Logic Gates

Digital Logic Design (DLD)

Introduction

- Digital systems are concerned with digital signals
- Digital signals can take many forms
- Here we will concentrate on **binary signals** since these are the most common form of digital signals
  - can be used individually
    - perhaps to represent a single binary quantity or the state of a single switch
  - can be used in combination
    - to represent more complex quantities

Boolean Constants and Variables

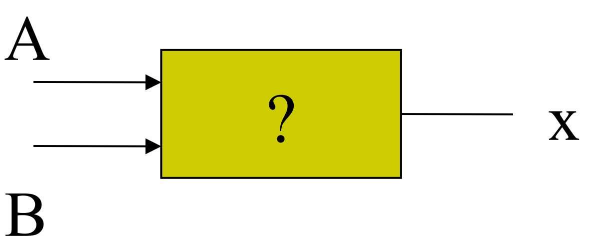
- Boolean 0 and 1 do not represent actual numbers but instead represent the <u>state</u>, or <u>logic level</u>.

Logic 0	Logic 1
False	True
Off	On
Low	High
No	Yes
Open switch	Closed switch

Truth Tables

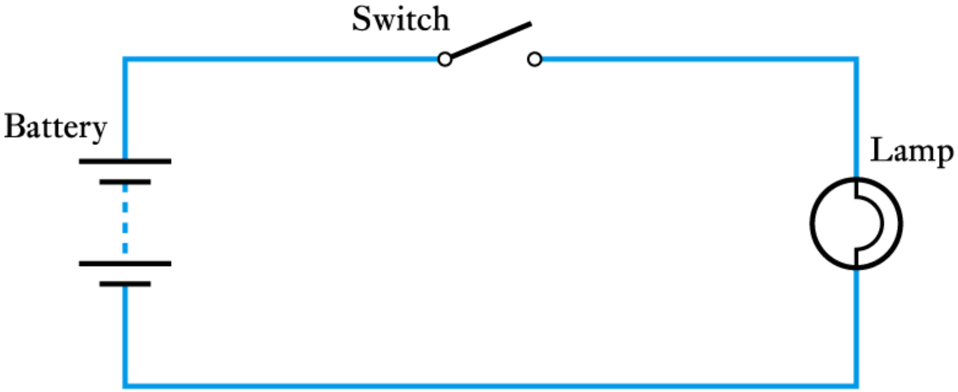
- A truth table is a means for describing how a logic circuit's output depends on the logic levels present at the circuit's inputs.

