HARDWARE FOR ARITHMETIC

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

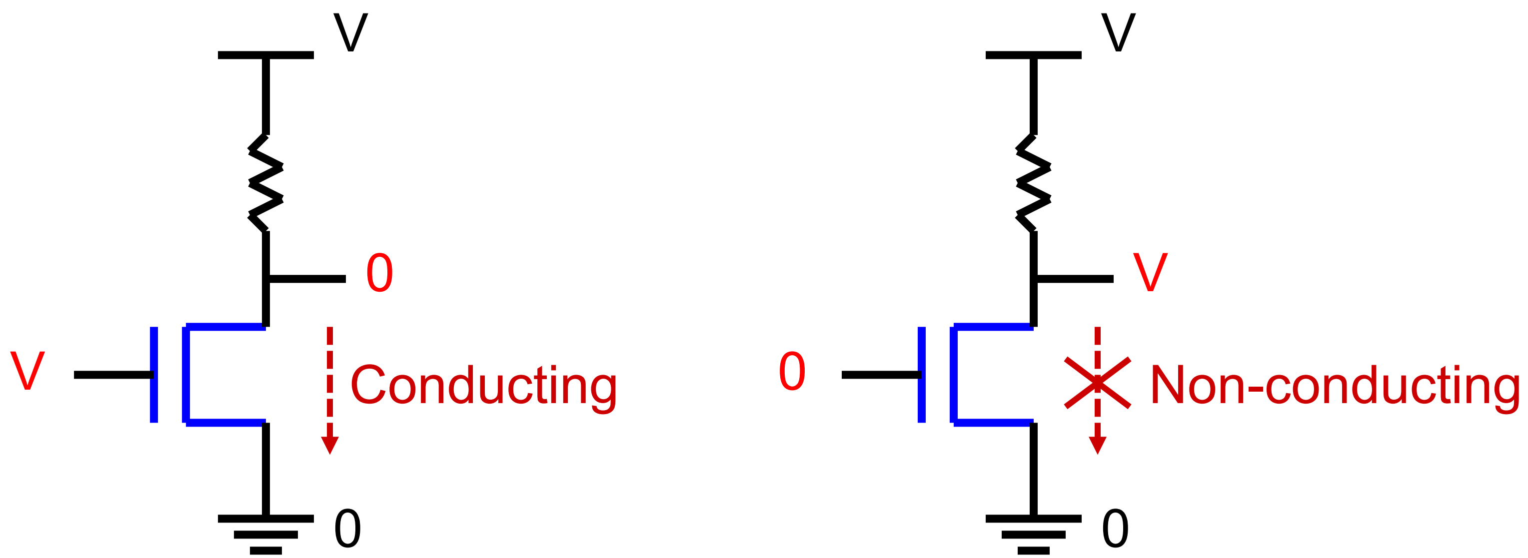
Overview

- Past lectures
  - High-level stuff mostly on ISA, Assembly, and number representation

- This lecture
  - Basics of logic design
  - Hardware for arithmetic

Fundamentals of Digital Design

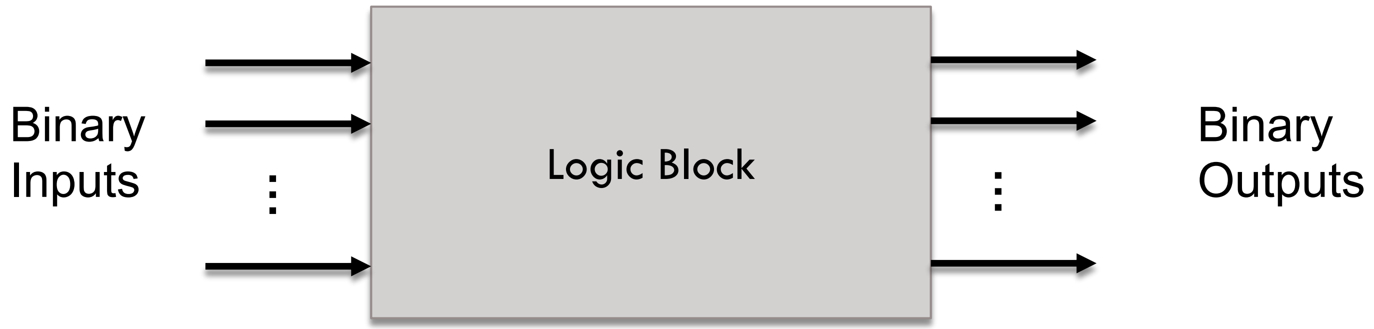
- Binary logic: two voltage levels
  - high and low; 1 and 0; true and false
- Binary arithmetic
  - Based on a 3-terminal device that acts as a switch



