Chapter 2 – Data Representation and Boolean Logic

Lecture 1

Number Systems

Class XI – Computer Science

By Mohammad Haris

Learning Objectives

By the end of today's lecture, you will be able to:

- Understand why computers use number systems
- Define a number system
- Understand the concept of base/radix
- Identify Decimal, Binary, Octal and Hexadecimal number systems
- Identify the valid digits of each number system
- Understand positional notation

Think About It 🤔

How does a computer represent numbers?

Can a computer directly understand:

25

101

A5

?

Let's find out!

What is a Number System?

A number system is a method of representing numbers using a specific set of digits/symbols and a base.

Base / Radix

The base of a number system tells us the number of`m

digits/symbols used in that system.

Common Number Systems

Number System	Base	Digits Used
Decimal	10	0–9
Binary	2	0–1
Octal	8	0–7
Hexadecimal	16	0–9, A–F

1. Decimal Number System

Base = 10

Digits:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

This is the number system commonly used in our everyday calculations.

2. Binary Number System

Base = 2

Digits:

0 and 1

The digits 0 and 1 are called binary digits.

Binary numbers are fundamental to the way computers represent information.

3. Octal Number System

Base = 8

Digits:

0, 1, 2, 3, 4, 5, 6, 7

4. Hexadecimal Number System

Base = 16

Digits:

0–9 and A–F

A = 10 B = 11 C = 12 D = 13 E = 14 F = 15

Quick Check 🧠

Identify the number system:

1. $(101101)_2$: Binary
2. $(725)_8$: Octal
3. $(39)_{10}$: Decimal
4. $(2A)_{16}$: Hexa decimal

Challenge: Valid or Invalid?

Identify whether each number is valid in the given number system.

1. $(10101)_2$: Valid
2. $(10201)_2$: Invalid
3. $(765)_8$: Valid
4. $(789)_8$: Invalid
5. $(198)_{10}$: Valid
6. $(1AF)_{16}$: Valid
7. $(1G)_{16}$: Invalid

Positional Notation

The value of a digit depends on:

1. The digit itself
2. Its position
3. The base of the number system

Example

Consider:

572

The place values are:

5×10^2

7×10^1

2×10^0

$= 5 \times 100$

$+ 7 \times 10$

$+ 2 \times 1$

$= 500 + 70 + 2$

$= 572$

Binary Positional Notation

Consider:

$(1011)_2$

Positions from right to left:

$2^0\ 2^1\ 2^2\ 2^3$

1×2^3

0×2^2

1×2^1

1×2^0

$= 8 + 0 + 2 + 1$

$= 11$

$(1011)_2 = (11)_{10}$

General Positional Representation

For a number in base b:

Digit × b^position

Positions are counted from right to left, starting from 0.

Example $(1011)_2$

$= 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$

Let's Try Together

Convert $(11001)_2$ into decimal.

Step 1: Write the positional powers.

1×2^4

1×2^3

0×2^2

0×2^1

1×2^0

Step 2: Calculate.

16 + 8 + 0 + 0 + 1

= 25

Therefore:

$(11001)_2 = (25)_{10}$

Practice – Try Yourself ✍️

Convert the following binary numbers into decimal:

1. $(101)_2$
2. $(110)_2$
3. $(1001)_2$
4. $(1010)_2$
5. $(1111)_2$

Common Mistakes ⚠️

❌ Starting the power from 1

Correct:

Rightmost digit → 2^0

❌ Reading binary digits as decimal digits

$1011_2 \neq 1011_{10}$

❌ Using invalid digits

1021_2 is NOT a valid binary number

Quick Quiz 🎯

Q1. What is the base of the binary number system?

Q2. How many digits are used in hexadecimal?

Q3. Is 8 a valid octal digit?

Q4. What is the rightmost positional power?

Q5. Convert $(1011)_2$ into decimal.

Q6. Is $(1021)_2$ valid?

Important Notes

1. A number system is a method of representing numbers using a specific set of digits and a base.
2. Decimal → Base 10 → Digits 0–9
3. Binary → Base 2 → Digits 0 and 1
4. Octal → Base 8 → Digits 0–7
5. Hexadecimal → Base 16 → Digits 0–9 and A–F
6. The value of a digit depends on its position and the base of the number system.
7. The rightmost position has power 0.

