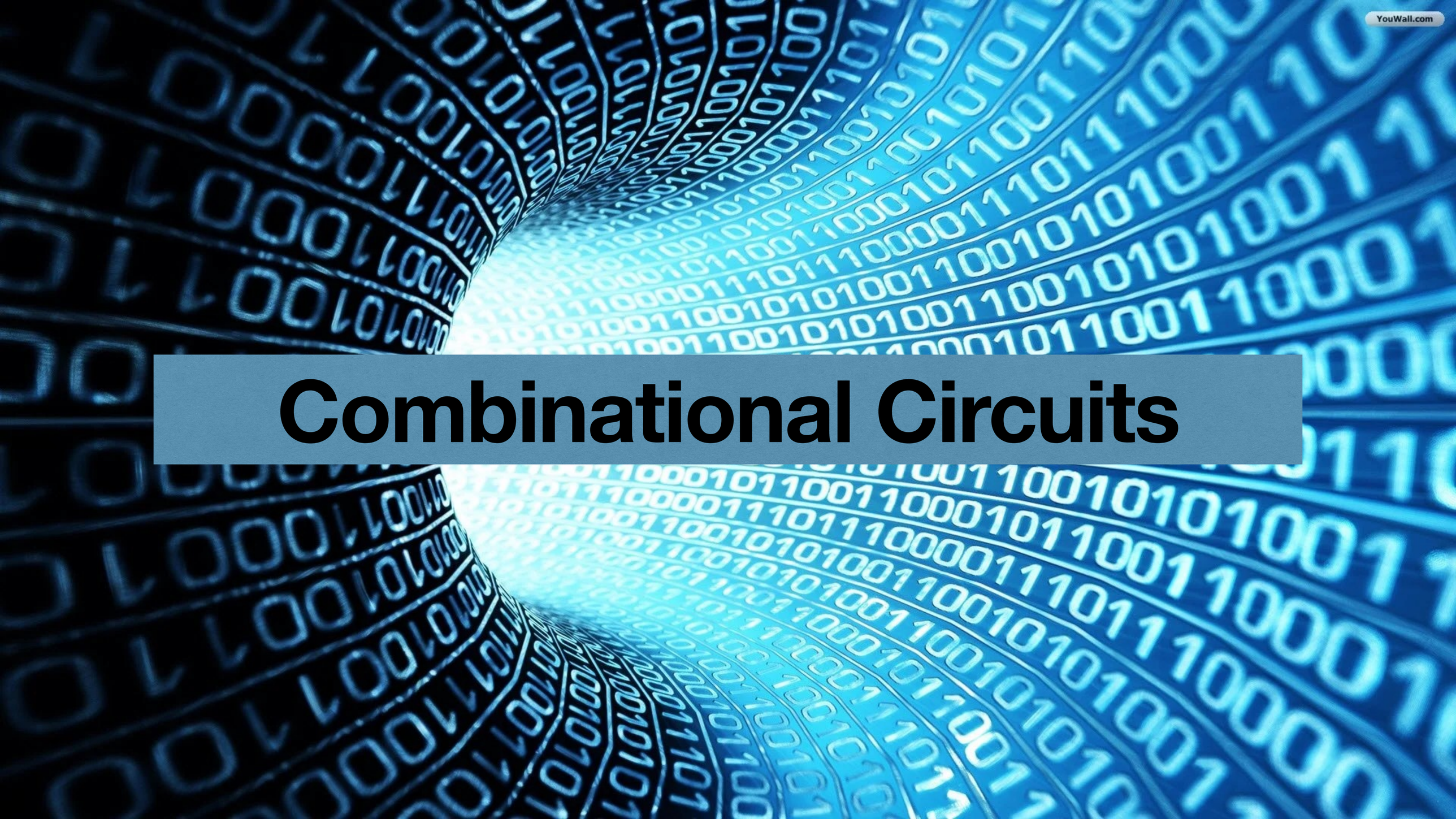
Digital Logic Design + Computer Architecture

