Chapter 3

Arithmetic for Computers
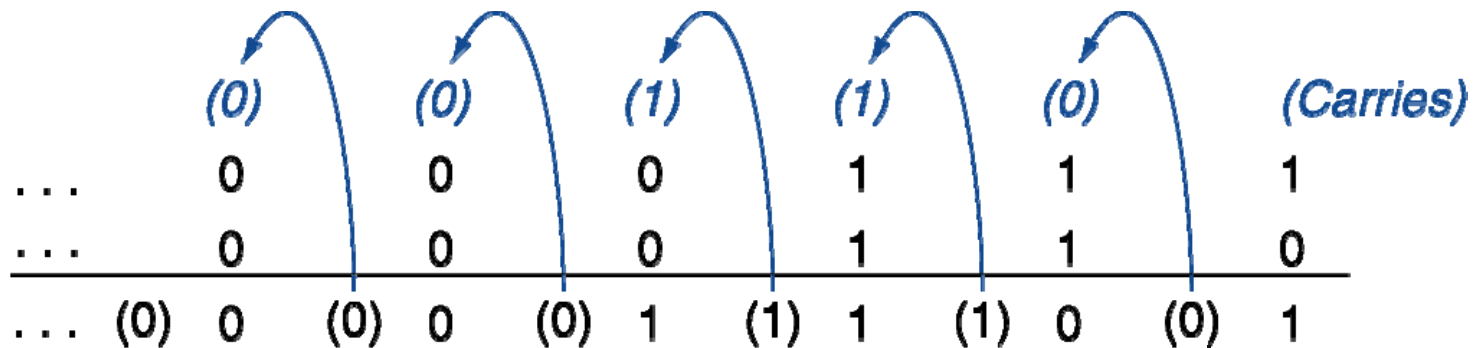
Arithmetic for Computers

- ## Operations on integers

  - ### Addition and subtraction

  - ### Multiplication and division

  - ### Dealing with overflow

- ## Floating-point real numbers

  - ### Representation and operations

Integer Addition

- ## Example: 7 + 6

- ## Overflow if result out of range

  - ### Adding +ve and –ve operands, no overflow

  - ### Adding two +ve operands

    - Overflow if result sign is 1

  - ### Adding two –ve operands

    - Overflow if result sign is 0

Integer Subtraction

- Add negation of second operand

- Example: 7 − 6 = 7 + (−6)

+7:	0000 0000 … 0000 0111
−6:	1111 1111 … 1111 1010
+1:	0000 0000 … 0000 0001

- Overflow if result out of range

  - Subtracting two +ve or two −ve operands, no overflow

  - Subtracting +ve from −ve operand

    - Overflow if result sign is 0

  - Subtracting −ve from +ve operand
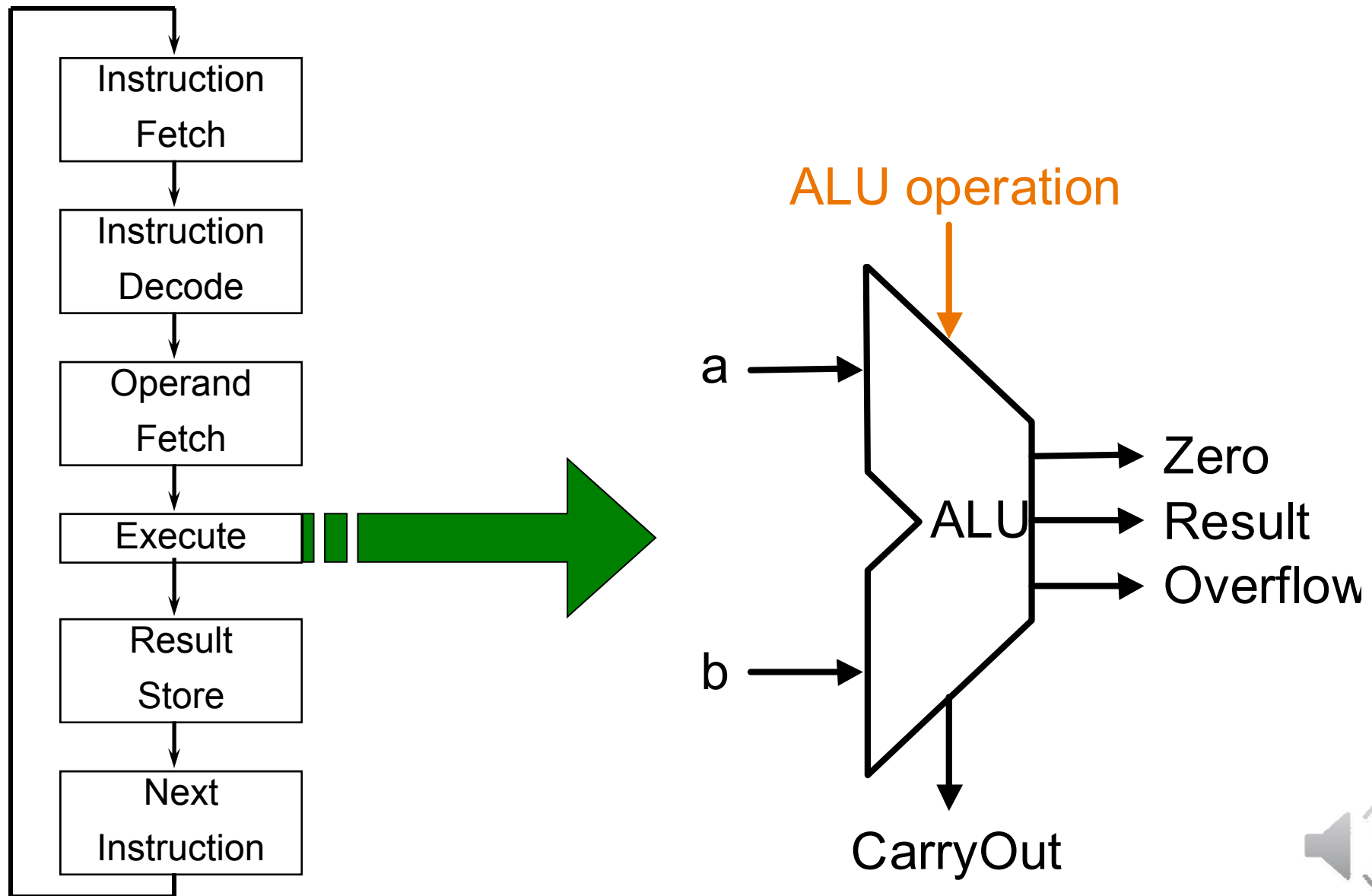
    - Overflow if result sign is 1