Inputs		Output
A	B	x
0	0	1
0	1	0
1	0	1
1	1	0



Binary Quantities and Variables

- A **binary quantity** is one that can take only 2 states



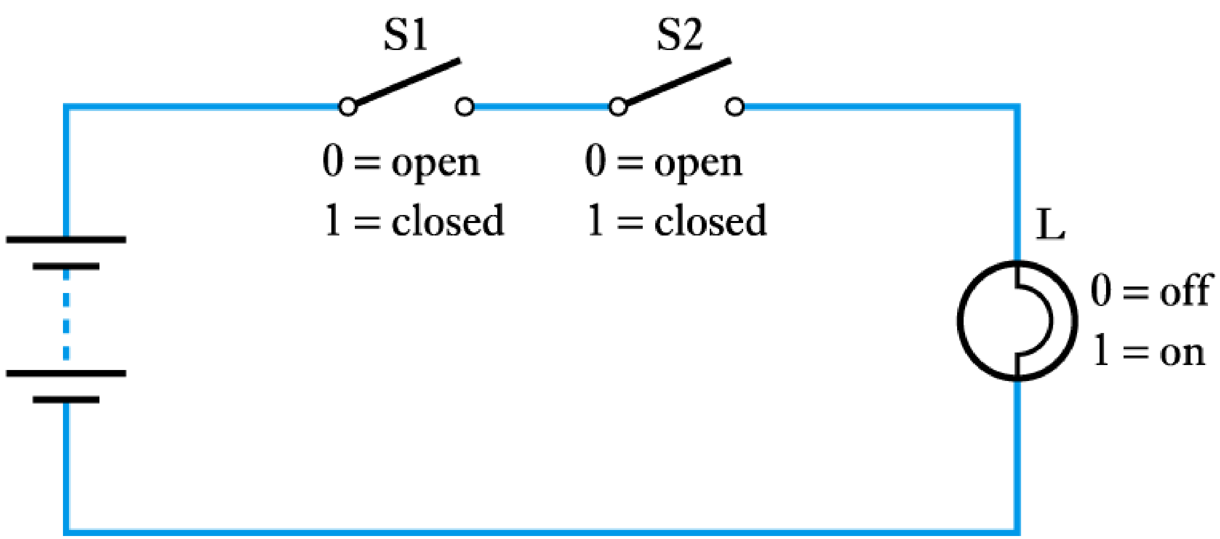
A simple binary arrangement

S	*L*
OPEN	OFF
CLOSED	ON

S	*L*
0	0
1	1

A truth table

- A binary arrangement with two switches in series



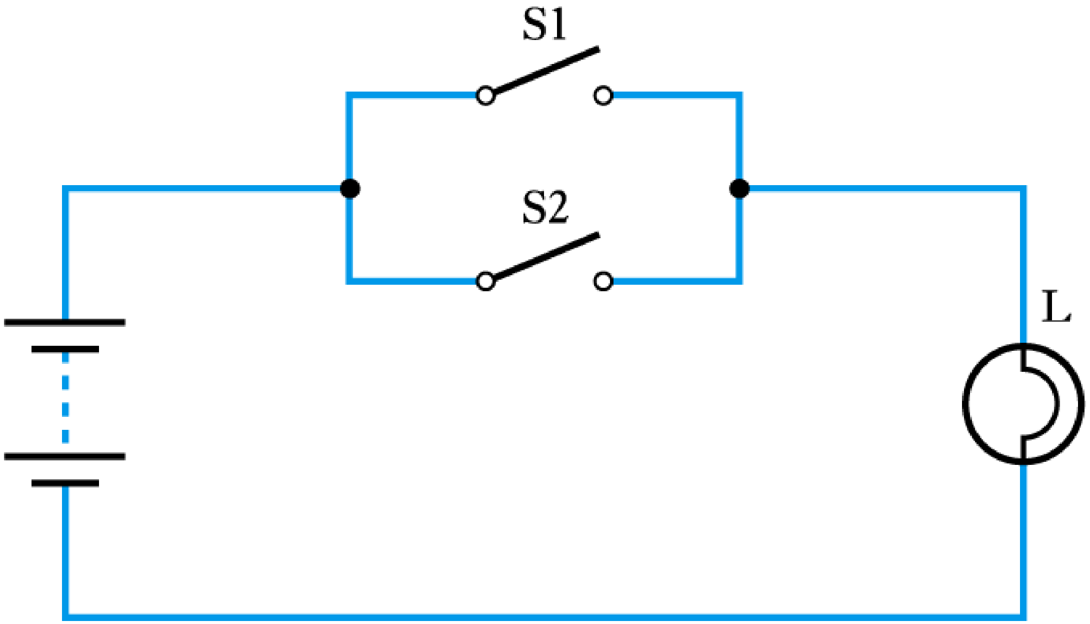
(a) Circuit

S1	S2	L
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

L = *S1* AND *S2*

- A binary arrangement with two switches in parallel



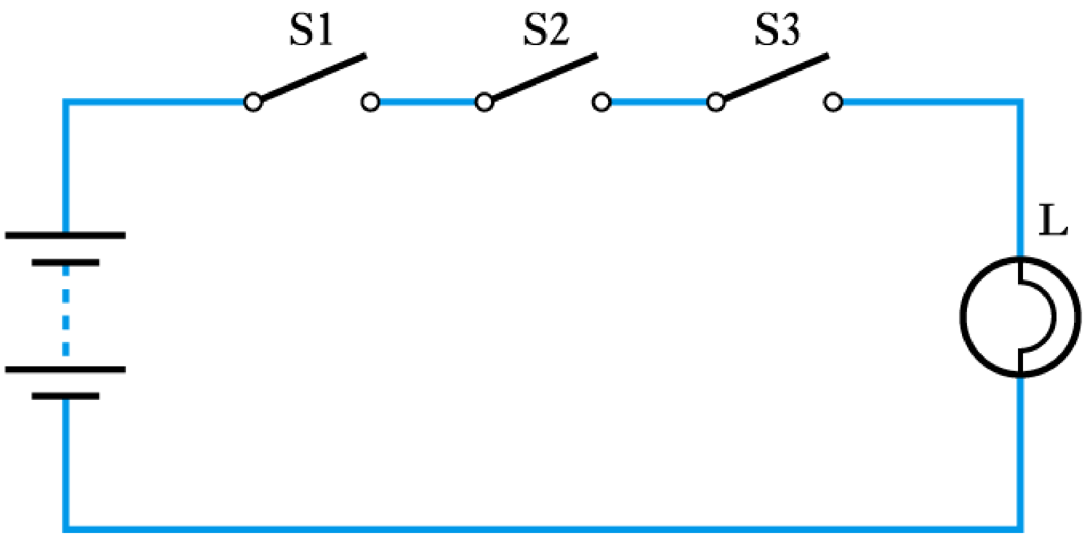
(a) Circuit

S1	S2	L
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

$L = S1$ OR $S2$

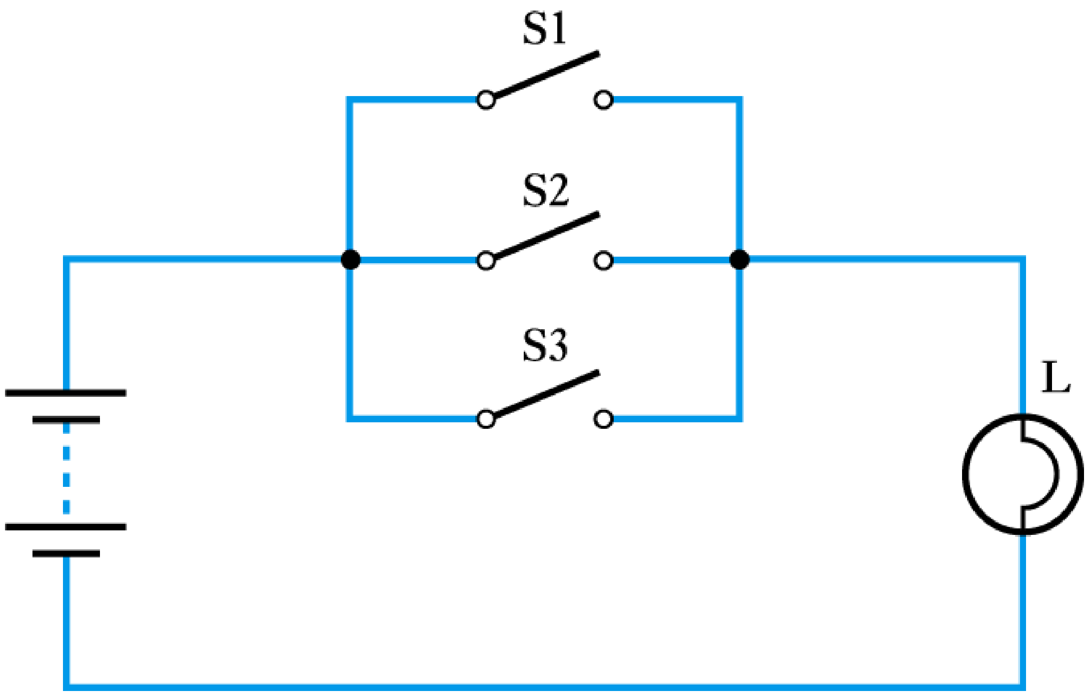
- Three switches in series



S1	S2	S3	L
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

L = *S1* AND *S2* AND *S3*

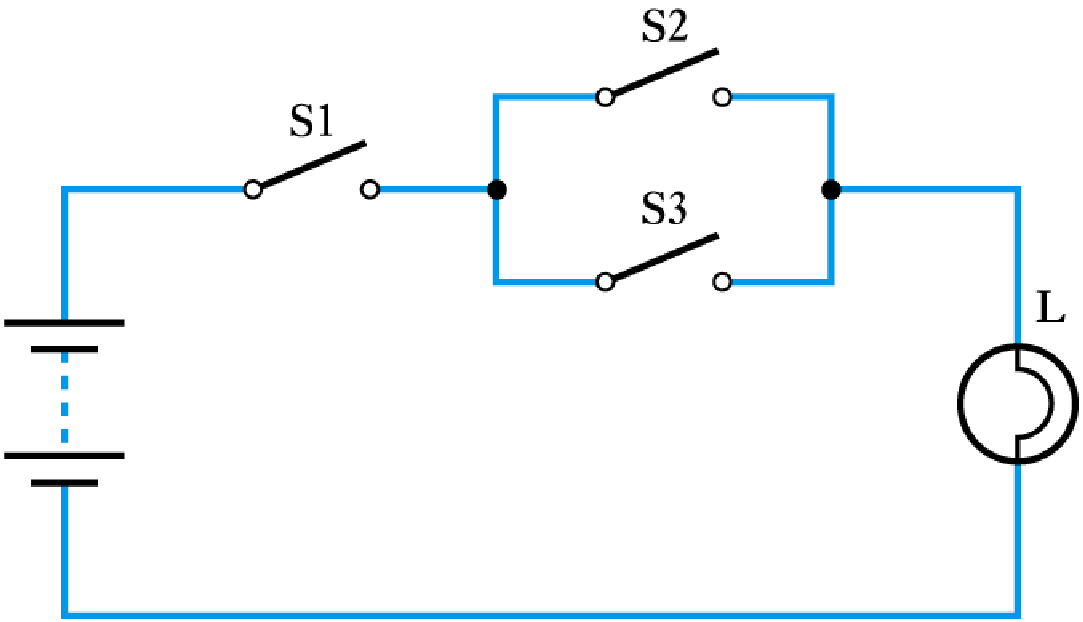
- Three switches in parallel



S1	S2	S3	L
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$L = S1$ OR $S2$ OR $S3$