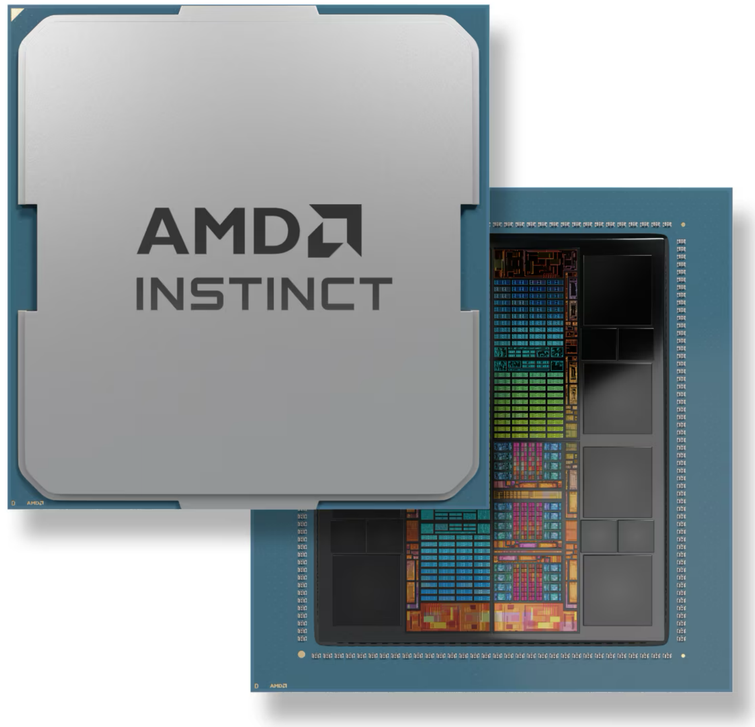
Sayandeep Saha

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

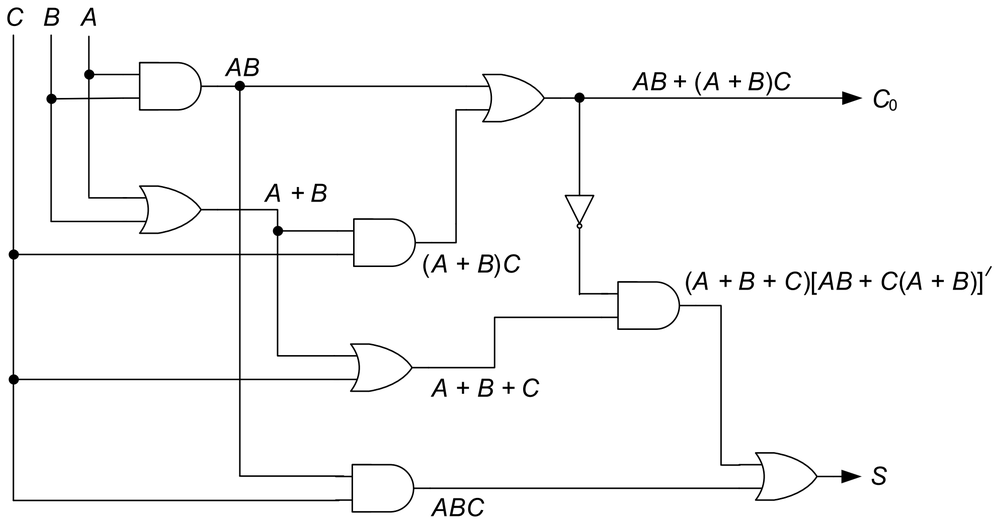
- **Logic gates**: perform logical operations on input signals
- **Positive (negative) logic polarity**: constant 1 (0) denotes a high voltage and constant 0 a low (high) voltage
- **Combinational circuits:** No memorization
- **Synchronous sequential circuits**: have memory; driven by a clock that produces 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



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

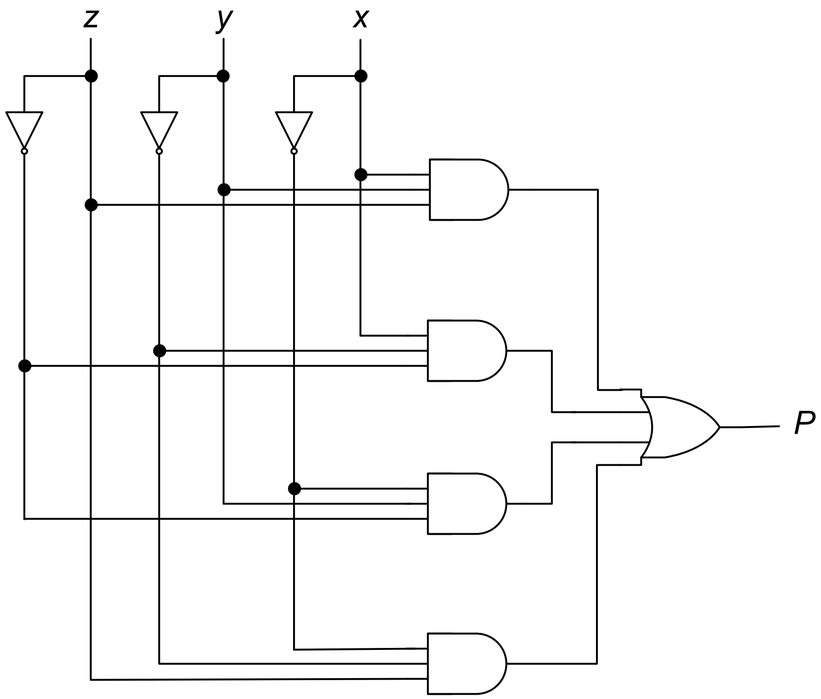
$$S = (A + B + C)[AB + (A + B)C]' + ABC$$
$$= (A + B + C)(A' + B')(A' + C')(B' + C') + ABC$$
$$= AB'C' + A'BC' + A'B'C + ABC$$
$$= A \oplus B \oplus C$$

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

z \ xy	00	01	11	10
0	0	1	0	1
1	1	0	1	0

(*a*) Map.



