CHAPTER VI

COMBINATIONAL LOGIC BUILDING BLOCKS

COMBINAT. LOGIC

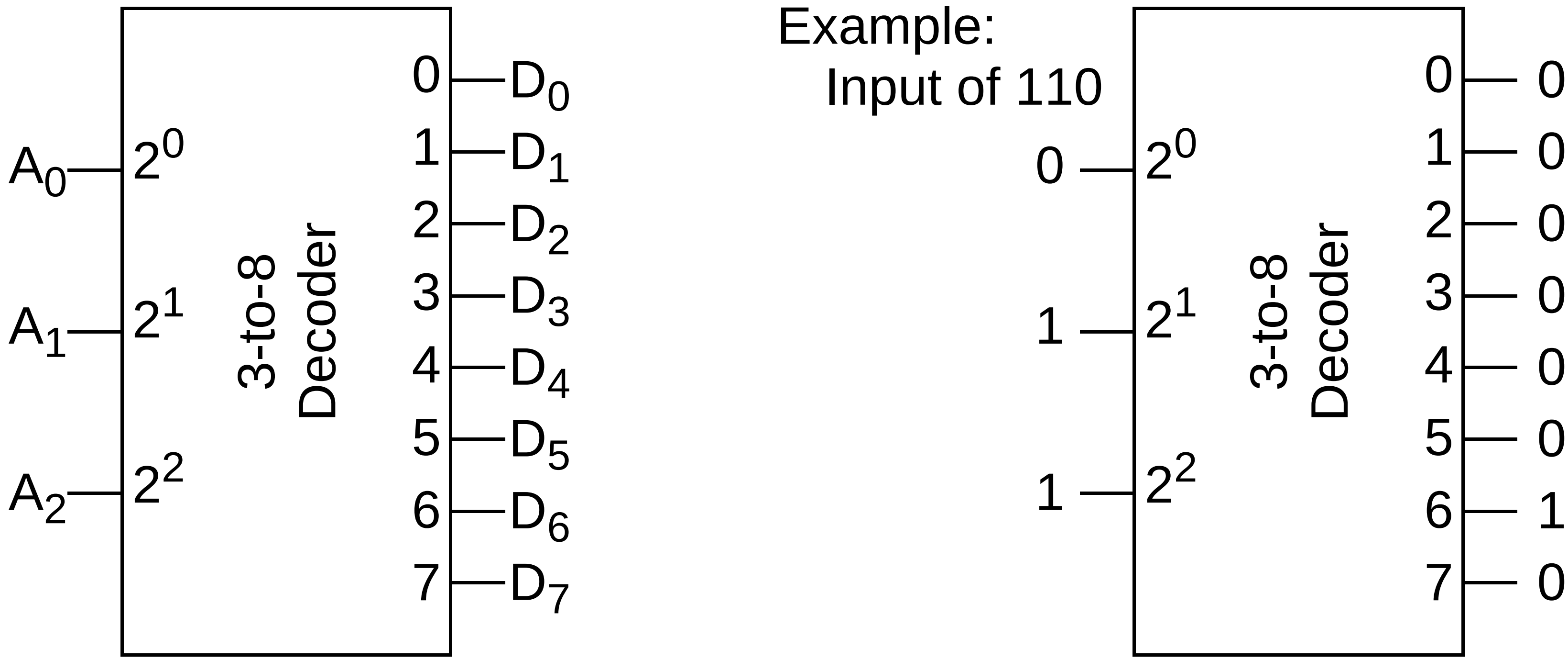
INTRODUCTION

- Combinational logic
  - Output at any time is determined completely by the current input.
    - We will later consider circuits where the output is determined by the input and the current state (memory) of the system.
  - In this chapter we will consider some useful building blocks that can be pieced together and used in larger designs. This will include:
    - Multiplexers (selectors) and demultiplexers (distributors)
    - Encoders, priority encoders, decoders
    - Adders (full and half)
    - Parity generators and parity checkers
    - Shifters and rotators
    - Comparators

DECODERS

BASIC DECODER

- Standard decoder is an n-to-m-line decoder, where $m \leq 2^n$.
  - Example: 3-to-8-line decoder



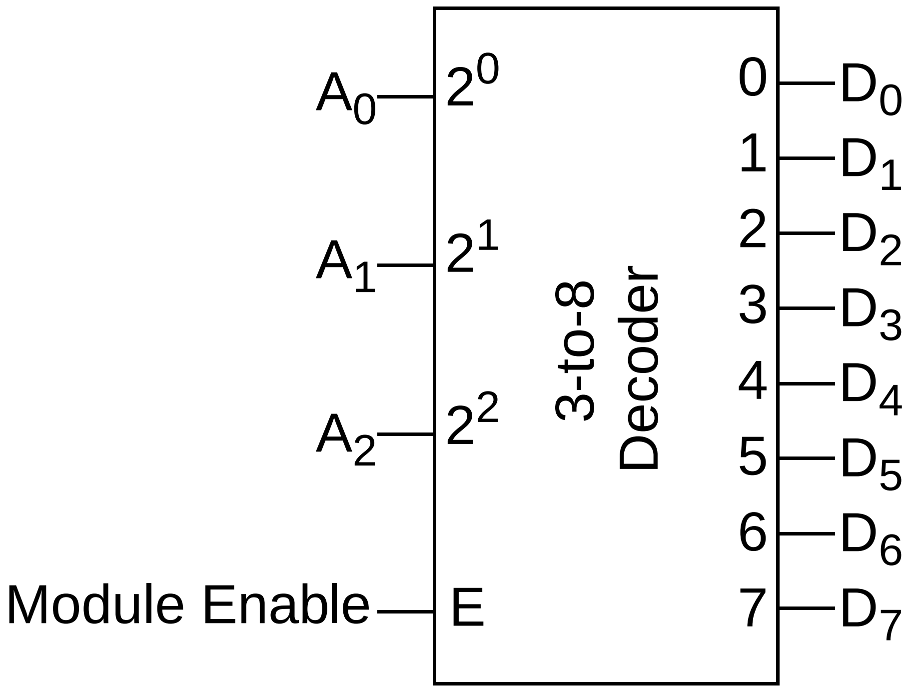
- All outputs D_m are low except for the one corresponding to the binary value of the input $A_n \ldots A_1 A_0$.

DECODERS

DECODERS WITH ENABLE

- Often, combinational logic building blocks will also have an enable line that turns on outputs or leaves them off.

