ECE 2300
Digital Logic & Computer Organization
Spring 2025

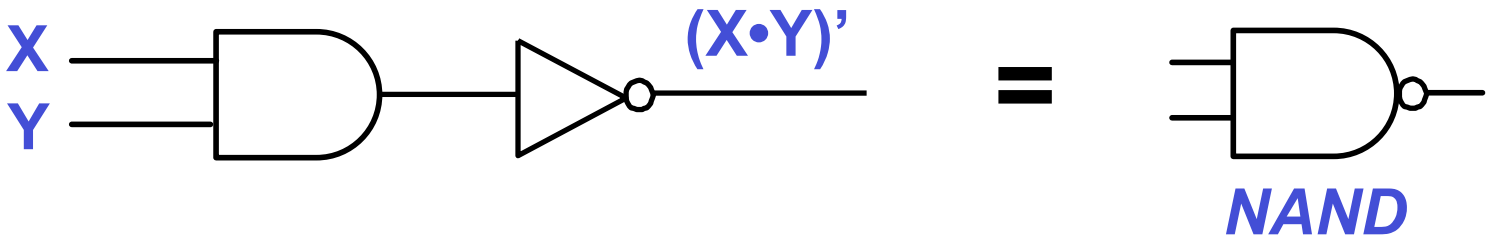
Combinational Building Blocks

Cornell University

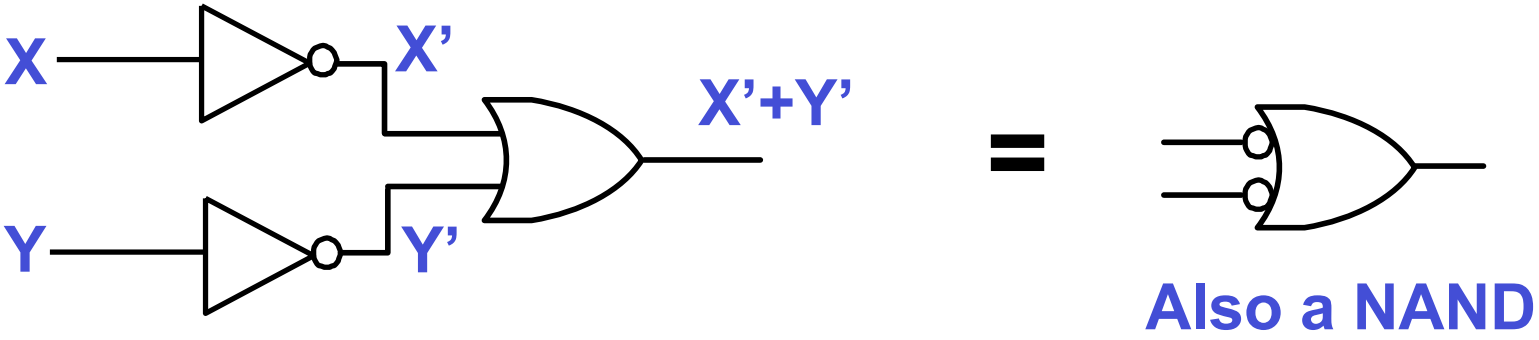
Announcements

- Lab 1 released
- HW 1 due tomorrow
- HW 2 will be posted soon

NAND Logic Gate



Using De Morgan's Law: (X•Y)' = X'+Y'



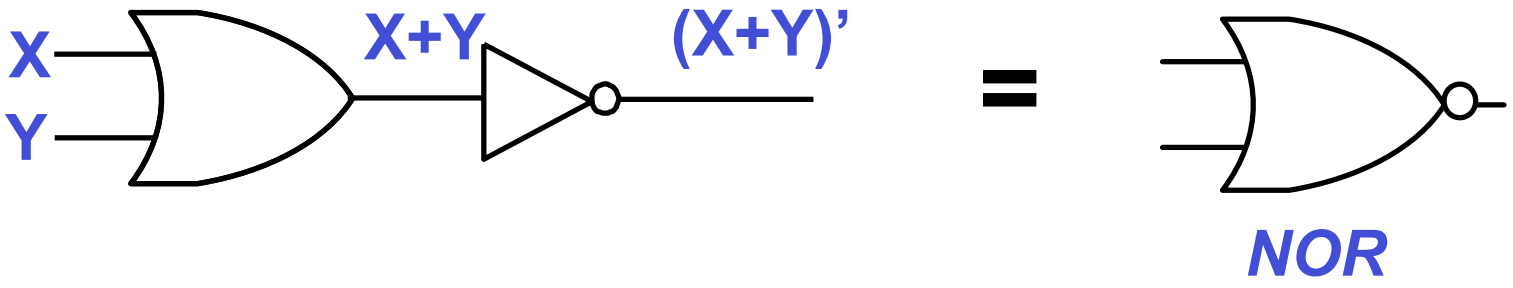
We can build circuits from NAND only!

