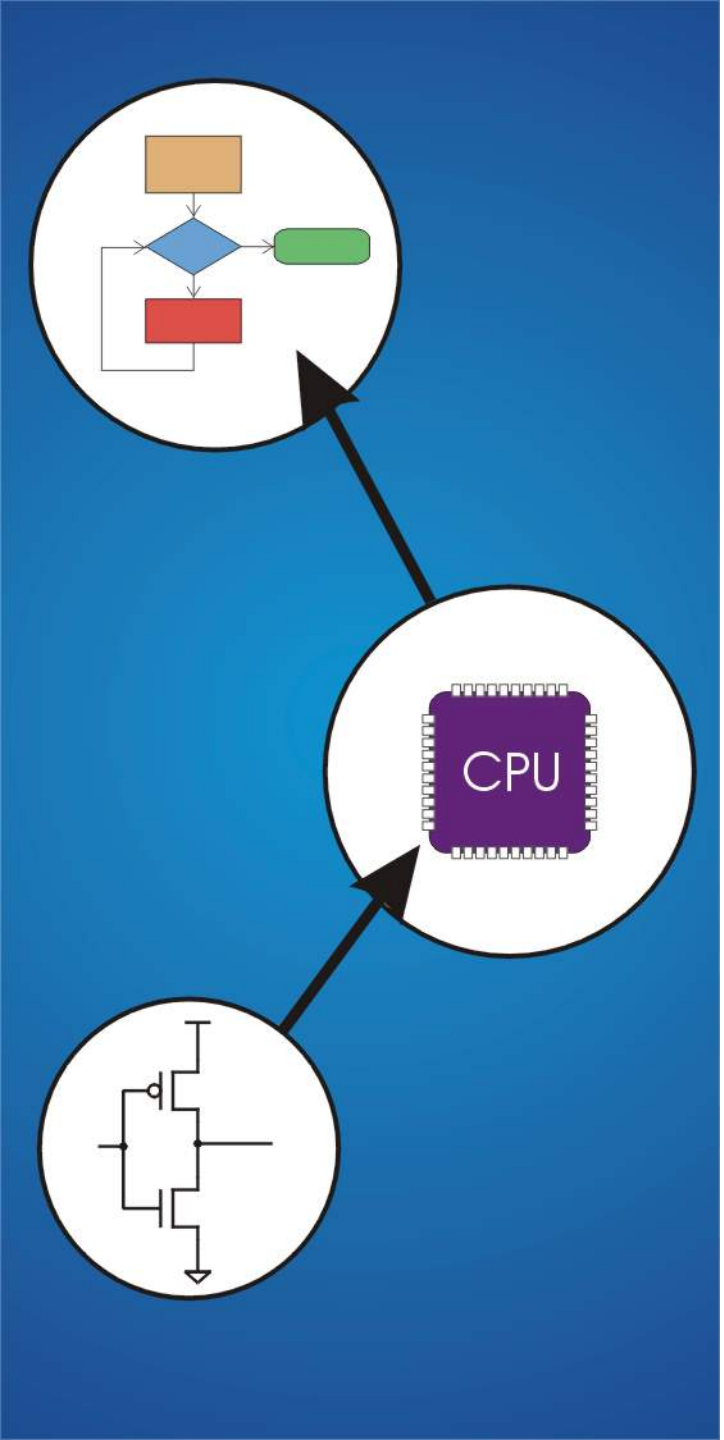
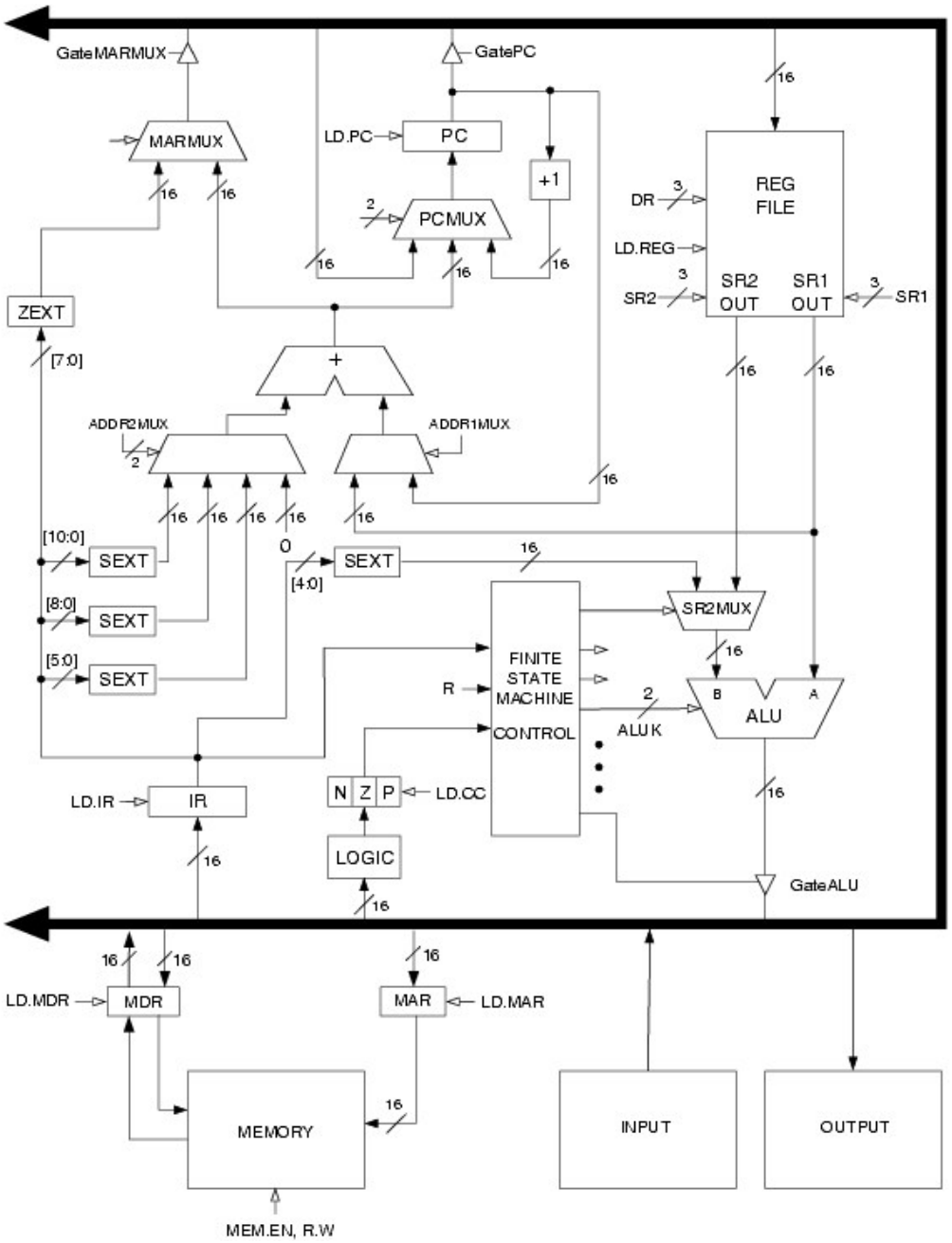


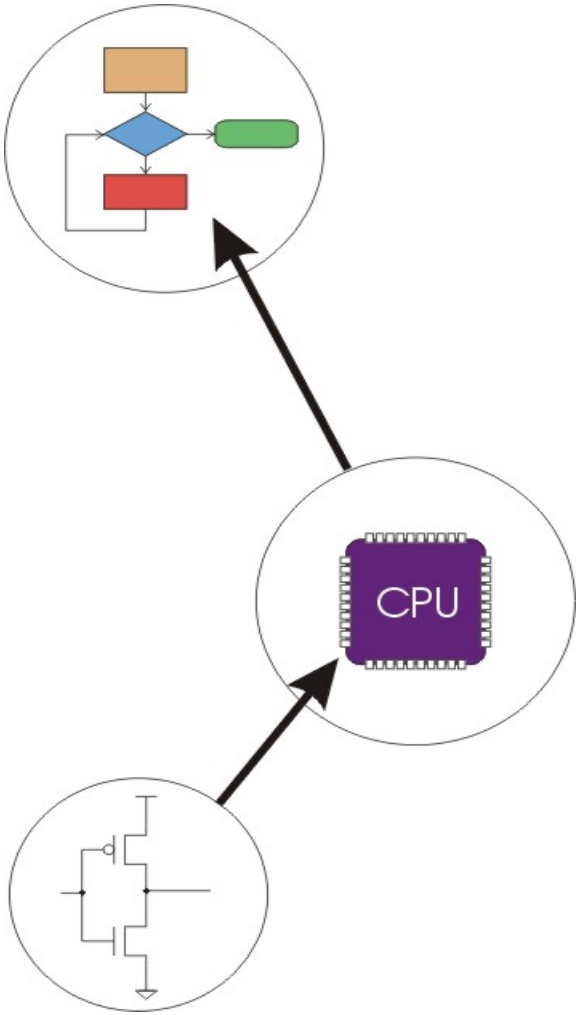
Lecture 22
Chapters 3
Logic Circuits Part 1

LC-3 Data Path Revisited

How are the components Seen here implemented?