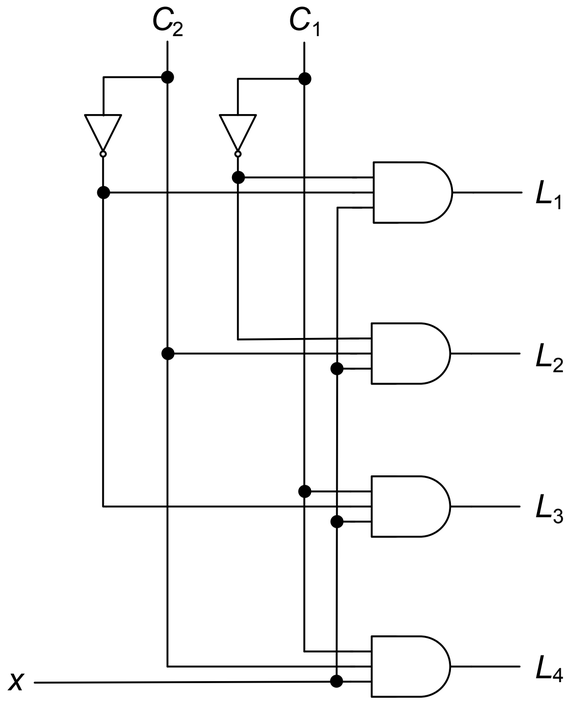
(*b*) Implementation.

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

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

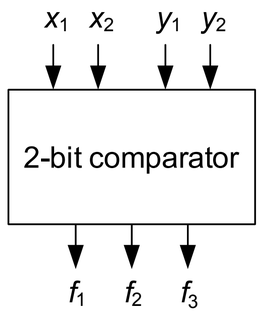
Control		*Output lines*				*Logic equations*
C_1	C_2	L_1	L_2	L_3	L_4	
0	0	x	0	0	0	$L_1 = xC_1'C_2'$
0	1	0	x	0	0	$L_2 = xC_1'C_2$
1	0	0	0	x	0	$L_3 = xC_1C_2'$
1	1	0	0	0	x	$L_4 = xC_1C_2$



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 \ x_1x_2	00	01	11	10
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.

$$f_1 = ?$$

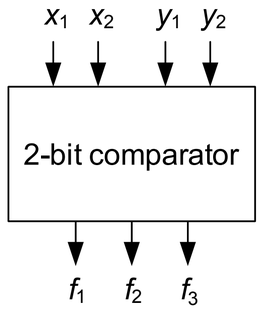
$$f_2 = ?$$

$$f_3 = ?$$

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.

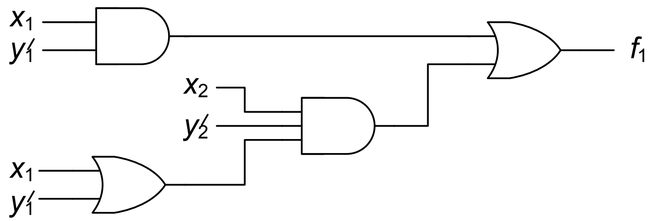
y_1y_2 \ x_1x_2	00	01	11	10
00	2	1	1	1
01	3	2	1	1
11	3	3	2	3
10	3	3	1	2

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

$$f_1 = x_1x_2y_2' + x_2y_1'y_2' + x_1y_1'$$
$$= (x_1 + y_1')x_2y_2' + x_1y_1'$$

$$f_2 = x_1'x_2'y_1'y_2' + x_1'x_2y_1'y_2 + x_1x_2'y_1y_2' + x_1x_2y_1y_2$$
$$= x_1'y_1'(x_2'y_2' + x_2y_2) + x_1y_1(x_2'y_2' + x_2y_2)$$
$$= (x_1'y_1' + x_1y_1)(x_2'y_2' + x_2y_2)$$

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



(*c*) Circuit for f_1.