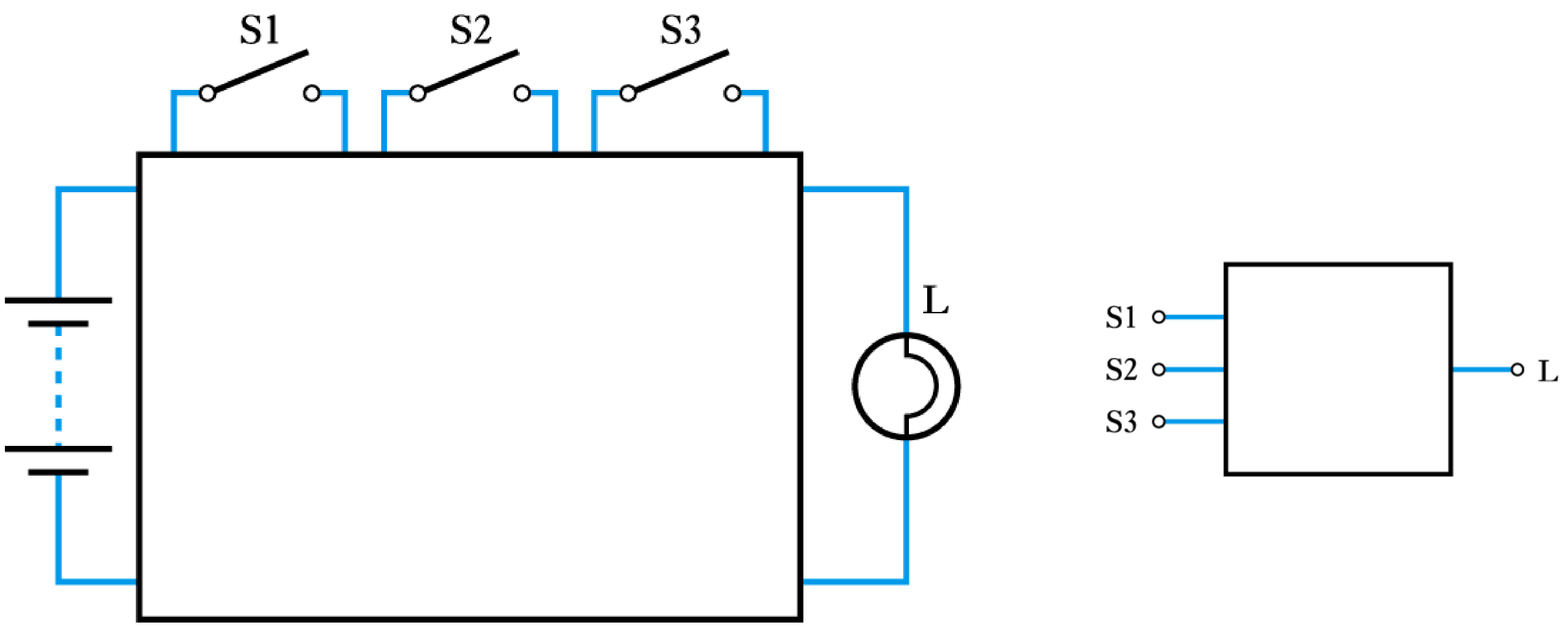
- A series/parallel arrangement



S1	S2	S3	L
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

L = *S1* AND (*S2* OR *S3*)

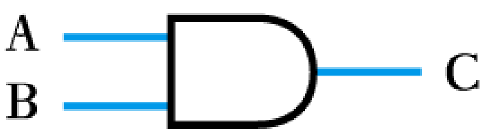
- Representing an unknown network



Logic Gates

- The building blocks used to create digital circuits are **logic gates**
- There are three elementary logic gates and a range of other simple gates
- Each gate has its own **logic symbol** which allows complex functions to be represented by a logic diagram
- The function of each gate can be represented by a **truth table** or using **Boolean notation**

The AND gate



(a) Circuit symbol

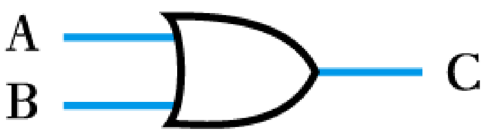
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

$$C = A \cdot B$$

(c) Boolean expression

The OR gate



(a) Circuit symbol

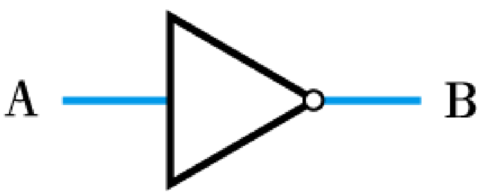
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

$$C = A + B$$

(c) Boolean expression

The NOT gate (or inverter)



(a) Circuit symbol

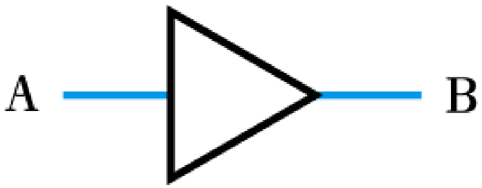
A	B
0	1
1	0

(b) Truth table

$$B = \overline{A}$$

(c) Boolean expression

A logic buffer gate



(a) Circuit symbol

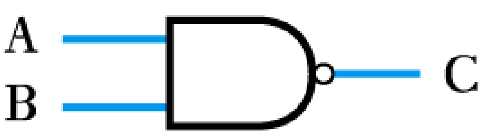
A	B
0	0
1	1

(b) Truth table

$$B = A$$

(c) Boolean expression

The NAND gate



(a) Circuit symbol

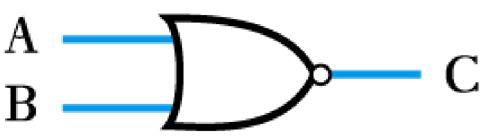
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

(b) Truth table

$$C = \overline{A \cdot B}$$

(c) Boolean expression

- ## The NOR gate



(a) Circuit symbol

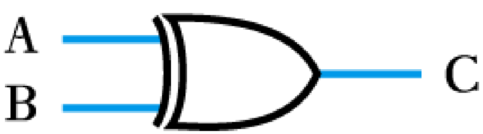
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