3-to-8 Decoder
with Enable



DECODERS

TRUTH TABLES

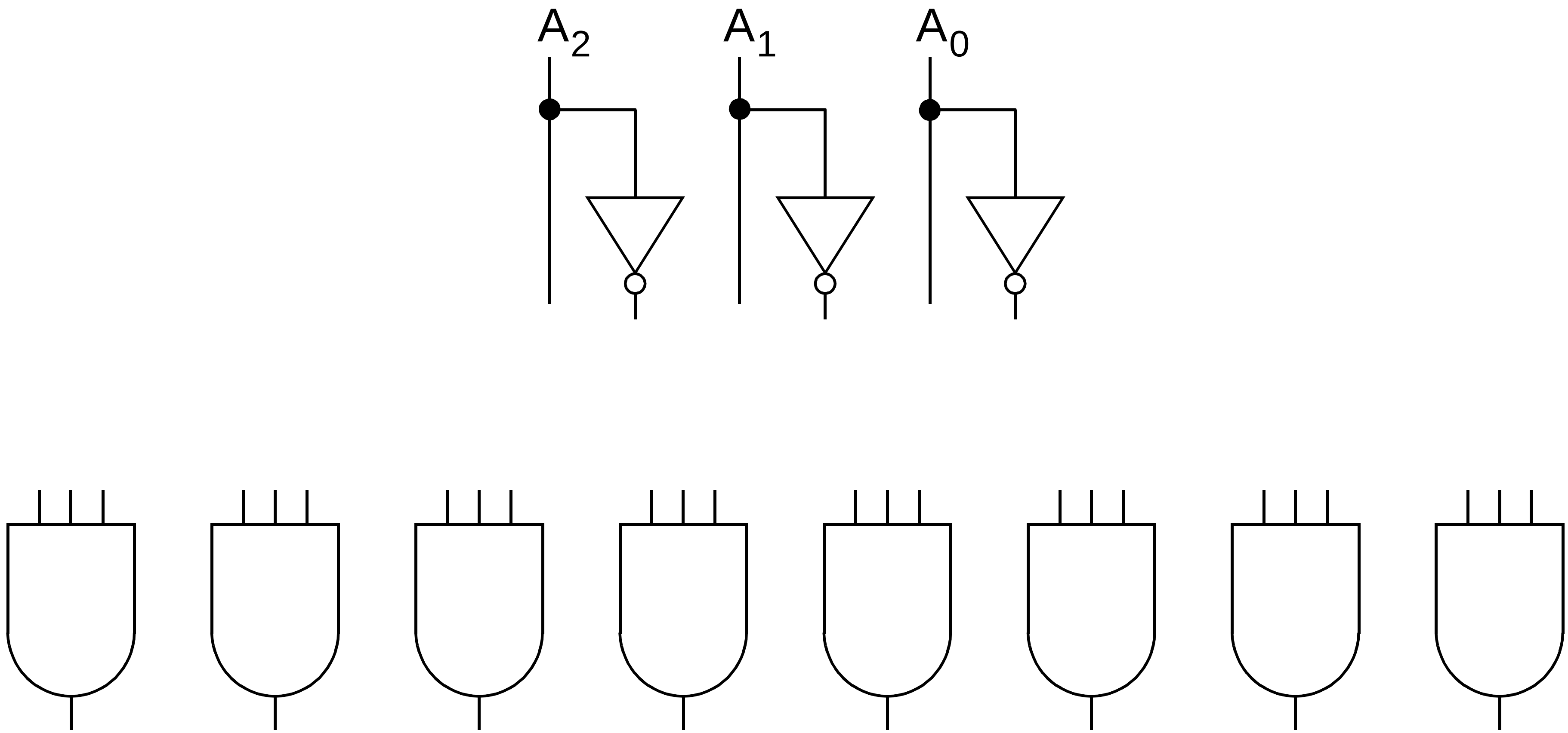
- Truth table for a **3-to-8-line decoder**:

Inputs				Outputs							
A_2	A_1	A_0	E	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
X	**X**	**X**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
1	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
1	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
1	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
1	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

DECODERS

IMPLEMENTATION

- How can a decoder be implemented? Fill in the circuit!



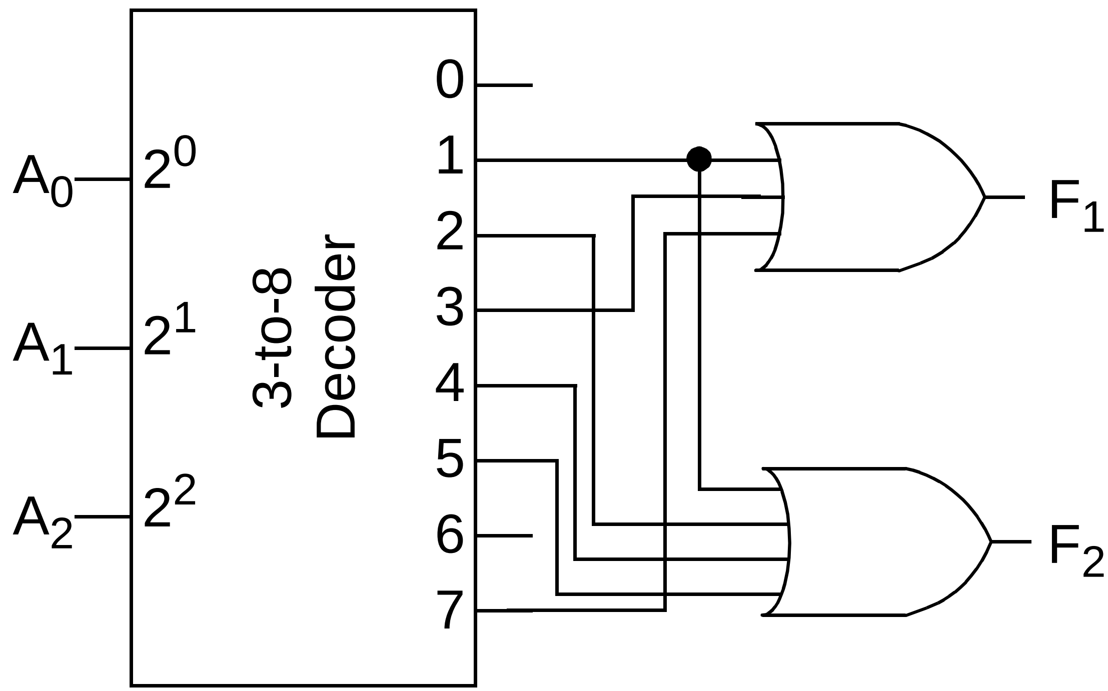
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
$A_2A_1A_0$	$A_2A_1\overline{A_0}$	$A_2\overline{A_1}A_0$	$A_2\overline{A_1}\,\overline{A_0}$	$\overline{A_2}A_1A_0$	$\overline{A_2}A_1\overline{A_0}$	$\overline{A_2}\,\overline{A_1}A_0$	$\overline{A_2}\,\overline{A_1}\,\overline{A_0}$

DECODERS

DESIGNING WITH DECODERS

- Any Boolean function can implemented using a **decoder** and **OR** gates by ORing together the function's **minterms**.

Inputs			Outputs	
A_2	A_1	A_0	F_1	F_2
0	**0**	**0**	**0**	**0**
0	**0**	**1**	**1**	**1**
0	**1**	**0**	**0**	**1**
0	**1**	**1**	**1**	**0**
1	**0**	**0**	**0**	**1**
1	**0**	**1**	**0**	**1**
1	**1**	**0**	**0**	**0**
1	**1**	**1**	**1**	**0**