Combinational Circuits: Comparators

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

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

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

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

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

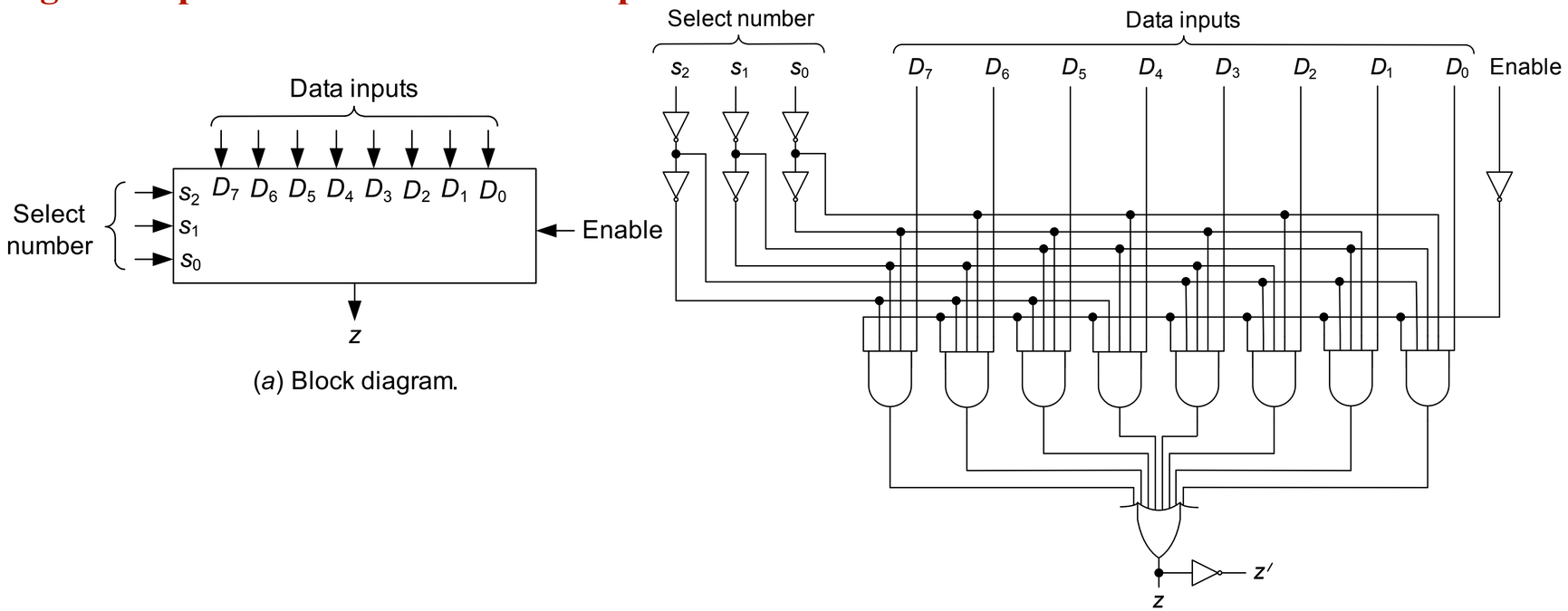
Combinational Circuits: Multiplexers

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

Data selector: application of multiplexer

- n data input lines, $D_0, D_1, \ldots, D_{n-1}$
- m select digit inputs $s_0, s_1, \ldots, s_{m-1}$
- 1 output

- **Can you design a simple data selectors with 2 input data lines?**



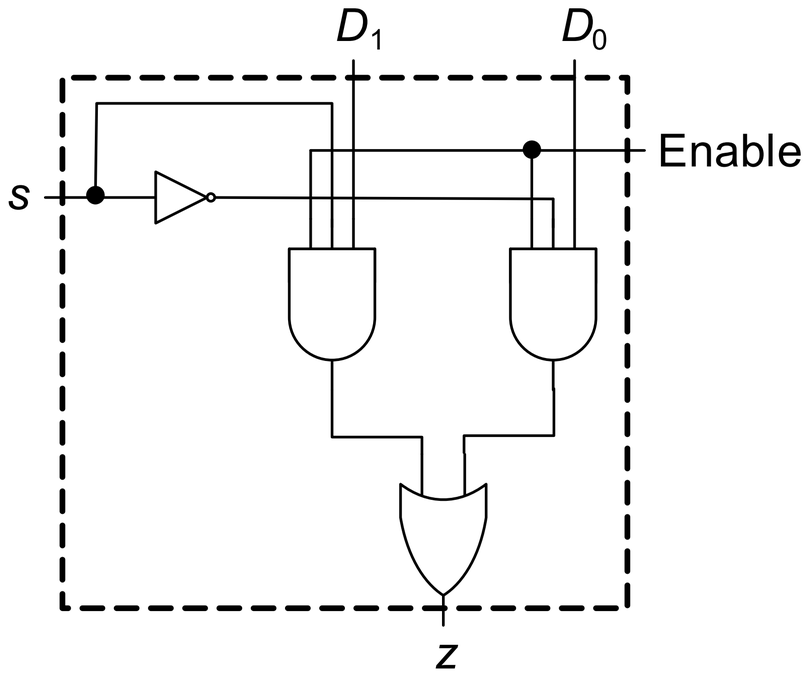
(*a*) Block diagram.

