CPT_S 260 Intro to Computer Architecture
Lecture 23

Digital Design II
March 4, 2022

Ganapati Bhat

School of Electrical Engineering and Computer Science

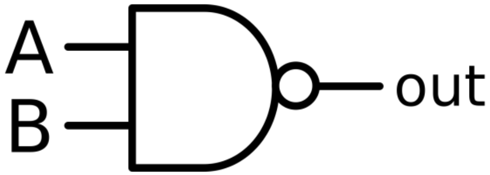
Washington State University

Announcements

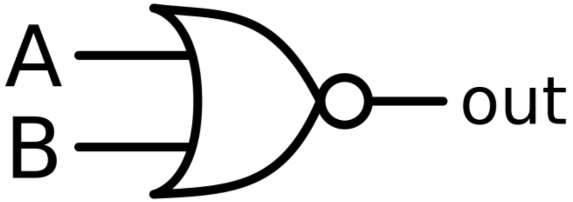
- Homework 4 due next week
- No quiz this week

Recap: NAND and NOR

- NAND : NOT of AND : A nand B = $\overline{A . B}$
- NOR : NOT of OR : A nor B = $\overline{A + B}$
- NAND and NOR are *universal gates*, i.e., they can be used to construct any complex logical function



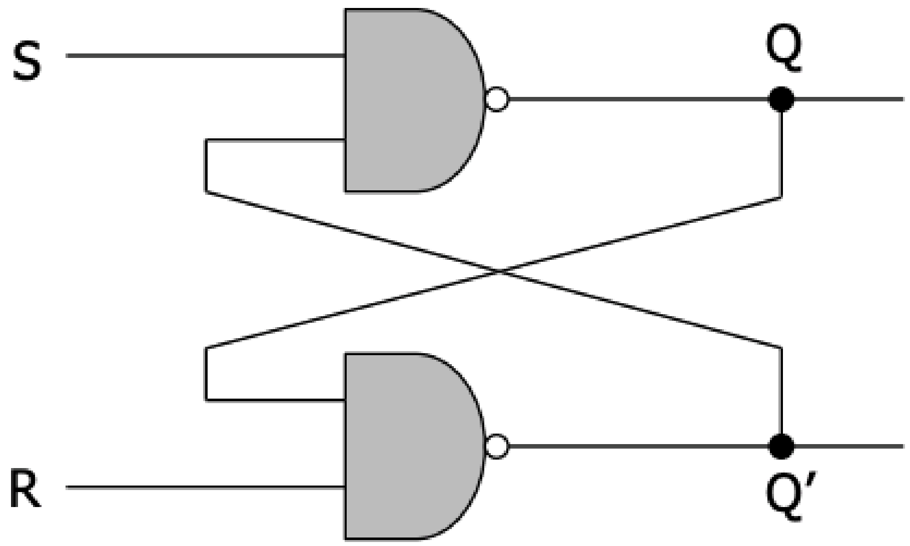
NAND



NOR

Recap: Stateful (Sequential) Digital Circuits

- This circuit utilizes one of the wire loops from the right to the left providing feedback
- "flip flop" building blocks to preserve state.



R-S Latch

Input		Output
R	S	Q
1	1	Q_{prev}
1	0	1
0	1	0
0	0	Invalid

Truth Tables

- *Truth table* – a tabular listing of the values of a function for all possible combinations of values on its arguments
- Example: Truth tables for the basic logic operations:

AND		
X	Y	$Z = X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1

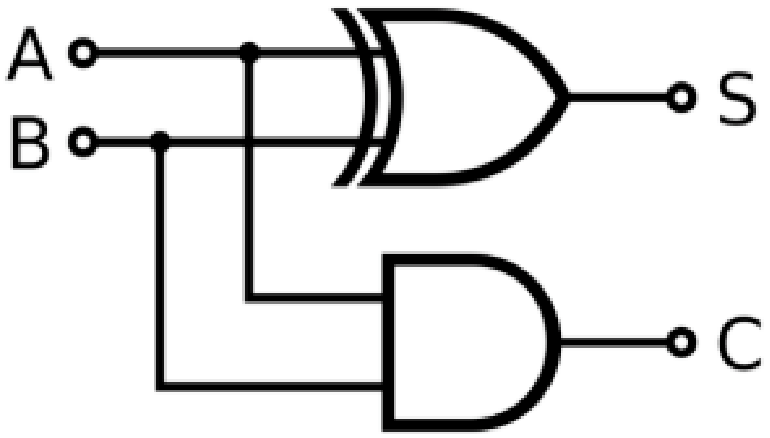
OR		
X	Y	$Z = X + Y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT	
X	$Z = \overline{X}$
0	1
1	0