Dealing with Overflow

- ## Some languages (e.g., C) ignore overflow
  - Use MIPS `addu`, `addui` , `subu` instructions
- ## Other languages (e.g., Ada, Fortran) require raising an exception
  - Use MIPS `add`, `addi` , `sub` instructions
  - On overflow, invoke exception handler
    - Save PC in exception program counter (EPC) register
    - Jump to predefined handler address
    - `mfc0` (move from coprocessor reg) instruction can retrieve EPC value, to return after corrective action
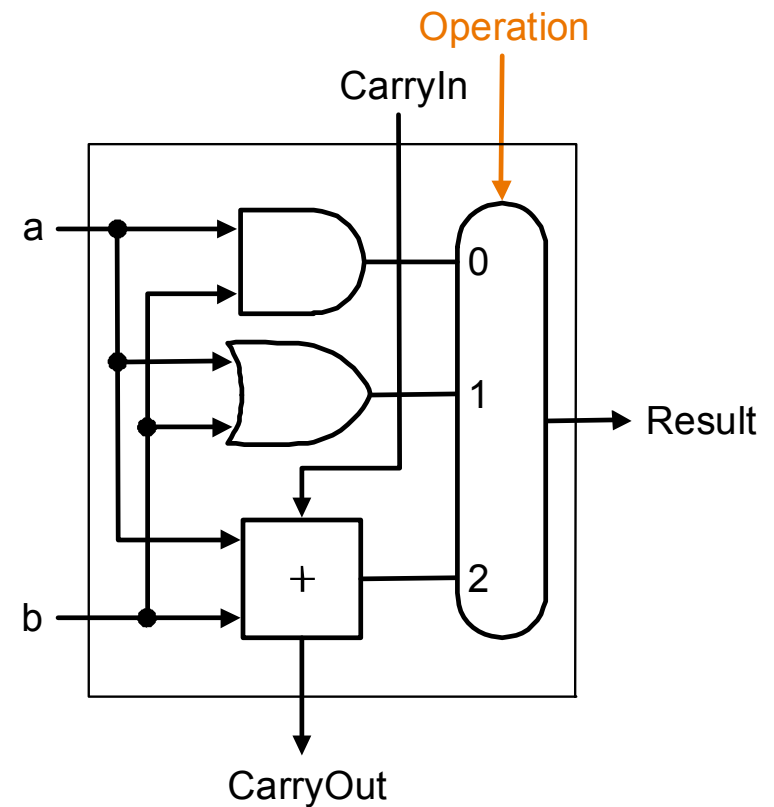
Arithmetic Logic Unit Design

Instruction Fetch
↓
Instruction Decode
↓
Operand Fetch
↓
Execute
↓
Result Store
↓
Next Instruction

ALU operation

a →
b →

ALU

→ Zero
→ Result
→ Overflow

↓ CarryOut

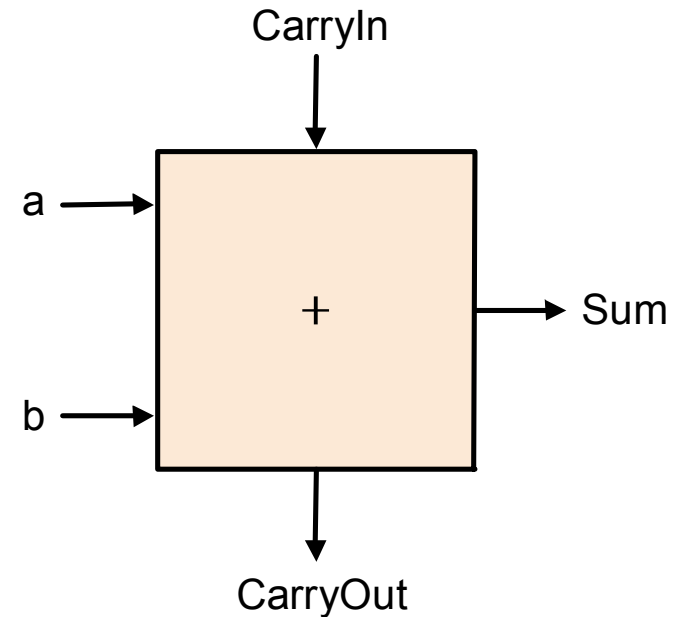Reinman 4-6

One Bit ALU

- Performs AND, OR, and ADD
  - on 1-bit operands
  - components:
    - AND gate

    - OR gate

    - 1-bit adder

    - Multiplexor

One Bit Full Adder

- Also known as a (3,2) adder
- Half Adder
  - no CarryIn

Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	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+0+1=10
1	1	1	1	1	1+1+1=11

CarryIn

a →

+ → Sum

b →

CarryOut

Reinman 4-8

CarryOut Logic Equation

- CarryOut = (!a & b & CarryIn) | (a & !b & CarryIn)
          | (a & b & !CarryIn) | (a & b & CarryIn)

- CarryOut = (b & CarryIn) | (a & CarryIn) | (a & b)

Inputs			Outputs		
a	b	CarryIn	CarryOut	Sum	Comments
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+0+1=10
1	1	1	1	1	1+1+1=11

Reinman 4-9

Sum Logic Equation

- Sum = (!a & !b & CarryIn) | (!a & b & !CarryIn)
  | (a & !b & !CarryIn) | (a & b & CarryIn)