NAND is a Universal Gate

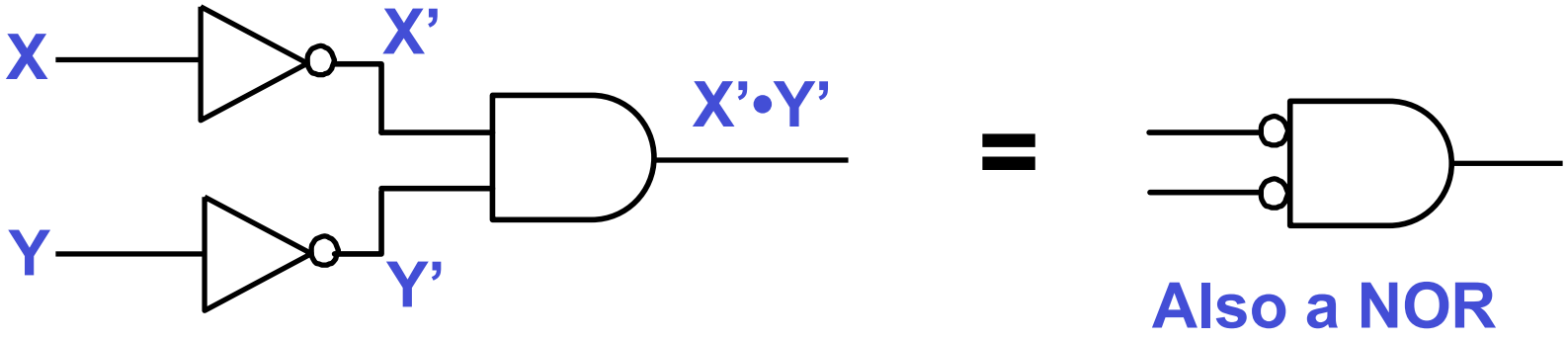
- NOT
- AND
- OR

We can build circuits from NAND only!

NOR Logic Gate



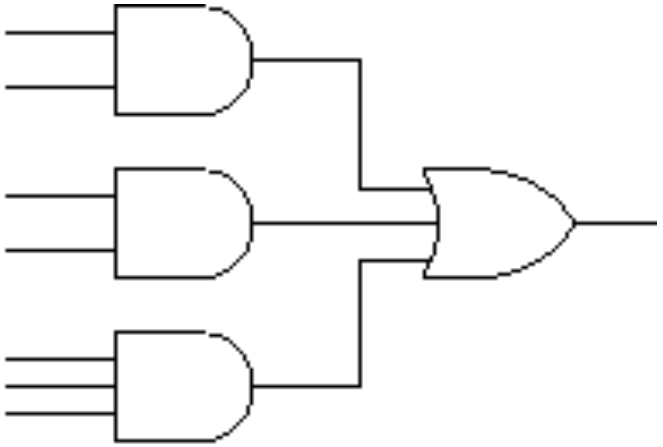
Using De Morgan's Law: (X+Y)' = X'•Y'



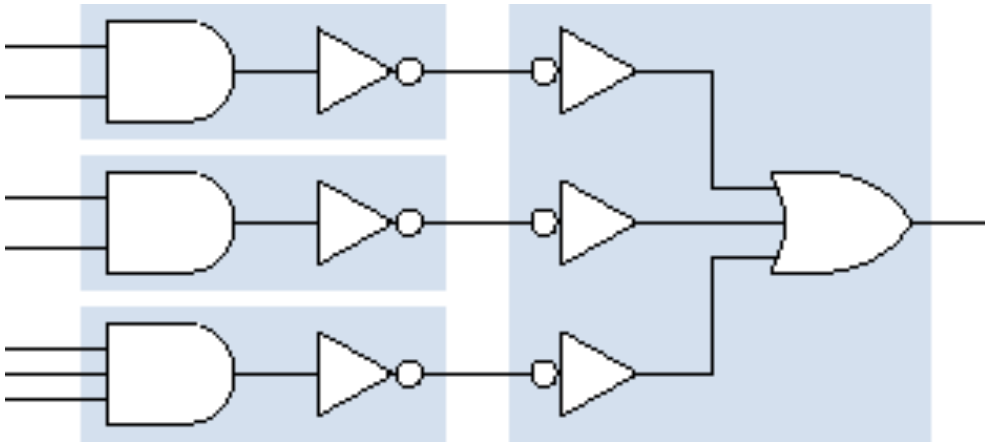
NOR is also a universal gate

We can build circuits from NOR only!