Checkpoint – Truth tables

- S=$(A+B)\cdot(\bar{A}+\bar{B})$

- C=$A\cdot B$



Selecting

- **Selecting of data or information is a critical function in digital systems and computers**
- **Circuits that perform selecting have:**
  - A set of information inputs from which the selection is made
  - A single output
  - A set of control lines for making the selection
- **Logic circuits that perform selecting are called *multiplexers***
- **Selecting can also be done by three-state logic or transmission gates**

Multiplexers

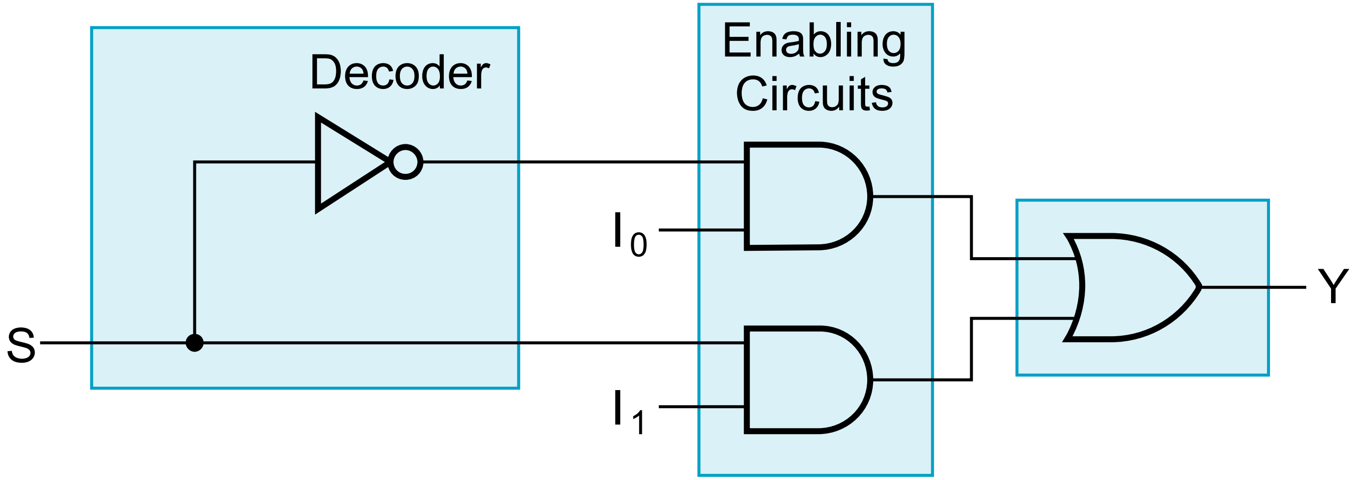
- **A multiplexer selects information from an input line and directs the information to an output line**
- **A typical multiplexer has *n* control inputs ($S_{n-1}, \dots S_0$) called *selection inputs*, 2^n information inputs ($I_{2^n-1}, \dots I_0$), and one output Y**
- **A multiplexer can be designed to have *m* information inputs with $m < 2^n$ as well as *n* selection inputs**

2-to-1-Line Multiplexer

- **Since 2 = 2^1, n = 1**
- **The single selection variable S has two values:**
  - S = 0 selects input I_0
  - S = 1 selects input I_1
- **The equation:**