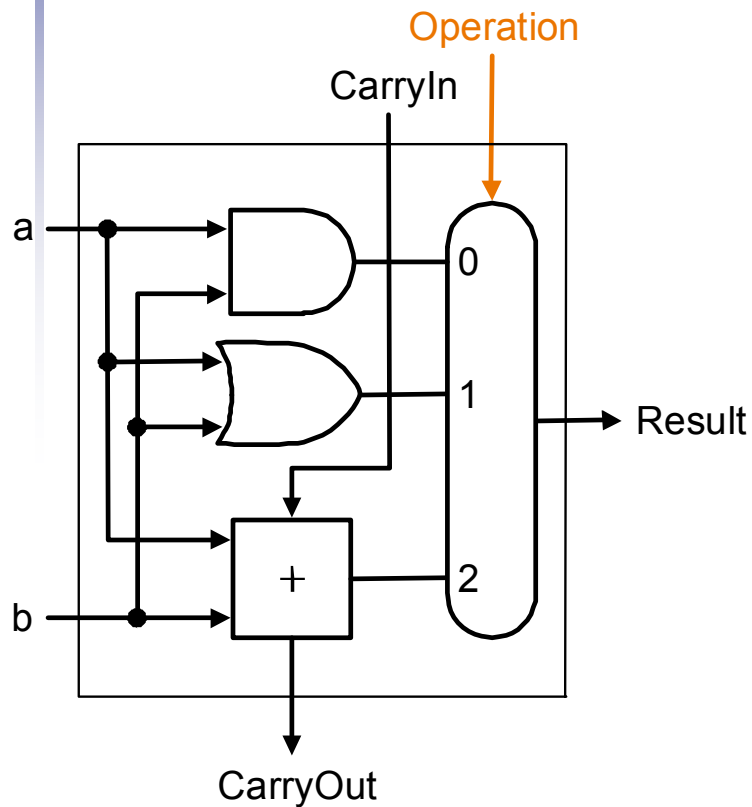| Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
0	0	0	0	0	0+0+0=00
0	0	1	0	1	0+0+1=01
0	1	0	0	1	0+1+0=01
0	1	1	1	0	0+1+1=10
1	0	0	0	1	1+0+0=01
1	0	1	1	0	1+0+1=10
1	1	0	1	0	1+0+1=10
1	1	1	1	1	1+1+1=11

