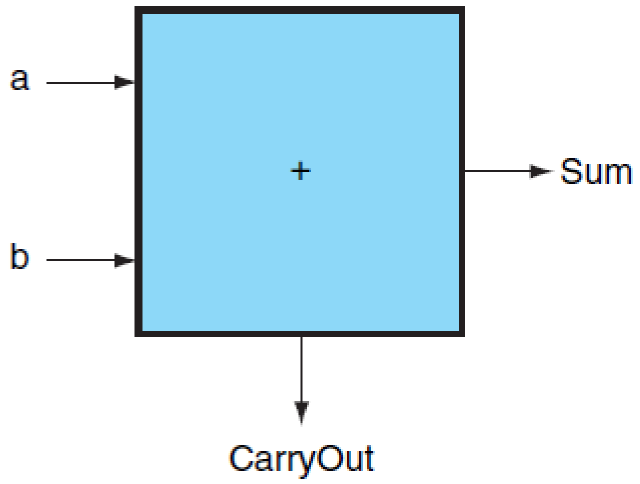
- Sum = A’B + AB’

  = A XOR B



1-Bit Half Adder

- Block Diagram



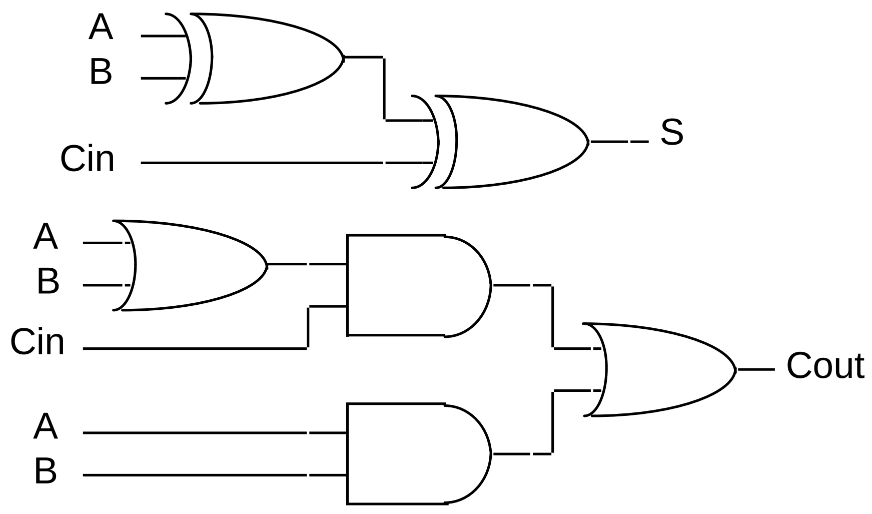
1-Bit Full Adder

- Sum = A XOR B XOR Cin
- Cout = AB + ACin + BCin

Inputs			Outputs		Comments
A	B	Cin	Cout	Sum	
0	0	0	0	0	0+0+0=00
0	0	1	0	1	0+0+1=01
0	1	0	0	1	0+1+0=01
0	1	1	1	0	0+1+1=10
1	0	0	0	1	1+0+0=01
1	0	1	1	0	1+0+1=10
1	1	0	1	0	1+1+0=10
1	1	1	1	1	1+1+1=11

1-Bit Full Adder

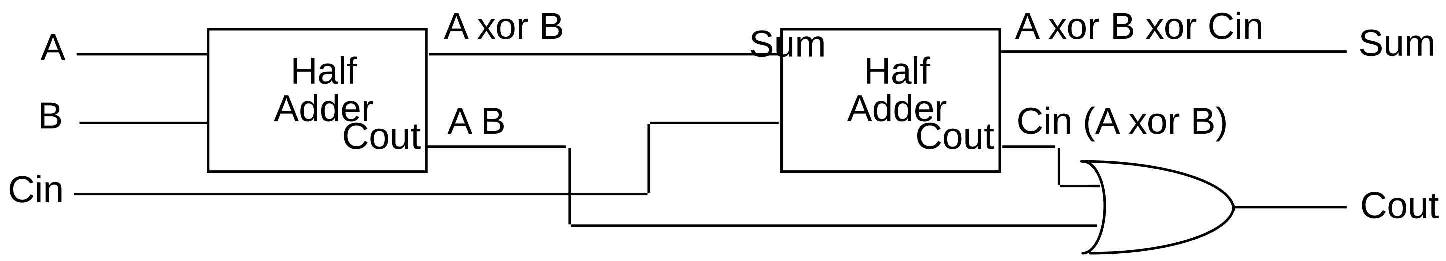
- Standard approach
  - 6 gates
  - 2 XORs, 2 ANDs, 2 ORs



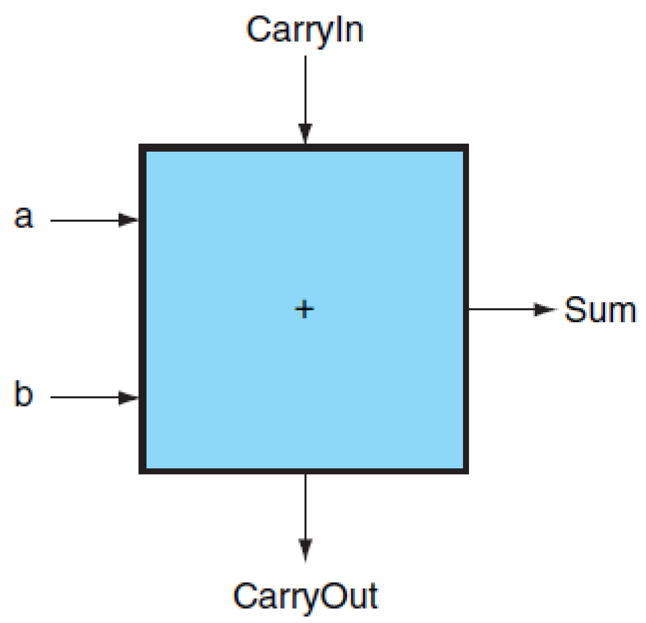
Cout = A B + Cin (A or B) = A B + B Cin + A Cin

1-Bit Full Adder

- Alternative implementation
  - 5 gates
  - half adder is an XOR gate and AND gate
  - 2 XORs, 2 ANDs, 1 OR

