COMP 2280 - Introduction to Computer Systems

Module 5- Combinational Circuits

Introduction

- **Microprocessors contain millions of transistors**
  - AMD AM286 (1983): 134,000
  - AMD AM486 (1993): 1,185,000
  - Intel Pentium 4 (2000): 48 million
  - Intel core i7 quad (2008): 731 million
  - Intel i7 - 6 cores (2010): 2.27 billion
  - **AMD Ryzen 9 – 16 cores (2020): 19.2 billion**
- Logically, each transistor acts as a switch
  - Combined to implement logic functions AND, OR, NOT
  - Combined to build higher-level structures such as adders, multiplexers, decoders, registers, ...
  - Combined to build processor such as the LC-3

Introduction

- In digital design, we are concerned with designing and building circuits that work with binary data.
- We will start off by looking a very simple circuits and then building on top of them to get more complicated circuits.
- This will be a very brief introduction to design logic.
- Let’s start off with the most basic of components that we need to build digital circuits: AND, OR, NOT,NOR,NAND.

Logic Gates

- **Perform logic functions:**
  - inversion (NOT), AND, OR, NAND, NOR, etc.
- **Single-input:**
  - NOT gate, buffer
- **Two-input:**
  - AND, OR, XOR, NAND, NOR, XNOR
- **Multiple-input**

Reference also found on Learn:

Boolean Algebra Basics

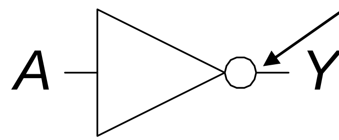
Two constants: 0 and 1

- 0 is equivalent to false
- 1 is equivalent to true

Operators		
Symbol	Description	Example
•	AND	a•b
+	OR	a+b
$\bar{}$	NOT	$\bar{a}$
'	NOT	a'
⊕	XOR (exclusive or)	a⊕b

Single-Input Logic Gates

NOT



Note: The circle is important, we'll see it again with the negated/"NOT-versions" of the other gates

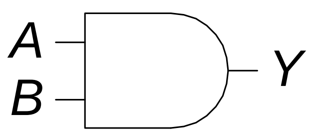
$$Y = \overline{A}$$

A	Y
0	1
1	0

Note: a line over an expression denotes negation

Two-Input Logic Gates

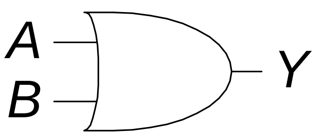
AND



$Y = AB$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

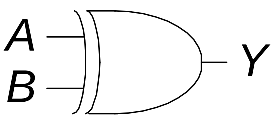


$Y = A + B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

More Two-Input Logic Gates

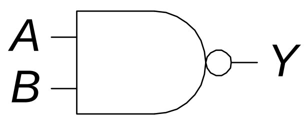
XOR



$$Y = A \oplus B$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

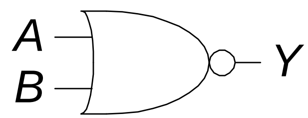
NAND



$$Y = \overline{AB}$$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

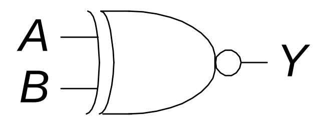
NOR



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

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

XNOR



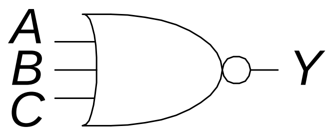
$$Y = \overline{A \oplus B}$$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Note the circles on the gates, and the lines over the expression

Multiple-Input Logic Gates

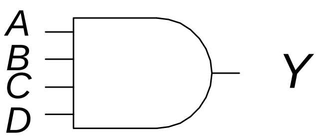
NOR3



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

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

AND4



$Y = ABCD$

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

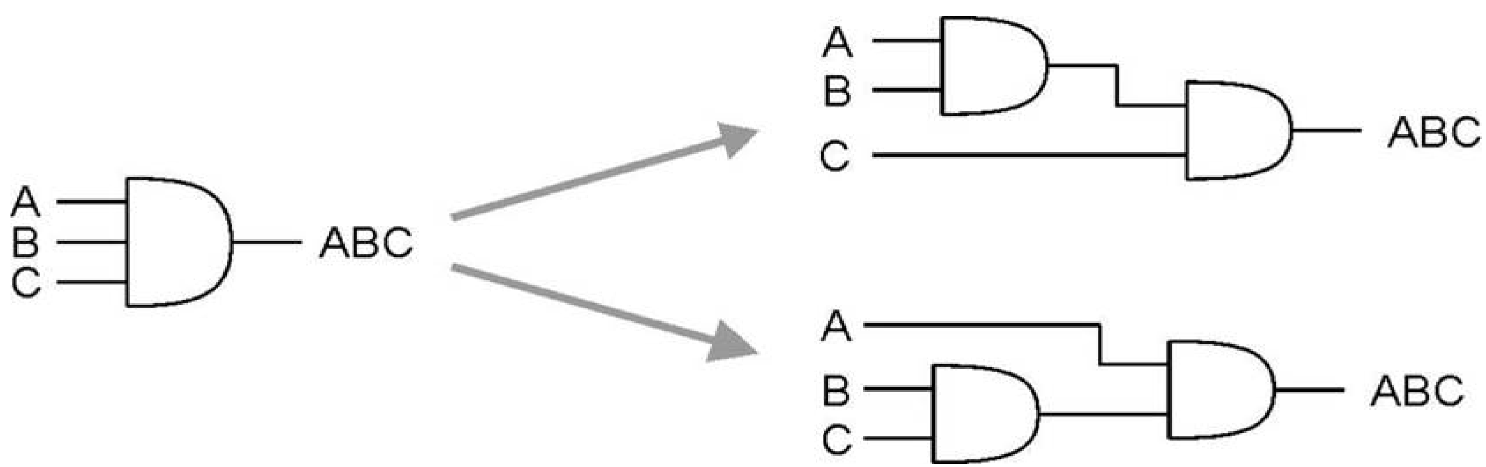
The Column for 'D' is missing but the output 'Y' is still only going to be true when all inputs are true

Note: Multi-input gates are equivalent to multiple two-input gates in series

Multiple-Input Logic Gates

Note: Multi-input gates are equivalent to multiple two-input gates in series

We draw a 3-input AND gate by combining two 2-input AND gates.



Classes of Digital Circuits

- Digital circuits can be classified as either
  - Combinational
  - Sequential
- Combinational circuits are those whose output depends only on the current inputs.
- Sequential circuits are those whose output depends on current inputs and its current state.

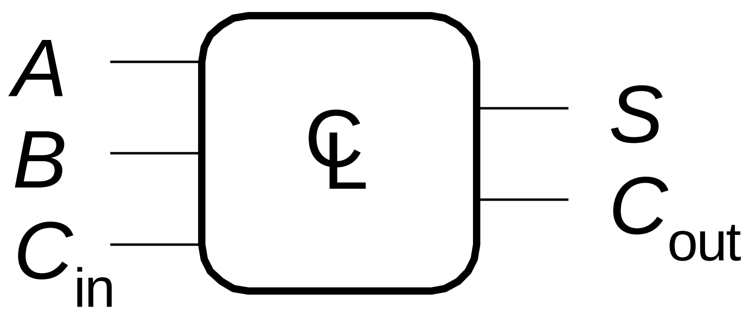
Combinational Circuits

- output depends on the current inputs only
- Stateless
- outputs are defined by Boolean functions of the inputs
- any combinational circuit can be implemented as a sum-of-products (SOP) or product-of-sums (POS)
- always gives 2 level circuits which are very fast.

