

# Overview

---

## ◆ Last lecture

- PLDs
- ROMs
- Tristates
- Design examples

## ◆ Today

- Adders
  - ⇒ Ripple-carry
  - ⇒ Carry-lookahead
  - ⇒ Carry-select
- Parallel prefix trees
- The conclusion of combinational logic!!!

# Arithmetic circuits

---

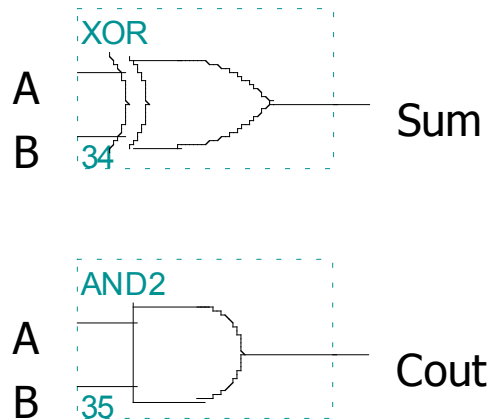
- ◆ General-purpose building blocks
  - Critical components in processor datapaths
    - ⇒ Adders
    - ⇒ Multipliers (integer, floating-point)
    - ⇒ ALUs
  - Perform most computer instructions
  - Time  $\leftrightarrow$  space tradeoff
    - ⇒ Fast circuits usually require more logic

# Binary half adder

## ◆ 1-bit half adder

- Computes sum, carry-out
  - ⇒ No carry-in
- $\text{Sum} = A'B + AB' = A \text{ xor } B$
- $\text{Cout} = AB$

A	B	S	C <sub>out</sub>
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

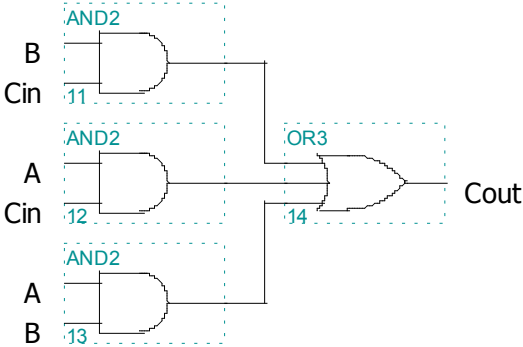
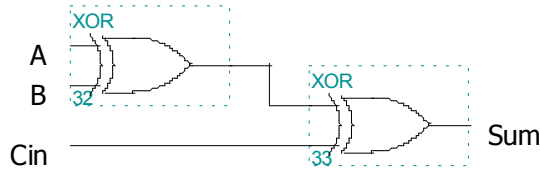


# Binary full adder

◆ 1-bit full adder

- Computes sum, carry-out
  - ↳ Carry-in allows cascaded adders
- Sum = Cin xor A xor B
- Cout = ACin + BCin + AB

A	B	C <sub>in</sub>	S	C <sub>out</sub>
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: Alternative implementation

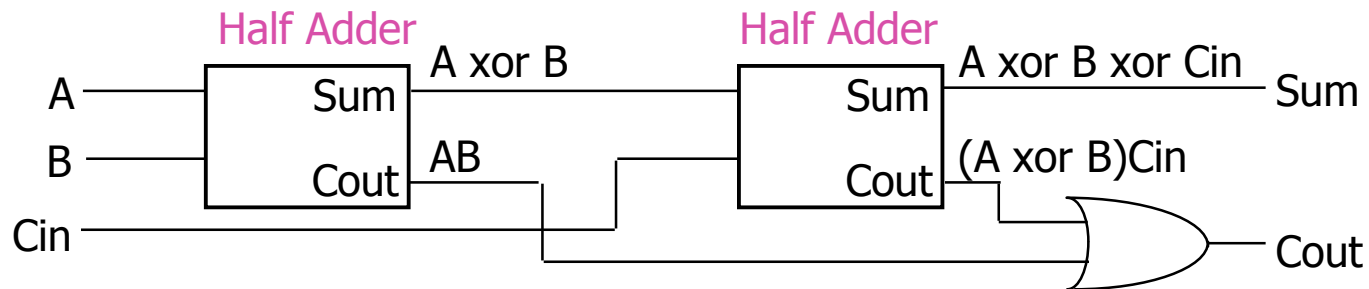
## ◆ Multilevel logic

- Slower
- Fewer gates
  - ⇒ 2 XORs, 2 ANDs, 1 OR

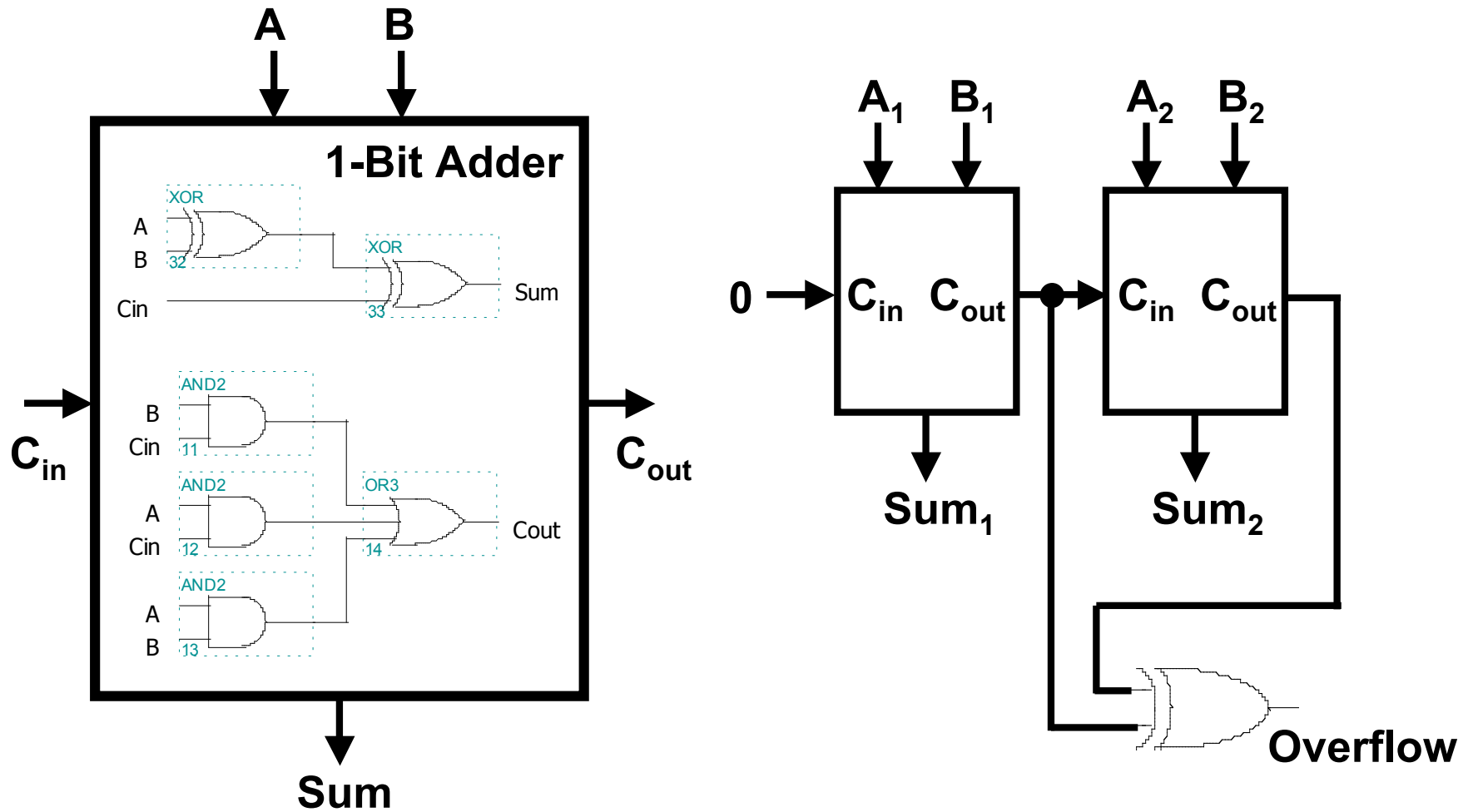
$$\text{Sum} = (A \oplus B) \oplus C_{in}$$