(*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:

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

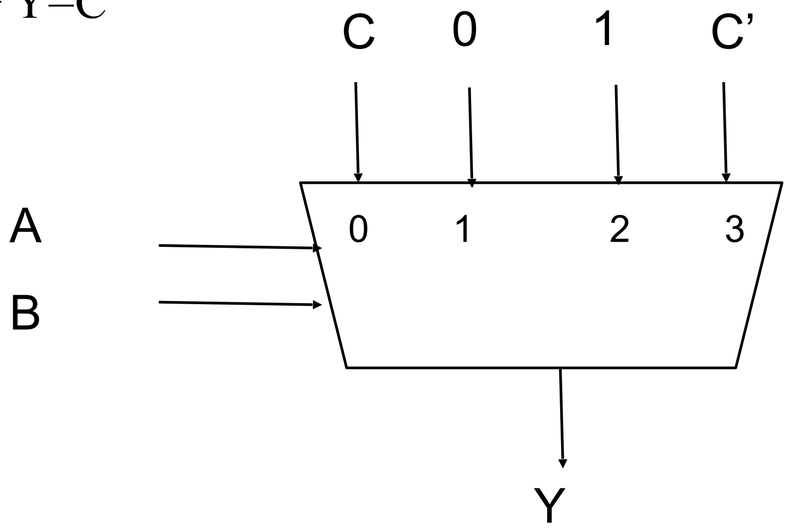
Example: $s_1 = A$, $s_2 = B$, $D_0 = C$, $D_1 = 1$, $D_2 = 0$, $D_3 = C'$

$$z = A'B'C + AB' + ABC'$$
$$= AC' + B'C$$

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 => Y=C
  - A,B=0,1 => Y=0
  - A,B=1,0 => Y=C’+C=1
  - A,B=1,1 => Y=C’

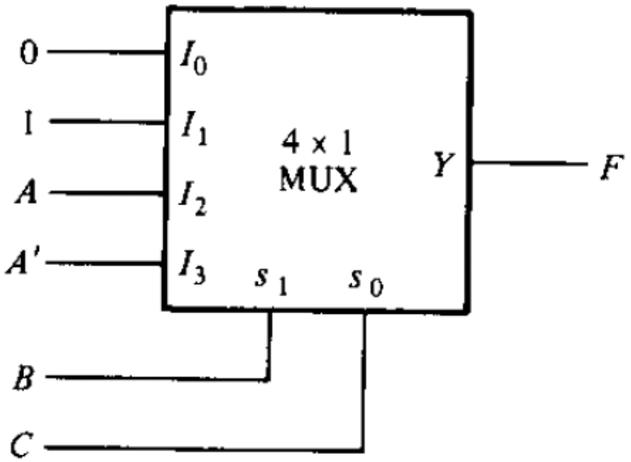


Implementing Switching Function with Mux

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

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 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

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 adder

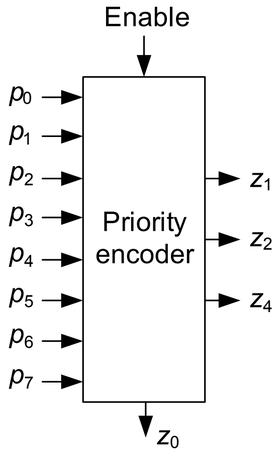
Try it yourself…!!!

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
ϕ	1	0	0	0	0	0	0	0	0	1
ϕ	ϕ	1	0	0	0	0	0	0	1	0
ϕ	ϕ	ϕ	1	0	0	0	0	0	1	1
ϕ	ϕ	ϕ	ϕ	1	0	0	0	1	0	0
ϕ	ϕ	ϕ	ϕ	ϕ	1	0	0	1	0	1
ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	1	0	1	1	0
ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	1	1	1	1