Boolean Equations

- Functional specification of outputs in terms of inputs
- **Example:** $S = F(A, B, C_{in})$

  $C_{out} = F(A, B, C_{in})$



This may look complex, but you have already seen and used this Boolean equation.

Any ideas on what it is?

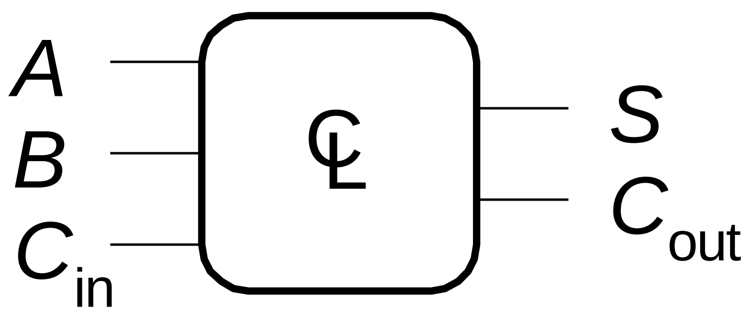
$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + AC_{in} + BC_{in}$$

Boolean Equations

- Functional specification of outputs in terms of inputs
- **Example:** $S = F(A, B, C_{in})$

  $C_{out} = F(A, B, C_{in})$



It's like a binary addition with the Carry-in and Carry-out bits!

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Truth Tables

- Truth tables specify the output of a circuit based on its inputs.
- Given a truth table, there are standard techniques for building a circuit that realizes the truth table.
- Two techniques
  - sum-of-products (SOP) technique
  - product-of-Sums (POS) Form
- Let’s do a simple example.

Sum-of-Products (SOP) Form

- All equations can be written in SOP form
- Each row has a **minterm**
- A minterm is a product (AND) of all input variables
- Each minterm is TRUE for that row (and only that row)
- Form function by ORing minterms where the output is TRUE
- Thus, a sum (OR) of products (AND terms)

A	B	Y	minterm	minterm name
0	0	0	$\overline{A}\ \overline{B}$	m_0
0	1	1	$\overline{A}\ B$	m_1
1	0	0	$A\ \overline{B}$	m_2
1	1	1	$A\ B$	m_3

$$Y = F(A, B) = \overline{A}B + AB$$

- Aka: "List all possible ways for it to be True"

How to get a Minterm?
"With the current input on this row, how can I build an AND expression with NOTs that gives me TRUE?"

Product-of-Sums (POS) Form

- All Boolean equations can be written in POS form
- Each row has a **maxterm**
- A maxterm is a sum (OR) of all input variable
- Each maxterm is FALSE for that row (and only that row)
- Form function by ANDing the maxterms for which the output is FALSE
- Thus, a product (AND) of sums (OR terms)

A	B	Y	maxterm	maxterm name
0	0	0	$A + B$	M_0
0	1	1	$A + \overline{B}$	M_1
1	0	0	$\overline{A} + B$	M_2
1	1	1	$\overline{A} + \overline{B}$	M_3

$$Y = F(A, B) = (A + B)(\overline{A} + B)$$

- Aka: “List (eliminate?) all possible ways for it to be False”

Boolean Equations Example

- You are going to the cafeteria for lunch
  - You will only eat lunch (E):
    - If it's open (O) and
    - They **don't** serve corndogs (C)
  - Write a truth table for determining if you will eat lunch (E).

O	C	E
0	0	0
0	1	0
1	0	1
1	1	0

SOP & POS Form

- SOP – sum-of-products

O	C	E	minterm
0	0	0	$\overline{O}\ \overline{C}$
0	1	0	$\overline{O}\ C$
1	0	1	$O\ \overline{C}$
1	1	0	$O\ C$

$$Y = O\overline{C}$$

Aka: "List all possible ways for it to be True"

- POS – product-of-sums

O	C	E	maxterm
0	0	0	$O + C$
0	1	0	$O + \overline{C}$
1	0	1	$\overline{O} + C$
1	1	0	$\overline{O} + \overline{C}$

$$Y = (O + C)(O + \overline{C})(\overline{O} + \overline{C})$$

Aka: "List (eliminate?) all possible ways for it to be False"

Boolean Algebra

- Axioms and theorems to **simplify** Boolean equations
- Like regular algebra, but simpler: variables have only two values (1 or 0)
- **Duality** in axioms and theorems:
  - ANDs and ORs, 0's and 1's interchanged

Boolean Axioms

	Axiom		Dual	Name
A1	$B = 0$ if $B \neq 1$	A1′	$B = 1$ if $B \neq 0$	Binary field
A2	$\overline{0} = 1$	A2′	$\overline{1} = 0$	NOT
A3	$0 \bullet 0 = 0$	A3′	$1 + 1 = 1$	AND/OR
A4	$1 \bullet 1 = 1$	A4′	$0 + 0 = 0$	AND/OR
A5	$0 \bullet 1 = 1 \bullet 0 = 0$	A5′	$1 + 0 = 0 + 1 = 1$	AND/OR

	Theorem		Dual	Name
T1	$B \bullet 1 = B$	T1′	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2′	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3′	$B + B = B$	Idempotency
T4		$\overline{\overline{B}} = B$		Involution
T5	$B \bullet \overline{B} = 0$	T5′	$B + \overline{B} = 1$	Complements