Reinman 4-10

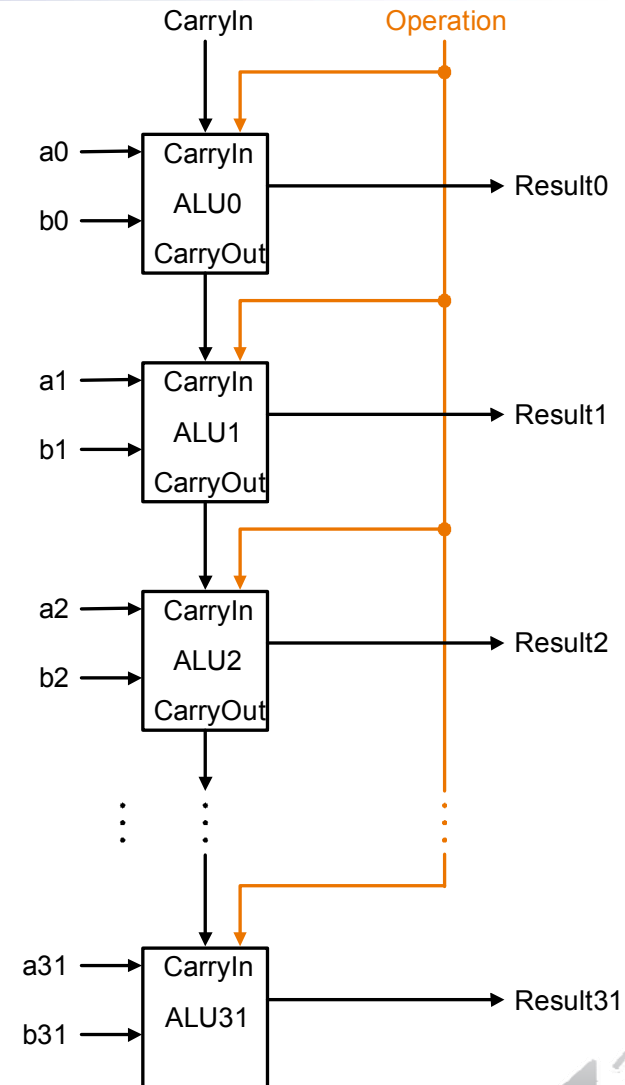# Chapter 3

Arithmetic for Computers

32-bit ALU

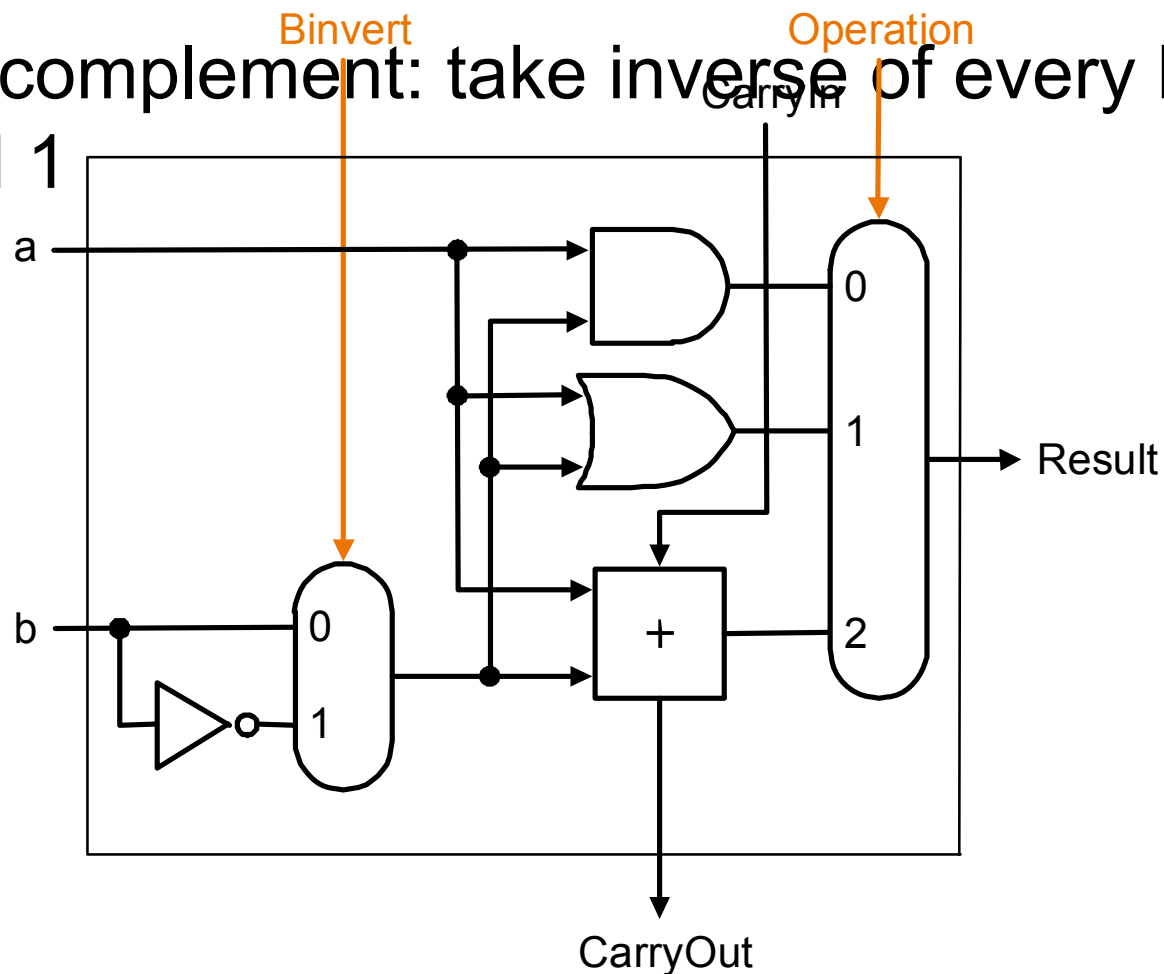- Ripple Carry ALU

1-bit ALU

32-bit ALU

Subtraction?
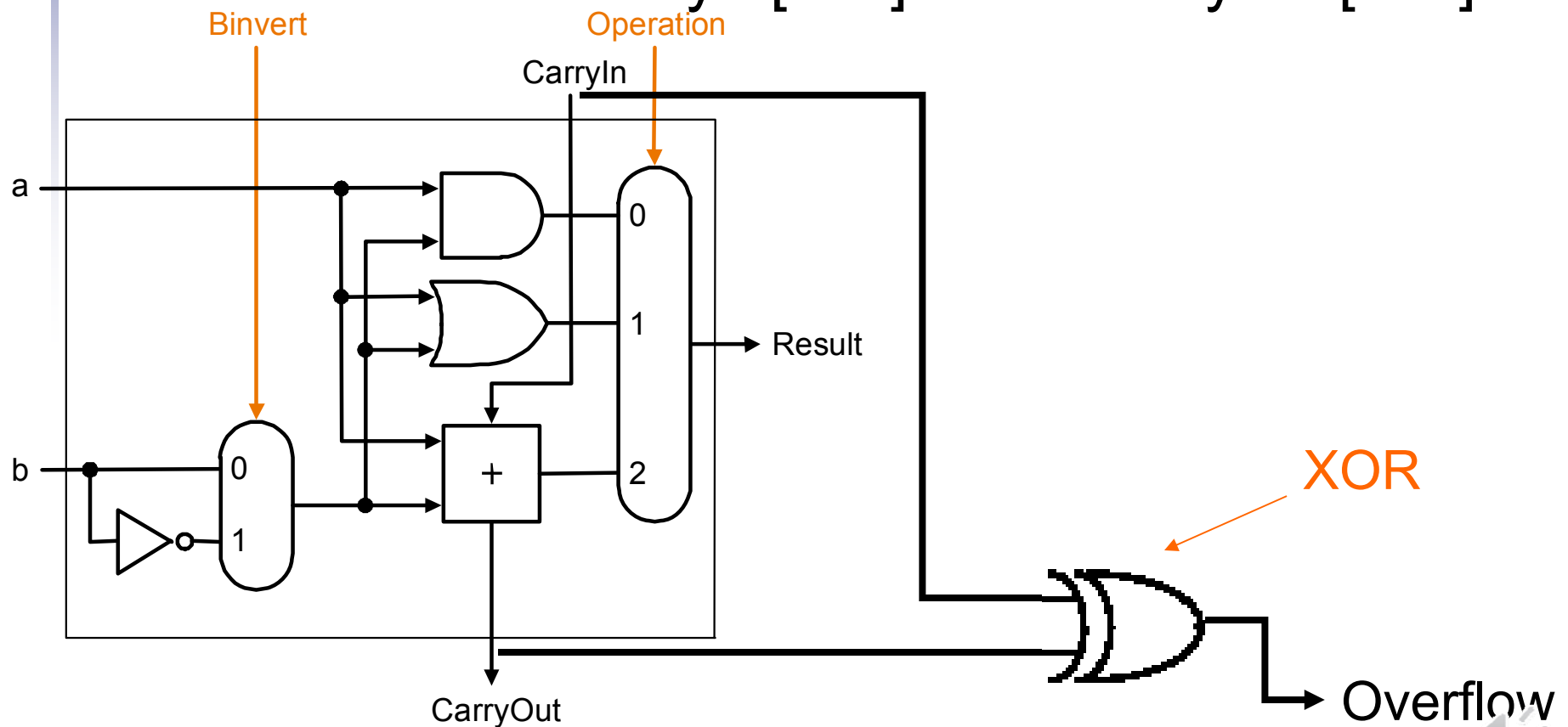
- Expand our 1-bit ALU to include an inverter
  - 2's complement: take inverse of every bit and add 1

Overflow

- For N-bit ALU
  - Overflow = CarryIn[N-1] XOR CarryOut[N-1]

