

# Digital Logic Design + Computer Architecture

Sayandeep Saha

Assistant Professor  
Department of Computer  
Science and Engineering  
Indian Institute of Technology  
Bombay

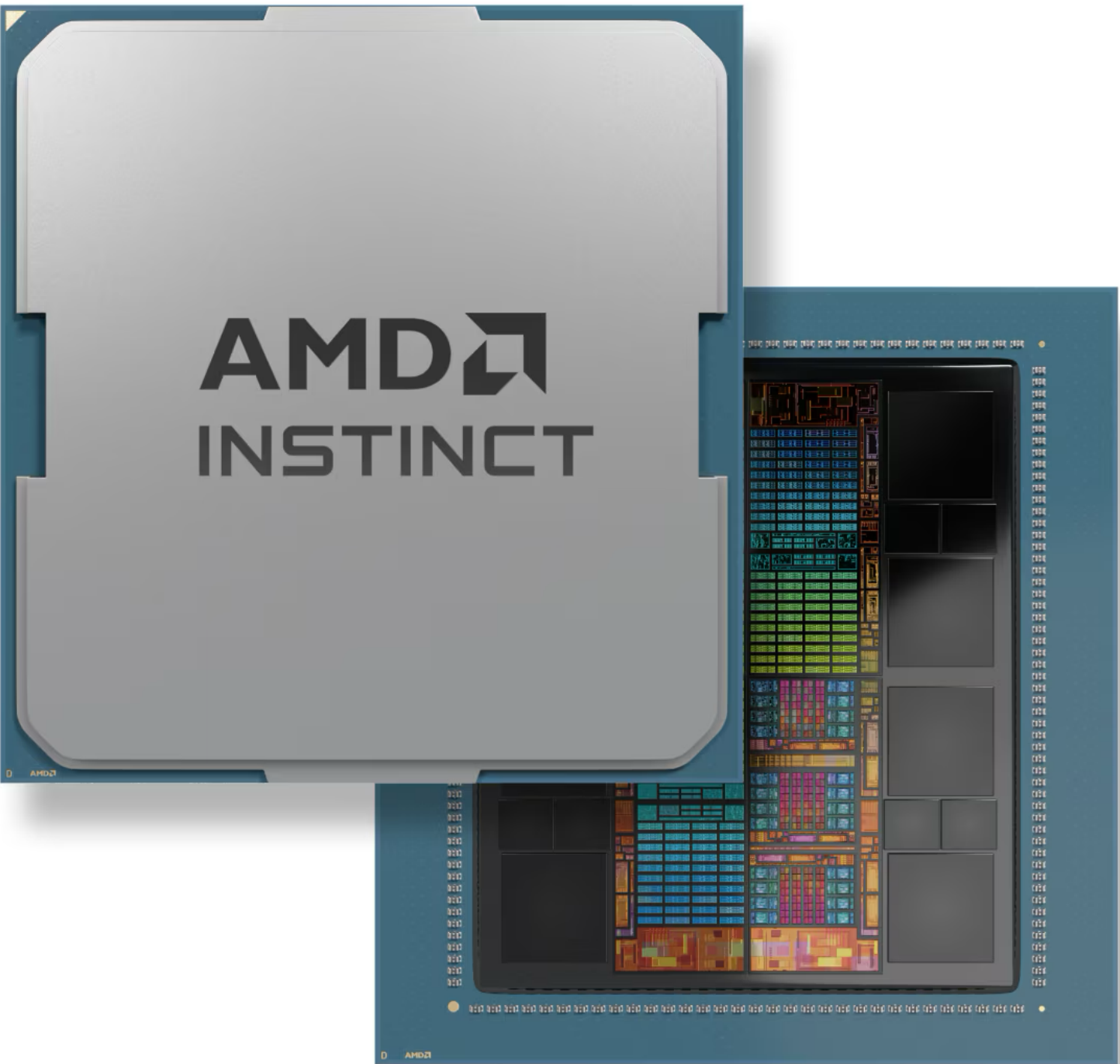




# Combinational Circuits



# Do You Want to Design Some Day?



# Design with Gates

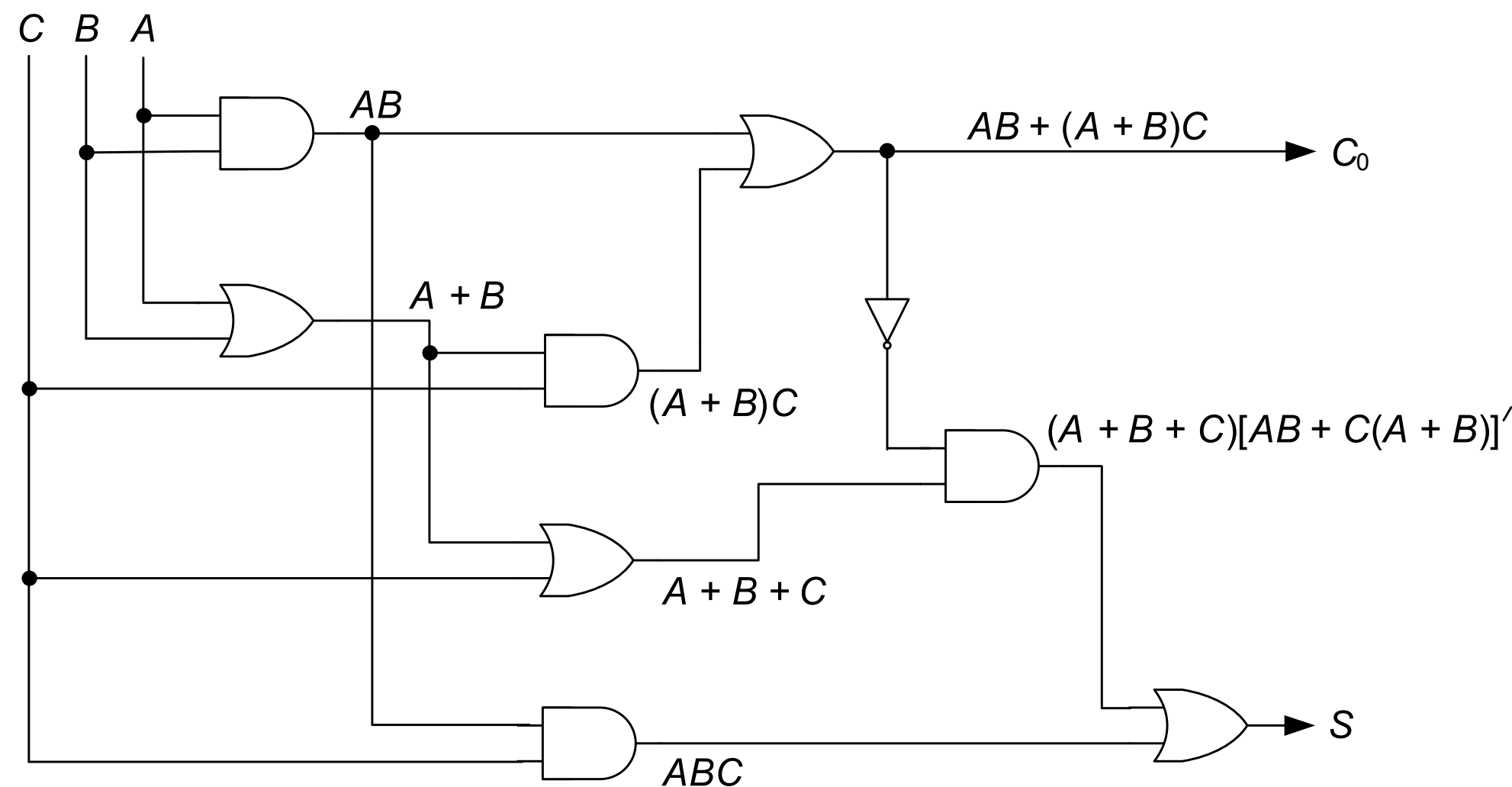
- **Logic gates:** perform logical operations on input signals
- **Positive (negative) logic polarity:** constant 1 (0) denotes a high voltage and constant 0 a low (high) voltage
- **Combinational circuits:** No memorization
- **Synchronous sequential circuits:** have memory; driven by a clock that produces a train of equally spaced pulses
- **Propagation delay:** time to propagate a signal through a gate
- **Asynchronous circuits:** are almost free-running and do not depend on a clock; controlled by initiation and completion signals

# Combinational Circuits

**Circuit analysis:** determine the Boolean function that describes the circuit

- Done by tracing the output of each gate, starting from circuit inputs and continuing towards each circuit output

**Example:** a multi-level realization of a full binary adder



$$C_0 = AB + (A + B)C$$
$$= AB + AC + BC$$

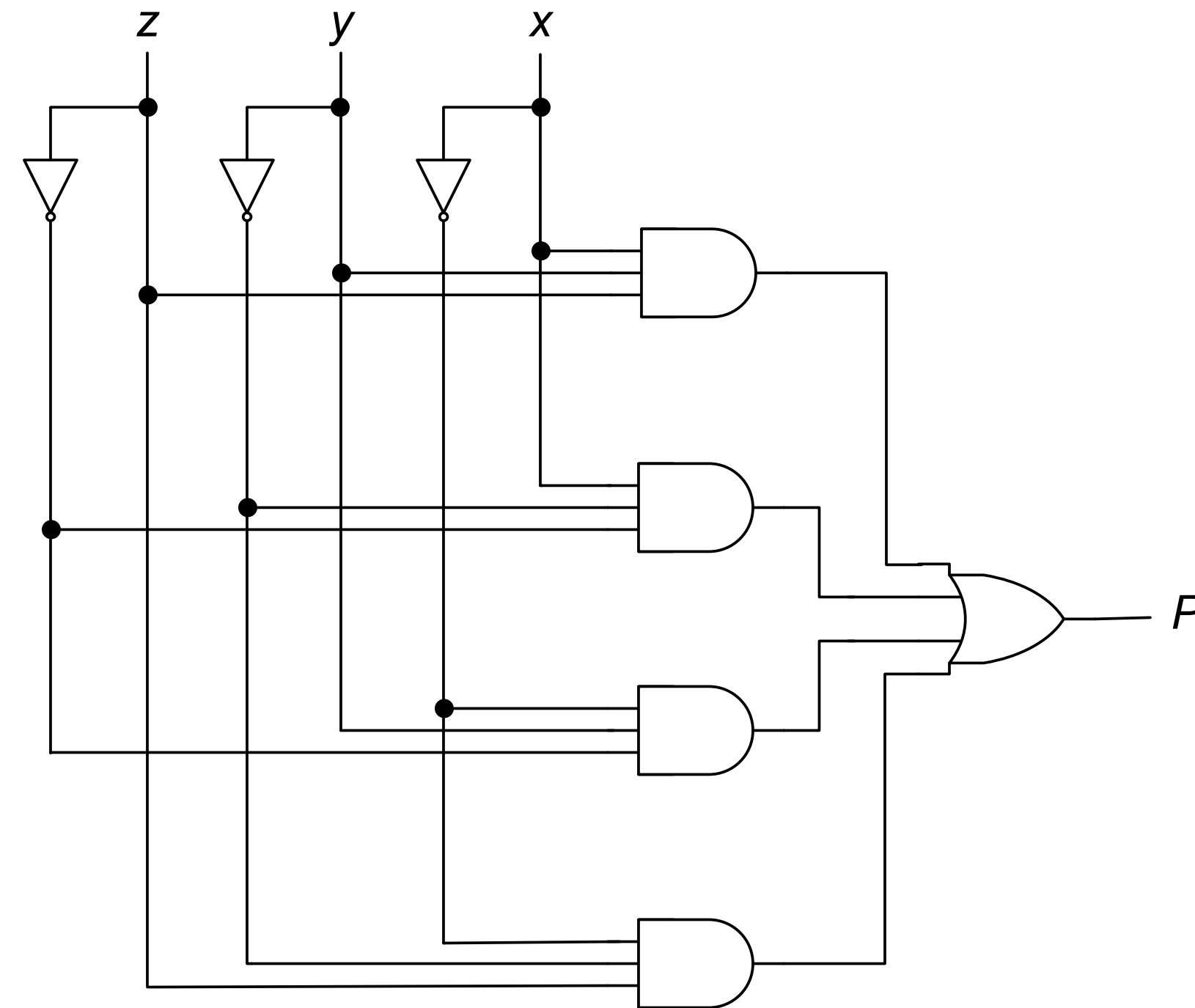
$$S = (A + B + C)[AB + (A + B)C]' + ABC$$
$$= (A + B + C)(A' + B')(A' + C')(B' + C') + ABC$$
$$= AB'C' + A'BC' + A'B'C + ABC$$
$$= A \oplus B \oplus C$$

# Combinational Circuits: Parity-bit Generator

**Parity-bit generator:** produces output value 1 if and only if an odd number of its inputs have value 1

| $xy$ |   | $z$ |    |    |    |
|------|---|-----|----|----|----|
|      |   | 00  | 01 | 11 | 10 |
| $z$  | 0 | 0   | 1  | 0  | 1  |
|      | 1 | 1   | 0  | 1  | 0  |

(a) Map.