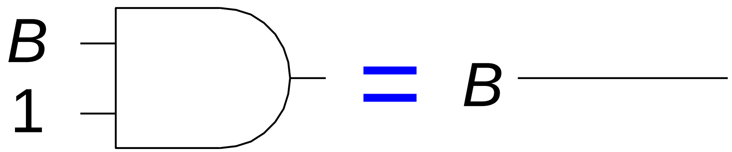
End of Part A

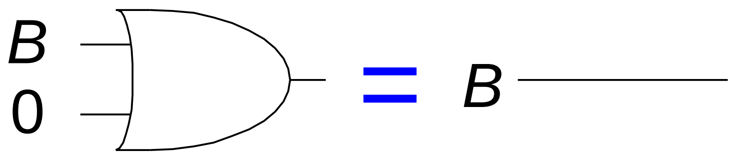
T1: Identity Theorem

- $B \cdot 1 = B$
- $B + 0 = B$

B and 1 gives us B

B or 0 gives us B as well



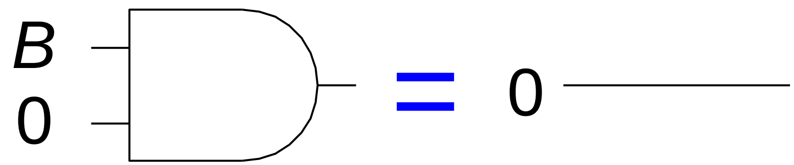


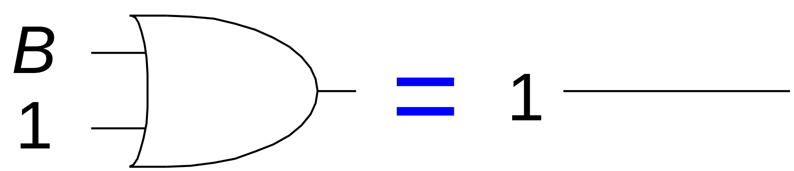
T2: Null Element Theorem

- $B \cdot 0 = 0$
- $B + 1 = 1$

B and 0 gives us 0

B or 1 gives us 1



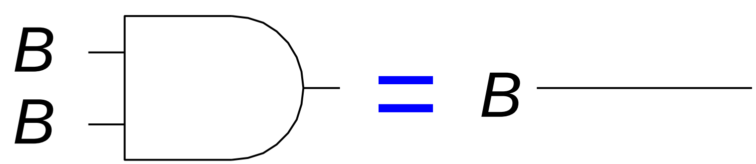


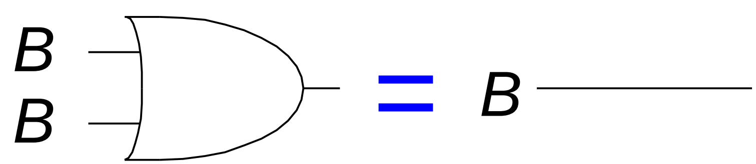
T3: Idempotency Theorem

- $B \cdot B = B$
- $B + B = B$

B and B gives us B

B or B gives us B



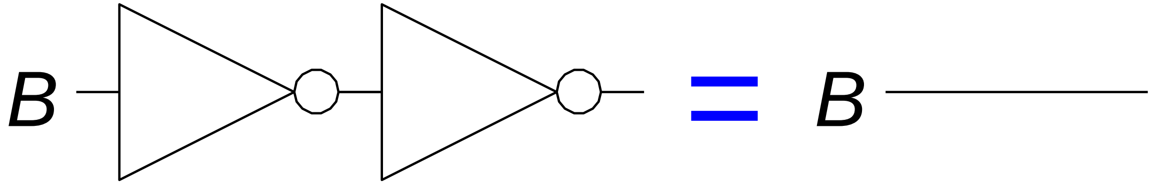


T4: Involution Theorem

- $\overline{\overline{B}} = B$

The opposite of the opposite of B is… B

Not(Not(B)) = B

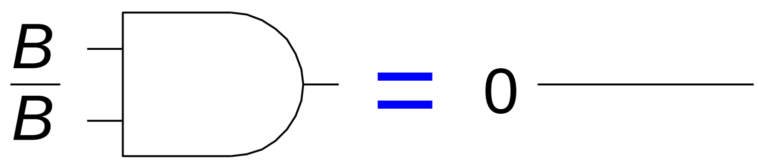


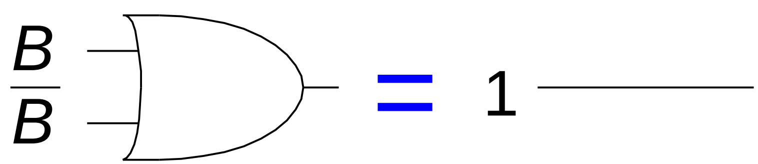
T5: Complement Theorem

- $B \cdot \overline{B} = 0$
- $B + \overline{B} = 1$

B and NOT B is 0

B or NOT(B) is 1





Boolean Theorems Summary

	Theorem		Dual	Name
T1	$B \bullet 1 = B$	T1′	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2′	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3′	$B + B = B$	Idempotency
T4		$\overline{\overline{B}} = B$		Involution
T5	$B \bullet \overline{B} = 0$	T5′	$B + \overline{B} = 1$	Complements

Boolean Theorems of Several Vars

	Theorem		Dual	Name
T6	$B \bullet C = C \bullet B$	T6′	$B + C = C + B$	Commutativity
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	T7′	$(B + C) + D = B + (C + D)$	Associativity
T8	$(B \bullet C) + B \bullet D = B \bullet (C + D)$	T8′	$(B + C) \bullet (B + D) = B + (C \bullet D)$	Distributivity
T9	$B \bullet (B + C) = B$	T9′	$B + (B \bullet C) = B$	Covering
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	T10′	$(B + C) \bullet (B + \overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D)$ $= B \bullet C + \overline{B} \bullet D$	T11′	$(B + C) \bullet (\overline{B} + D) \bullet (C + D)$ $= (B + C) \bullet (\overline{B} + D)$	Consensus
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots}$ $= (\overline{B_0} + \overline{B_1} + \overline{B_2} \ldots)$	T12′	$\overline{B_0 + B_1 + B_2 \ldots}$ $= (\overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2})$	De Morgan's Theorem

Think Math Algebra

Simplifying Boolean Equations

Example 1:

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

$= B(A + \overline{A})$ *T8, Distributivity*

We "factor out" the common term B

$= B(1)$ *T5', Complements*

A or Not(A), that is the question…

$= B$ *T1, Identity*

*B and 1, same as B*1*

Simplifying Boolean Equations

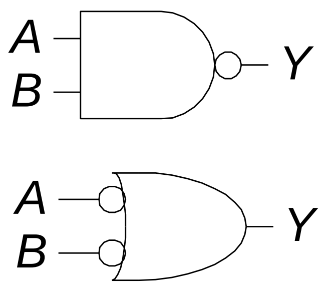
Example 2:

- $Y = A(AB + ABC)$

$= A(AB(1 + C))$	T8, common term AB
$= A(AB(1))$	T2', C or 1, always 1
$= A(AB)$	T1, AB and 1, easy
$= (AA)B$	T7, A*(A*B)=(A*A)*B
$= AB$	T3, A and A, redundant

DeMorgan's Theorem

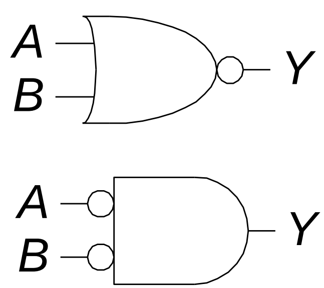
- $Y = \overline{AB} = \overline{A} + \overline{B}$



I go camping when there are:
NOT (rain and thunderstorms)
NOT(rain) or NOT(thunderstorms)

A	B	Not(AB)	Not(A) + Not(B)
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

- $Y = \overline{A + B} = \overline{A} \bullet \overline{B}$

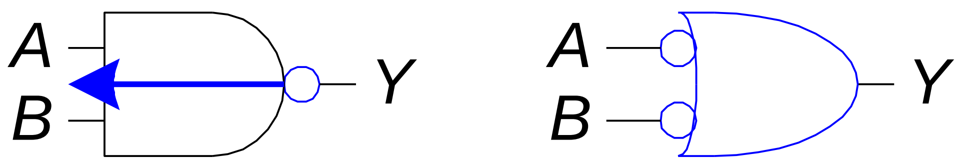


I go hiking when there are:
NOT(Wolves or Bears)
NOT(Wolves) and NOT(bears)

A	B	Not(A+B)	Not(A) * Not(B)
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Bubble Pushing

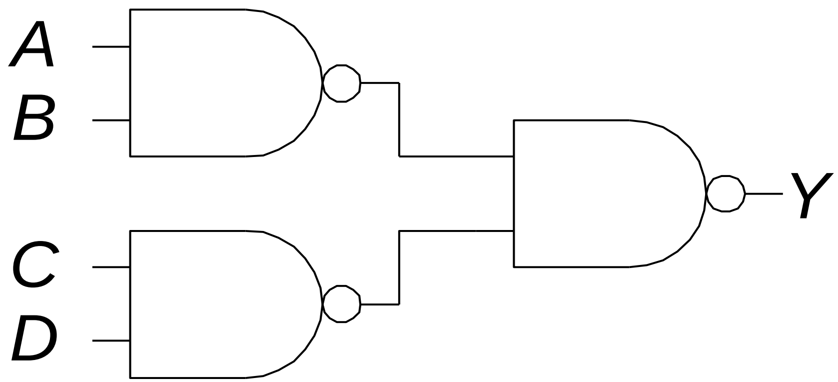
- **Backward:**
  - Body changes
  - Adds bubbles to inputs



- **Forward:**
  - Body changes
  - Adds bubble to output

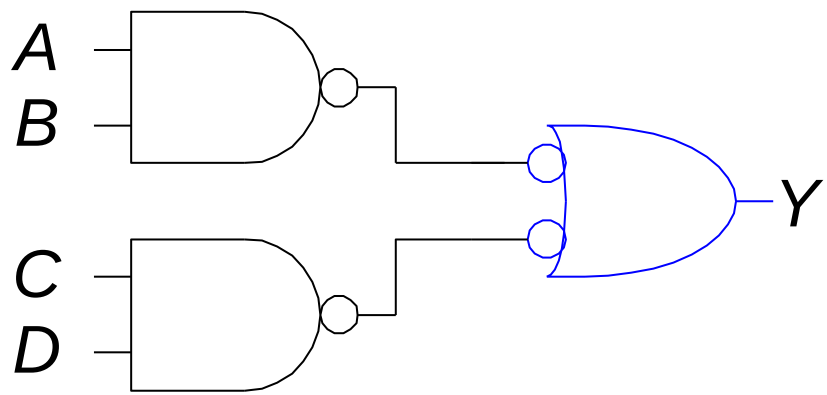
Bubble Pushing

- What is the Boolean expression for this circuit?