Computing Layers



Problems

Algorithms

Language

Instruction Set Architecture

Microarchitecture

Circuits

Devices

Transistor: Building Block of Computers

Logically, each transistor acts as a switch

Combined to implement logic functions (gates)

- **AND, OR, NOT**

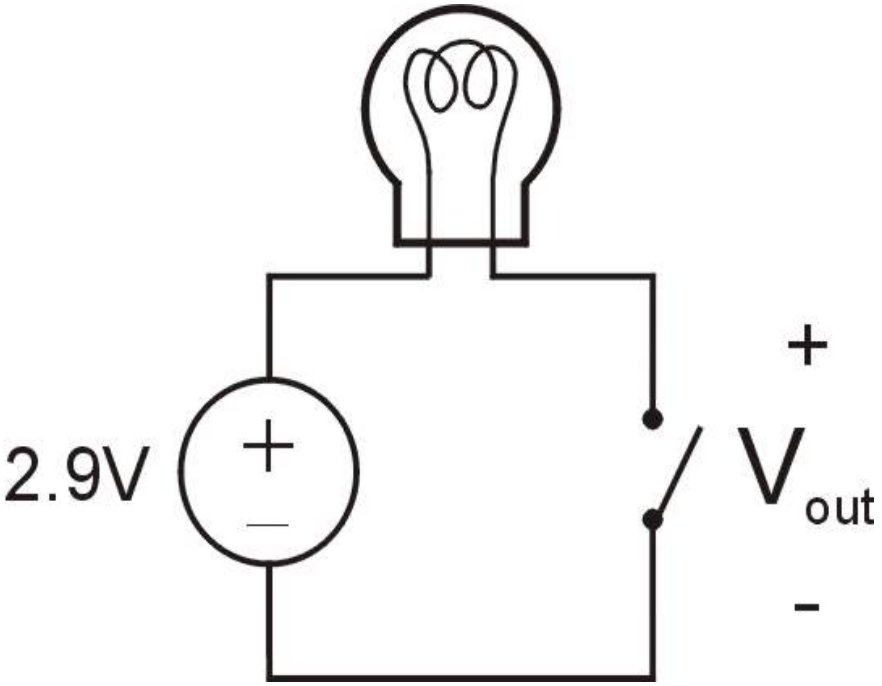
Combined to build higher-level structures

- **Adder, multiplexer, decoder, register, memory …**
- **Adder, multiplier …**

Combined to build simple processor

- **LC-3**

Simple Switch Circuit



Switch open:

- **Open circuit, no current**
- **Light is off**
- **V_{out} is +2.9V**

Switch closed:

- **Short circuit across switch, current flows**
- **Light is on**
- **V_{out} is 0V**

Switch-based circuits can easily represent two states: on/off, open/closed, voltage/no voltage.

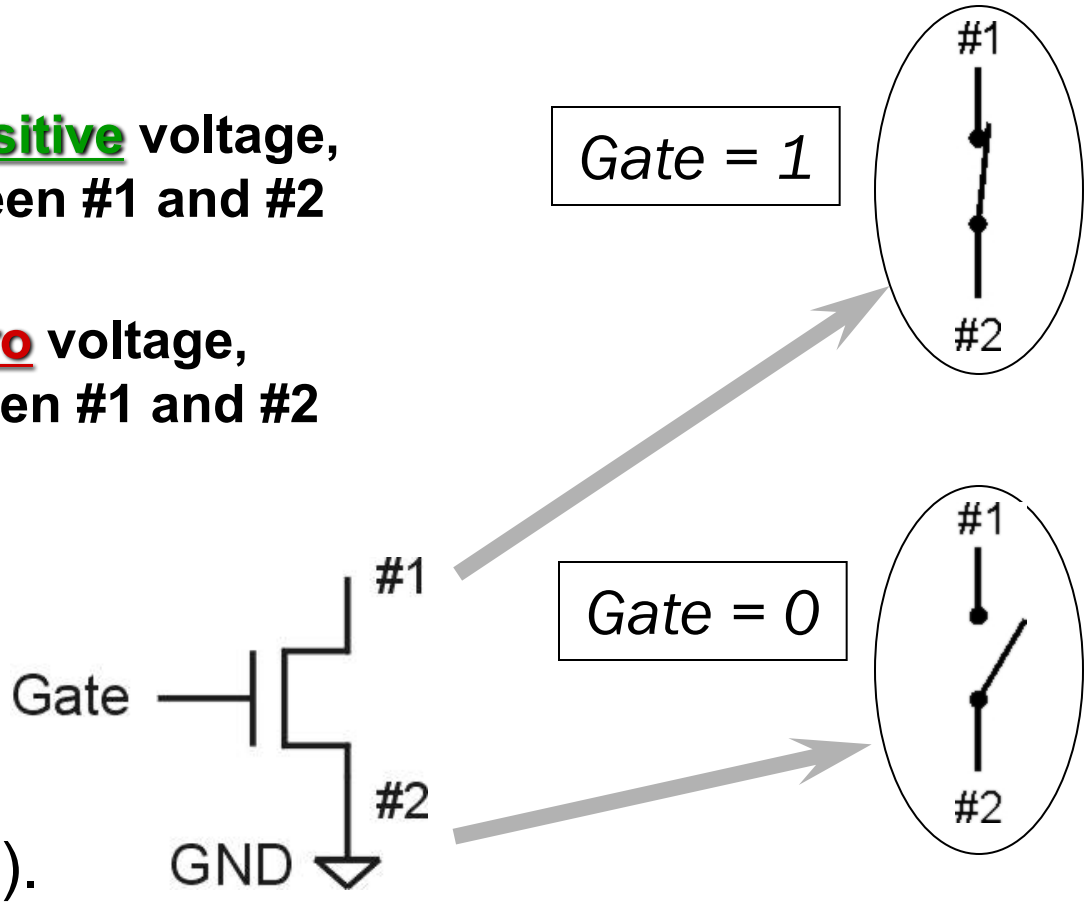
n-type MOS Transistor

MOS = Metal Oxide Semiconductor

- **two types: n-type and p-type**

n-type

- **when Gate has positive voltage, short circuit between #1 and #2 (switch closed)**
- **when Gate has zero voltage, open circuit between #1 and #2 (switch open)**

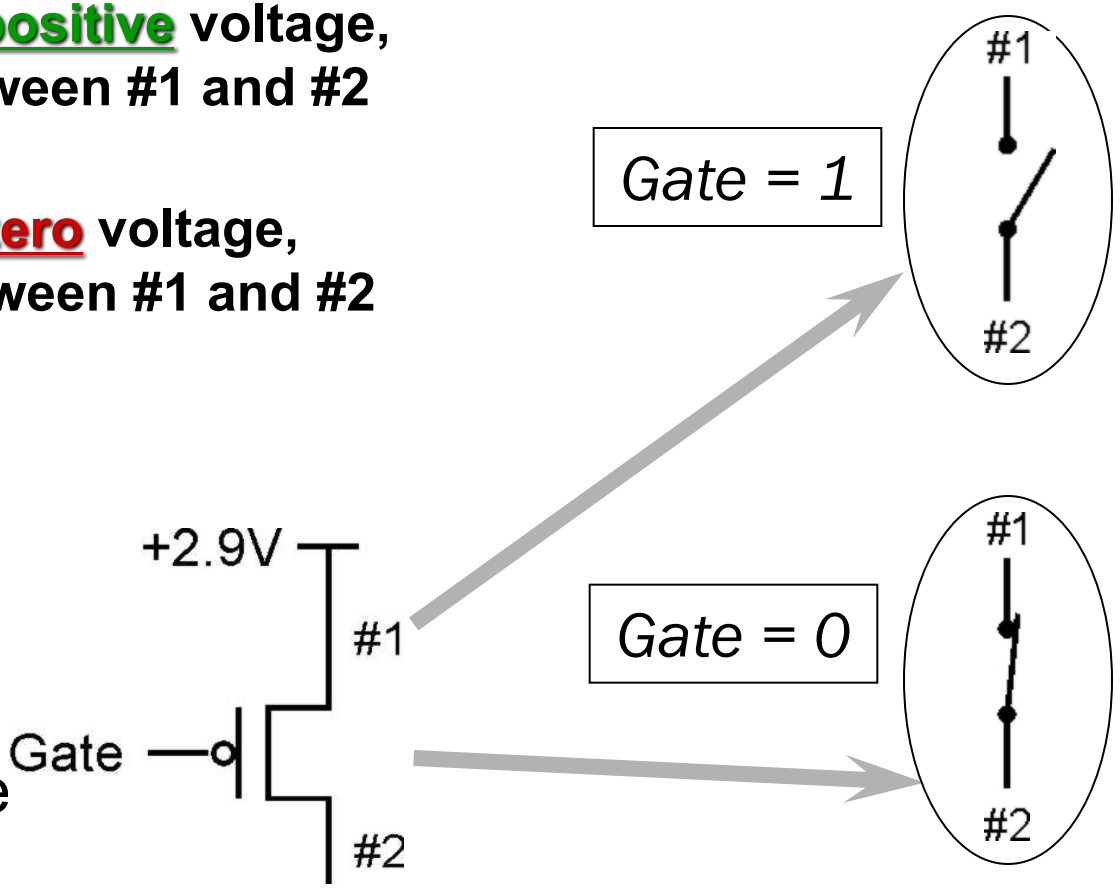


Terminal #2 must be connected to GND (0V).

p-type MOS Transistor

p-type is *complementary* to n-type

- **when Gate has positive voltage, open circuit between #1 and #2 (switch open)**
- **when Gate has zero voltage, short circuit between #1 and #2 (switch closed)**



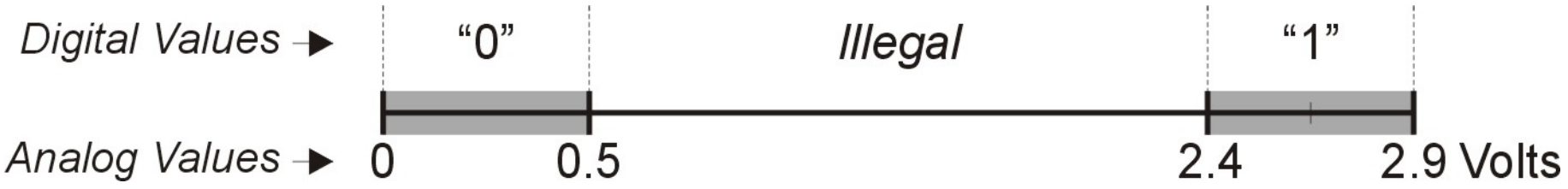
Terminal #1 must be connected to +2.9V.