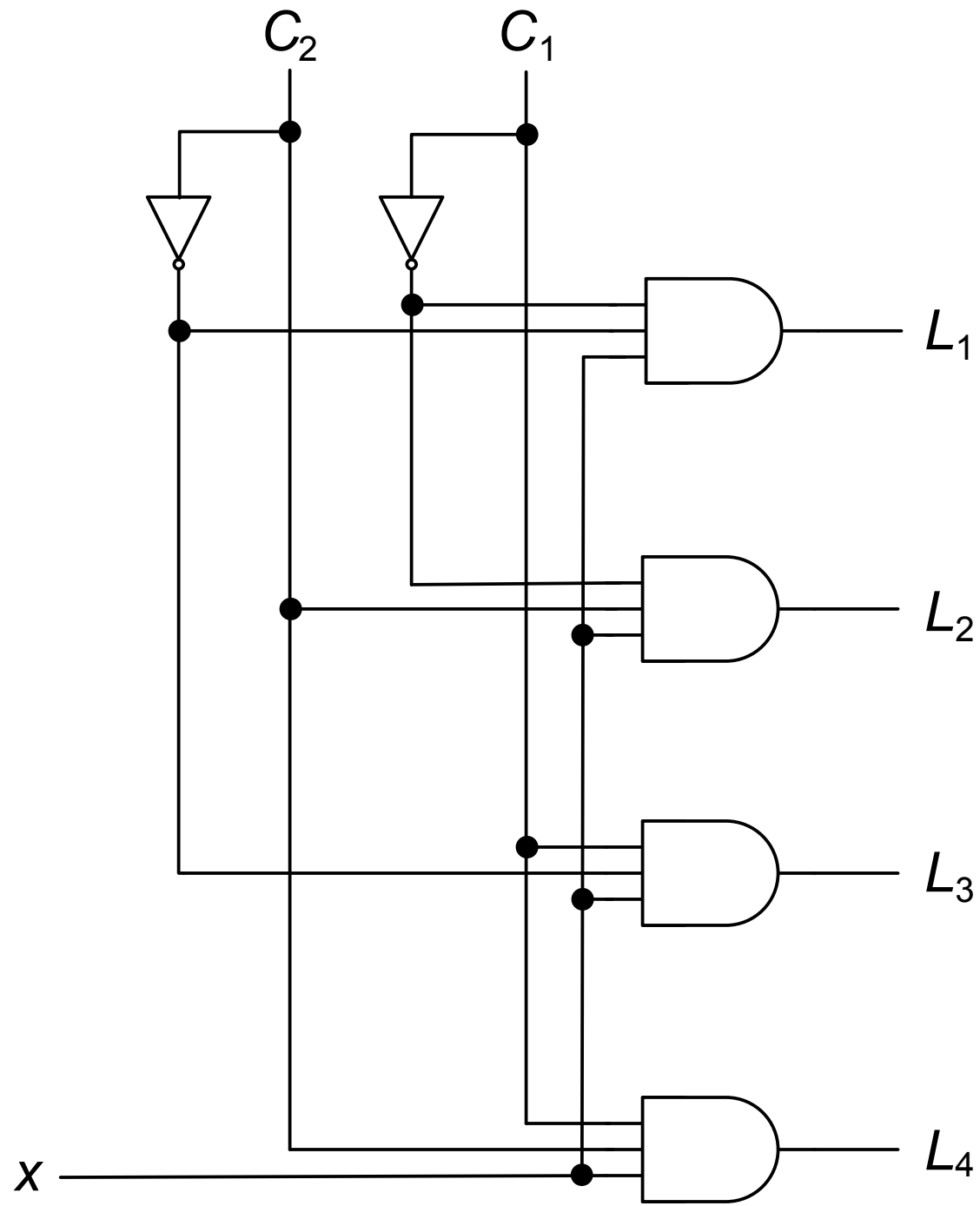
(b) Implementation.

$$P = x'y'z + x'yz' + xy'z' + xyz = x \oplus y \oplus z$$

# Combinational Circuits: Serial to Parallel

**Serial-to-parallel converter:** distributes a sequence of binary digits on a serial input to a set of different outputs, as specified by external control signals

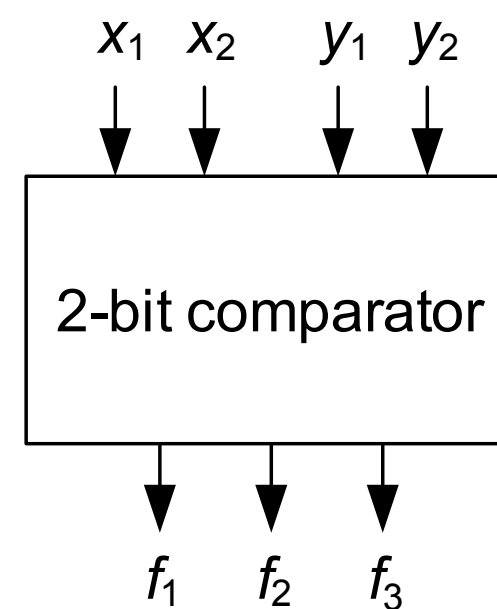
| <i>Control</i> |       | <i>Output lines</i> |       |       |       | <i>Logic equations</i> |
|----------------|-------|---------------------|-------|-------|-------|------------------------|
| $C_1$          | $C_2$ | $L_1$               | $L_2$ | $L_3$ | $L_4$ |                        |
| 0              | 0     | $x$                 | 0     | 0     | 0     | $L_1 = xC_1' C_2'$     |
| 0              | 1     | 0                   | $x$   | 0     | 0     | $L_2 = xC_1' C_2$      |
| 1              | 0     | 0                   | 0     | $x$   | 0     | $L_3 = xC_1 C_2'$      |
| 1              | 1     | 0                   | 0     | 0     | $x$   | $L_4 = xC_1 C_2$       |



# Combinational Circuits: Comparators

**$n$ -bit comparator:** compares the magnitude of two numbers  $X$  and  $Y$ , and has three outputs  $f_1, f_2$ , and  $f_3$

- $f_1 = 1$  iff  $X > Y$
- $f_2 = 1$  iff  $X = Y$
- $f_3 = 1$  iff  $X < Y$



(a) Block diagram.

| $y_1y_2 \backslash x_1x_2$ |    | $x_1x_2$ |    |    |    |
|----------------------------|----|----------|----|----|----|
|                            |    | 00       | 01 | 11 | 10 |
| $y_1y_2$                   | 00 | 2        | 1  | 1  | 1  |
|                            | 01 | 3        | 2  | 1  | 1  |
|                            | 11 | 3        | 3  | 2  | 3  |
|                            | 10 | 3        | 3  | 1  | 2  |

(b) Map for  $f_1, f_2$ , and  $f_3$ .

$$f_1 = ?$$

$$f_2 = ?$$

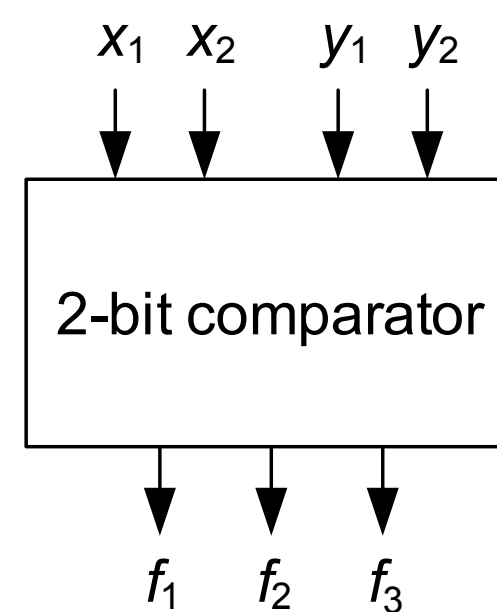
$$f_3 = ?$$



# Combinational Circuits: Comparators

**$n$ -bit comparator:** compares the magnitude of two numbers  $X$  and  $Y$ , and has three outputs  $f_1, f_2$ , and  $f_3$

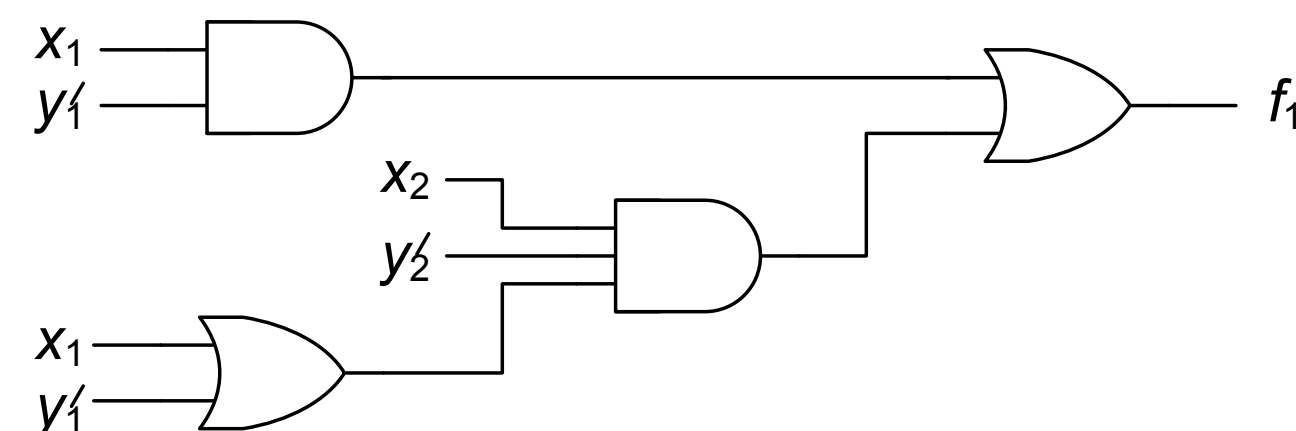
- $f_1 = 1$  iff  $X > Y$
- $f_2 = 1$  iff  $X = Y$
- $f_3 = 1$  iff  $X < Y$



(a) Block diagram.

| $x_1x_2 \backslash y_1y_2$ |    | 00 | 01 | 11 | 10 |
|----------------------------|----|----|----|----|----|
| 00                         | 00 | 2  | 1  | 1  | 1  |
|                            | 01 | 3  | 2  | 1  | 1  |
|                            | 11 | 3  | 3  | 2  | 3  |
|                            | 10 | 3  | 3  | 1  | 2  |

(b) Map for  $f_1, f_2$ , and  $f_3$ .



(c) Circuit for  $f_1$ .

$$f_1 = x_1x_2y_2' + x_2y_1'y_2' + x_1y_1'$$

$$= (x_1 + y_1')x_2y_2' + x_1y_1'$$

$$f_2 = x_1'x_2'y_1'y_2' + x_1'x_2y_1'y_2 + x_1x_2'y_1y_2' + x_1x_2y_1y_2$$

$$= x_1'y_1'(x_2'y_2' + x_2y_2) + x_1y_1(x_2'y_2' + x_2y_2)$$

$$= (x_1'y_1' + x_1y_1)(x_2'y_2' + x_2y_2)$$

$$f_3 = x_2'y_1y_2 + x_1'x_2'y_2 + x_1'y_1$$

$$= x_2'y_2(y_1 + x_1') + x_1'y_1$$



# Combinational Circuits: Comparators

**Four-bit comparator:** 8 inputs (four for  $A$ , four for  $B$ , and three outputs  $A > B$ ,  $A < B$  and  $A = B$ )

$$x_i = A_i B_i + A_i' B_i' \quad i = 0, 1, 2, 3$$

$$(A = B) = x_3 x_2 x_1 x_0$$

$$(A > B) = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$$

$$(A < B) = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0$$

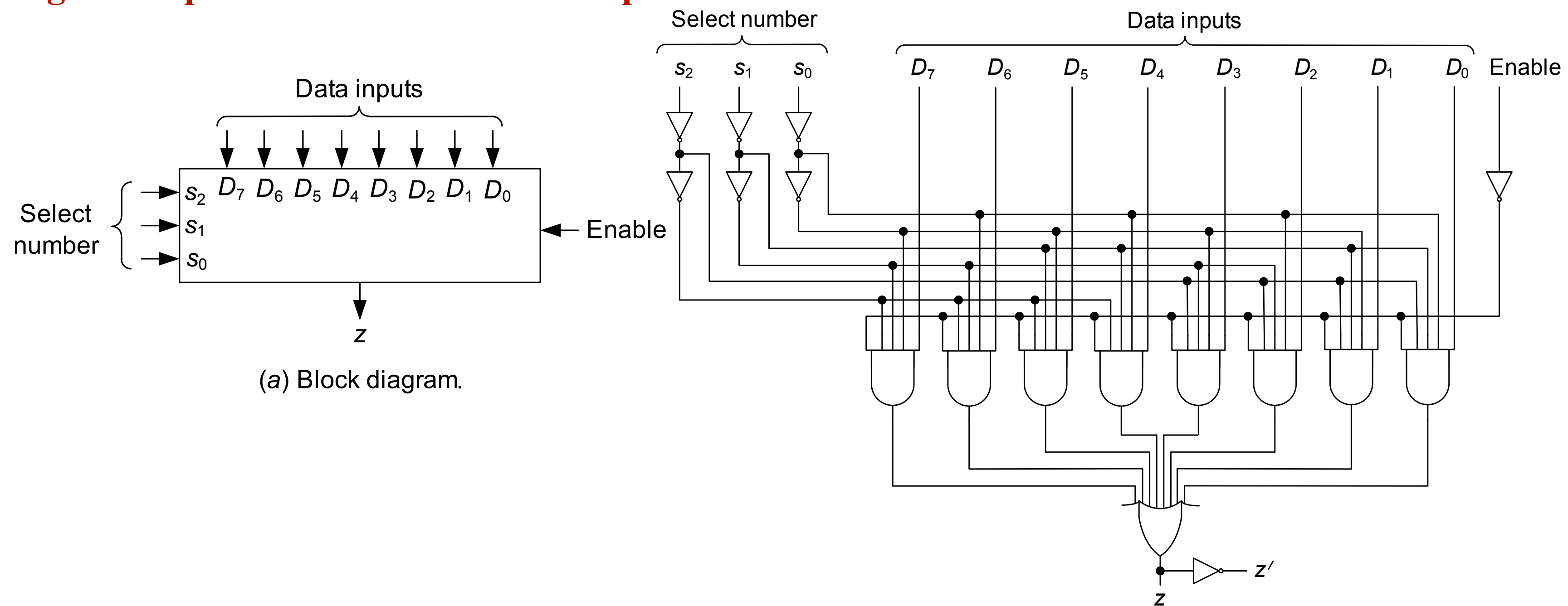


# Combinational Circuits: Multiplexers

**Multiplexer:** electronic switch that connects one of  $n$  inputs to the output

**Data selector:** application of multiplexer

- $n$  data input lines,  $D_0, D_1, \dots, D_{n-1}$
- $m$  select digit inputs  $s_0, s_1, \dots, s_{m-1}$
- 1 output
- **Can you design a simple data selectors with 2 input data lines?**