Bubble Pushing

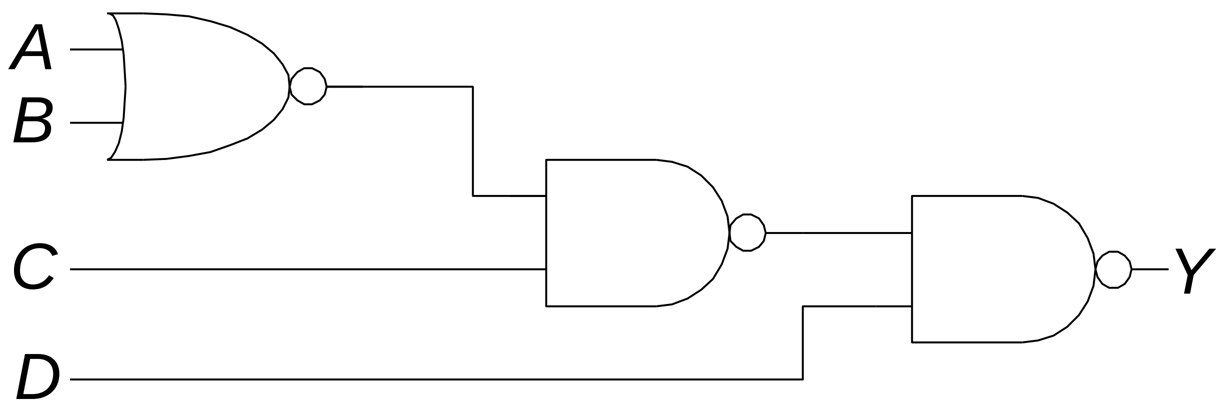
- What is the Boolean expression for this circuit?



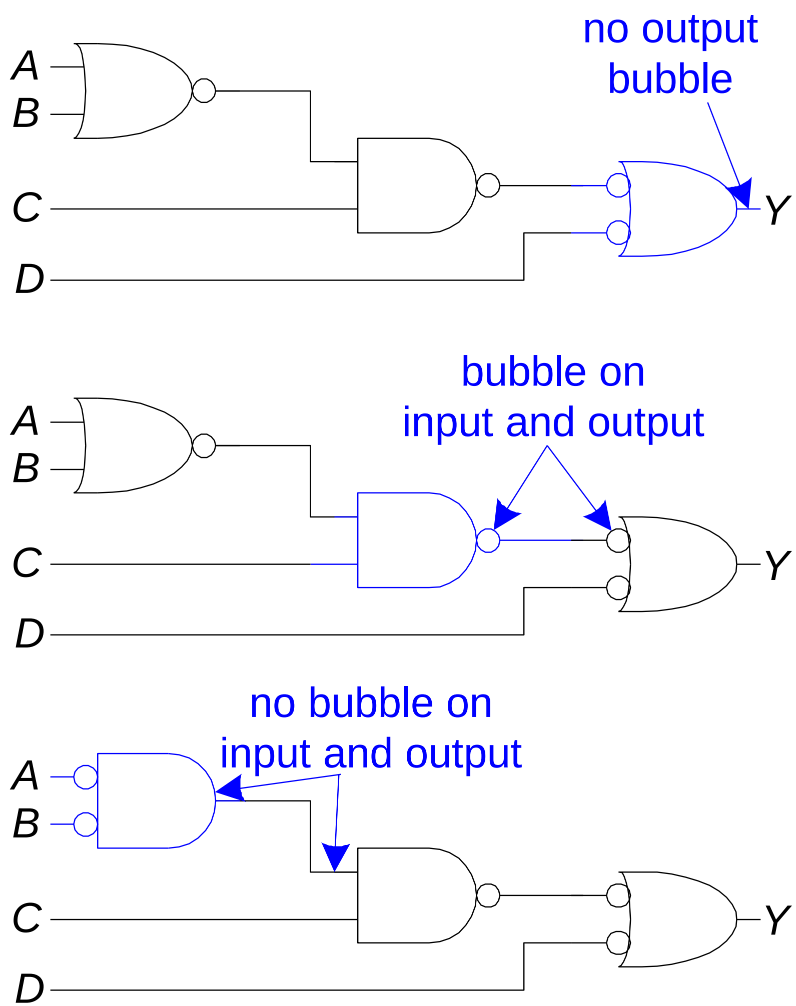
$$Y = AB + CD$$

Bubble Pushing Rules

- Begin at output, then work toward inputs
- Push bubbles on final output back
- Draw gates in a form so bubbles cancel