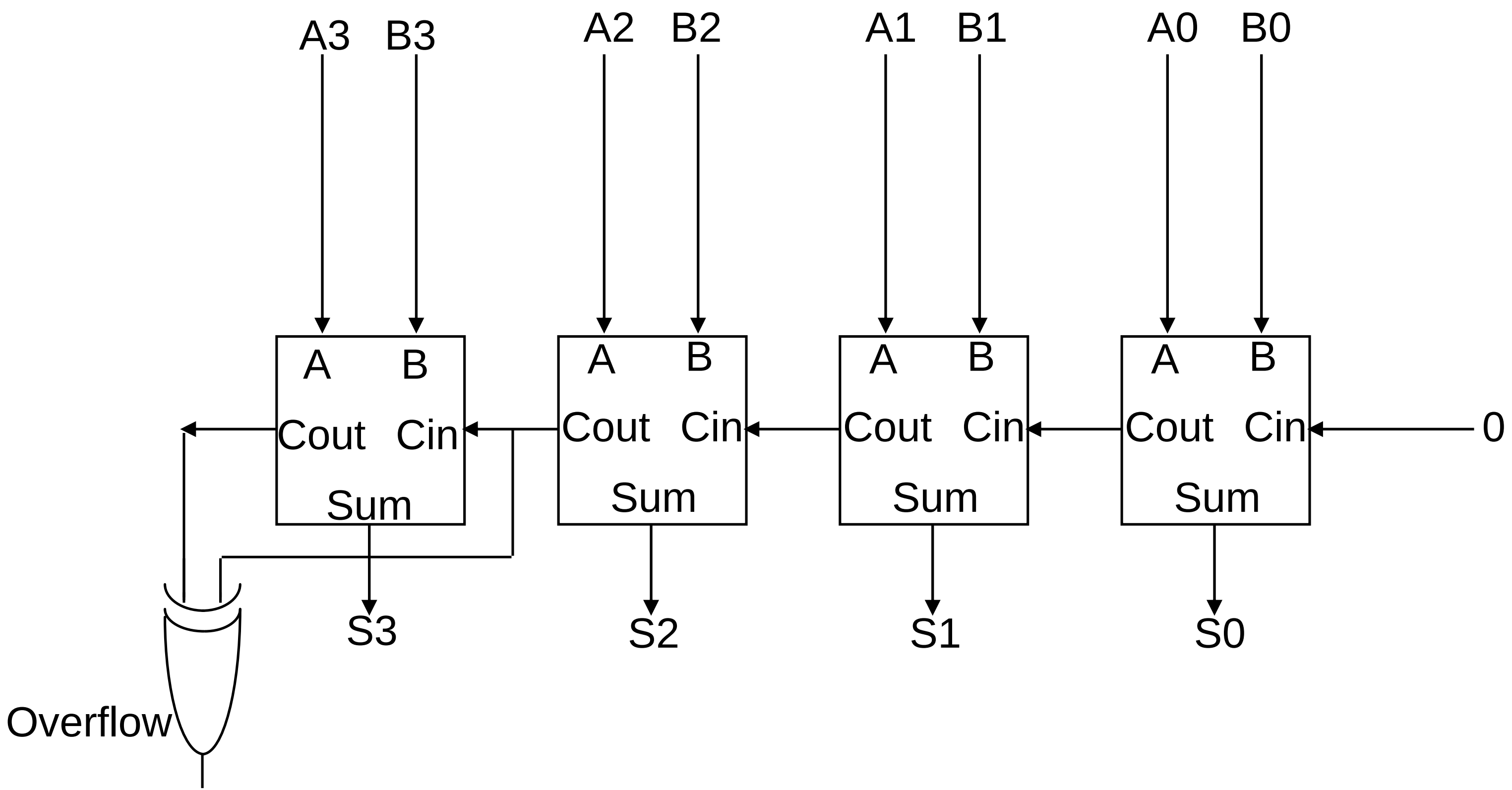


1-Bit Full Adder



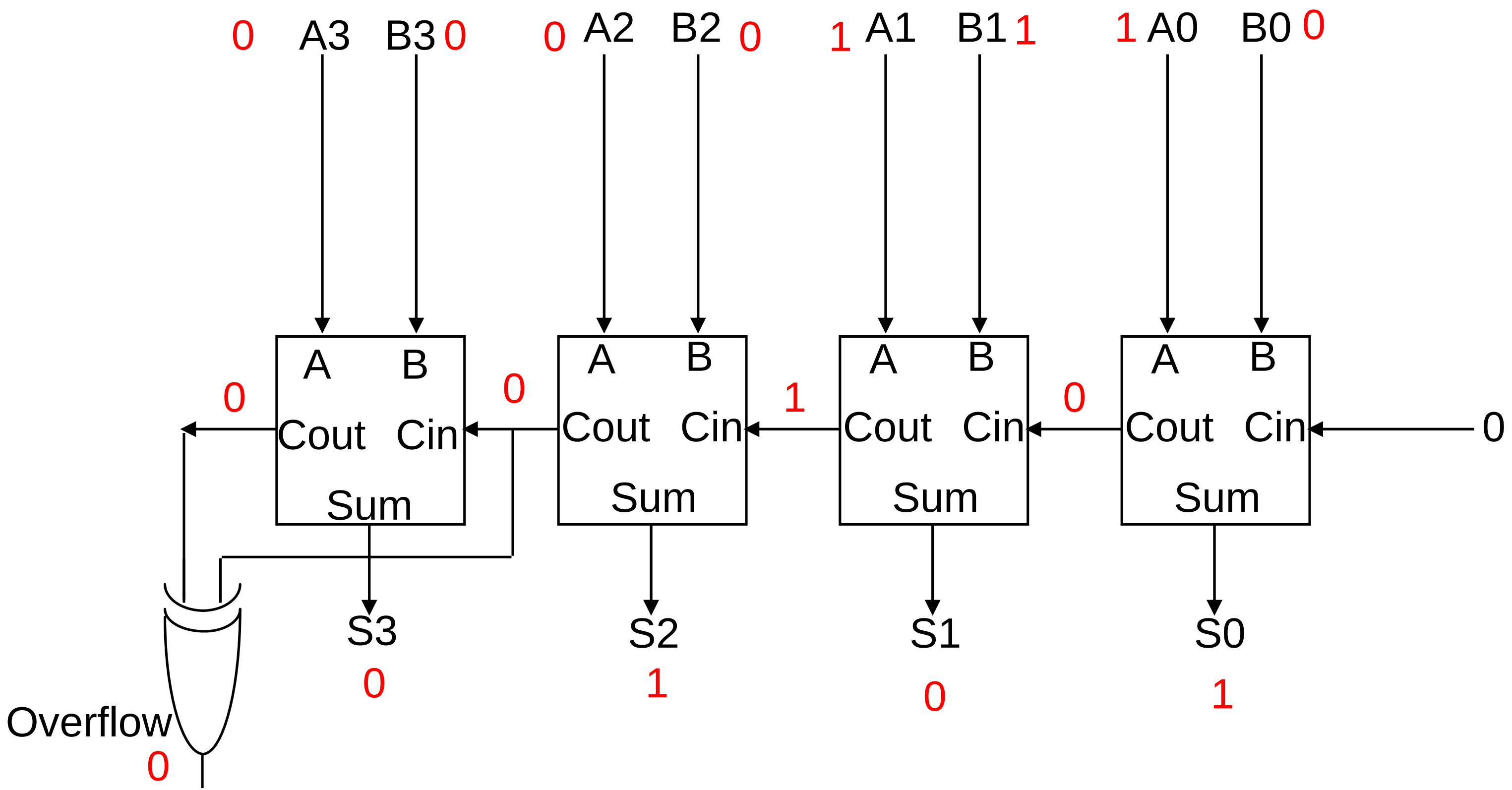
Ripple-Carry Adder

- To add binary numbers of lengths greater than 1, we cascade one bit adders together.