$$Y = \bar{S} I_0 + S I_1$$

- **The circuit:**

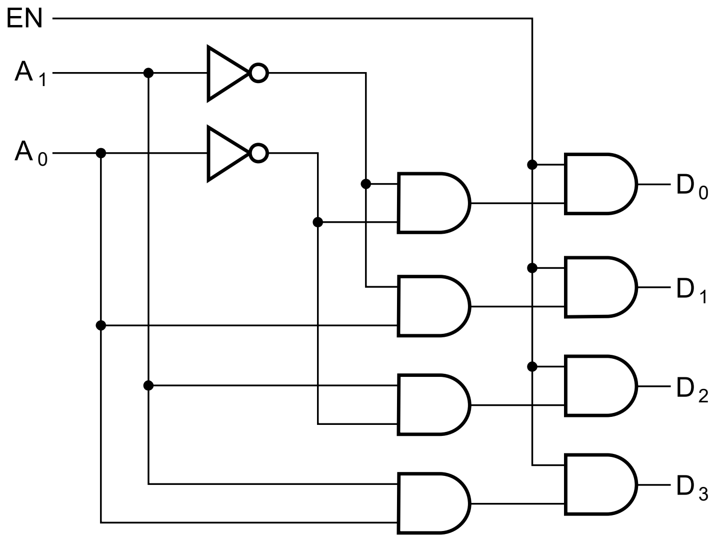


Decoders

- **Decoding - the conversion of an *n*-bit input code to an *m*-bit output code with n ≤ m ≤ 2^n such that each valid code word produces a unique output code**
- **Circuits that perform decoding are called *decoders***
- **Here, functional blocks for decoding are**
  - called n-to-m line decoders, where m ≤ 2^n, and
- **Example: 2-to-4 decoder:**

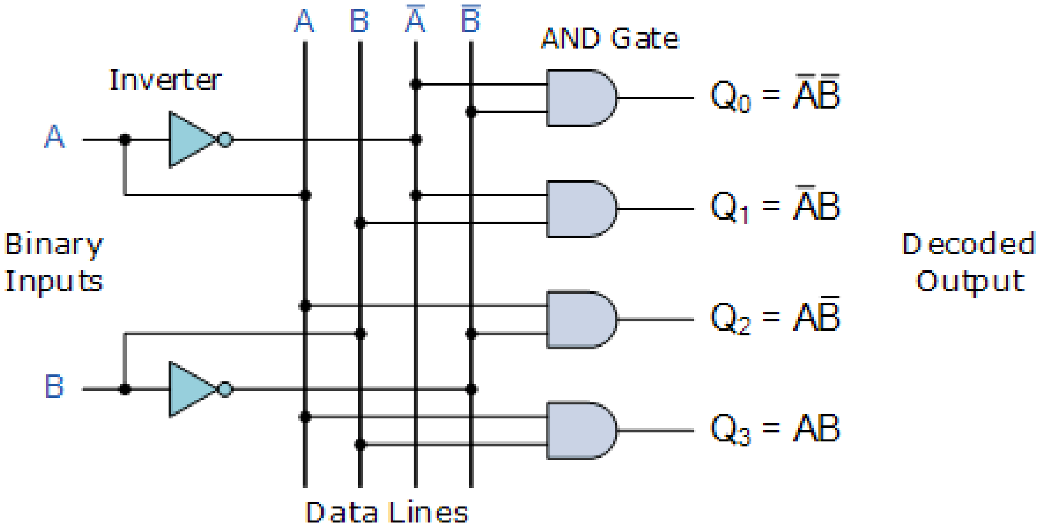
EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

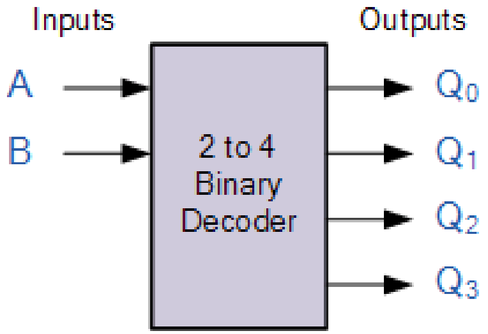
(a)



(b)