Logic Gates

Use switch behavior of MOS transistors to implement logical functions: AND, OR, NOT.

Digital symbols:

- **recall that we assign a range of analog voltages to each digital (logic) symbol**



- **assignment of voltage ranges depends on electrical properties of transistors being used**
  - **typical values for "1": +5V, +3.3V, +2.9V**
  - **from now on we'll use +2.9V**

CMOS Circuit

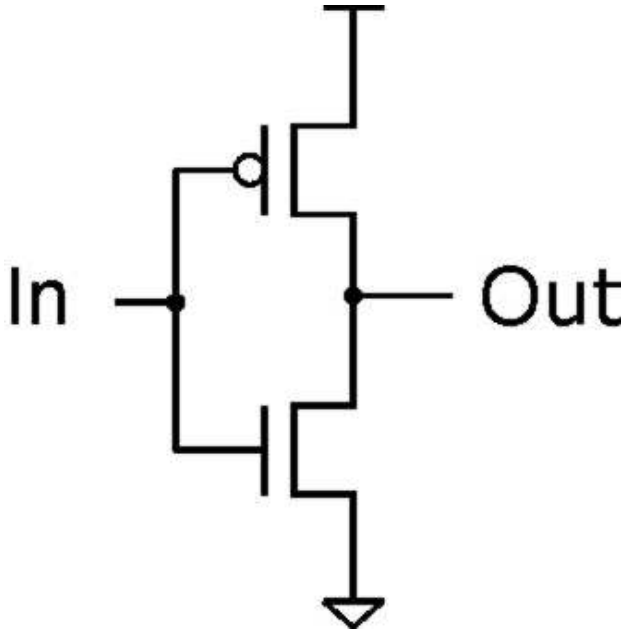
Complementary MOS uses both n-type and p-type MOS transistors

- **p-type**
  - **Attached to + voltage (2.9v)**
  - **Pulls output voltage UP when input is zero**
- **n-type**
  - **Attached to GND (0v)**
  - **Pulls output voltage DOWN when input is one**

For all inputs, output is either connected to GND or to +, but not both!

No direct connection between + and GND, except switching. Low power consumption.

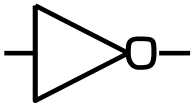
Inverter (NOT Gate)



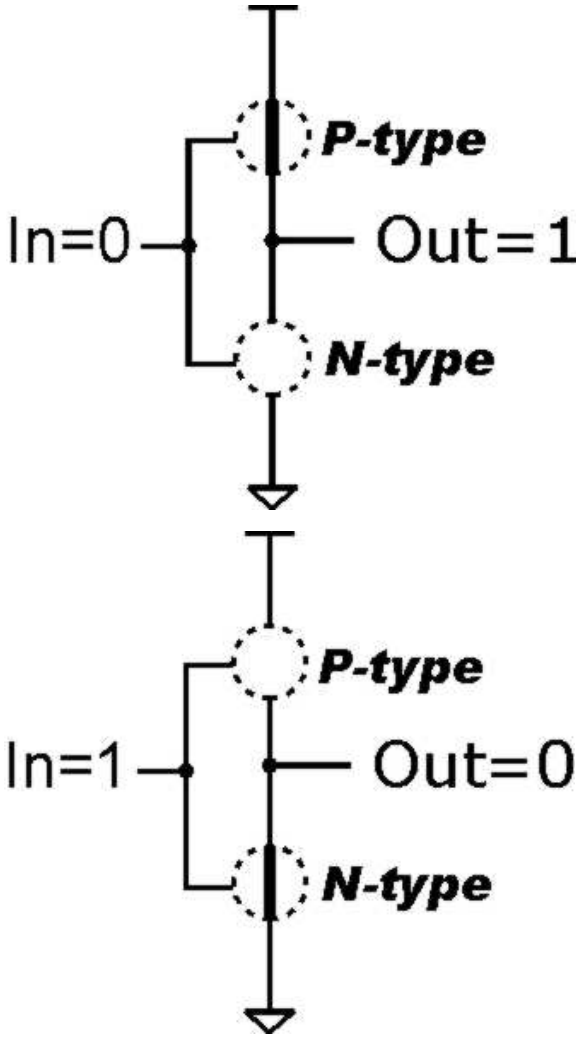
In	Out
0 V	2.9 V
2.9 V	0 V

In	Out
0	1
1	0

Truth table



Symbol



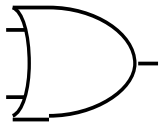
Logical Operation: OR and NOR

Truth tables

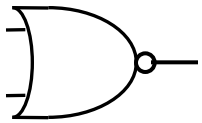
A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Inputs: *2 or more*



Logic symbols



Output=A+B

Boolean algebra notation

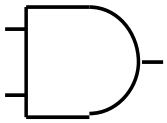
Output=$\overline{A+B}$

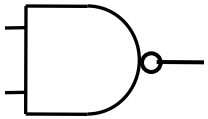
AND and NAND

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

Inputs: 2 or more

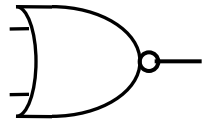




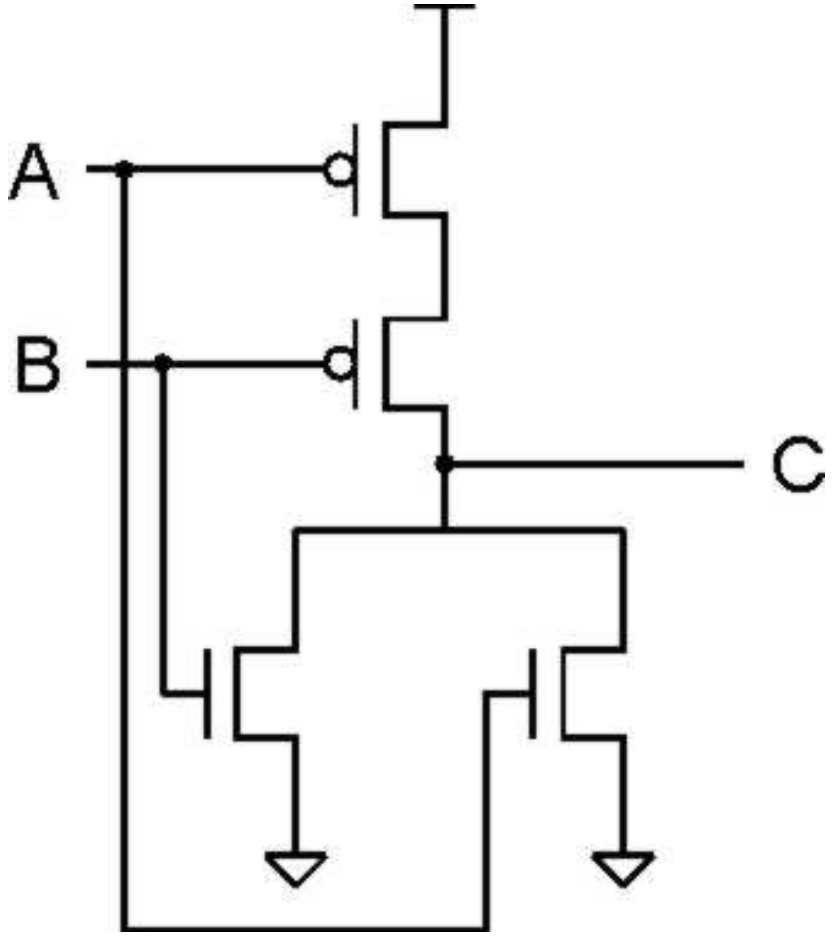
Output = A.B

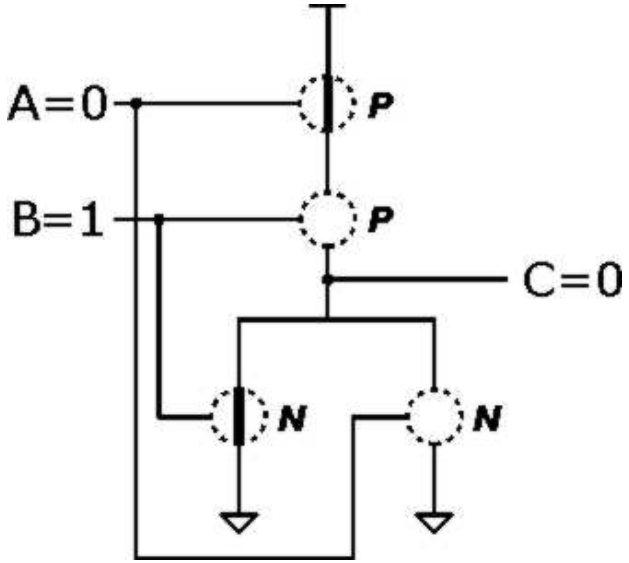
Output = $\overline{A.B}$

NOR Gate (OR-NOT)