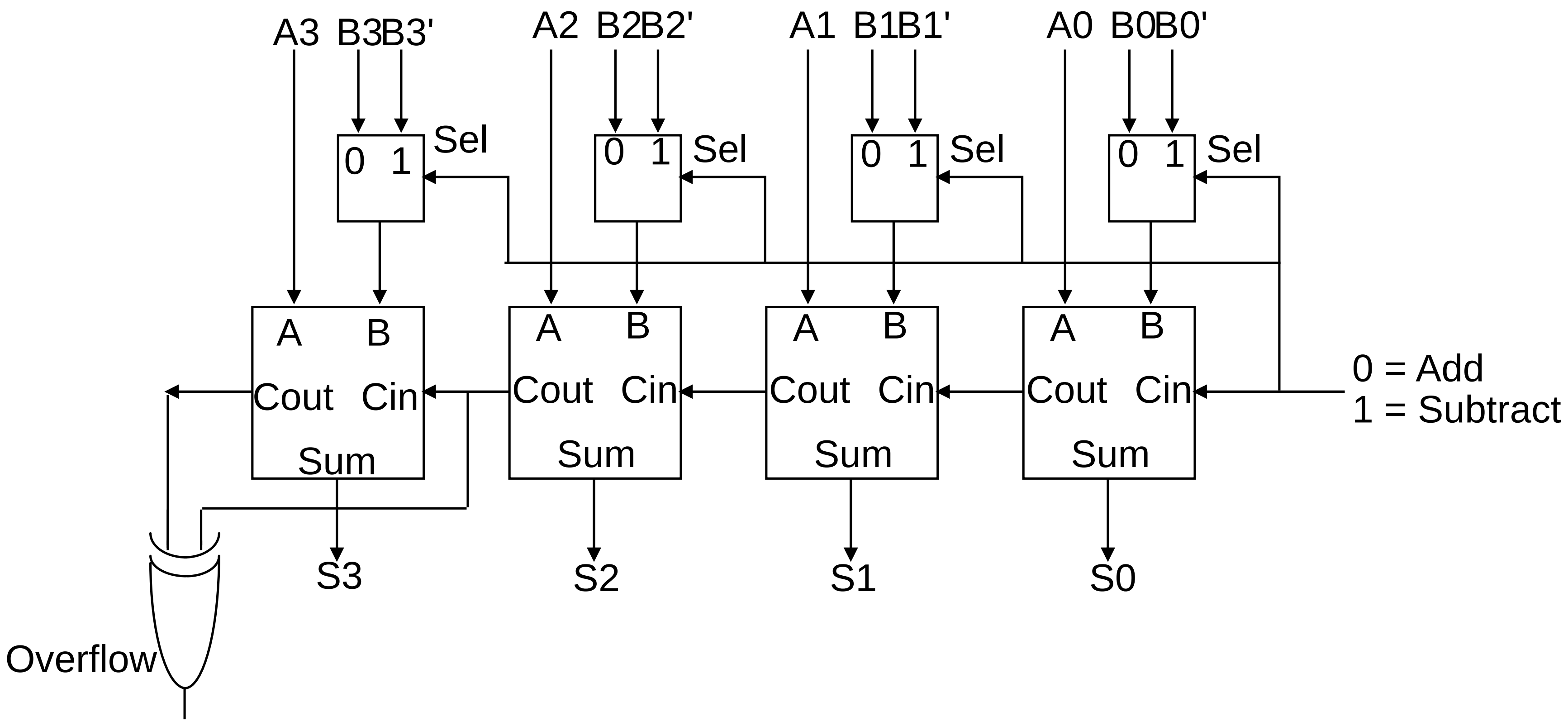
Ripple-Carry Adder: Example

- 3+2 = 5
- 0011 + 0010 = 0101



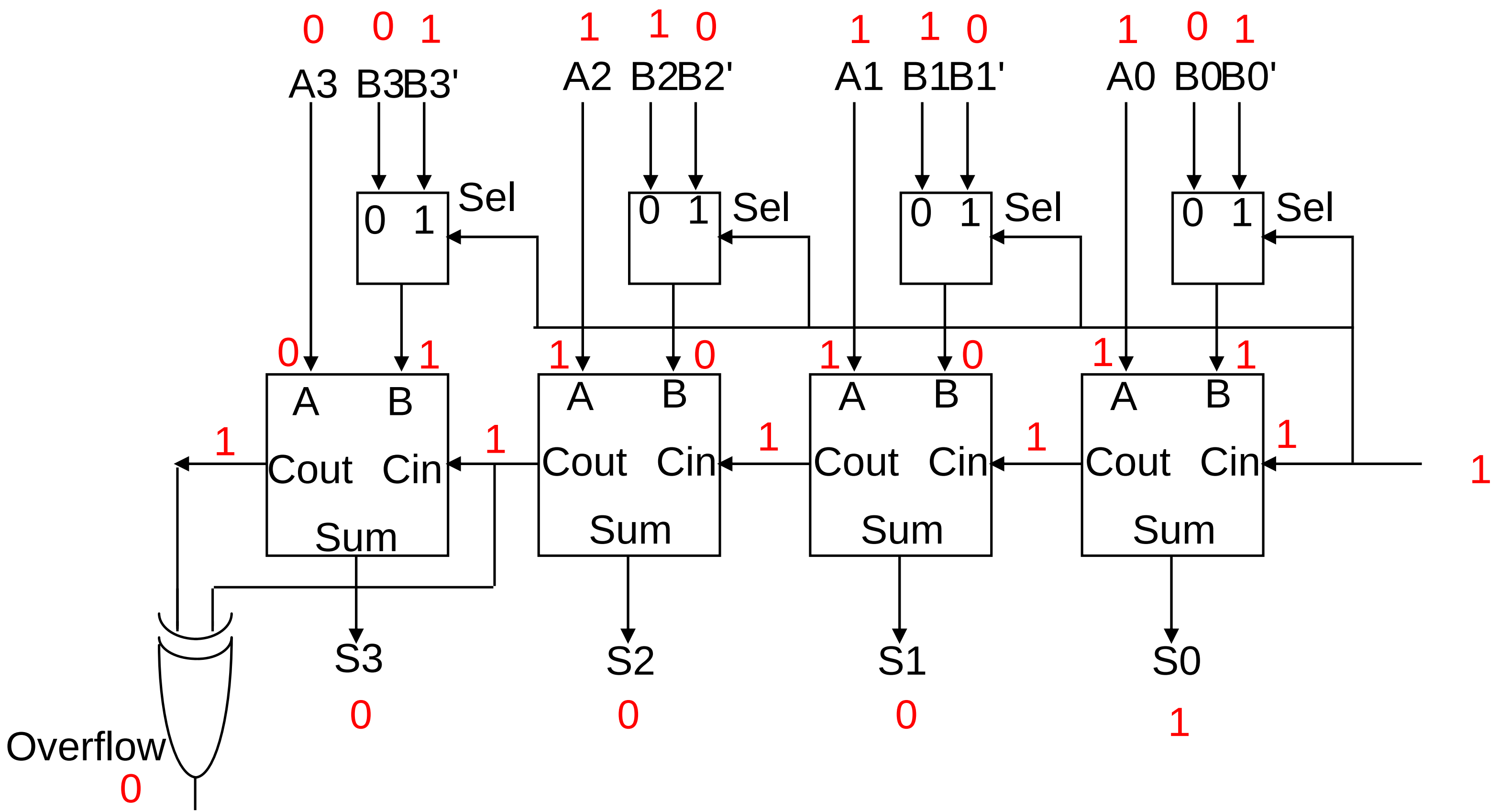
Ripple-Carry Adder: Subtraction

- Use adders to subtract with 2s complement representation
  - A – B = A + (– B) = A + B' + 1



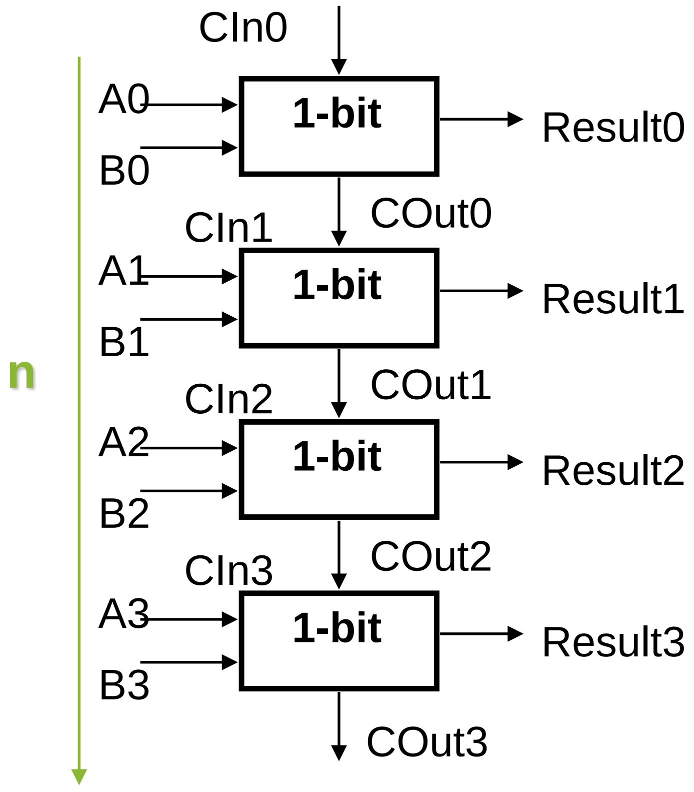
Ripple-Carry Adder: Subtraction

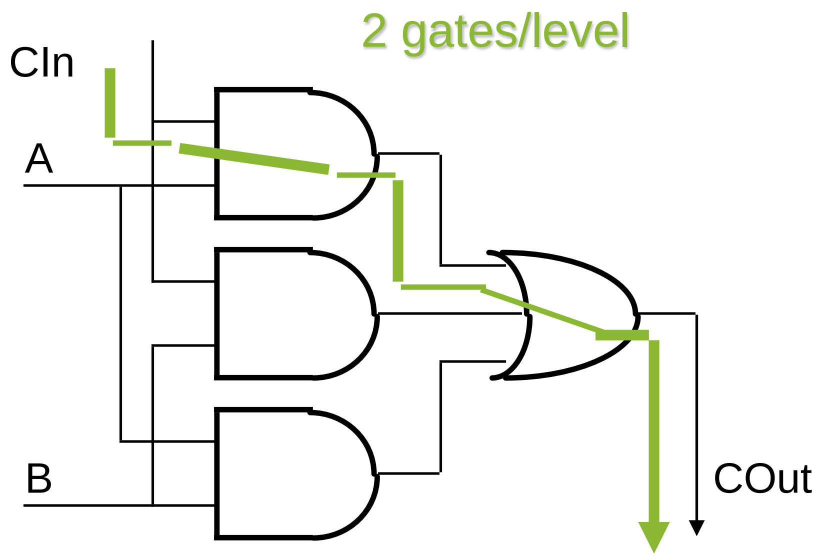
- 7 – 6 = 1
- 0111 – 0110 = 0001



Ripple-Carry Adder: Disadvantage

- Long Delay
  - Carry bit may have to propagate from LSB to MSB
  - Worst case delay for n-bit adder = 2n gate delays





Ripple-Carry Adder

- Long delay due to the propagation of carry from low to high order stages

- The key to speeding up addition is determining carry values sooner.

Carry-Lookahead

- Determine the CarryOut:
  - Cout = AB + ACin + BCin = AB + Cin(A+B)

A	B	Cout	Comment
0	0	0	Carry terminate
0	1	Cin	Carry propagate
1	0	Cin	Carry propagate
1	1	1	Carry generate

Carry-Lookahead Logic

- Carry generate: $G_i = A_i B_i$
  - Must generate carry when A = B = 1
- Carry propagate: $P_i = A_i + B_i$
  - Carry-in will equal carry-out here
- Cout can be re-expressed these terms:
  - $C_{i+1} = A_i B_i + A_i C_i + B_i C_i$
    $= A_i B_i + C_i (A_i + B_i)$
    $= G_i + C_i P_i$

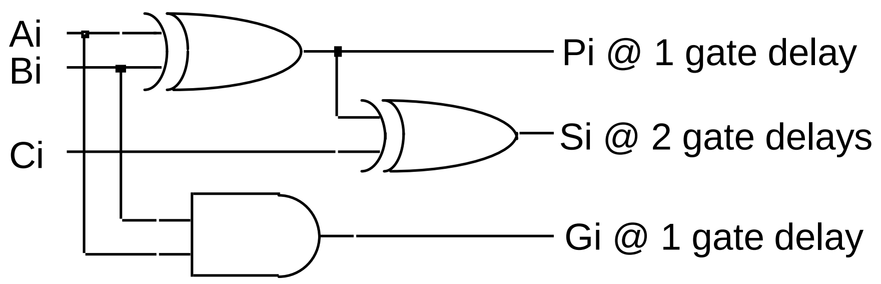
Carry-Lookahead Logic

- Assume we have a 4-bit adder:
  - C1 = G0 + P0 C0
  - C2 = G1 + P1 C1 = G1 + P1 G0 + P1 P0 C0
  - C3 = G2 + P2 C2 = G2 + P2 G1 + P2 P1 G0 + P2 P1 P0 C0
  - C4 = G3 + P3 C3 = G3 + P3 G2 + P3 P2 G1 + P3 P2 P1 G0 + P3 P2 P1 P0 C0

