የኢንፎርሜሽን ሳይንስ ት/ቤት
School of Information Science

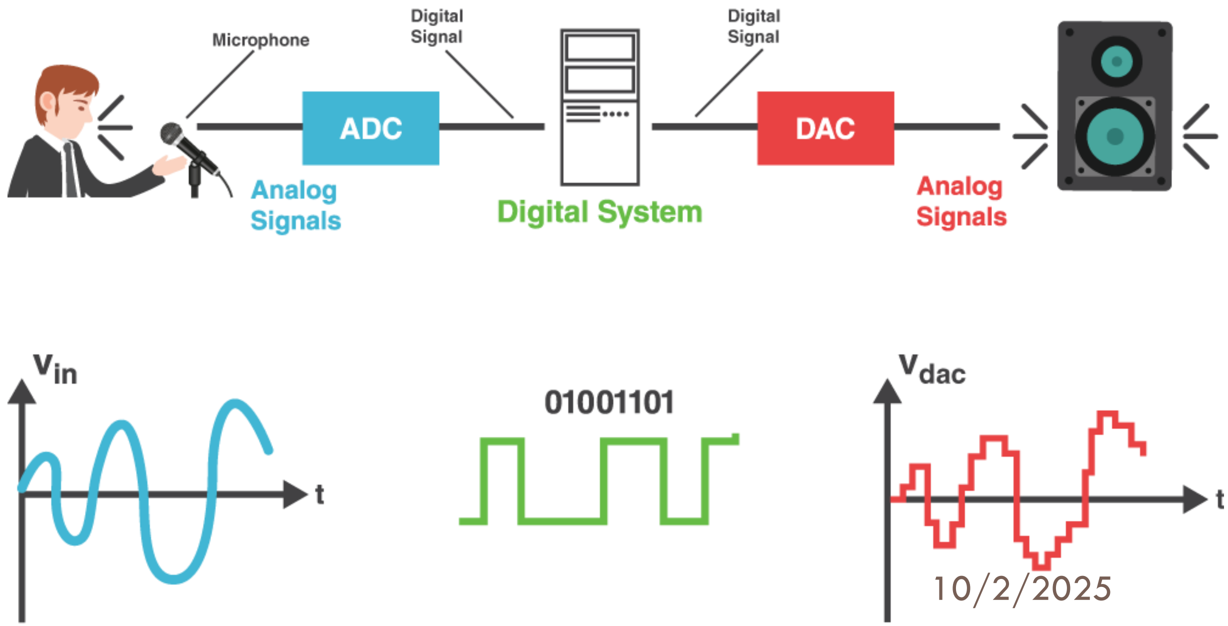
CHAPTER ONE
FOUNDATIONS OF DIGITAL SYSTEMS AND DATA REPRESENTATION

CH-1 Contents

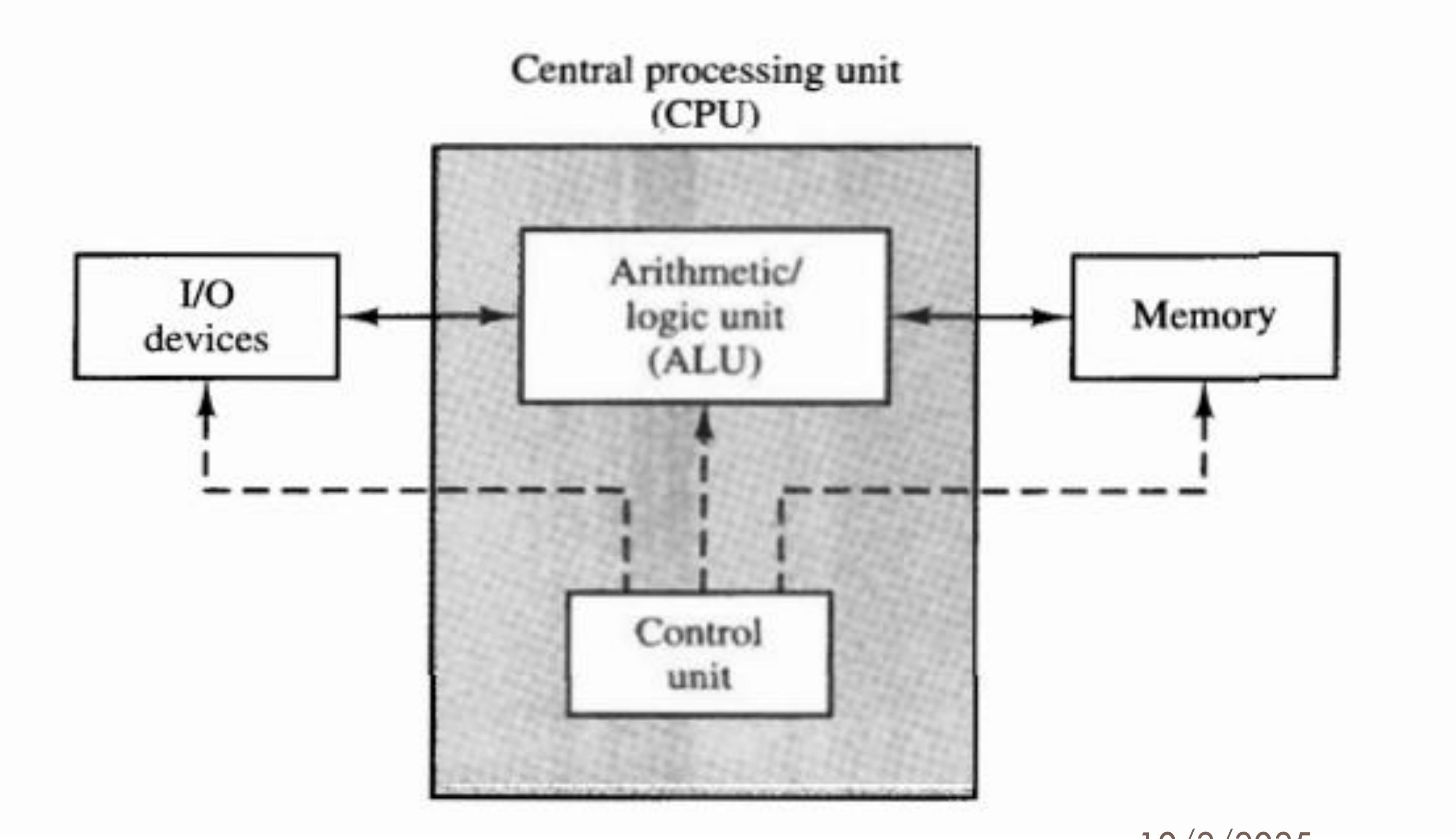
1. Overview of computer organization and architecture
2. Number system and Data representation
3. Logic gates and Boolean Algebra
4. Combinational and Sequential Circuits

Overview of Computer Organization and Architecture

- **Key Terms**
  - Signal
  - Analog Signal vs Digital Signal
  - Analog System vs Digital System