Most significant (N-1) bit ALU

Zero Detection

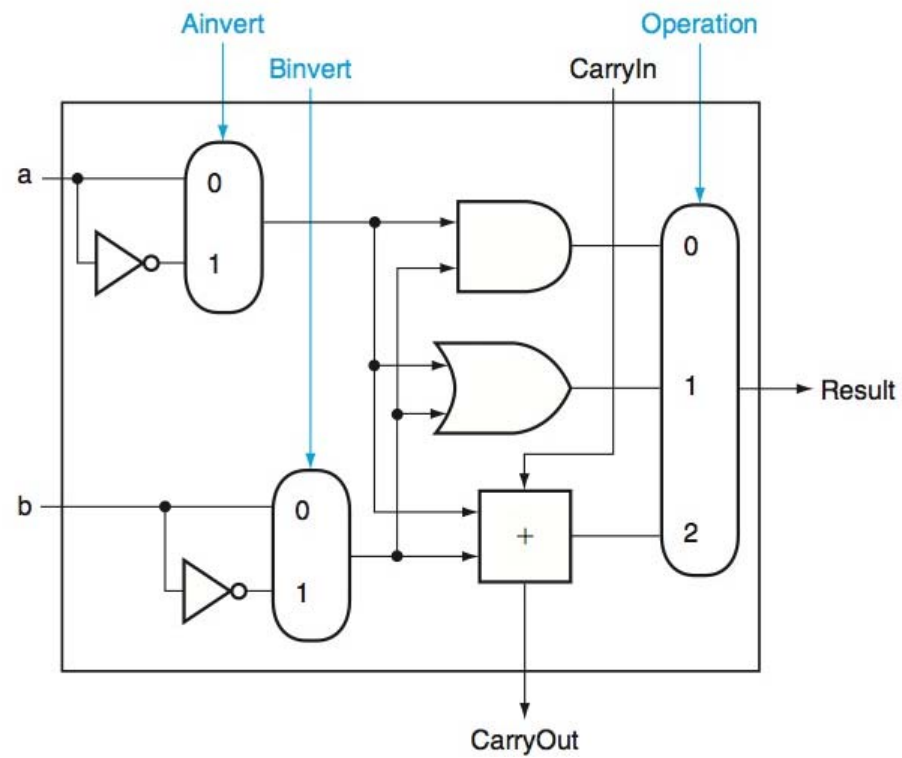- Conditional Branches

- One big NOR gate

- Zero = $(Result_{N-1}+Result_{N-2}+....Result_1+Result_0)$

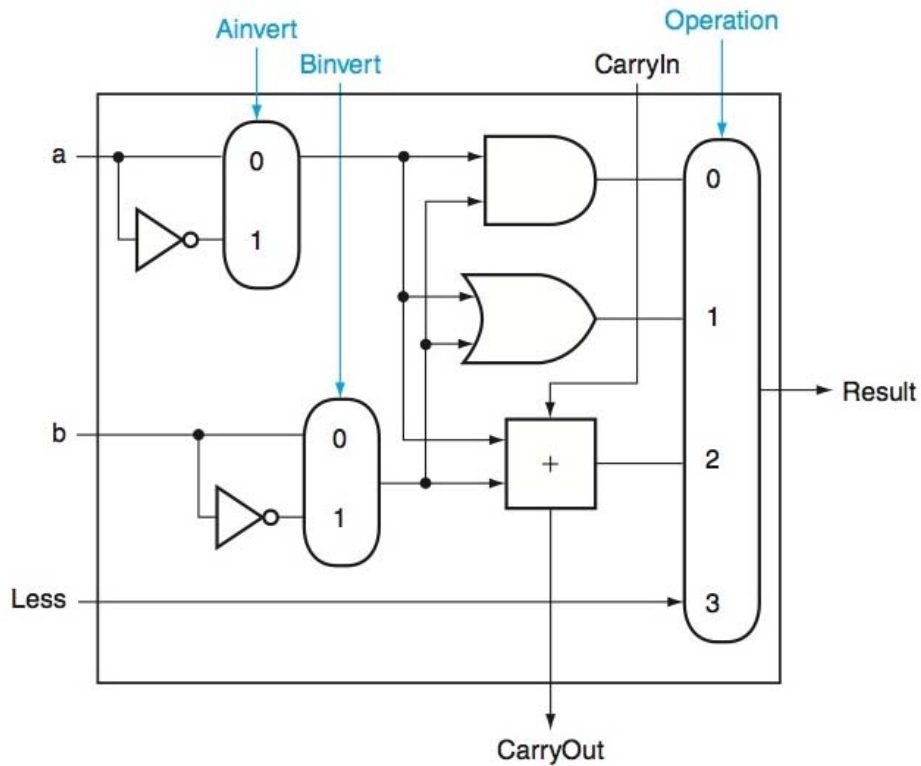- Any non-zero result will cause zero detection output to be zero
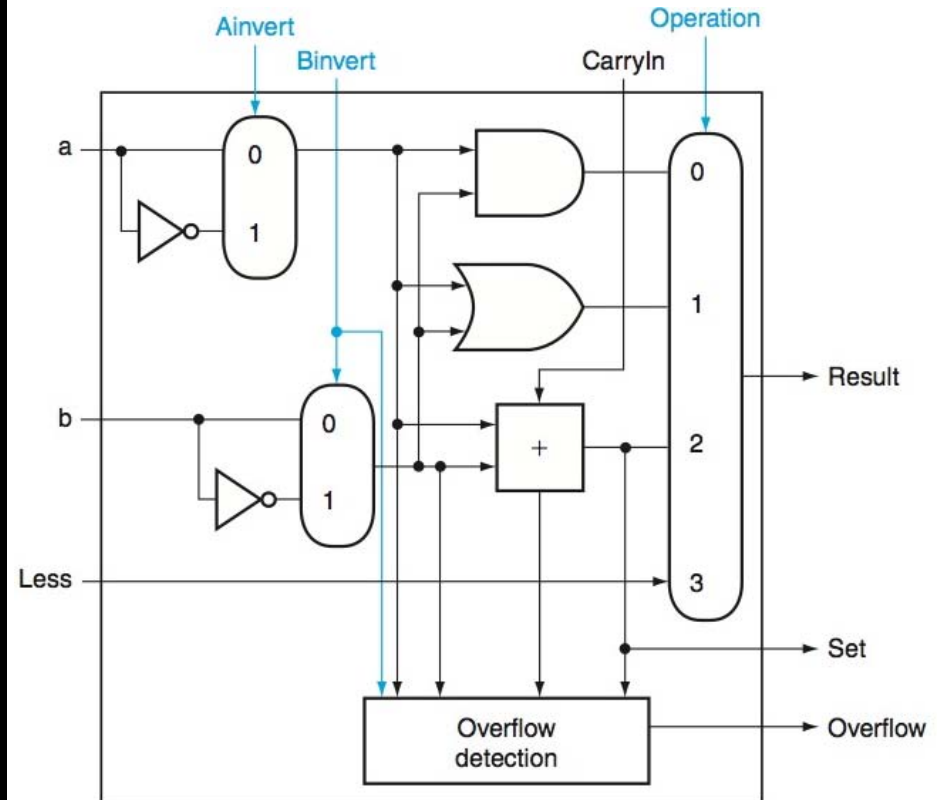
NOR