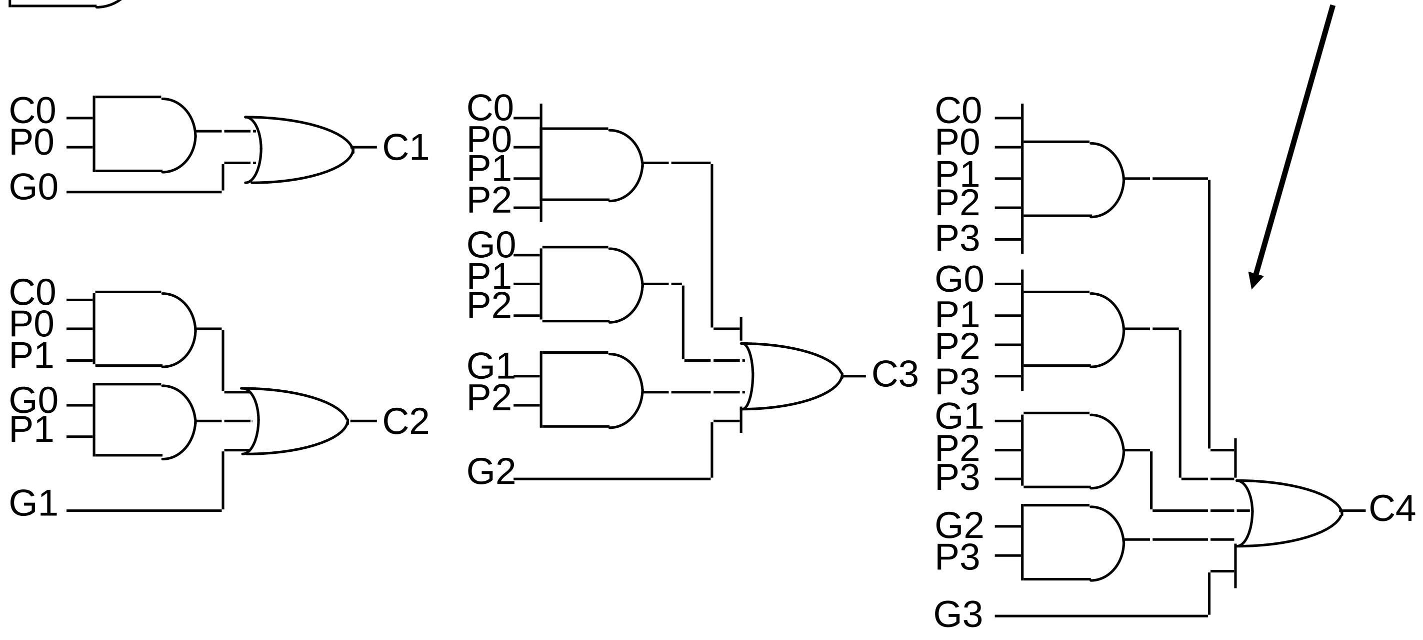
- Each of the carry equations can be implemented with two-level logic
  - All inputs are now directly derived from data inputs and not from intermediate carries
  - this allows computation of all sum outputs to proceed in parallel

Carry-Lookahead Implementation

- Adder with propagate and generate outputs

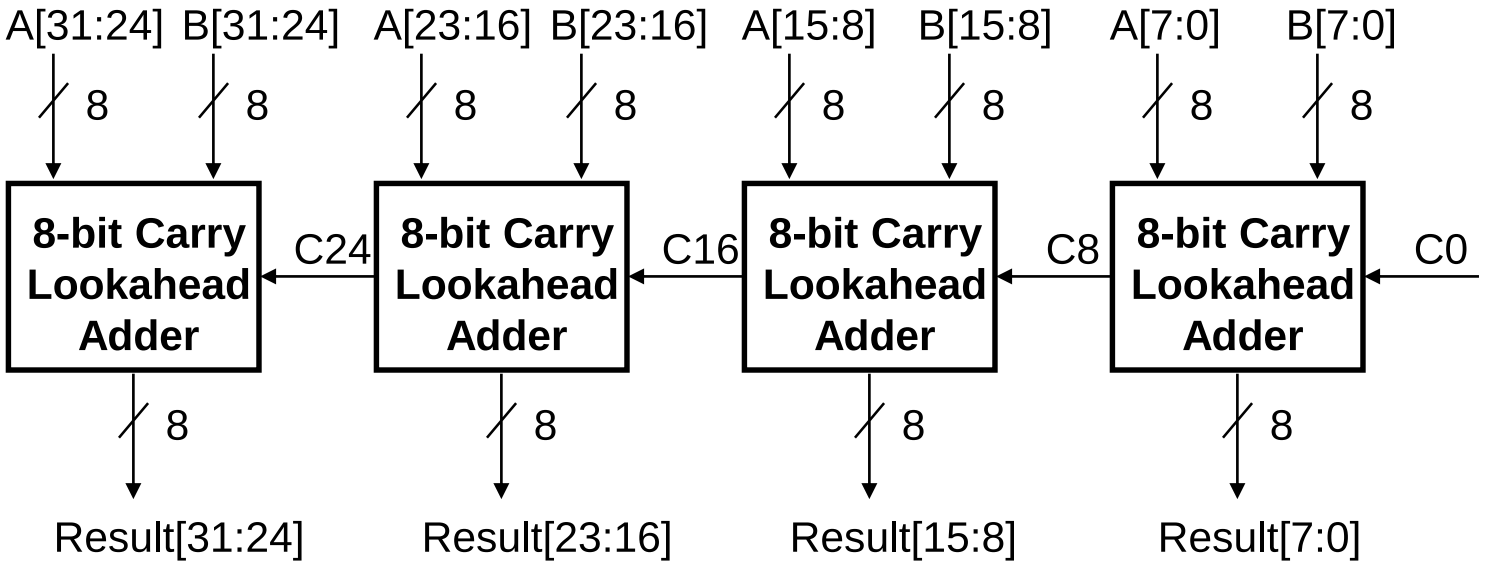


increasingly complex logic for carries



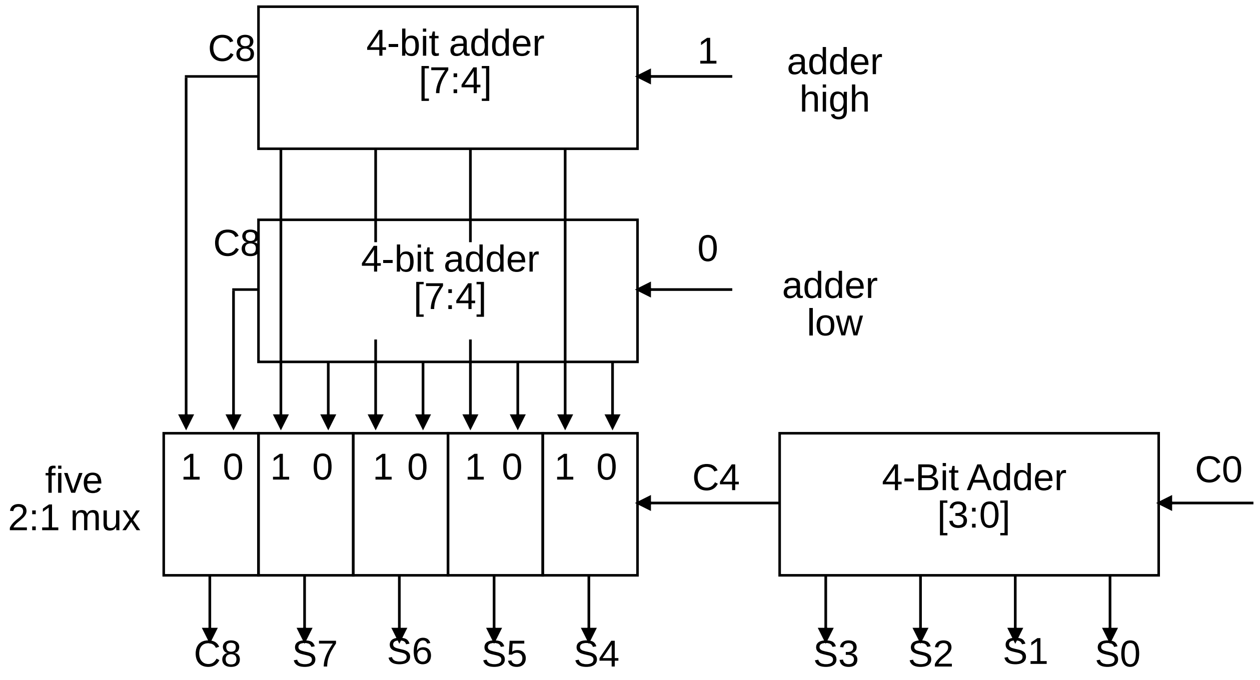
Partial Carry Lookahead Adder

- Very expensive to build "full" carry lookahead adder
    - Imagine the length of the equation for Cin31!
- In practice:
    - Connect several N-bit lookahead adders
    - Example: four 8-bit carry lookahead adders can form a 32-bit partial carry lookahead adder