Logic Blocks

- A logic block comprises binary inputs and binary outputs
  - Combinational: the output is only a function of the inputs
  - Sequential: the block has some internal memory (state) that also influences the output

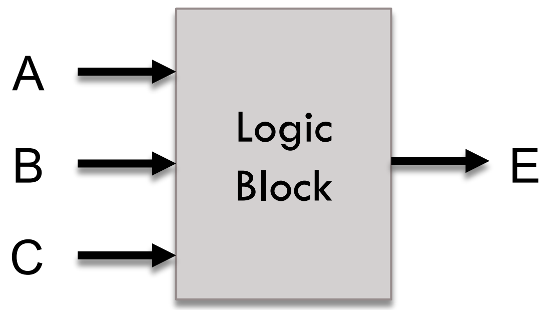
- Gate: a basic logic block that implements AND, OR, NOT, etc.



Logic Blocks: Truth Table

- A truth table defines the outputs of a logic block for each set of inputs
  - Example: consider a block with 3 inputs A, B, C and an output E that is true only if exactly 2 inputs are true

A	B	C	E
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



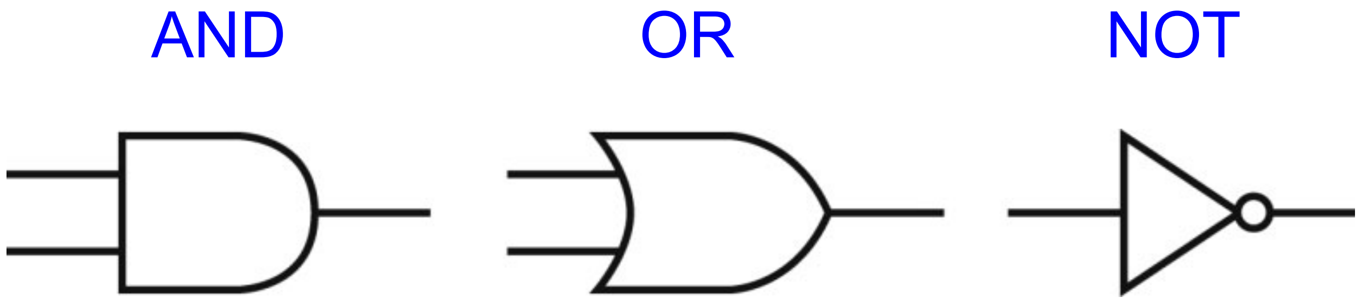
Boolean Algebra

- Three primary operators are used to realize Boolean functions

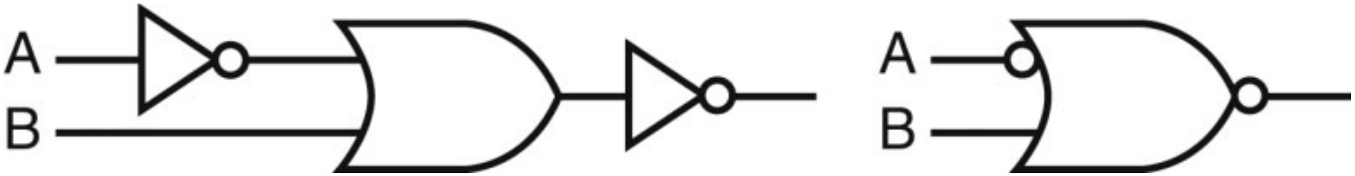
- Boolean operations
  - OR (symbol +)
    - $X = A + B$: X is true if at least one of A or B is true
  - AND (symbol .)
    - $X = A . B$: X is true if both A and B are true
  - NOT (symbol ¯)
    - $X = \overline{A}$: X is the inverted value of A

Pictorial Representation

- Logic gates



- What function is the following?