Set-On-Less-Than (SLT)

- SLT produces a 1 if rs < rt, and 0 otherwise
  - all but least significant bit will be 0
  - how do we set the least significant bit?
  - can we use subtraction?
    - rs - rt < 0
  - set the least significant bit to the sign-bit of (rs - rt)
- New input: LESS
- New output: SET

SLT Implementation

All but MSB

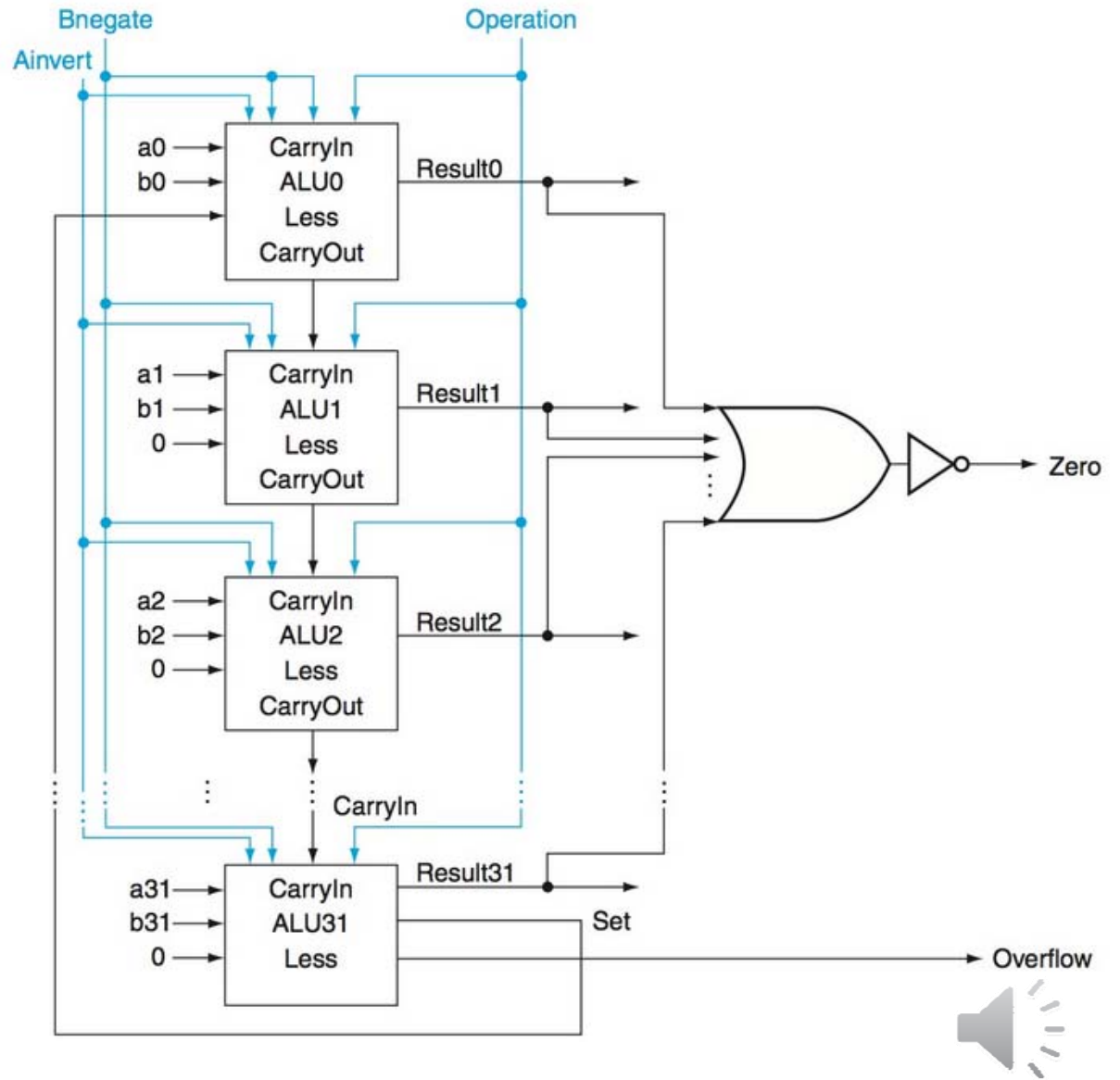Most Significant Bit

SLT Implementation

- Set of MSB is connected to Less of LSB!

Final ALU

- You should feel comfortable identifying what signals accomplish:
  - add
  - sub
  - and
  - or
  - nor
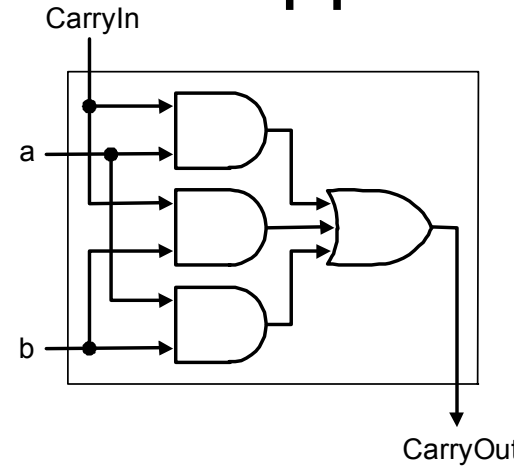 - slt

Chapter 3

Arithmetic for Computers

Can We Make a Faster Adder?