Homework 📝

A. Theory

1. Define a number system.
2. What is meant by base/radix?
3. Write the base and digits used in: a) Decimal b) Binary c) Octal d) Hexadecimal

B. Identify Valid/Invalid Numbers

1. $(10101)_2$
2. $(10201)_2$
3. $(765)_8$
4. $(789)_8$
5. $(2AF)_{16}$

C. Convert to Decimal

1. $(1011)_2$
2. $(11001)_2$
3. $(11101)_2$
4. $(100101)_2$

Chapter 2 – Data Representation and Boolean Logic

Lecture 2

Number System Conversions – Part 1

Class XI – Computer Science

Quick Recap

Last lecture we learned:

- Decimal → Base 10
- Binary → Base 2
- Octal → Base 8
- Hexadecimal → Base 16

We also learned:

Rightmost position = 0

and

Digit × Base^Position

Today's Goal 🎯

Today we will learn how to convert:

Decimal → Binary

Decimal → Octal

Decimal → Hexadecimal

Decimal to Another Base

To convert a decimal integer into another base:

1. Divide the number by the new base.
2. Record the remainder.
3. Divide the quotient again by the same base.
4. Continue until the quotient becomes 0.
5. Read the remainders from bottom to top.

Decimal → Binary

Since binary has base 2,

we repeatedly divide the decimal number by 2.

Example 1

Convert $(25)_{10}$ to binary.

25 ÷ 2 = 12 remainder 1

12 ÷ 2 = 6 remainder 0

6 ÷ 2 = 3 remainder 0

3 ÷ 2 = 1 remainder 1

1 ÷ 2 = 0 remainder 1

Now Read the Remainders

We got:

1

0

0

1

1

But we read them from BOTTOM → TOP

Therefore:

11001

Final Answer

$(25)_{10} = (11001)_2$

Verification

We obtained:

$(25)_{10} = (11001)_2$

Let's verify.

$(11001)_2$

$= 1\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0$

$= 16 + 8 + 0 + 0 + 1$

$= 25$

✓ Correct!

Example 2

Convert $(37)_{10}$ to binary.

37 ÷ 2 = 18 R1

18 ÷ 2 = 9 R0

9 ÷ 2 = 4 R1

4 ÷ 2 = 2 R0

2 ÷ 2 = 1 R0

1 ÷ 2 = 0 R1

Read bottom → top:

100101

Therefore:

$(37)_{10} = (100101)_2$

Practice 1 ✍️

Convert the following decimal numbers into binary:

1. $(10)_{10}$
2. $(15)_{10}$
3. $(20)_{10}$
4. $(32)_{10}$
5. $(45)_{10}$

Decimal → Octal

Octal has base 8.

Therefore:

Repeatedly divide the decimal number by 8.

Example

Convert $(125)_{10}$ to octal.

125 ÷ 8 = 15 remainder 5

15 ÷ 8 = 1 remainder 7

1 ÷ 8 = 0 remainder 1

Read bottom → top:

175

Therefore:

$(125)_{10} = (175)_8$

Verification

$(175)_8$

$= 1\times8^2 + 7\times8^1 + 5\times8^0$

= 64 + 56 + 5

= 125

✓ Correct

Practice 2 ✍️

Convert decimal to octal:

1. $(20)_{10}$
2. $(35)_{10}$
3. $(64)_{10}$
4. $(100)_{10}$
5. $(150)_{10}$

Answers

1. 24_8
2. 43_8
3. 100_8
4. 144_8
5. 226_8

Decimal → Hexadecimal

Hexadecimal has base 16.

Digits:

0 1 2 3 4 5 6 7 8 9 A B C D E F

Hexadecimal Values

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4

Decimal	Hexadecimal
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Example

Convert $(254)_{10}$ to hexadecimal.

254 ÷ 16 = 15 remainder 14

15 ÷ 16 = 0 remainder 15

Now convert the remainders:

14 = E , 15 = F

Read bottom → top:

FE

Therefore:

$(254)_{10} = (FE)_{16}$

Example 2

Convert $(100)_{10}$ to hexadecimal.

Solution:

100 ÷ 16 = 6 remainder 4

6 ÷ 16 = 0 remainder 6

Read bottom → top:

64

Therefore:

$(100)_{10} = (64)_{16}$

Practice 3 ✍️

Convert decimal to hexadecimal:

1. $(10)_{10}$
2. $(15)_{10}$
3. $(16)_{10}$
4. $(26)_{10}$
5. $(50)_{10}$
6. $(255)_{10}$

Answers

1. A_{16}
2. F_{16}
3. 10_{16}
4. $1A_{16}$
5. 32_{16}
6. FF_{16}

Decimal → Other Base

Conversion	Divide Repeatedly By
Decimal → Binary	2
Decimal → Octal	8
Decimal → Hexadecimal	16

⭐ Memory Trick

Divide by the BASE you are converting TO.