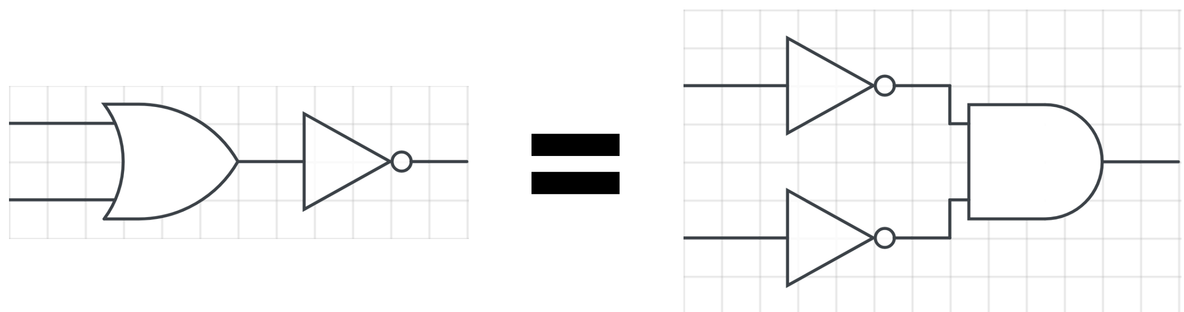


Boolean Algebra Rules

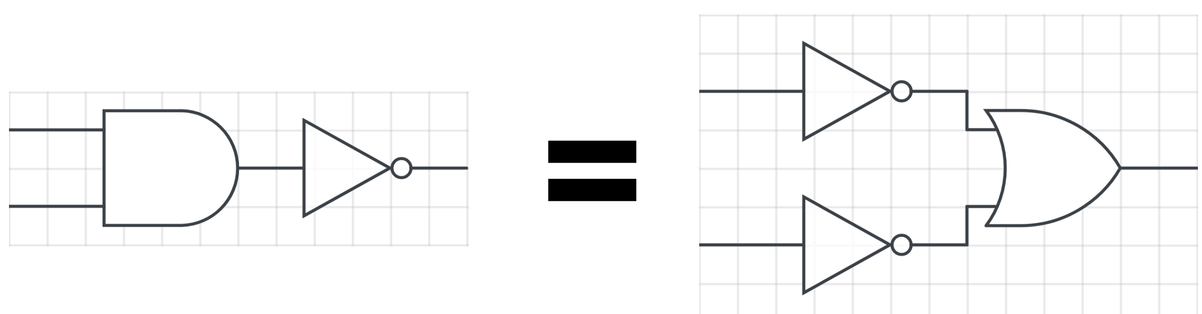
- Identity law
  - $A + 0 = A$
  - $A \,.\, 1 = A$
- Commutative laws
  - $A + B = B + A$
  - $A \,.\, B = B \,.\, A$
- Zero and One laws
  - $A + 1 = 1$
  - $A \,.\, 0 = 0$
- Associative laws
  - $A + (B + C) = (A + B) + C$
  - $A \,.\, (B \,.\, C) = (A \,.\, B) \,.\, C$
- Inverse laws
  - $A \,.\, \overline{A} = 0$
  - $A + \overline{A} = 1$
- Distributive laws
  - $A \,.\, (B + C) = (A \,.\, B) + (A \,.\, C)$
  - $A + (B \,.\, C) = (A + B) \,.\, (A + C)$

DeMorgan's Law

- $\overline{A + B} = \overline{A} \, . \, \overline{B}$



- $\overline{A \, . \, B} = \overline{A} + \overline{B}$



Example: Boolean Equation

- Consider the logic block that has an output E that is true only if exactly two of the three inputs A, B, C are true

- Multiple correct equations
  - Two must be true, but all three cannot be true
    - $E = ((A \,.\, B) + (B \,.\, C) + (A \,.\, C)) \,.\, \overline{(A \,.\, B \,.\, C)}$
  - Identify the three cases where it is true
    - $E = (A \,.\, B \,.\, \overline{C}) + (A \,.\, C \,.\, \overline{B}) + (C \,.\, B \,.\, \overline{A})$

Implementing Boolean Functions

- Can realize any logic block with the AND, OR, NOT
  - Draw the truth table
  - For each true output, represent the corresponding inputs as a product
  - The final equation is a sum of these products

A	B	C	E
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Implementing Boolean Functions

- Can realize any logic block with the AND, OR, NOT
  - Draw the truth table
  - For each true output, represent the corresponding inputs as a product
  - The final equation is a sum of these products

A	B	C	E	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}.B.C$
1	0	0	0	
1	0	1	1	$A.\overline{B}.C$
1	1	0	1	$A.B.\overline{C}$
1	1	1	0	