(b) Truth table

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

(c) Boolean expression

The Exclusive OR gate



(a) Circuit symbol

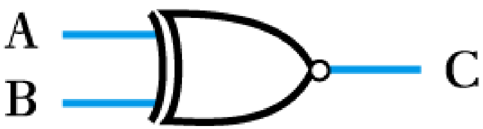
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

(b) Truth table

$$C = A \oplus B$$

(c) Boolean expression

The Exclusive NOR gate



(a) Circuit symbol

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

(b) Truth table

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

(c) Boolean expression

Boolean Algebra

- **Boolean Constants**
  – these are ‘0’ (false) and ‘1’ (true)
- **Boolean Variables**
  – variables that can only take the vales ‘0’ or ‘1’
- **Boolean Functions**
  – each of the logic functions (such as AND, OR and NOT) are represented by symbols as described above
- **Boolean Theorems**
  – a set of **identities** and **laws** – see text for details

Boolean identities

AND Function	OR Function	NOT function
$0 \bullet 0 = 0$	$0 + 0 = 0$	$\overline{0} = 1$
$0 \bullet 1 = 0$	$0 + 1 = 1$	$\overline{1} = 0$
$1 \bullet 0 = 0$	$1 + 0 = 1$	$\overline{\overline{A}} = A$
$1 \bullet 1 = 1$	$1 + 1 = 1$	
$A \bullet 0 = 0$	$A + 0 = A$	
$0 \bullet A = 0$	$0 + A = A$	
$A \bullet 1 = A$	$A + 1 = 1$	
$1 \bullet A = A$	$1 + A = 1$	
$A \bullet A = A$	$A + A = A$	
$A \bullet \overline{A} = 0$	$A + \overline{A} = 1$	

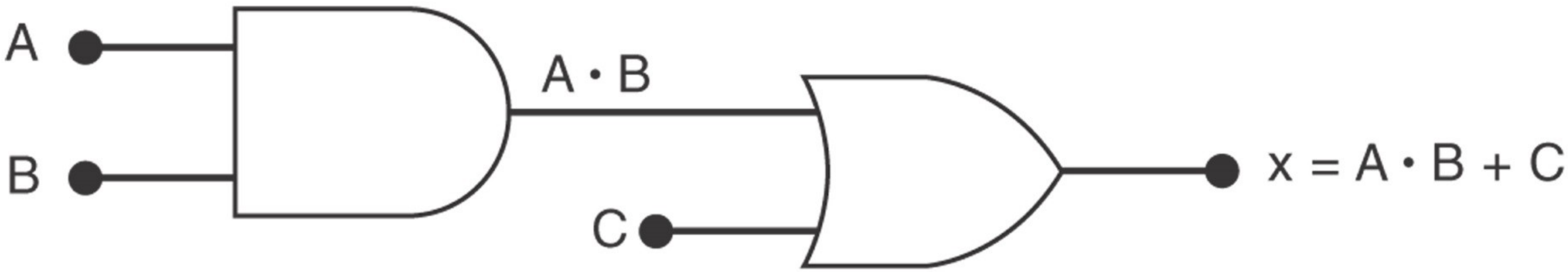
Boolean laws

Commutative law	Absorption law
$AB = BA$ $A + B = B + A$	$A + AB = A$ $A(A + B) = A$
Distributive law	**De Morgan's law**
$A(B + C) = AB + BC$ $A + BC = (A + B)(A + C)$	$\overline{A + B} = \overline{A} \bullet \overline{B}$ $\overline{A \bullet B} = \overline{A} + \overline{B}$
Associative law	**Note also**
$A(BC) = (AB)C$ $A + (B + C) = (A + B) + C$	$A + \overline{A}B = A + B$ $A(\overline{A} + B) = AB$

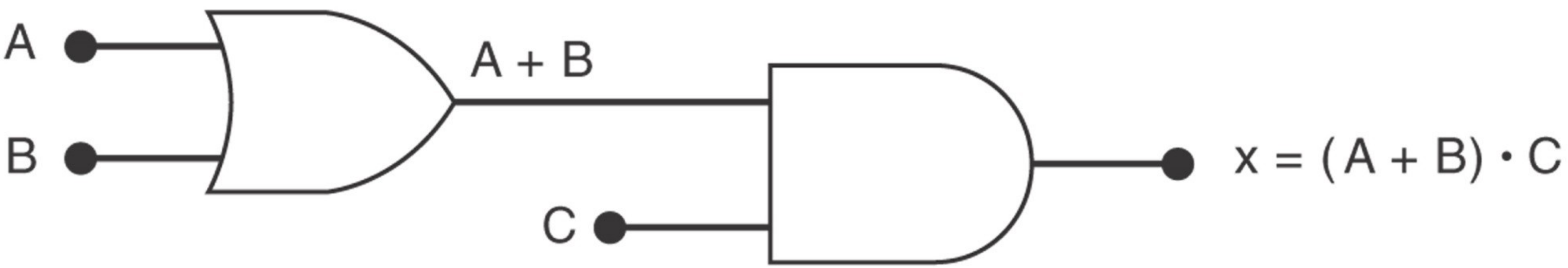
Combinational Logic

- Digital systems may be divided into two broad categories:
  - **combinational logic**
    - where the outputs are determined solely by the current states of the inputs
  - **sequential logic**
    - where the outputs are determined not only by the current inputs but also by the sequence of inputs that led to the current state
- In this lecture we will look at combination logic

Examples 1,2

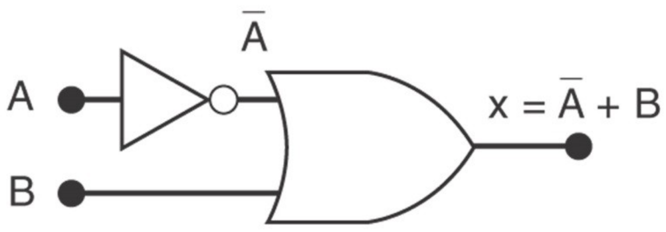


(a)

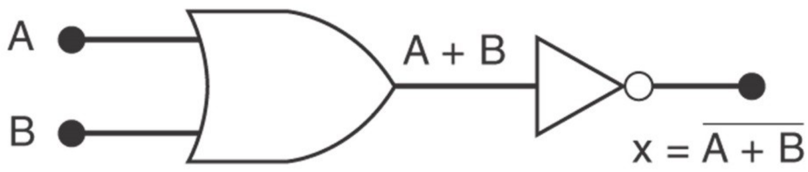


(b)