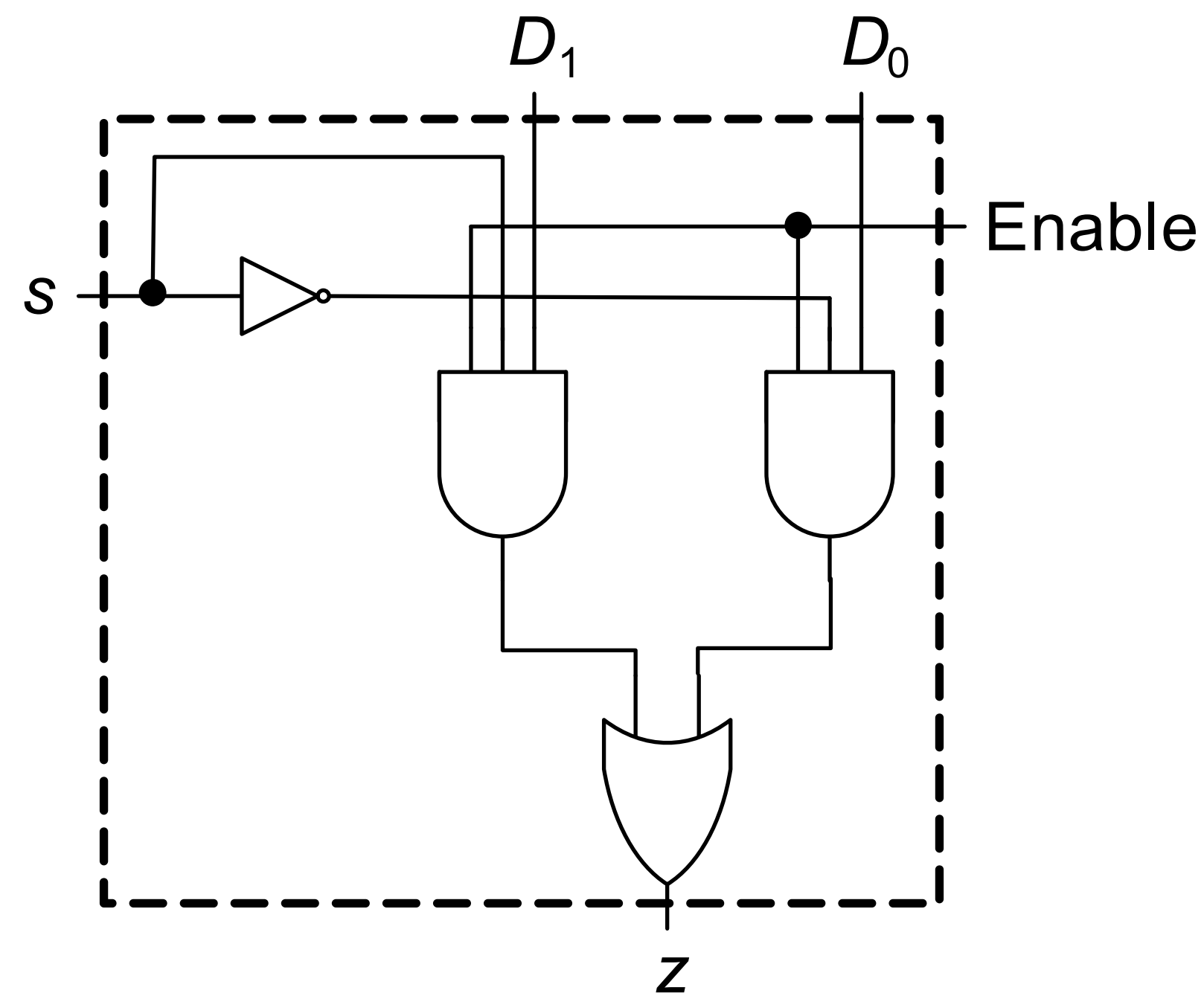
(b) Logic diagram.



# Combinational Circuits: Multiplexers

**Data selectors:** can implement arbitrary switching functions

**Example:** implementing two-variable functions



$$z = sD_1 + s'D_0$$

If  $s = A$ ,  $B = D_0$ , and  $B' = D_1$ , then  $z = A \oplus B$ .

If  $s = A$ ,  $D_0 = 1$ , and  $D_1 = B'$ , then  $z = A' + B'$ .



# Implementing Switching Function with Mux

**To implement an  $n$ -variable function:** a data selector with  $n-1$  select inputs and  $2^{n-1}$  data inputs

**Implementing three-variable functions:**

$$z = s_2's_1'D_0 + s_2's_1D_1 + s_2s_1'D_2 + s_2s_1D_3$$

**Example:**  $s_1 = A, s_2 = B, D_0 = C, D_1 = 1, D_2 = 0, D_3 = C'$

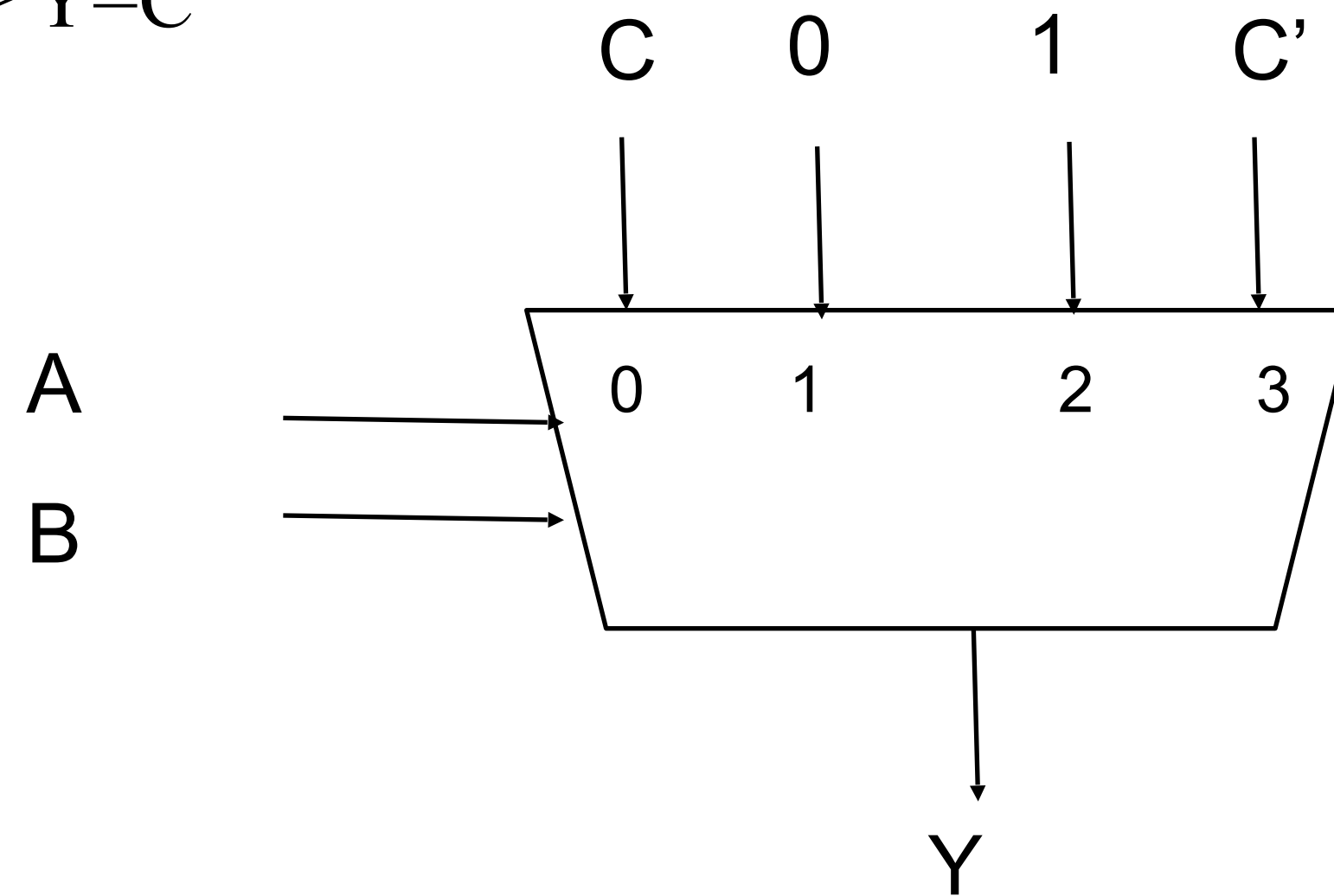
$$\begin{aligned} z &= A'B'C + AB' + ABC' \\ &= AC' + B'C \end{aligned}$$

**General case:** Assign  $n-1$  variables to the select inputs and last variable and constants 0 and 1 to the data inputs such that desired function results



# Implementing Switching Function with Mux

- $Y = AC' + B'C$
- Make A, B as select lines.
  - A,B=0,0  $\Rightarrow Y=C$
  - A,B=0,1  $\Rightarrow Y=0$
  - A,B=1,0  $\Rightarrow Y=C'+C=1$
  - A,B=1,1  $\Rightarrow Y=C'$



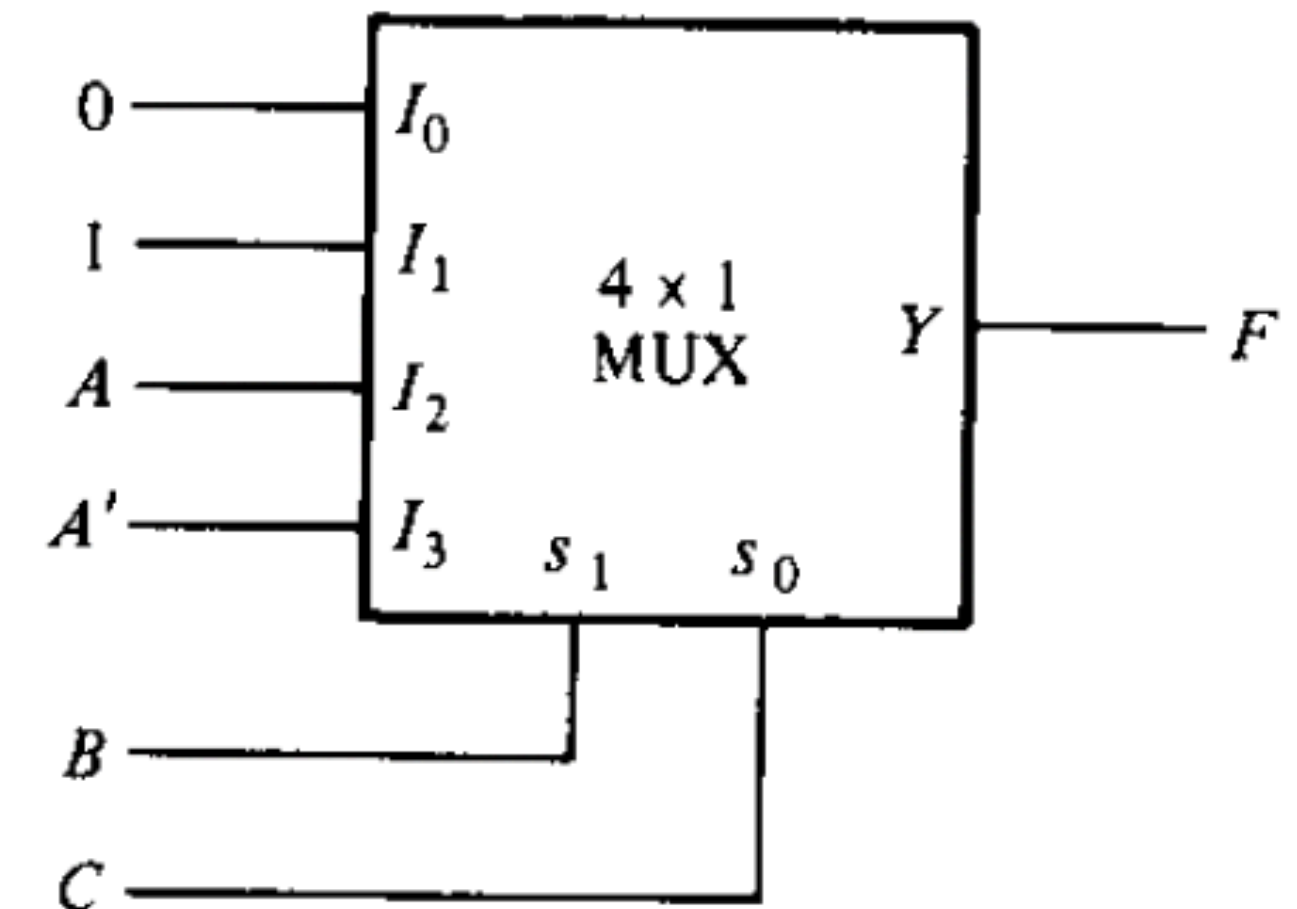


# Implementing Switching Function with Mux

$$F(A, B, C) = \sum (1, 3, 5, 6)$$

| Minterm | A | B | C | F |
|---------|---|---|---|---|
| 0       | 0 | 0 | 0 | 0 |
| 1       | 0 | 0 | 1 | 1 |
| 2       | 0 | 1 | 0 | 0 |
| 3       | 0 | 1 | 1 | 1 |
| 4       | 1 | 0 | 0 | 0 |
| 5       | 1 | 0 | 1 | 1 |
| 6       | 1 | 1 | 0 | 1 |
| 7       | 1 | 1 | 1 | 0 |

|      | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|------|-------|-------|-------|-------|
| $A'$ | 0     | ①     | 2     | ③     |
| $A$  | 4     | ⑤     | ⑥     | 7     |
|      | 0     | 1     | $A$   | $A'$  |



# Adders: Half Adder

Add two variables and generate the sum and carry

| $x$ | $y$ | $C$ | $S$ |
|-----|-----|-----|-----|
| 0   | 0   | 0   | 0   |
| 0   | 1   | 0   | 1   |
| 1   | 0   | 0   | 1   |
| 1   | 1   | 1   | 0   |

$$S = x \oplus y$$

$$C = xy$$



# Adders: Full Addder

Add two variables and an input carry..

| $x$ | $y$ | $z$ | $C$ | $S$ |
|-----|-----|-----|-----|-----|
| 0   | 0   | 0   | 0   | 0   |
| 0   | 0   | 1   | 0   | 1   |
| 0   | 1   | 0   | 0   | 1   |
| 0   | 1   | 1   | 1   | 0   |
| 1   | 0   | 0   | 0   | 1   |
| 1   | 0   | 1   | 1   | 0   |
| 1   | 1   | 0   | 1   | 0   |
| 1   | 1   | 1   | 1   | 1   |

Try it yourself...!!!