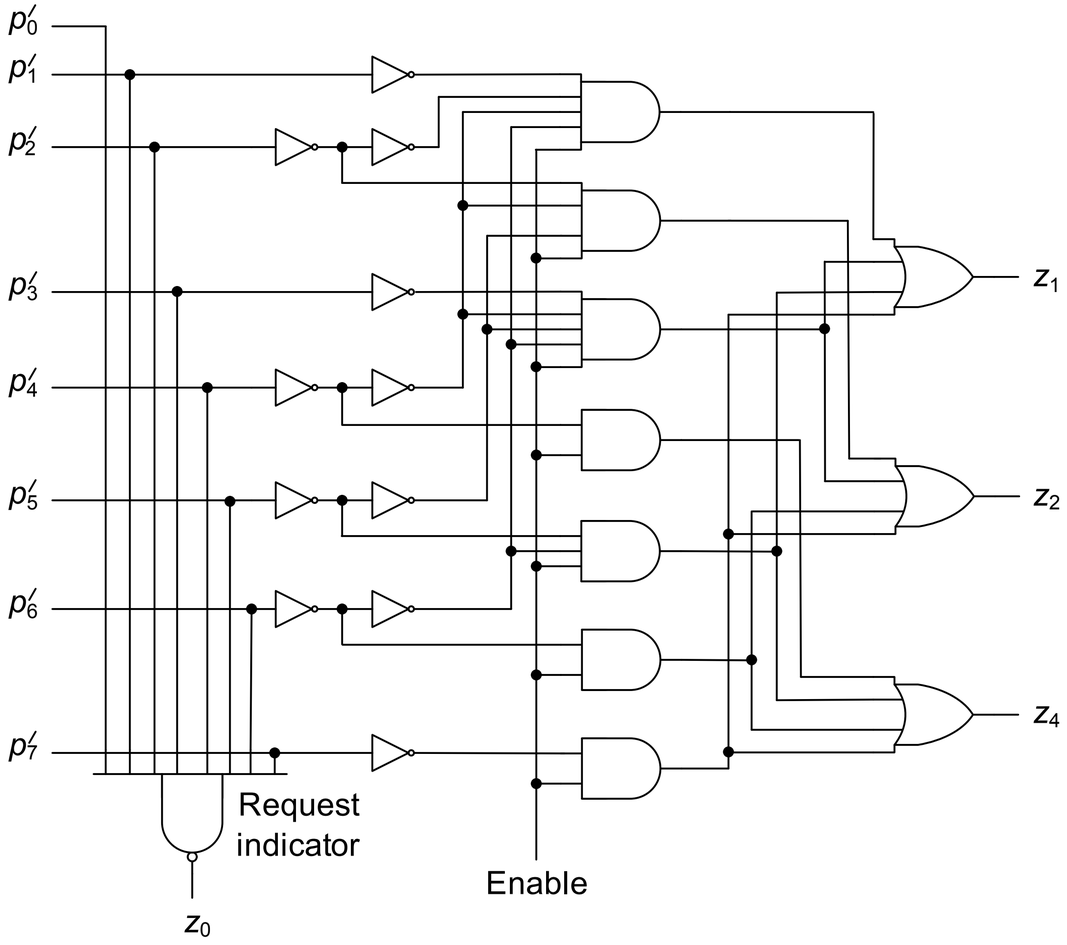
(*b*) Truth table.

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

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

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

Priority Encoders



(*c*) Logic diagram.

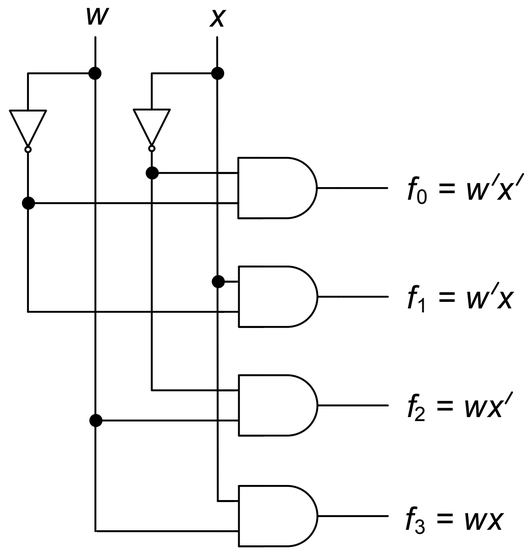
Decoders

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

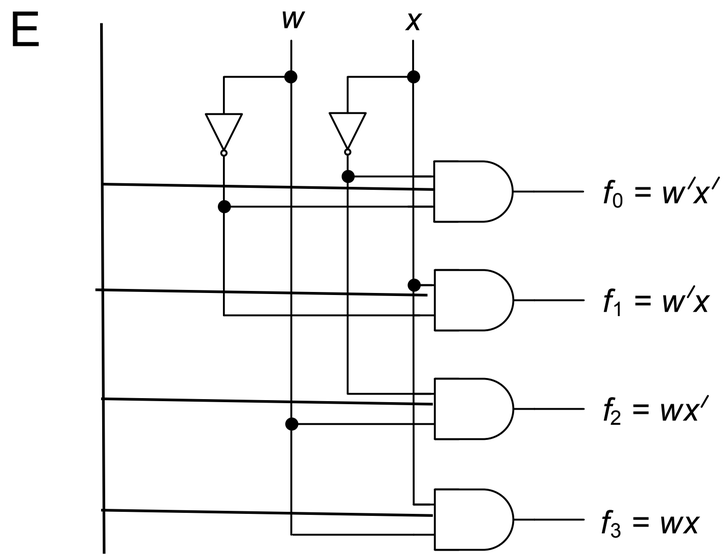
Useful for:

- 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

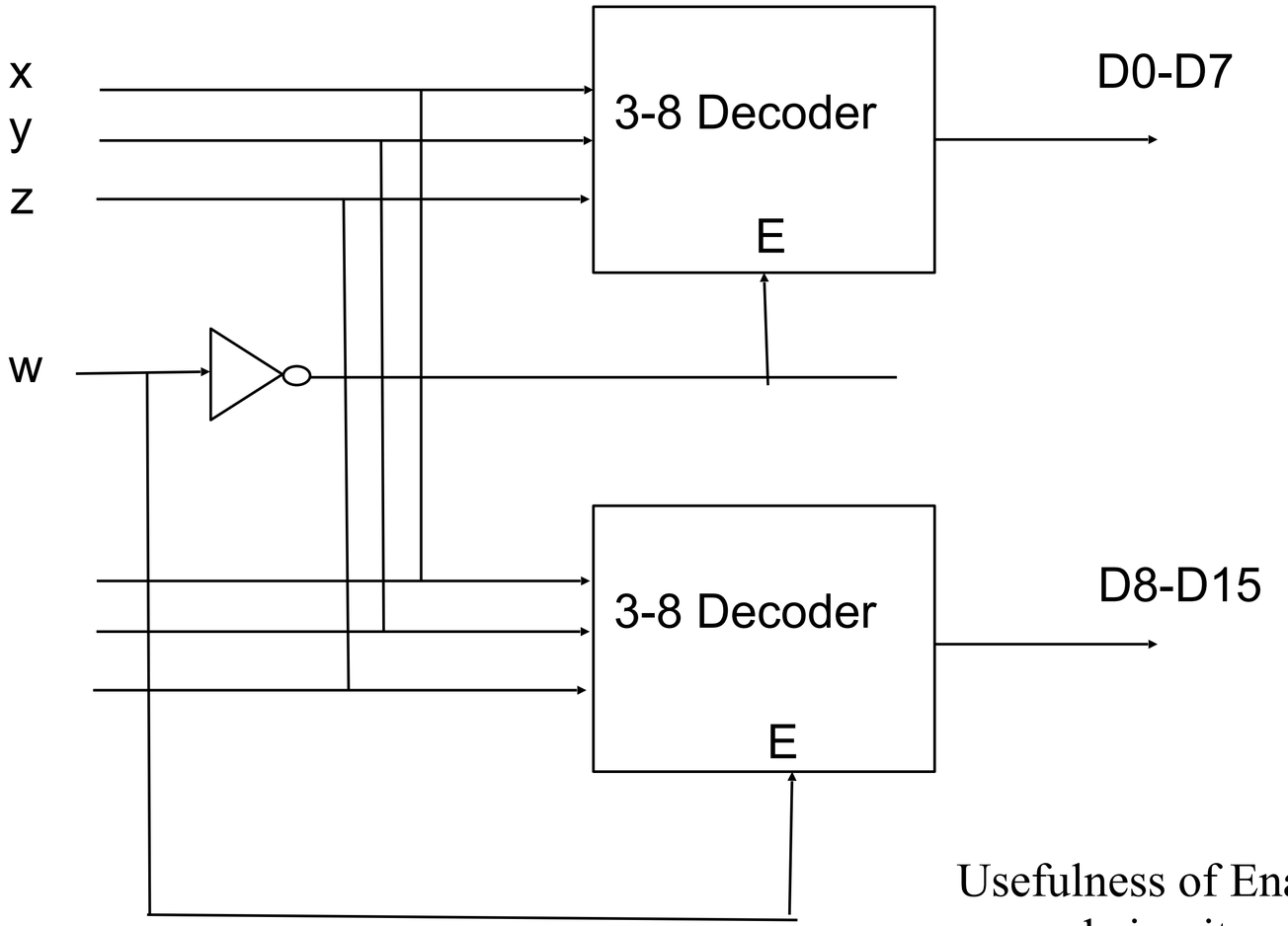
Example: 2-to-4- decoder



Decoders with Enable



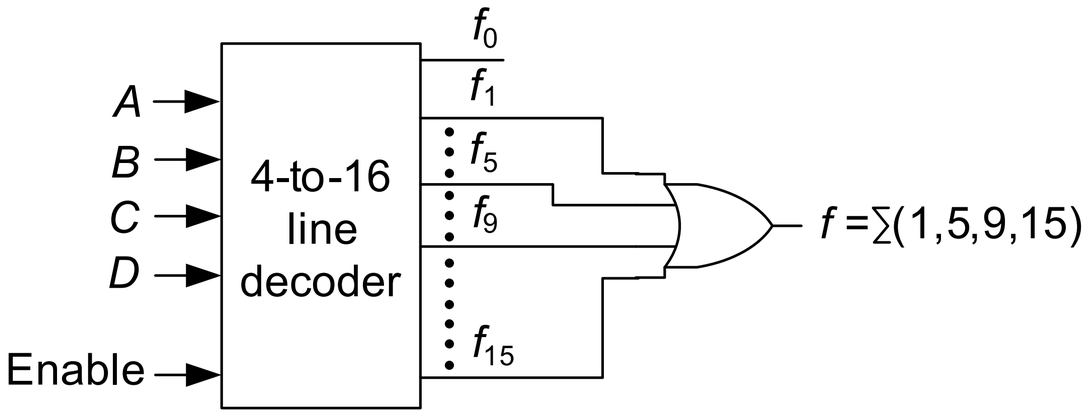
4-16 decoder using 3-8 decoder



Usefulness of Enable to expand circuits.

Realizing Arbitrary Functions

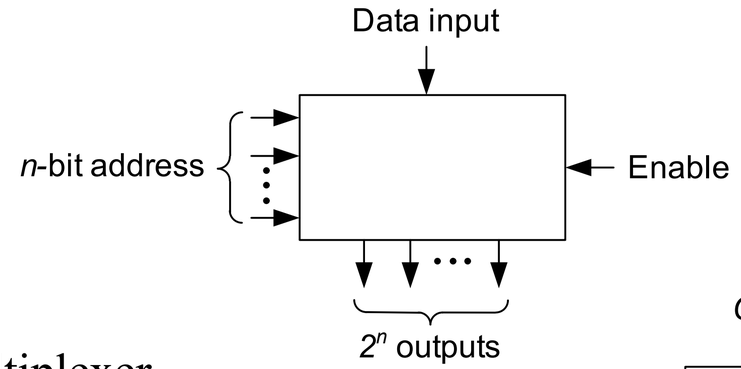
Idea: Realize a distinct minterm at each output



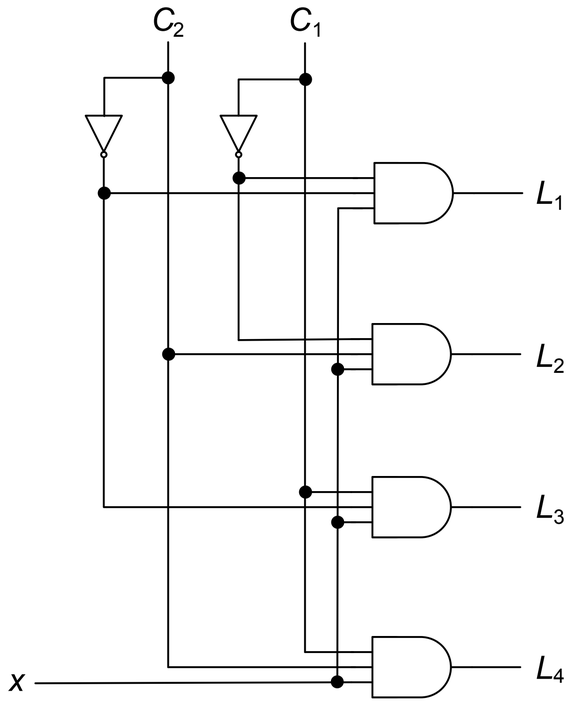
Demultiplexer

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



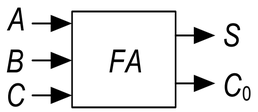
Adders Again

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.

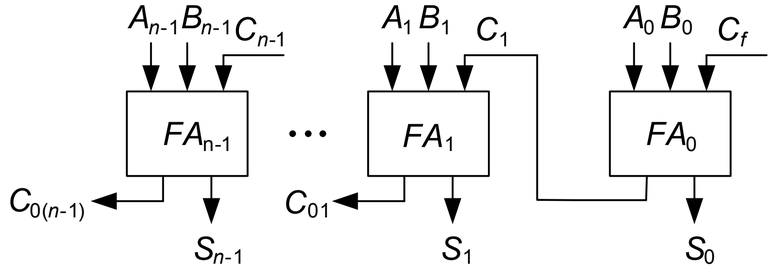
$$S = A \oplus B \oplus C$$

$$C_0 = AB + BC + CA$$

Ripple Carry Adder

Ripple-carry adder: Stages of full adders

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



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

$$C_{0i} = A_iB_i + B_iC_{0i} + C_{0i}A_i$$

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**