Examples 3

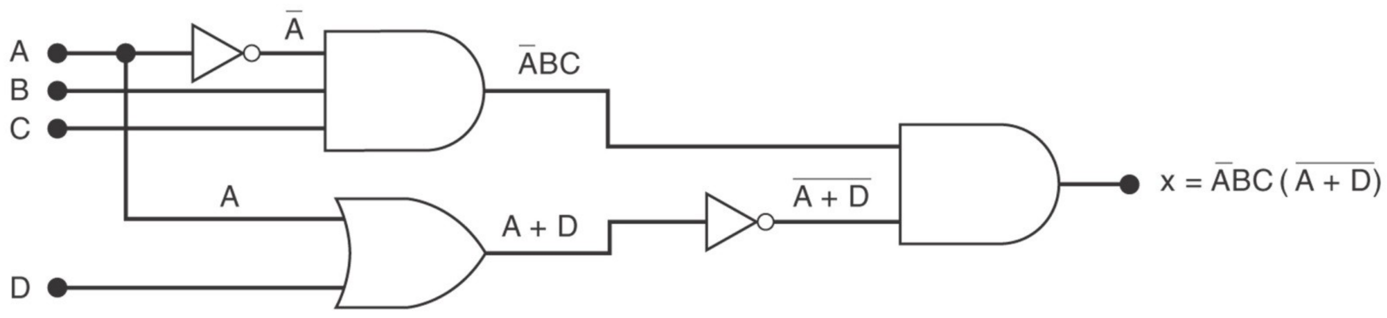


(a)

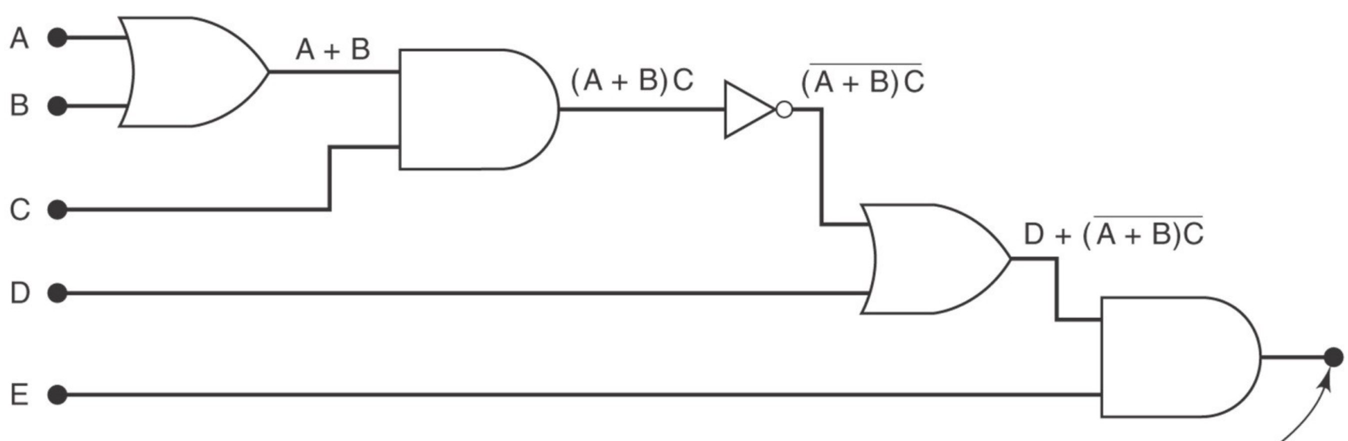


(b)

Example 4


A
B
C
D
$\overline{A}$
$\overline{A}BC$
A
A + D
$\overline{A + D}$
$x = \overline{A}BC(\overline{A + D})$

(a)

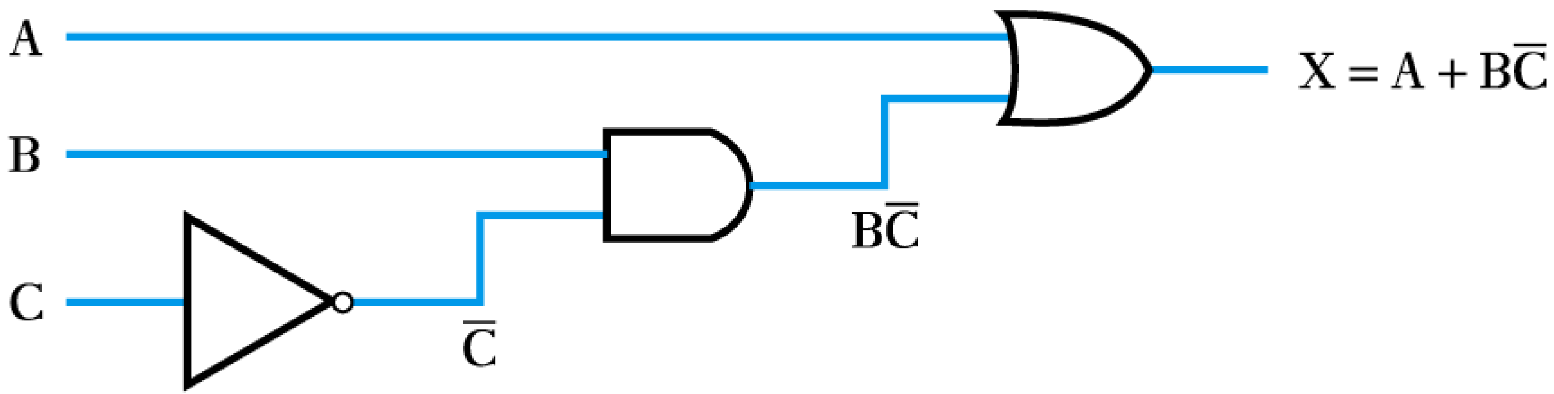

A
B
C
D
E
A + B
(A + B)C
$\overline{(A + B)C}$
$D + \overline{(A + B)C}$
$x = [D + \overline{(A + B)C}] \cdot E$

(b)