# Adders: Full Adder

Add two variables and an input carry..

| $x$ | $y$ | $z$ | $C$ | $S$ |
|-----|-----|-----|-----|-----|
| 0   | 0   | 0   | 0   | 0   |
| 0   | 0   | 1   | 0   | 1   |
| 0   | 1   | 0   | 0   | 1   |
| 0   | 1   | 1   | 1   | 0   |
| 1   | 0   | 0   | 0   | 1   |
| 1   | 0   | 1   | 1   | 0   |
| 1   | 1   | 0   | 1   | 0   |
| 1   | 1   | 1   | 1   | 1   |

$$S = x \oplus y \oplus z$$

$$C = xy + yz + zx$$



# Adders: Full Adder with Half Adders

Use two half adder and something else to generate a full addder

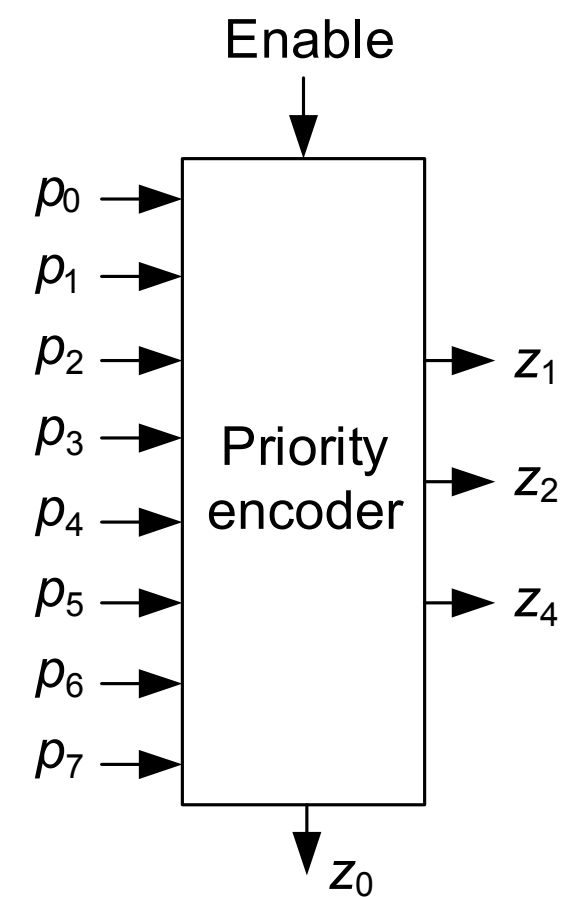
Try it yourself...!!!

# Priority Encoders

**Priority encoder:**  $n$  input lines and  $\log_2 n$  output lines

- Input lines represent units that may request service
- When inputs  $p_i$  and  $p_j$ , such that  $i > j$ , request service simultaneously, line  $p_i$  has priority over line  $p_j$
- Encoder produces a binary output code indicating which of the input lines requesting service has the highest priority

**Example:** Eight-input, three-output priority encoder



(a) Block diagram.

| Input lines |        |        |        |        |        |        |       | Outputs |       |       |
|-------------|--------|--------|--------|--------|--------|--------|-------|---------|-------|-------|
| $p_0$       | $p_1$  | $p_2$  | $p_3$  | $p_4$  | $p_5$  | $p_6$  | $p_7$ | $z_4$   | $z_2$ | $z_1$ |
| 1           | 0      | 0      | 0      | 0      | 0      | 0      | 0     | 0       | 0     | 0     |
| $\phi$      | 1      | 0      | 0      | 0      | 0      | 0      | 0     | 0       | 0     | 1     |
| $\phi$      | $\phi$ | 1      | 0      | 0      | 0      | 0      | 0     | 0       | 1     | 0     |
| $\phi$      | $\phi$ | $\phi$ | 1      | 0      | 0      | 0      | 0     | 0       | 1     | 1     |
| $\phi$      | $\phi$ | $\phi$ | $\phi$ | 1      | 0      | 0      | 0     | 1       | 0     | 0     |
| $\phi$      | $\phi$ | $\phi$ | $\phi$ | $\phi$ | 1      | 0      | 0     | 1       | 0     | 1     |
| $\phi$      | $\phi$ | $\phi$ | $\phi$ | $\phi$ | $\phi$ | 1      | 0     | 1       | 1     | 0     |
| $\phi$      | $\phi$ | $\phi$ | $\phi$ | $\phi$ | $\phi$ | $\phi$ | 1     | 1       | 1     | 1     |

(b) Truth table.

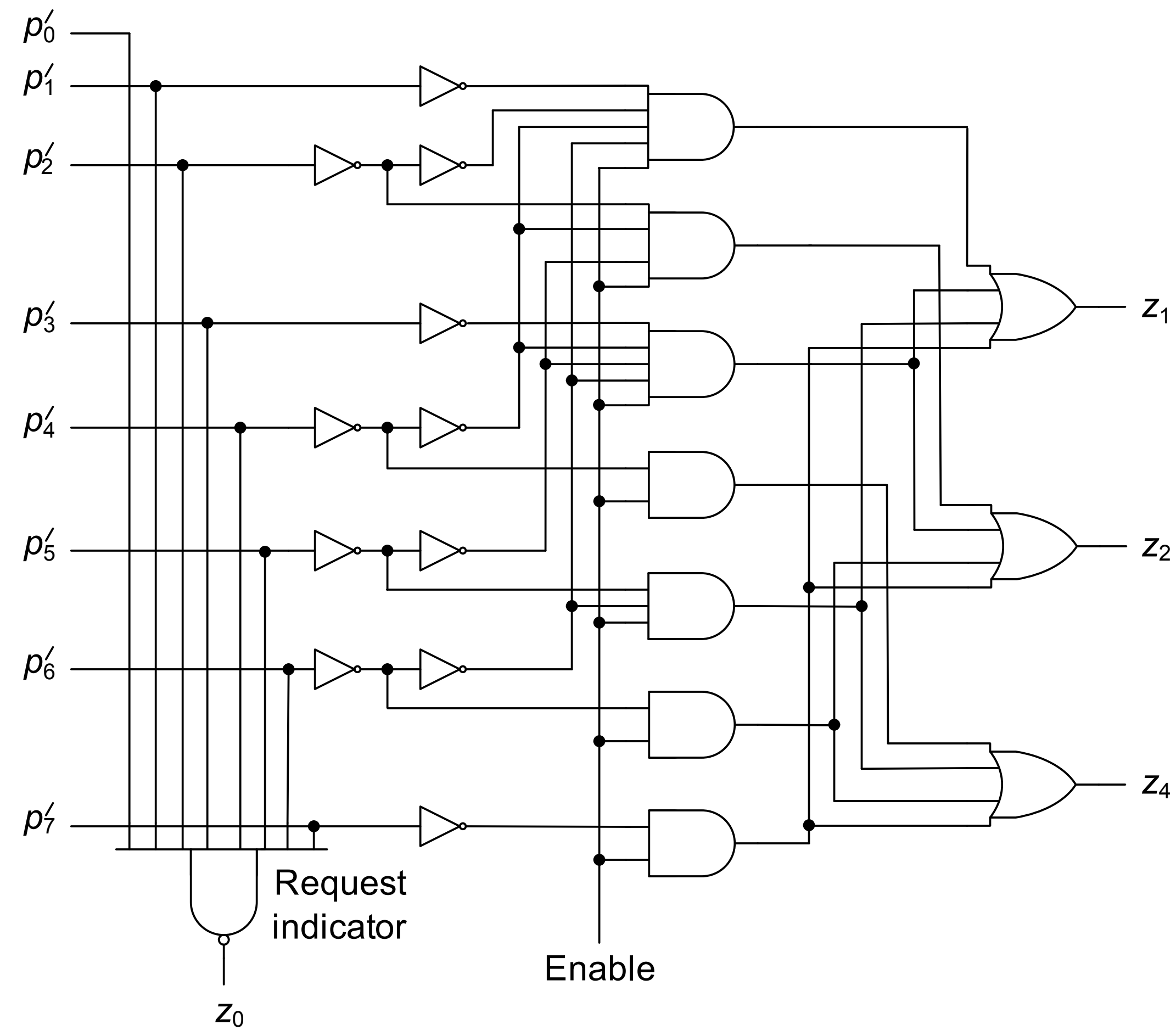
$$z_4 = p_4 p_5' p_6' p_7' + p_5 p_6' p_7' + p_6 p_7' + p_7 = p_4 + p_5 + p_6 + p_7$$

$$z_2 = p_2 p_3' p_4' p_5' p_6' p_7' + p_3 p_4' p_5' p_6' p_7' + p_6 p_7' + p_7 = p_2 p_4' p_5' + p_3 p_4' p_5' + p_6 + p_7$$

$$z_1 = p_1 p_2' p_3' p_4' p_5' p_6' p_7' + p_3 p_4' p_5' p_6' p_7' + p_5 p_6' p_7' + p_7 = p_1 p_2' p_4' p_6' + p_3 p_4' p_6' + p_5 p_6' + p_7$$



# Priority Encoders



(c) Logic diagram.

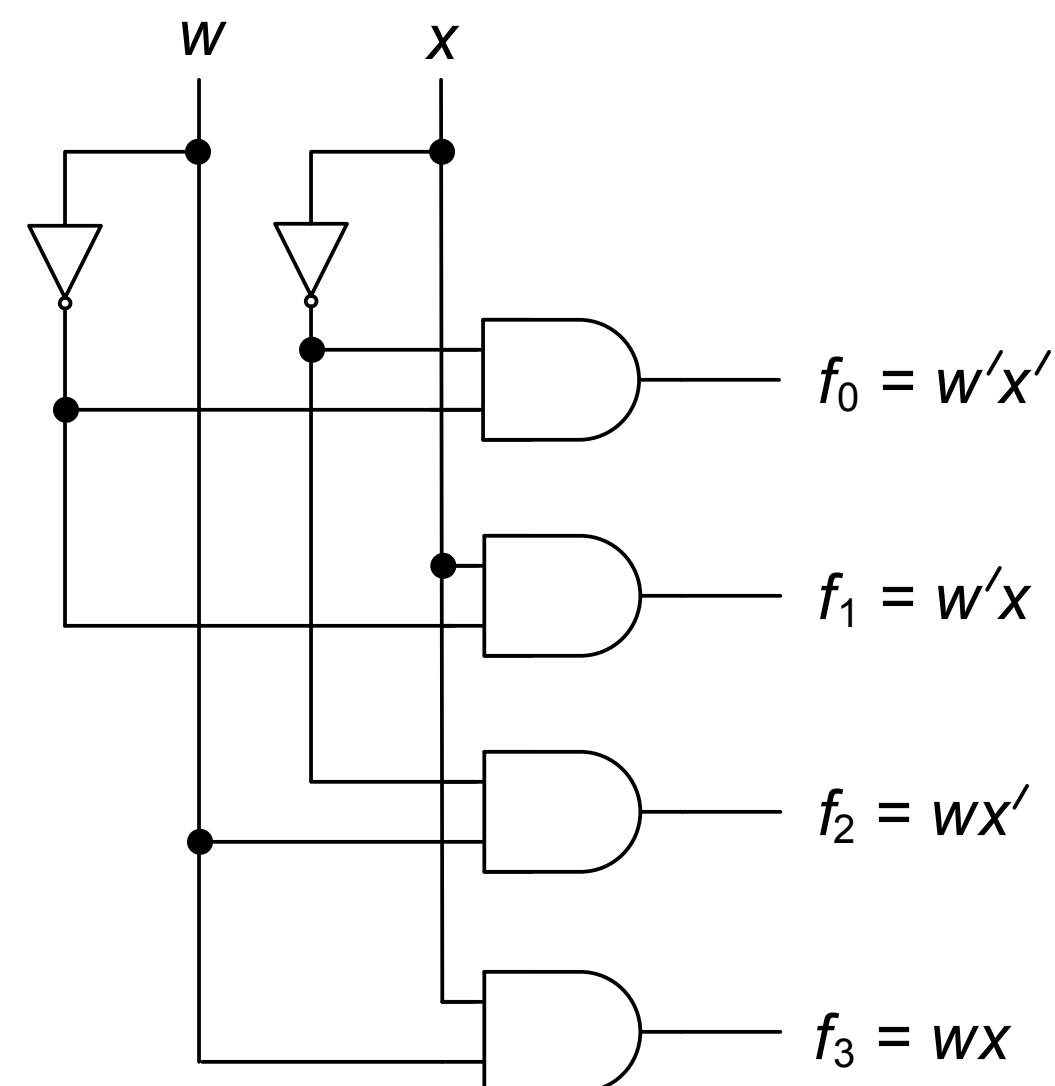
# Decoders

**Decoders with  $n$  inputs and  $2^n$  outputs:** for any input combination, only one output is 1

**Useful for:**

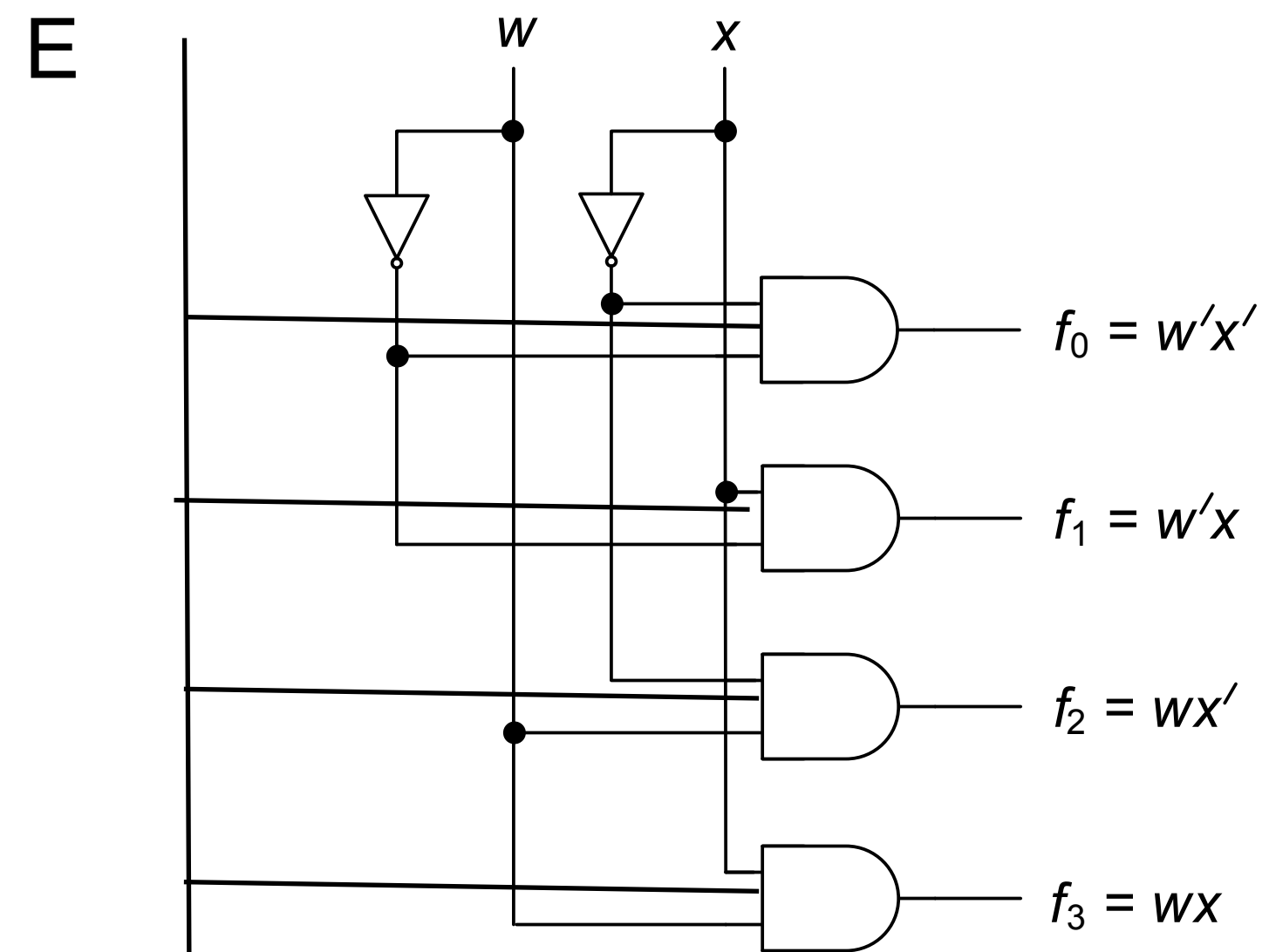
- Routing input data to a specified output line, e.g., in addressing memory
- Basic building blocks for implementing arbitrary switching functions
- Code conversion
- Data distribution