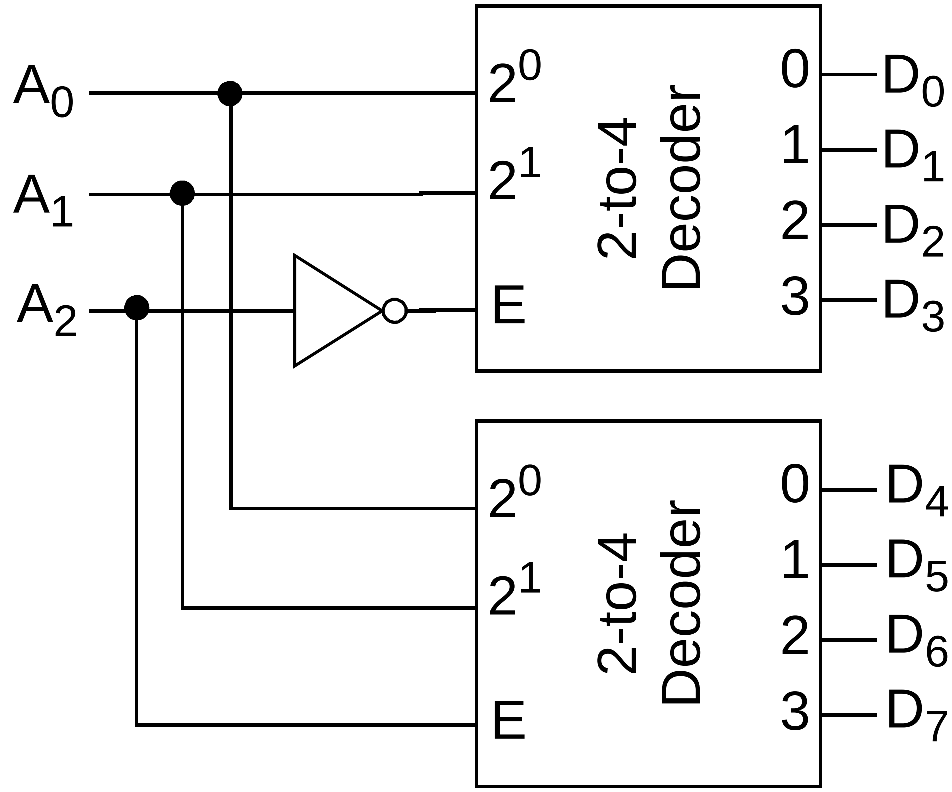


DECODERS

DECODER NETWORKS

- We can also use multiple decoders to form a larger decoder.

3-to-8 Decoder Implemented with two 2-to-4 Decoders



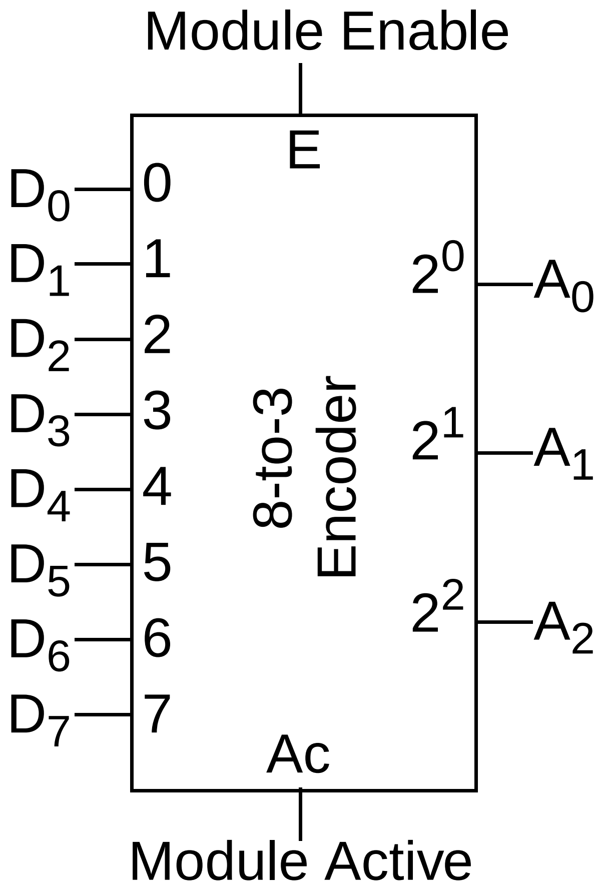
A_2 used with enable input to control which decoder will output the **1**.

A_1 and **A_0** used to select which output on specific decoder will output **1**.

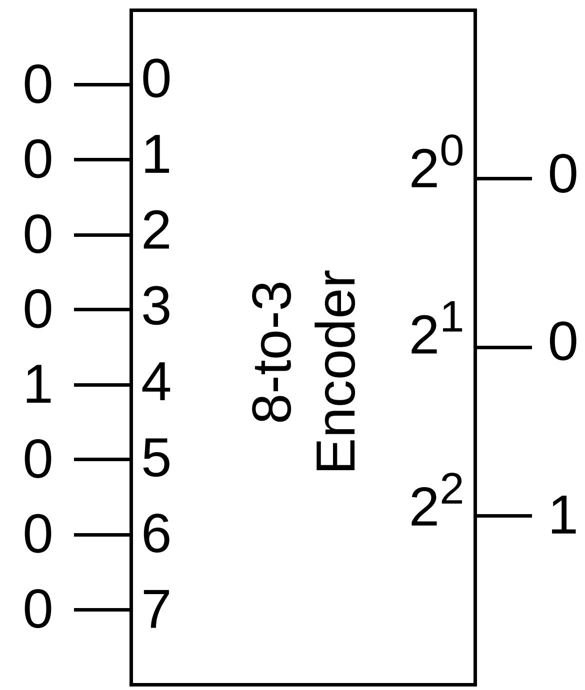
ENCODERS

BASIC ENCODER

- Standard binary encoder is an m-to-n-line encoder, where $m \leq 2^n$.
  - Example: 8-to-3-line encoder



Example:
Input of 00010000



ENCODERS

ENCODER TRUTH TABLE

- Truth table for an **8-to-3-line encoder**:

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**
0	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
0	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**

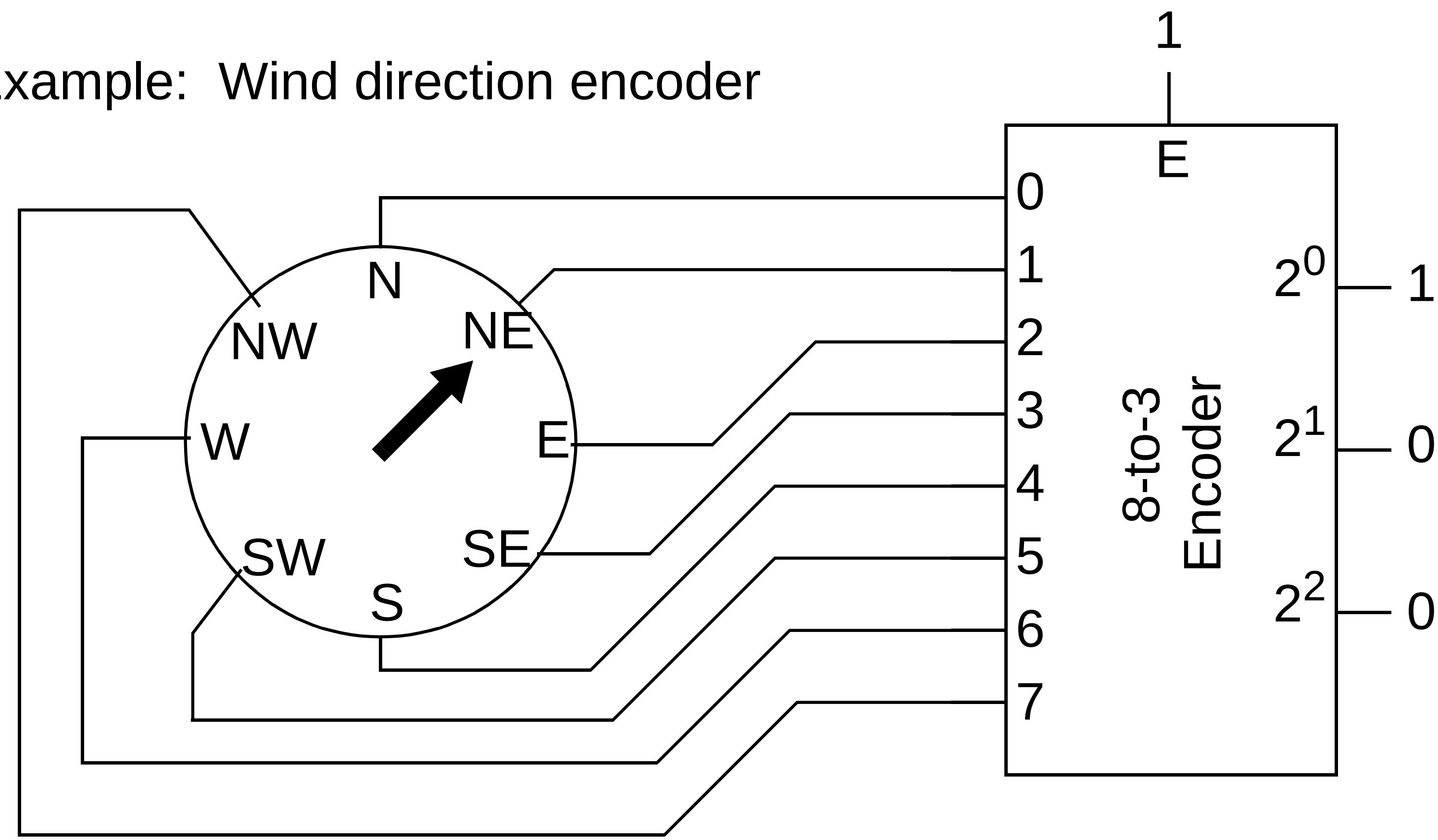
- Assumed that only one input is **1**. What happens if more then one is **1**?