Implementing a function from a Boolean expression

Example –

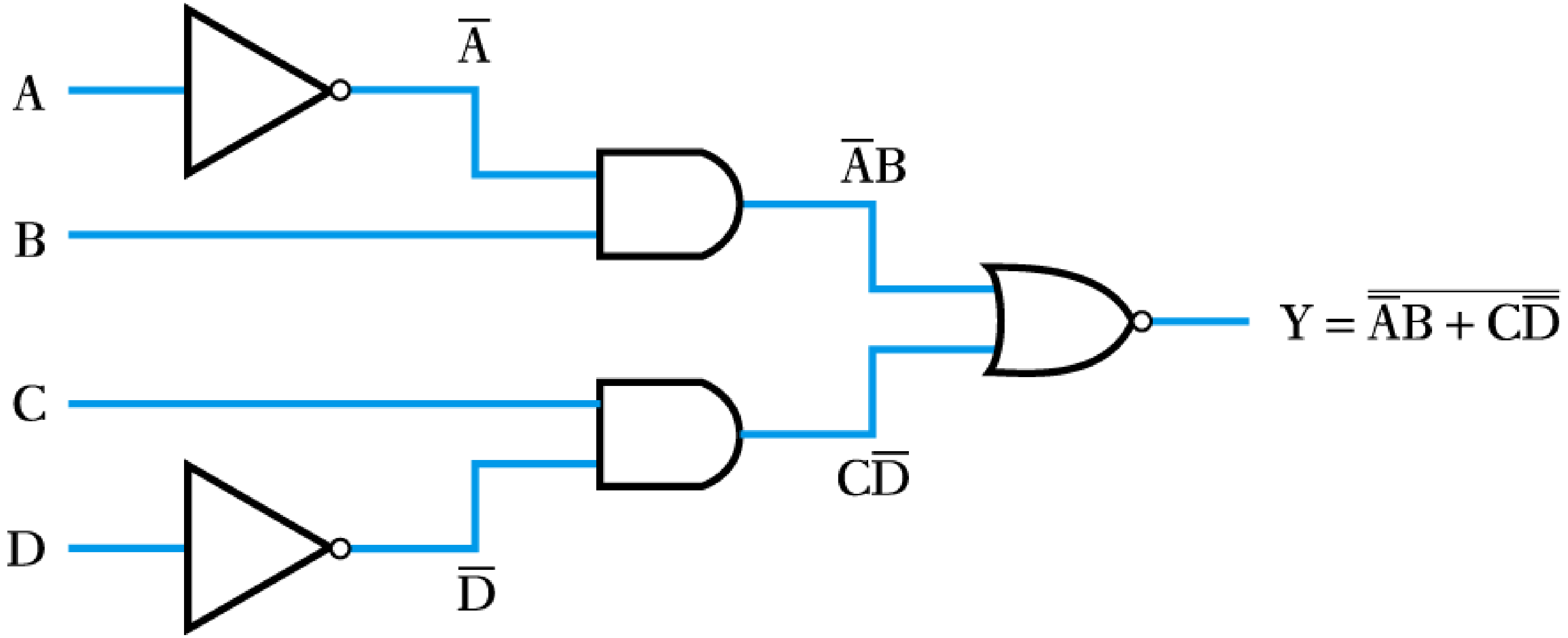
$$X = A + B\overline{C}$$



- Implementing a function from a Boolean expression

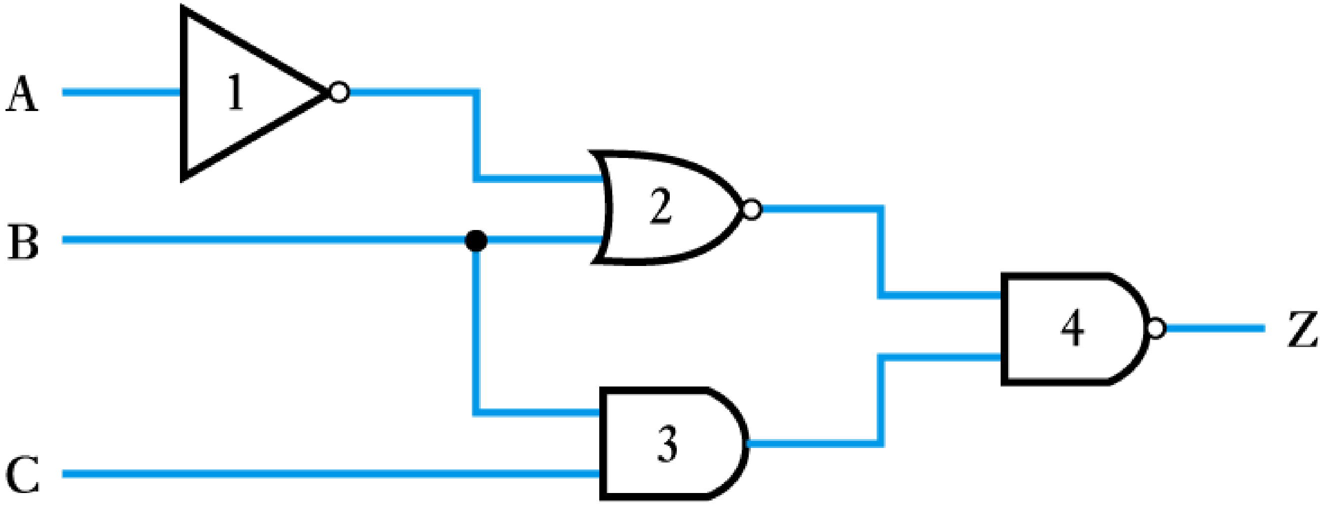
Example –

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



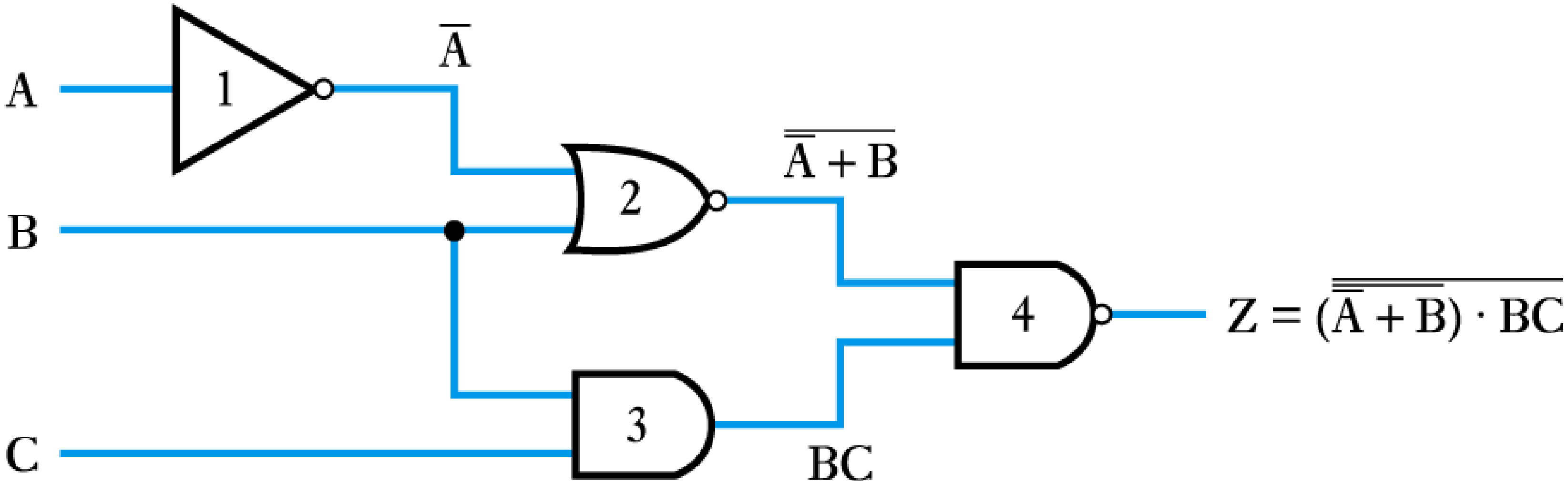
- **Generating a Boolean expression from a logic diagram**

Example –



Example (continued)

- work progressively from the inputs to the output adding logic expressions to the output of each gate in turn



- **Implementing a logic function from a description**

Example –

The operation of the Exclusive OR gate can be stated as:

"The output should be true if either of its inputs are true, but not if both inputs are true."

