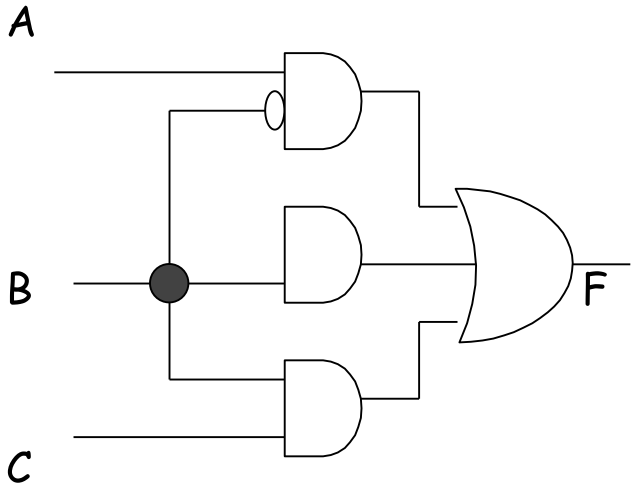
Simplification of Boolean functions

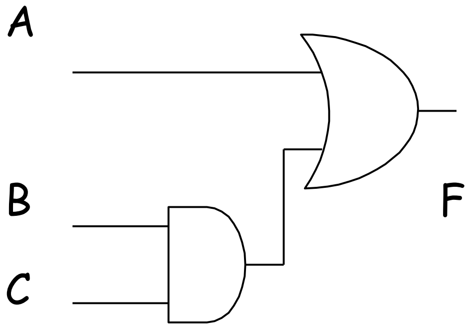
Using the theorems of Boolean Algebra, the algebraic forms of functions can often be simplified, which leads to simpler (and cheaper) implementations.

Example 1

$$\begin{aligned} F &= A.\overline{B} + A.B + B.C \\ &= A.\ (\overline{B} + B) + B.C \\ &= A.1 + B.C \\ &= A + B.C \end{aligned}$$

How many gates do you save from this simplification?





Example 2

$$F = \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$$

$$= \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C + A.B.C + A.B.C$$

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

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

$$= B.C + C.A + A.B$$

Example 3

Show that $A + A.B = A$

$$A + AB$$

$$= A.1 + A.B$$

$$= A.\ (1 + B)$$

$$= A.\ 1$$

$$= A$$

Simplification using Karnaugh Maps

A

B	0	1
1	0	1
0	1	1

K-map of 2-variable OR function

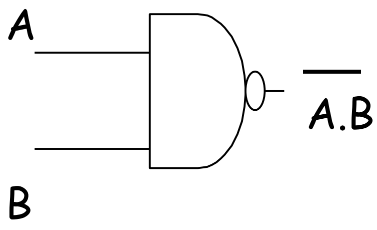
BC

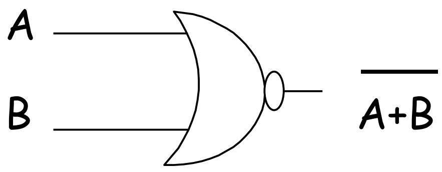
A	00	01	11	10
0			1	
1		1	1	1

K-map of majority function

Follow the class lectures to understand how to simplify Boolean functions using K-maps. Several examples will be worked out in the class.

Other types of gates

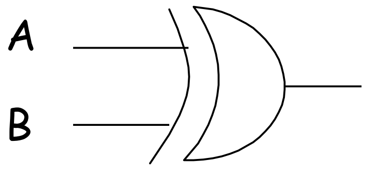




NAND gate

NOR gate

Be familiar with the truth tables of these gates.



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

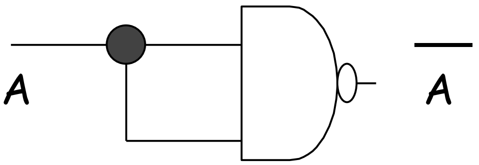
Exclusive OR (XOR) gate

NAND and NOR are universal gates

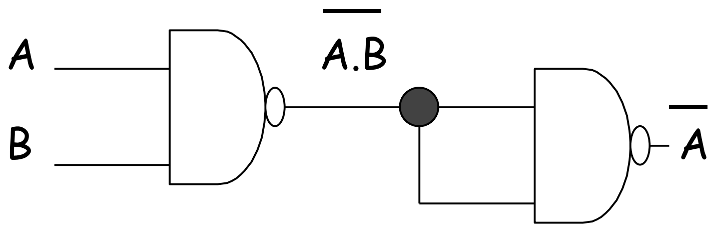
Any function can be implemented using **only NAND** or **only NOR** gates. How can we prove this?

(Proof for NAND gates) Any boolean function can be implemented using AND, OR and NOT gates. So if AND, OR and NOT gates can be implemented using NAND gates only, then we prove our point.