Decoder

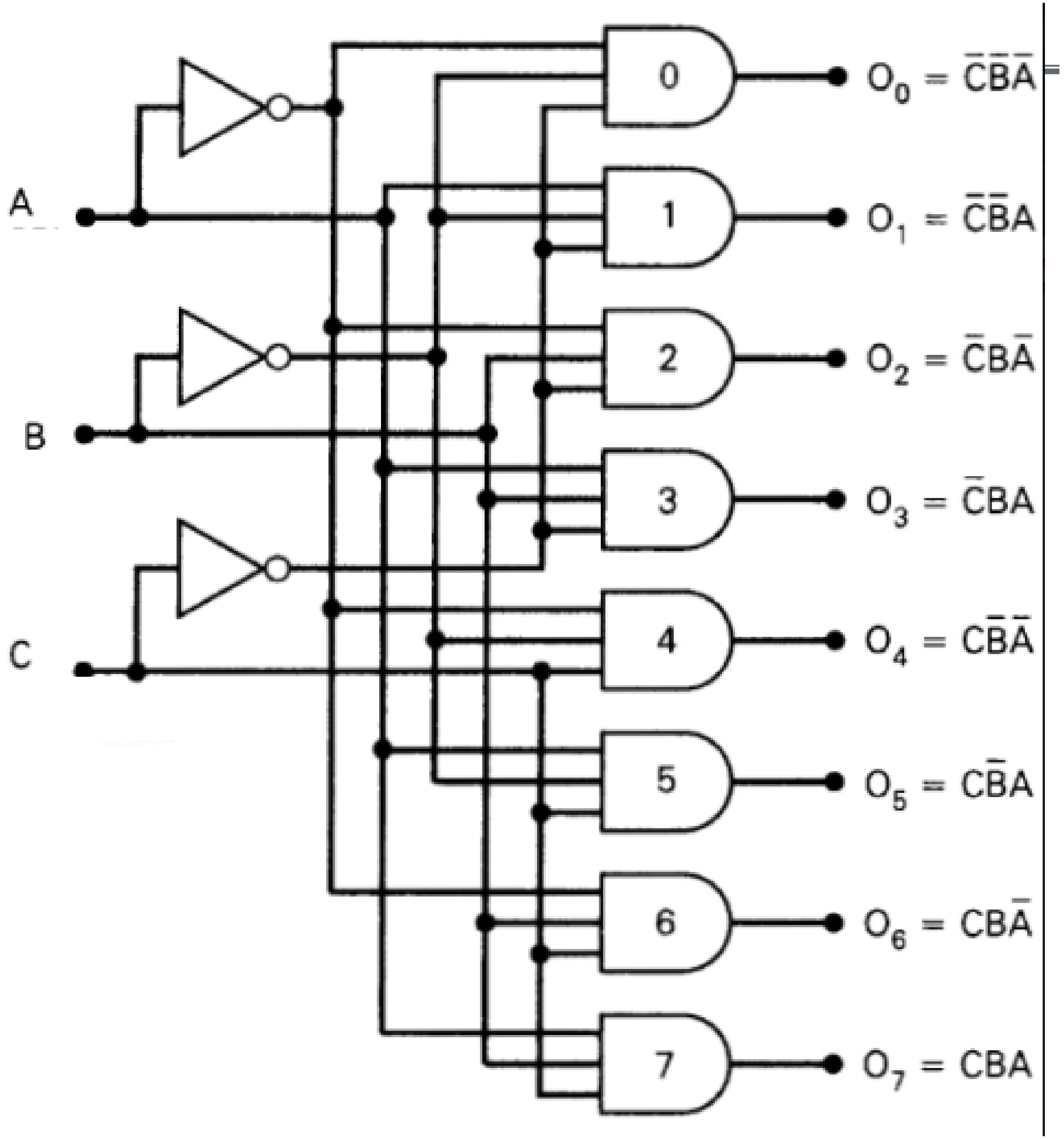




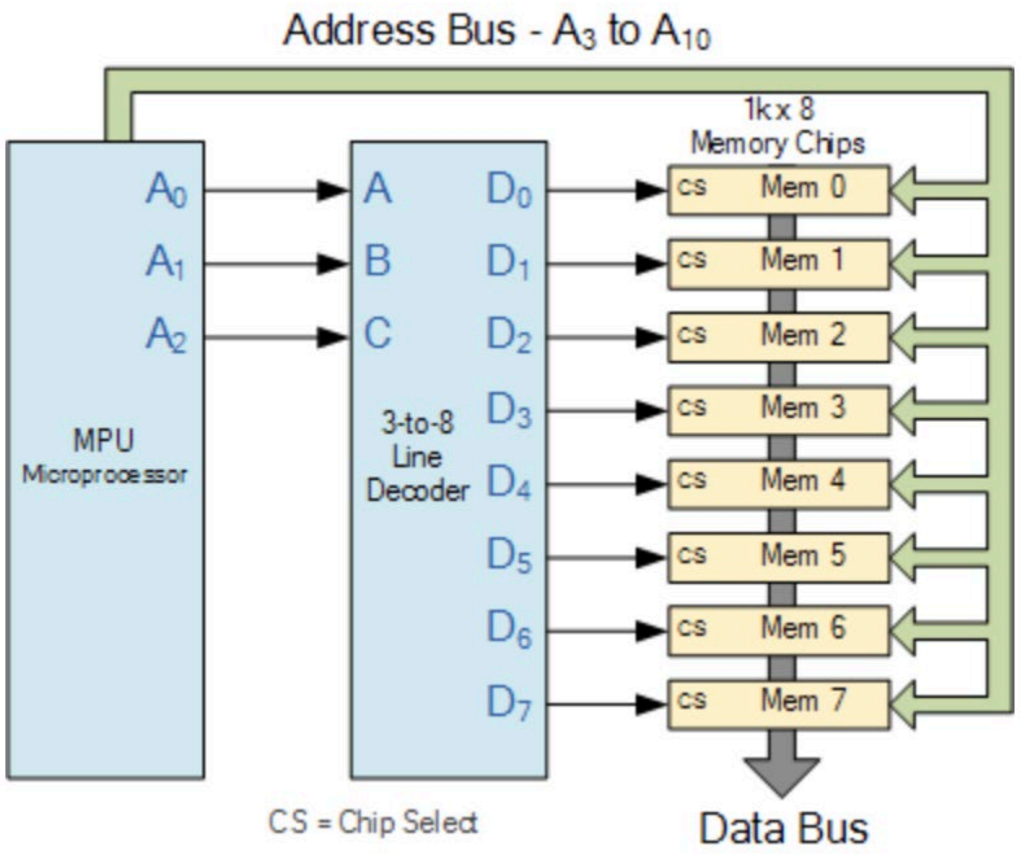
Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

