ECE 2300
Digital Logic & Computer Organization

Spring 2025

More Binary Arithmetic
ALU

Cornell University

Announcements

- **Supplementary notes on timing analysis posted on course web**

Thu 3/6	**11: Timing Analysis** [slides] [supplementary notes]

- **HW 4 due tomorrow**

- **HW 5 will be released soon**

Review: Overflow in 2's C

- When performing fixed-size addition in 2's C, overflow occurs if
  - Signs of both operands are the same, and
  - Sign of sum is different

- Another test (easy to do in hardware)
  - Carry into MSB does not equal carry out

Did Overflow Occur?

carries → 1 1 1

```
  011100
+ 010101
--------
  110001
```

YES

carries → 1 1 1 1

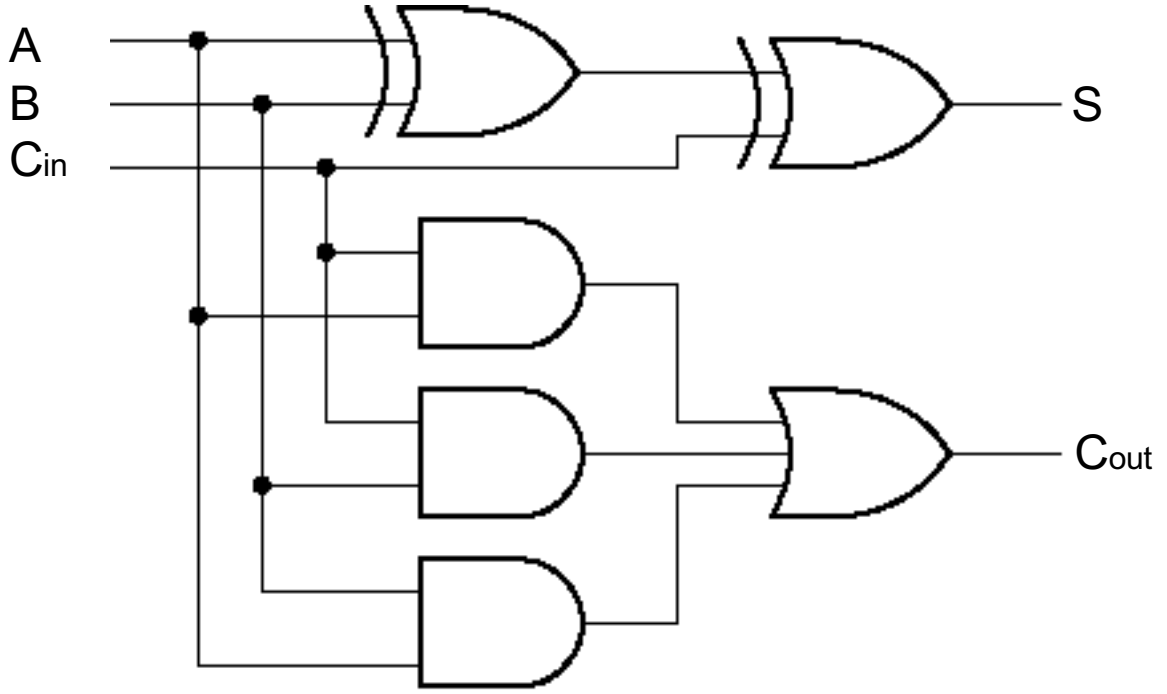
```
  111101
+ 010100
--------
  010001
```

NO

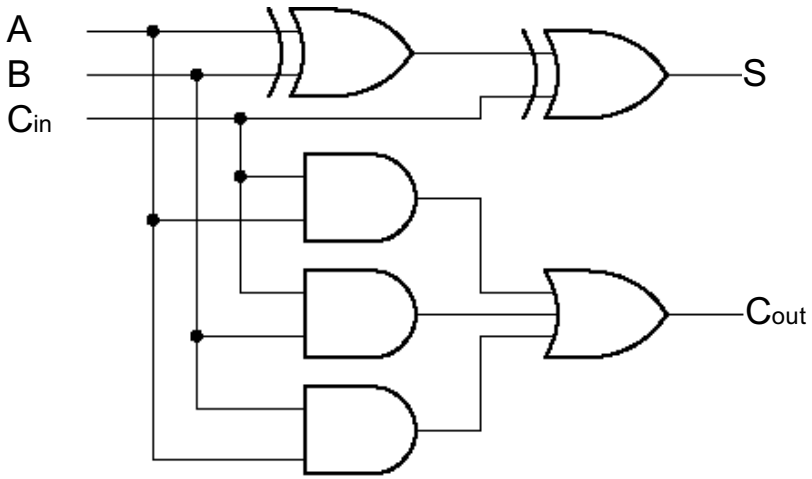
Full Adder (1 bit Adder with Carry)

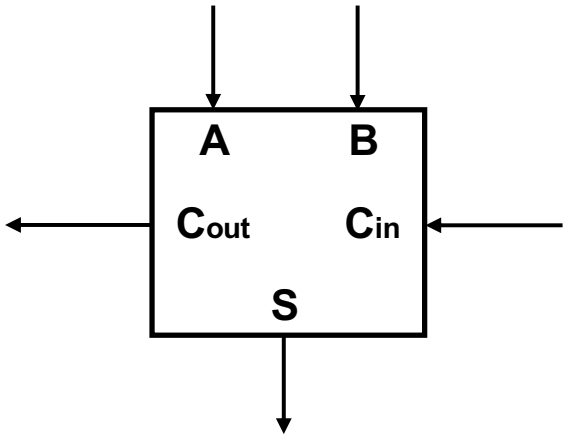
- Inputs: A, B and C_{in} (carry-in)
- Outputs: S (sum) and C_{out} (carry-out)

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

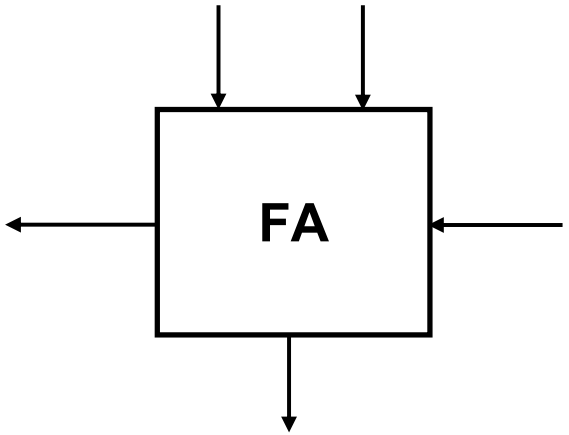


Full-Adder Circuit Symbol

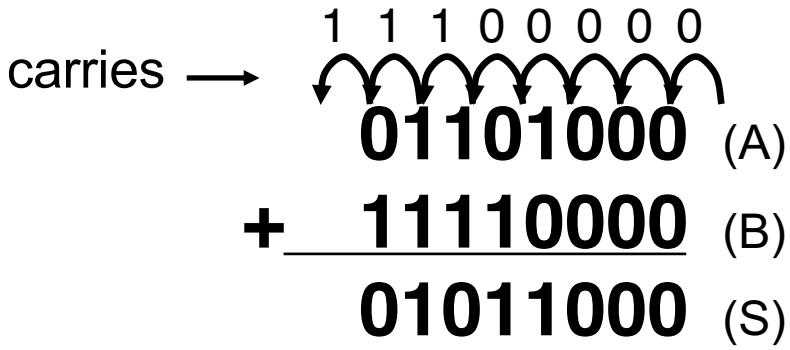




OR



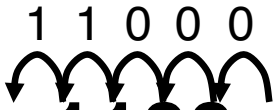
Building a Multi-Bit Binary Adder



- **Inputs & outputs for the i^{th} bit position**
  - **Inputs: A_i, B_i and C_i (carry-in)**
  - **Outputs: S_i (sum) and C_{i+1} (carry-out)**