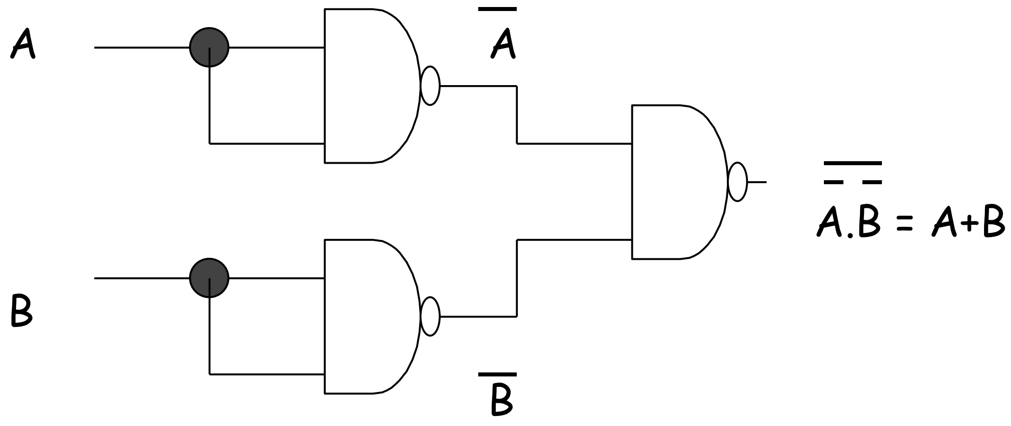
1. Implement NOT using NAND



2. Implementation of AND using NAND



1. Implementation of OR using NAND



(Exercise) Prove that NOR is a universal gate.

Example (to be worked out in class)

How to convert any circuit that uses AND, OR and NOT gates to a version that uses NAND (or NOR gates only)?

Additional properties of XOR

XOR is also called modulo-2 addition. Why?

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

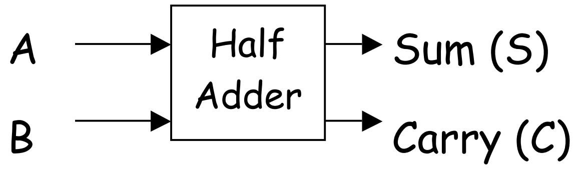
$A \oplus B = 1$ only when there are an odd number of 1's in (A,B). The same is true for $A \oplus B \oplus C$ also.

$$\left.\begin{array}{l} 1 \oplus A = \overline{A} \\ 0 \oplus A = A \end{array}\right\}$$

Why?

Logic Design Exercise

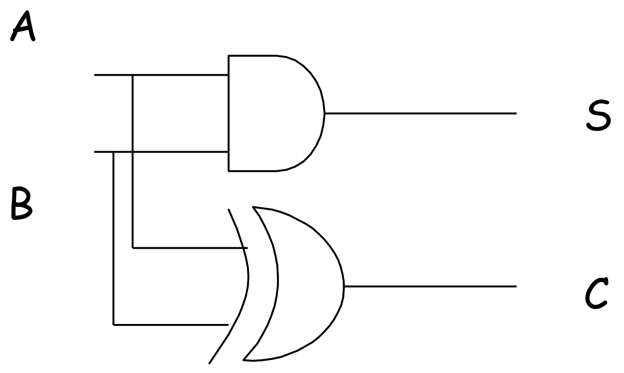
Half Adder



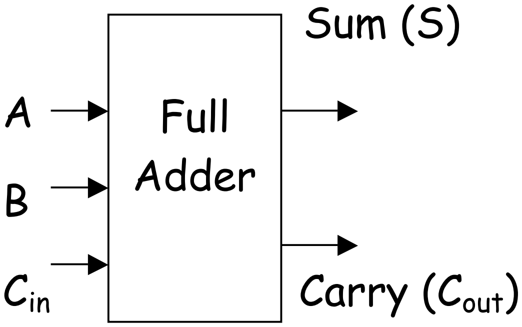
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$S = A \oplus B$

$C = A.B$



Full Adder



A	B	C	S	Cout
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

$S = A \oplus B \oplus C_{in}$

$C_{out} = A.B + B.C_{in} + A.C_{in}$

How can you add two 32-bit numbers? It will be discussed in the class.

Combinational vs. Sequential Circuits

Combinational circuits.

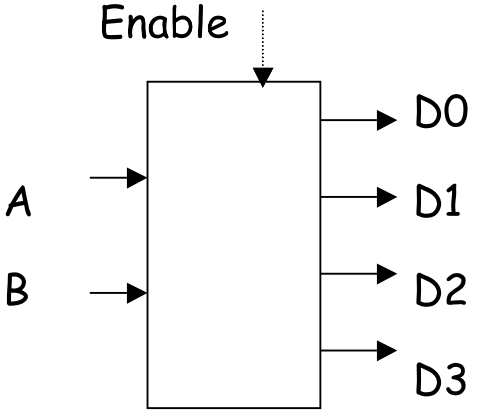
The output depends only on the current values of the inputs and not on the past values. Examples are adders, subtractors, and all the circuits that we have studied so far

Sequential circuits.