$$\begin{aligned} \text{Cout} &= AC_{in} + BC_{in} + AB \\ &= (A \oplus B)C_{in} + AB \end{aligned}$$

A	B	C <sub>in</sub>	S	C <sub>out</sub>	C <sub>out</sub>
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	1



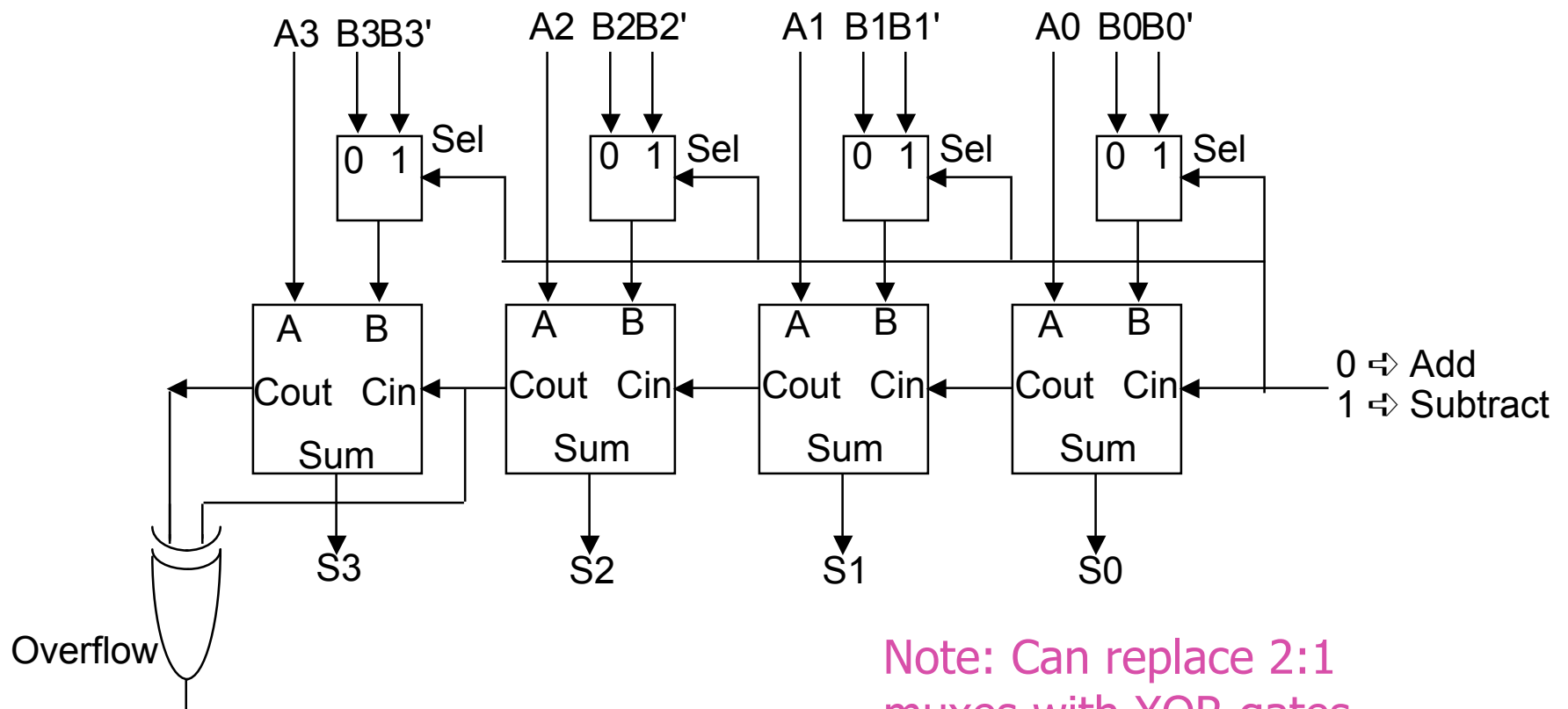
# 2-bit ripple-carry adder



# 4-bit ripple-carry adder/subtractor

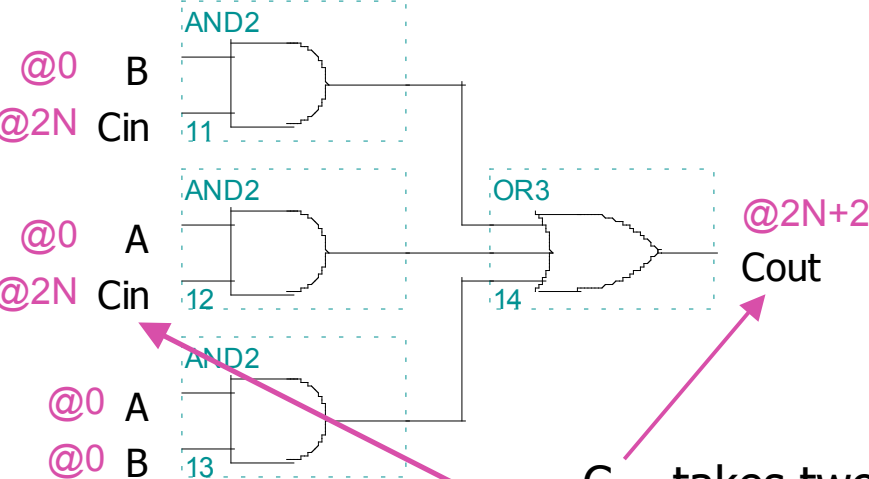
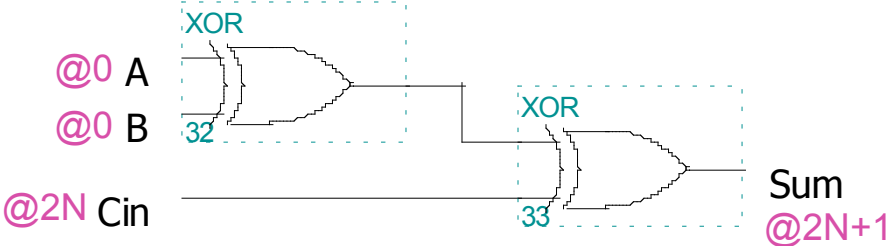
- ◆ Circuit adds or subtracts