**Example: 2-to-4- decoder**

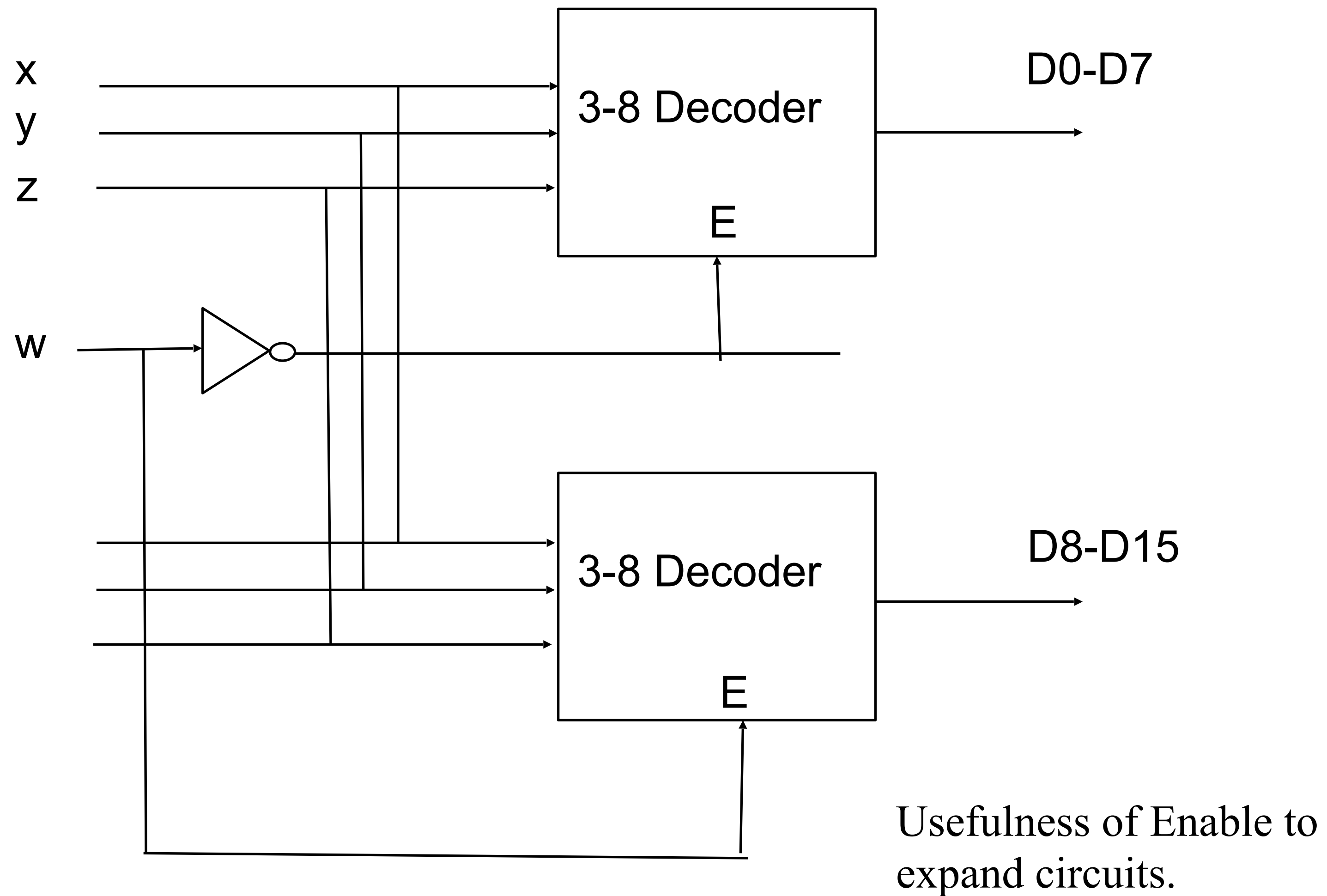




# Decoders with Enable



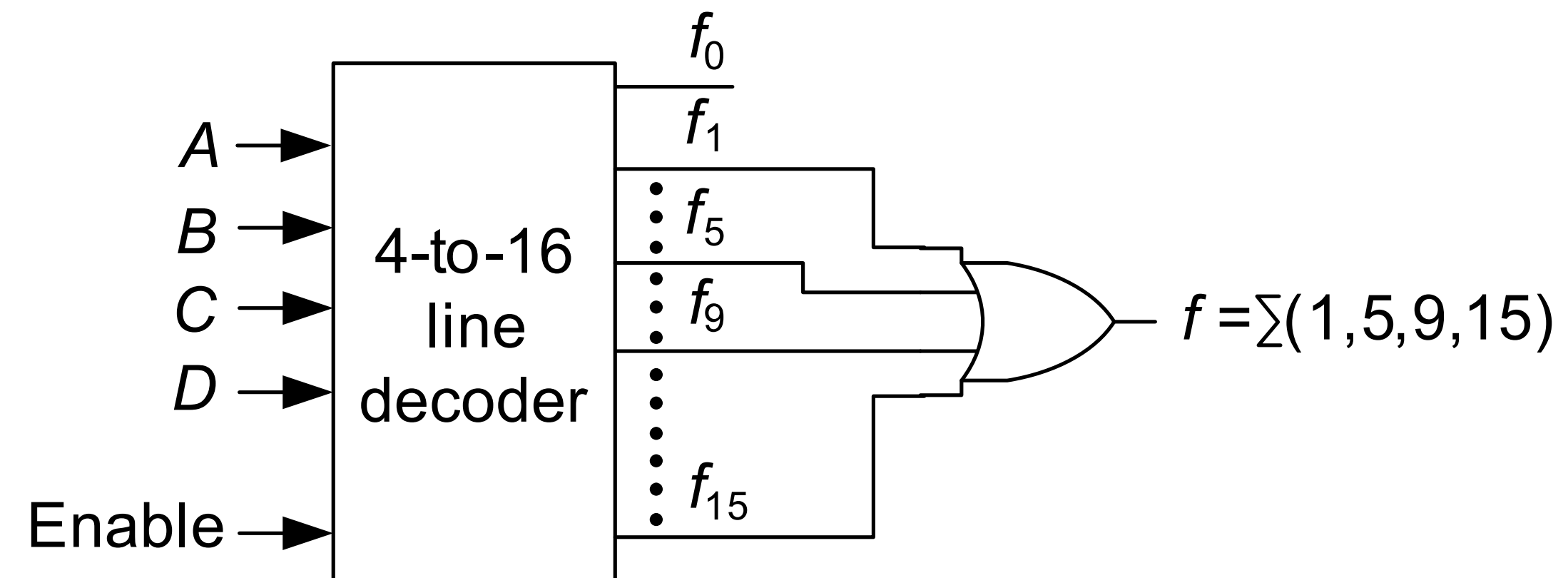
# 4-16 decoder using 3-8 decoder





# Realizing Arbitrary Functions

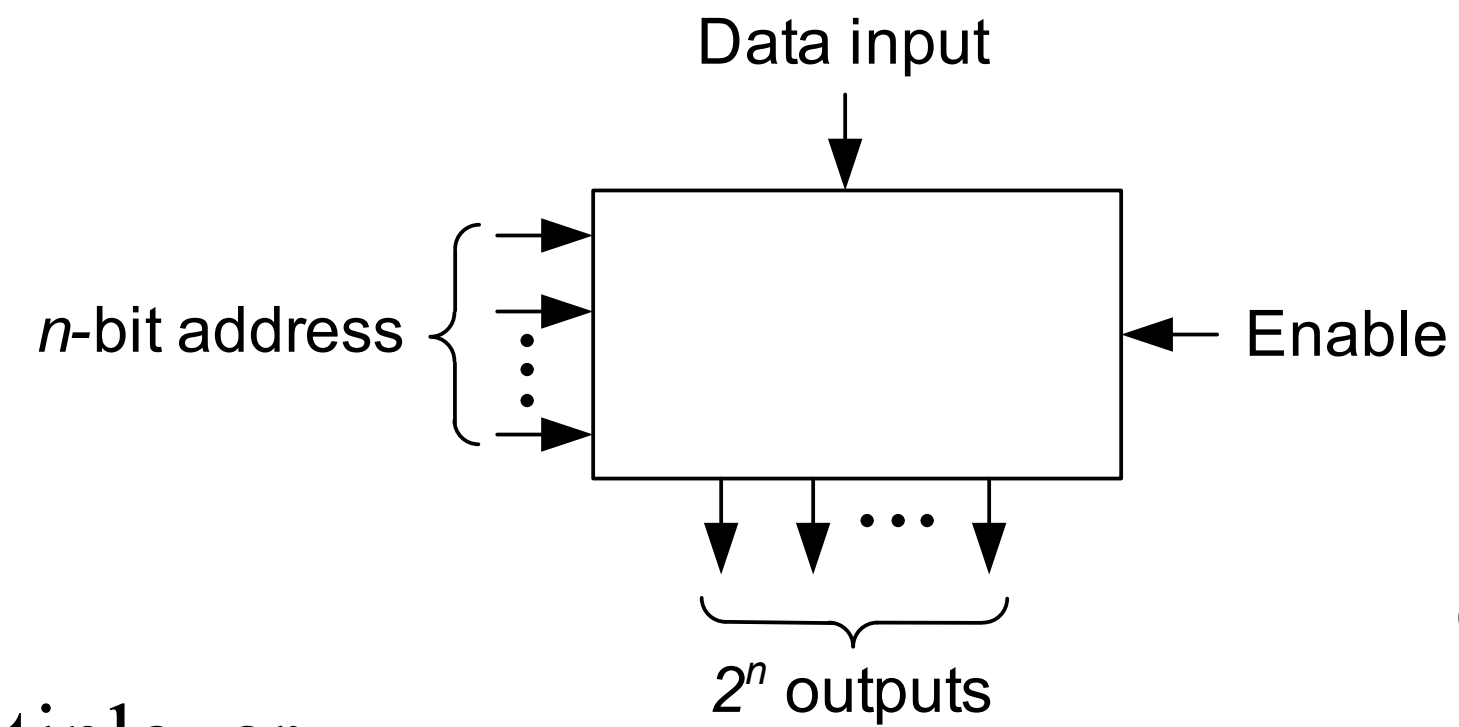
**Idea:** Realize a distinct minterm at each output



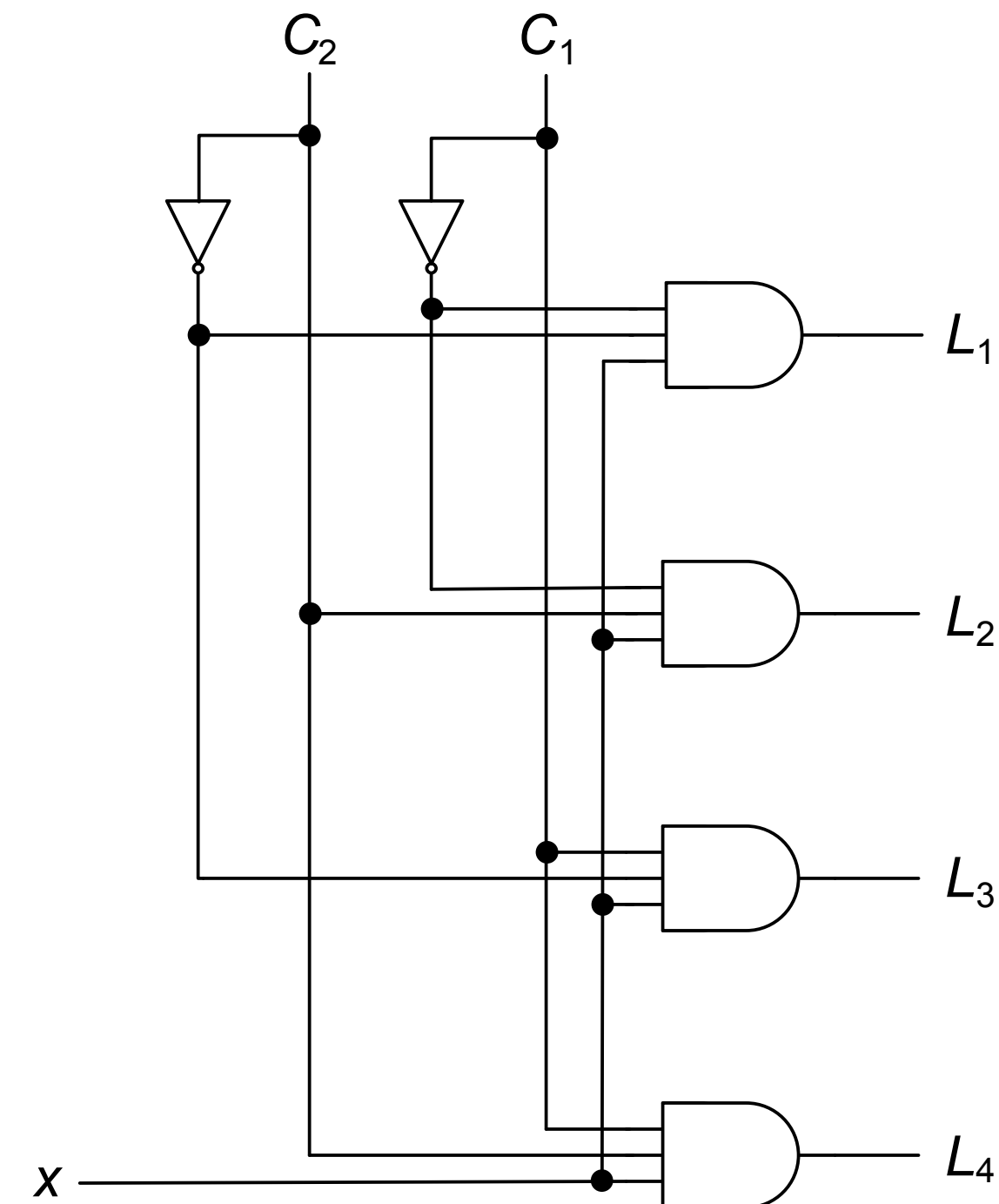
# Demultiplexer

**Demultiplexers:** decoder with 1 data input and  $n$  address inputs

- Directs input to any one of the  $2^n$  outputs



**Example:** A 4-output demultiplexer





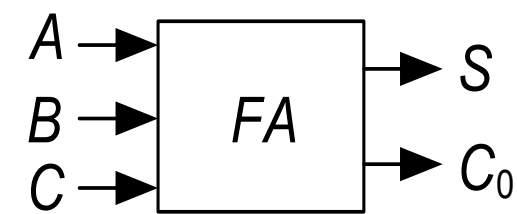
# Adders Again

**Full adder:** performs binary addition of three binary digits

- Inputs: arguments  $A$  and  $B$  and carry-in  $C$
- Outputs: sum  $S$  and carry-out  $C_0$