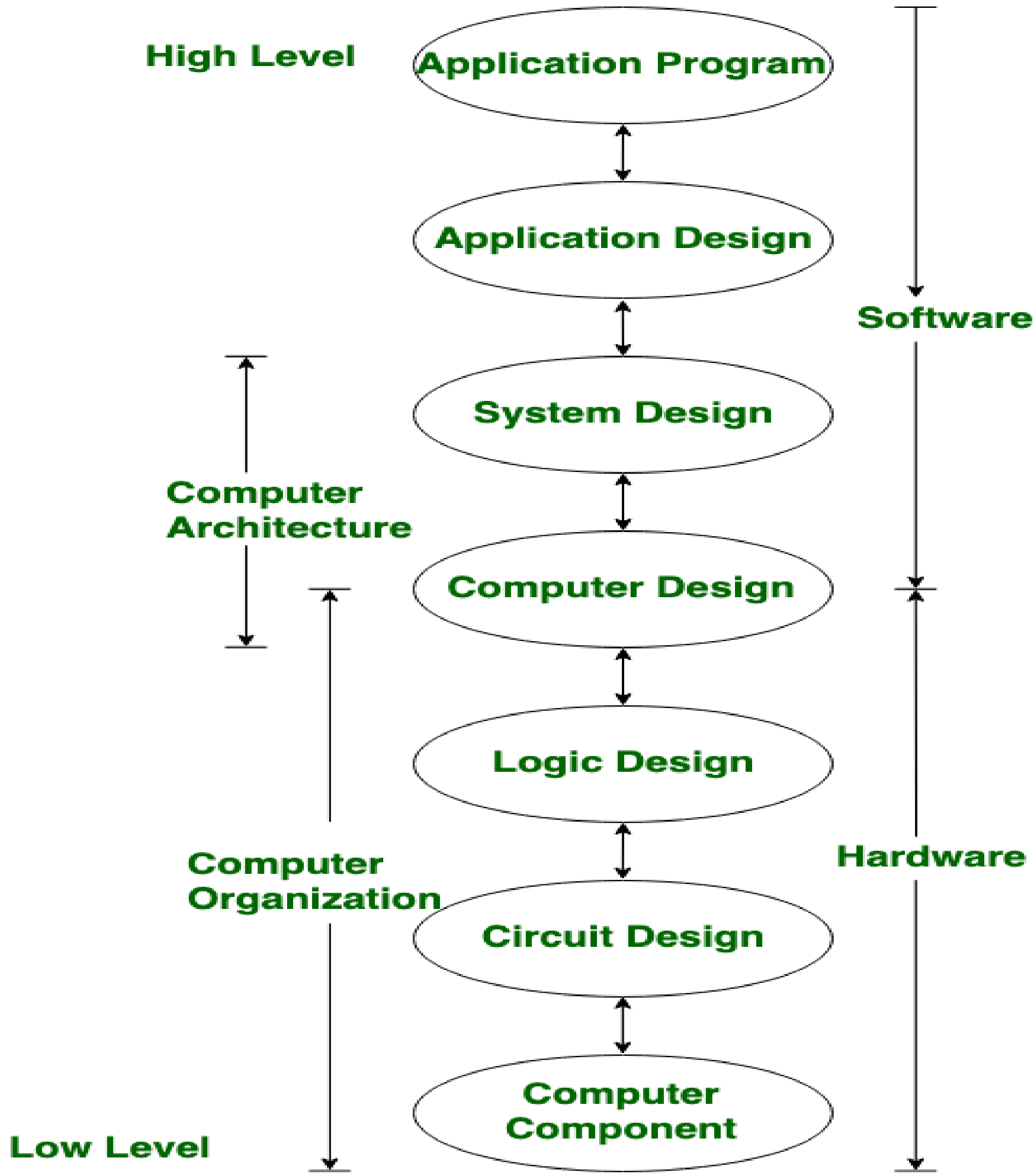


Organization of A Digital Computer



Computer Architecture vs Computer Organization

- **Computer system:** is subdivided into two functional entities: Hardware and Software.
  - **Hardware:** consists of all the electronic components and electromechanical devices that comprise the physical entity of the device.
  - **Software:** consists of the instructions and data that the computer manipulates to perform various data-processing tasks.
    - A sequence of instructions for the computer is called a program. The data that are manipulated by the program constitute the database.



Computer System

- **Computer Architecture**
  - **Logical aspects** of a computer system
  - Focuses on the **structure and behavior** of a computer system
    - It is operational attributes are linked together and to realize the architectural specifications
    - **What the computer does?**
  - **Example**: instruction set and format, the number of bits used to represent different data types, I/O mechanisms, and techniques for addressing memory.

- **Computer Organization**
  - **Physical aspect** of a computer system
  - It concerned with the way the **hardware are connected** together to form the computer system
    - Study on how various circuits and components fit together to create a working computer
    - **How the computer does it?**
  - **Example:** circuit design, control signal, interface between the computer and peripherals, and the memory technology used

- **Why study COA?**
  - **Design** better programs
    - Including system software such as compiler, operating system, and device drivers
  - **Optimize** program behavior
  - **Evaluate** (benchmark) computer system performance
  - **Understand** time, space, and price tradeoffs

Number system and Data representation

- **Computers don't process human language directly** → everything **stored/processed as binary.**
- **Data types** in registers:
  1. **Numbers** (arithmetic)
  2. **Letters** (data processing)
  3. **Symbols** (special purposes)
- All types of data, except binary numbers, are represented in computer registers in **binary coded form.**

- **The Alphanumeric Representation**
  - **Alphanumeric = characters, digits, punctuation.**
  - Three (3) alphanumeric codes are in common use.
    - **ASCII** (American Standard Code for Information Interchange)
    - **Unicode** (Universal code)
    - **EBCDIC** (Extended Binary Coded Decimal Interchange Code).

Character	Binary	Hex
A	100 0001	41
B	100 0010	42
C	100 0011	43
D	100 0100	44
E	100 0101	45
F	100 0110	46
G	100 0111	47
H	100 1000	48
I	100 1001	49
J	100 1010	4A
K	100 1011	4B
Space	010 0000	20
Full stop	010 1110	2E
(	010 1000	28
+	010 1011	2B
$	010 0100	24

- **The Decimal Representation**
  - **BCD (Binary Coded Decimal)** is often used to represent decimal number in binary.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

- **Number Representation in Computers**
  - **Two main types:**
    1. **Integer (Fixed-Point) Representation**
       - Used for integer numbers (positive & negative).
       - **Stored in binary form** with **fixed number of bits.**
       - Simple, fast, but limited range.
    2. **Floating-Point Representation**
       - Used for very large or small numbers and fractions.
       - Stored in form: **Sign | Exponent | Fraction.**
       - Wide range, but **more complex and may lose precision.**

- **Integer (fixed point) number representation**
  - Fixed number: **8-bit, 16-bit, 32-bit or 64-bit.**
  - Besides bit-lengths, there are two representation schemes for integers:
    1. **Unsigned** integers (0 and +VE integers)
    2. **Signed** integers (0 , -VE and +VE integers)
       A. Sign-Magnitude representation
       B. 1's complement representation
       C. 2's complement representation