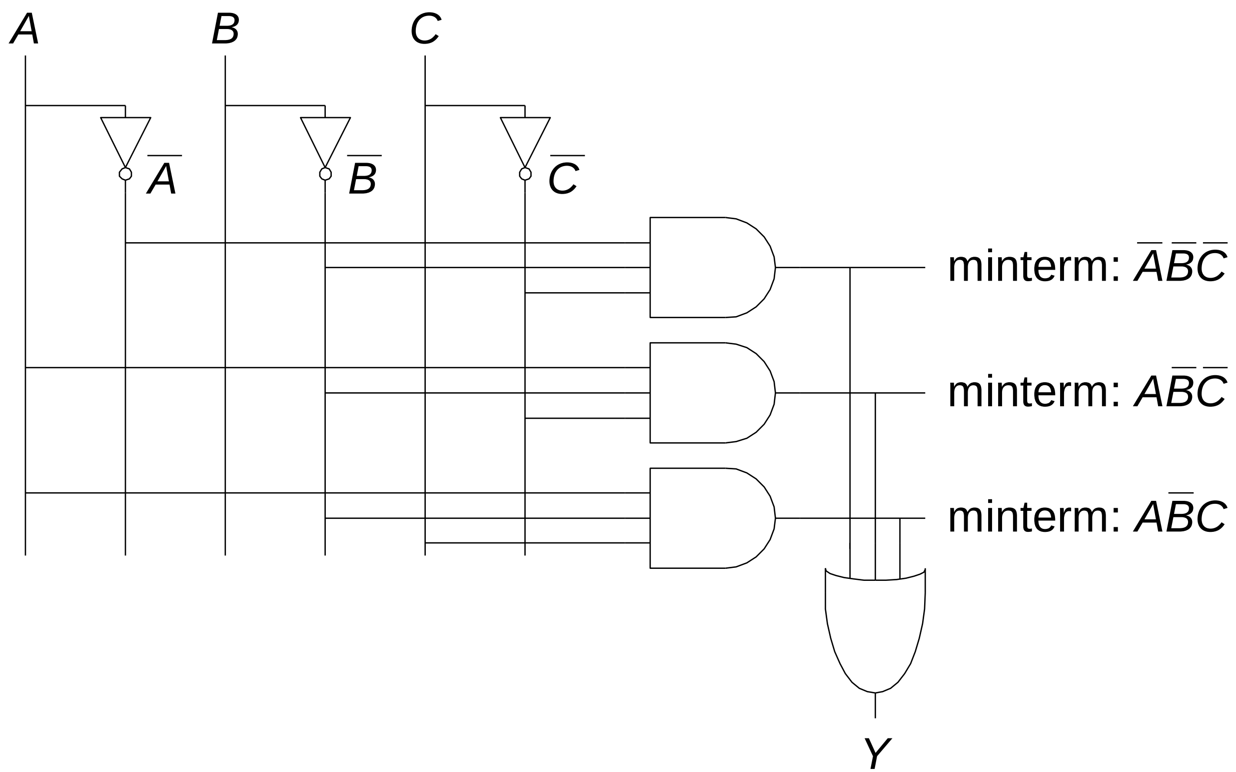
Bubble Pushing Example



$$Y = \overline{A}\,\overline{B}C + \overline{D}$$

From Logic to Gates

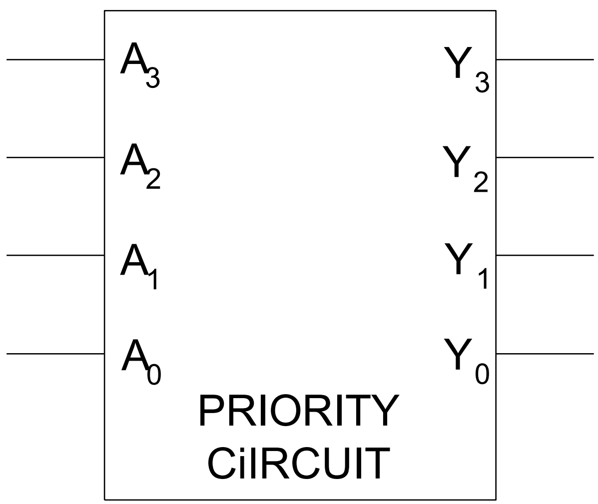
- Two-level logic: ANDs followed by ORs
- Example: $Y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C$



Multiple-Output Circuits

- **Example: Priority Circuit**

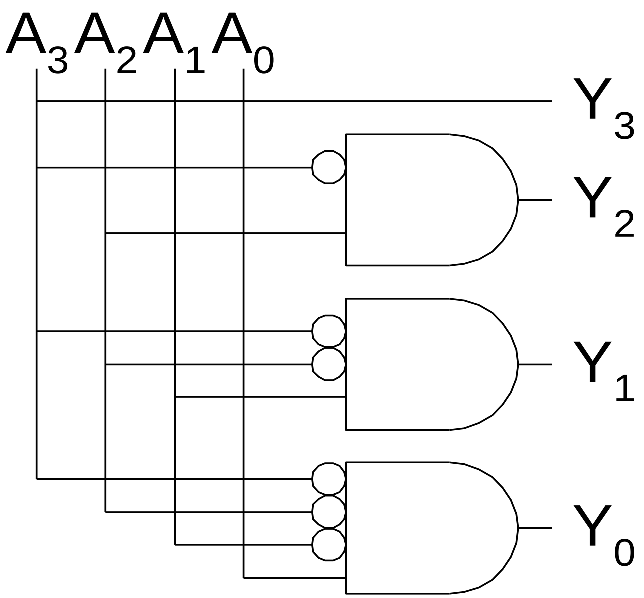
  Output asserted corresponding to most significant TRUE input



A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

Priority Circuit Hardware

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0



Don't Cares

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	**0**	**0**	**0**	**0**
0	0	0	1	**0**	**0**	**0**	**1**
0	0	1	0	**0**	**0**	**1**	**0**
0	0	1	1	**0**	**0**	**1**	**0**
0	1	0	0	**0**	**1**	**0**	**0**
0	1	0	1	**0**	**1**	**0**	**0**
0	1	1	0	**0**	**1**	**0**	**0**
0	1	1	1	**0**	**1**	**0**	**0**
1	0	0	0	**1**	**0**	**0**	**0**
1	0	0	1	**1**	**0**	**0**	**0**
1	0	1	0	**1**	**0**	**0**	**0**
1	0	1	1	**1**	**0**	**0**	**0**
1	1	0	0	**1**	**0**	**0**	**0**
1	1	0	1	**1**	**0**	**0**	**0**
1	1	1	0	**1**	**0**	**0**	**0**
1	1	1	1	**1**	**0**	**0**	**0**

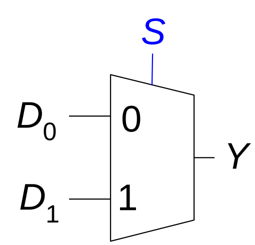
A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	**0**	**0**	**0**	**0**
0	0	0	1	**0**	**0**	**0**	**1**
0	0	1	X	**0**	**0**	**1**	**0**
0	1	X	X	**0**	**1**	**0**	**0**
1	X	X	X	**1**	**0**	**0**	**0**

Some Common Circuits

- We now look at some common combinational circuits that are used in designing a CPU.
- These include: multiplexor, decoder and adders.

Multiplexer (Mux)

- Selects between one of N inputs to connect to output
- $\log_2 N$-bit select input – control input
- **Example:** **2:1 Mux**

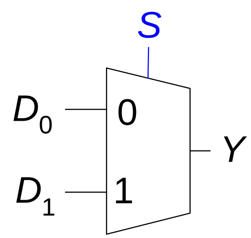


S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S	Y
0	D_0
1	D_1

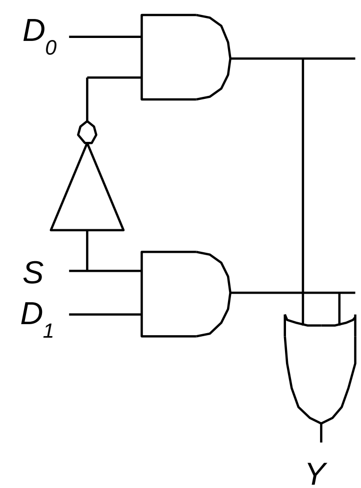
Multiplexer Implementations

- **Logic gates**



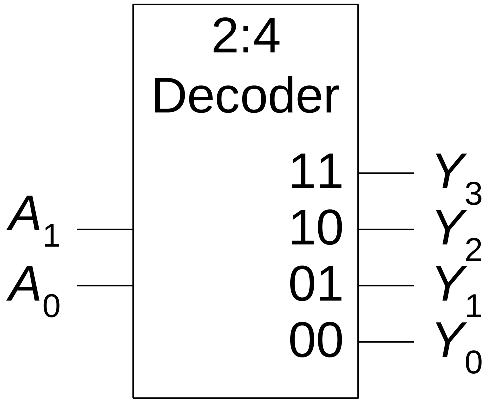
S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S	Y
0	D_0
1	D_1