3-to-8 Decoder



Memory Address Decoding



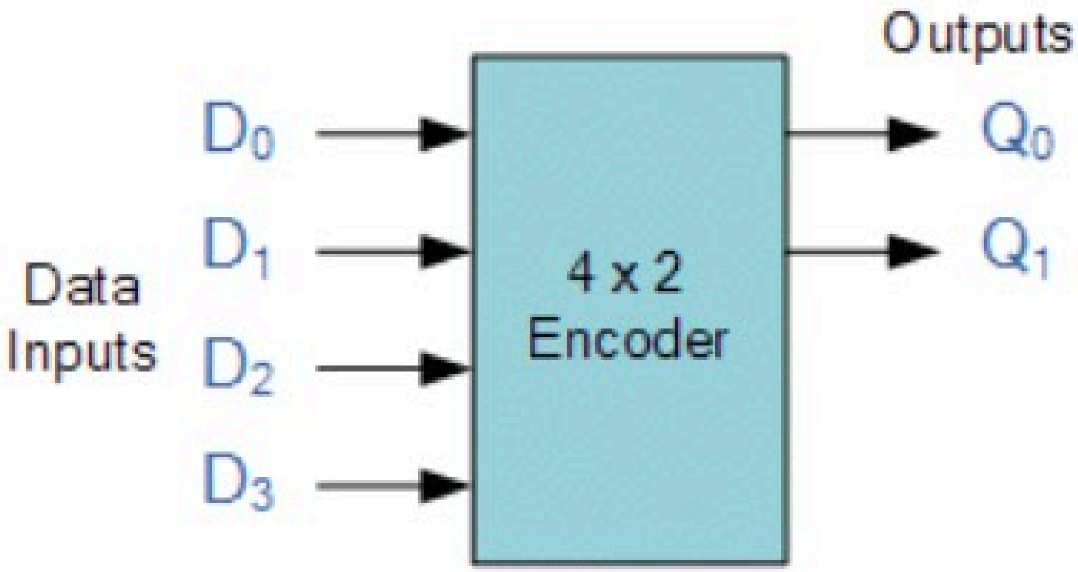
Encoder

- Encoding - the opposite of decoding - the conversion of an *m*-bit input code to a *n*-bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code

- An encoder has 2^n (or fewer) input lines and *n* output lines which generate the binary code corresponding to the input values