- 2s complement:  $A - B = A + (-B) = A + B' + 1$

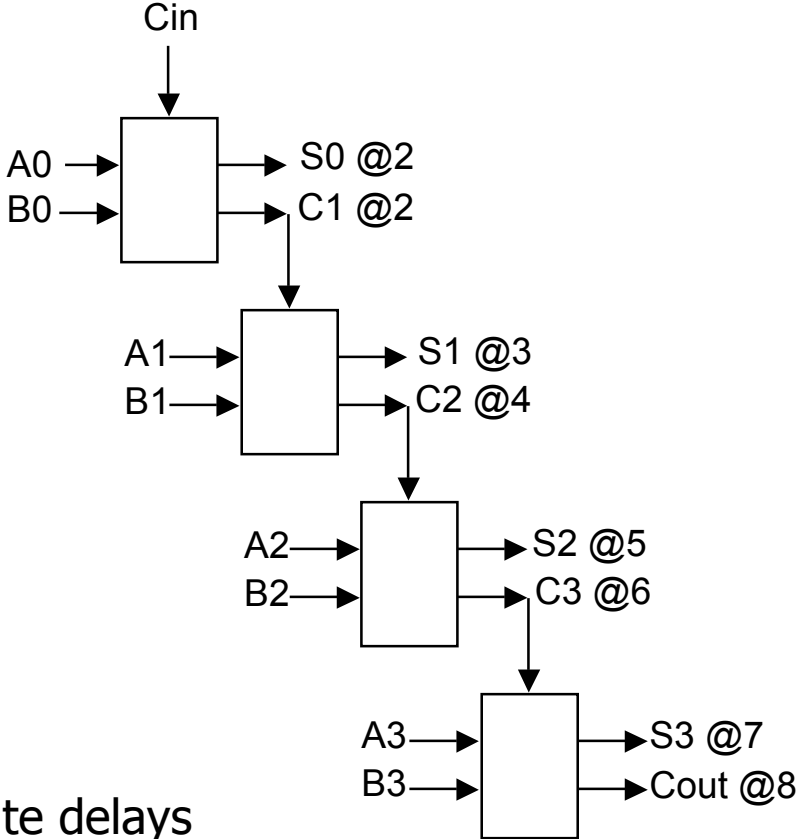


# Problem: Ripple-carry delay

- ◆ Carry propagation limits adder speed



$C_{out}$  takes two gate delays  
 $C_{in}$  arrives late

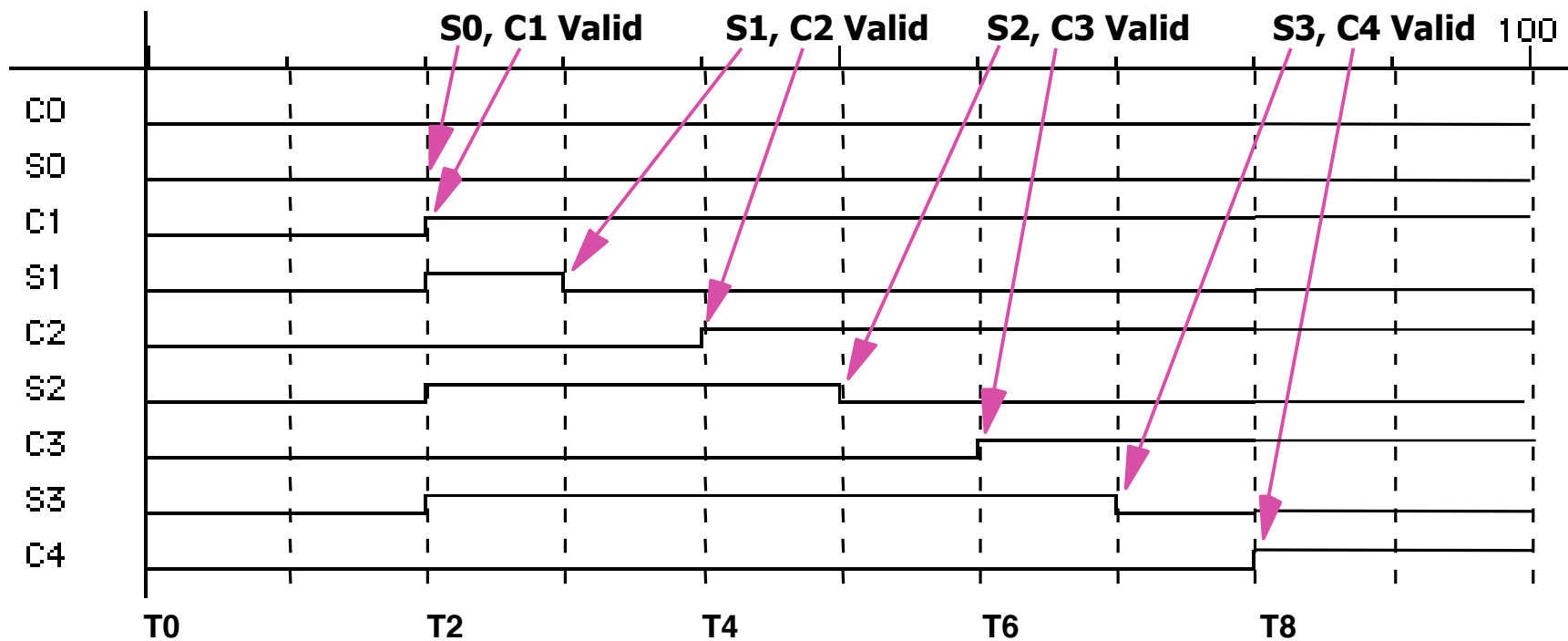




# Ripple-carry adder timing diagram

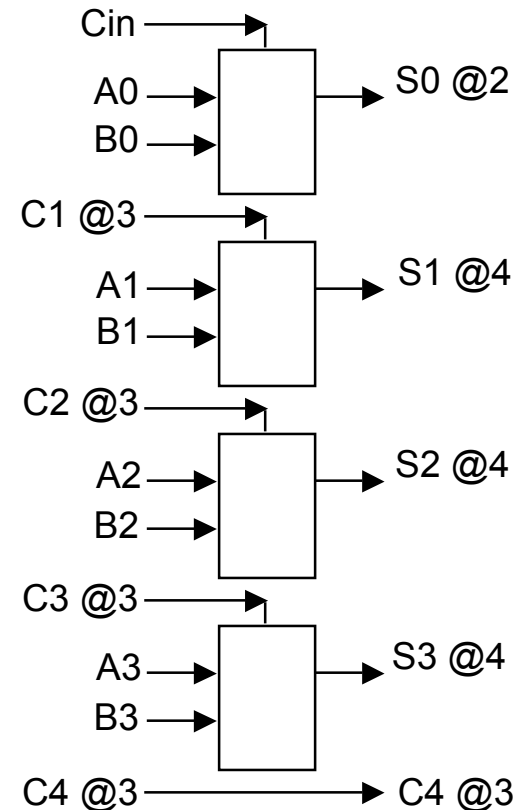
- ◆ Critical delay

- Carry propagation
- $1111 + 0001 = 10000$  is worst case



# One solution: Carry lookahead logic

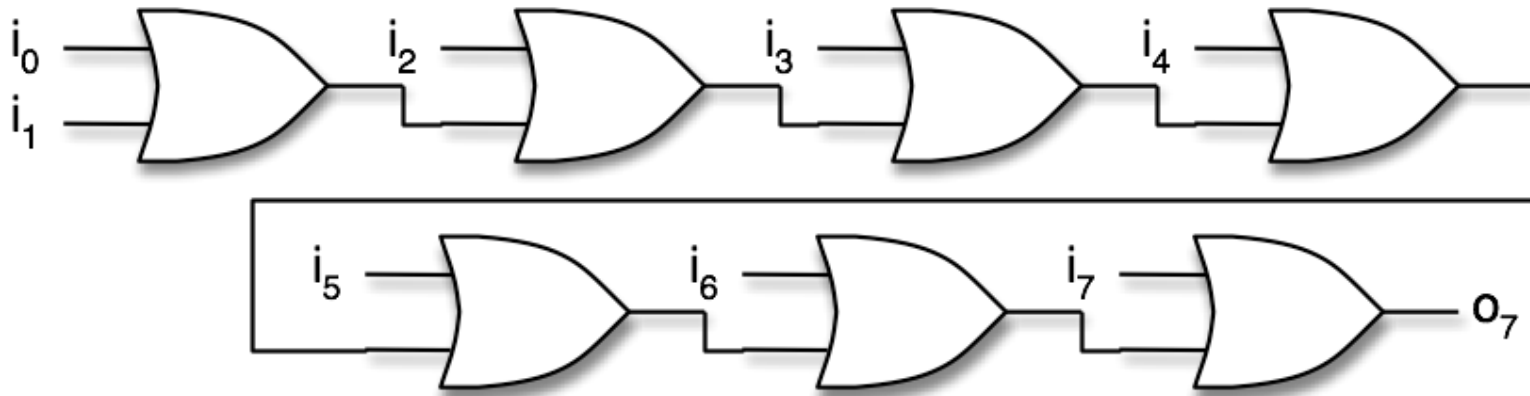
- ◆ Compute all the carries in parallel
  - Derive carries from the data inputs
    - ⇒ Not from intermediate carries
    - ⇒ Use two-level logic
  - Compute all sums in parallel
- ◆ Cascade simple adders to make large adders
- ◆ Speed improvement
  - 16-bit ripple-carry:  $\sim 32$  gate delays
  - 16-bit carry-lookahead:  $\sim 8$  gate delays
- ◆ Issues
  - Complex combinational logic