Logic symbol



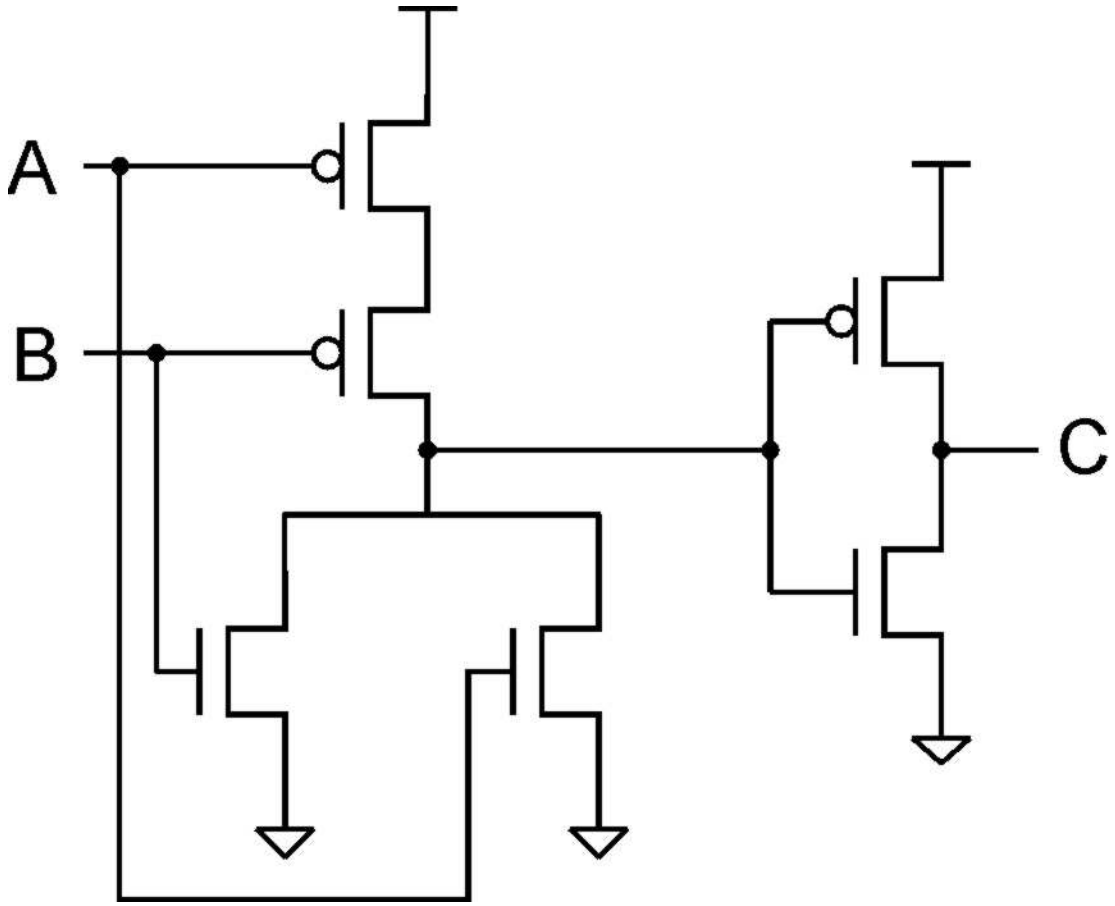


A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Truth table

Note: Serial structure on top, parallel on bottom.

OR Gate

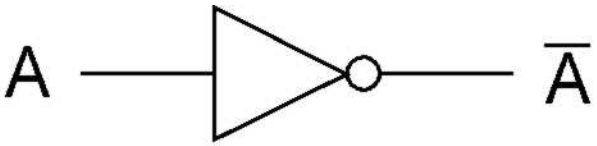


A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

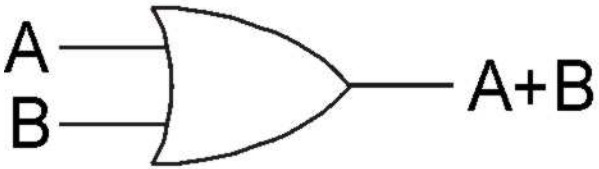
Truth table

Add inverter to NOR.

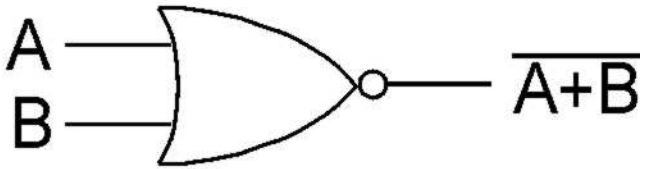
Basic Logic Gates



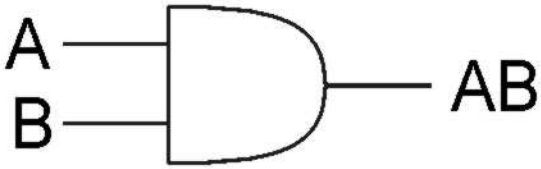
NOT



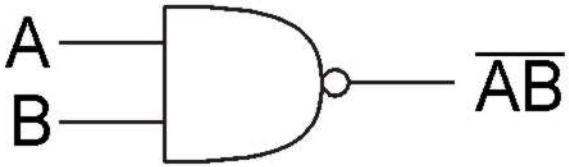
OR



NOR

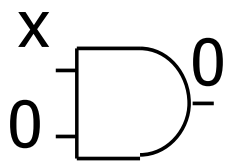


AND

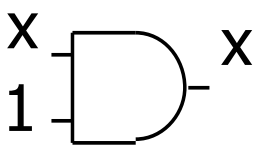


NAND

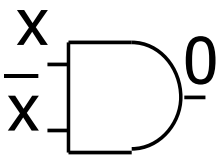
Boolean Algebra



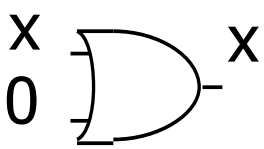
$x.0 = 0$



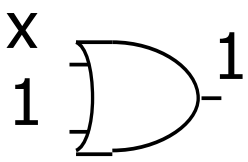
$x.1 = x$



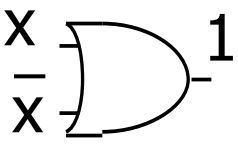
$x.\bar{x} = 0$



$X+0 = x$



$x+1 =$



$x+\bar{x} =$

Boolean Algebra Laws (2)

Commutative $A+B = B+A$ $A.B = B.A$

Associative

- $A+(B+C)=(A+B)+C = A+B+C$
- $A.(B.C)=(A.B).C = ABC$

Distributive

- $A.(B+C)=A.B+A.C$
- $A+(B.C)=(A+B).(A+C)$

Some Useful Identities for simplification

$AB+A\overline{B} = A$

Proof: $AB+A\overline{B} = A(B+\overline{B})$

$= A$

$A+AB = A$

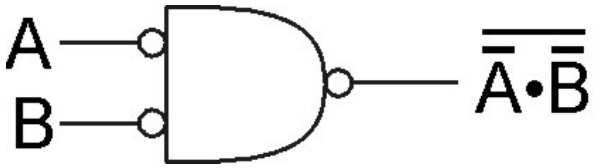
Proof: $A+AB = A(1+B)$

$= A$

DeMorgan's Law

Converting AND to OR (with some help from NOT)

Consider the following gate:



A	B	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$	$\overline{\overline{A} \cdot \overline{B}}$
0	0	1	1	1	**0**
0	1	1	0	0	**1**
1	0	0	1	0	**1**
1	1	0	0	0	**1**

Same as A OR B!