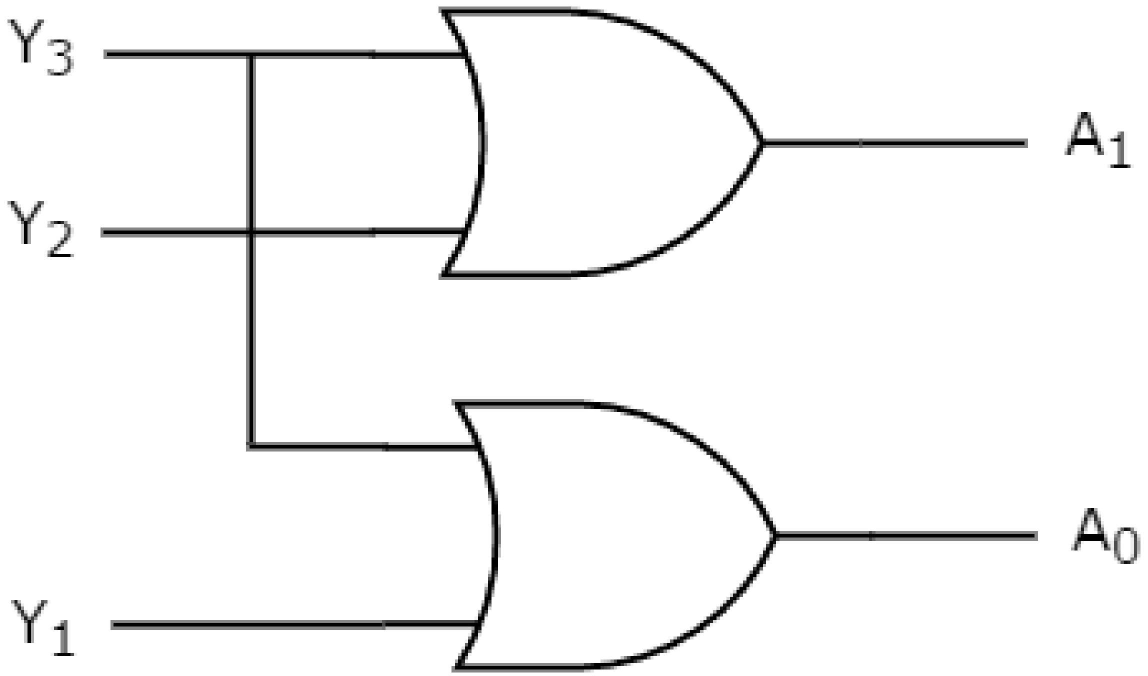
- Takes ALL its data inputs one at a time and then converts them into a single encoded output.