Sum-of-Products Revisited

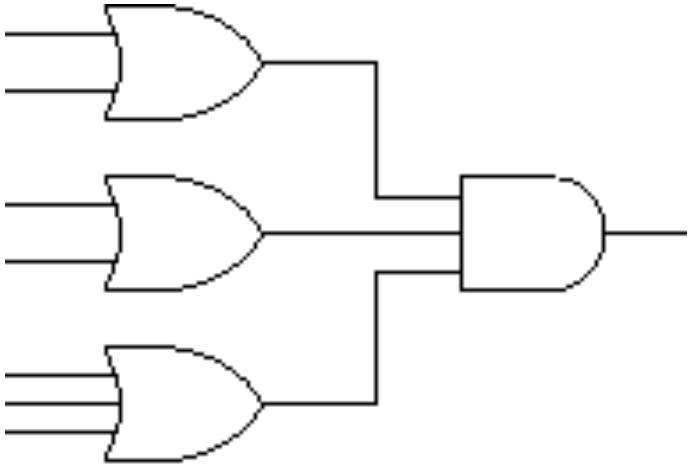


AND-OR

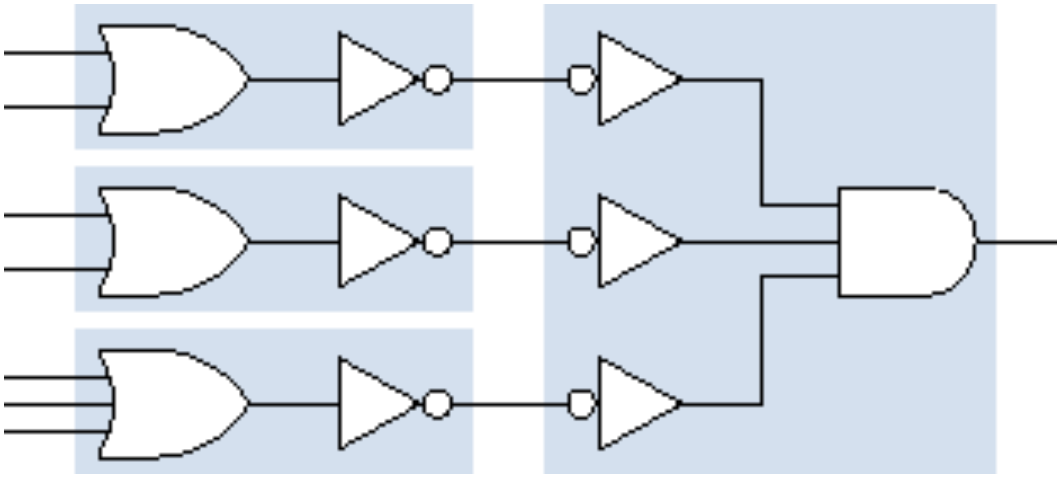


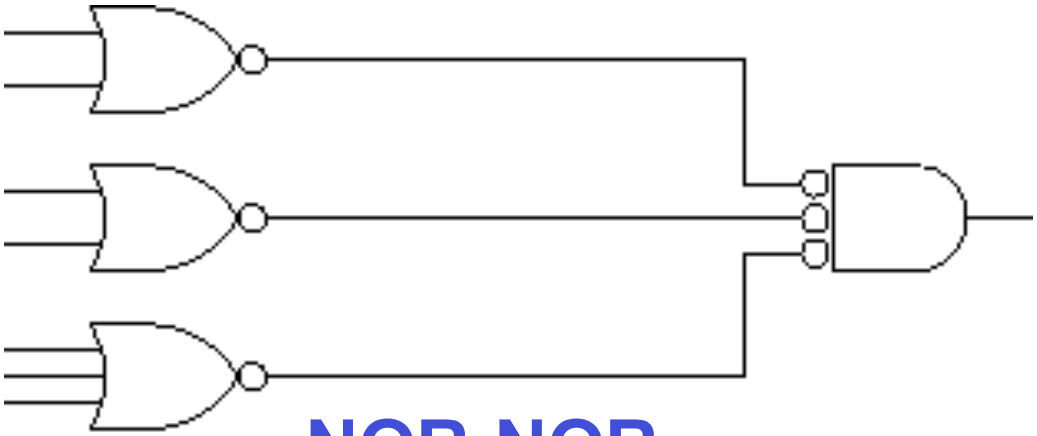
NAND-NAND

Product-of-Sums Revisited



OR-AND



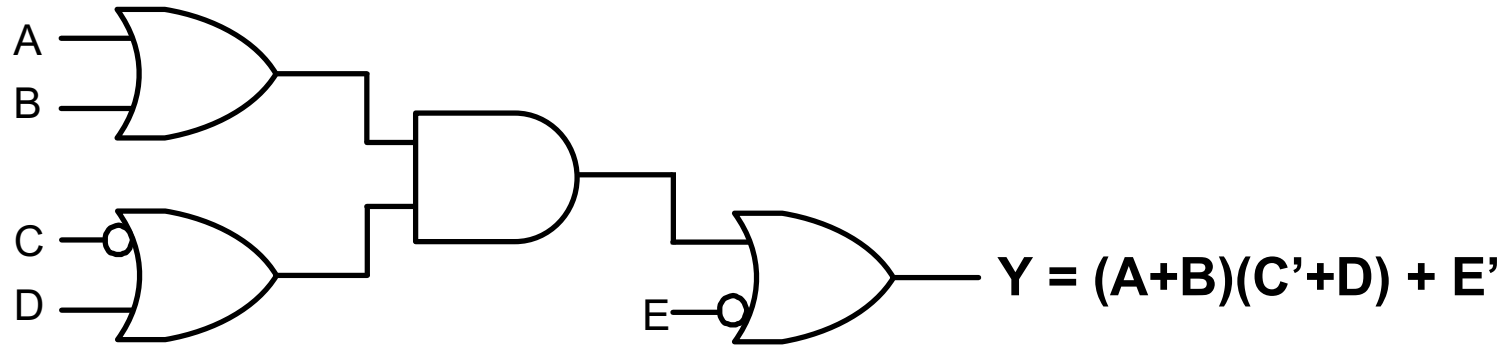


NOR-NOR

Multi-Level Logic

- So far we have primarily focused on two-level representations for combinational logic

- Multi-level logic is typically more compact (i.e., cost-efficient) in practice

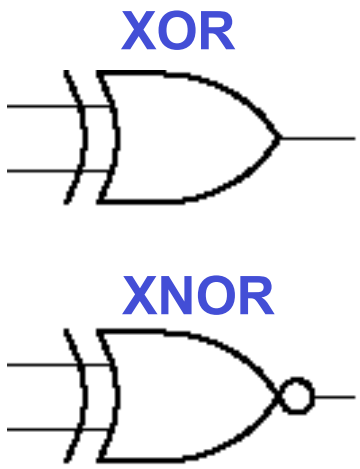


Combinational Building Blocks

- More complex functions built from basic gates
  - Comparators
  - Multiplexers
  - Decoders
  - Encoders
  - …
- Typically tens to hundreds of transistors
  - *Used to be* called Medium Scale Integration (MSI)
- Common building blocks for digital systems

XOR Gate

X	Y	X⊕Y	(X⊕Y)'
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

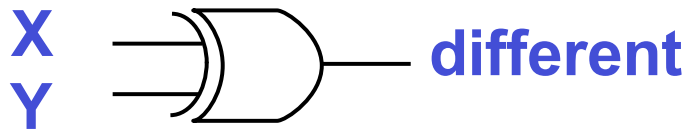


- **XOR:** F=X•Y'+X'•Y
  - Similar to OR gate, except when inputs are 1
  - Used for comparisons, error checking, etc.