- **Carry-out of a bit position is the carry-in for next bit position**

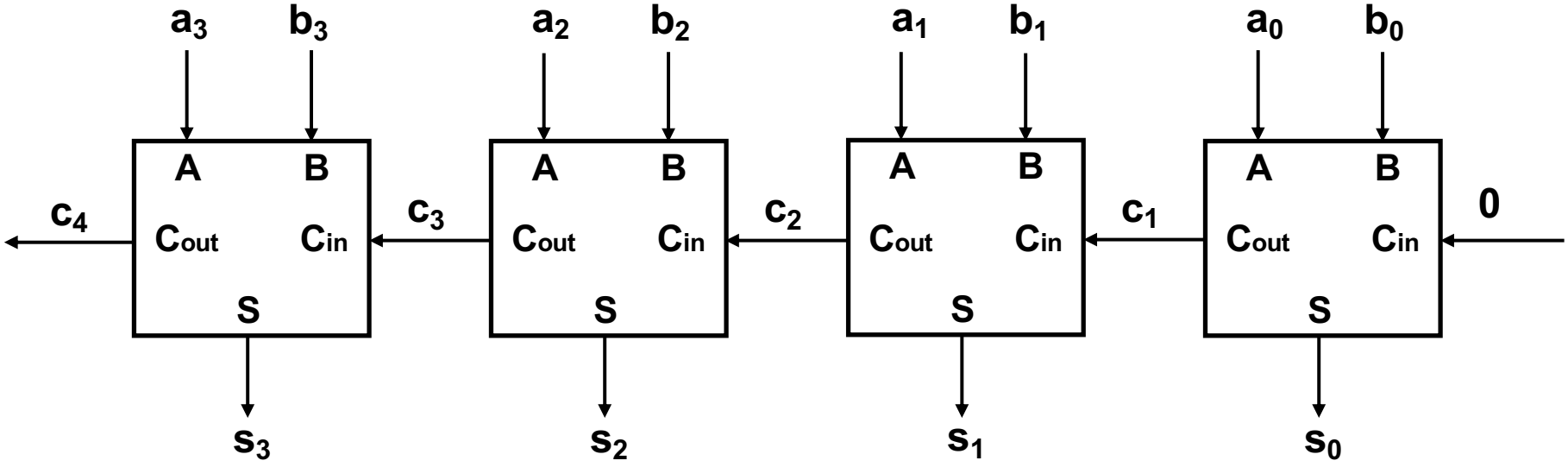
Four Bit Adder



1100

\+ **0110**

0010



Ripple Carry Adder (RCA)

Two's Complement Subtraction

- **Negate second operand and add**

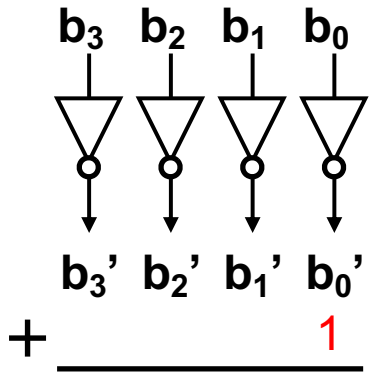
01101000 (104)
- 00010001 (17)

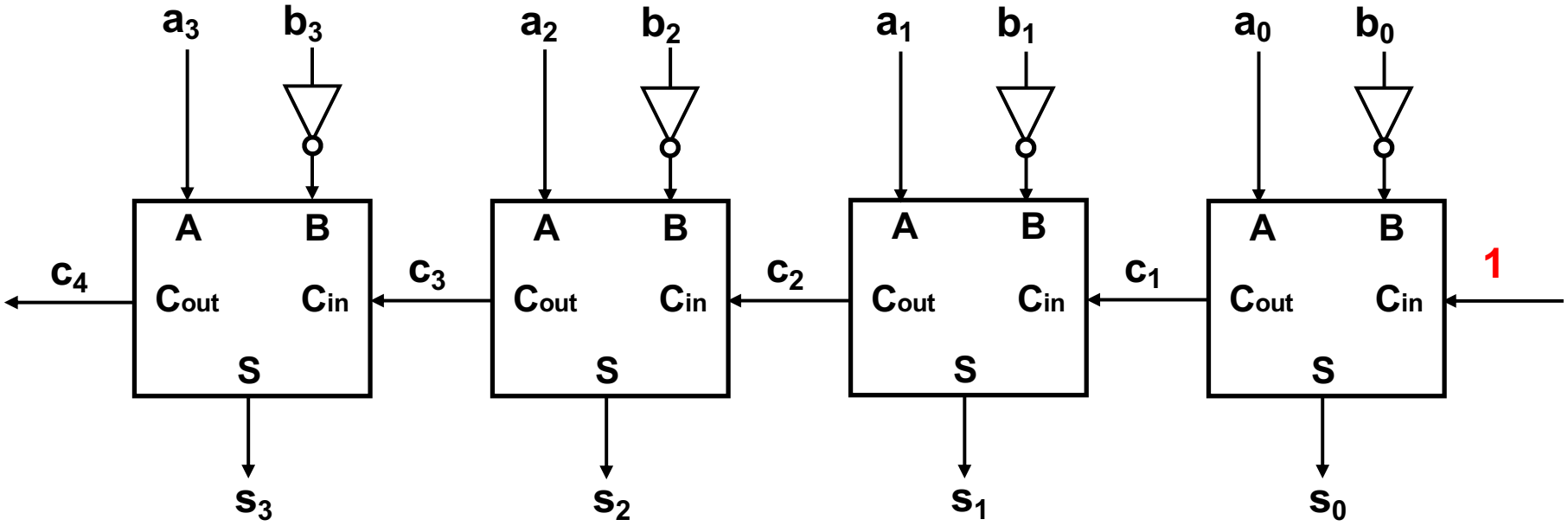
01101000 (104)
+ 11101111 (-17)
01010111 (87)

Four Bit Subtractor

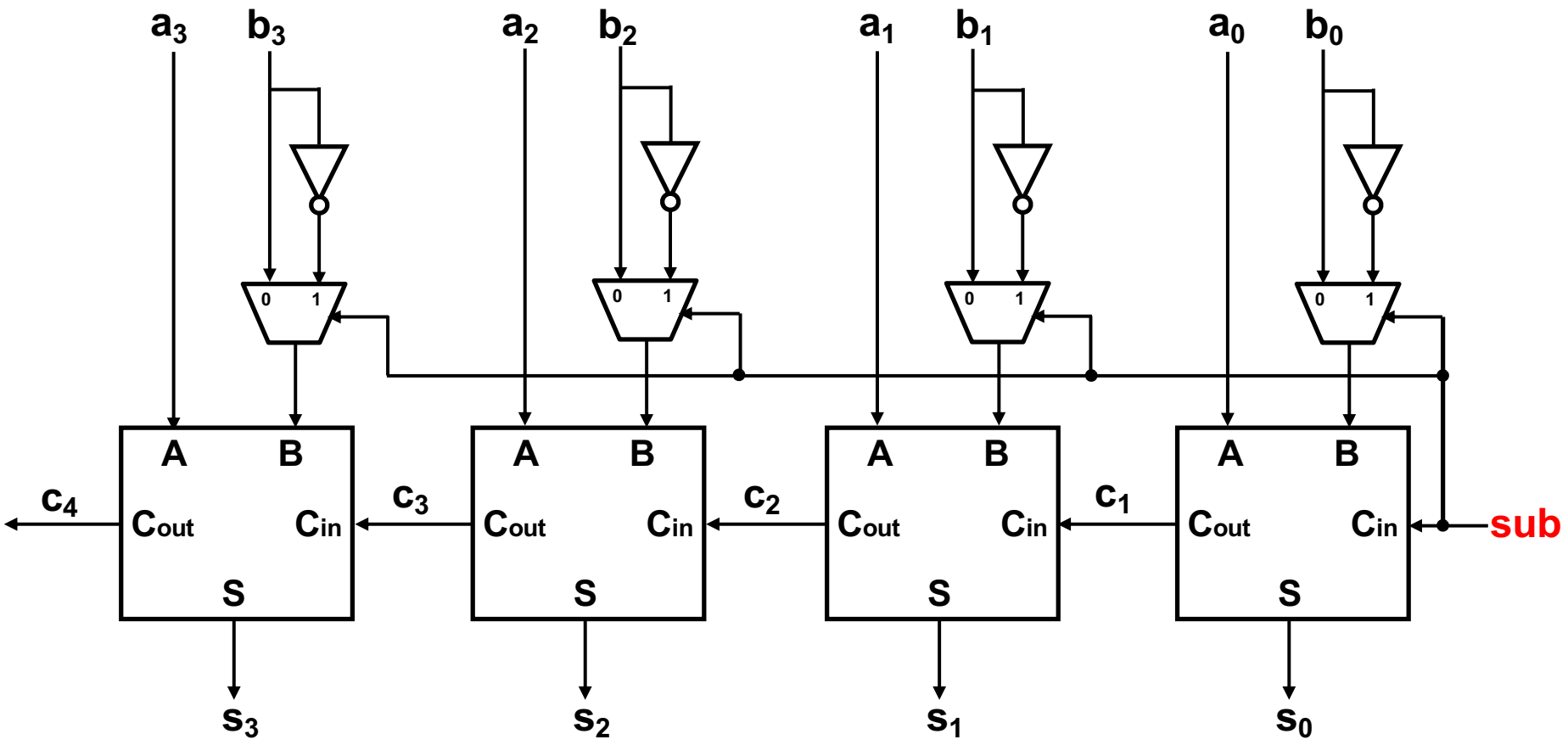
negating the second operand

-b = (b'+1)