# A Digression: Linear chains vs. trees

---

- ◆ Lets say we want to implement an 8-input Or function with only 2-input gates



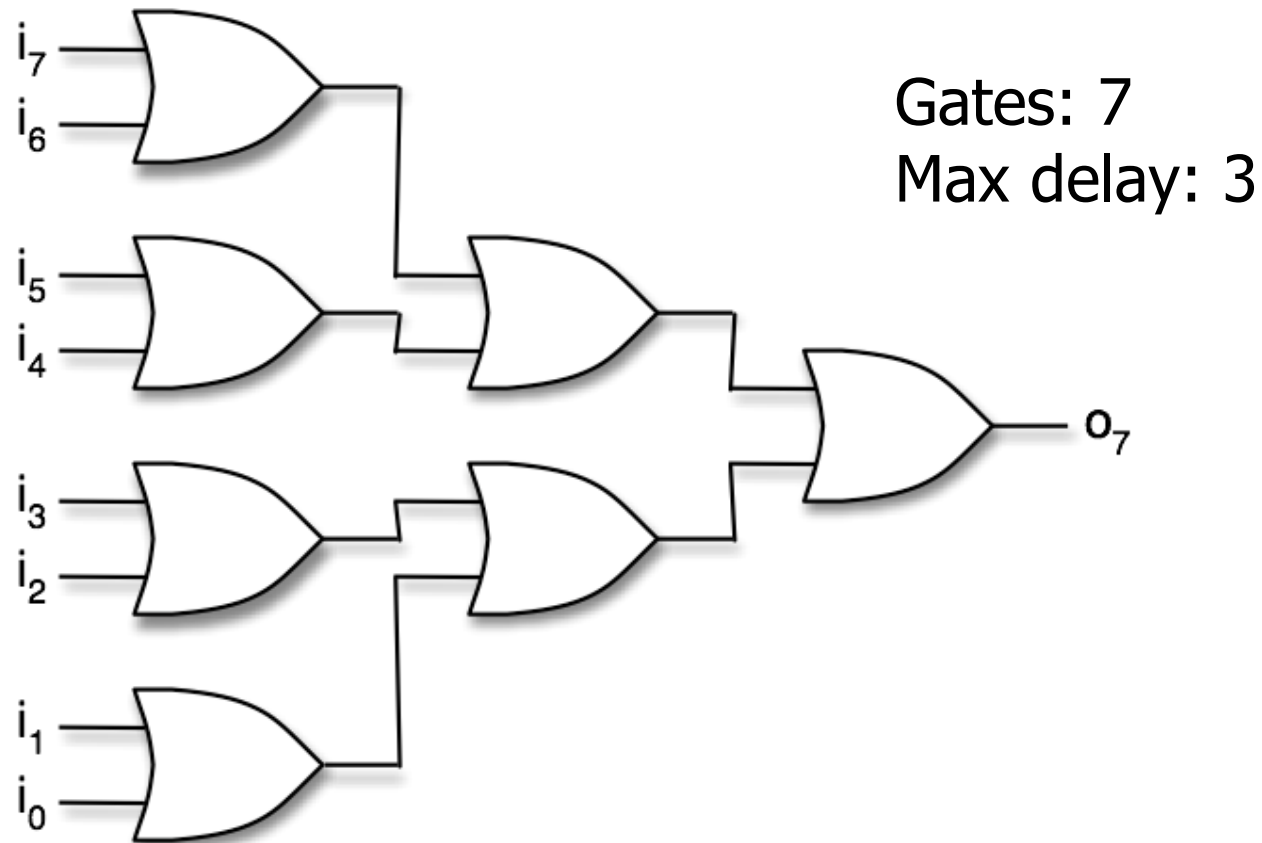
Gates: 7

Max delay: 7

# Linear chains vs. trees, part 2

---

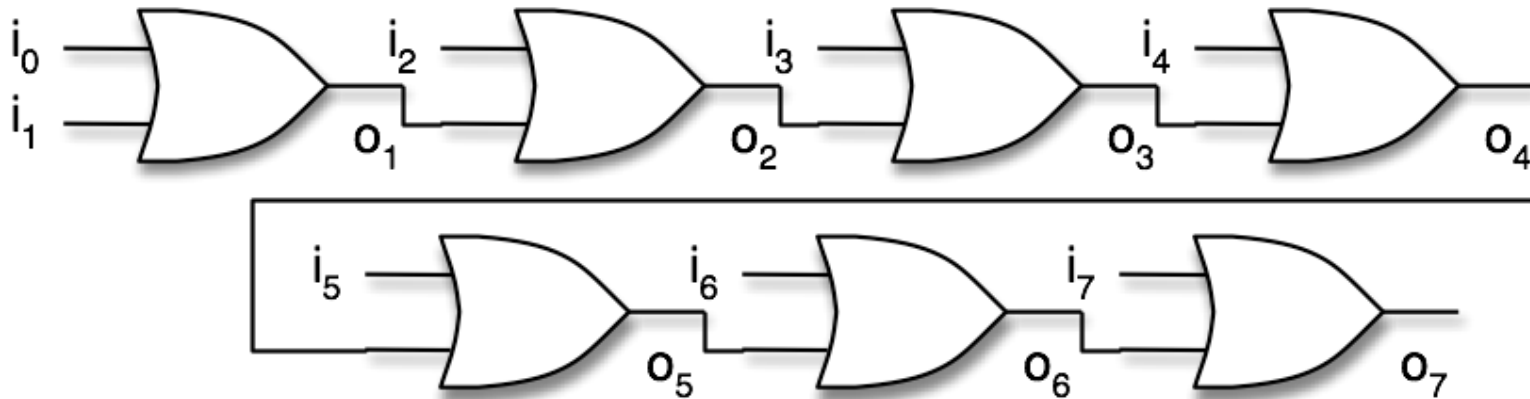
- ◆ Now consider the tree version



# And now we change the problem slightly

---

- ◆ Build a circuit that takes 8 single bit inputs and calculates the Or of the first 2, the Or of the first 3, the Or of the first 4, and so on, up to the Or of all 8