Carry-Select Adder

- Redundant hardware to make carry calculation go faster
  - Compute two high-order sums in parallel while waiting for carry-in
  - One assuming carry-in is 0 and another assuming carry-in is 1
  - Select correct result once carry-in is finally computed



Arithmetic Logic Unit

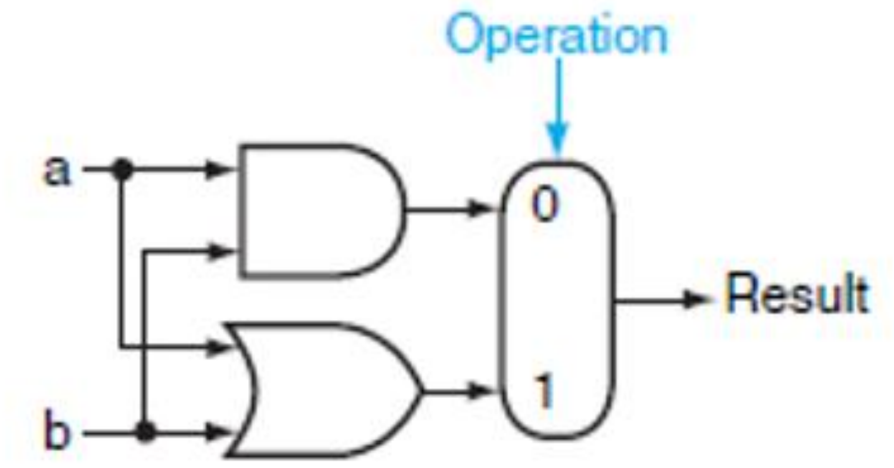
- The Arithmetic Logic Unit (ALU) is the brawn of the computer
  - Performs arithmetic operations (addition, subtraction)
  - Performs logical operations (AND, OR, NOR)
  - Performs logical comparisons (Less Than, Equal To)

Arithmetic Logic Unit

- We will need (at least) a 32-bit ALU
  - Connect 32 1-bit ALUs together

AND, OR

- Uses: AND gate, OR gate, 2:1 mux

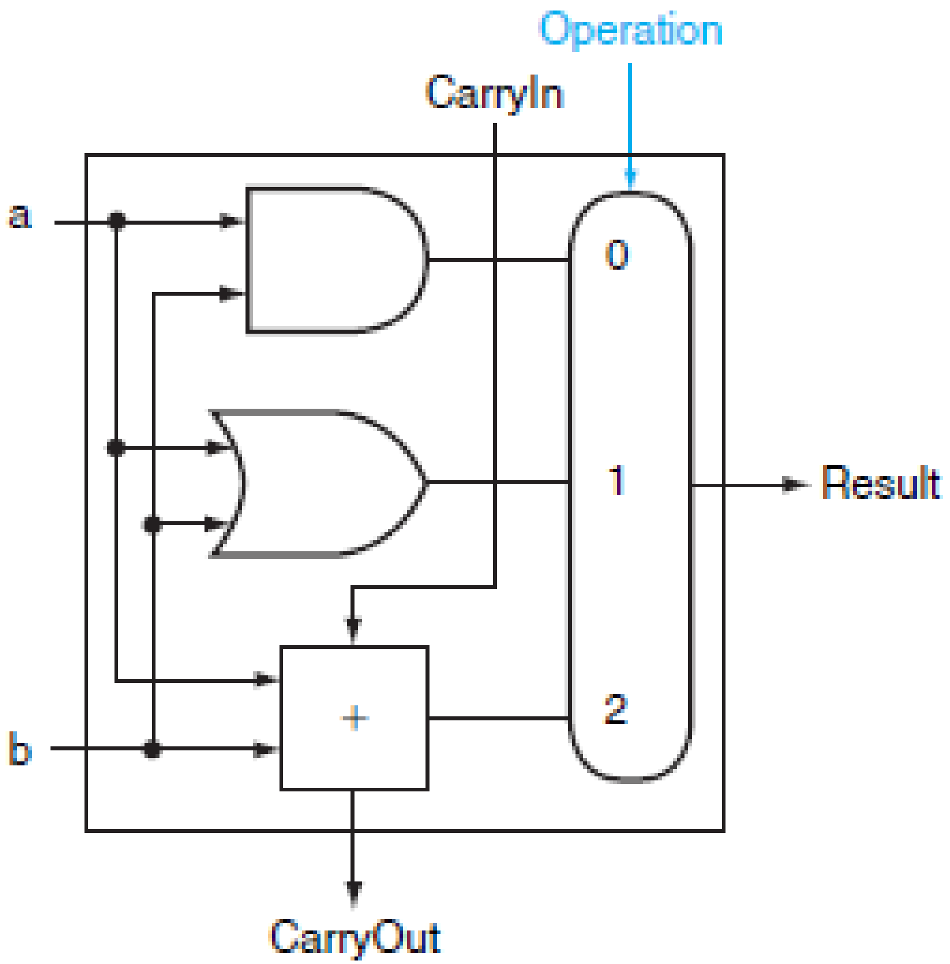


Addition

- Add a 1-bit full adder

Addition

- This 1-bit ALU will perform AND, OR, and ADD

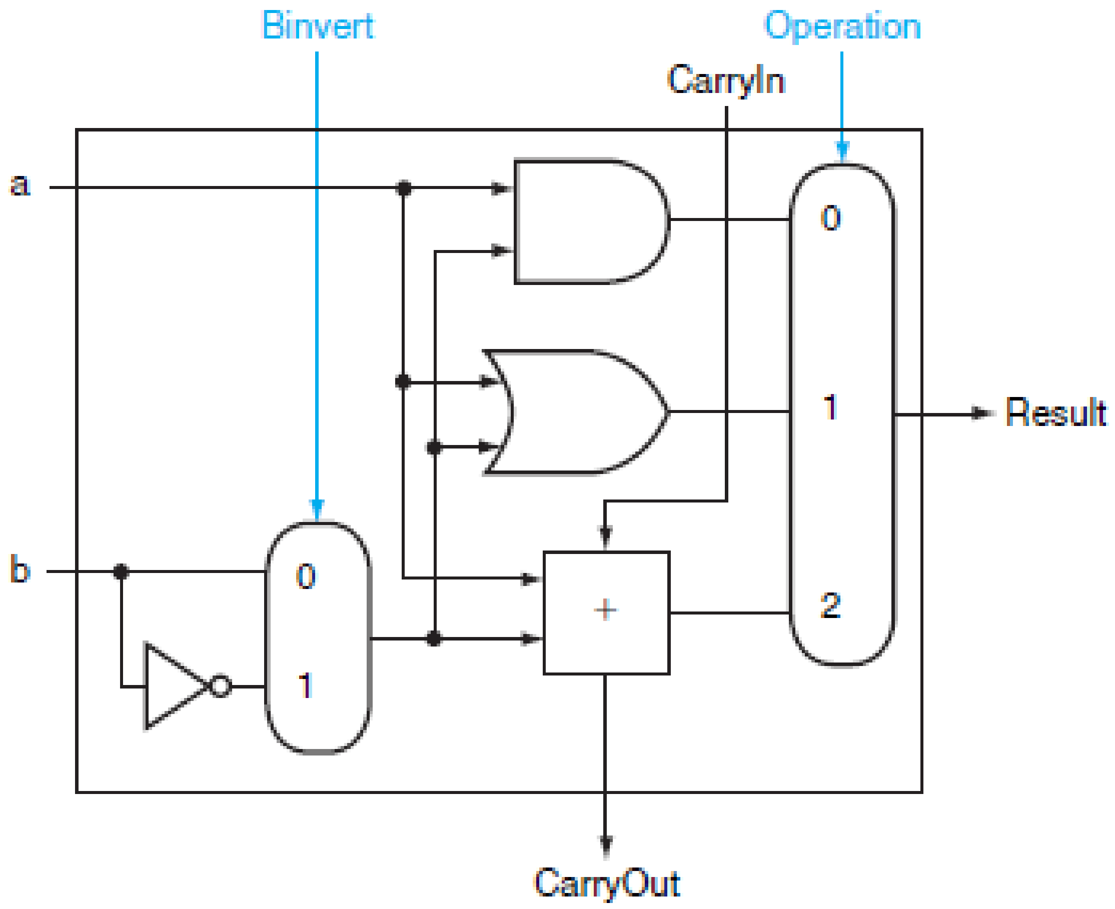


Subtraction

- (A – B) is equivalent to A + (–B)
  - Take 2’s complement of B and add A
  - 2’s Complement: invert every bit and add 1

- To invert B
  - Add a 2:1 mux that chooses between b and b’
- To add 1
  - Set the CarryIn to 1 instead of 0