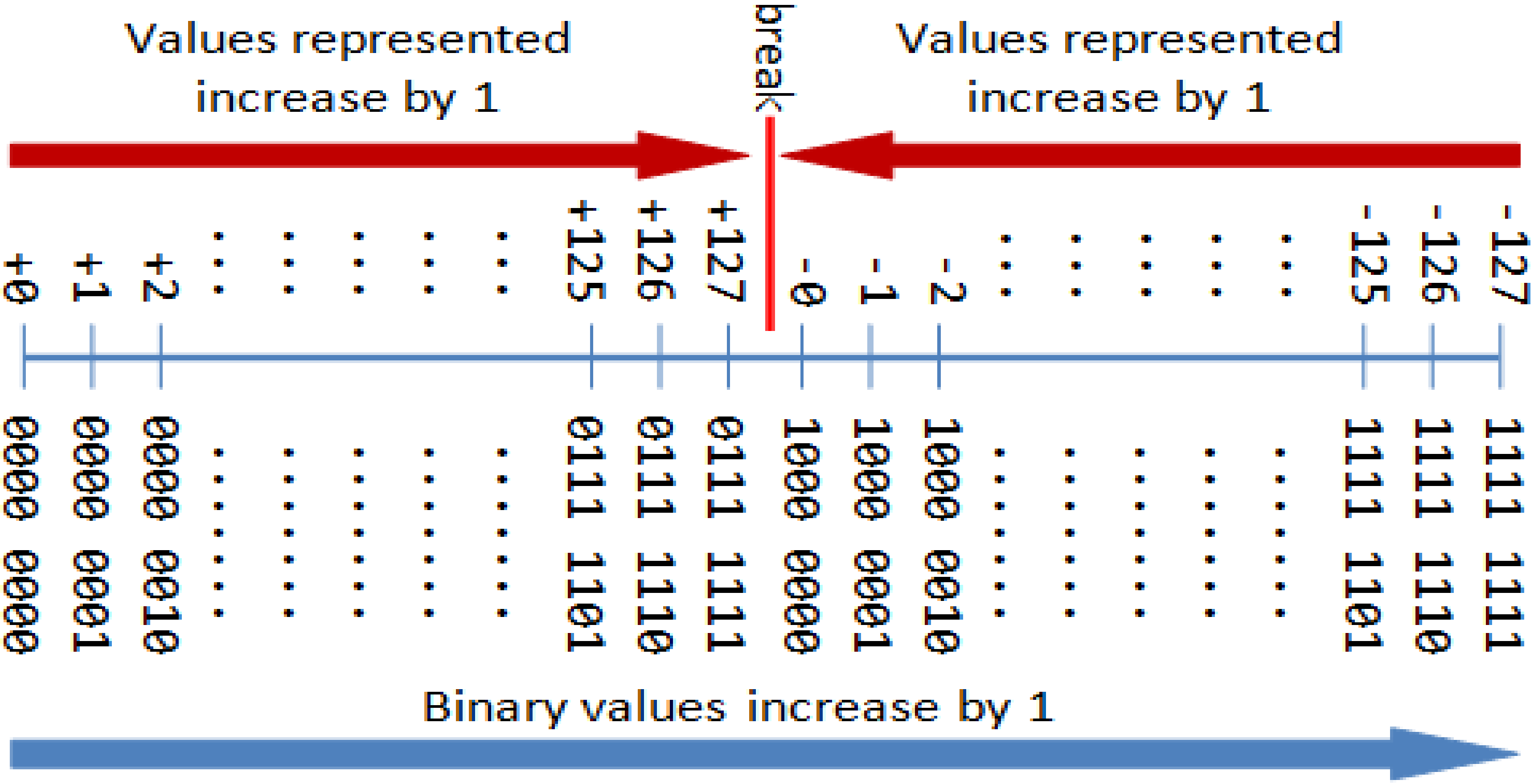
1. **n-bit Unsigned Integers**

- Represent only 0 and positive values.
- Value = binary magnitude.
- Example (8-bit): 0100 0001B = 65D.
- **Exercise (8 bits):**
  1. 0000 1010B = ?D
  2. 1111 1111B = ?D
  3. 0001 0010B = ?D

2. Signed Integers

A. Sign-and-Magnitude

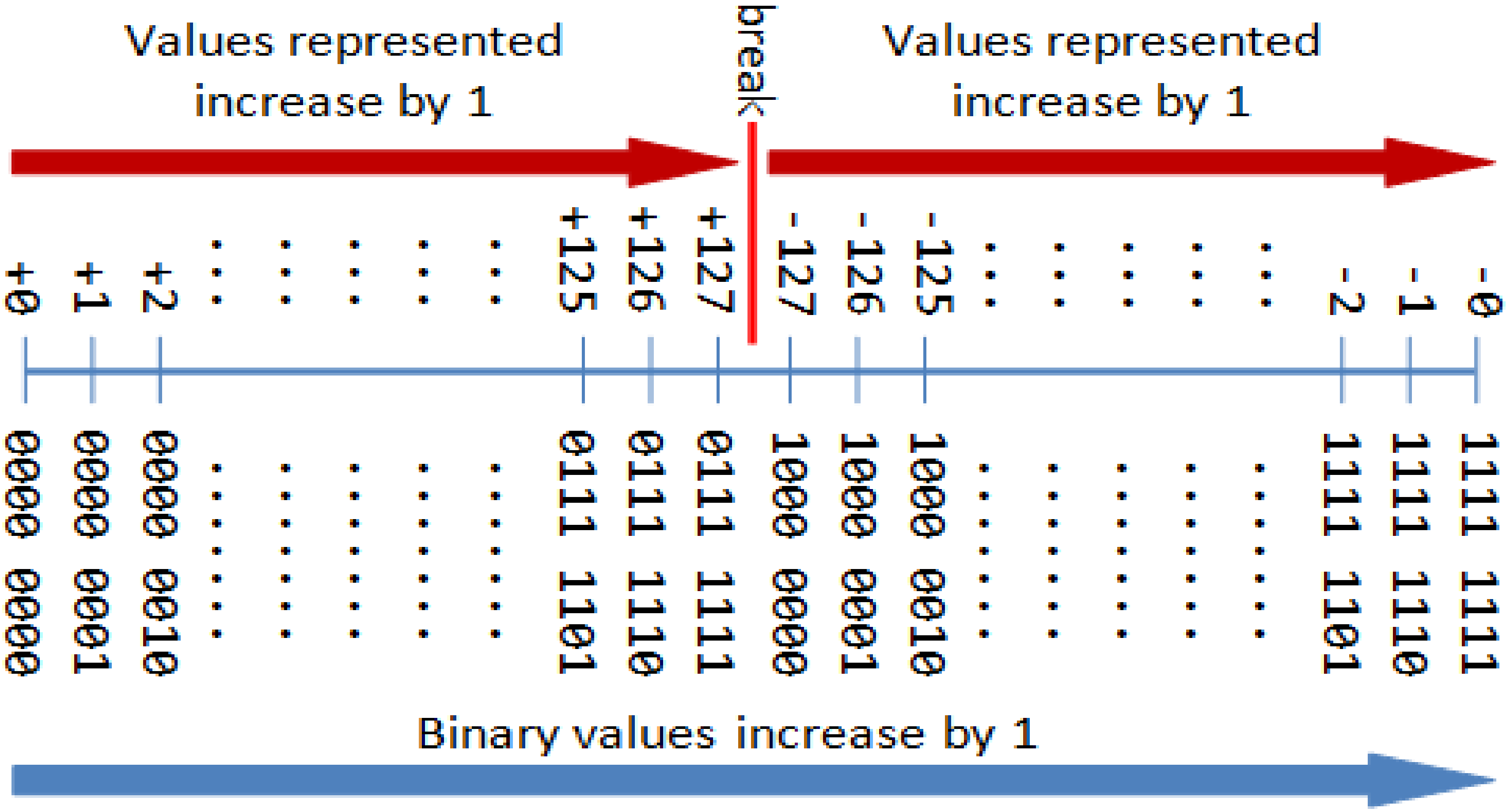
- MSB = sign, rest = magnitude.
  - **Sign bit (MSB): 0 = +, 1 = −**
- **Drawback:** two representations for 0.
- **Examples** (8-bit):
  - 0 1000001B = +65
  - 1 0000001B = −1
- **Exercise (8 bits)**
  1. 0000 0000B = ?D
  2. 1111 1111B = ?D
  3. 1000 0000B = ?D