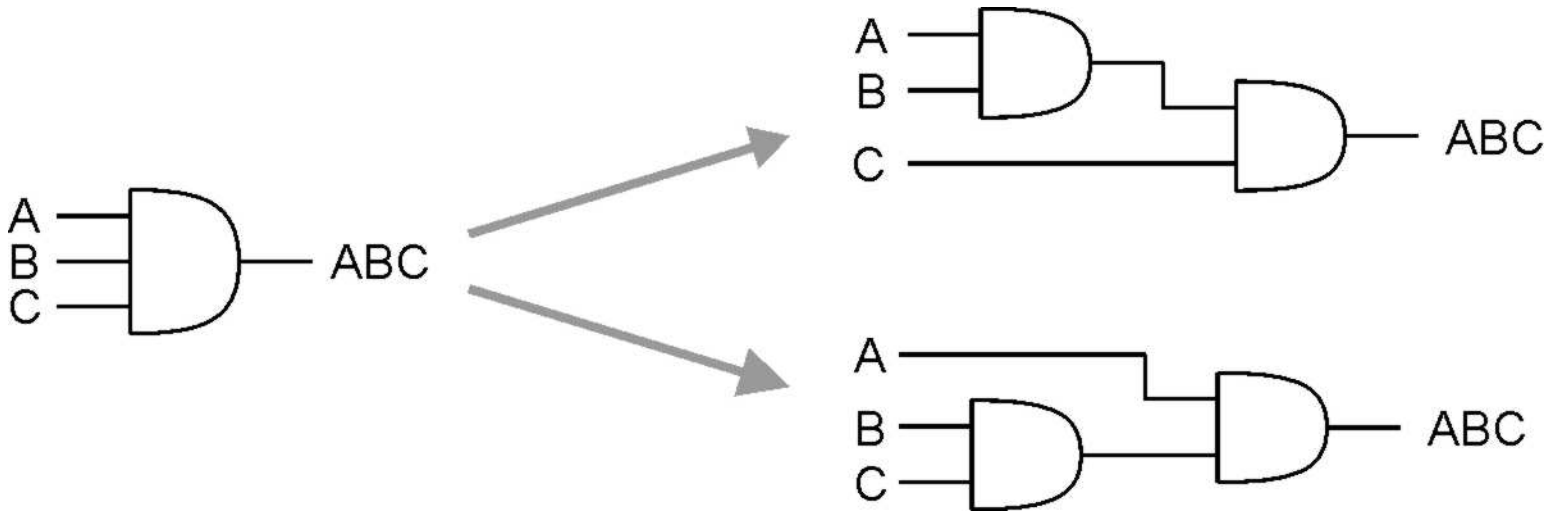
To convert AND to OR (or vice versa), invert inputs and output.

More than 2 Inputs?

AND/OR can take any number of inputs.

- **AND = 1 if all inputs are 1.**
- **OR = 1 if any input is 1.**
- **Similar for NAND/NOR.**

Can implement with multiple two-input gates, or with single CMOS circuit.



Propagation Delay

- **Each gate has a propagation delay, typically fraction of a nanosecond (10^{-9} sec).**
- **Delays add depending on the chain of gates the signals have to go trough.**
- **Clock frequency is determined by the delay of the longest combinational path between storage elements. Measured in GHz (10^{9} cycles per sec).**

Summary

MOS transistors are used as switches to implement logic functions.

- **n-type: connect to GND, turn on (1) to pull down to 0**
- **p-type: connect to +2.9V, turn on (0) to pull up to 1**

Basic gates: NOT, NOR, NAND

- **Boolean Algebra: Logic functions are usually expressed with AND, OR, and NOT**

DeMorgan's Law

- **Convert AND to OR (and vice versa) by inverting inputs and output**

Building Functions from Logic Gates

Combinational Logic Circuit

- **output depends only on the current inputs**
- **stateless**

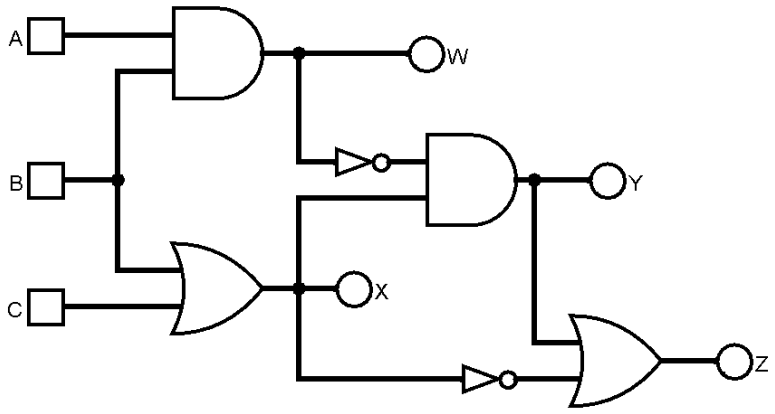
Sequential Logic Circuit

- **output depends on the sequence of inputs (past and present)**
- **stores information (state) from past inputs**

We'll first look at some useful combinational circuits, then show how to use sequential circuits to store information.

Combinatorial Logic

Cascading set of logic gates



Digital circuit

A	B	C	W	X	Y	Z
0	0	0	0	0	0	1
0	0	1	0	1	1	1
0	1	0	0	1	1	1
0	1	1	0	1	1	1
1	0	0	0	0	0	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	1	1	0	0

Truth table

Logisim Simulator

- Logic simulator: allows interactive design and layout of circuits with AND, OR, and NOT gates
- Simulator web page (linked on class web page)

  http://www.cburch.com/logisim
- Overview, tutorial, downloads, etc.
- Windows or Linux operating systems
- Logisim demonstration

Functional Blocks

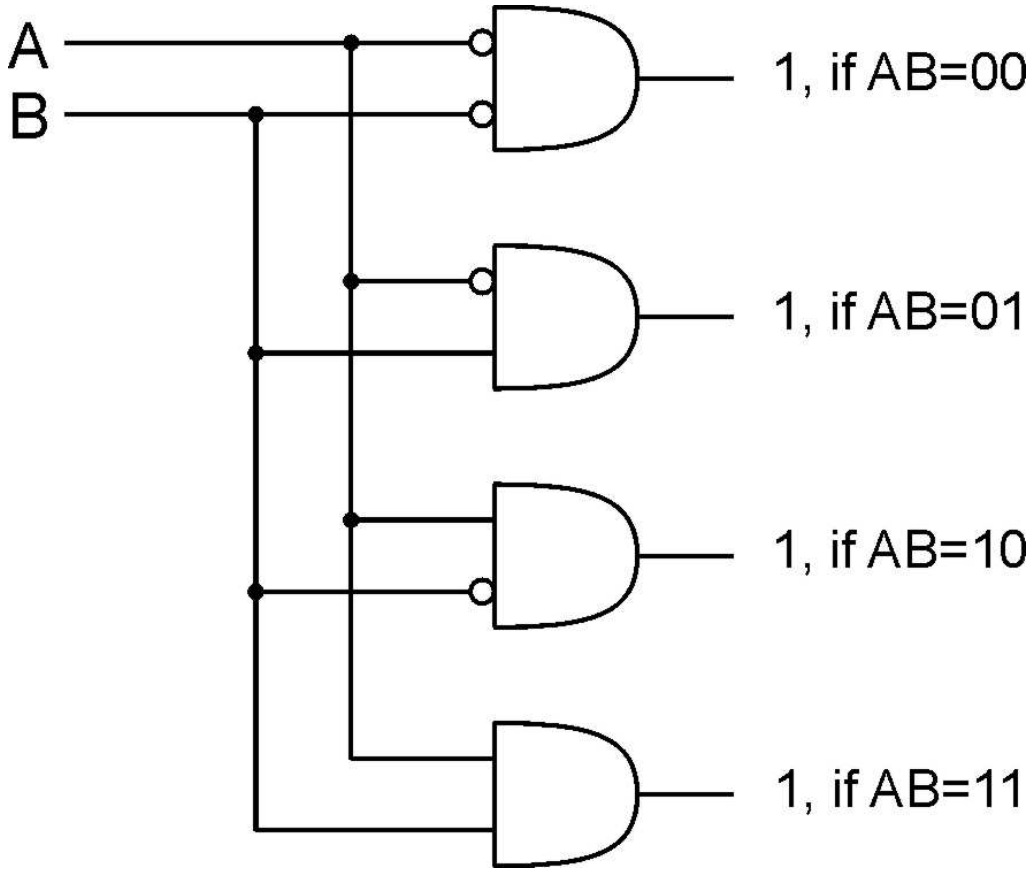
- Decoder
- Multiplexer
- Full Adder
- Any general function