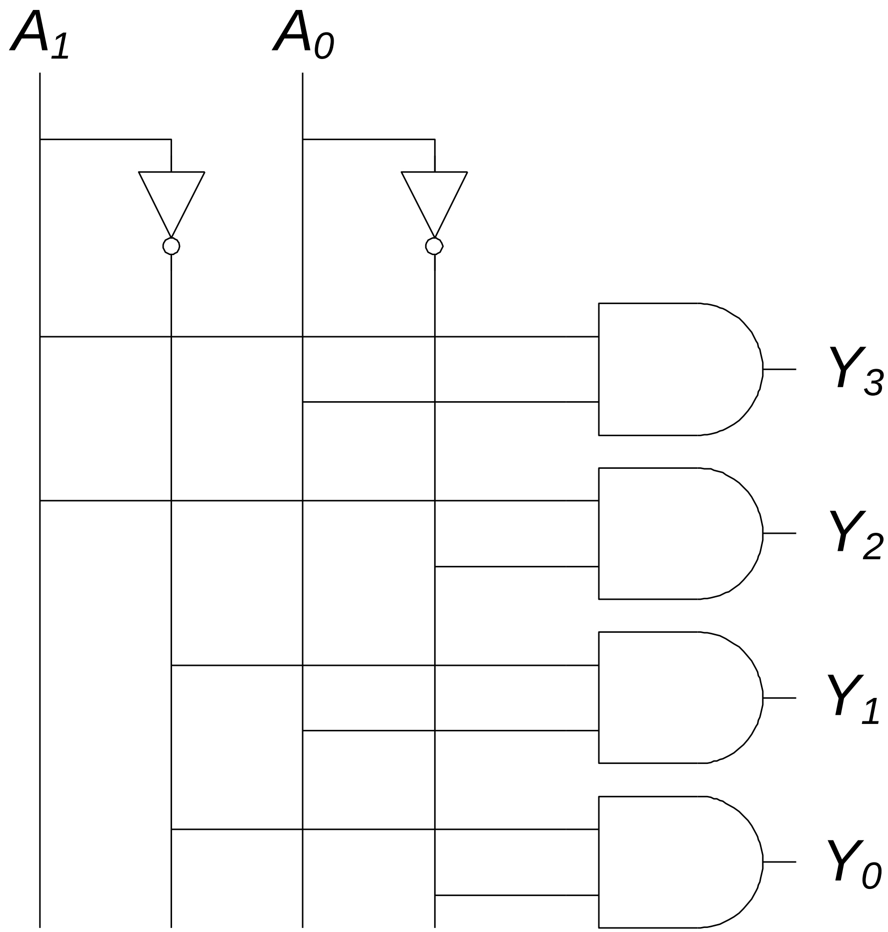
Decoders

- N inputs, 2^N outputs
- One-hot outputs: only one output HIGH at once



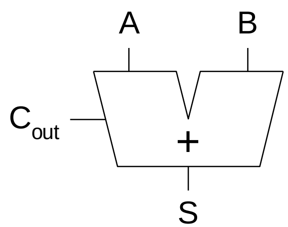
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Decoder Implementation



1-Bit Adders

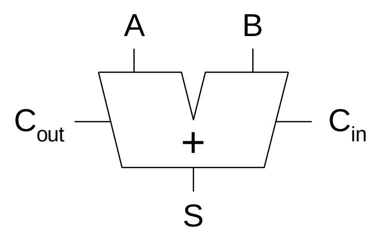
Half Adder



A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$
$$C_{out} = AB$$

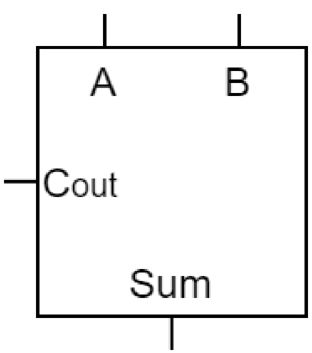
Full Adder



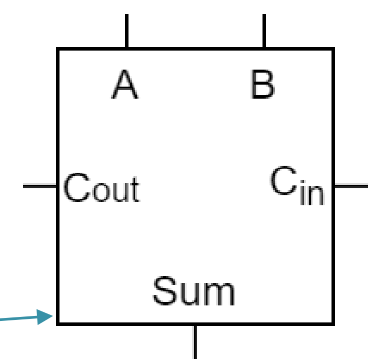
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Half Adder

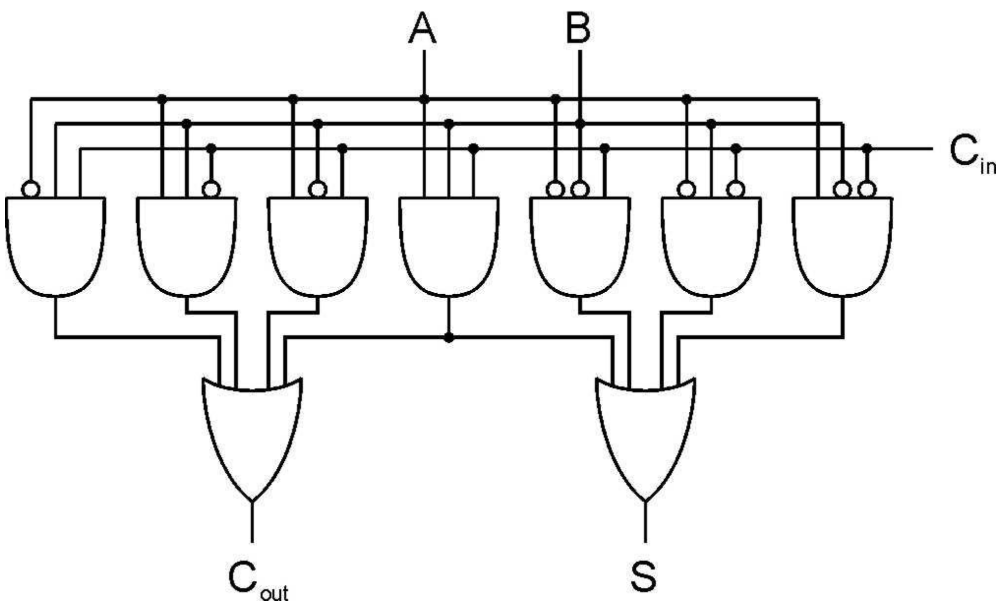


Full Adder



Alternate Drawings

Full Adder (1-bit addition)



C_{in}	A	B	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

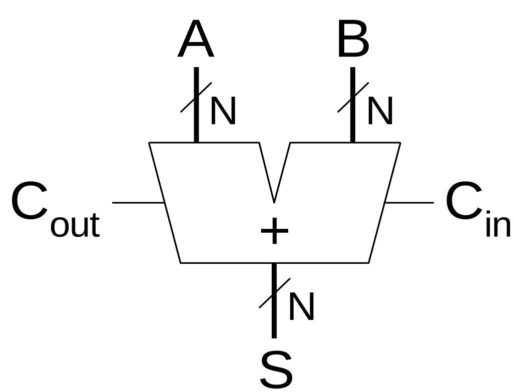
Multibit Adders (CPAs)

- Types of Carry Propagate Adders (CPAs):
  - Ripple-carry (slow but simple)
  - Carry-lookahead (fast)
  - Prefix (faster)

The Carry-lookahead and Prefix Adders are NOT in any labs, assignments, or tests

- Carry-lookahead and Prefix adders are faster for large adders with several bits, but require more hardware.