| $A$ | $B$ | $C$ | $S$ | $C_0$ |
|-----|-----|-----|-----|-------|
| 0   | 0   | 0   | 0   | 0     |
| 0   | 0   | 1   | 1   | 0     |
| 0   | 1   | 1   | 0   | 1     |
| 0   | 1   | 0   | 1   | 0     |
| 1   | 1   | 0   | 0   | 1     |
| 1   | 1   | 1   | 1   | 1     |
| 1   | 0   | 1   | 0   | 1     |
| 1   | 0   | 0   | 1   | 0     |

(a) Truth table for  $S$  and  $C_0$ .



(b) Block diagram.

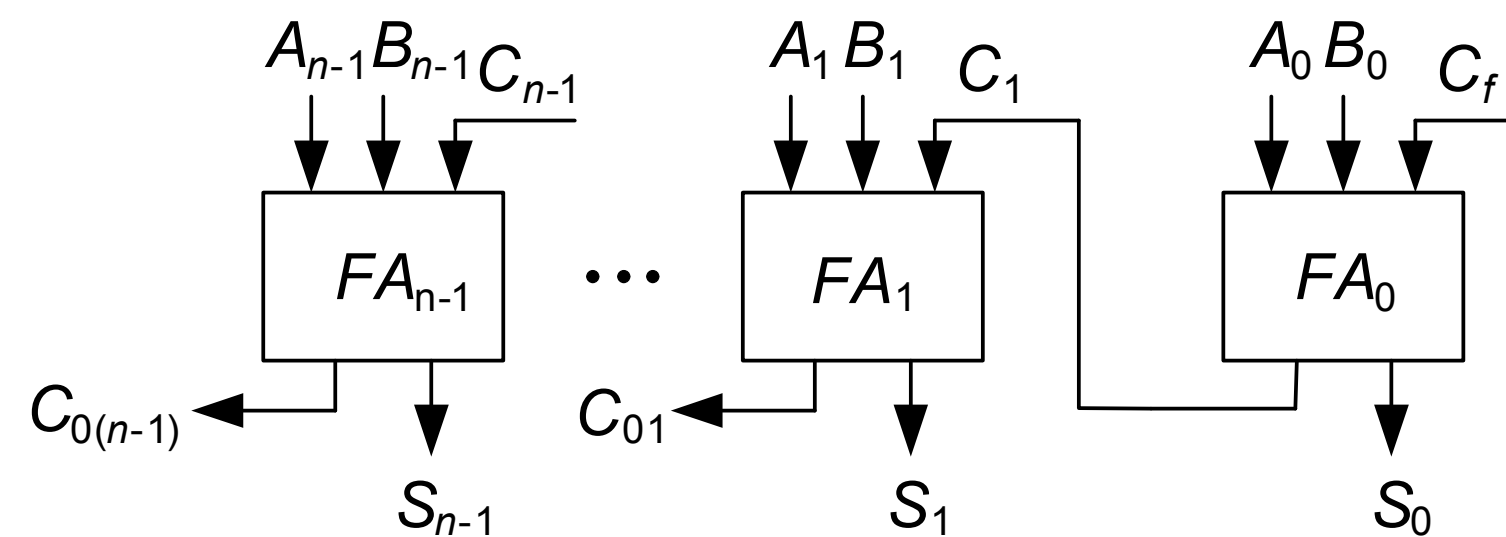
$$S = A \oplus B \oplus C$$

$$C_0 = AB + BC + CA$$

# Ripple Carry Adder

**Ripple-carry adder:** Stages of full adders

- $C_f$ : forced carry
- $C_{0(n-1)}$ : overflow carry



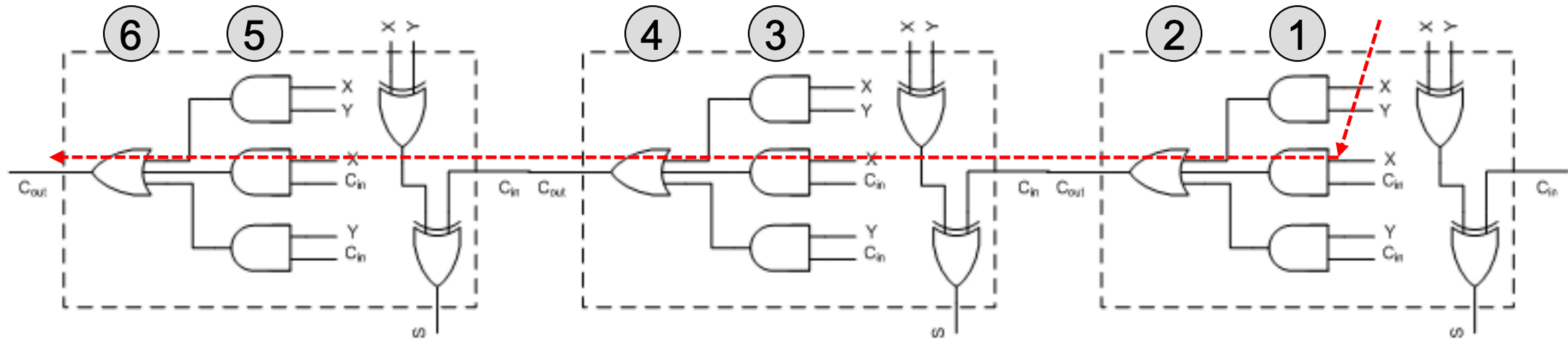
$$S_i = A_i \oplus B_i \oplus C_i$$

$$C_{0i} = A_i B_i + B_i C_{0i} + C_{0i} A_i$$

**Time required:**

- **Carry propagation takes longest time** — in the worst case, the carry propagates through all the stages
- Time per full adder: 2 units (assuming each gate takes one unit of time)
  - Time for carry generation
  - Assumption: two level circuit realisation with 2 input gates
- Time for ripple-carry adder:  **$2n$  units**

# Ripple Carry Adder





# Carry Lookahead Adder

**Carry-lookahead adder:** several stages simultaneously examined and their carries generated in parallel

- Generate signal  $D_i = A_i B_i$
- Propagate signal  $T_i = A_i \oplus B_i$
- Thus,  $C_{0i} = D_i + T_i C_i$

**To generate carries in parallel:** convert recursive form to nonrecursive

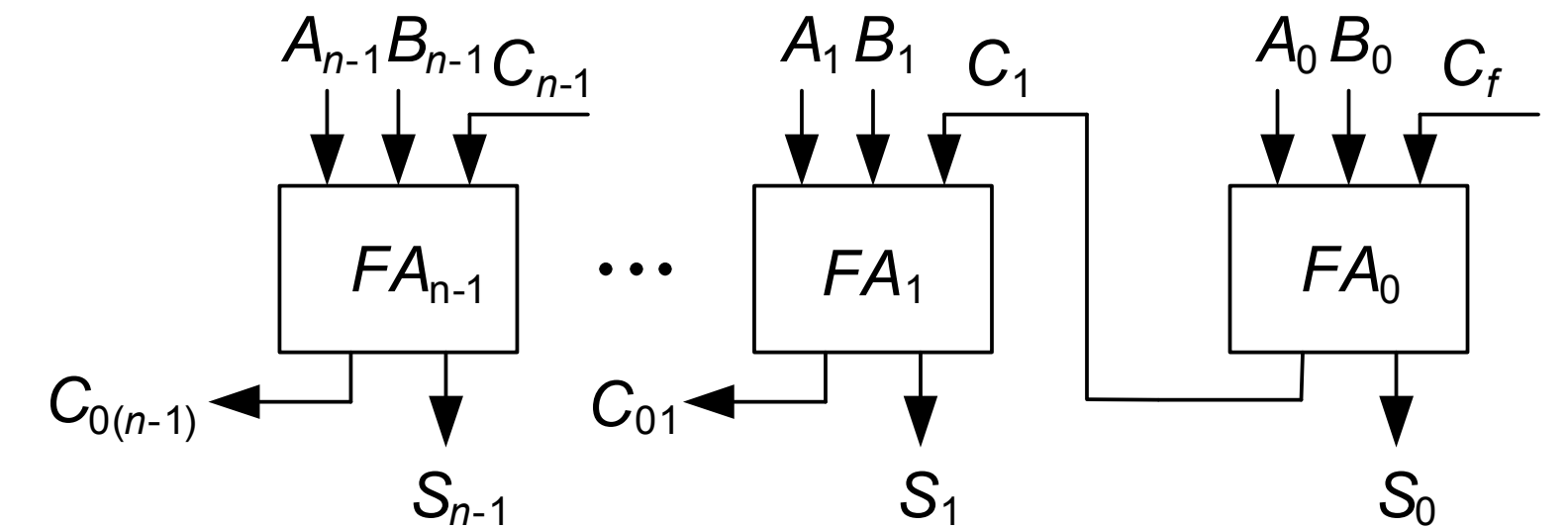
$$C_{0i} = D_i + T_i C_i$$

$$C_i = C_{0(i-1)}$$

$$\begin{aligned} C_{0i} &= D_i + T_i(D_{i-1} + T_{i-1}C_{i-1}) \\ &= D_i + T_i D_{i-1} + T_i T_{i-1}(D_{i-2} + T_{i-2}C_{i-2}) \\ &= D_i + T_i D_{i-1} + T_i T_{i-1} D_{i-2} + T_i T_{i-1} T_{i-2} C_{i-2} \end{aligned}$$

... ..

$$C_{0i} = D_i + T_i D_{i-1} + T_i T_{i-1} D_{i-2} + \dots + T_i T_{i-1} T_{i-2} \dots T_0 C_f$$



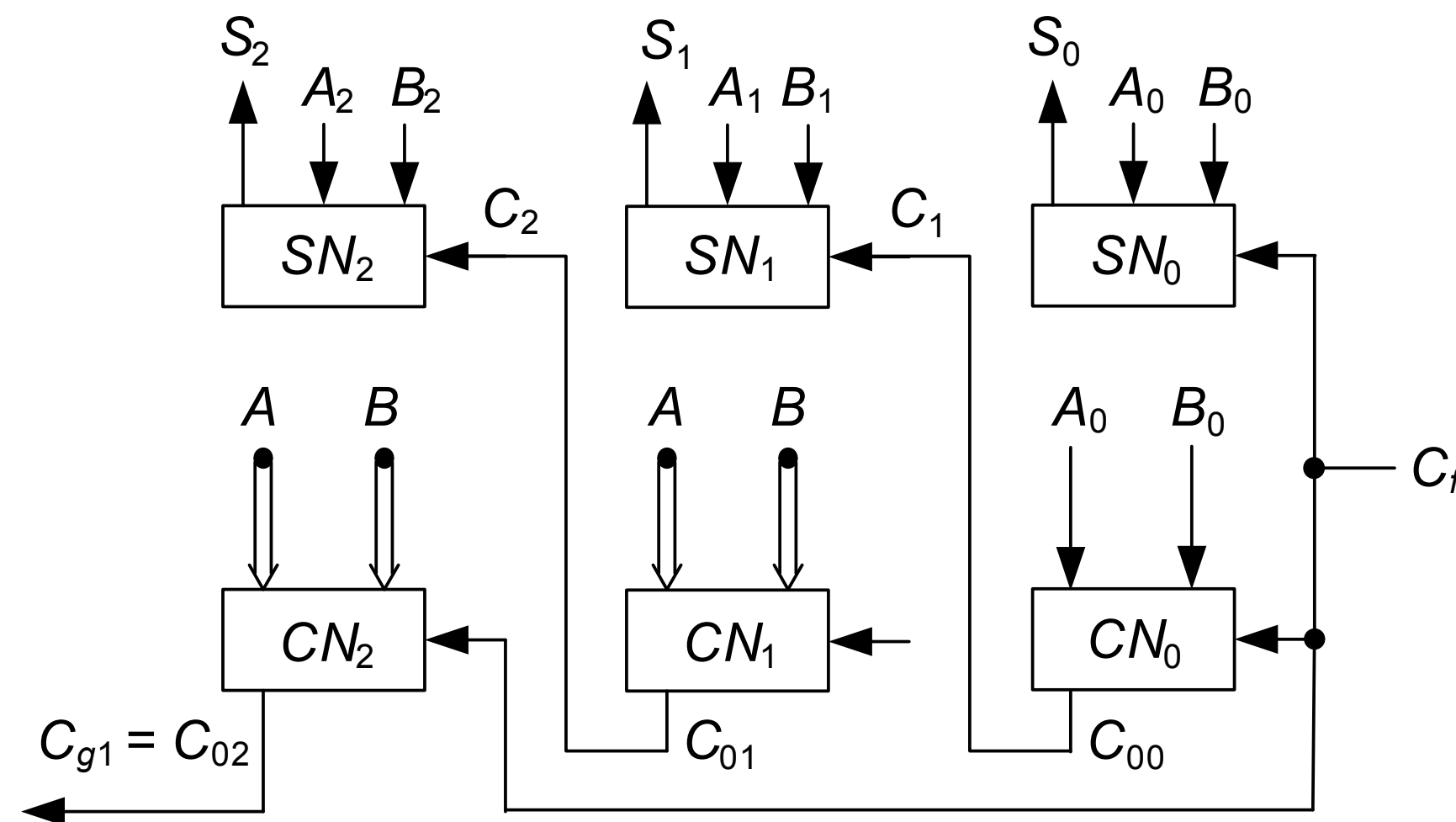
Thus,  $C_{0i} = 1$  if it has been generated in the  $i^{\text{th}}$  stage or originated in a preceding stage and propagated to all subsequent stages