Four Bit Adder / Subtractor



"sub" is a 1-bit control signal that selects the operation mode of the adder/subtractor

Adder Implementations

- Many different adder implementations exist, which differ in speed and circuit complexity
- Ripple carry adder is simple but slow
  - Each 1-bit full adder must wait for the carry bit to be calculated from the previous full adder
- Common techniques to speed up carry propagation
  - Carry lookahead
  - Carry select
  - Carry save
  - etc.

Carry Lookahead Adder (CLA)

- **Calculation of carry out of MSB is slow for large RCAs**
  - **Carry has to propagate from LSB to the MSB, which forms the longest path (critical path)**

- **CLA adder calculates groups of carries in parallel**

Carry Generation and Propagation

- C_{out} = 1 from a bit position happens for 2 reasons
  - Carry is *generated* from this bit position
  - Carry in to this position is *propagated* to the next

Carry Generation and Propagation

- Carry from i^{th} position: c_{i+1}
  - namely, the carry into $(i+1)^{th}$ position

- Carry generation function for the i^{th} position

  $G_i = a_i \cdot b_i$ `(if position i generates a carry)`

- Carry propagation function for the i^{th} position

  $P_i = a_i + b_i$ `(if position i can propagate a carry)`

- Carry out of the i^{th} position

  $c_{i+1} = G_i + P_i \cdot c_i$

Carry Out Equations for 4-bit Adder

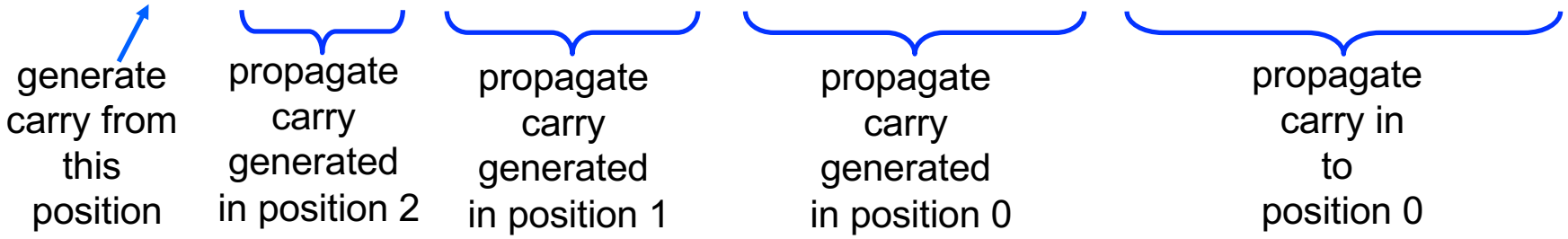
$$G_i = a_i \cdot b_i$$
$$P_i = a_i + b_i$$

$$c_1 = G_0 + P_0 \cdot c_0$$

$$c_2 = G_1 + P_1 \cdot c_1$$
$$= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot c_0$$

$$c_3 = G_2 + P_2 \cdot c_2$$
$$= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot c_0$$

$$c_4 = G_3 + P_3 \cdot c_3$$
$$= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot c_0$$



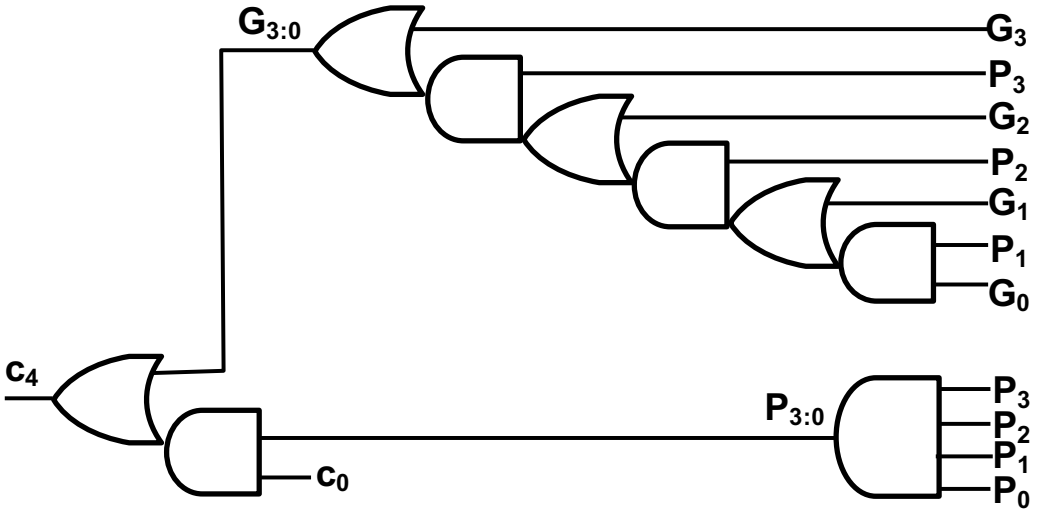
Carry Out Logic for 4-bit Adder

$$G_i = a_i \cdot b_i$$
$$P_i = a_i + b_i$$

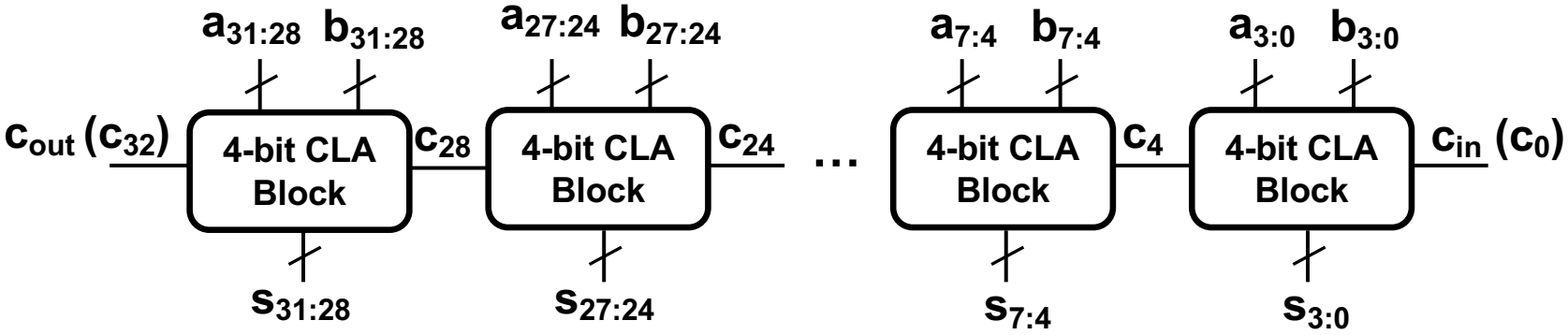
$$c_4 = G_3 + P_3 \cdot c_3$$
$$= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot c_0$$
$$= \underbrace{G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0))}_{G_{3:0}} + \underbrace{(P_3 \cdot P_2 \cdot P_1 \cdot P_0)}_{P_{3:0}} \cdot c_0$$