Sign-Magnitude Representation

B. 1's Complement Representation

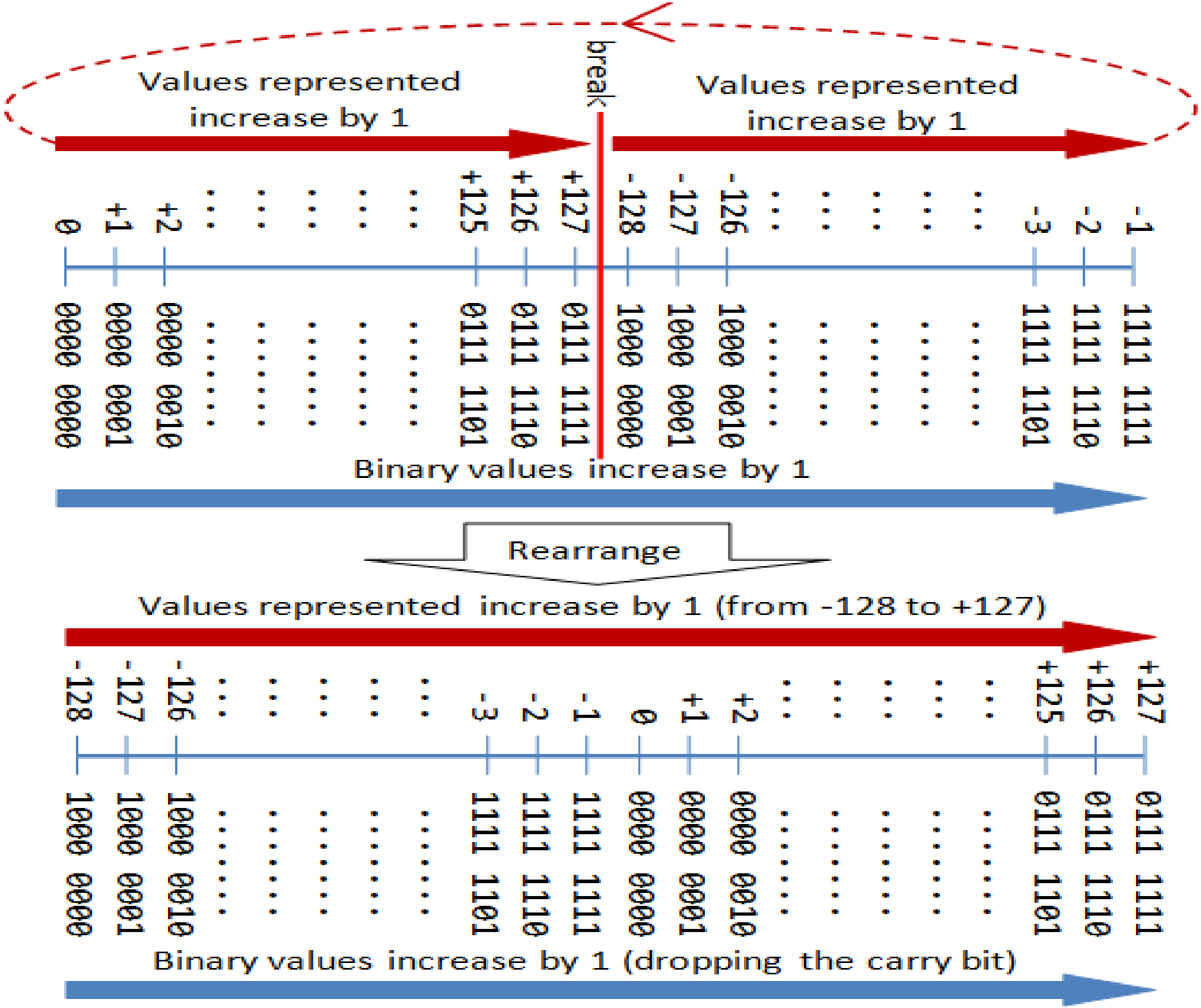
- MSB = sign (0 = +, 1 = −), rest = magnitude.
  - **Positive numbers**: value = magnitude of remaining bits.
  - **Negative numbers**: value = complement (invert) of remaining bits.
- **Drawback**: two representations for 0.
- **Examples** (8-bit):
  - 0 1000001B = +65
  - 1 0000001B = −1
- **Exercise (8 bits)**
  1. 0000 0000B = ?D
  2. 1111 1111B = ?D
  3. 1000 0000B = ?D



1's Complement Representation

c. 2's Complement Representation

- MSB = sign bit (0 = +, 1 = −), rest = magnitude.
  - Positive numbers: value = magnitude of remaining bits.
  - Negative numbers: value = 2's complement (invert bits + 1) of remaining bits.
- **Advantages:**
  - Only one zero
  - Addition/subtraction simpler (same logic).
- **Example:** 1 0000001B = −127.
- **Exercise (8 bits)**
  1. 0000 0000B = ?D
  2. 1111 1111B = ?D



2's Complement Representation

Exercise-1

- Show the actual binary operation
  1. **Addition of Two Positive Integers:** Suppose that n=8.
     - 15D + 25D = 40D
  2. **Subtraction is treated as Addition of a Positive and a Negative Integers:** Suppose that n=8.
     - 15D + (-20D) = -5D
  3. **Addition of Two Negative Integers:** Suppose that n=8.
     - -15D - 20D = (-15D) + (-20D) = -40D

- **Floating point representation**
  - Represents very small(-1.23×10^{88}) or a very large(1.23×10^{-88}) numbers.
  - **Trade-off:**
    - More range, but **loss of precision (not all real numbers can be represented).**
    - **Slower than integer arithmetic** → often **uses floating-point coprocessor.**
  - Form: **F × r^E (fraction × radix^exponent).**
    - Decimal numbers use radix of 10 ($F\times10^E$); while binary numbers use radix of 2 ($F\times2^E$).
  - **Requires normalization.**

- For example, the number 55.66 can be represented as
  - **5.566×10^1**
  - 0.5566×10^2
  - 0.05566×10^3, and so on.
- We need **NORMALIZATION** ….
  - The fractional part can be ***normalized***
  - In the normalized form, there is only ***a single non-zero digit before the radix point***.

IEEE 754 Standard

- Modern computers adopt **IEEE 754** standard for representing floating-point numbers
  - **Formats:**
    1. **32-bit Single Precision**
    2. **64-bit Double Precision**
    3. **128-bit Quad Precision**
- Stored as: **Sign | Exponent | Fraction.**
- Exponent uses **biasing** (offset binary).
  - **Biasing** is done because exponents have to be signed values in order to able **to represent both tiny and huge values**, but two's complement, the usual representation for signed values, would make **comparison** harder.