# Carry Lookahead Adder

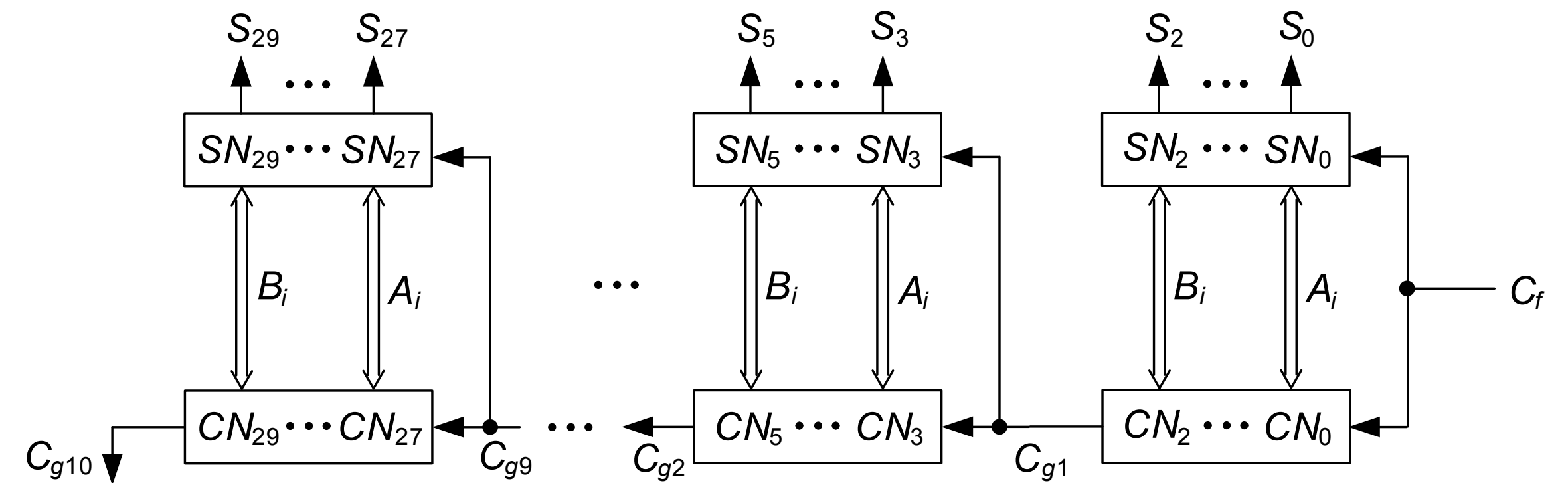
**Implementation of lookahead for the complete adder impractical:**

- Divide the  $n$  stages into groups
- Full carry lookahead within group
- Ripple carry between groups

**Example:** Three-digit adder group with full carry lookahead



(a) Block diagram of initial three-stage group

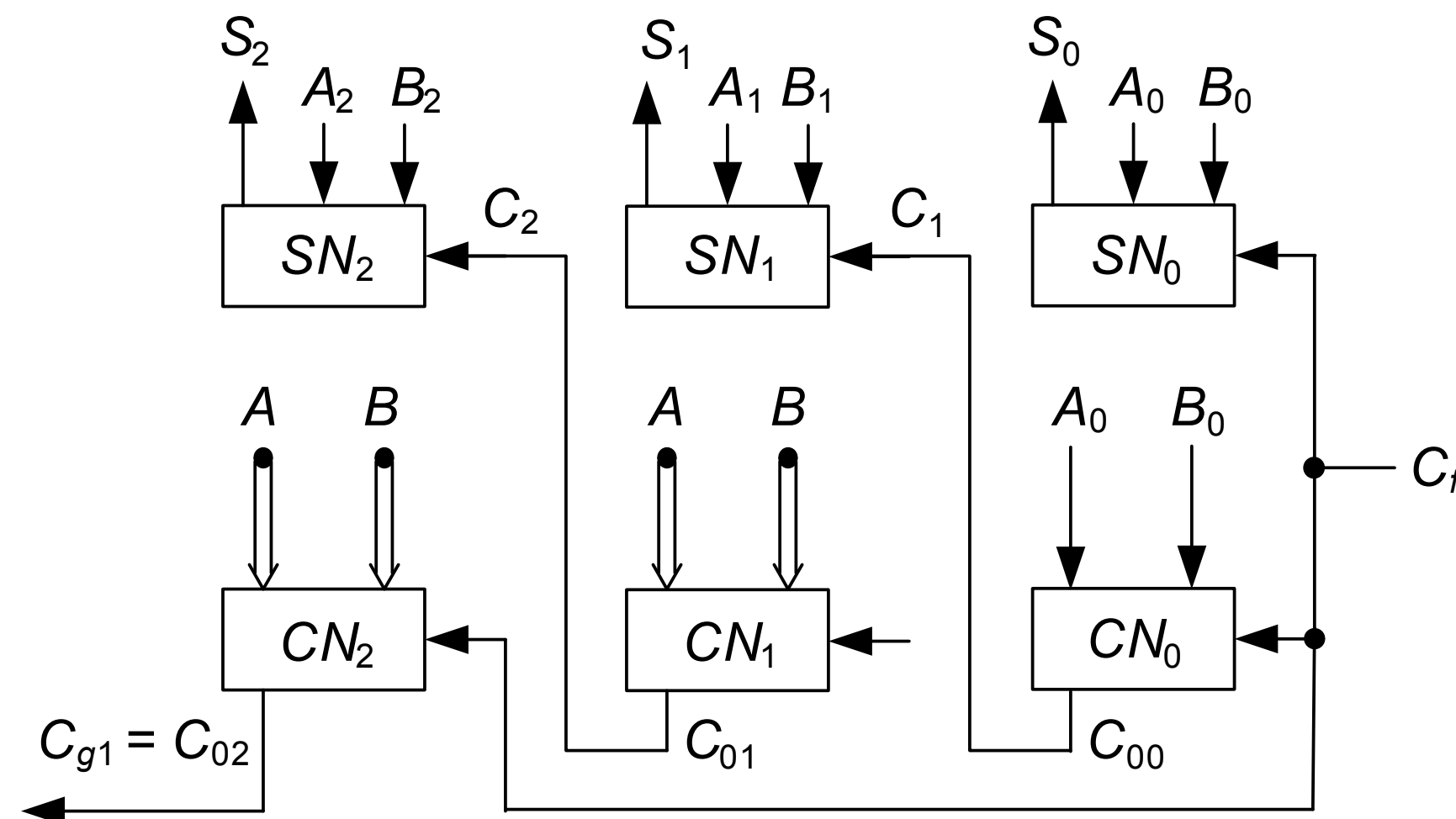


# Carry Lookahead Adder

**Implementation of lookahead for the complete adder impractical:**

- Divide the  $n$  stages into groups of 3-bit adders
- Full carry lookahead within group
- Ripple carry between groups

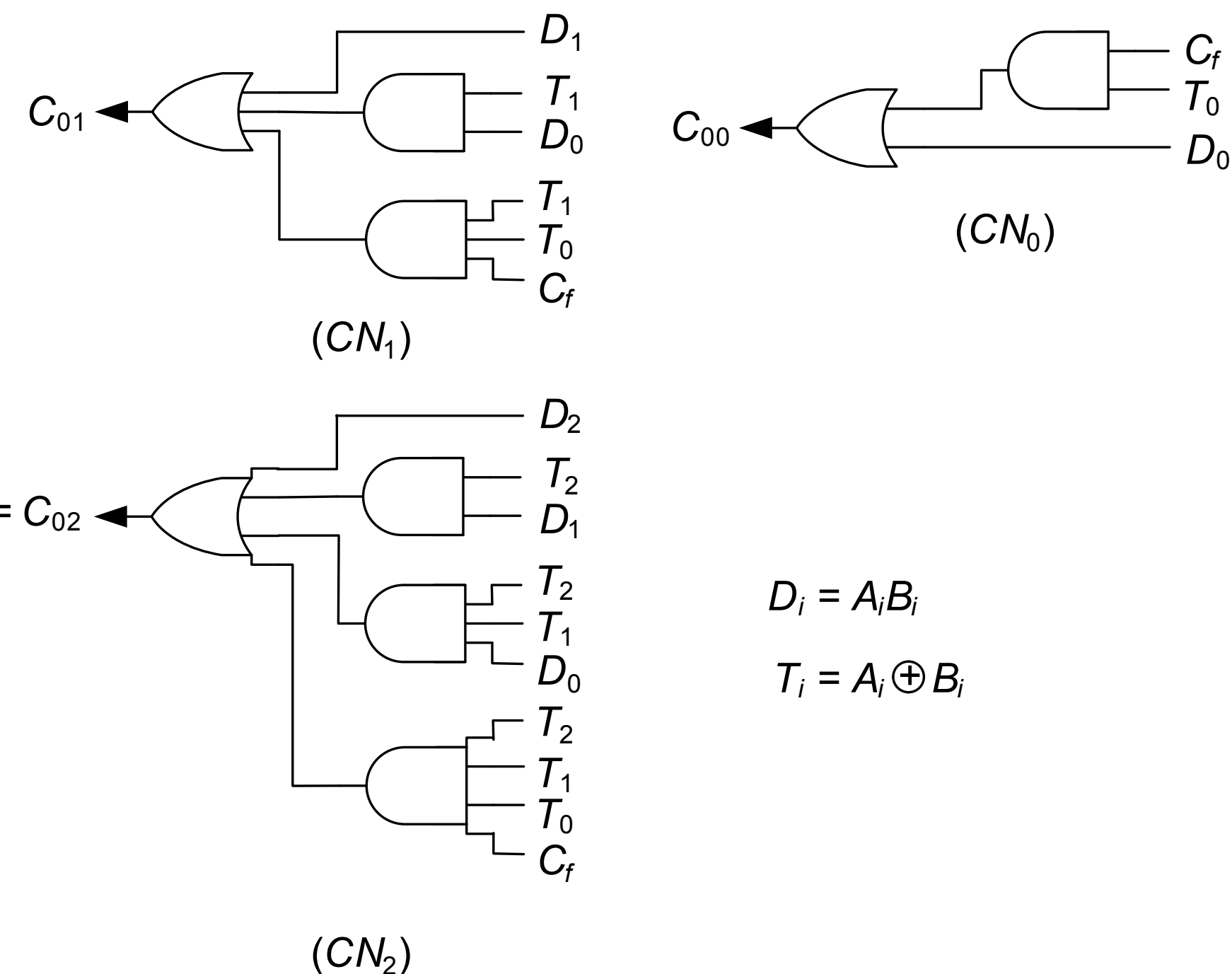
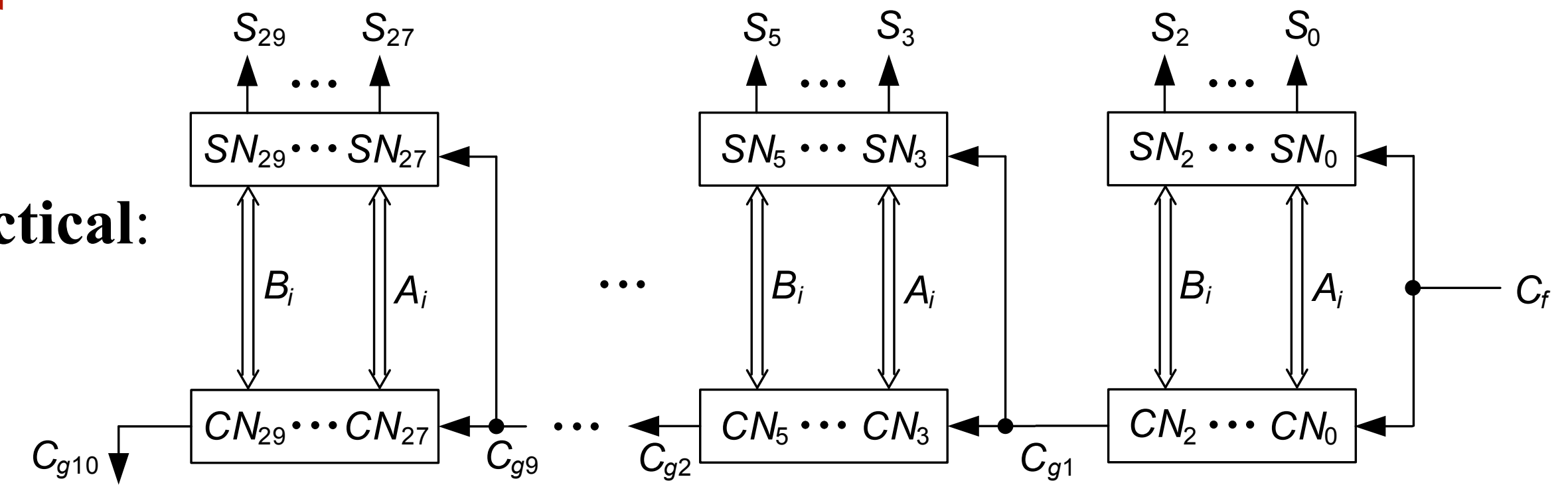
**Example:** Three-digit adder group with full carry lookahead



(a) Block diagram of initial three-stage group

**Time taken:**

- 4 time units for  $C_{g1}$  **why 4?**
- Only 2 time units for  $C_{g2}$  and other group carries **why?**