$G_{3:0}$ – Carry generation from bits 3-0

$P_{3:0}$ – Carry propagation through bits 3-0

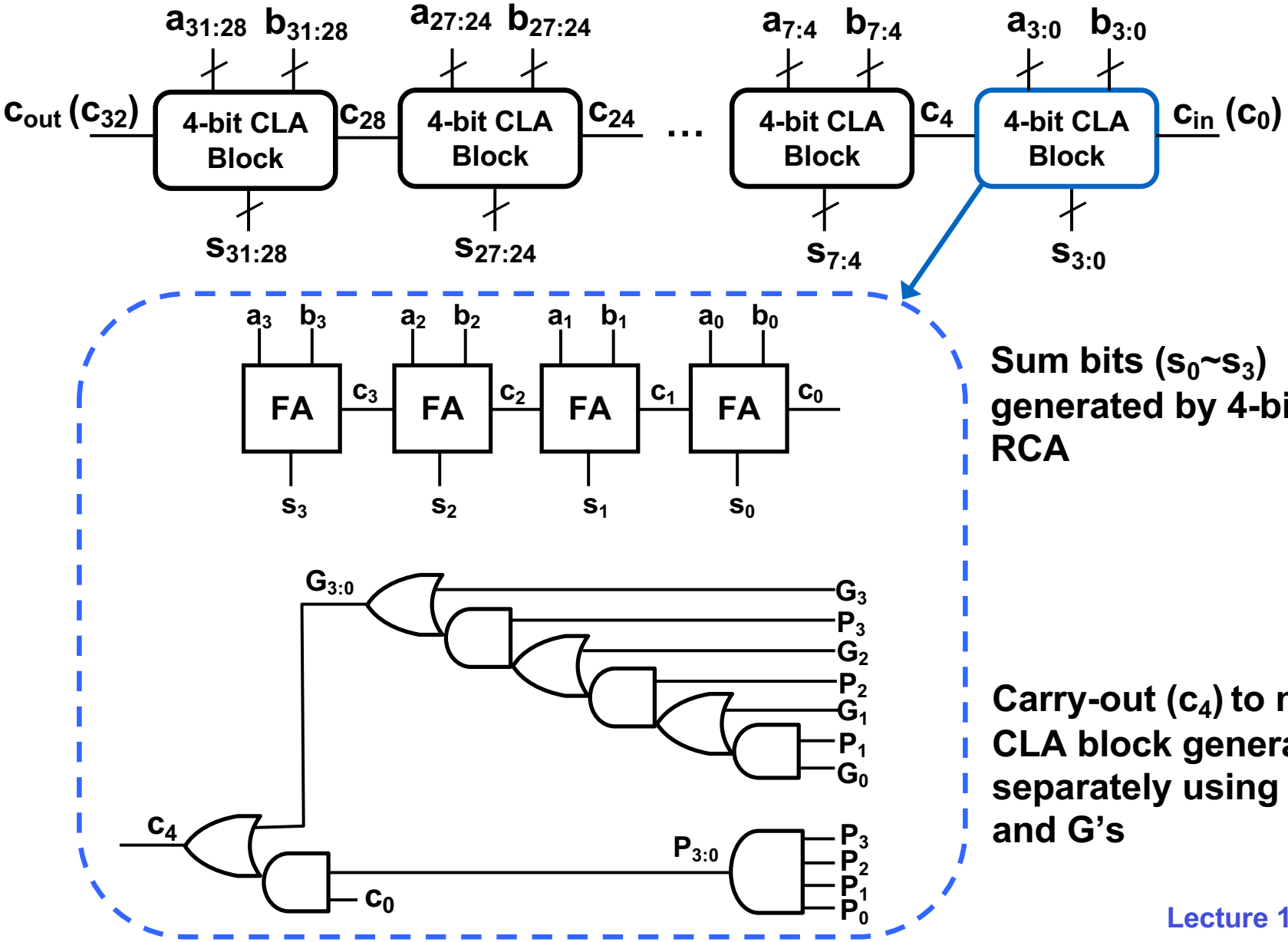


32-bit CLA with 4-bit RCAs



- Carry out logic gets more complicated beyond 4 bits
- CLAs are often implemented as 4-bit modules and instantiated in a hierarchical way to realize wider adders