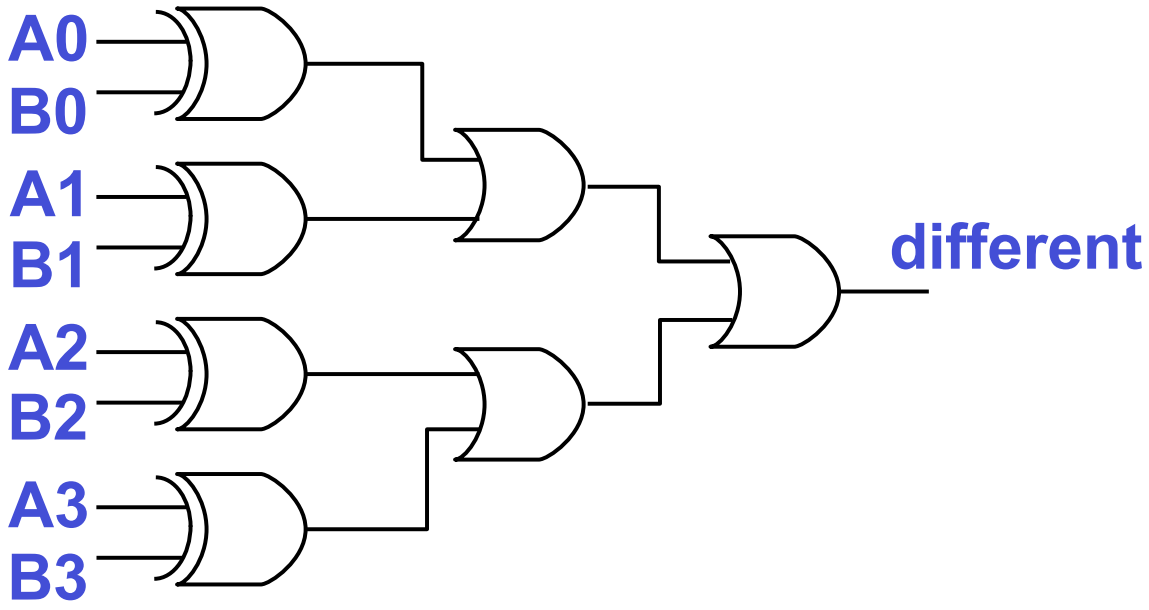
- **XNOR:** F=X'•Y'+X•Y
  - Complemented version of XOR

Equality Comparators Using XOR

1-bit comparator



4-bit comparator



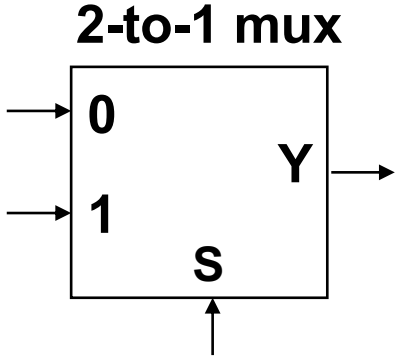
Multiplexer ("mux")

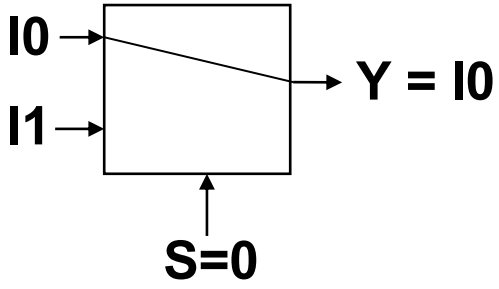
- Connects one of *n* inputs to the output
  - *select* control signals pick one of the *n* sources
    - $\lceil \log_2 n \rceil$ *select* bits

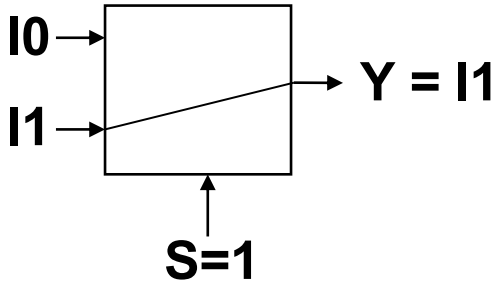
- Useful when multiple data sources need to be routed to a single destination
  - Often arises from resource sharing
  - Example: select 1-of-n *data* inputs to an adder

2-to-1 Mux

- **Selects one of two inputs to appear at the output**
  - Two data inputs (I0, I1) and one control input (S)
  - Output is I0 if S = 0 and I1 if S = 1