(b) The carry networks

$$D_i = A_i B_i$$

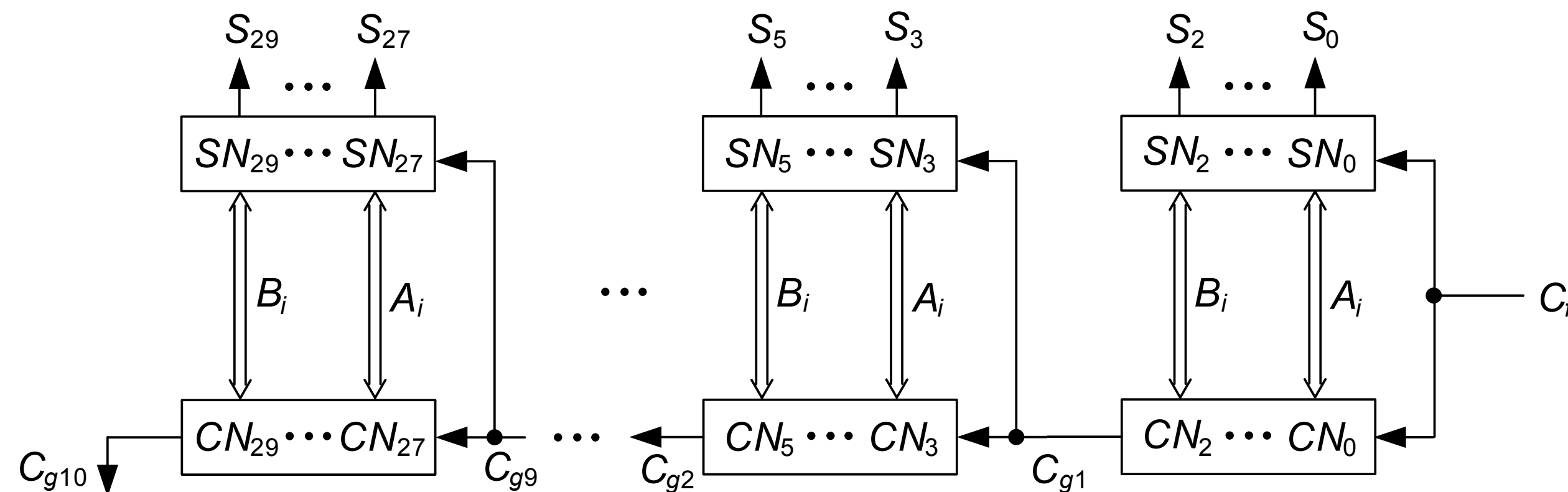
$$T_i = A_i \oplus B_i$$



# Carry Lookahead Adder

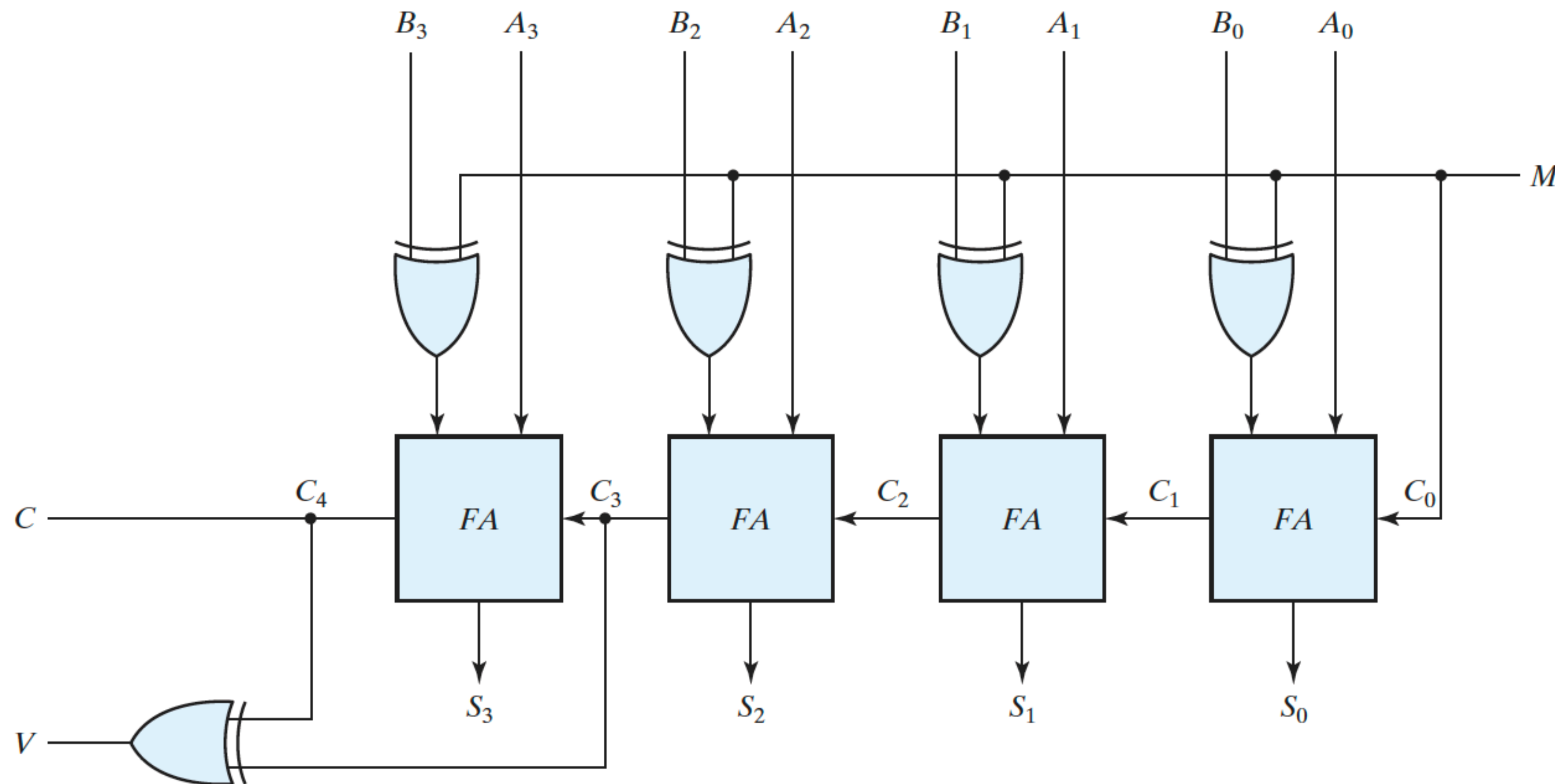
**Example:** divide  $n$  stages into groups of **three bits** for a **30 bit adder**

- **Very very important!!!**
  - First stage requires more time (2 units extra to generate  $T_i$ ).
  - Last stage requires more time (2 extra units for the final sum, for the other units it's not coming in the critical path)
- **Time taken:**  $4 + 2n/3$  time units
- 50% additional hardware for a threefold speedup



# Adder Subtractor

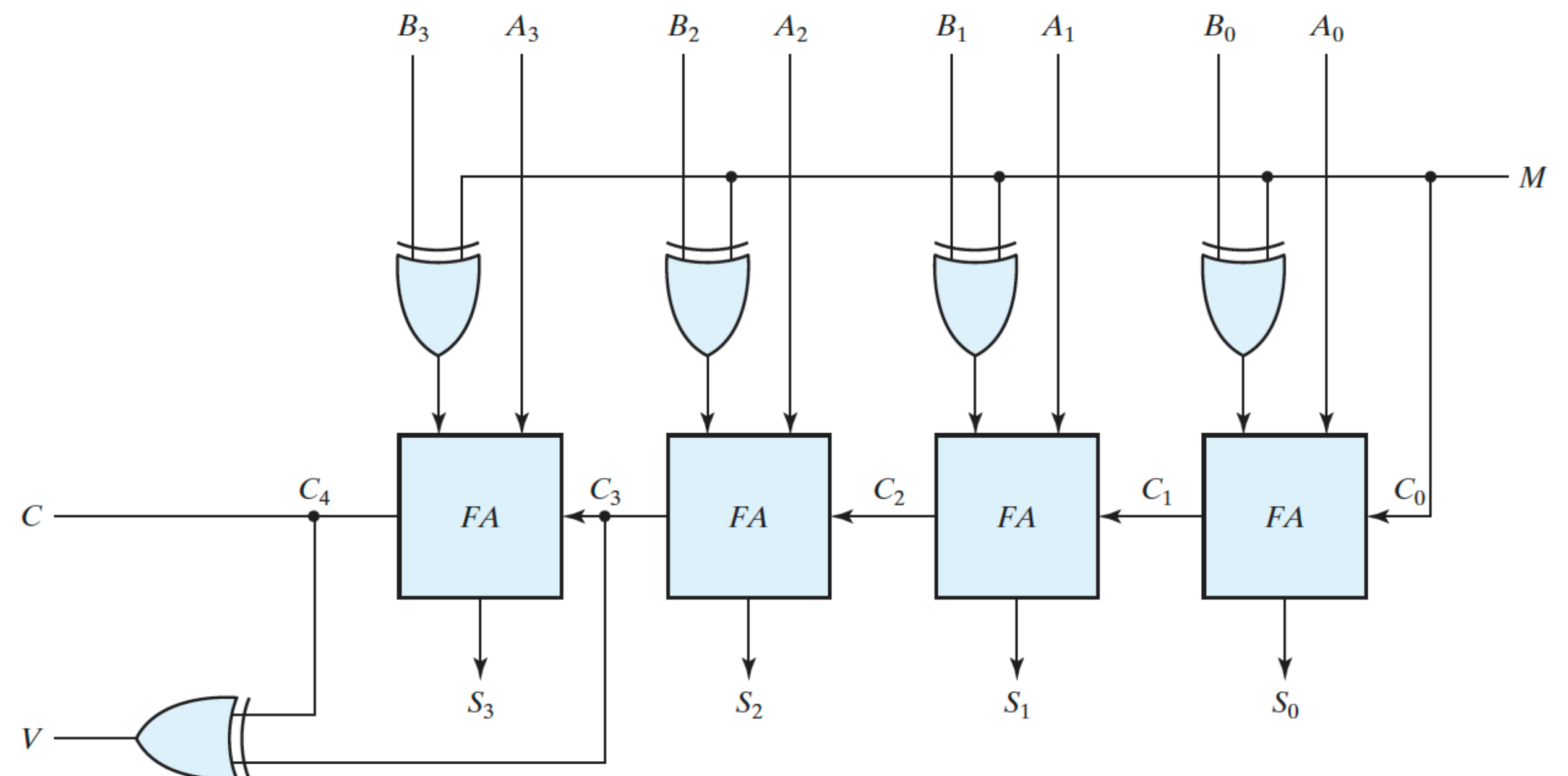
- Implements two's complement subtraction
- The idea is to have both an adder and subtractor in the same circuit
- **Overflow detection**



# Adder Subtractor: The Overflow

- **Overflow:** When two numbers with  $n$  digits each are added and the sum is a number occupying  $n + 1$  digits, we say that an overflow occurred.
- This is true for binary or decimal numbers, signed or unsigned
- An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. If these two carries are not equal, an overflow has occurred.
- If the numbers are unsigned
  - C bit detects the overflow
- If the numbers are signed
  - V bit detects the overflow

|          |           |          |           |
|----------|-----------|----------|-----------|
| carries: | 0 1       | carries: | 1 0       |
| +70      | 0 1000110 | -70      | 1 0111010 |
| +80      | 0 1010000 | -80      | 1 0110000 |
| +150     | 1 0010110 | -150     | 0 1101010 |





# BCD Adder

- Adds two BCD Numbers
- First add in binary and then convert the sum to BCD
- Values beyond 9 requires “correction”
  - For values beyond 9, 0110 needs to be added to generate the BCD encoded sum
- We use a bit C to detect when correction is needed
- $C = 1$ , when
  - Carry K or the binary sum is 1
  - $Z_8 = 1$  and  $Z_4 = 1$
  - $Z_8 = 1$  and  $Z_2 = 1$
- $C = K + Z_8Z_4 + Z_8Z_2$

| Binary Sum |                |                |                |                | BCD Sum |                |                |                |                | Decimal |
|------------|----------------|----------------|----------------|----------------|---------|----------------|----------------|----------------|----------------|---------|
| K          | Z <sub>8</sub> | Z <sub>4</sub> | Z <sub>2</sub> | Z <sub>1</sub> | C       | S <sub>8</sub> | S <sub>4</sub> | S <sub>2</sub> | S <sub>1</sub> |         |
| 0          | 0              | 0              | 0              | 0              | 0       | 0              | 0              | 0              | 0              | 0       |
| 0          | 0              | 0              | 0              | 1              | 0       | 0              | 0              | 0              | 1              | 1       |
| 0          | 0              | 0              | 1              | 0              | 0       | 0              | 0              | 1              | 0              | 2       |
| 0          | 0              | 0              | 1              | 1              | 0       | 0              | 0              | 1              | 1              | 3       |
| 0          | 0              | 1              | 0              | 0              | 0       | 0              | 1              | 0              | 0              | 4       |
| 0          | 0              | 1              | 0              | 1              | 0       | 0              | 1              | 0              | 1              | 5       |
| 0          | 0              | 1              | 1              | 0              | 0       | 0              | 1              | 1              | 0              | 6       |
| 0          | 0              | 1              | 1              | 1              | 0       | 0              | 1              | 1              | 1              | 7       |
| 0          | 1              | 0              | 0              | 0              | 0       | 1              | 0              | 0              | 0              | 8       |
| 0          | 1              | 0              | 0              | 1              | 0       | 1              | 0              | 0              | 1              | 9       |
| 0          | 1              | 0              | 1              | 0              | 1       | 0              | 0              | 0              | 0              | 10      |
| 0          | 1              | 0              | 1              | 1              | 1       | 0              | 0              | 0              | 1              | 11      |
| 0          | 1              | 1              | 0              | 0              | 1       | 0              | 0              | 1              | 0              | 12      |
| 0          | 1              | 1              | 0              | 1              | 1       | 0              | 0              | 1              | 1              | 13      |
| 0          | 1              | 1              | 1              | 0              | 1       | 0              | 1              | 0              | 0              | 14      |
| 0          | 1              | 1              | 1              | 1              | 1       | 0              | 1              | 0              | 1              | 15      |
| 1          | 0              | 0              | 0              | 0              | 1       | 0              | 1              | 1              | 0              | 16      |
| 1          | 0              | 0              | 0              | 1              | 1       | 0              | 1              | 1              | 1              | 17      |
| 1          | 0              | 0              | 1              | 0              | 1       | 1              | 0              | 0              | 0              | 18      |
| 1          | 0              | 0              | 1              | 1              | 1       | 1              | 0              | 0              | 1              | 19      |

# BCD Adder