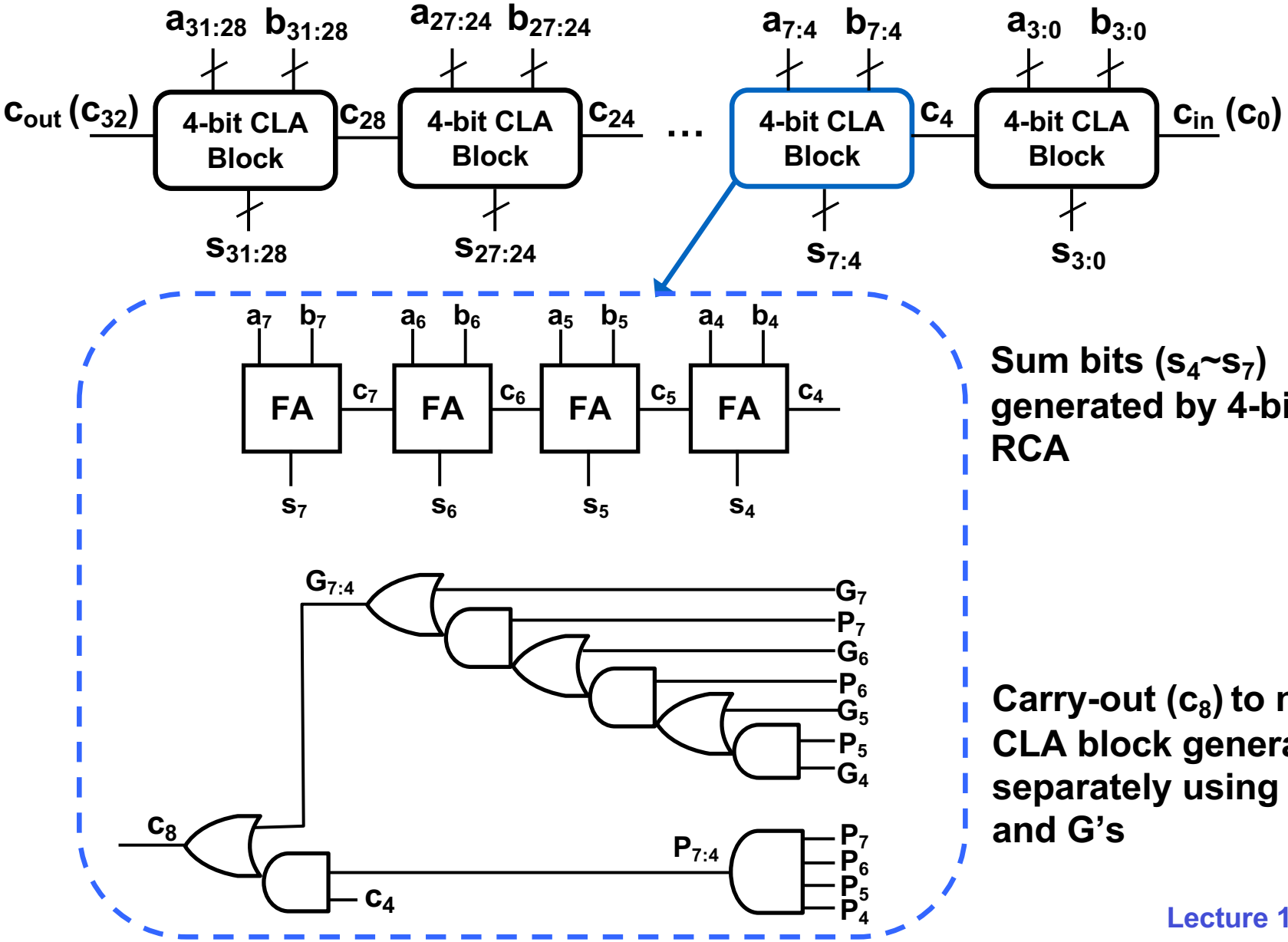
32-bit CLA with 4-bit RCAs



Sum bits (s_0~s_3) generated by 4-bit RCA

Carry-out (c_4) to next CLA block generated separately using P's and G's

32-bit CLA with 4-bit RCAs

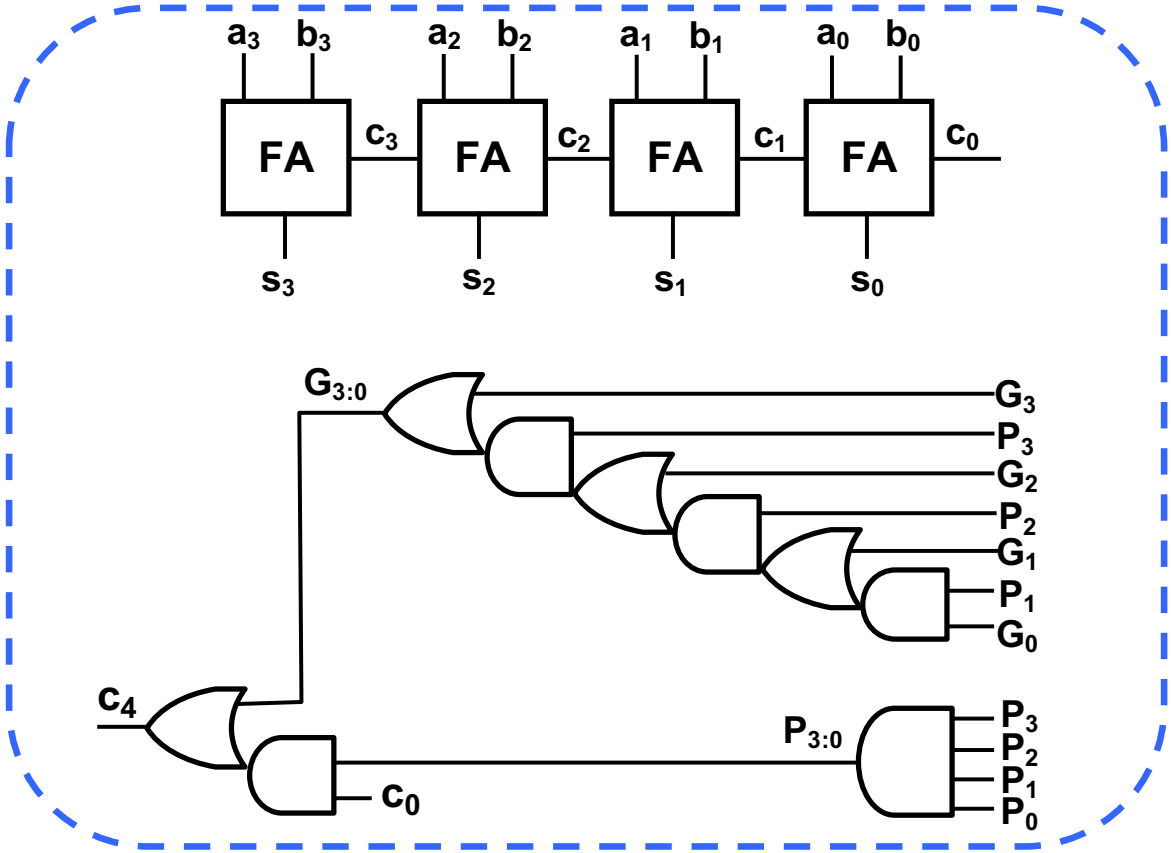


Sum bits (s_4~s_7) generated by 4-bit RCA

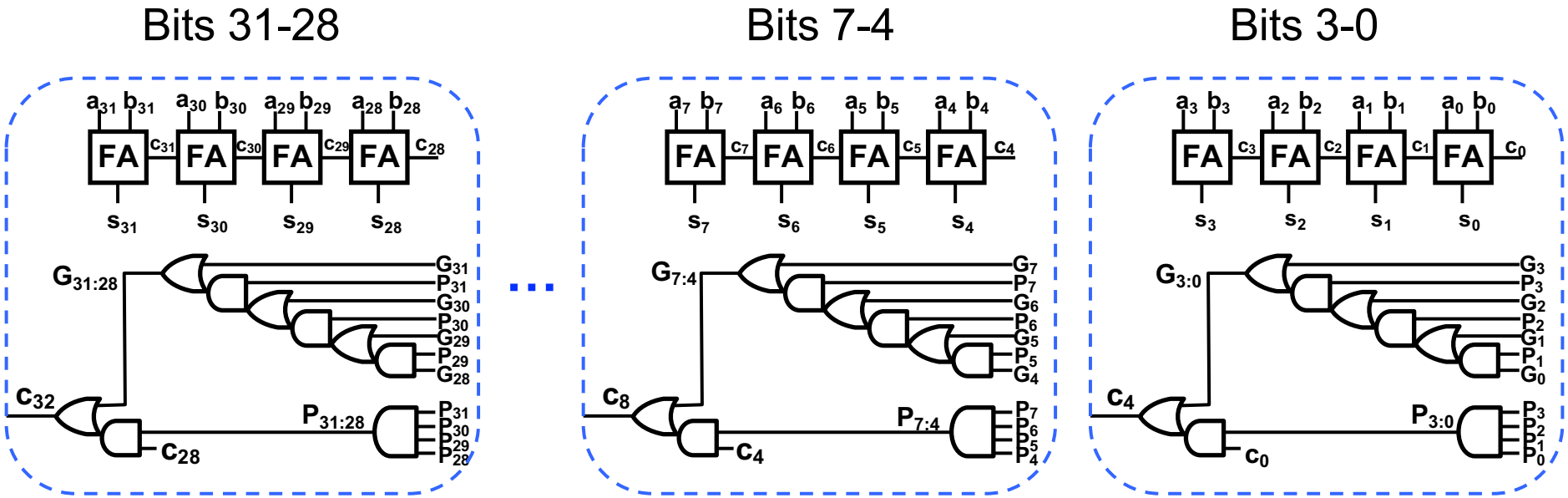
Carry-out (c_8) to next CLA block generated separately using P's and G's

Discussion: Delay of a 4-bit CLA Block

- **Assumptions**: Each logic gate has a delay of 1ns, and each full adder (FA) takes 3ns. Calculating Gs and Ps requires 1ns.
- Analyze the longest path delays for outputs s_3 and c_4, respectively.



Analyzing the Critical Path (Longest Delay)

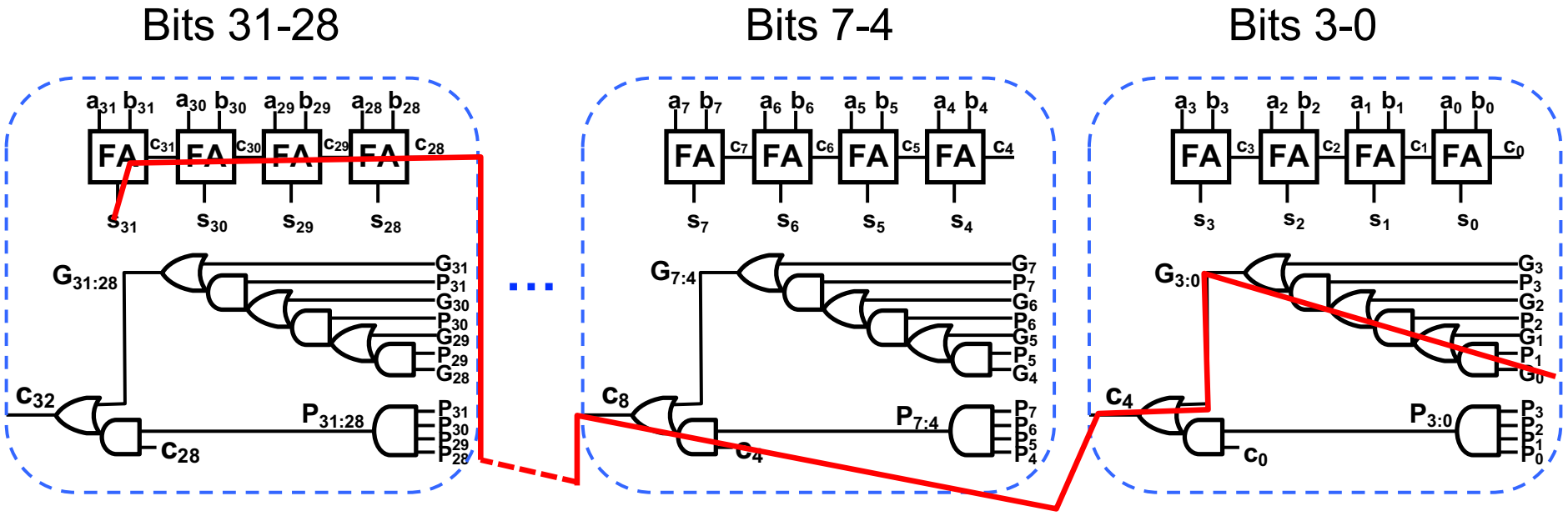


Carry generation and propagation functions (e.g., $G_{7:4}$, $P_{7:4}$) in a 4-bit CLA block do not depend on results from other CLA blocks