This can be rephrased as:

"*The output is true if A* OR *B is true*, AND *if A* AND *B are* NOT *true*."

We can write this in Boolean notation as

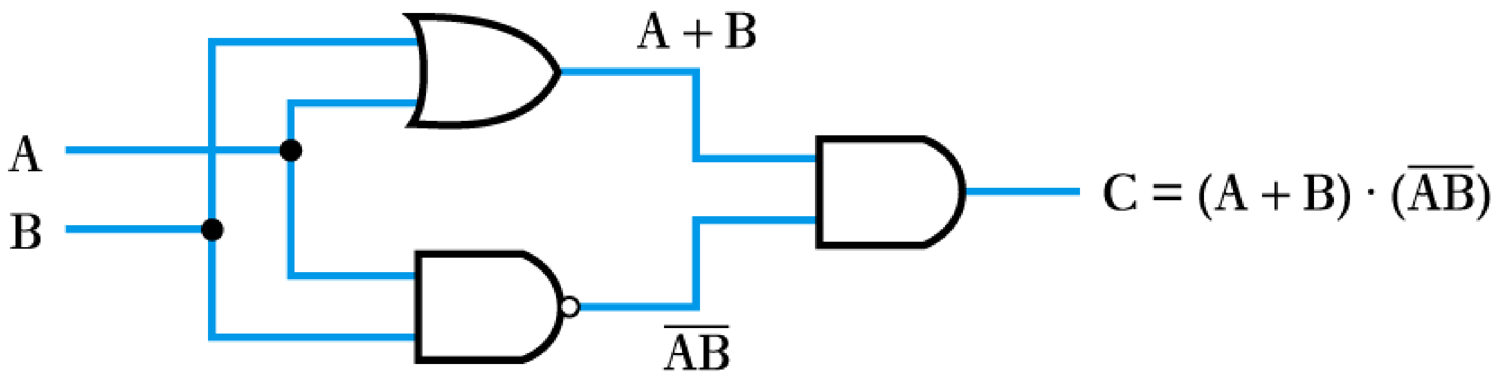
$$X = (A + B) \bullet \overline{(AB)}$$

Example (continued)

The logic function

$$X = (A+B) \bullet \overline{(AB)}$$

can then be implemented as before



Implementing a logic function from a truth table

Example –

Implement the function of the following truth table

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

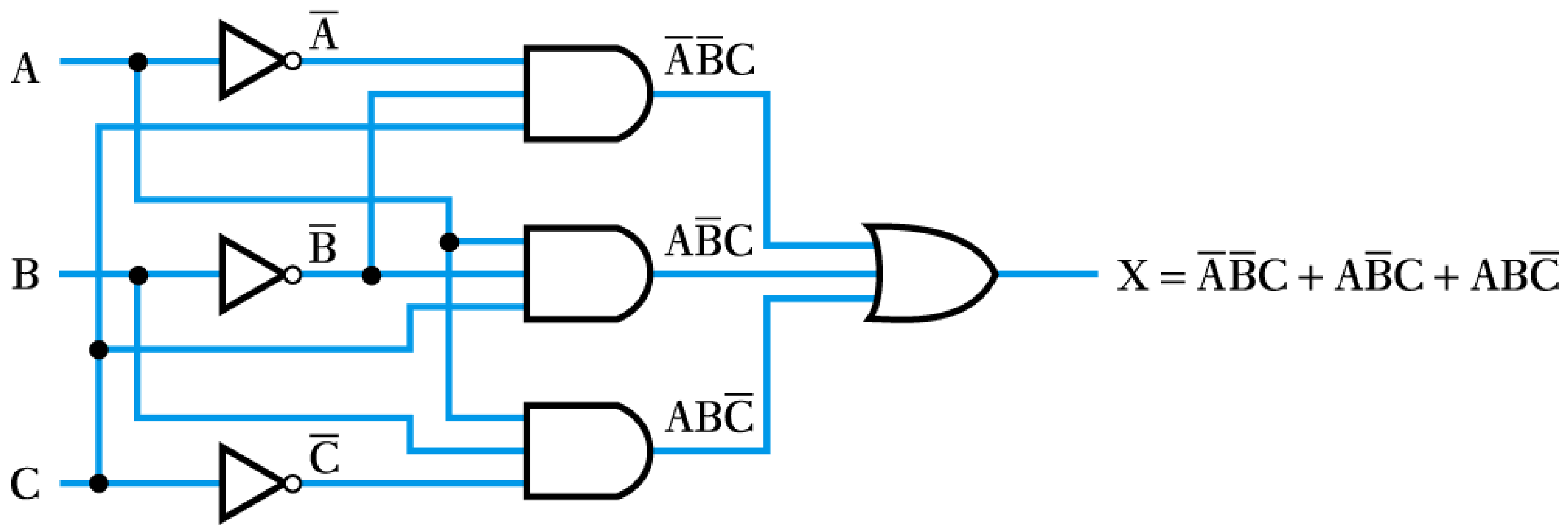
- first write down a Boolean expression for the output
- then implement as before
- in this case

$$X = \overline{A}\,\overline{B}C + A\overline{B}C + AB\overline{C}$$

Example (continued)

The logic function $X = \overline{A}\,\overline{B}C + A\overline{B}C + AB\overline{C}$

can then be implemented as before

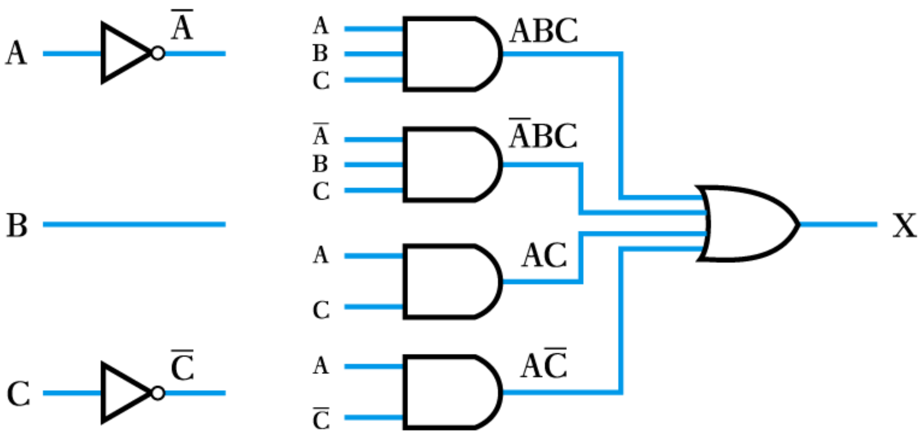


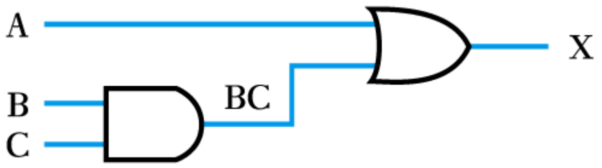
- In some cases it is possible to *simplify* logic expressions using the rules of Boolean algebra