- **Basic IEEE floating point number**

	Single	Double	Quadruple
No. of **sign** bit	1	1	1
No. of **biased exponent** bit	8	11	15
No. of **fraction** bit	23	52	112
Total bit used	**32**	**64**	**128**
Bias (2^{e-1} -1)	(2^{8-1} -1) 127	(2^{11-1} -1) 1023	(2^{15-1} -1) 16383

- **The IEEE Floating Point Representation**

Convert **153.75** to the **IEEE floating point format.**

- Step 1 : convert into binary
- Step 2 : put into 1.xxxx X 2^y format
- Step 3 : get the biased exponent (identify the exponent first)
- Step 4 : get the sign (from the question)
- Step 5 : identify significand and mantissa
- Step 6 : put into the IEEE single precision format
- Step 7 : convert into hexadecimal

<table>
<tr><td>Convert to binary</td><td colspan="8">153.75 = $1001\ 1001.11_2$</td></tr>
<tr><td>Put into 1.xxxx X 2^y</td><td colspan="8">= $1.0011\ 0011\ 1_2 \times 2^7$</td></tr>
<tr><td>Biased exponent</td><td colspan="8">Exponent = 7
Biased exponent = 127 + y = 127 + 7 = 134
= $\mathbf{1000\ 0110_2}$ (8 bits)</td></tr>
<tr><td>Sign</td><td colspan="8">Sign = +ve (0)</td></tr>
<tr><td>Significand
Mantissa</td><td colspan="8">Significand = 1.0011 0011 1
Mantissa = 001 1001 1100 0000 0000 0000 (23bits)</td></tr>
<tr><td rowspan="4">IEEE format</td><td colspan="8">IEEE format :</td></tr>
<tr><td>Sign</td><td colspan="3">Biased Exponent</td><td colspan="4">Mantissa (23)</td></tr>
<tr><td>0</td><td colspan="3">1000 0110</td><td colspan="4">001 1001 1100 0000 0000 0000</td></tr>
<tr><td colspan="8"></td></tr>
<tr><td rowspan="3">Hexadecimal</td><td>0100</td><td>0011</td><td>0001</td><td>1001</td><td>1100</td><td>0000</td><td>0000</td><td>0000</td></tr>
<tr><td>4</td><td>3</td><td>1</td><td>9</td><td>C</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="8">= 4319 C000 h</td></tr>
</table>

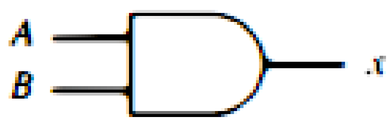
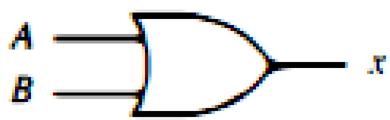
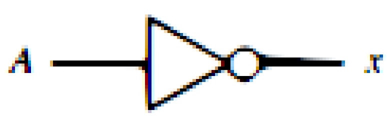
<table>
<tr><td>Convert to binary</td><td colspan="8">153.75 = 1001 1001.11$_2$</td></tr>
<tr><td>Put into 1.xxxx X 2^y</td><td colspan="8">= 1.0011 0011 1$_2$ x 2^7</td></tr>
<tr><td>Biased exponent</td><td colspan="8">Exponent = 7
Biased exponent = 127 + y = 127 + 7 = 134
= 1000 0110$_2$ (8 bits)</td></tr>
<tr><td>Sign</td><td colspan="8">Sign = +ve (0)</td></tr>
<tr><td>Significand
Mantissa</td><td colspan="8">Significand = 1.0011 0011 1
Mantissa = 001 1001 1100 0000 0000 0000 (23bits)</td></tr>
<tr><td rowspan="4">IEEE format</td><td colspan="8">IEEE format :</td></tr>
<tr><td>Sign</td><td colspan="3">Biased Exponent</td><td colspan="4">Mantissa (23)</td></tr>
<tr><td>0</td><td colspan="3">1000 0110</td><td colspan="4">001 1001 1100 0000 0000 0000</td></tr>
<tr><td colspan="8"></td></tr>
<tr><td rowspan="3">Hexadecimal</td><td>0100</td><td>0011</td><td>0001</td><td>1001</td><td>1100</td><td>0000</td><td>0000</td><td>0000</td></tr>
<tr><td>4</td><td>3</td><td>1</td><td>9</td><td>C</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="8">= 4319 C000 h</td></tr>
</table>

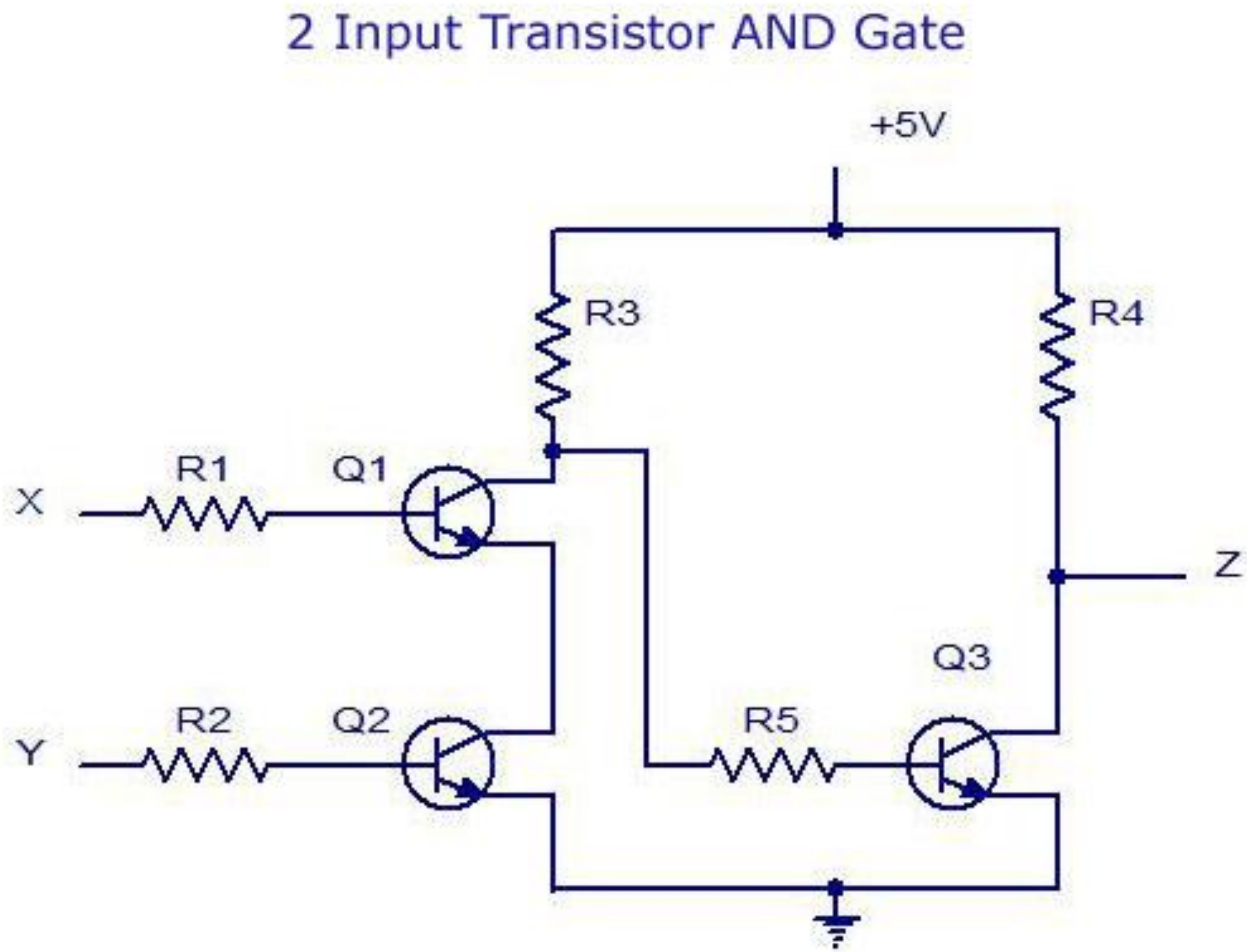
Exercise -2

1. Convert **3.75 to IEEE-754 single** precision.
2. **Decode** 01000001100011001110000000000000 (single) , and **give decimal value.**