⇩

P's and G's for all CLA blocks are generated *in parallel*

Critical Path of 32-bit CLA



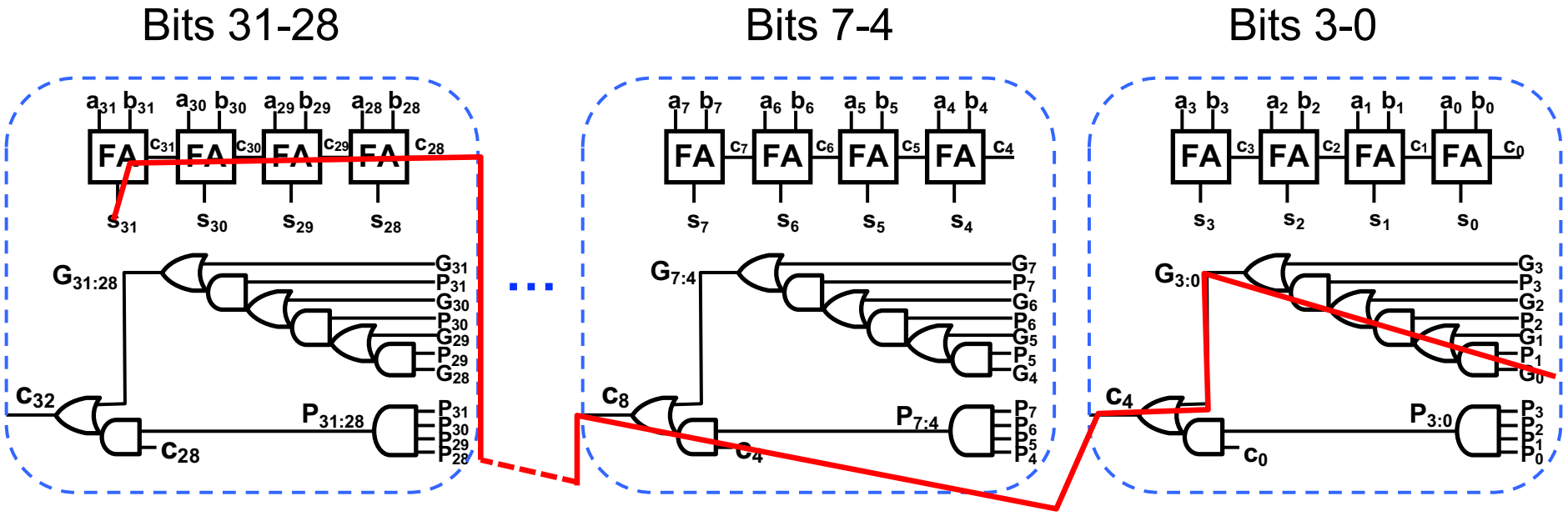
Carry generation and propagation functions (e.g., $G_{7:4}$, $P_{7:4}$) in a 4-bit CLA block do not depend on results from other CLA blocks

⇩

P's and G's for all CLA blocks are generated *in parallel*

Discussion: Critical Path Delay of 32-bit CLA

- **Assumptions: Each logic gate has a delay of 1ns, and each full adder (FA) takes 3ns. Calculating Gs and Ps requires 1ns.**

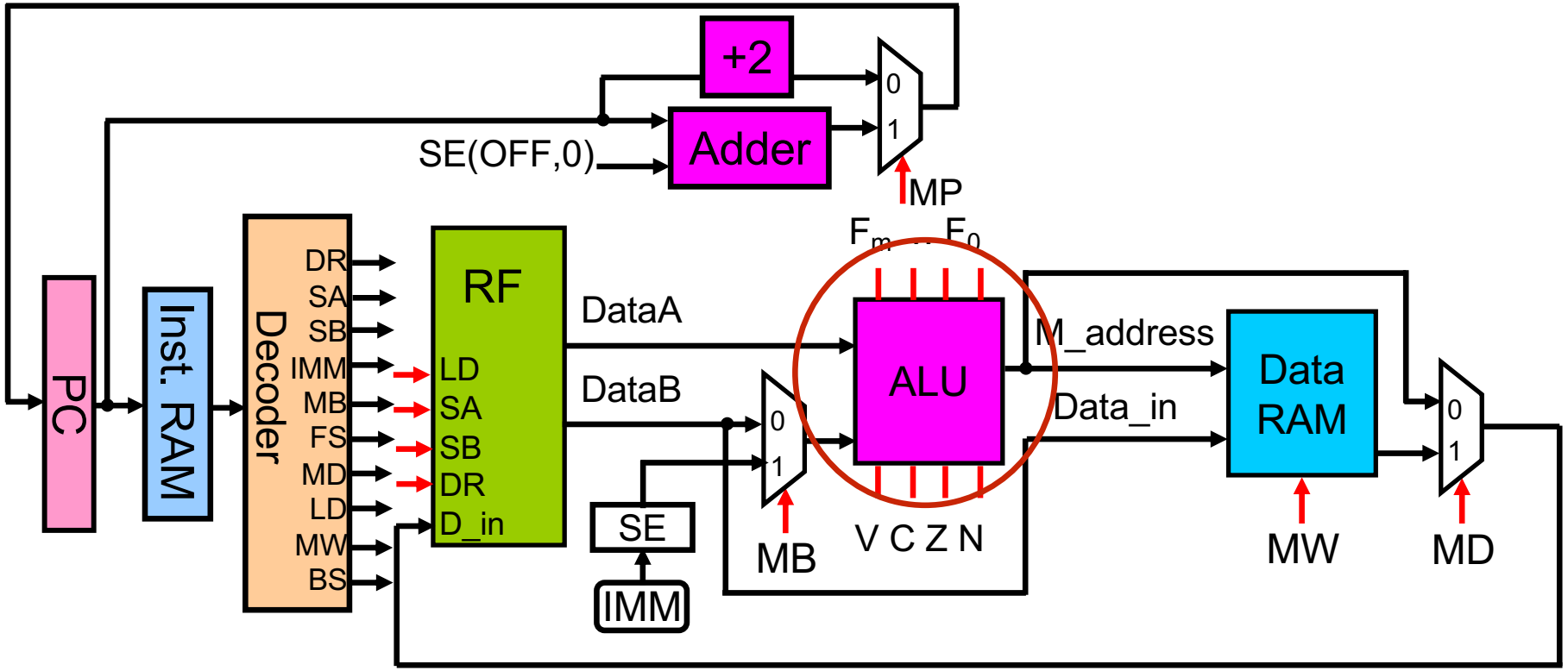


c_4 stabilizes at 8ns, c_8 at 8+2, c_{12} at 8+2*2, c_{16} at 8+2*3, …

→ c_{28} stabilizes at 8 + 2*6 = 20ns

→ s_{31} depends on c_{28}; it stabilizes at 20 + 4*$Delay_{FA}$ = 20 + 4*3 = 32ns