Decoder

n inputs, 2^n outputs

- **exactly one output is 1 for each possible input pattern**

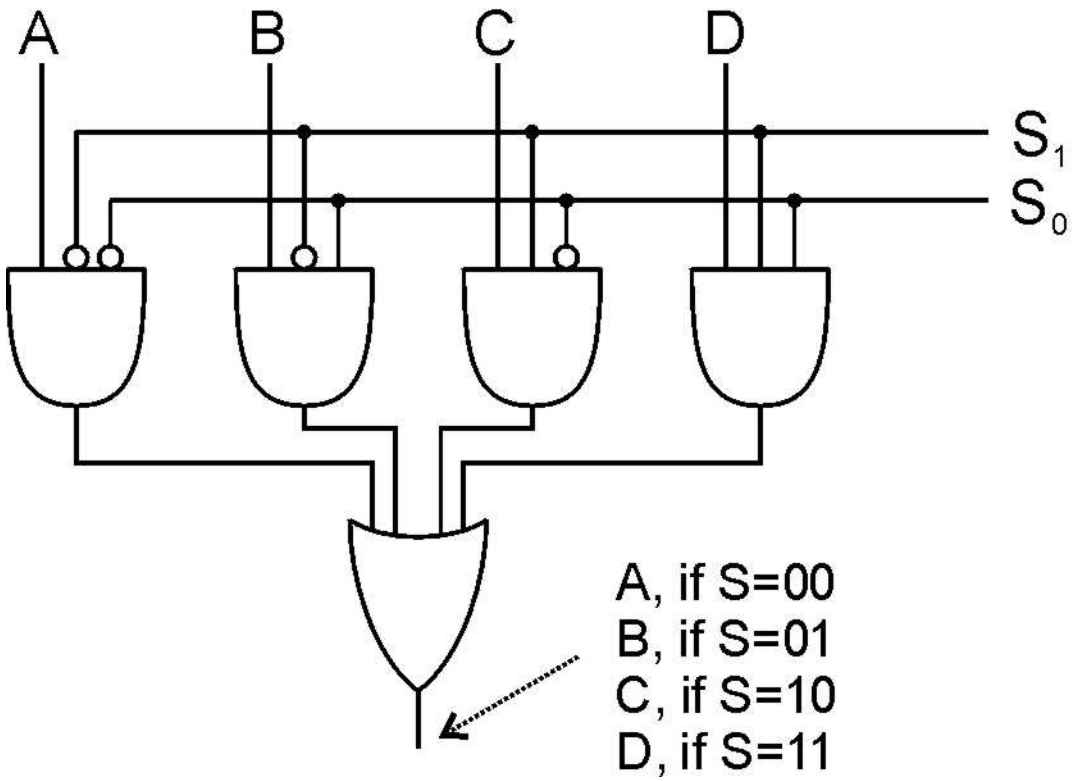
2-bit decoder



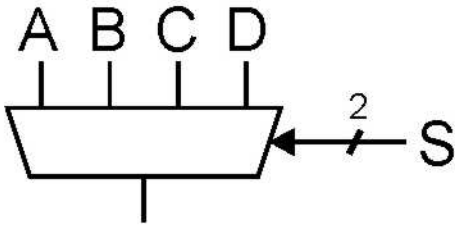
Multiplexer (MUX)

***n*-bit selector and 2^n inputs, one output**

- **output equals one of the inputs, depending on selector**



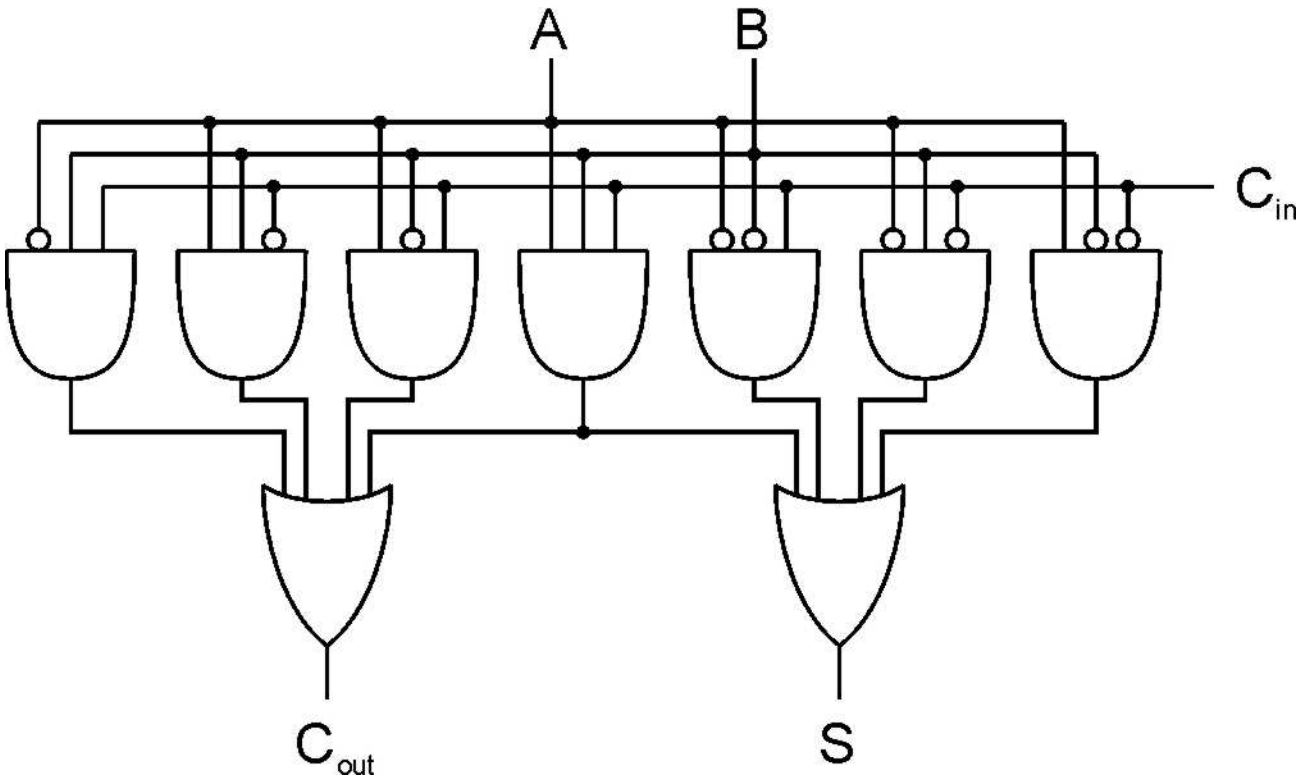
Functional representation



4-to-1 MUX

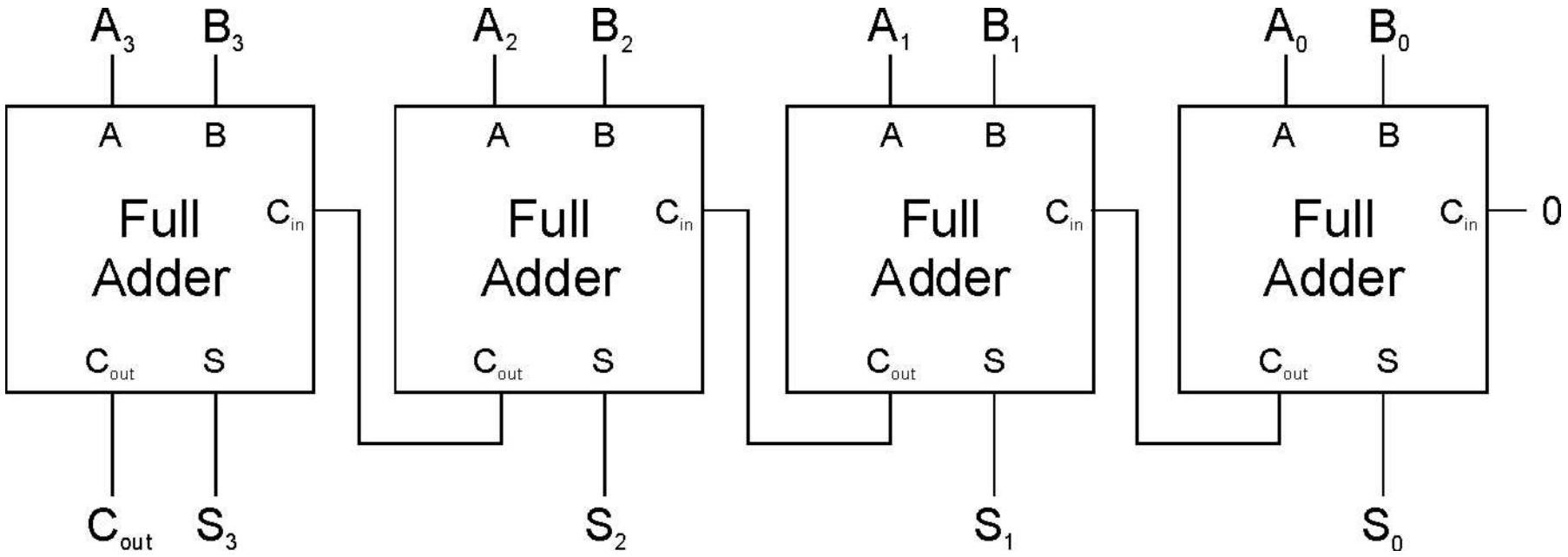
Full Adder

Add two bits and carry-in,
produce one-bit sum and carry-out.



A	B	C_{in}	S	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

Four-bit Adder *(ripple carry)*

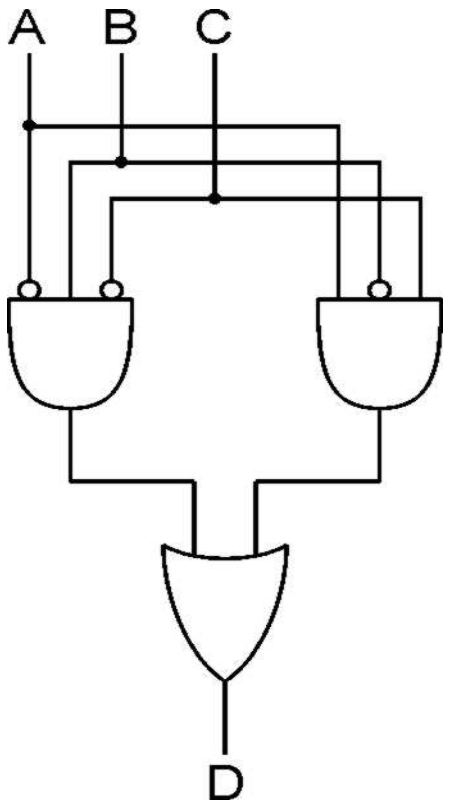


2 levels of delay per stage

Logical Completeness

Can implement <u>ANY</u> truth table with combo of AND, OR, NOT gates.