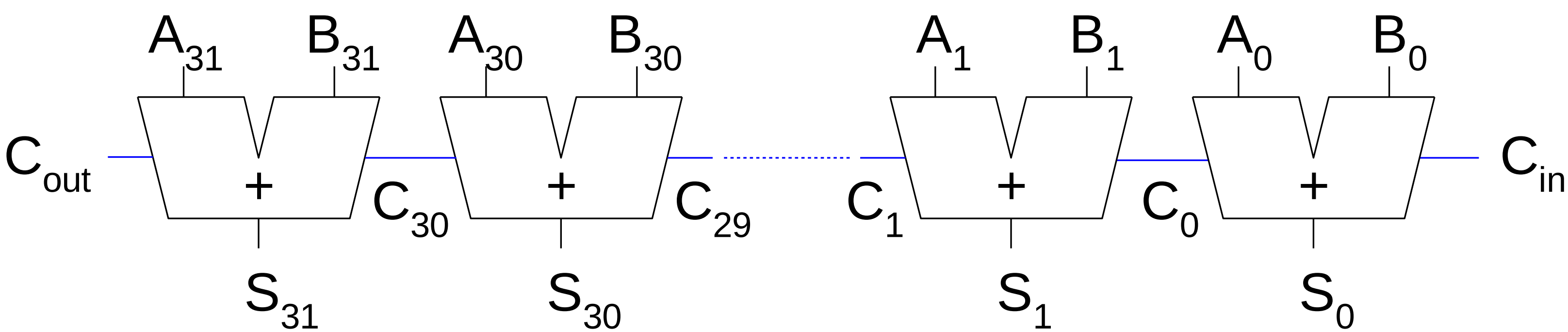
Symbol



The '/ N' denotes N input lines
In this case, N bits of A and B

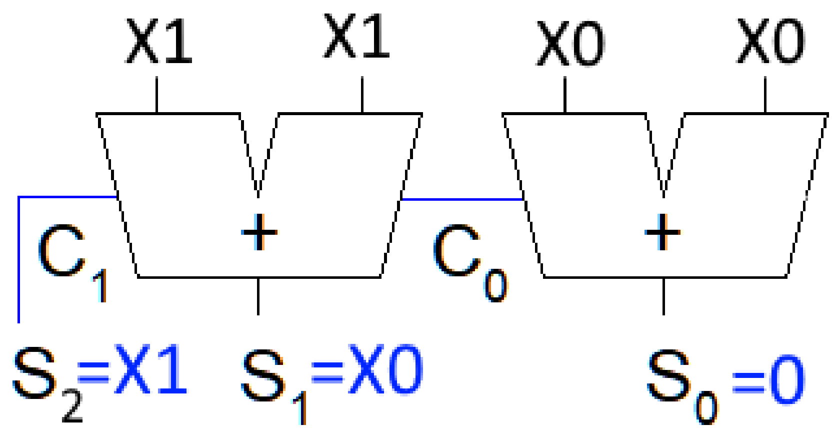
Ripple-Carry Adder

- Chain 1-bit adders together
- Carry "ripples" through entire chain
  - Ex: 0111 1111 1111 1111 + 1
- Disadvantage: **slow**

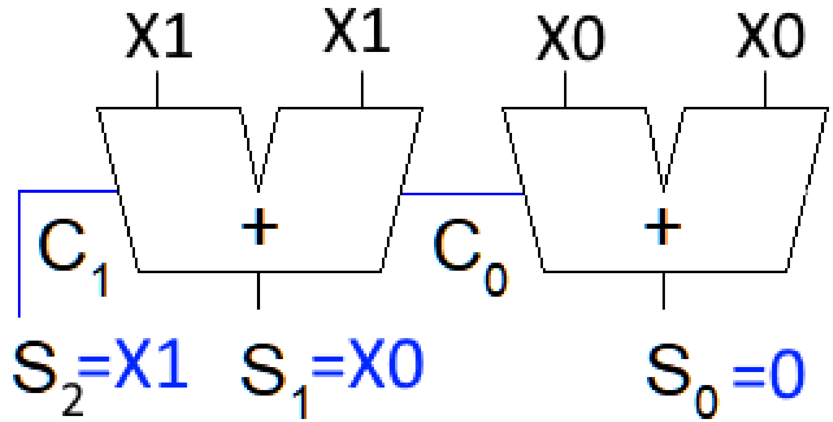


Ripple-Carry Adder - Example

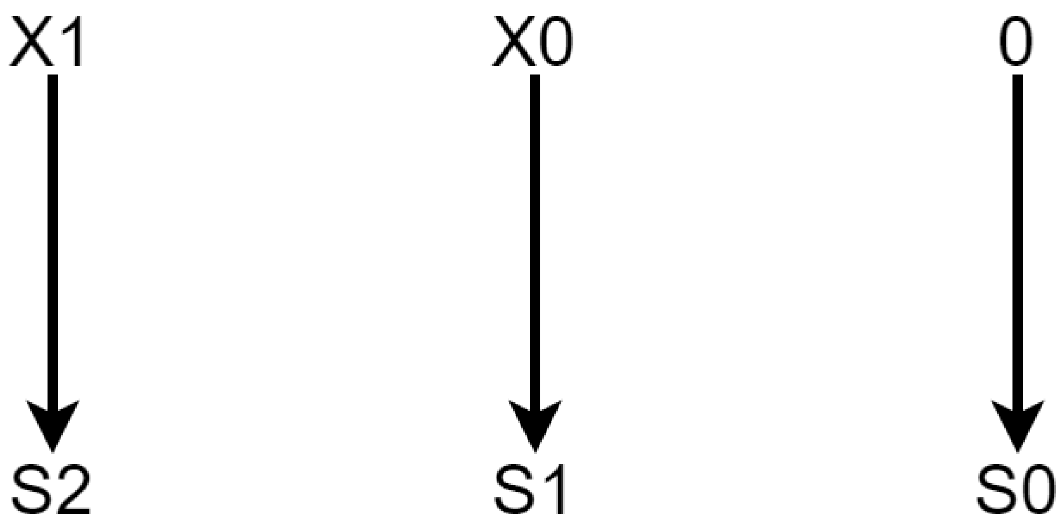
- Let's add X to itself
- Let X be a 2-bit number, therefore: $X = X_1 X_0$
- It just so happens that $X_1X_0 + X_1X_0 = X_1X_00$
- **Why? Could we take a shortcut?**