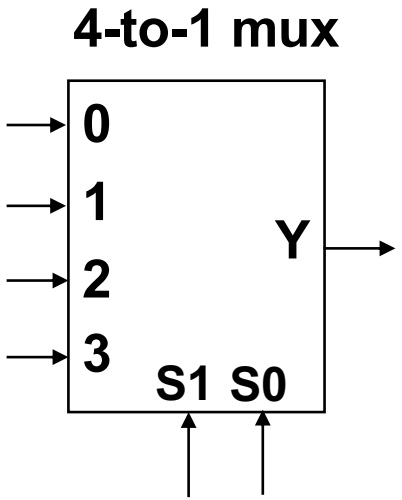




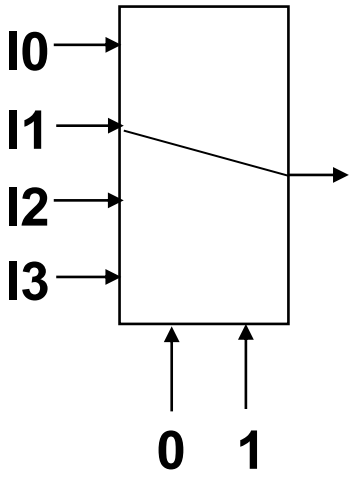
$$Y = S' \cdot I0 + S \cdot I1$$

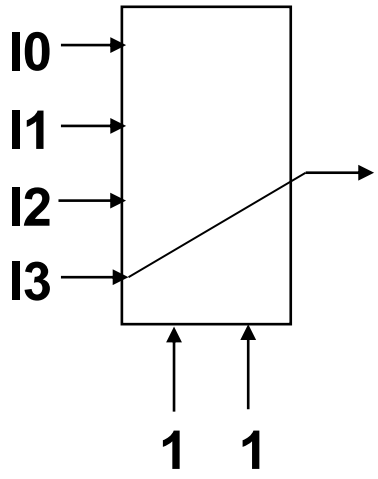
4-to-1 Mux

- Selects one of four inputs to appear at the output



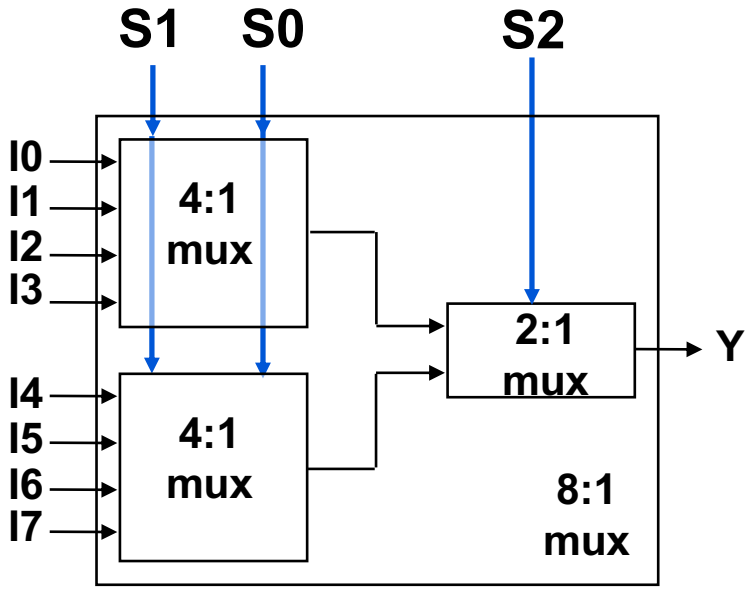
$$Y = S1' \cdot S0' \cdot I0 + S1' \cdot S0 \cdot I1 + S1 \cdot S0' \cdot I2 + S1 \cdot S0 \cdot I3$$





Cascading Multiplexers

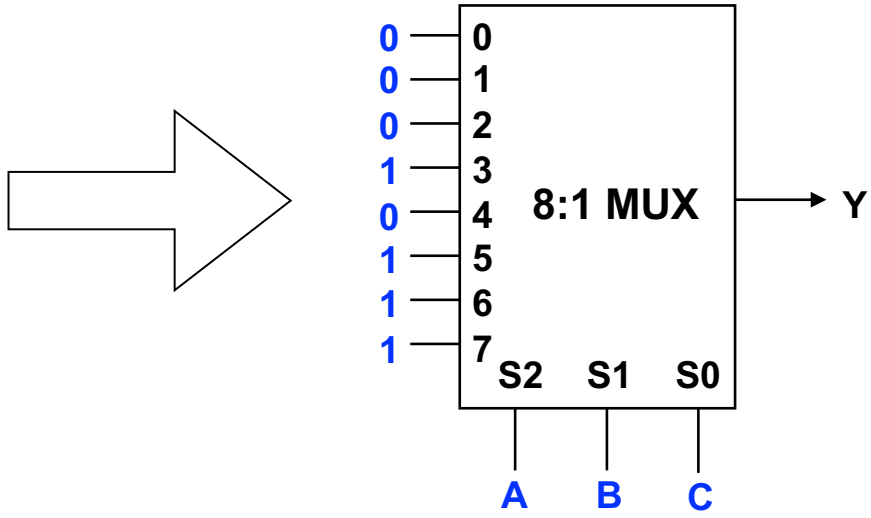
- **Large multiplexers can be implemented by cascading smaller ones**



Logic Functions Using Muxes

- **Any function of n variables can be implemented with a 2^n:1 multiplexer**
  - **Input variables connected to *select* inputs**
  - **Data inputs tied to 0 or 1 according to truth table**

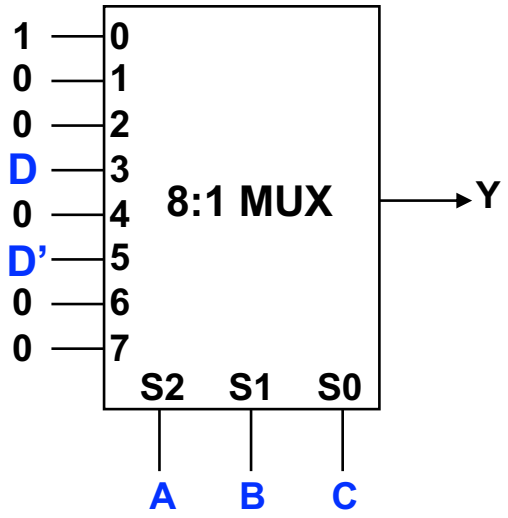
ABC	Y
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1



Getting Away with a Smaller Mux

- Can use 2^{n-1}:1 multiplexer and at most one inverter
  - Connect n-1 input variables to *select* inputs
  - Data inputs tied to 0, 1, n^{th} variable, or inverted n^{th} variable

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Decoder

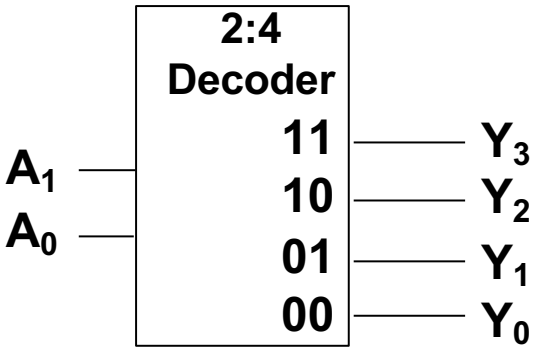
- **Binary decoders**
  - n inputs, 2^n outputs
  - Each output corresponds to a unique input value
  - Only one output is asserted (true) at a time, also known as the *one-hot* scheme