Example –

$X = ABC + \overline{A}BC + AC + A\overline{C}$ can be simplified to $X = BC + A$

hence the following circuits are equivalent





Number Systems and Binary Arithmetic

- Most number systems are order dependent
- **Decimal**

$$1234_{10} = (1 \times 10^3) + (2 \times 10^2) + (3 \times 10^1) + (4 \times 10^0)$$

- **Binary**

$$1101_2 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$

- **Octal**

$$123_8 = (1 \times 8^3) + (2 \times 8^2) + (3 \times 8^1)$$

- **Hexadecimal**

$$123_{16} = (1 \times 16^3) + (2 \times 16^2) + (3 \times 16^1)$$

here we need 16 characters – 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

- **Number conversion**
  - conversion to decimal
    - add up decimal equivalent of individual digits

Example – see **Example 9.8** in the course text

Convert 11010_2 to decimal

$$110102 = (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$
$$= 16 + 8 + 0 + 2 + 0$$
$$= 2610$$

- **Number conversion**
  - conversion from decimal
    - repeatedly divide by the base and remember the remainder

Example – see **Example 9.9** in the course text

Convert 26_{10} to binary

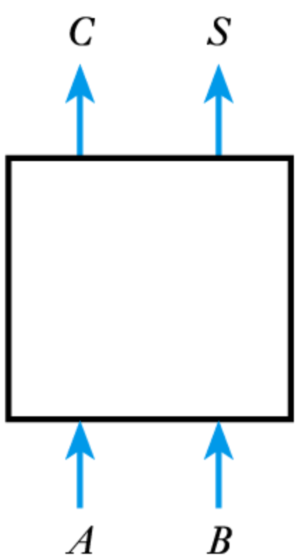
	Number	Remainder
Starting point	26	
÷ 2	13	0
÷ 2	6	1
÷ 2	3	0
÷ 2	1	1
÷ 2	0	1

read number from this

end

=11010

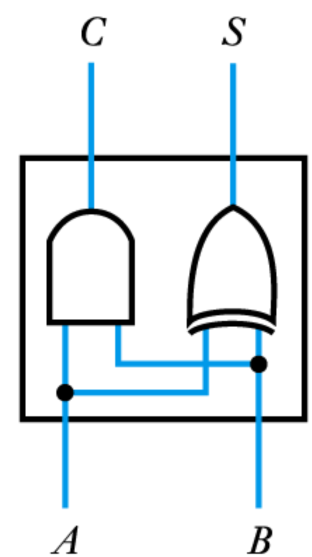
- **Binary arithmetic**
  - much simpler than decimal arithmetic
  - can be performed by simple circuits, e.g. half adder



(a) Block diagram

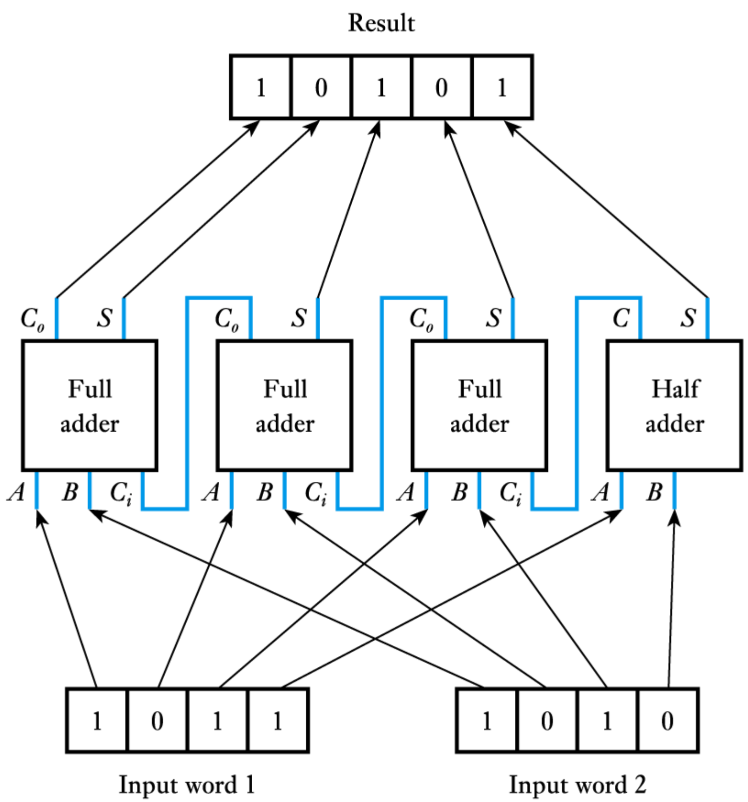
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) Truth table



(c) Circuit diagram

- More complex circuits can add digital words



- Similar circuits can be constructed to perform subtraction – see text
- More complex arithmetic (such as multiplication and division) *can* be done by dedicated hardware but is more often performed using a microcomputer or complex logic device

Numeric and Alphabetic Codes

- **Binary code**
  - by far the most common way of representing numeric information
  - has advantages of simplicity and efficiency of storage

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
etc.	etc.

Numeric and Alphabetic Codes

- **Binary-coded decimal code**
  - formed by converting each digit of a decimal number individually into binary
  - requires more digits than conventional binary
  - has advantage of very easy conversion to/from decimal
  - used where input and output are in decimal form

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	10000
11	10001
12	10010
etc.	etc.

Numeric and Alphabetic Codes

- **ASCII code**
  - **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
  - an **alphanumeric code**
  - each character represented by a 7-bit code
    - gives 128 possible characters
    - codes defined for upper and lower-case alphabetic characters, digits 0 – 9, punctuation marks and various non-printing control characters (such as carriage-return and backspace)

Numeric and Alphabetic Codes

- **Error detecting and correcting codes**
  - adding redundant information into codes allows the *detection* of transmission errors
    - examples include the use of parity bits and checksums
  - adding additional redundancy allows errors to be not only detected but also *corrected*
    - such techniques are used in CDs, mobile phones and computer disks

Key Points

- It is common to represent the two states of a binary variable by ‘0’ and ‘1’
- Logic circuits are usually implemented using logic gates
- Circuits in which the output is determined solely by the current inputs are termed combinational logic circuits
- Logic functions can be described by truth tables or using Boolean algebraic notation
- Binary digits may be combined to form digital words
- Digital words can be processed using binary arithmetic
- Several codes can be used to represent different forms of information