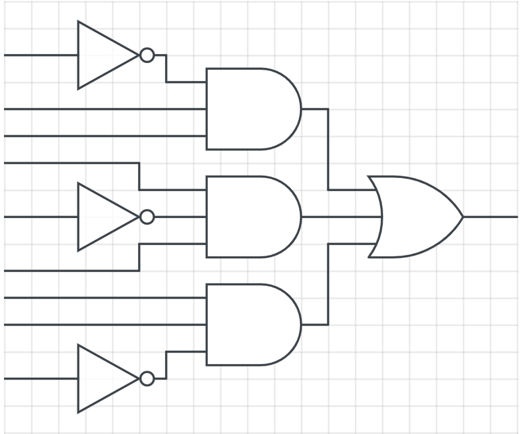
Implementing Boolean Functions

- Can realize any logic block with the AND, OR, NOT
  - Draw the truth table
  - For each true output, represent the corresponding inputs as a product
  - The final equation is a sum of these products

A	B	C	E	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}.B.C$
1	0	0	0	
1	0	1	1	$A.\overline{B}.C$
1	1	0	1	$A.B.\overline{C}$
1	1	1	0	

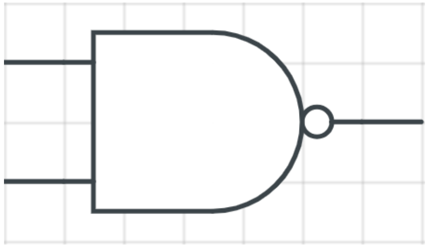
Sum of Products

$$E = (\overline{A}.B.C) + (A.\overline{B}.C) + (A.B.\overline{C})$$

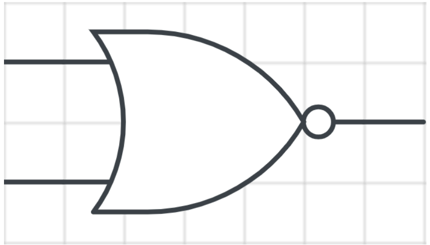


Universal Gates

- Universal gate is a logic that can be used to implement any complex function
  - NAND
    - Not of AND
    - A *nand* B = $\overline{A.B}$

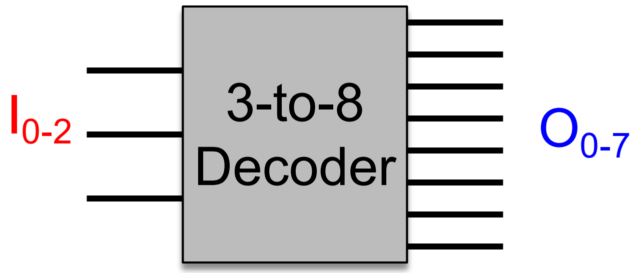


  - NOR
    - Not of OR
    - A *nor* B = $\overline{A+B}$



Common Logic Block

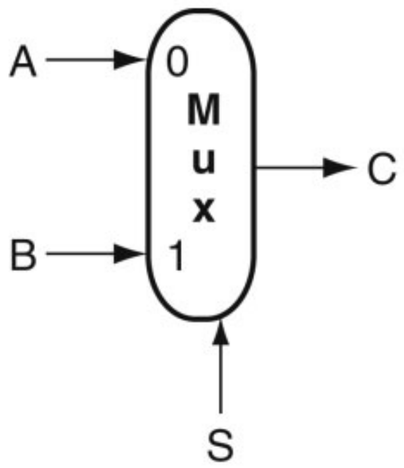
- An n-input decoder takes n inputs, based on which only one out of 2^n outputs is activated

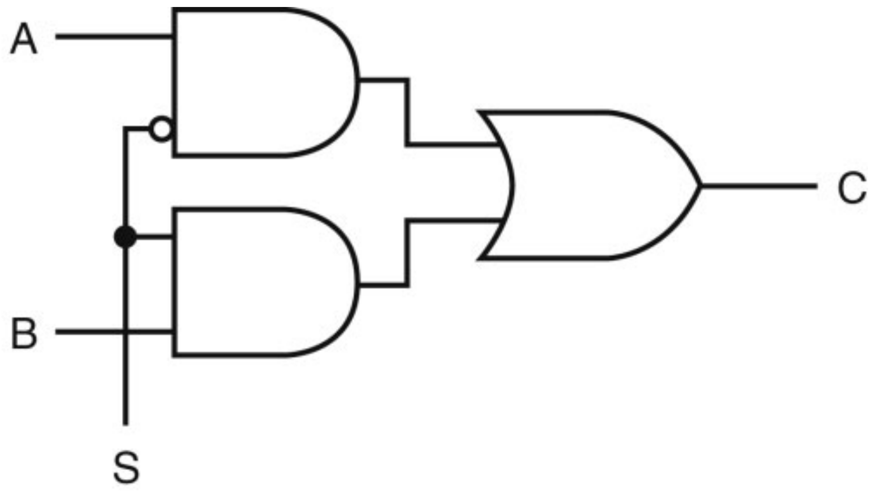


I_0	I_1	I_2	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Common Logic Block