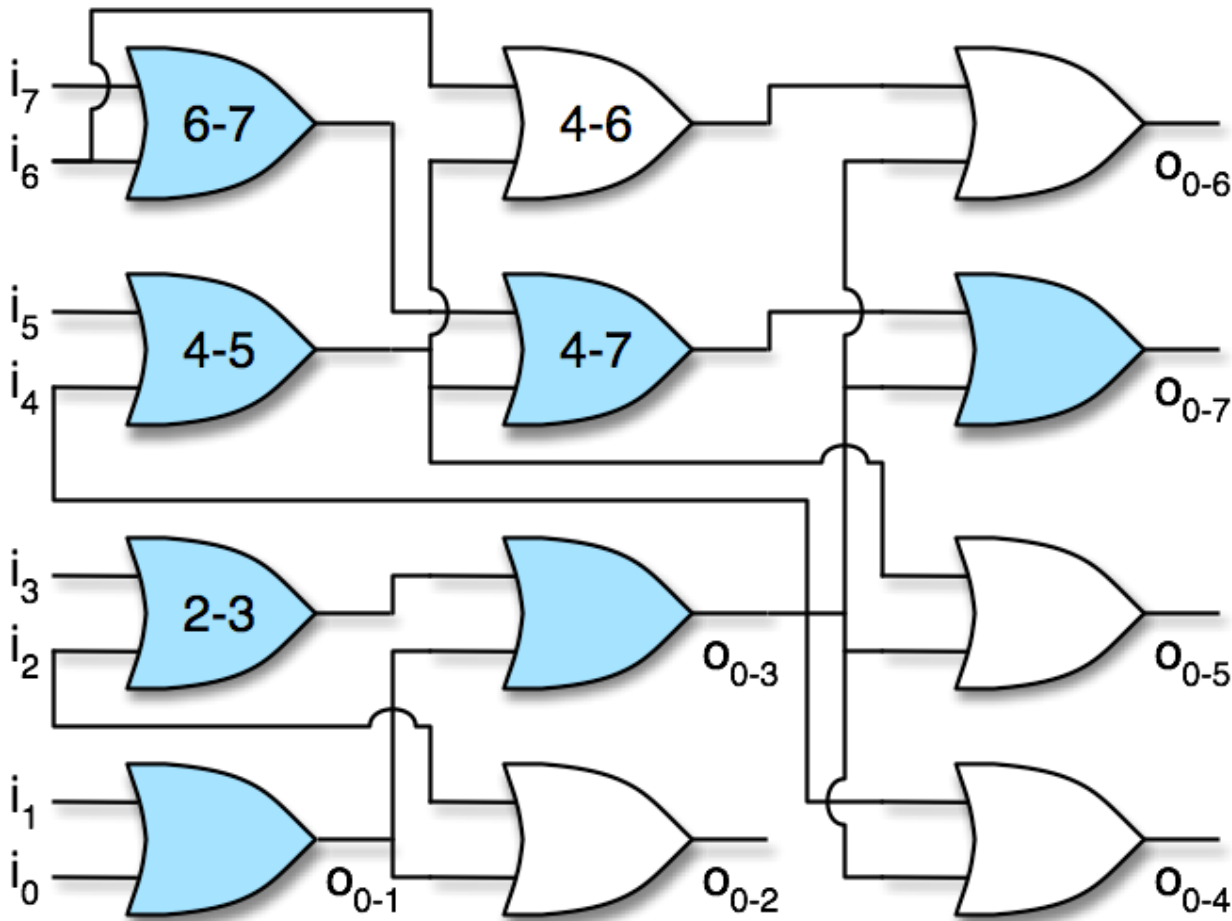


Gates: 7

Max delay: 7

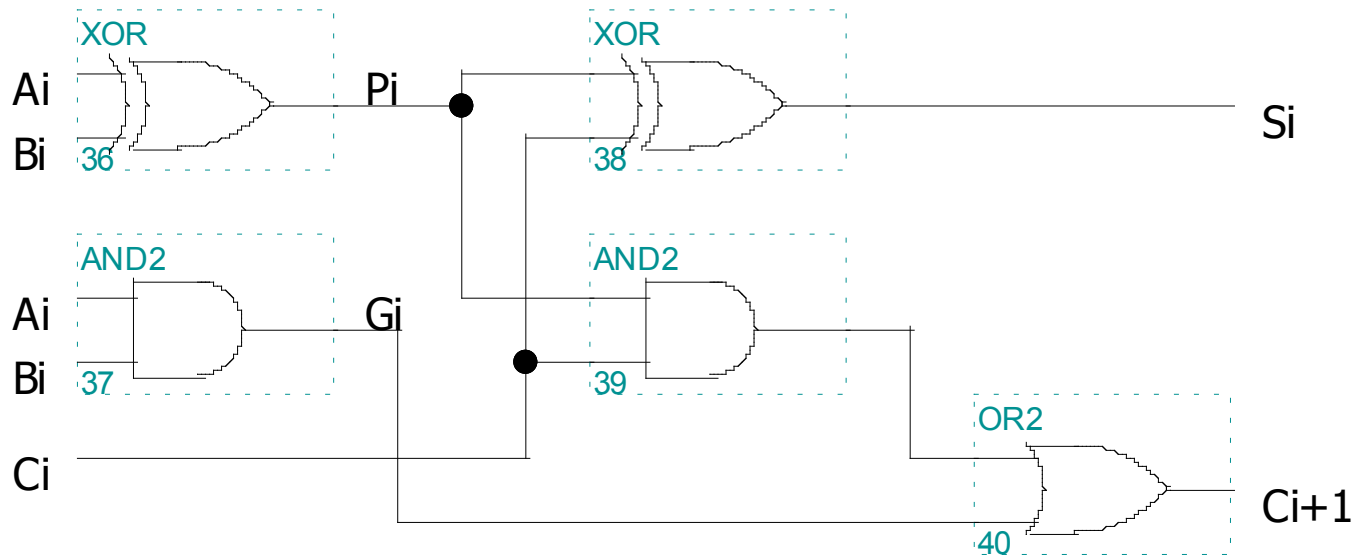
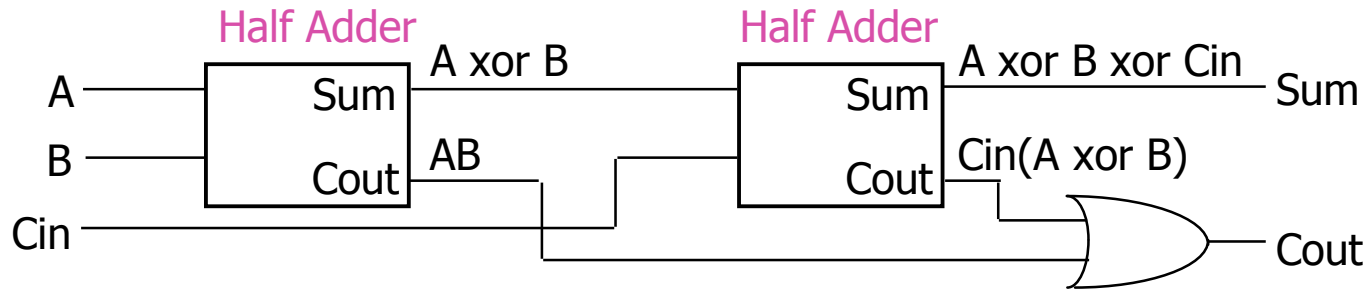
# The tree version of the prefix circuit

---



Gates: 12  
Max delay: 3

# Full adder again



# Carry-lookahead logic

---

- ◆ Carry **generate**:  $G_i = A_i B_i$ 
  - Generate carry when  $A = B = 1$
- ◆ Carry **propagate**:  $P_i = A_i \text{ xor } B_i$ 
  - Propagate carry-in to carry-out when  $(A \text{ xor } B) = 1$
- ◆ Sum and Cout in terms of generate/propagate:
  - $S_i = A_i \text{ xor } B_i \text{ xor } C_i$   
 $= P_i \text{ xor } C_i$
  - $C_{i+1} = A_i B_i + C_i (A_i \text{ xor } B_i)$   
 $= G_i + C_i P_i$