Encoder



Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

Encoder Digital Circuit



Logic Simplification

Rules of Boolean Algebra

- **Associative Law of multiplication**

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

- **Distributive Law of multiplication**

$$A + BC = (A + B) \cdot (A + C)$$

- **Annulment law:**

$$A \cdot 0 = 0$$
$$A + 1 = 1$$

- **Identity law:**

$$A \cdot 1 = A$$
$$A + 0 = A$$

Rules of Boolean Algebra

- **Complement law:**

$$A + \bar{A} = 1$$
$$A \cdot \bar{A} = 0$$

- **Double negation law:**

$$\bar{\bar{A}} = A$$

- **Absorption law:**

$$A \cdot (A + B) = A$$
$$A + AB = A$$
$$A + \bar{A}B = A + B$$

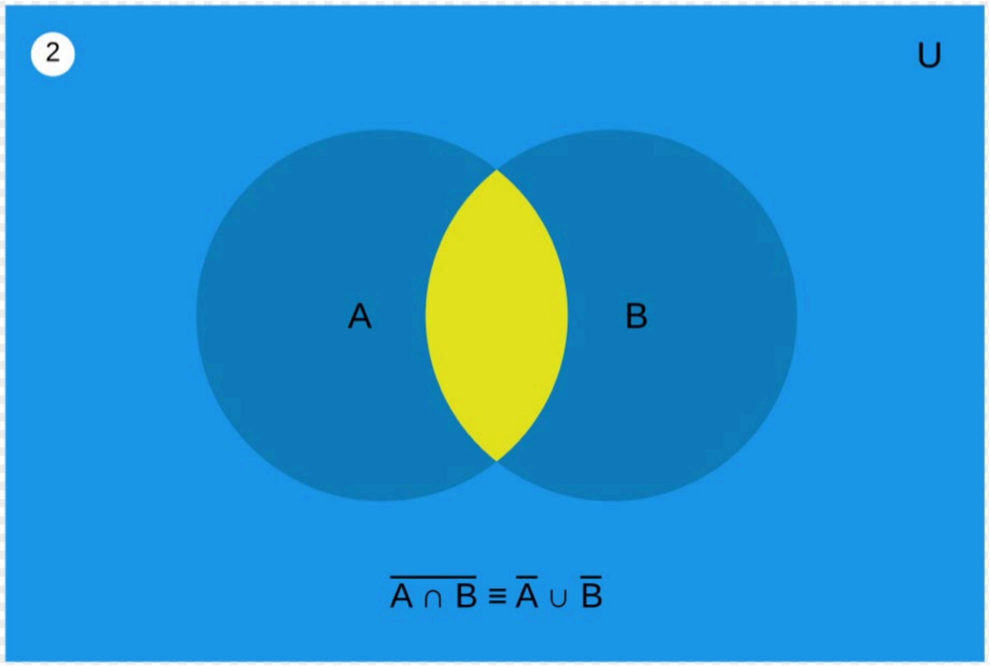
- **Idempotent law:**

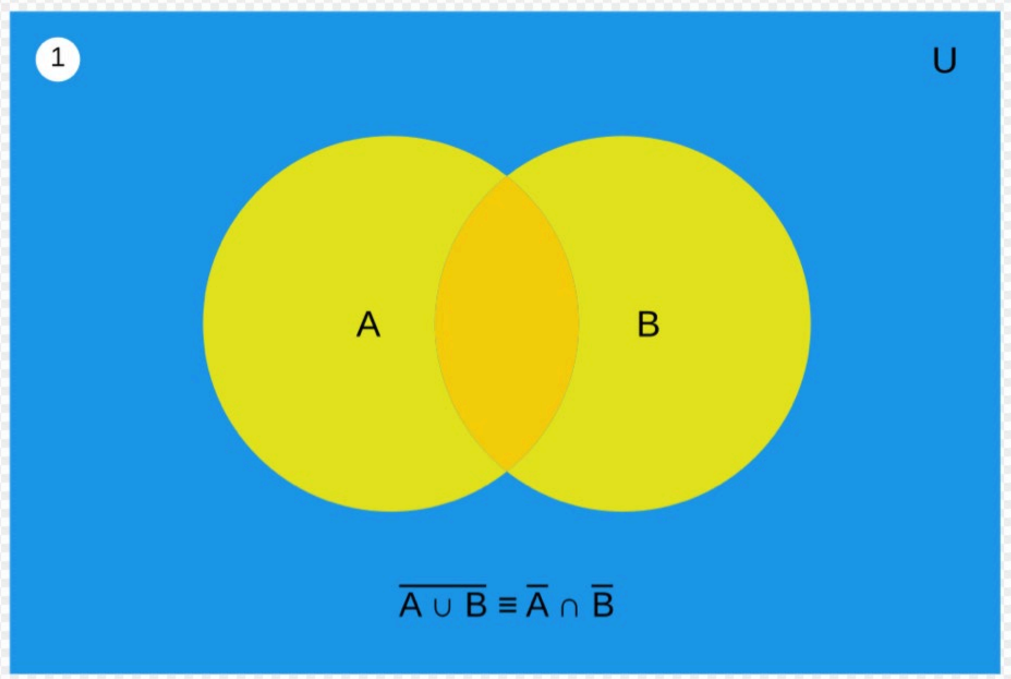
$$A + A = A$$
$$A \cdot A = A$$

De Morgan’s Laws

- **Transformation rules that help simplification of negations**
- **Statement:**

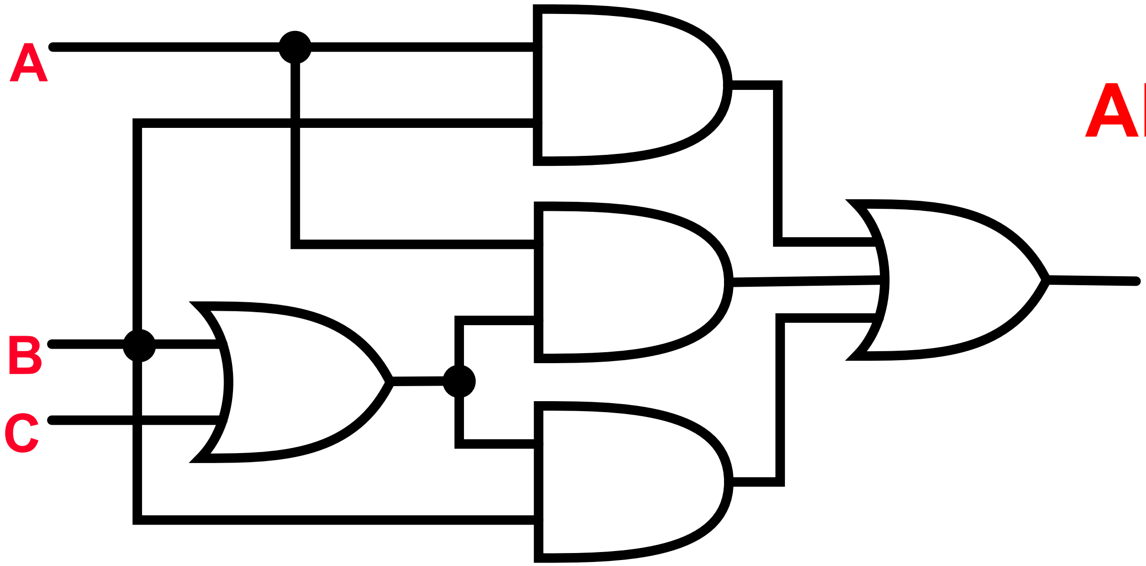
$$\overline{AB} = \bar{A} + \bar{B}$$
$$\overline{(A + B)} = \bar{A} \cdot \bar{B}$$