For example:

Decimal → Binary → divide by 2

Decimal → Octal → divide by 8

Decimal → Hexadecimal → divide by 16

Common Mistakes ⚠️

❌ Reading remainders from top to bottom

✓ Read bottom to top

❌ Dividing by the original decimal base

✓ Divide by the target base

❌ Forgetting A–F in hexadecimal

✓ 10=A, 11=B, 12=C, 13=D, 14=E, 15=F

❌ Stopping before quotient becomes 0

✓ Continue until quotient = 0

Challenge Round 🔥

Convert:

1. $(18)_{10}$ → Binary
2. $(27)_{10}$ → Binary
3. $(75)_{10}$ → Octal
4. $(100)_{10}$ → Octal
5. $(31)_{10}$ → Hexadecimal
6. $(200)_{10}$ → Hexadecimal

Answers

1. 10010_2
2. 11011_2
3. 113_8
4. 144_8
5. $1F_{16}$
6. $C8_{16}$

Quick Quiz 🎯

Q1. To convert decimal to binary, which number do we divide by?

Q2. To convert decimal to octal, which number do we divide by?

Q3. What is decimal 10 in hexadecimal?

Q4. What is decimal 15 in hexadecimal?

Q5. In which direction do we read the remainders?

Q6. Convert $(13)_{10}$ to binary.

Answers

1. 2
2. 8
3. A
4. F
5. Bottom to top
6. 1101_2

Important Notes

Decimal to Other Base

Use the repeated division method.

1. Divide the decimal number by the required base.
2. Write the remainder.
3. Divide the quotient again by the same base.
4. Continue until quotient becomes 0.
5. Read the remainders from bottom to top.

Decimal → Binary → Divide by 2

Decimal → Octal → Divide by 8

Decimal → Hexadecimal → Divide by 16

Hexadecimal:

10 = A

11 = B

12 = C

13 = D

14 = E

15 = F

Homework 📝

Convert the following:

Decimal → Binary

1. $(12)_{10}$
2. $(25)_{10}$
3. $(40)_{10}$
4. $(55)_{10}$

Decimal → Octal

5. $(18)_{10}$
6. $(50)_{10}$
7. $(100)_{10}$

Decimal → Hexadecimal

8. $(20)_{10}$
9. $(45)_{10}$
10. $(125)_{10}$

Show all steps.

📘 Chapter 2: Data Representation and Boolean Logic

Lecture 3 — Other Base to Decimal Number System

🎯 Learning Objectives

By the end of this lecture, you will be able to:

- Understand how to convert a number from **Binary to Decimal**.
- Convert **Octal to Decimal**.
- Convert **Hexadecimal to Decimal**.
- Use the **positional value method** for conversions.
- Verify decimal conversion answers.

---

1. 🔢 Positional Number System

In a positional number system, the **value of a digit depends on its position** in the number.

The general formula is:

Value = Digit × Base^Position

For a number with several digits:

Decimal Value = $d_n \times Base^n + d_{n-1} \times Base^{n-1} + \ldots + d_1 \times Base^1 + d_0 \times Base^0$

Important Bases

Number System	Base	Digits Used
Binary	2	0, 1
Octal	8	0–7
Decimal	10	0–9
Hexadecimal	16	0–9, A–F

> **Remember:** When converting **any number system to decimal**, multiply each digit by the corresponding power of its base and add the results.

---

2. 💻 Binary to Decimal Conversion

Binary has **base 2**, so the positional values are powers of 2.

From right to left:

$2^0, 2^1, 2^2, 2^3, 2^4, \ldots$

Example 1: Convert $(1011)_2$ to Decimal

Write the positional values:

Binary Digit	1	0	1	1
Power of 2	2^3	2^2	2^1	2^0
Value	8	4	2	1

Now multiply and add:

$(1011)_2$

$= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$= 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$

$= 8 + 0 + 2 + 1$

$= \mathbf{11}$

Therefore:

$\mathbf{(1011)_2 = (11)_{10}}$

Example 2: Convert $(110101)_2$ to Decimal

$= 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 32 + 16 + 0 + 4 + 0 + 1$

$= \mathbf{53}$

Therefore:

$\mathbf{(110101)_2 = (53)_{10}}$

⭐ Quick Tip

For binary numbers, remember:

... 32, 16, 8, 4, 2, 1

These are the powers of 2 from right to left.

---

3. 🧮 Octal to Decimal Conversion

Octal has **base 8**, so its positional values are powers of 8.