ENCODERS

DESIGNING WITH ENCODERS

- Encoders are useful when the occurrence of one of several disjoint events needs to be represented by an integer identifying the event.

Example: Wind direction encoder



pp. 253-254 of Ercegovac, Lang and Moreno, “Introduction to Digital Systems”, 1999.

ENCODERS

PRIORITY ENCODERS

- A priority encoder takes the input of 1 with the highest index and translates that index to the output.

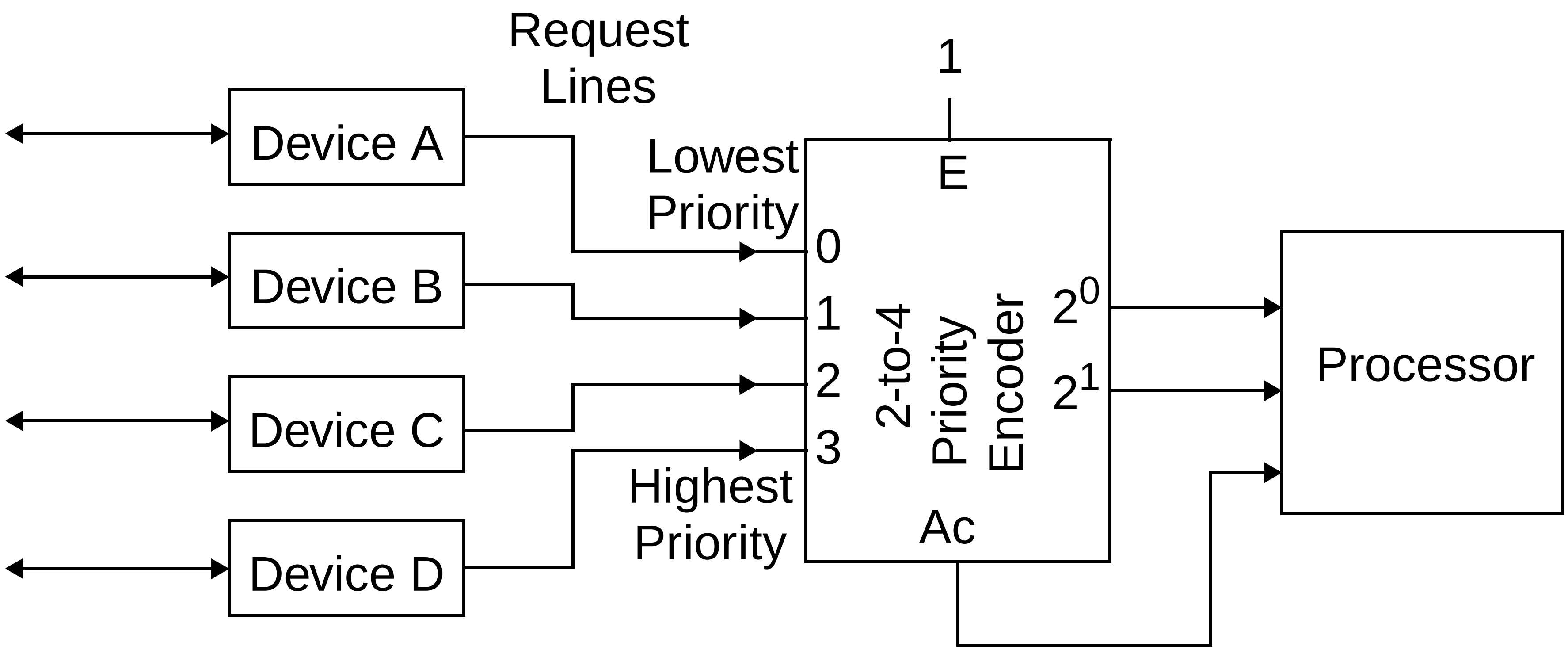
Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**1**	**X**	**0**	**0**	**1**
0	**0**	**0**	**0**	**0**	**1**	**X**	**X**	**0**	**1**	**0**
0	**0**	**0**	**0**	**1**	**X**	**X**	**X**	**0**	**1**	**1**
0	**0**	**0**	**1**	**X**	**X**	**X**	**X**	**1**	**0**	**0**
0	**0**	**1**	**X**	**X**	**X**	**X**	**X**	**1**	**0**	**1**
0	**1**	**X**	**X**	**X**	**X**	**X**	**X**	**1**	**1**	**0**
1	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**1**	**1**	**1**

ENCODERS

DESIGN WITH P-ENCODER

- Priority encoders are useful when inputs have a predefined priority and we wish to select the input with the highest priority.

Example: Resolving interrupt requests

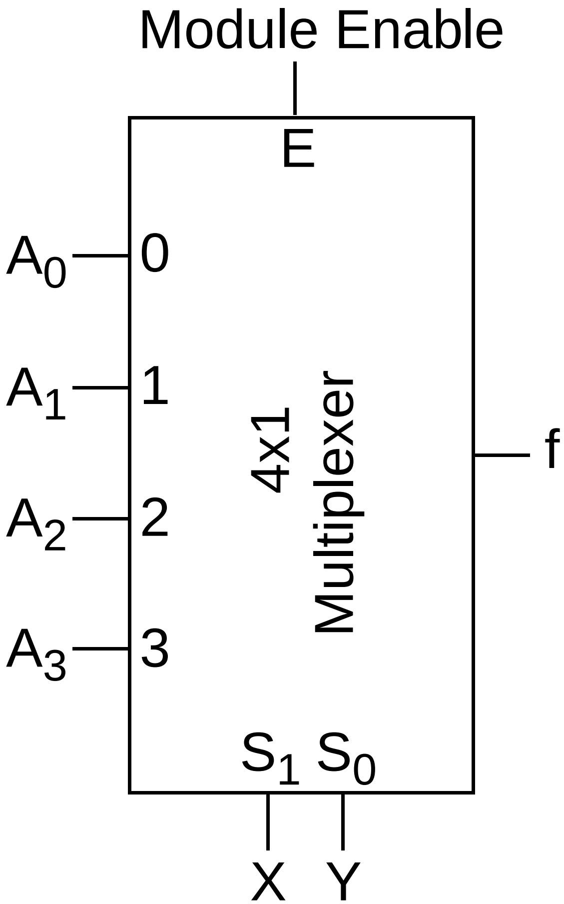


pp. 253-256 of Ercegovac, Lang and Moreno, “Introduction to Digital Systems”, 1999.

MULTIPLEXERS

BASIC MULTIPLEXER (MUX)

- Selects one of many inputs to be directed to an output.