Subtraction

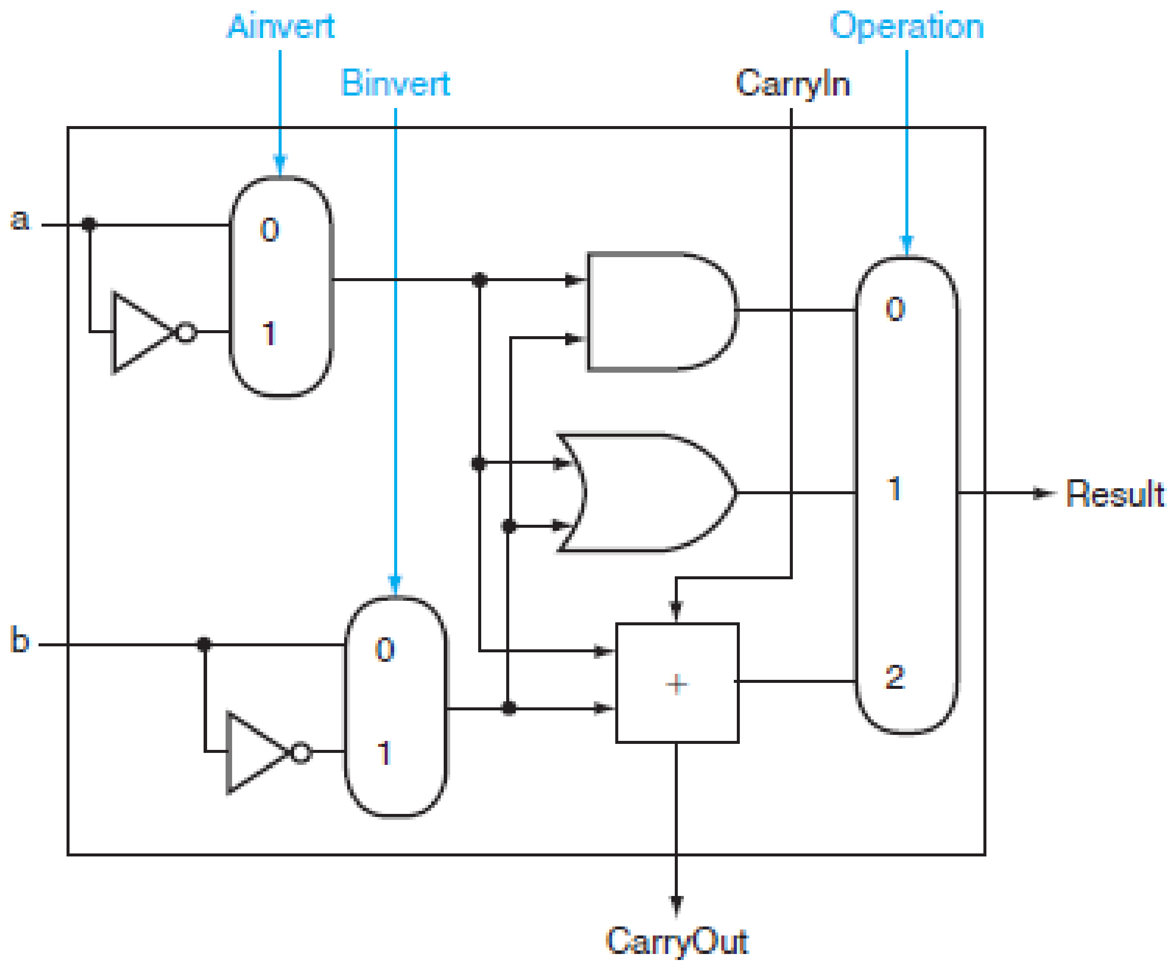


NOR

- NOR = NOT (A OR B)
  $= (A + B)'$
  $= A'*B'$
  $= A'$ AND B'

- We already have an AND gate and an inverter for B
  - Add an inverter for A

NOR


Ainvert
Binvert
CarryIn
Operation
a
b
0
1
0
1
0
1
2
+
Result
CarryOut

Set on Less Than

- Set on Less Than (slt) is a MIPS instruction
- For inputs A and B it produces 1 if A<B, 0 otherwise
- All bits are set to zero except for the least significant
    - Least significant bit determined by the result of A<B

Set on Less Than

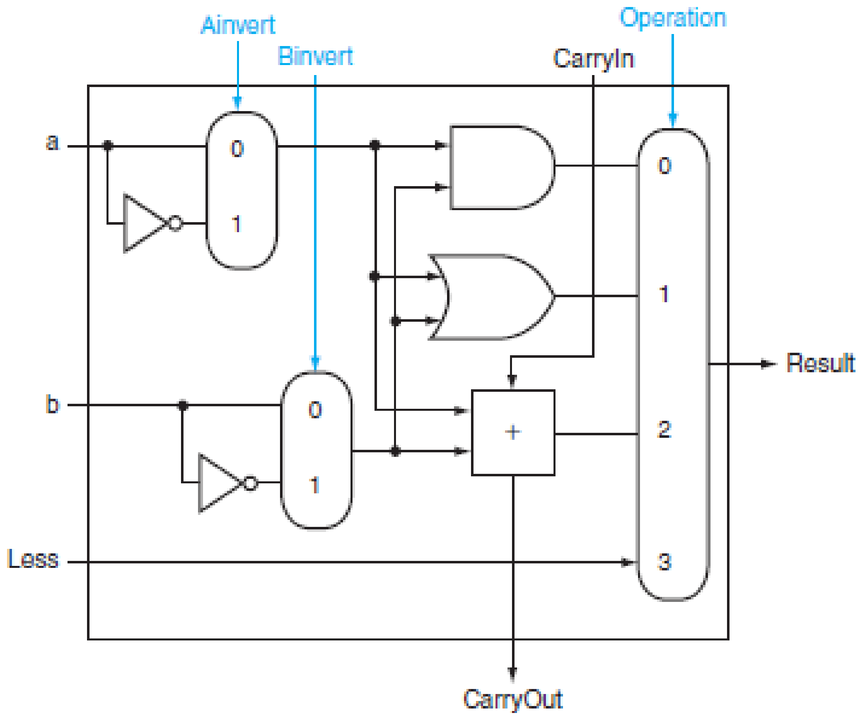
- For our ALU to perform this operation we need:
  - A bigger mux
  - Input for the slt (called Less)
  - Method for determining if A<B

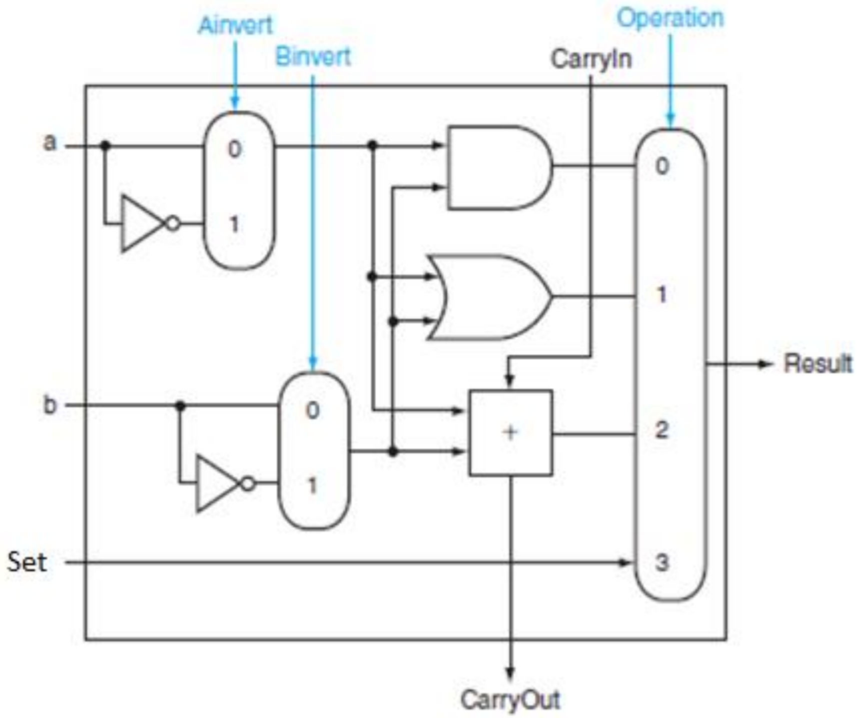
Set on Less Than

- $A < B$
- $A-B < 0$
  - Subtract and check the sign of the result
  - 1 if less than 0
  - 0 otherwise