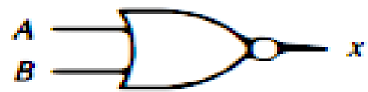
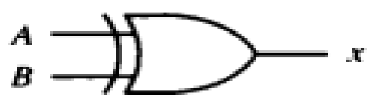
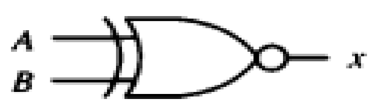
Logic gates and Boolean Algebra

- **Signal: a physical quantity carrying information.**
  - Analog: continuous values
  - Digital: discrete values (e.g., 3V $\rightarrow$ 1, 0.5V $\rightarrow$ 0)
- **Logic Gates: basic building blocks of digital systems.**
  - Circuit with one or more inputs, one output.
  - Output depends on input values (defined by a truth table).
- **Basic Gates:**
  - **AND**
  - **OR**
  - **NOT (inverter)**

Name	Graphic symbol	Algebraic function	Truth table
AND		$x = A \cdot B$ or $x = AB$	A B | x 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$x = A + B$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$x = A'$	A | x 0 | 1 1 | 0


2 Input Transistor AND Gate
+5V
R3
R4
R1
Q1
X
Z
Q3
R2
Q2
R5
Y

- **Combined Gates** *(built from basics for efficiency):*
  - ***NAND, NOR, XOR, XNOR***

NAND		$x = (AB)'$	A B | x 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$x = (A + B)'$	A B | x 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$x = A \oplus B$ or $x = A'B + AB'$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$x = (A \oplus B)'$ or $x = A'B' + AB$	A B | x 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Boolean Algebra

- An algebra that deals with binary variables (0,1) and logic operations.
  - Used to analyze and simplify digital circuits.
  - Also called **Binary Algebra** or **Logical Algebra**.
  - Invented by George Boole (1854).
- **Basic Rules**
  - **Variables**: only two values → 1 (HIGH) and 0 (LOW).
  - **Complement**: written as A' → if A=0 then A'=1, if A=1 then A'=0.
  - **OR**: plus sign → A + B + C.
  - **AND**: dot or no sign → A · B · C or ABC.

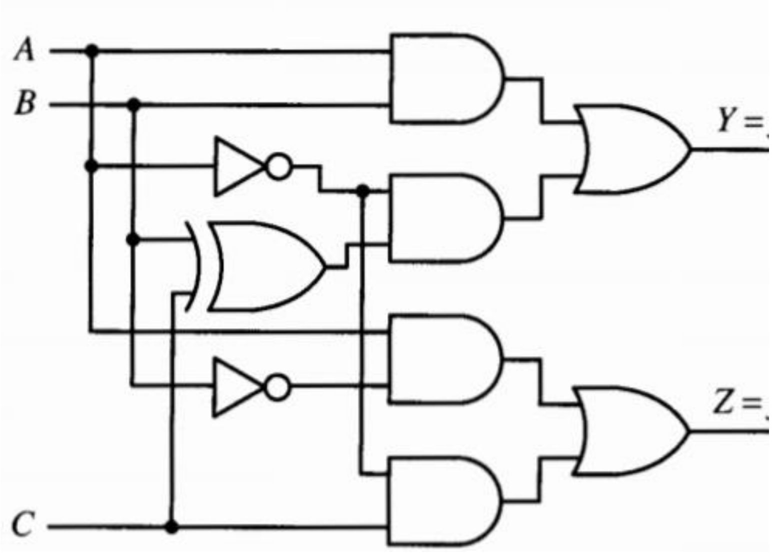
- **Boolean Functions**
  - Expressed with: binary variables, logic symbols, parentheses, equal sign.
  - Example: $F = x + y'z$.
  - Each function can be represented by a **truth table** with 2^n combinations for n variables.
- **Logic Diagrams**
  - Boolean expressions can be implemented with AND, OR, and NOT gates.
  - **Purpose:**
    - Express truth table relations in algebraic form.
    - Describe logic diagrams algebraically.
    - Simplify circuits to use fewer gates.

- **Example**
  - F= x+y'z
    - a) Represent the above function using truth table
    - b) Draw the logical diagram

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Exercise-3

1. $F = (B+C)+(AB)'(A' + C)'$
   a) Represent the above function using truth table
   b) Draw the logical diagram
2. Write the algebraic function?



Boolean Laws

- **AND Laws**
  1. $A.0 = 0$
  2. $A.A = A$
  3. $A.1 = A$
  4. $A.A' = 0$
- **OR Laws**
  5. $A + 0 = A$
  6. $A + A = A$
  7. $A + 1 = 1$
  8. $A + A' = 1$
- **Commutative Laws**
  9. $A.B = B.A$
  10. $A + B = B + A$

  (Order of variables doesn't affect output)
- **Associative Law**
  11. $(A.B).C = A.(B.C)$
  12. $(A + B) + C = A + (B + C)$

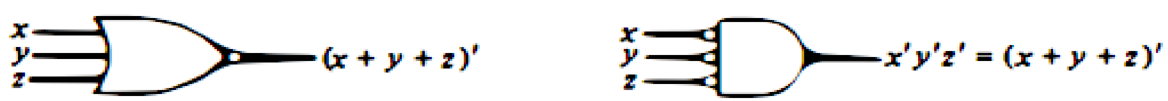
  (Order of grouping doesn't matter)
- **Distributive Law**
  13. $A.(B + C) = A.B + A.C$
  14. $A+(B . C) = (A+B) . (A+C)$
- **Inversion Law**
  15. $A'' = A$

  (double negation gives original)
- **DeMorgan's Theorem**
  16. $(A.B)' = A' + B'$
  17. $(A + B)' = A'.B'$

  Complement rule:
  - Swap AND ↔ OR
  - Complement each variable

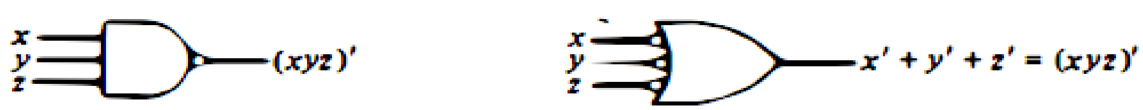
- **DeMorgan's Theorem Example**
  - F= AB + C'D' + B'D
  - F'=(A' + B')(C + D)(B + D')
- NOR and NAND

Two graphic symbols for NOR gate.



(a) OR-invert (b) invert-AND

Two graphic symbols for NAND gate.