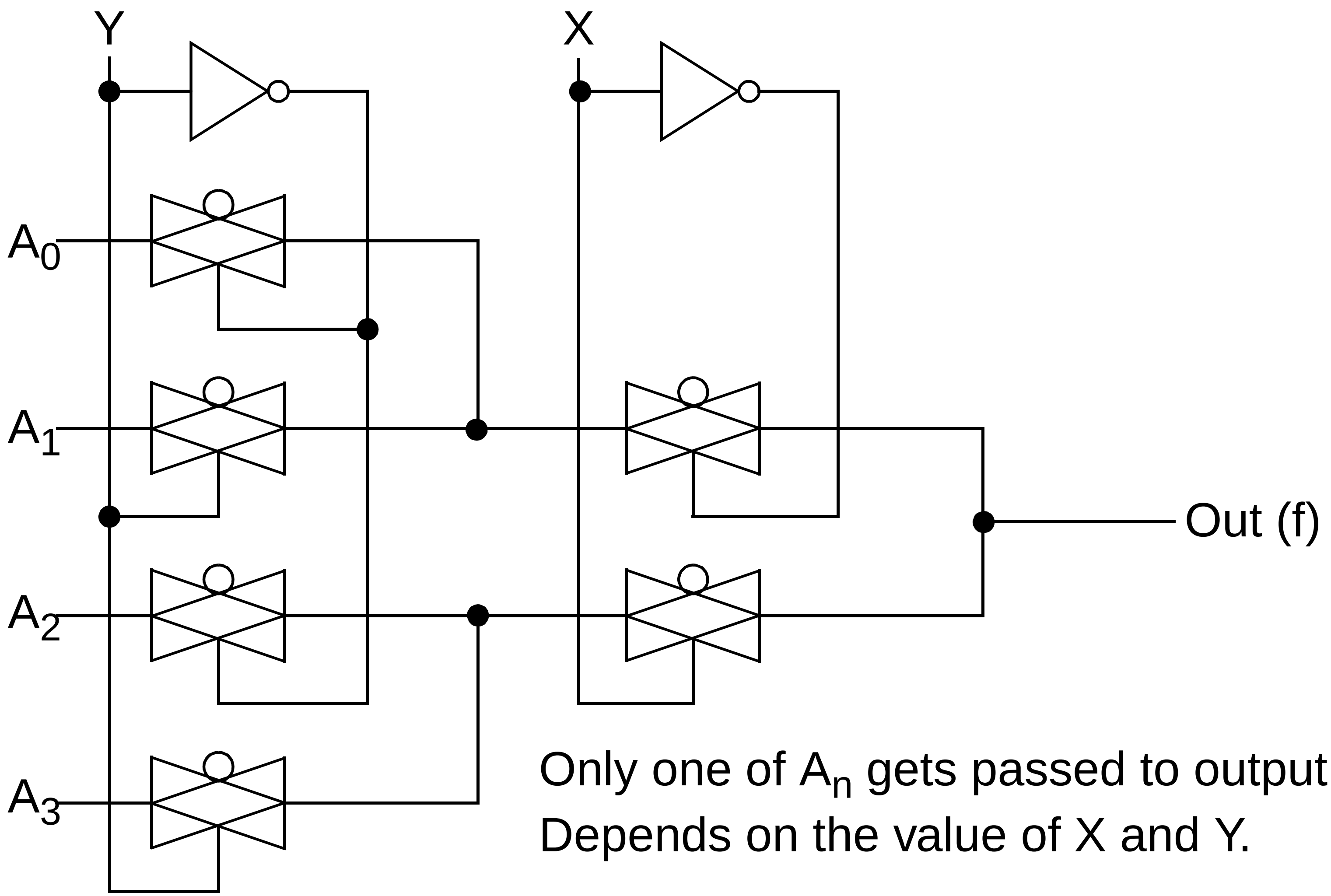


		Inputs				Output
X	Y	A_3	A_2	A_1	A_0	F
0	**0**	**X**	**X**	**X**	**0**	A_0=**0**
0	**1**	**X**	**X**	**0**	**X**	A_1=**0**
1	**0**	**X**	**0**	**X**	**X**	A_2=**0**
1	**1**	**0**	**X**	**X**	**X**	A_3=**0**
0	**0**	**X**	**X**	**X**	**1**	A_0=**1**
0	**1**	**X**	**X**	**1**	**X**	A_1=**1**
1	**0**	**X**	**1**	**X**	**X**	A_2=**1**
1	**1**	**1**	**X**	**X**	**X**	A_3=**1**

MULTIPLEXERS

USING PASS GATES

- The **4x1 mux** can be implemented with **pass gates** as follows.



Only one of A_n gets passed to output.
Depends on the value of X and Y.

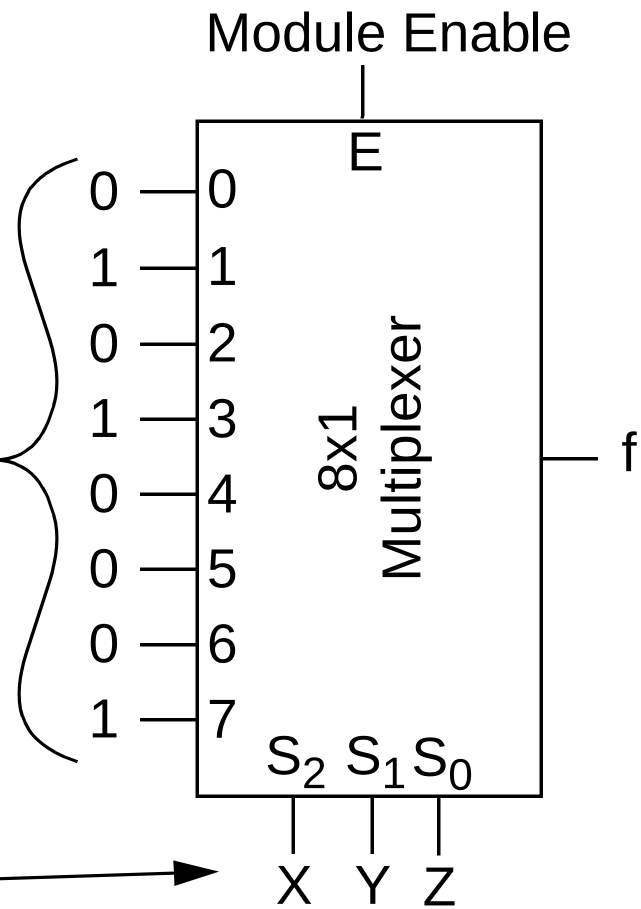
MULTIPLEXERS

DESIGN WITH MULTIPLEXERS

- Any Boolean function can be implemented by setting the inputs corresponding to the function and the selectors as the variables.