Set on Less Than

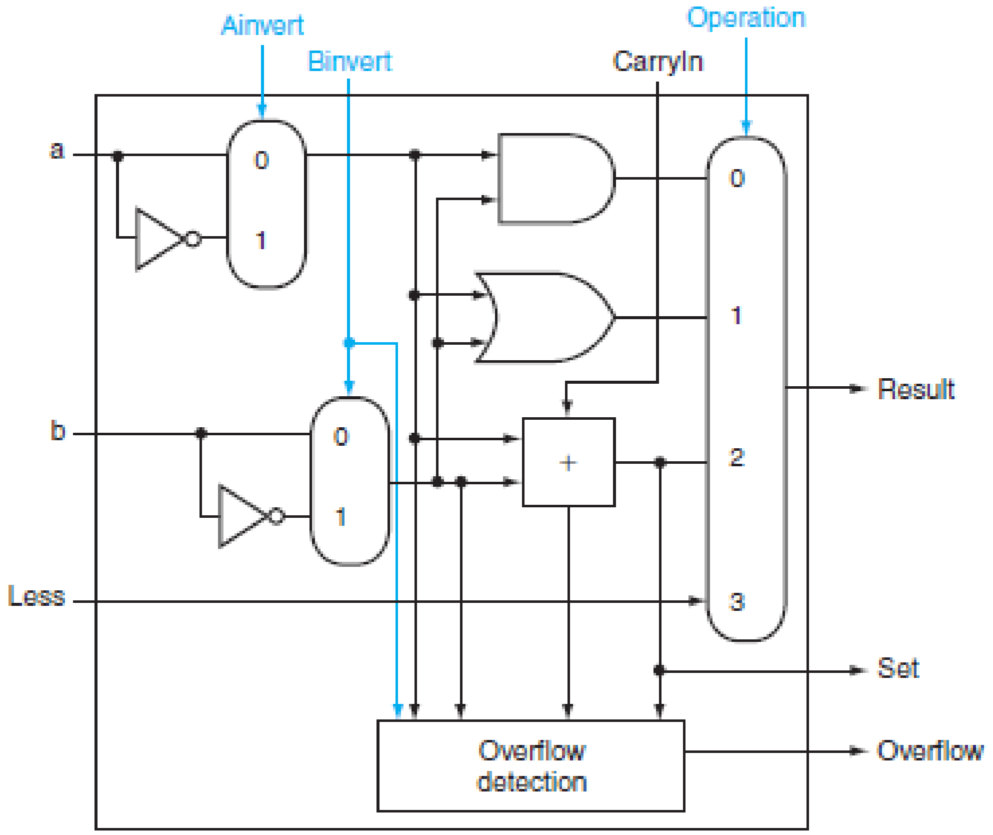
- Bits 1-31: Less = 0
- Bit 0: sign bit of A-B





Set on Less Than

- “Set” generate by most significant bit:

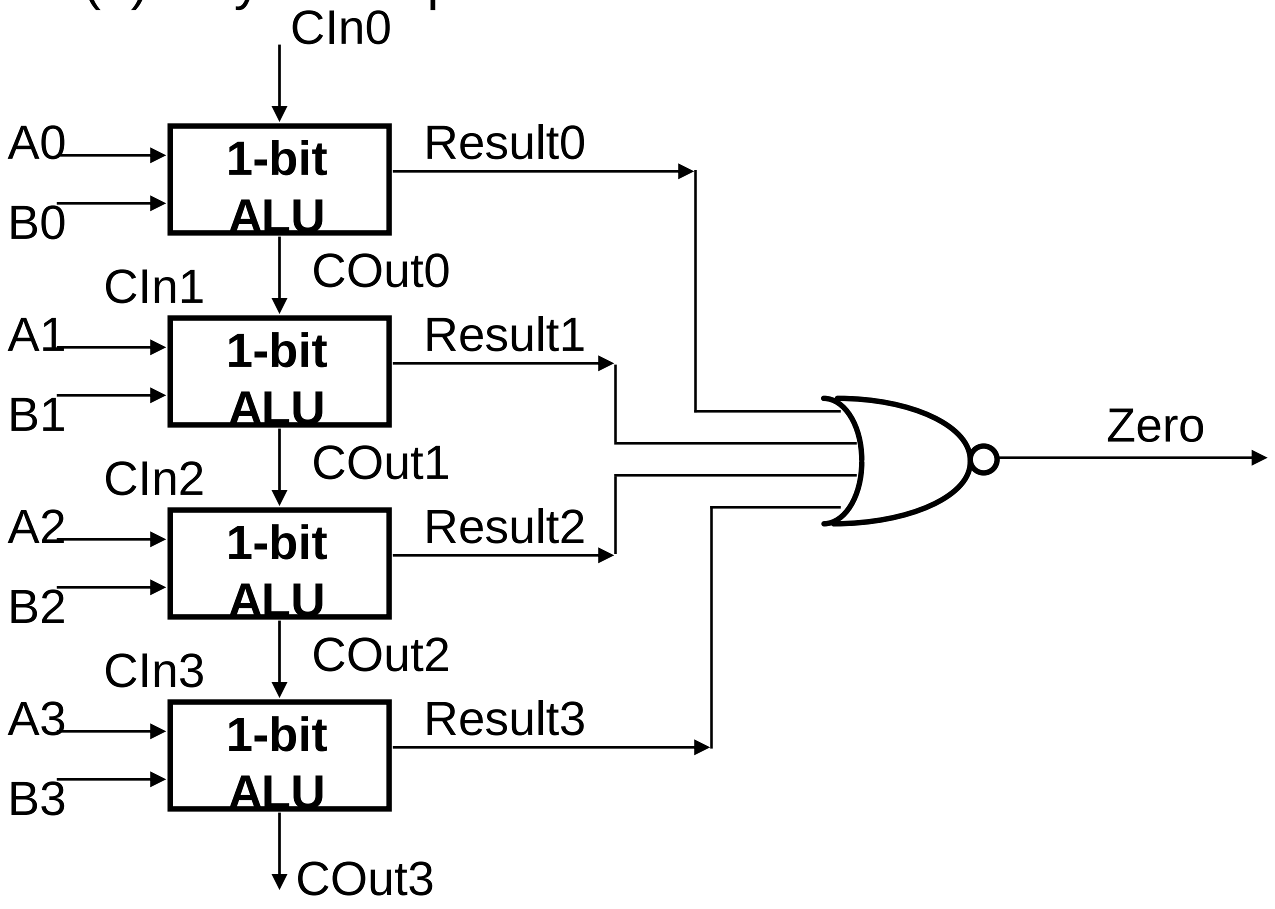


Equal To

- A = B
- A-B = 0
  - Subtract and check to see if all bits are zero
  - NOR all bits

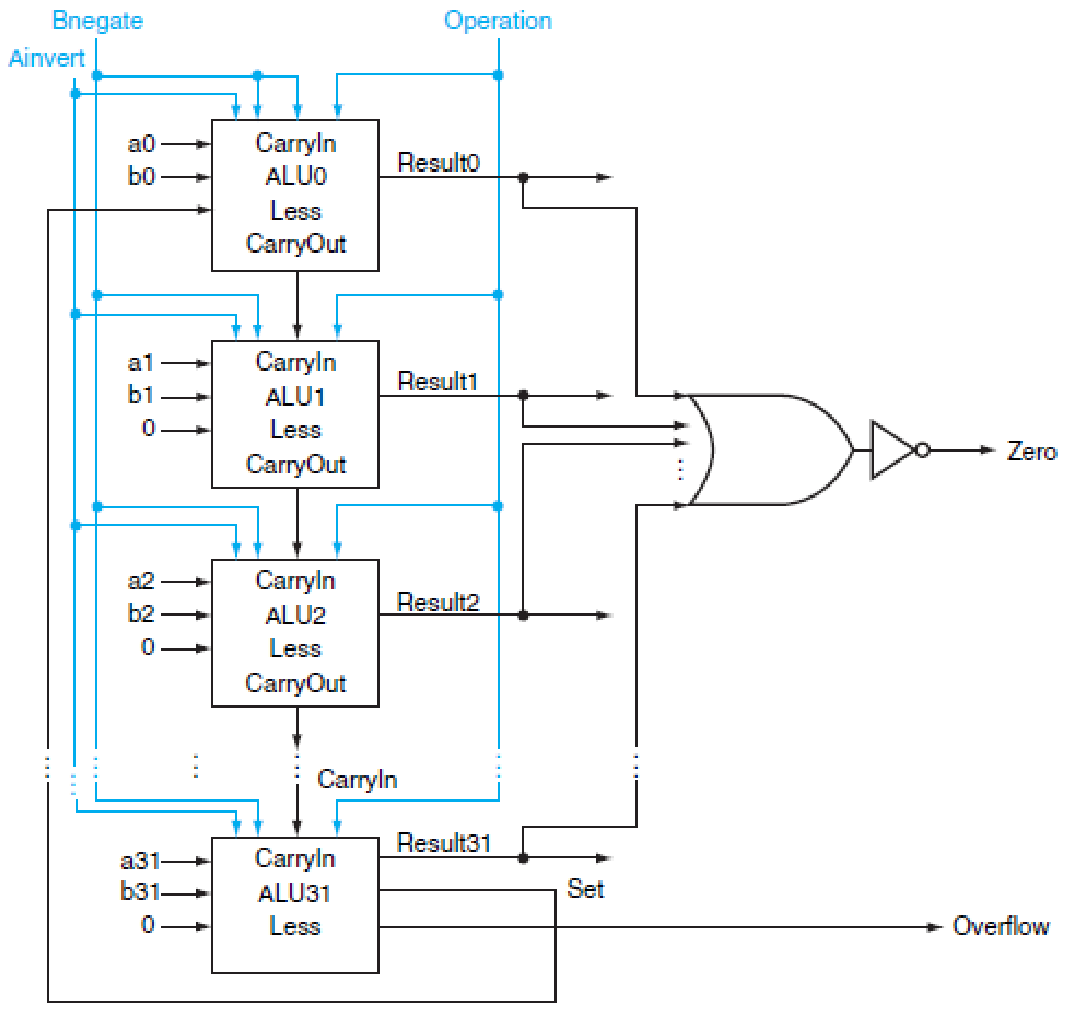
Equal To

- Zero Detection Logic is just a NOR gate
  - Output true (1) only if *all* inputs zero