- **Example: A 1-to-2 decoder**

A	Y_1	Y_0
0	0	1
1	1	0

$Y_0 = A'$
$Y_1 = A$

2-to-4 Decoder



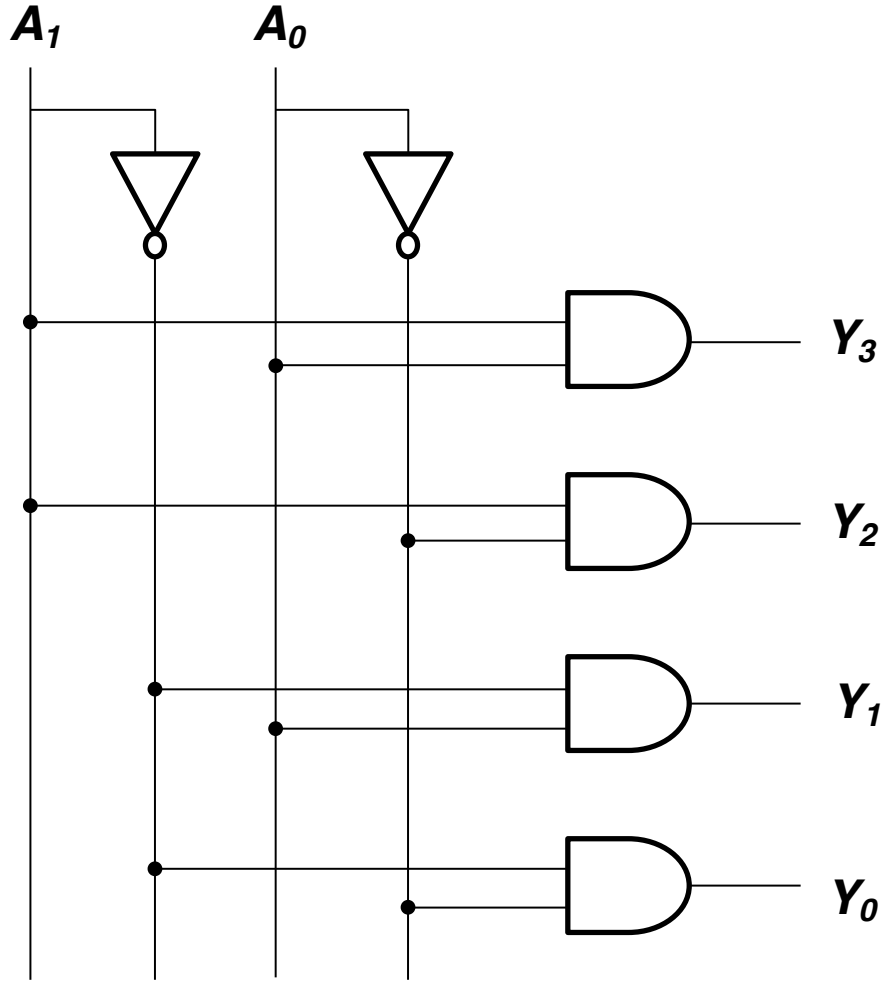
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$Y_0 = A_1' \bullet A_0'$

$Y_1 = A_1' \bullet A_0$

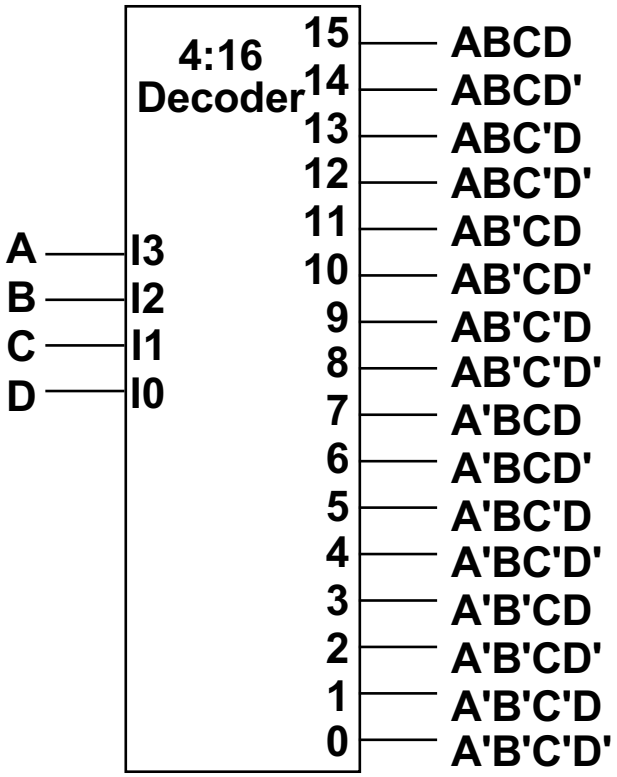
$Y_2 = A_1 \bullet A_0'$

$Y_3 = A_1 \bullet A_0$



Logic Functions Using Decoders

- **n:2^n decoder can be used to implement any function of n variables**
  - **Connect variables to inputs**
  - **Appropriate minterms summed using extra gates to form the function**