Example:

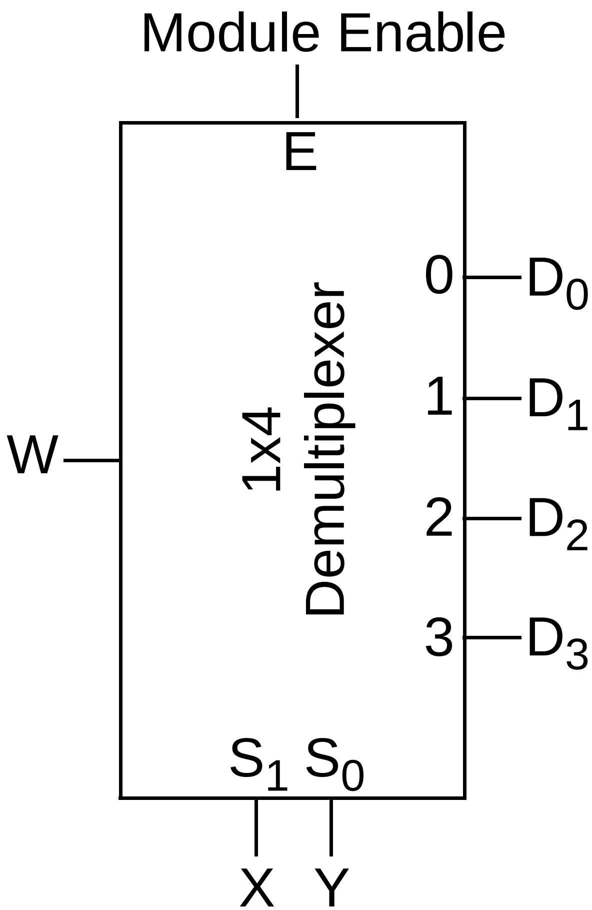
X	Y	Z	F_1
0	**0**	**0**	**0**
0	**0**	**1**	**1**
0	**1**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**0**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**0**
1	**1**	**1**	**1**



DEMULTIPLEXERS

BASIC DEMULTIPLEXER

- Takes one input and selects one of many outputs to direct the input.



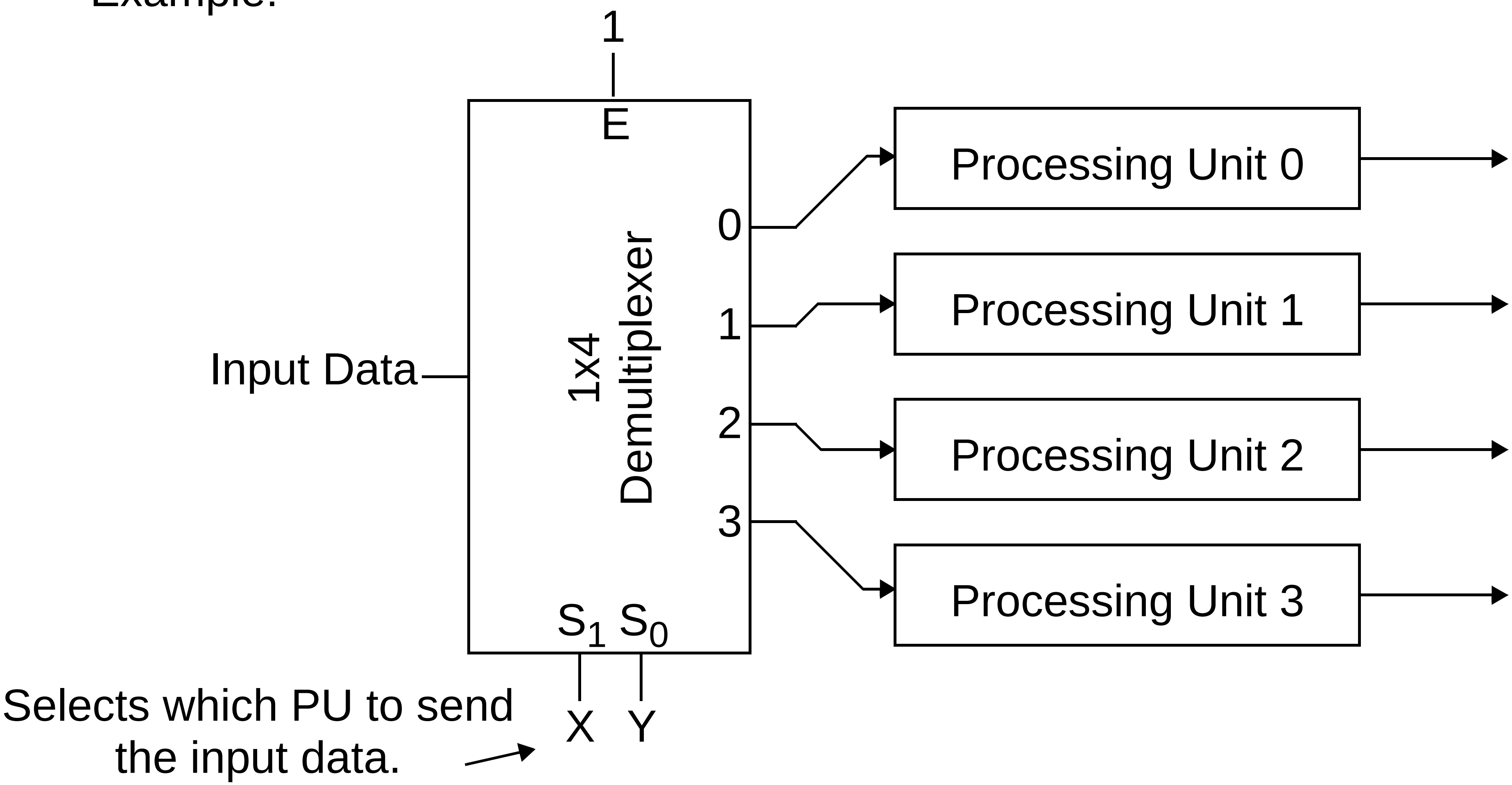
Inputs			Outputs			
X	Y	W	D_3	D_2	D_1	D_0
0	**0**	**0**	**0**	**0**	**0**	W=**0**
0	**1**	**0**	**0**	**0**	W=**0**	**0**
1	**0**	**0**	**0**	W=**0**	**0**	**0**
1	**1**	**0**	W=**0**	**0**	**0**	**0**
0	**0**	**1**	**0**	**0**	**0**	W=**1**
0	**1**	**1**	**0**	**0**	W=**1**	**0**
1	**0**	**1**	**0**	W=**1**	**0**	**0**
1	**1**	**1**	W=**1**	**0**	**0**	**0**

DEMULTIPLEXERS

DESIGN W/ DEMULTIPLEXERS

- A demultiplexer is useful for routing an input to a desired location.

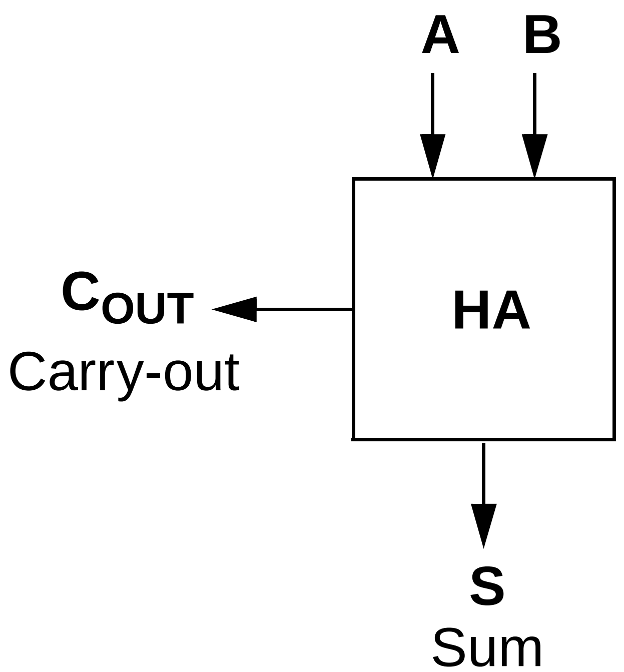
Example:



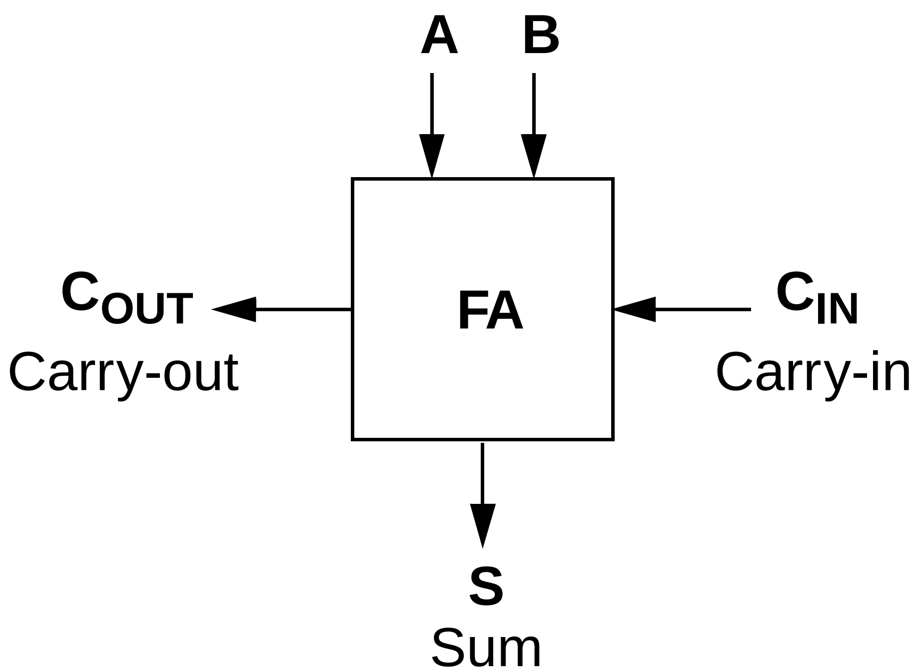
ADDERS

HALF- AND FULL-ADDERS

- Two basic building blocks for arithmetic are half- and full-adders as depicted by the block diagrams below.



Half-adder



Full-adder

ADDERS

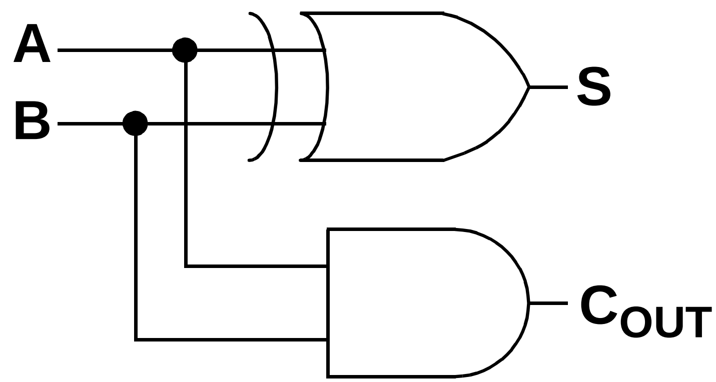
HALF-ADDER (HA)

- First of all, how do we add?
- 2’s complement arithmetic allows us to add numbers normally.

Inputs A	Inputs B	Sum S	Carry-out C_{OUT}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = \overline{A}B + A\overline{B} = A \oplus B$$

$$C_{OUT} = AB$$



ADDERS

FULL-ADDER (FA) (1)

- Half-adder missed a possible carry-in. A full-adder (FA) includes this additional carry-in.

Inputs A	Inputs B	Carry-in C_{IN}	Sum S	Carry-out 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

$$S = (A \oplus B) \oplus C_{IN}$$

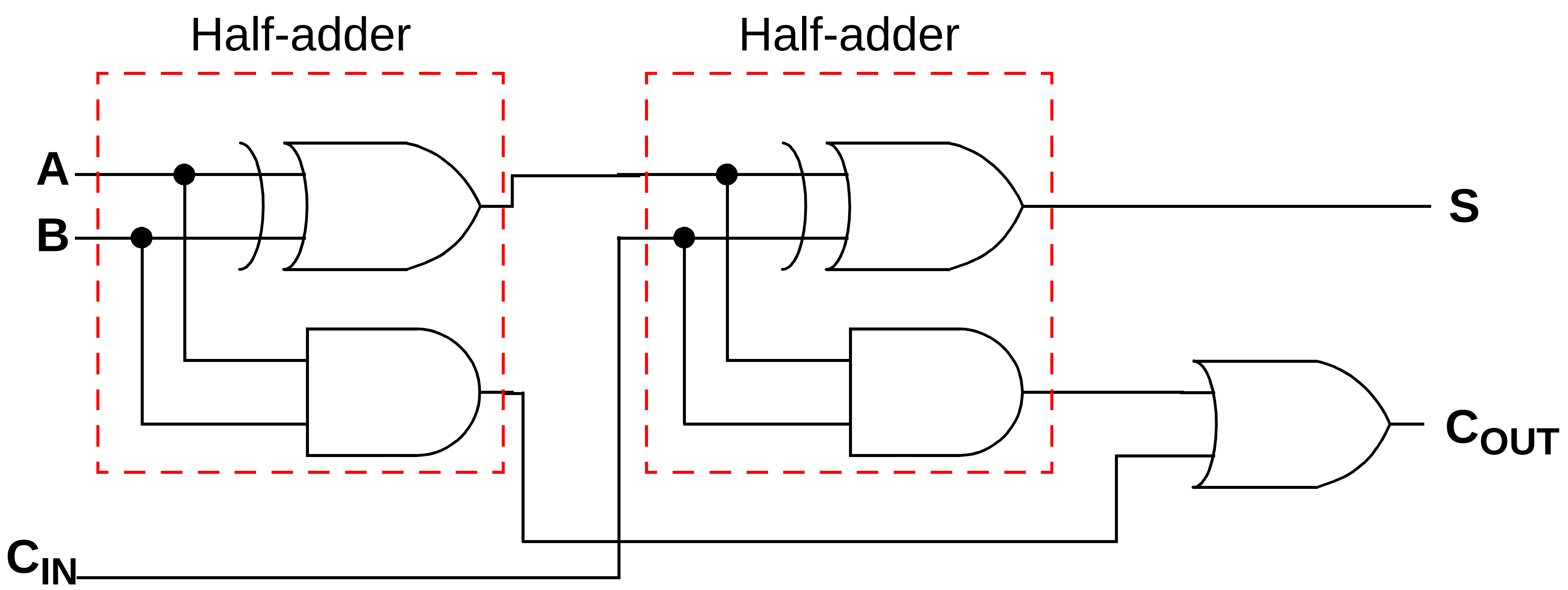
$$C_{OUT} = AB + C_{IN}(A \oplus B)$$

ADDERS

FULL-ADDER (FA) (2)