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



1. AND combinations that yield a "1" in the truth table.

2. OR the results of the AND gates.

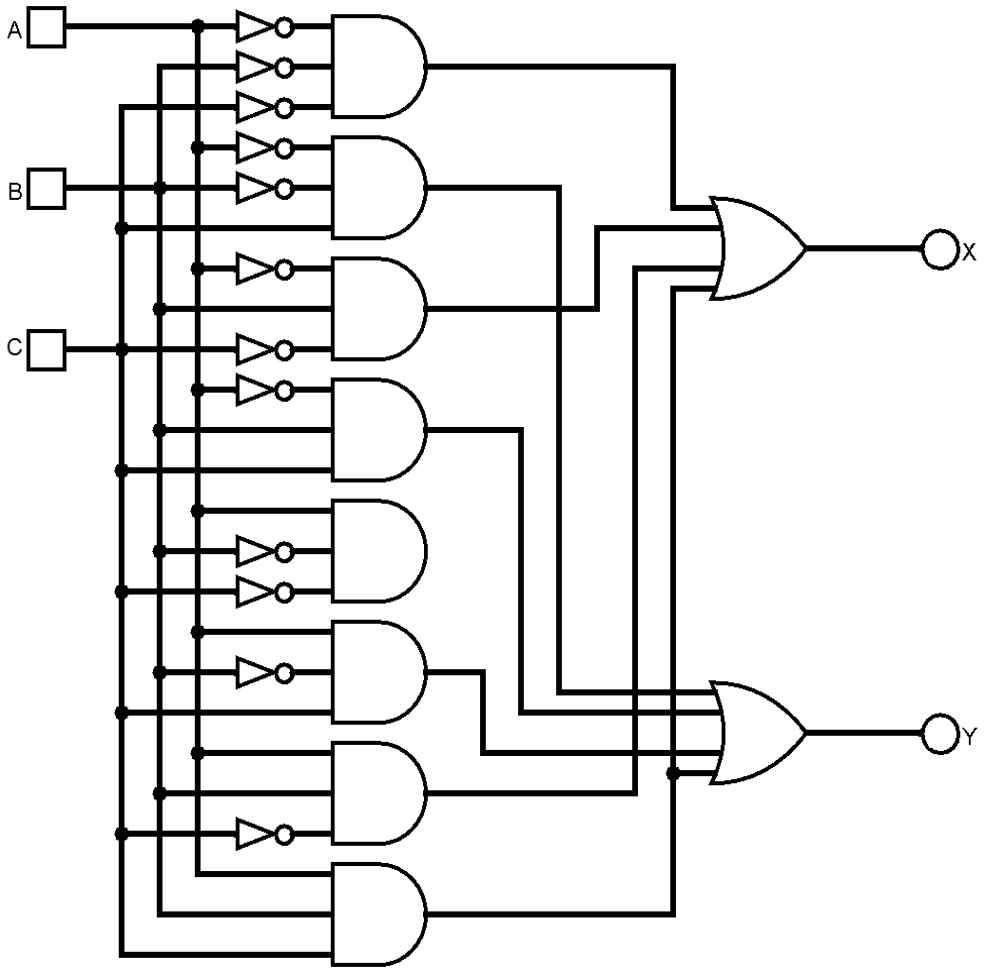
Truth Table (to circuit)

How do we design a circuit for this?

A	B	C	X	Y
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Programmable Logic Array

- **Front end is decoder for inputs**
- **Back end defines the outputs**
- **Any truth table can be built**
- **Not necessarily minimal circuit!**

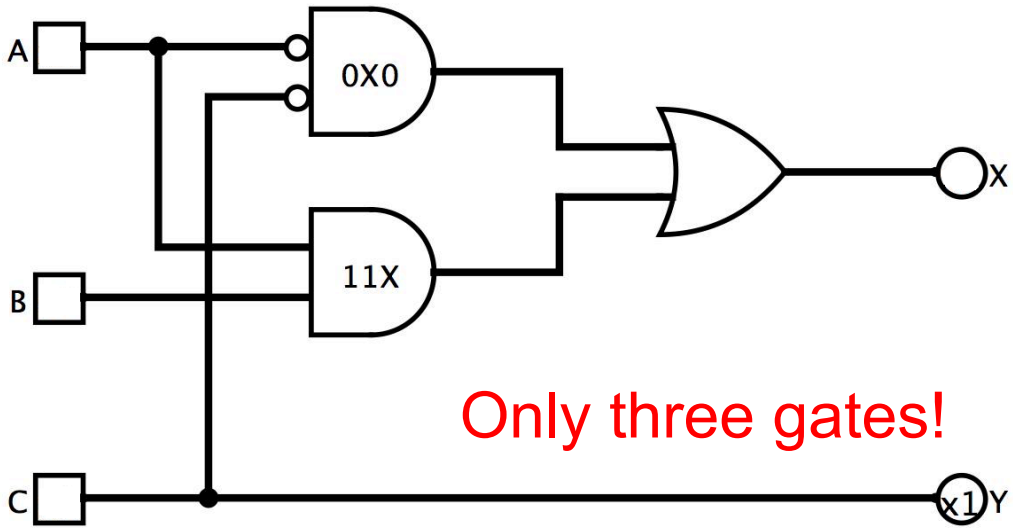


Requires (at least) ten gates.

Circuit Minimization using Boolean Algebra

Boolean logic lets us reduce the circuit

- **X = A'B'C' + A'BC' + ABC' + ABC =**
  = A'C' + AB
- **Y = A′B′C + A'BC + AB'C + ABC**
  = A'C+AC = C



A	B	C	X	Y
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Try with Logisim!

Karnaugh maps to minimize literals

Based on set-theory

- Visual representation of algebraic functions
- Allow algorithmic minimization of boolean functions in sum-of-products form
- "adjacent" terms can be combined.
  - Adjacent: differ in one variable, complemented in one, not complemented in the other.

Example:

- ABC+ABC' = AB(C+C')=AB
- Thus ABC and ABC' are two pieces of AB.

Combining Minterms

- For n-variables, there are 2^n minterms, corresponding to each row of truth table.
- Some of them can be combined into groups of 2, (or 4 or 8 ..) to simplify the function.

Karnaugh maps

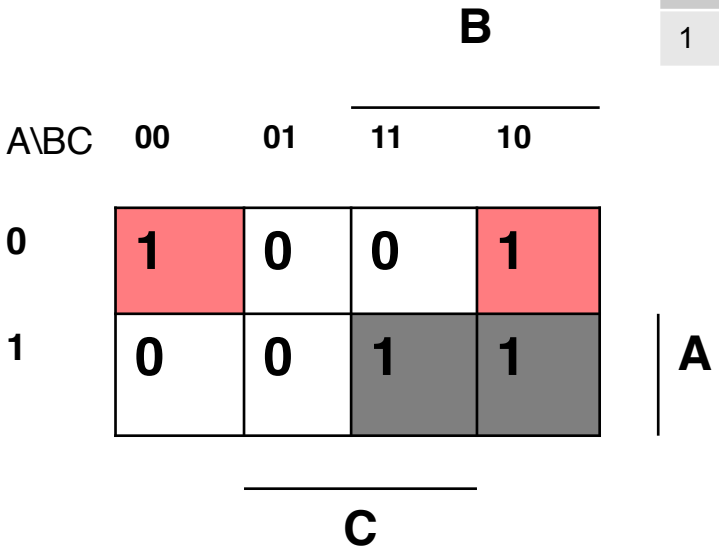
Visual representation of algebraic functions to make it easy to spot "adjacent" minterms"

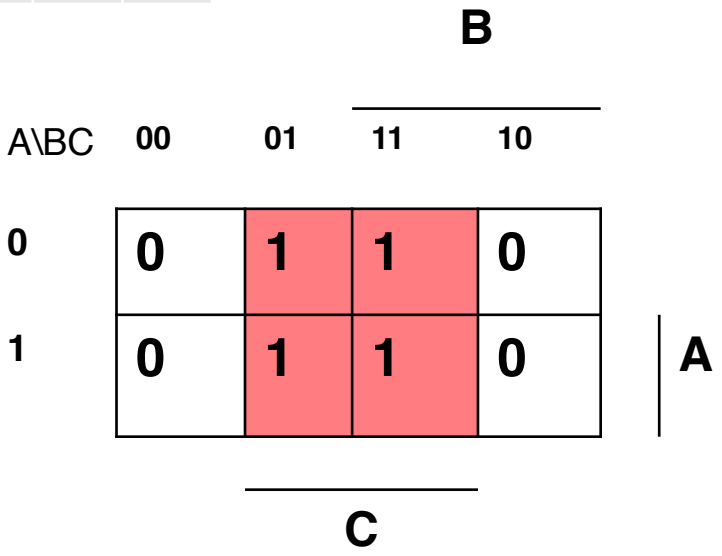
- Columns arranged so that *adjacent terms* are visually adjacent.
- Identify groups of 2, 4, 8 etc. terms that can be combined.
- All 1's must be covered.
- A 1 can be used more than once, if needed.
- Sometimes the solution is not unique
- Next: maps for X(A,B,C) and Y(A,B,C)