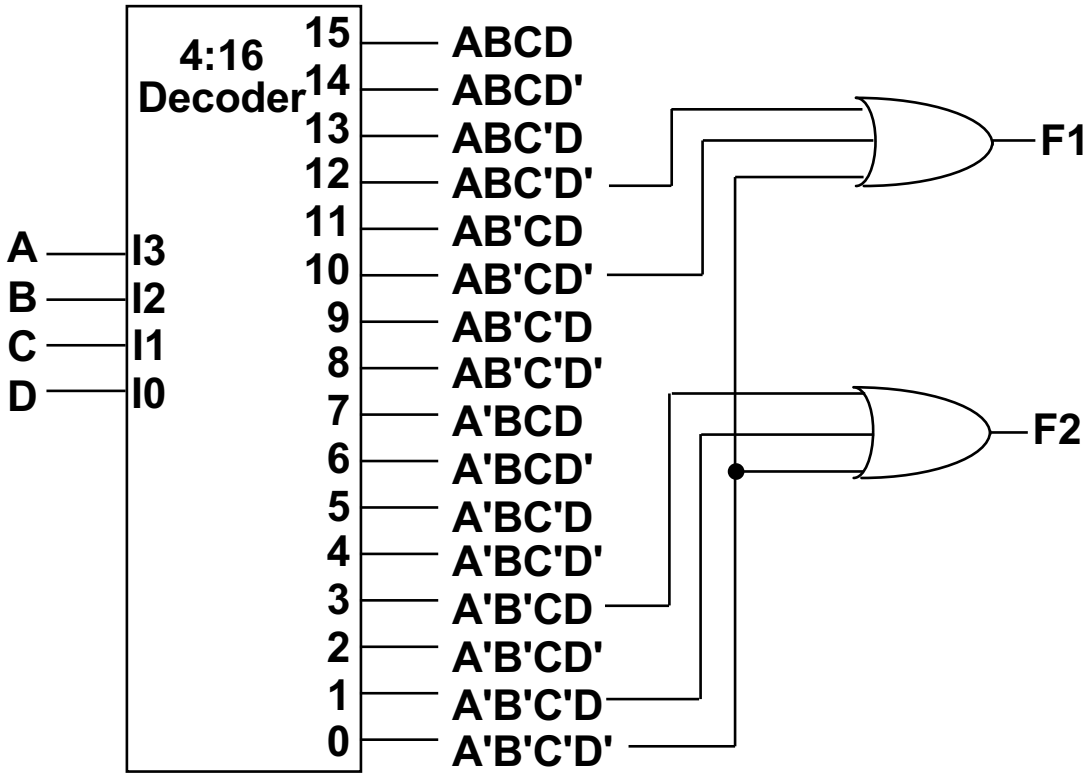


Logic Functions Using Decoders

F1 = A'B'C'D' + AB'CD' + ABC'D'

F2 = A'B'C' + A'B'CD

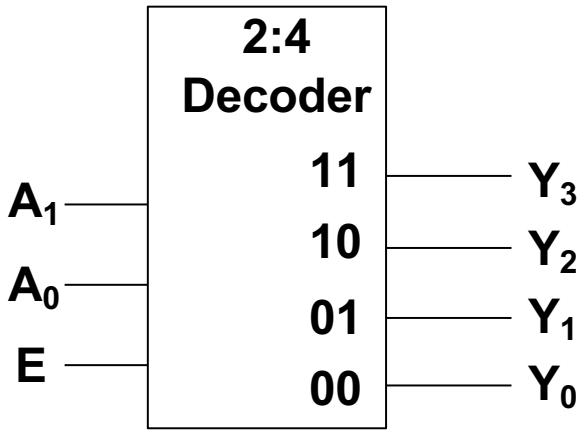
= A'B'C'D' + A'B'C'D + A'B'CD



Decoder with Enable

- Converts an n inputs into a one-hot 2^n outputs, but *only when the enable signal is active*

- Example: A 1-to-2 decoder with enable



E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

$Y_3 = A_1 \bullet A_0 \bullet E$

$Y_2 = A_1 \bullet A_0' \bullet E$

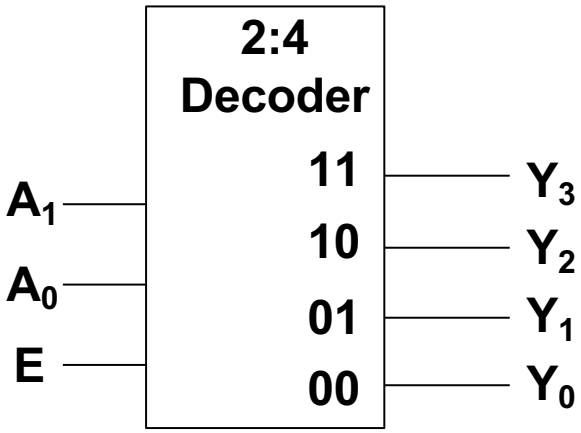
$Y_1 = A_1' \bullet A_0 \bullet E$

$Y_0 = A_1' \bullet A_0' \bullet E$

X: don't care input

Here 0XX covers four input combinations 000, 001, 010, 011

Decoder with Enable (2)