$$\mathbf{S} = (\mathbf{A} \oplus \mathbf{B}) \oplus \mathbf{C_{IN}}$$

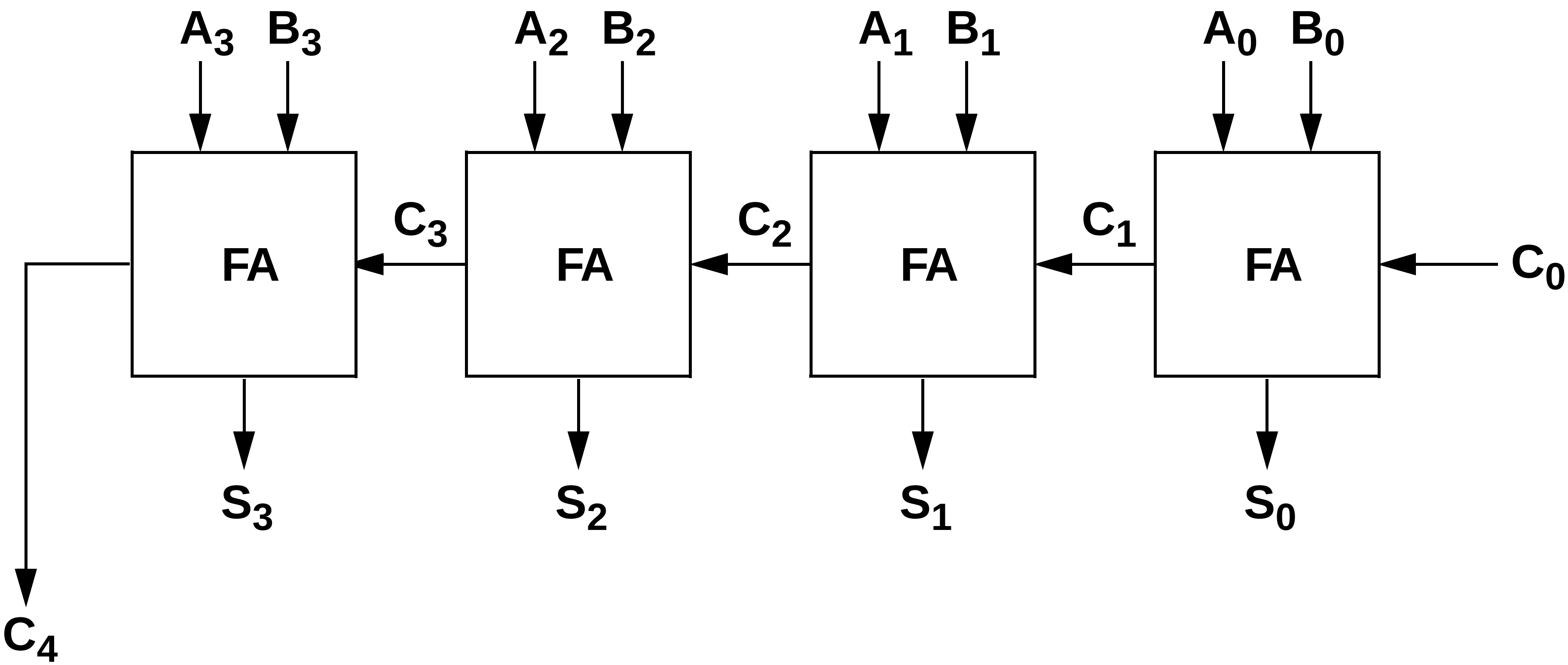
$$\mathbf{C_{OUT}} = \mathbf{AB} + \mathbf{C_{IN}}(\mathbf{A} \oplus \mathbf{B})$$



ADDERS

BINARY ADDITION

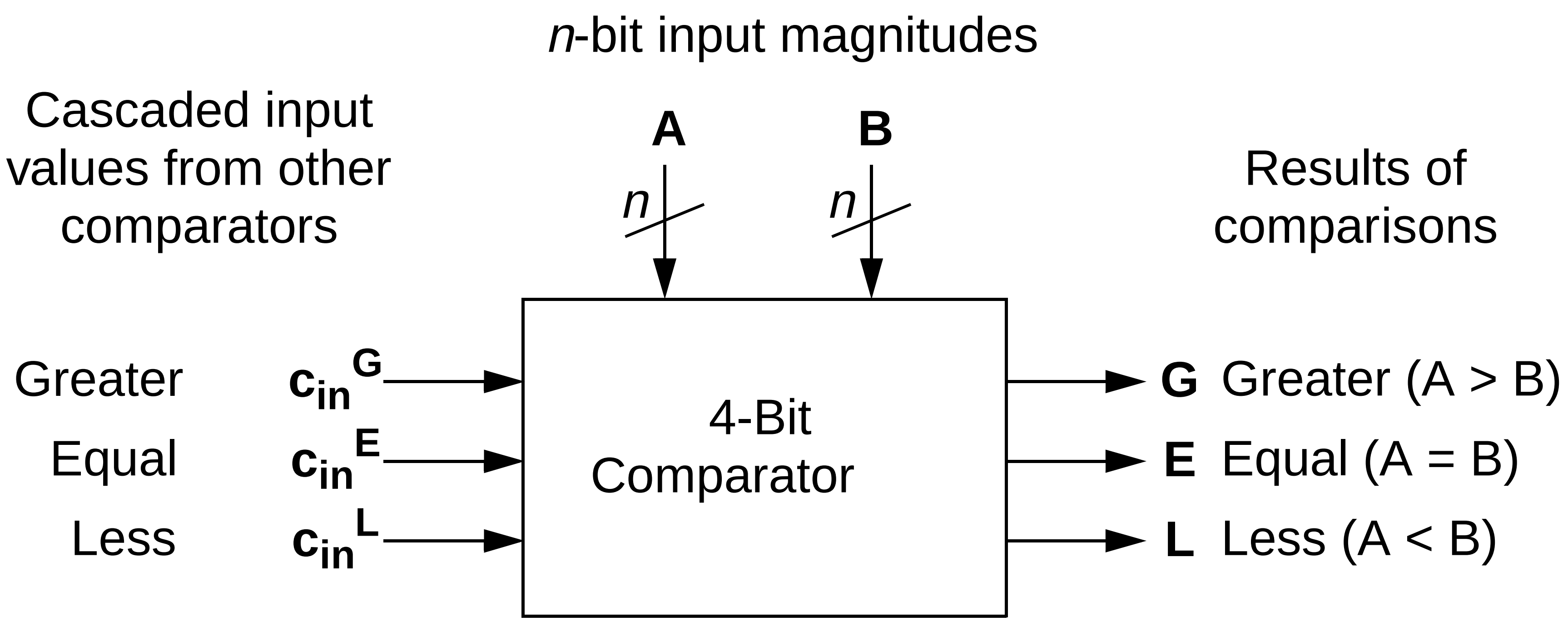
- A 4-bit binary adder can be formed with four full-adders as follows.



COMPARATORS

MAGNITUDE COMPARATOR

- Given two *n*-bit magnitudes, A and B, a comparator indicates whether
  - A = B, A > B, or A < B



COMPARATORS

MAGNITUDE COMPARATOR

- The approach is to use the XNOR function (equivalence) on each of the *n*-bits as follows

$$x_i = \mathbf{A}_i\mathbf{B}_i + \overline{\mathbf{A}}_i\overline{\mathbf{B}}_i = \overline{\mathbf{A}_i \oplus \mathbf{B}_i}$$

- The Boolean functions for a 4-bit magnitude comparator is as follows
  - $(\mathbf{A} = \mathbf{B}) = x_3x_2x_1x_0$
  - $(\mathbf{A} > \mathbf{B}) = \mathbf{A}_3\overline{\mathbf{B}_3} + x_3\mathbf{A}_2\overline{\mathbf{B}_2} + x_3x_2\mathbf{A}_1\overline{\mathbf{B}_1} + x_3x_2x_1\mathbf{A}_0\overline{\mathbf{B}_0}$
  - $(\mathbf{A} < \mathbf{B}) = \overline{\mathbf{A}_3}\mathbf{B}_3 + x_3\overline{\mathbf{A}_2}\mathbf{B}_2 + x_3x_2\overline{\mathbf{A}_1}\mathbf{B}_1 + x_3x_2x_1\overline{\mathbf{A}_0}\mathbf{B}_0$

Note: $\mathbf{A}_i\overline{\mathbf{B}}_i$ indicates whether $\mathbf{A}_i > \mathbf{B}_i$, $\overline{\mathbf{A}}_i\mathbf{B}_i$ indicates whether $\mathbf{A}_i < \mathbf{B}_i$, and x_i indicates whether $\mathbf{A}_i = \mathbf{B}_i$.