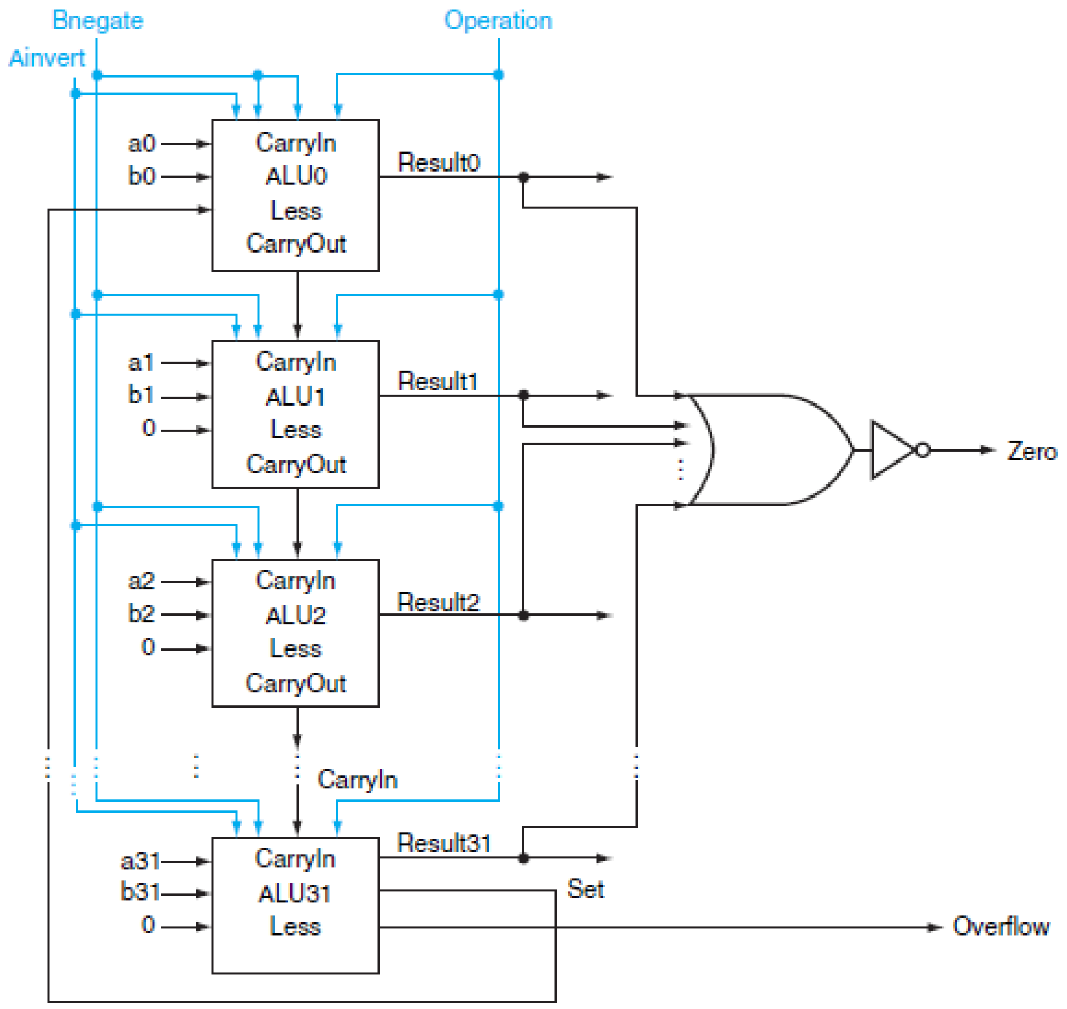
A 32-Bit ALU

- Connect:
  - 31 ALUs without overflow detection
  - 1 ALU with overflow detection (MSB)



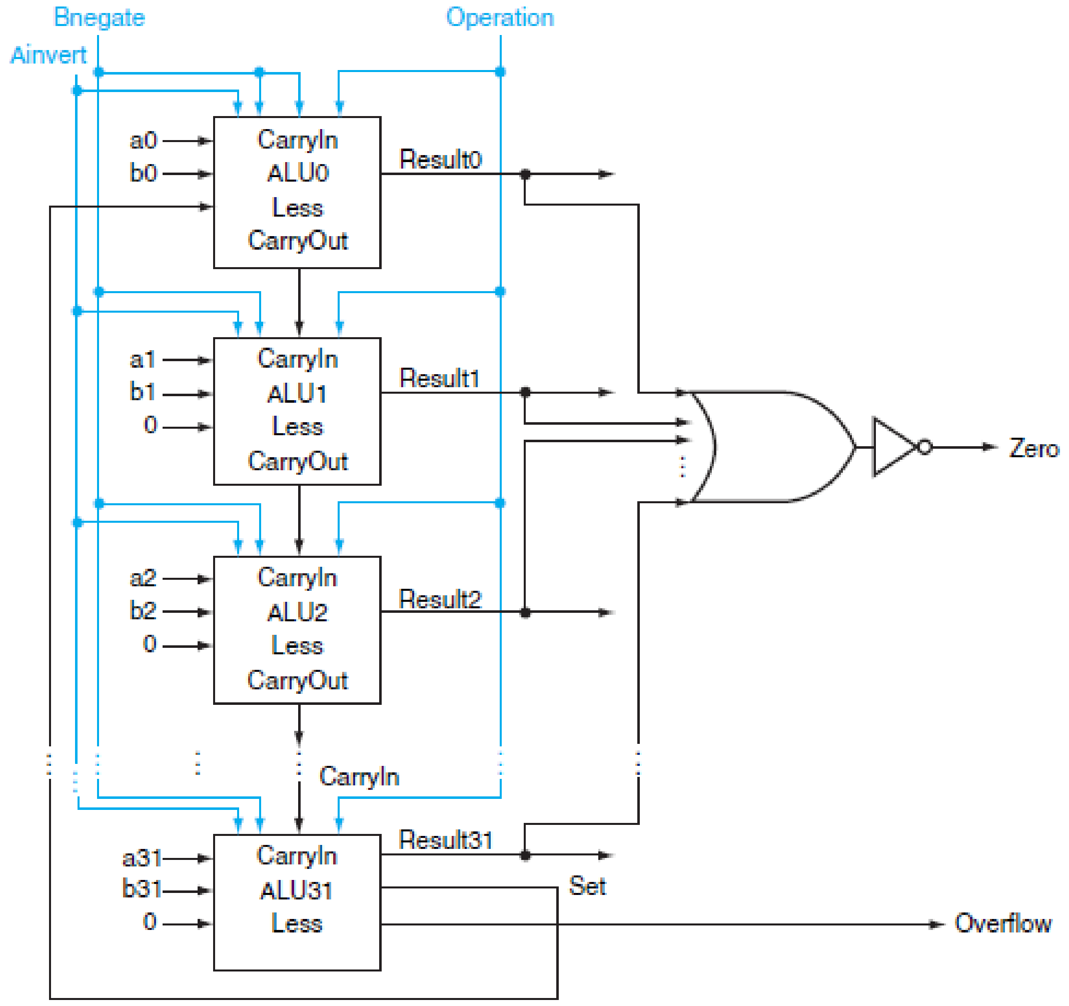
A 32-Bit ALU

- Subtraction:
  - Both CarryIn and Binvert are 1
  - Combine into 1 signal "Bnegate"



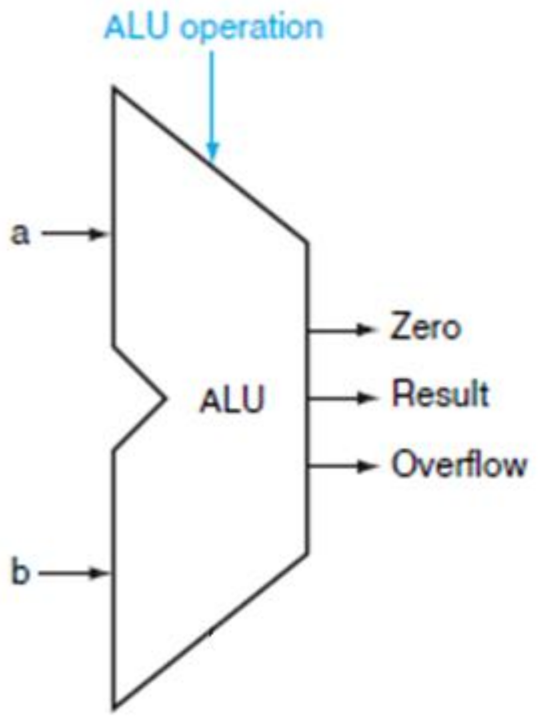
A 32-Bit ALU

- Set on Less Than:
  - Less = 0
    Bits 1-31
  - Less = sign bit
    Bit 0



A 32-Bit ALU

- Universal symbol for ALU


ALU operation
a
b
ALU
Zero
Result
Overflow