Our Microprocessor is Built to Compute



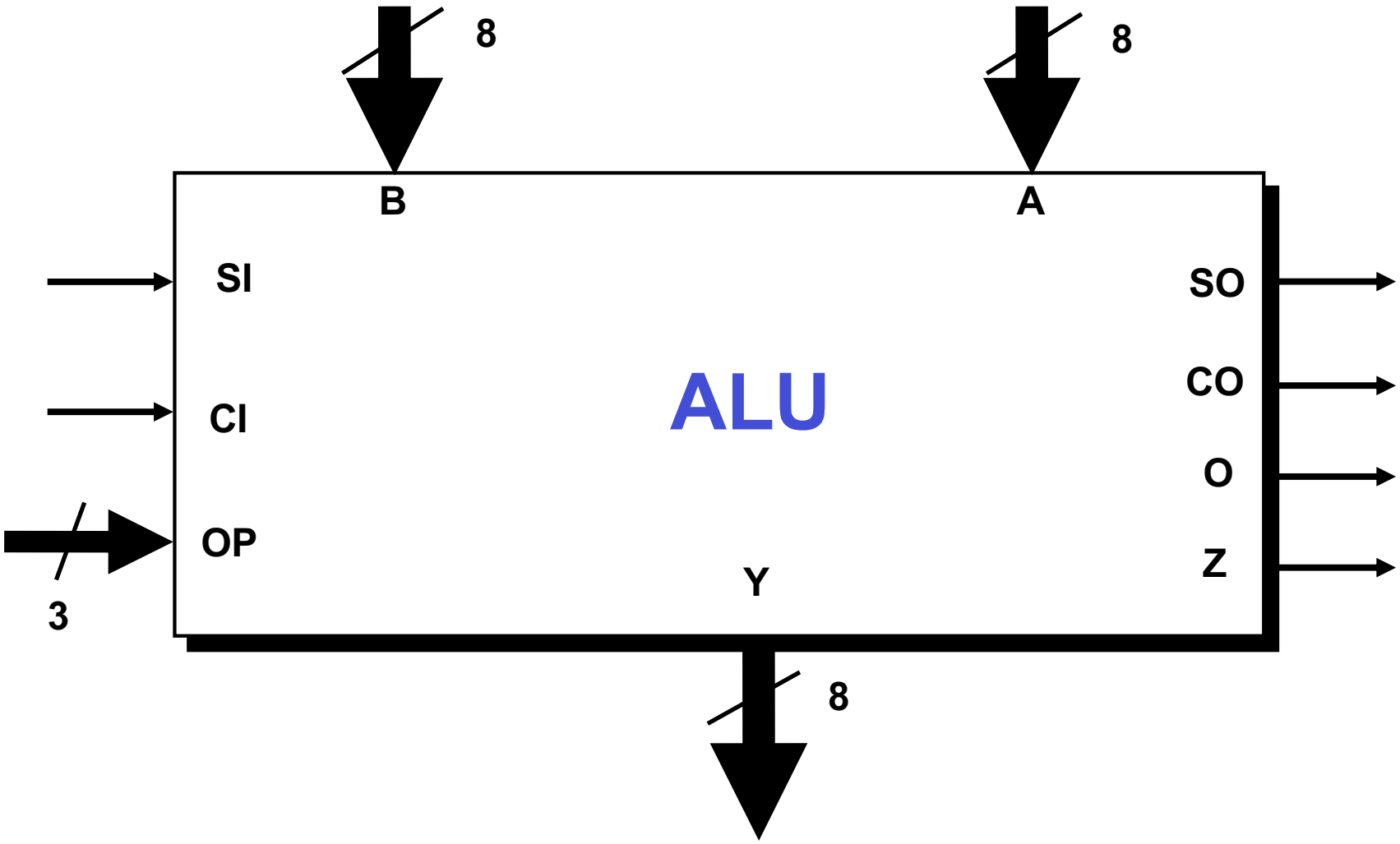
ALU is the computing core of a processor

What is an ALU?

- Arithmetic Logic Unit (ALU): Combinational logic circuit that combines a variety of operations into a single unit

- Common operations may include
  - Addition and subtraction
  - Logical (OR, AND)
  - Shift
  - Comparisons

A Simple 8-Bit ALU



ALU Operations, Inputs & Outputs

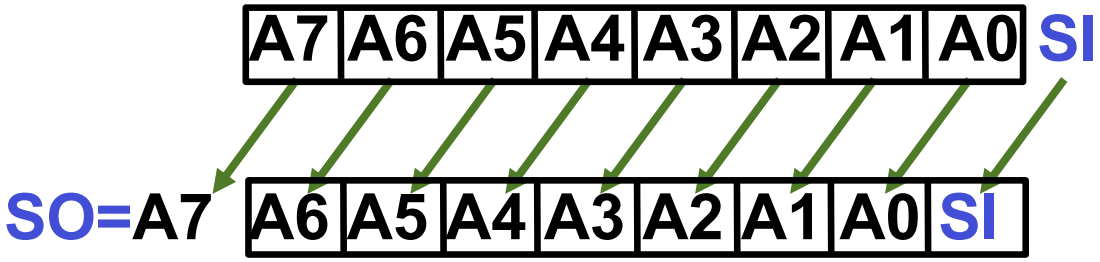
- **Operations**
  - **Addition and Subtraction** (and Comparison)
  - **Bitwise AND and OR**
  - **Left Shift and Right Shift**

- **Inputs**
  - **A, B, Carry In (CI)**
  - **Shift In (SI)**
  - **Code indicating operation to be performed (OP)**

- **Outputs**
  - **Y, Carry Out (CO)**
  - **Shift Out (SO)**
  - **Flags regarding the result of the operation**
    - Overflow flag (O), Zero flag (Z)

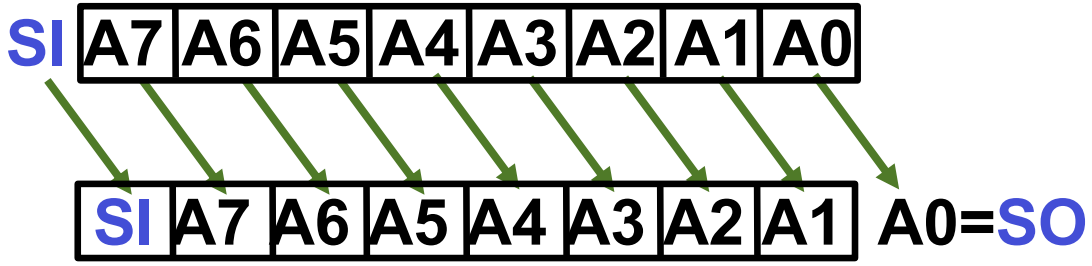
Shift Operations

- **Left Shift: shifts each bit left by 1 position**



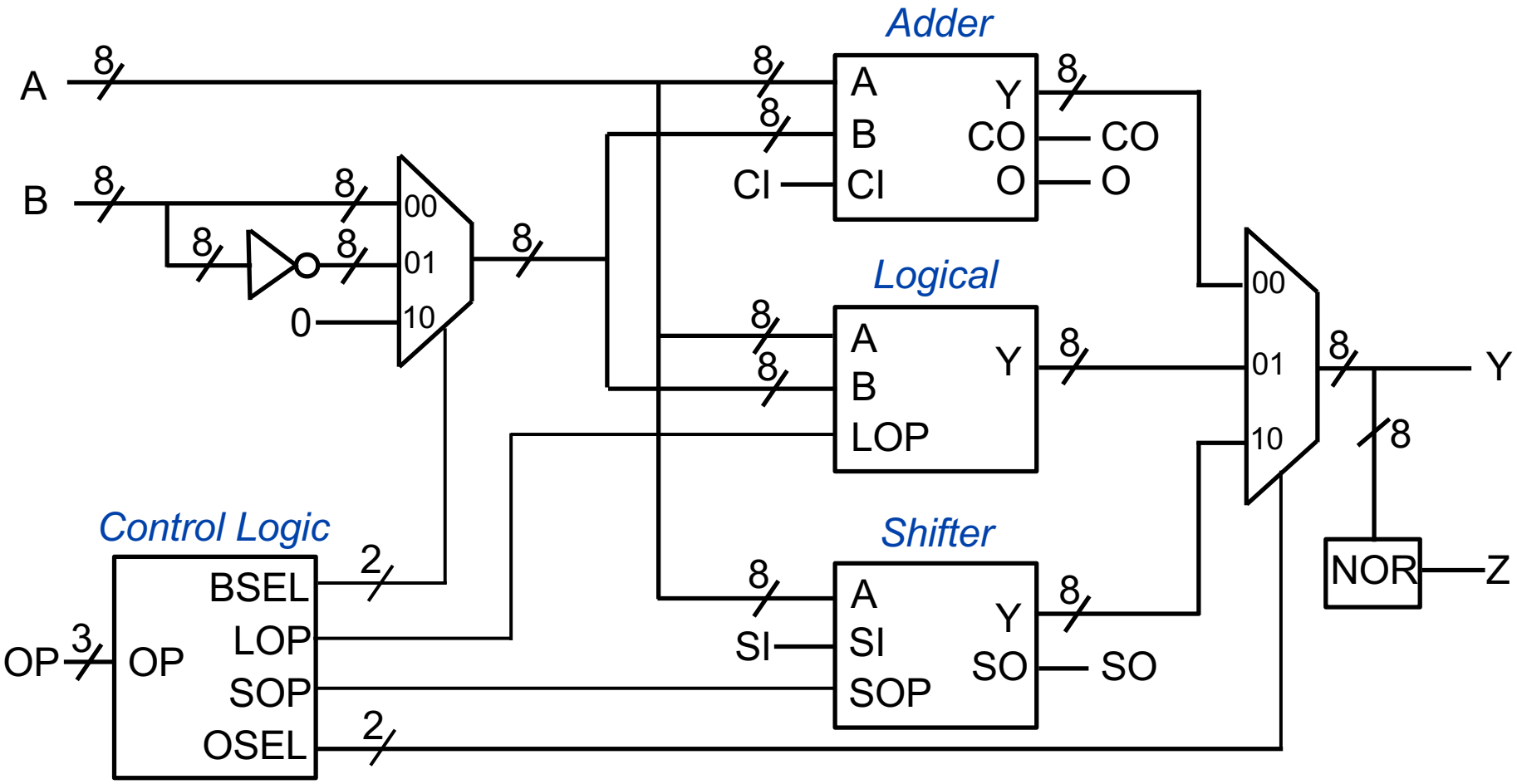
  - MSB shifted into SO, SI shifted into LSB