- Worst case delay for N-bit Ripple Carry Adder

  - 2N gate delays
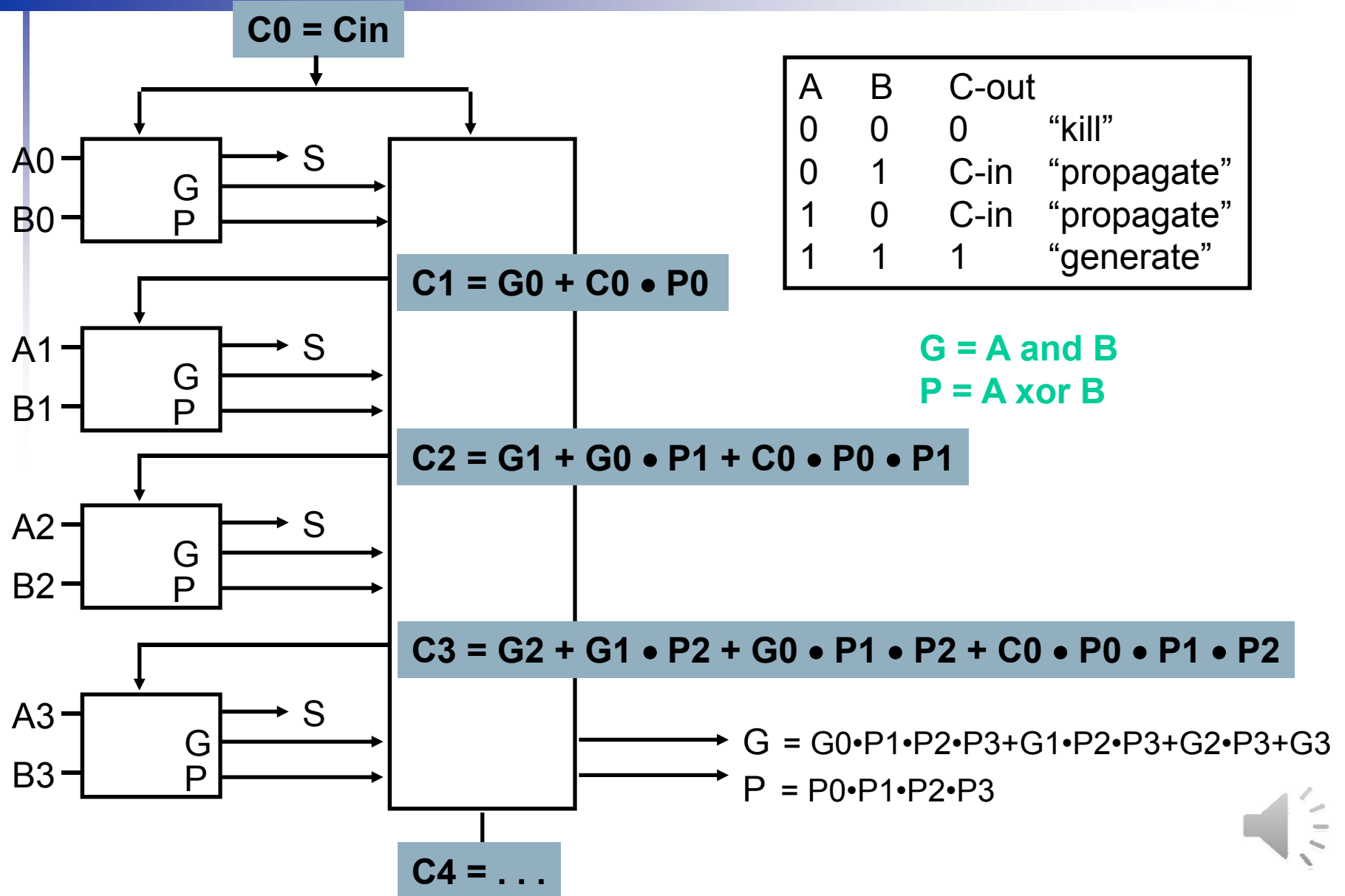 - 2 gates per CarryOut
  - N CarryOuts

- We will explore the Carry Lookahead Adder

  - Generate - Bit i creates new Carry
    - $g_i = A_i$ & B_i

Carry Look Ahead

C0 = Cin

A	B	C-out	
0	0	0	"kill"
0	1	C-in	"propagate"
1	0	C-in	"propagate"
1	1	1	"generate"

A0 — G P — S

$C1 = G0 + C0 \bullet P0$

A1 — G P — S

G = A and B
P = A xor B

$C2 = G1 + G0 \bullet P1 + C0 \bullet P0 \bullet P1$

A2 — G P — S

$C3 = G2 + G1 \bullet P2 + G0 \bullet P1 \bullet P2 + C0 \bullet P0 \bullet P1 \bullet P2$