Ripple-Carry Adder - Shortcut

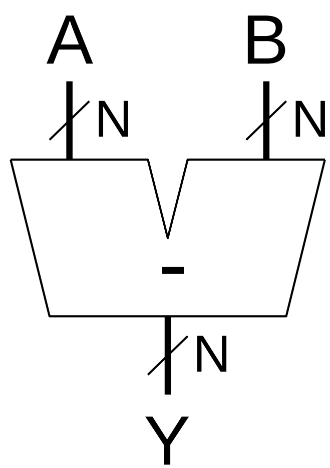


- **New and Improved:**

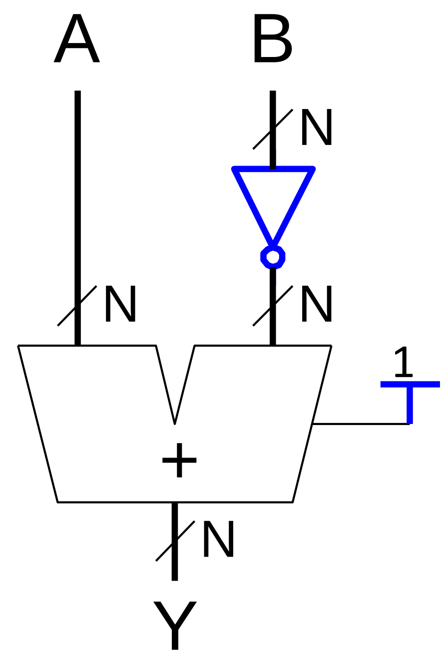


Subtracter

Symbol



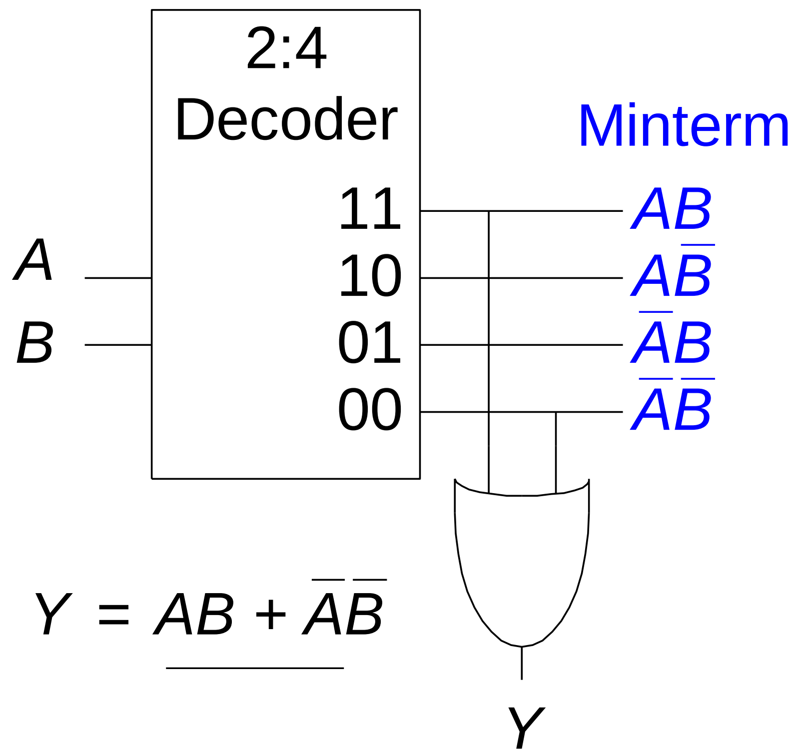
Implementation



The '/ N' denotes N input lines
In this case, N bits of A and B

Logic Using Decoders

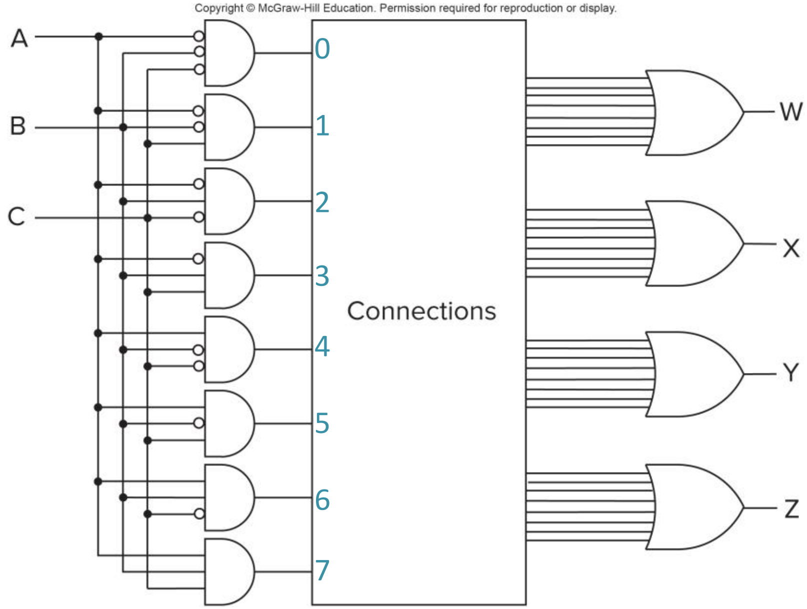
- Sum (OR) of Products (AND/minterms)



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

Programmable Logic Array (PLA)

- It is possible to build a logic circuit that uses logic circuits to decide what logic circuits to implement!



The Programmable Logic Array is NOT in any labs, assignments, or tests

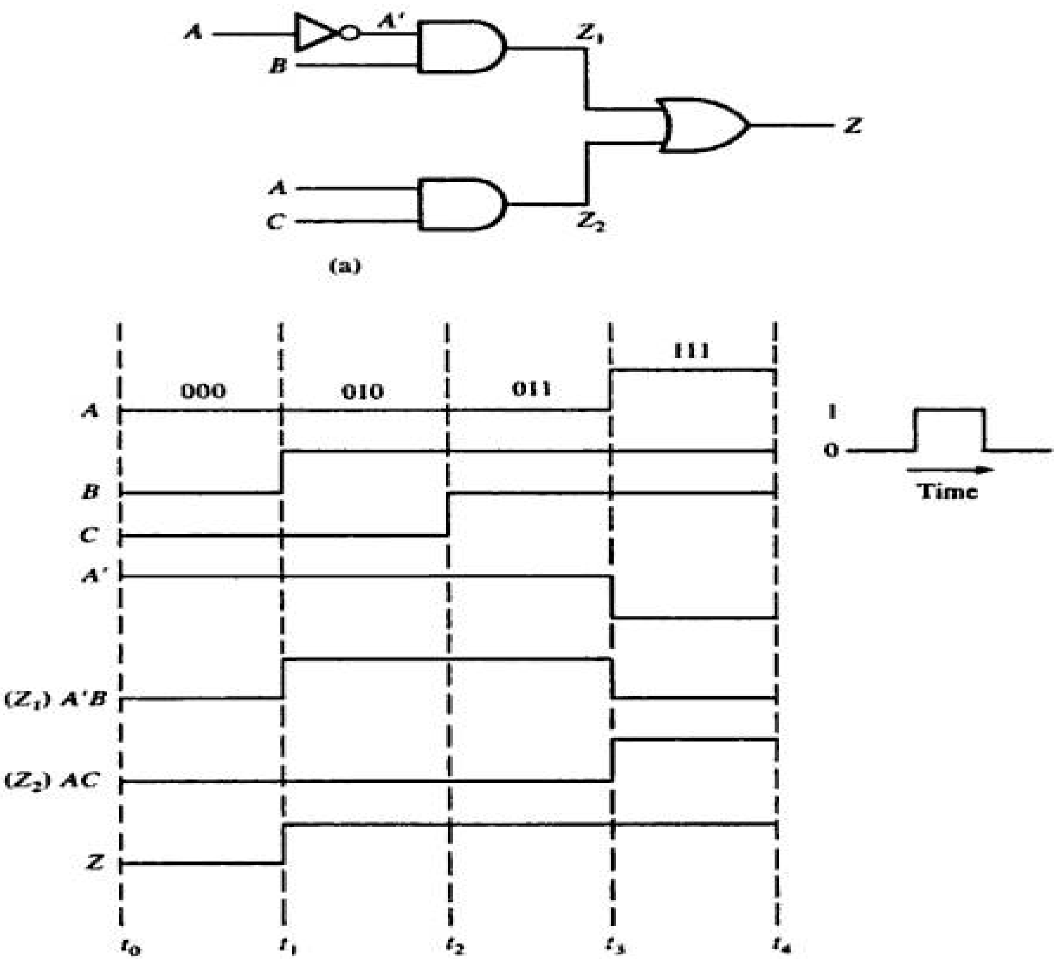
This PLA could, for instance, be used to build a 1-bit adder circuit by connecting ***W*** *to outputs 3, 5, 6 & 7 = C(i+1); and* ***X*** *to outputs 1, 2, 4, & 7 = S(i).*

Timing Diagrams

- A timing diagram is a graphical representation of the input and output signals (values) of circuit, where the domain is time.
- Timing diagrams are used to illustrate how a circuit works and are useful for debugging circuits.
- Commercial debugging tools such as oscilloscopes and logic analyzers generate timing diagrams for circuits they are connected to.
- When drawing a timing diagram for a circuit, you should display all your inputs.
- You should display the outputs you want to examine in your diagram.

Timing Diagrams

We will discuss this diagram in class.



Timing Diagrams

- In reality, each gate introduces a slight delay in a circuit. You can draw these into your diagram if you want.
- These delays can sometimes cause glitches in circuits.
  - An output jumps momentarily from 0 to 1 and then back to 0,

    or
  - An output jumps momentarily from 1 to 0 and then back to 1.
- These can be hard to debug, but can be dealt with by adding extra gates to the circuit.
- We will encounter timing diagrams throughout the course.