The output depends not only on the current values of the inputs, but also on their past values. These hold the secret of how to memorize information. We will study sequential circuits later.

Decoders

A typical decoder has n inputs and 2^n outputs.



A	B	D3	D2	D1	D0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

A 2-to-4 decoder and its truth table.

$D3 = A.B$

$D2 = A.\overline{B}$

$D1 = \overline{A}.B$

$D0 = \overline{A}.\overline{B}$

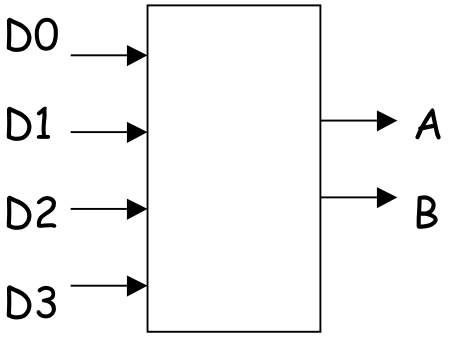
Draw the circuit of this decoder.

The decoder works per specs when (**Enable = 1**). When **Enable = 0**, all the outputs are 0.

Exercise. Design a 3-to-8 decoder.

Encoders

A typical encoder has 2^n inputs and n outputs.



D0	D1	D2	D3	A	B
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

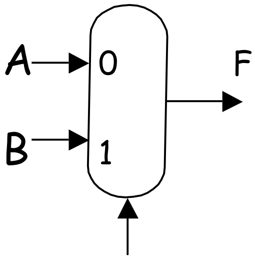
A 4-to-2 encoder and its truth table.

$$A = D1 + D3$$

$$B = D2 + D3$$

Multiplexor

It is a **many-to-one switch**, also called a **selector**.



Control S

$S = 0, F = A$

$S = 1, F = B$

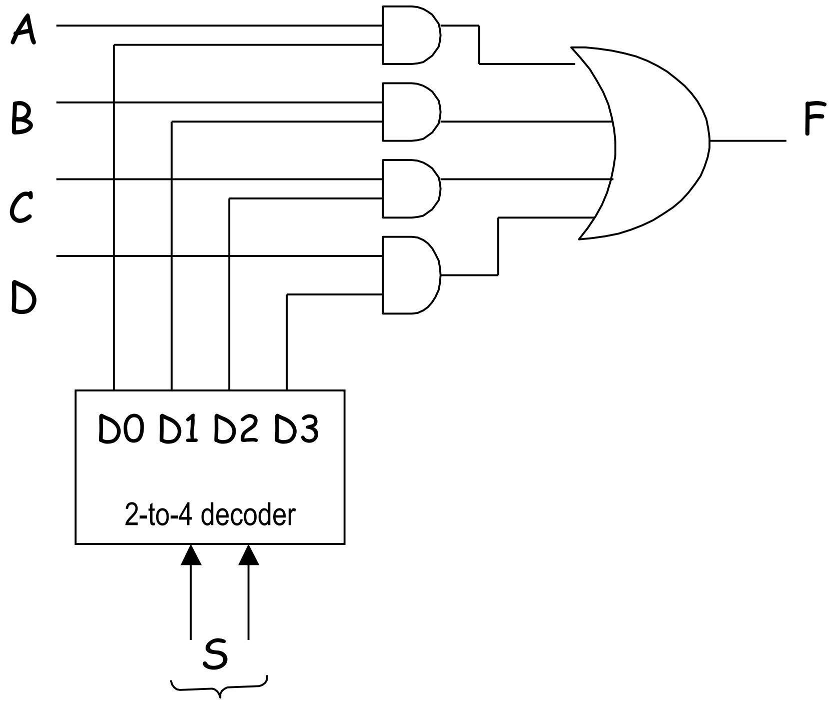
Specifications of the mux

A 2-to-1 mux

$$F = \overline{S}.\ A + S.\ B$$

Exercise. Design a 4-to-1 mux.

Another design of a decoder



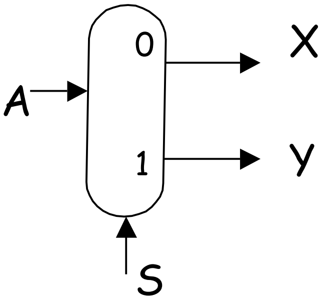
Exercise 1. Design a 2-to-4 decoder using 1-to-2 decoders only.

Exercise 2. Design a 4-to-1 multiplexor using 2-1 multiplexors only.

To be discussed in the class.

Demultiplexors

A demux is a one-to-many switch.



S = 0, X = A

S = 1, Y = B

A 1-to-2 demux, and its specification.

So, $X = \overline{S}.\ A$, and $Y = S.\ B$

Exercise. Design a 1-4 demux.

We will discuss the design of a 1-bit ALU in class.