A3 — G P — S

$G = G0 \bullet P1 \bullet P2 \bullet P3 + G1 \bullet P2 \bullet P3 + G2 \bullet P3 + G3$
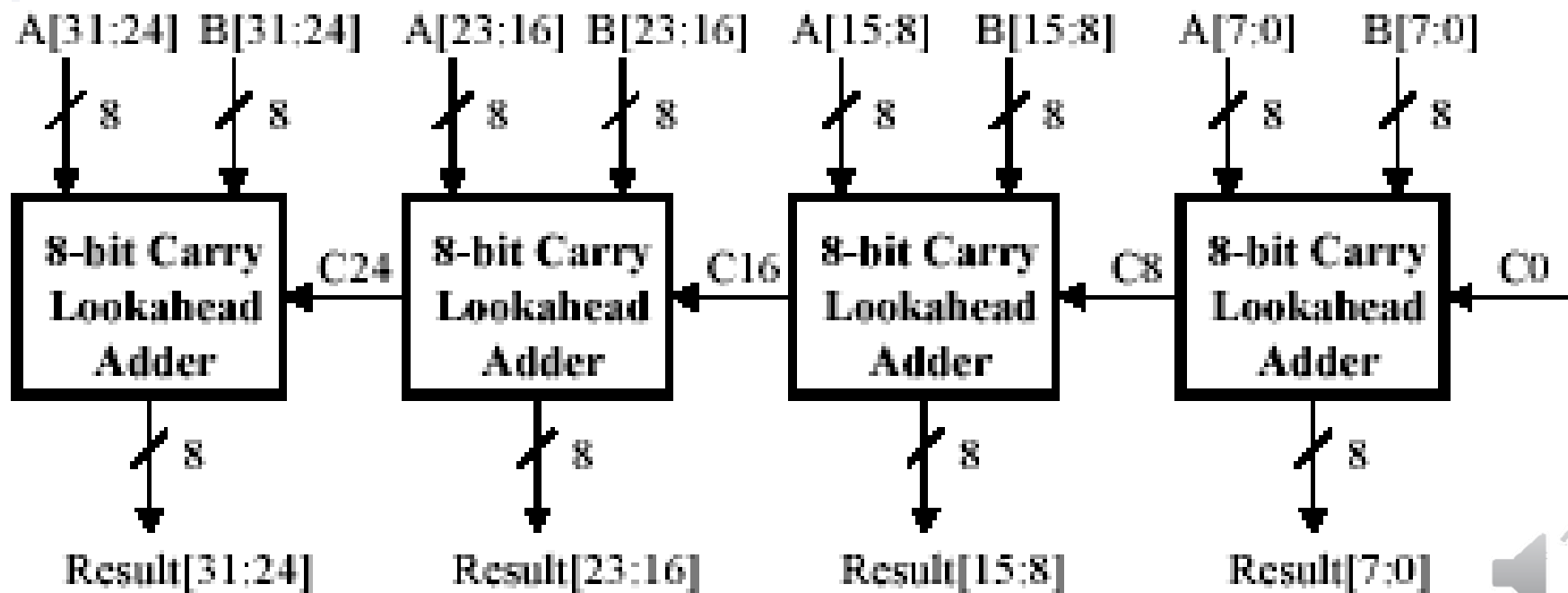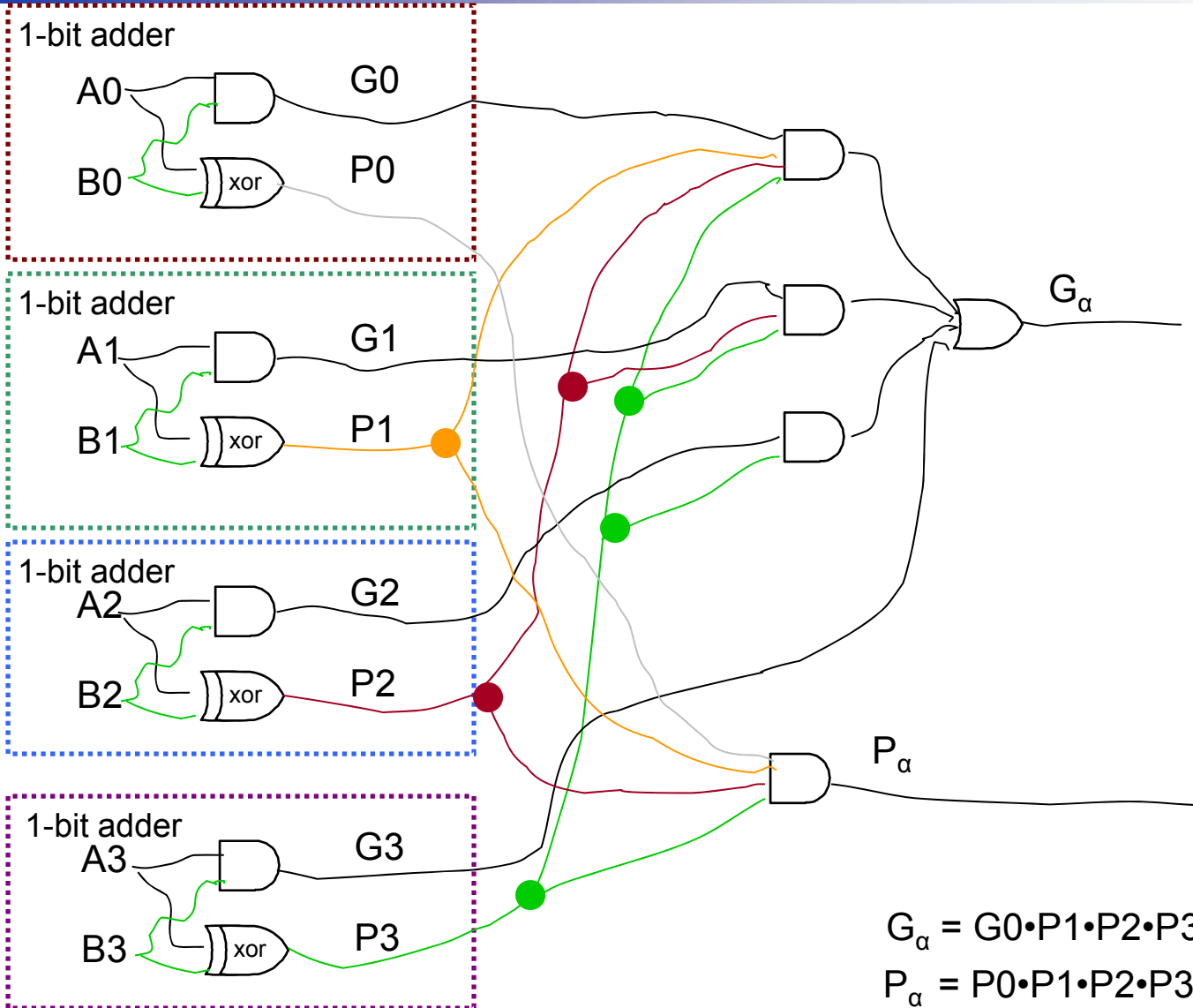$P = P0 \bullet P1 \bullet P2 \bullet P3$

C4 = . . .

Partial Carry Lookahead Adder

- Connect several N-bit Lookahead Adders together

- Four 8-bit carry lookahead adders can form a 32-bit partial carry lookahead adder

A[31:24] B[31:24] A[23:16] B[23:16] A[15:8] B[15:8] A[7:0] B[7:0]

/ 8 / 8   / 8 / 8   / 8 / 8   / 8 / 8

| 8-bit Carry Lookahead Adder | C24 | 8-bit Carry Lookahead Adder | C16 | 8-bit Carry Lookahead Adder | C8 | 8-bit Carry Lookahead Adder | C0 |

/ 8   / 8   / 8   / 8

Result[31:24]   Result[23:16]   Result[15:8]   Result[7:0]

Generate and Propagate

G_α = G0•P1•P2•P3+G1•P2•P3+G2•P3+G3

P_α = P0•P1•P2•P3

Hierarchical CLA

α

CLA

C0

G_α
P_α

C4 = G_α + C0 • P_α

β

4-bit Adder

C8 = G_β + G_α • P_β + C0 • P_α • P_β

γ

4-bit Adder

C12 = G_γ + G_β • P_γ + G_α • P_β • P_γ + C0 • P_α • P_β • P_γ

δ

4-bit Adder

C16 = . . .

Generate and Propagate

Carry Select Adder