# Carry-lookahead logic (cont'd)

---

- ◆ Re-express the carry logic in terms of G and P

$$C_1 = G_0 + P_0C_0$$

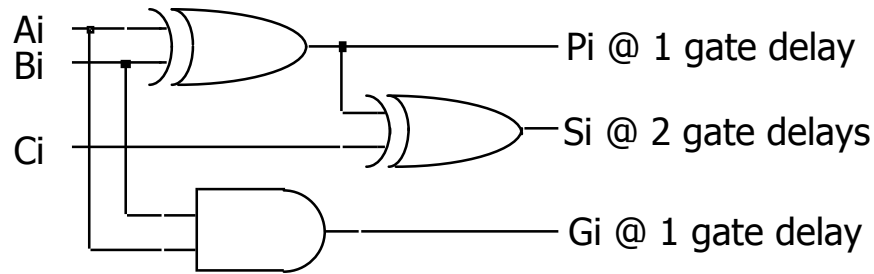
$$C_2 = G_1 + P_1C_1 = G_1 + P_1G_0 + P_1P_0C_0$$

$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

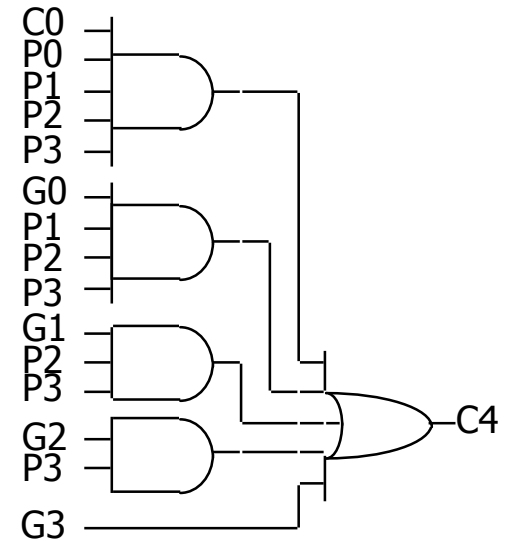
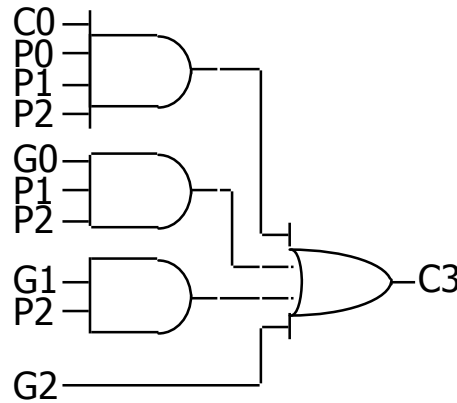
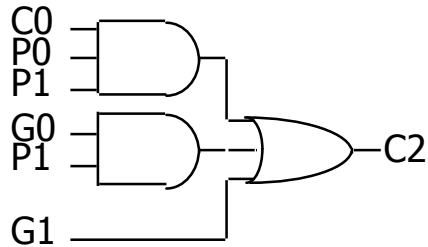
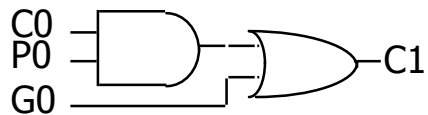
$$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

- ◆ Implement each carry equation with two-level logic
  - Derive intermediate results directly from inputs
    - ⇒ Rather than from carries
  - Allows "sum" computations to proceed in parallel