(a) AND-invert (b) invert-OR

Exercise-4:
Simplify using Boolean Algebra

1. $(X+Y) + (X+Y)Z$
2. $AB'(AB' + B'C)$
3. $AB'C + B'$
4. $B+AB'C'D$
5. $Y'(X+Y+Z)$
6. $(X+Y)((X+Y)'+Z)$
7. $AB + (AB)'CD'$
8. $ABC + AB'C$
9. $(AD+B+C)(AD+(B+C)')$

Simplifying Boolean functions
Reading Assignment

1. Minimization by Boolean Algebra
    1. Grouping
    2. Multiplication by redundant variables
    3. Application of DeMorgan’s Theorem
2. Minimization by Karnaugh Maps (K-Map)

Combinational and Sequential Circuits

- **Circuit Design:** The process of creating circuits that optimize speed, portability, memory use, and overall functionality.
  - **Circuit Design Goal**: Efficient, reliable, and space-optimized circuits.
- **Two Main Types:**
  1. **Combinational Circuits:** Output depends only on current inputs.
  2. **Sequential Circuits:** Output depends on current input and past output (requires memory).
  - **Practical circuits often combine both types.**

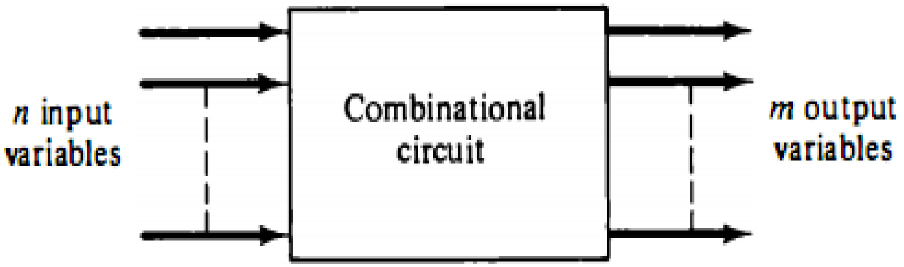
Combinational Circuits

Characteristics:

- Can have multiple inputs/outputs.
- Output depends only on current inputs.
- No memory

Examples:

- Arithmetic & Logic units
- Data transmission circuits
- Code converters



Block diagram of a combinational circuit.

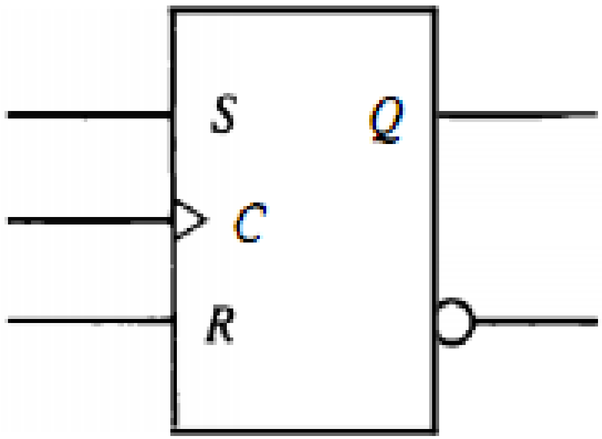
- **Design Steps: Combinational circuits**
  - The design of combinational circuits starts from the verbal outline of the problem and ends in a logic circuit diagram. The procedure involves the following steps:
    1. **Understand problem & assign input/output symbols**
    2. **Construct truth table**
    3. **Simplify Boolean functions**
    4. **Draw logic diagram**

Exercise-5

1. Design and implement a **Half Adder** circuit that performs the addition of two single-bit binary numbers.
2. Design and implement a **Full Adder** circuit that adds three single-bit binary numbers, including a carry from a previous addition
3. Design a **2-bit digital comparator** that compares two 2-bit binary numbers (A and B) and outputs whether $A > B$, $A < B$, or $A = B$.

- **Sequential Circuits**
  - **Characteristics:**
    - Output depends on current inputs + past states
    - Requires memory elements
  - **Memory Elements:**
    - Store data (1 bit per element)
    - **State** = snapshot of stored data
    - **Bi-stable:** 2 stable internal states
    - **Implemented using:**
      - **Latches:** No clock
      - **Flip-Flops:** Clocked storage, types: SR, D, JK, T

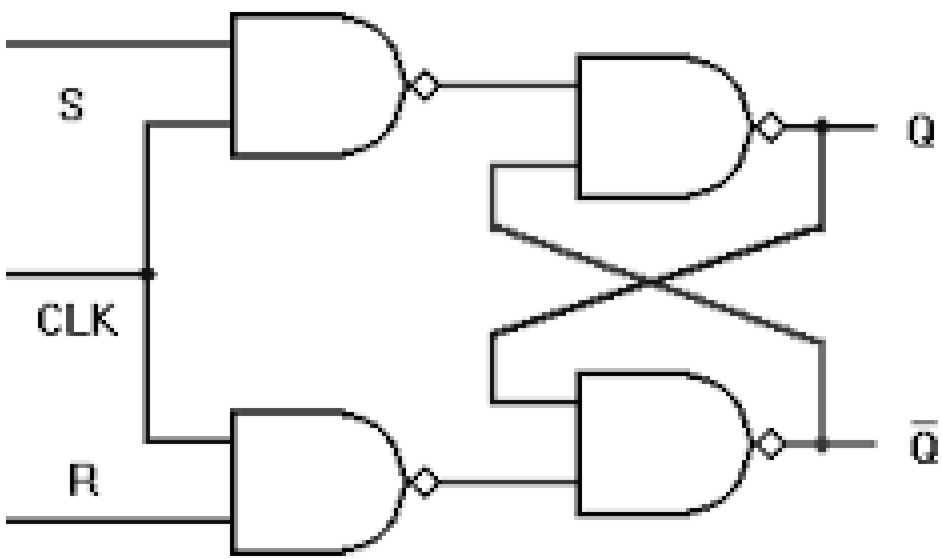
- **Flip-Flops**
  - Storage elements in **clocked sequential circuits**
  - Store **1 bit of data** with **normal** & **complement** outputs
  - **Edge-triggered** → state changes only on **a clock pulse**
  - **More reliable & widely used**
  - **Types**
    - SR Flip-Flop
    - D Flip-Flop
    - JK Flip-Flop
    - T Flip-Flop