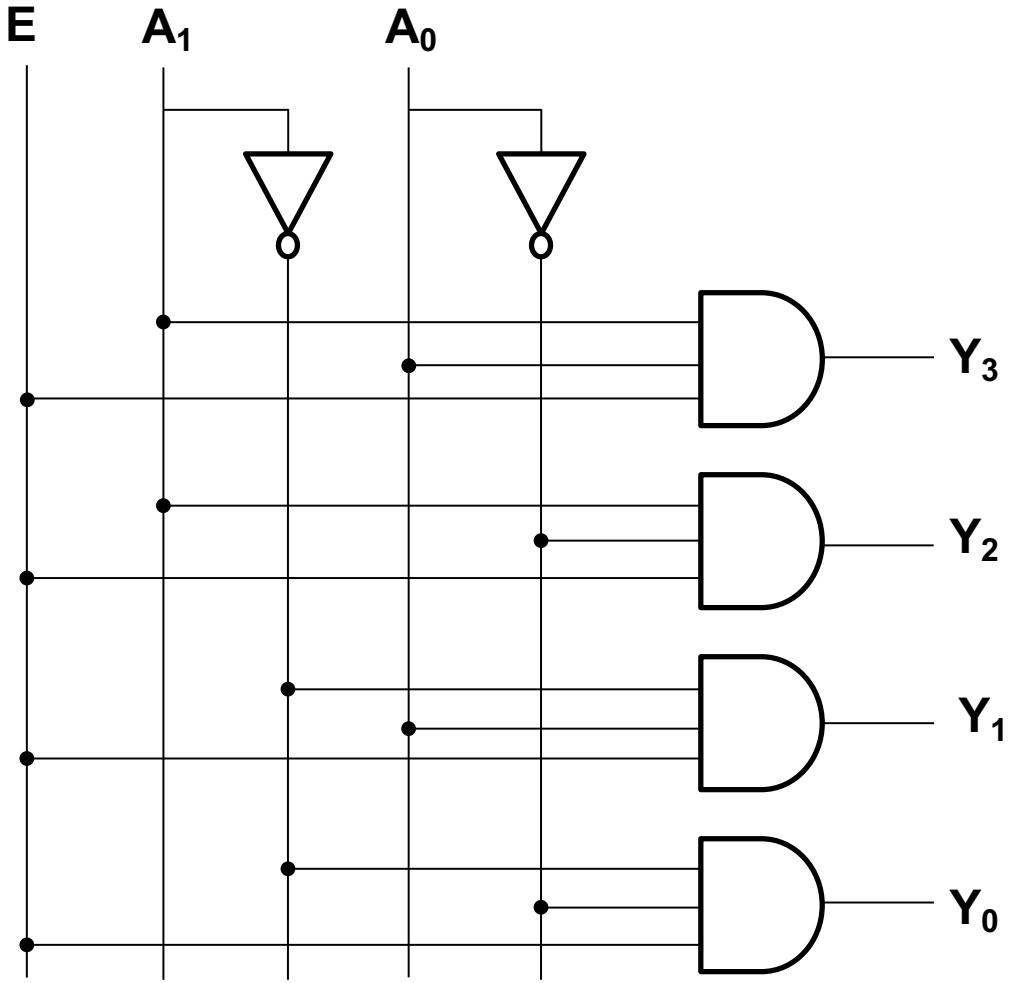


$Y_3 = A_1 \bullet A_0 \bullet E$

$Y_2 = A_1 \bullet A_0' \bullet E$

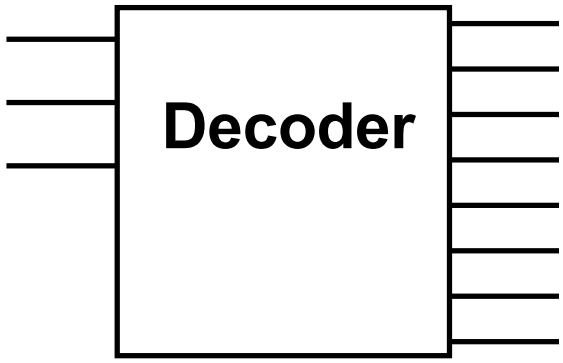
$Y_1 = A_1' \bullet A_0 \bullet E$

$Y_0 = A_1' \bullet A_0' \bullet E$

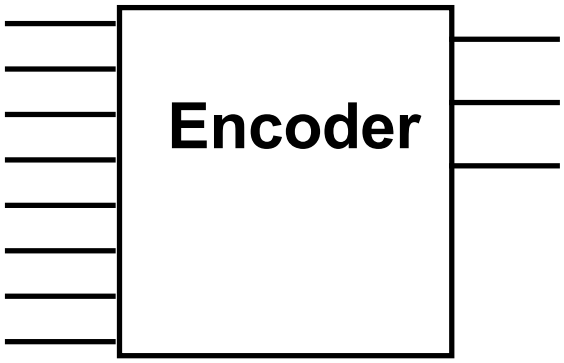


Encoders

- Opposite of decoders
- Binary encoders: 2^n inputs and n outputs



Binary code =>
(One-hot) positional code

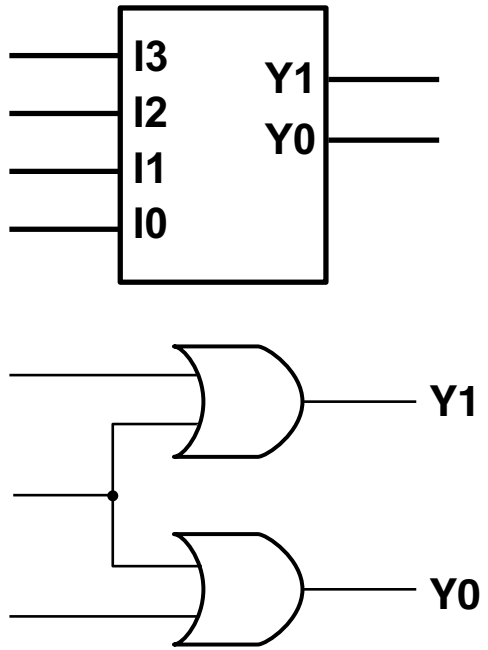


(One-hot) positional code =>
Binary code

4-to-2 Encoder

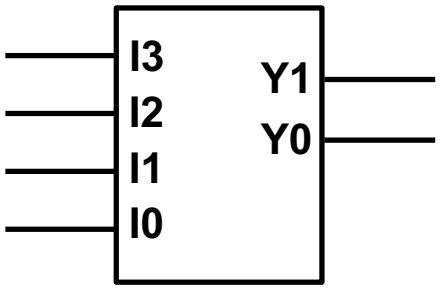
Exactly one input is asserted at any given time (one-hot inputs)

I3	I2	I1	I0	Y1	Y0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

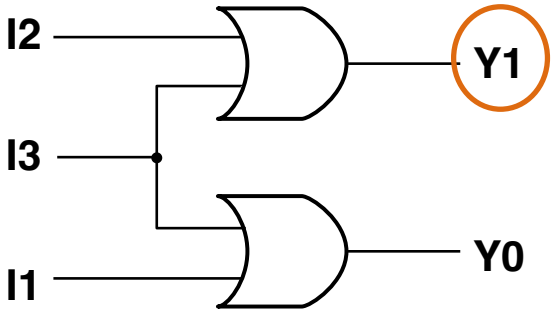


4-to-2 Encoder

Exactly one input is asserted at any given time (one-hot inputs)



I3	I2	I1	I0	Y1	Y0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



I1 I0 \ I3 I2	00	01	11	10
00	d	1	d	1
01	0	d	d	d
11	d	d	d	d
10	0	d	d	d

Next Class

CMOS Logic
(H&H 1.7)