- A multiplexer (or selector) reflects one of n inputs on the output depending on the value of the select bits
  - Example: 2-input mux





Common Logic Block

- A full adder generates the sum and carry for each bit position

1	0	0	1
0	1	0	1

Sum
Cout

A	B	C_{in}	Sum	C_{out}
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Common Logic Block

- A full adder generates the sum and carry for each bit position

A	B	C_{in}	Sum	C_{out}
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

	1	0	0	1
	0	1	0	1
Sum	1	1	1	0
Cout	0	0	0	1

Common Logic Block

- A full adder generates the sum and carry for each bit position

	1	0	0	1
	0	1	0	1
Sum	1	1	1	0
Cout	0	0	0	1

A	B	C_{in}	Sum	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

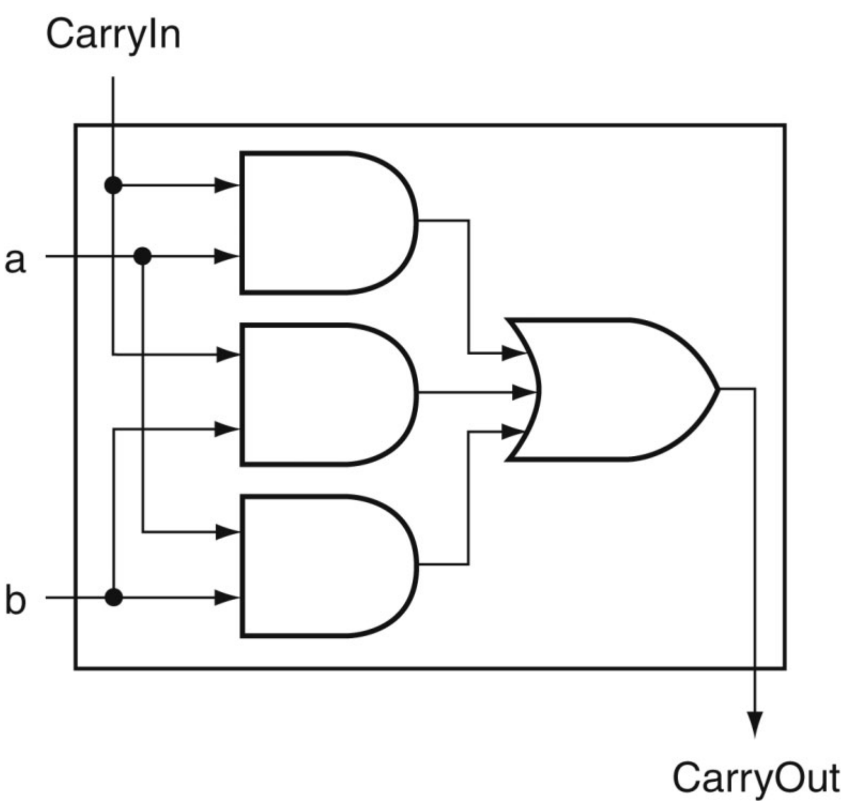
Equations:

$$\text{Sum} = C_{in}.\overline{A}.\overline{B} + B.\overline{C}_{in}.\overline{A} + A.\overline{C}_{in}.\overline{B} + A.B.C_{in}$$

$$\text{Cout} = A.B.C_{in} + A.B.\overline{C}_{in} + A.C_{in}.\overline{B} + B.C_{in}.\overline{A} = A.B + A.C_{in} + B.C_{in}$$

Common Logic Block

- A full adder generates the sum and carry for each bit position



	1	0	0	1
	0	1	0	1
Sum	1	1	1	0
Cout	0	0	0	1

Equations:

$$Sum = C_{in}.\overline{A}.\overline{B} + B.\overline{C_{in}}.\overline{A} + A.\overline{C_{in}}.\overline{B} + A.B.C_{in}$$

$$Cout = A.B.C_{in} + A.B.\overline{C_{in}} + A.C_{in}.\overline{B} + B.C_{in}.\overline{A} = A.B + A.C_{in} + B.C_{in}$$