# Implementing the carry-lookahead logic

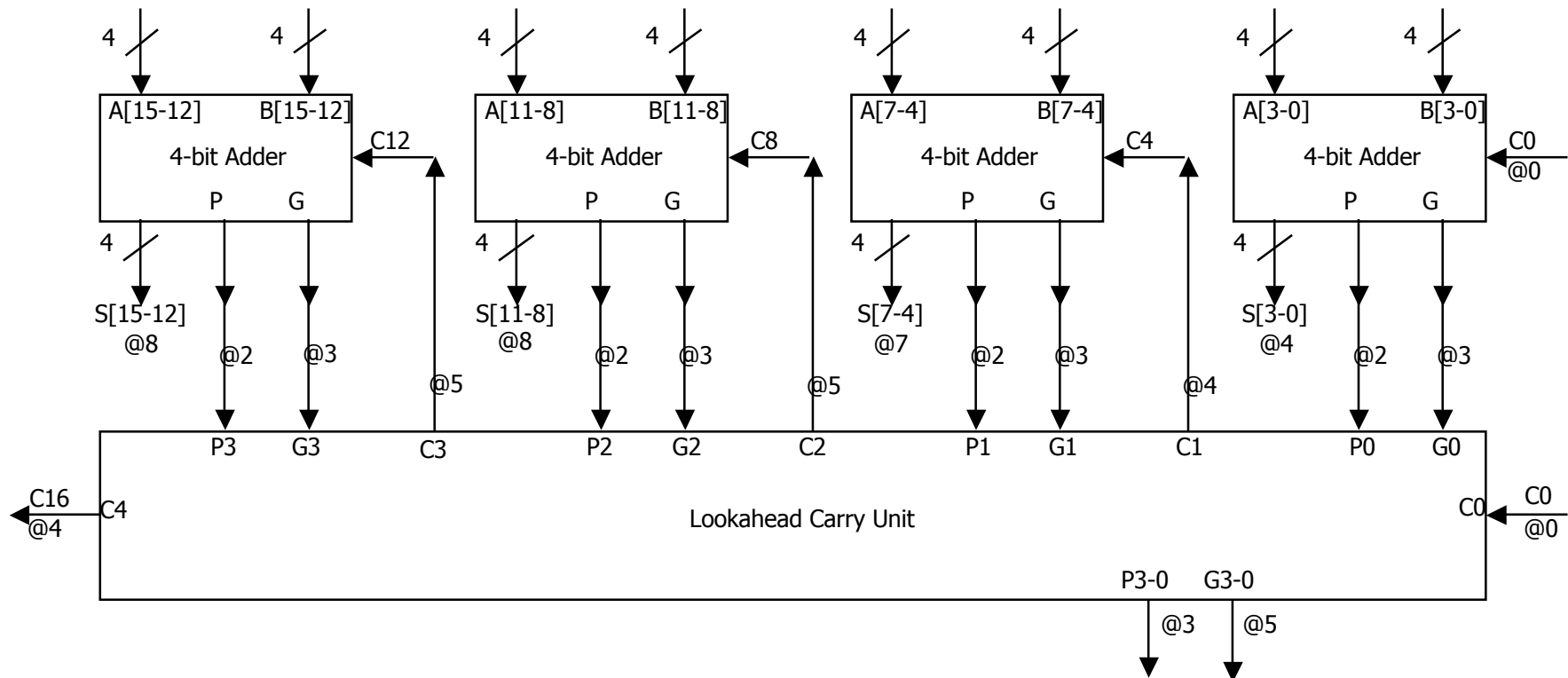


Logic complexity increases with adder size



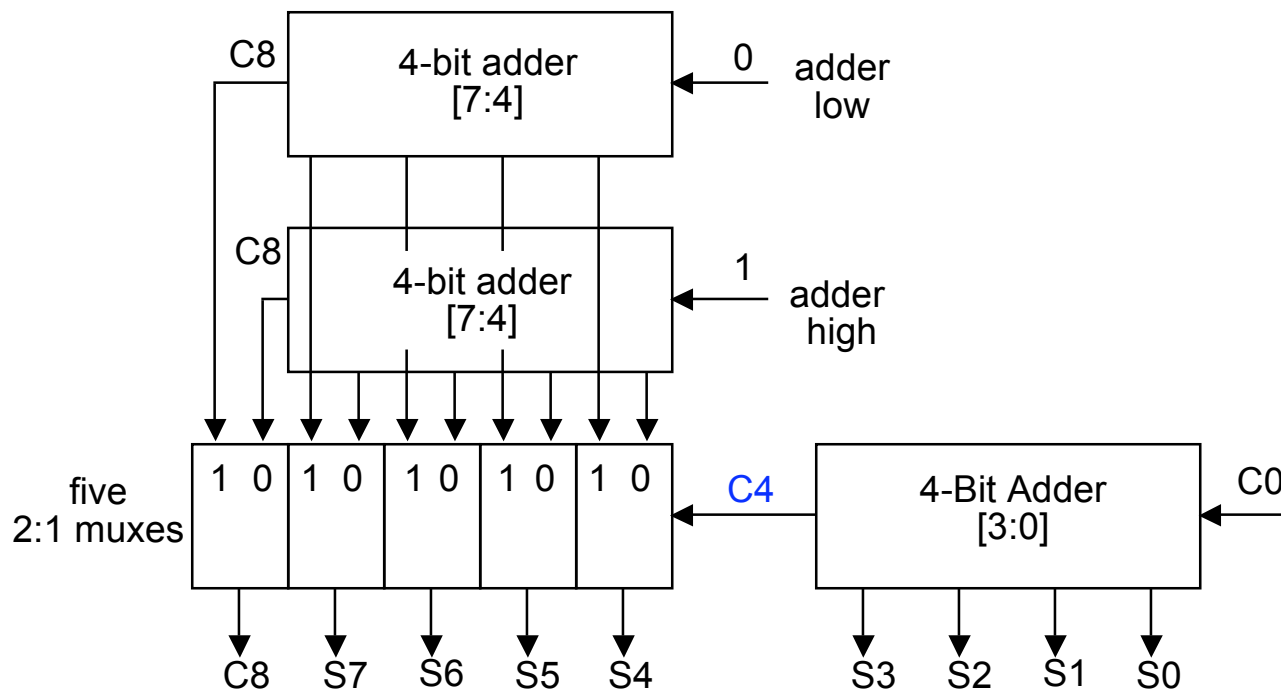
# Cascaded carry-lookahead adder

- ◆ 4 four-bit adders with internal carry lookahead
  - Second level lookahead extends adder to 16 bits



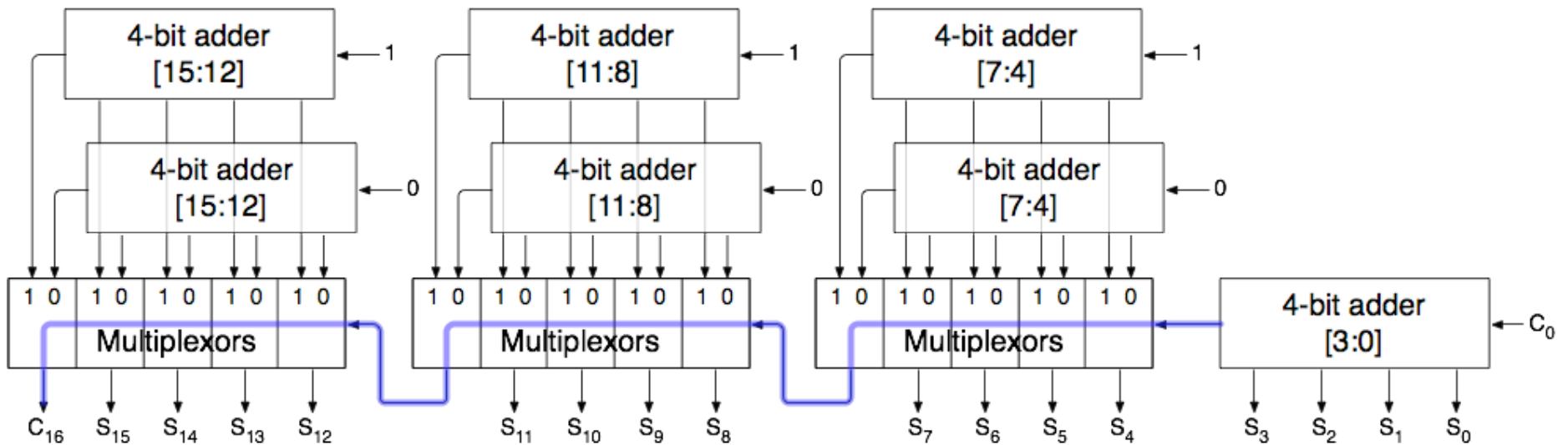
# Another solution: Carry-select adder

- ◆ Redundant hardware speeds carry calculation
  - Compute two high-order sums while waiting for carry-in (C4)
  - Select correct high-order sum after receiving C4



# Scaling of carry-select adders

- ◆ Size: roughly twice the size of a ripple-carry
- Delay: delay through a 4-bit ripple-carry plus the multiplexor path highlighted in blue (3 2-1 multiplexors, in this example)



# We've finished combinational logic...

---

- ◆ What you should know
  - Twos complement arithmetic
  - Truth tables
  - Basic logic gates
  - Schematic diagrams
  - Timing diagrams
  - Minterm and maxterm expansions (canonical, minimized)
  - de Morgan's theorem
  - AND/OR to NAND/NOR logic conversion
  - K-maps, logic minimization, don't cares
  - Multiplexers/demultiplexers
  - PLAs/PALs
  - ROMs
  - Adders