Simplification Using Boolean Algebra

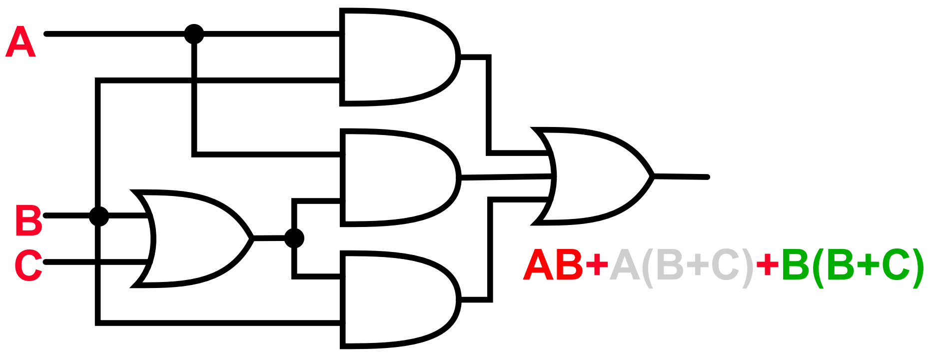
- **A simplified Boolean expression uses the fewest gates possible to implement a given expression.**

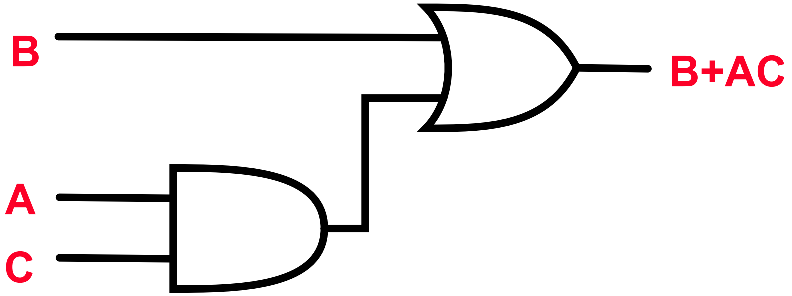


$AB + A(B+C) + B(B+C)$

Simplification Using Boolean Algebra

- **AB+A(B+C)+B(B+C)**
  - **(distributive law)**
    - » AB+AB+AC+BB+BC
  - (BB=B)
    - » AB+AB+AC+**B**+BC
  - (AB+AB=AB)
    - » **AB**+AC+B+BC
  - (B+BC=B)
    - » AB+AC+**B**
  - (AB+B=B)
    - » **B**+AC





Examples

- $[A\bar{B}(C + BD) + \bar{A}\bar{B}]C$

- $\bar{A}BC + A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + A\bar{B}C + ABC$

- $\overline{AB + AC} + \bar{A}\bar{B}C$

Standard Forms of Boolean Expressions

- **All Boolean expressions, regardless of their form, can be converted into either of two standard forms:**
  - The sum-of-products (SOP) form (minterms)
  - The product-of-sums (POS) form (maxterms)

- **Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier**

Sum of Products

- Minterm Expressions
- **If input is 0 we take the complement of the variable**
- **If input is 1 we take the variable as is**
- To get the desired canonical SOP expression we will add the minterms (product terms) for which the output is 1

$$F = \bar{A}B + A\bar{B} + AB$$

A	B	F	Minterm
0	0	0	A'B'
0	1	1	A'B
1	0	1	AB'
1	1	1	AB

Product of Sums

- **Maxterm Expressions**
- **If input is 1, we take the complement of the variable**
- **If input is 0, we take the variable as is**
- **To get the desired canonical POS expression we will multiply the maxterms (sum terms) for which the output is 0**

$$F = (A + B) \,.\, (\bar{A} + \bar{B})$$

A	B	F	Minterm
0	0	0	A'B'
0	1	1	A'B
1	0	1	AB'
1	1	1	AB