A	B	C	X	Y
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

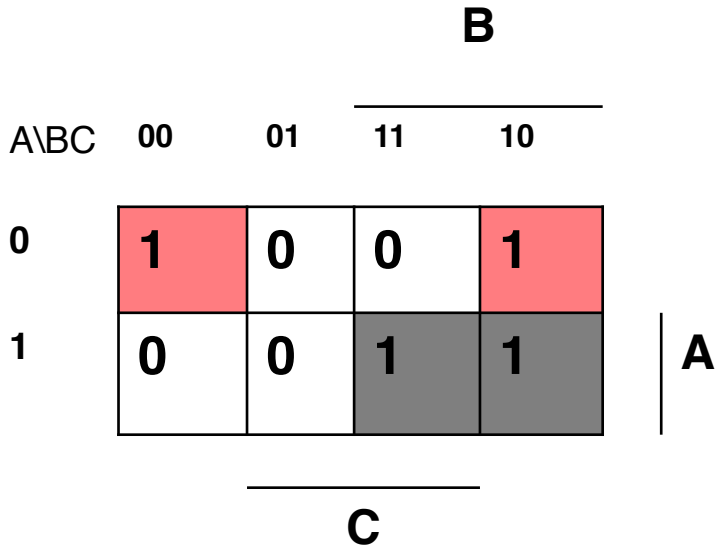
Karnaugh Maps: Visualization of algebra

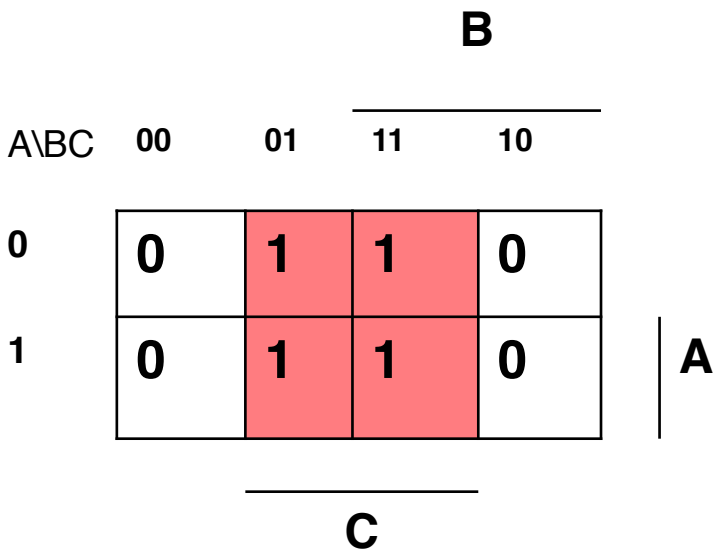
A	B	C	X	Y
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1





Karnaugh Maps: Visualization of algebra



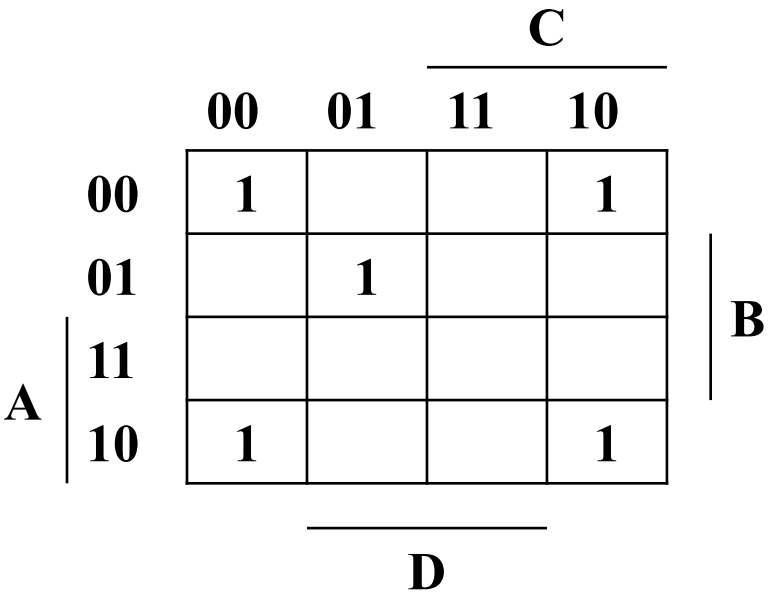


- A'B'C'+A'BC' = A'C'; ABC+ABC' = AB
- A'B'C+A'BC+AB'C+ABC= A'C+AC = C

Thus minimized function is

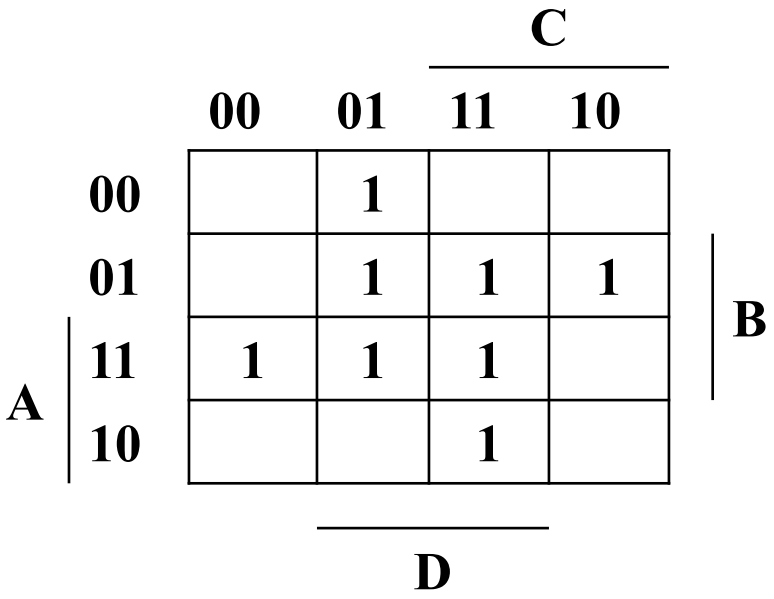
- X = A'C'+AB Y = C

4-variable Kmaps / Design



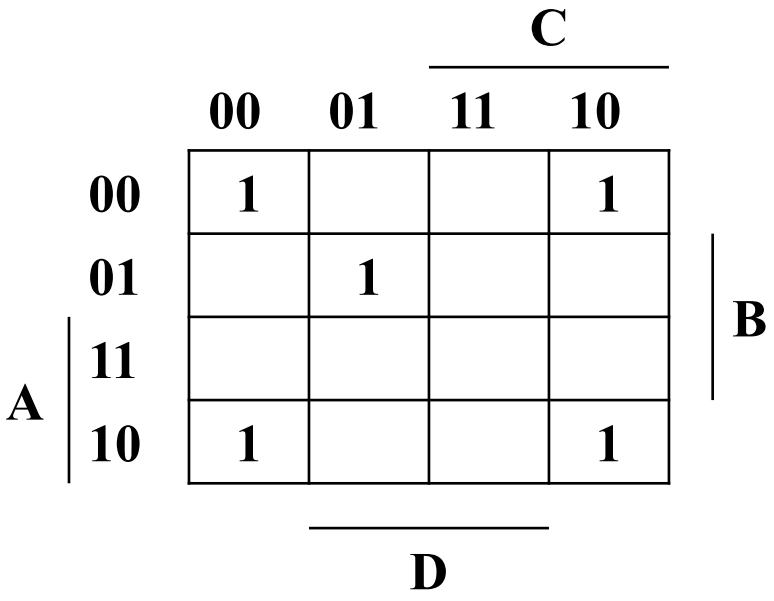
F(A,B,C,D)=B' D' +_____

F(A,B,C,D)=ABC' +A' C' D+
A' BC+ACD+ ?



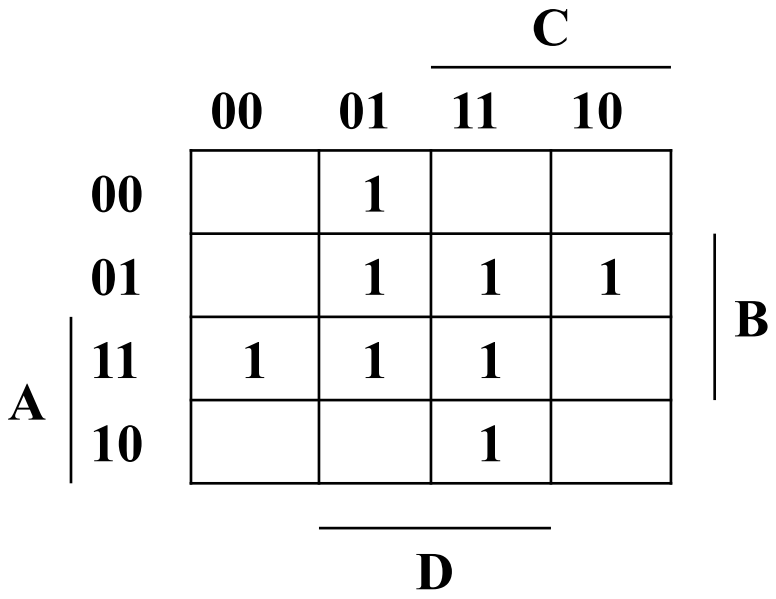
Try them with Logisim

4-variable Kmaps / Design



F(A,B,C,D)=B’ D’ +A’BC’D

F(A,B,C,D)=ABC’ +A’ C’ D+
A’ BC+ACD ~~+ ?~~



Try them with Logisim