- **Right Shift: shifts each bit right by 1 position**



  - LSB shifted into SO, SI shifted into MSB

ALU Block Diagram

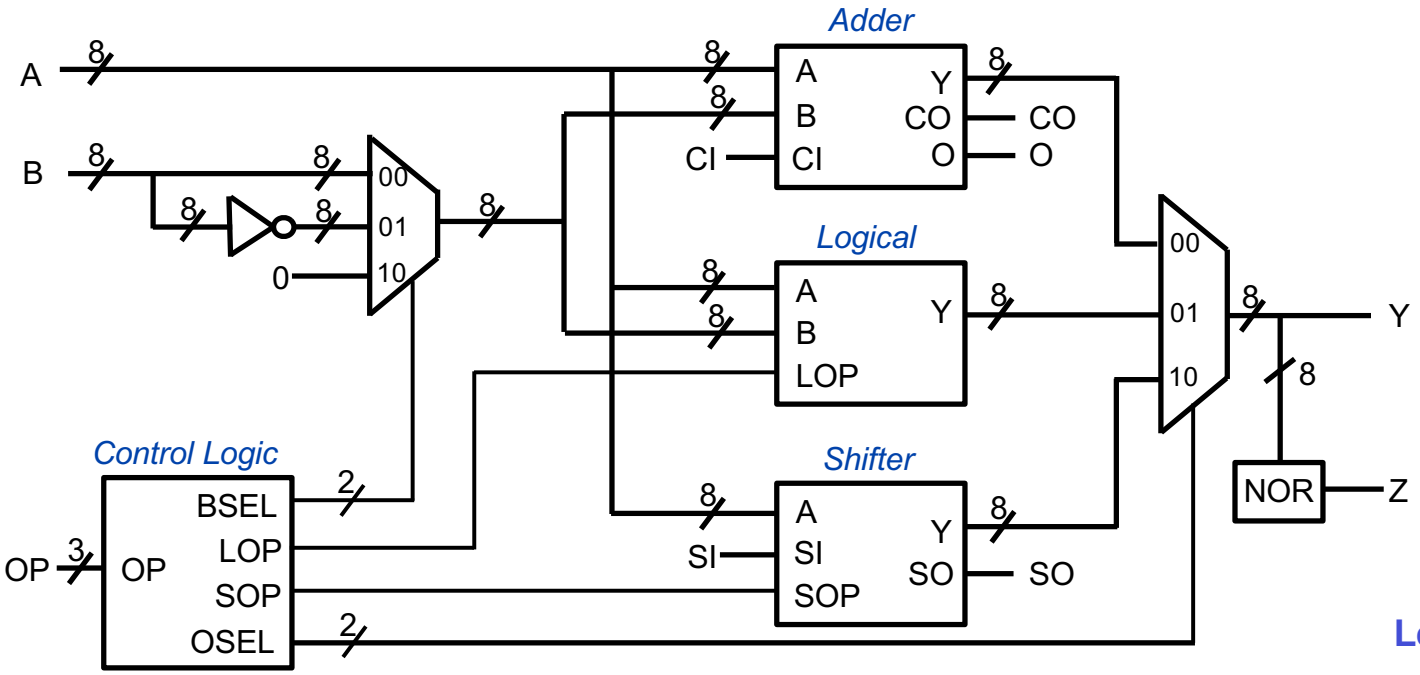


Control logic outputs

- **BSEL: B input mux select**
- **LOP: Logical unit opcode**
- **SOP: Shifter opcode**
- **OSEL: Output mux select**

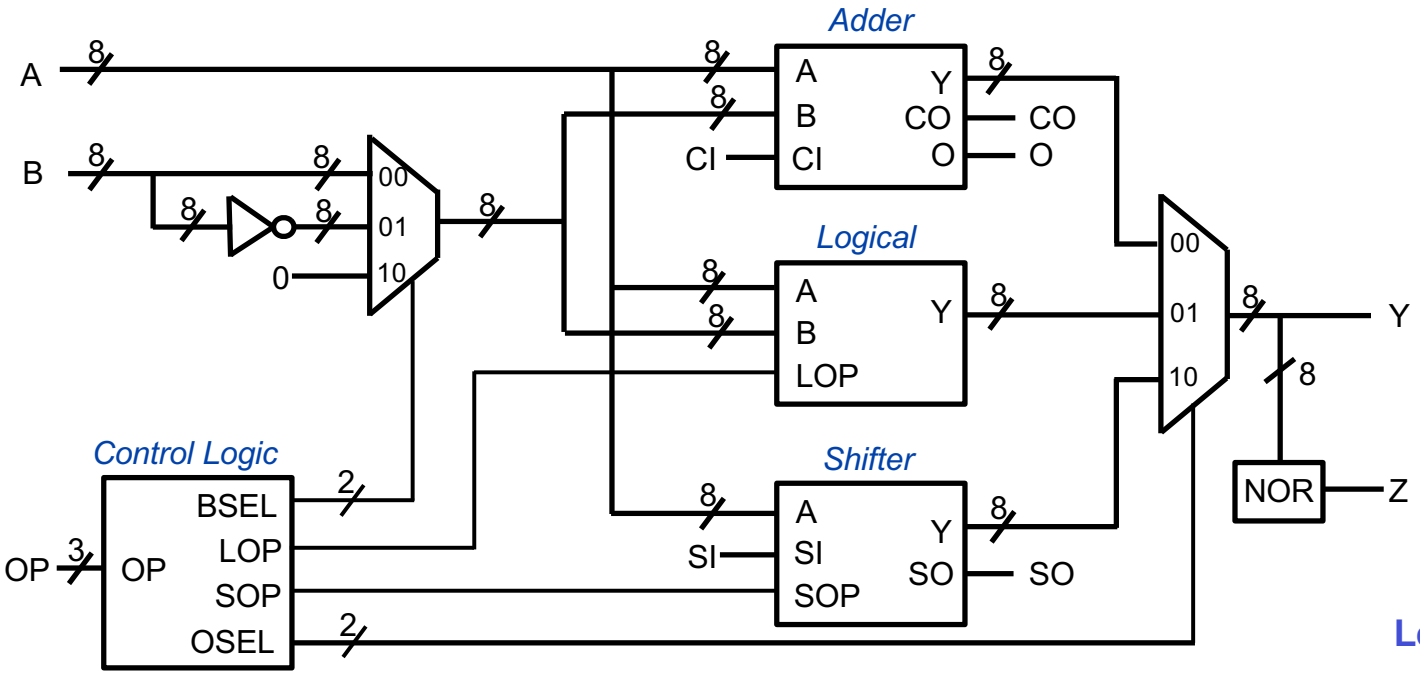
ALU Operation Encodings

OP Name	OP	BSEL	CI	LOP	SOP	OSEL	Operation
ADD	000	00	0	-	-	00	Y = A + B + CI
SUB	001	01	1	-	-	00	Y = A + B' + 1
AND	011	00	-	0	-	01	Y = A AND B
OR	100	00	-	1	-	01	Y = A OR B
SHL	101	-	-	-	0	10	Y = A[6..0],SI
SHR	110	-	-	-	1	10	Y = SI,A[7..1]
PASS A	111	10	0	-	-	00	Y = A



Comparison Operations (Byproduct of SUB)

- To compare A and B, perform A − B
  - If the result is 0, then A = B
  - Z flag set to 1 whenever ALU result is 0
    - 8-input NOR returns 1 only if all 8 input bits are 0
  - Can check for A ≥ B and A < B by observing the MSB of the result of A − B



Next Class

Memories
(H&H 5.5)