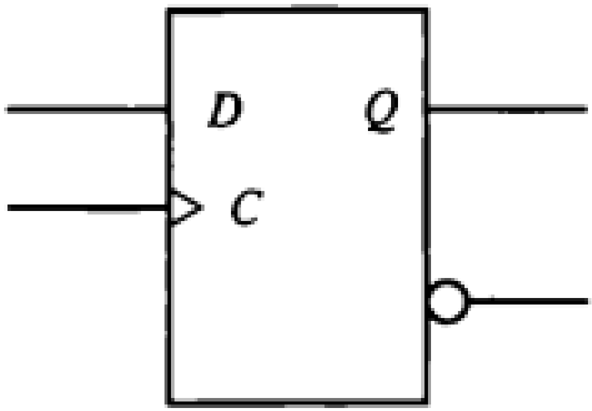


(a) Graphic symbol

S	R	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	?	Indeterminate

(b) Characteristic table

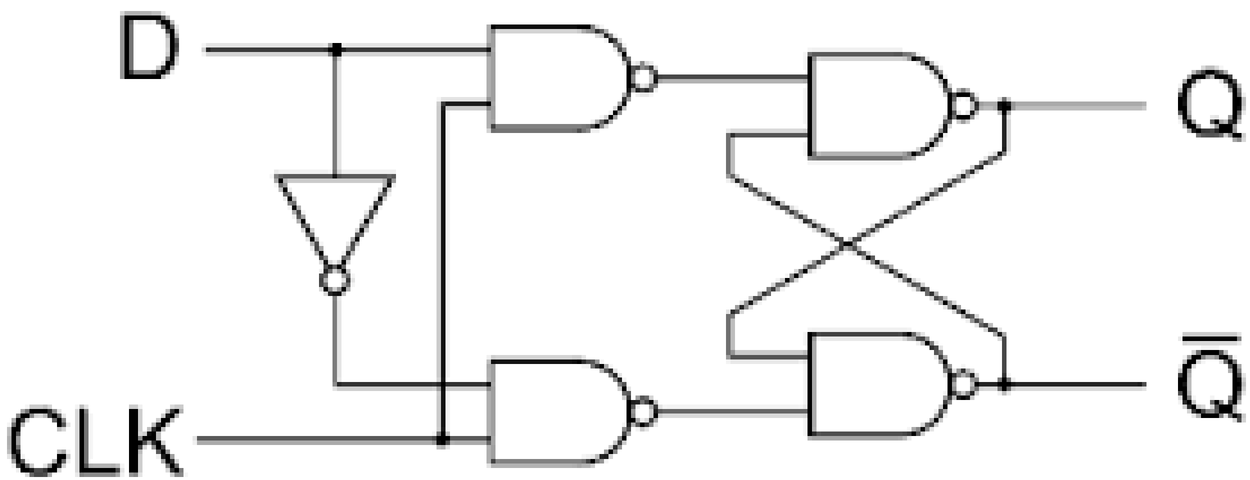


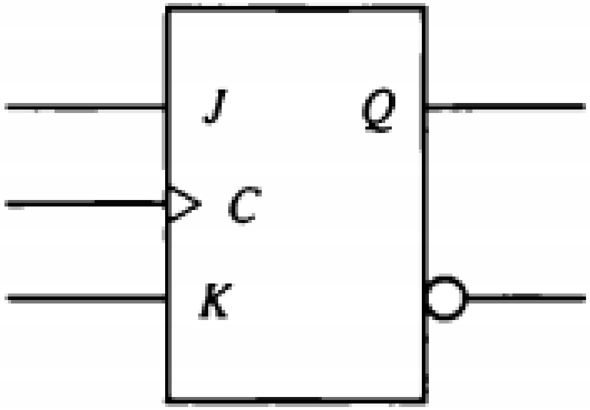


(a) Graphic symbol

D	$Q(t+1)$	
0	0	Clear to 0
1	1	Set to 1

(b) Characteristic table

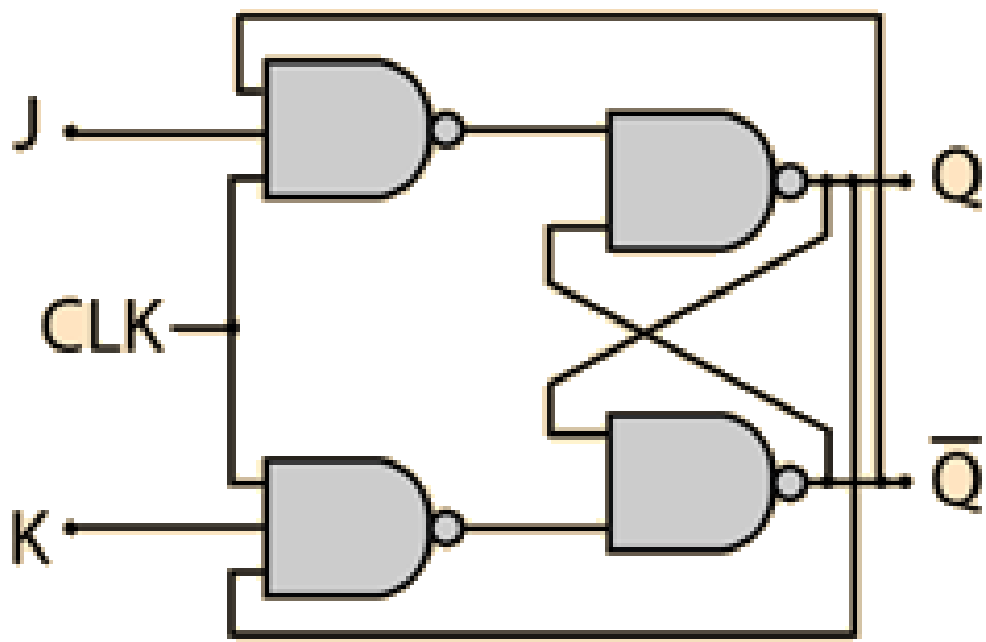


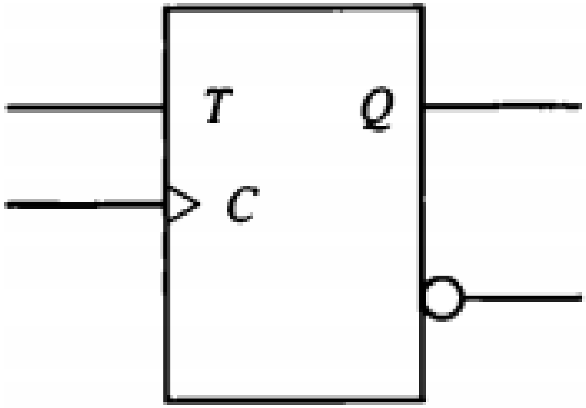


(a) Graphic symbol

J	K	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	$Q'(t)$	Complement

(b) Characteristic table

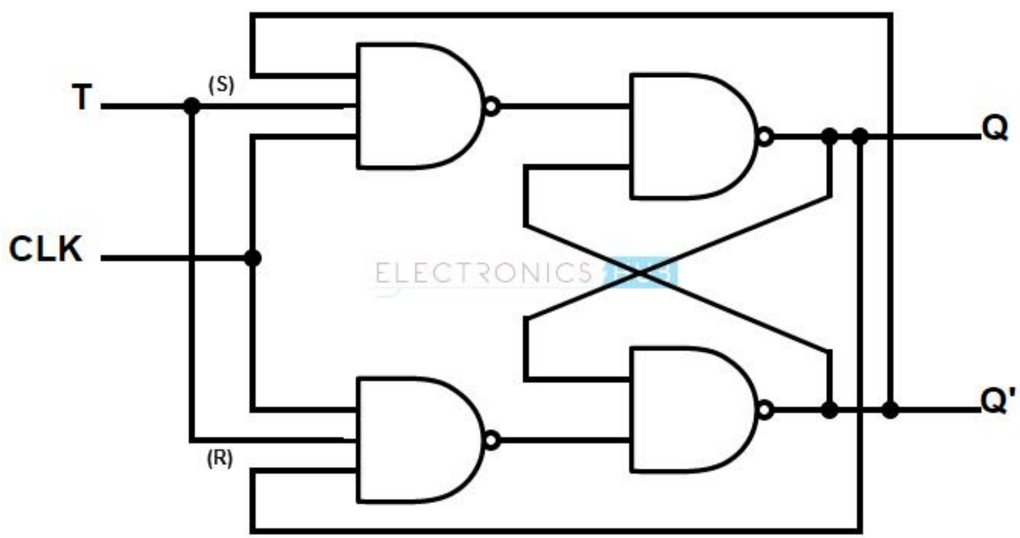




(a) Graphic symbol

T	$Q(t+1)$	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

(b) Characteristic table



- **Design Steps: Sequential circuits**
  1. Understand the circuit and write circuit specification
  2. Translating the circuit specifications into a **state diagram.**
  3. Convert the state diagram into a **state table.**
  4. Extend the state table into an **excitation table**
  5. Find the set of flip-flop input equation for the combinational circuit
  6. Draw the logic diagram
     - Read more: https://bob.cs.sonoma.edu/IntroCompOrg-RPi/sec-seqdes.html

Questions?

Thank You