From right to left:

$\mathbf{8^0, 8^1, 8^2, 8^3, ...}$

Example 1: Convert $(157)_8$ to Decimal

$\mathbf{(157)_8}$

$= 1 \times 8^2 + 5 \times 8^1 + 7 \times 8^0$

$= 1 \times 64 + 5 \times 8 + 7 \times 1$

= 64 + 40 + 7

= **111**

Therefore:

$(157)_8 = (111)_{10}$

Example 2: Convert $(246)_8$ to Decimal

$= 2 \times 8^2 + 4 \times 8^1 + 6 \times 8^0$

$= 2 \times 64 + 4 \times 8 + 6$

$= 128 + 32 + 6$

= **166**

Therefore:

$(246)_8 = (166)_{10}$

⭐ Quick Tip

For octal numbers, remember the powers:

1, 8, 64, 512, 4096, ...

Starting from the **rightmost digit**, use:

$8^0, 8^1, 8^2, 8^3$, ...

---

4. 🔠 Hexadecimal to Decimal Conversion

Hexadecimal has **base 16**.

It uses:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

The letters represent:

Hexadecimal	Decimal
A	10
B	11
C	12

Hexadecimal	Decimal
D	13
E	14
F	15

The positional values are:

16^0, 16^1, 16^2, 16^3, ...

Example 1: Convert $(2A)_{16}$ to Decimal

First convert **A = 10**.

$(2A)_{16}$

$= 2 \times 16^1 + 10 \times 16^0$

$= 2 \times 16 + 10 \times 1$

$= 32 + 10$

$=$ **42**

Therefore:

$(2A)_{16} = (42)_{10}$

Example 2: Convert $(3F)_{16}$ to Decimal

Here, **F = 15**.

$= 3 \times 16^1 + 15 \times 16^0$

$= 3 \times 16 + 15$

$= 48 + 15$

$=$ **63**

Therefore:

$(3F)_{16} = (63)_{10}$

Example 3: Convert $(1B4)_{16}$ to Decimal

Here, **B = 11**.

$= 1 \times 16^2 + 11 \times 16^1 + 4 \times 16^0$

= 1 × 256 + 11 × 16 + 4

= 256 + 176 + 4

= **436**

Therefore:

$(1B4)_{16} = (436)_{10}$

---

5. 📌 General Method

To convert **any number from another base to decimal**:

Step 1

Write the number.

Step 2

Starting from the rightmost digit, assign powers:

$Base^0$, $Base^1$, $Base^2$, $Base^3$, ...

Step 3

Multiply every digit by its corresponding power of the base.

Step 4

Add all the values.

Step 5

The final sum is the **decimal equivalent**.

Formula

Decimal Value = Σ (Digit × Base^Position)

---

6. 🖍️ Solved Mixed Examples

Example 1

Convert $\mathbf{(101101)_2}$ to decimal.

$= 1\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$

$= 32 + 0 + 8 + 4 + 0 + 1$

$= \mathbf{45}$

Answer: $(101101)_2 = (45)_{10}$

Example 2

Convert $\mathbf{(375)_8}$ to decimal.

$= 3\times8^2 + 7\times8^1 + 5\times8^0$

$= 3\times64 + 7\times8 + 5$

$= 192 + 56 + 5$

$= \mathbf{253}$

Answer: $(375)_8 = (253)_{10}$

Example 3

Convert $\mathbf{(4C)_{16}}$ to decimal.

C = 12

$= 4\times16^1 + 12\times16^0$

$= 64 + 12$

$= \mathbf{76}$

Answer: $(4C)_{16} = (76)_{10}$

---

7. 🧠 Important Points to Remember

- Binary has **base 2**.
- Octal has **base 8**.
- Hexadecimal has **base 16**.
- The rightmost digit always has position **0**.
- In hexadecimal:

  - A = 10
  - B = 11
  - C = 12
  - D = 13
  - E = 14
  - F = 15
- For conversion to decimal, **multiply and add**.
- The base is raised according to the position of each digit.
- A valid octal number cannot contain **8 or 9**.
- A valid binary number can contain only **0 and 1**.

📝 Practice Questions

Convert the following numbers into Decimal:

1. $\mathbf{(10101)_2}$
2. $\mathbf{(111001)_2}$
3. $\mathbf{(1011110)_2}$
4. $\mathbf{(157)_8}$
5. $\mathbf{(324)_8}$
6. $\mathbf{(567)_8}$
7. $\mathbf{(2B)_{16}}$
8. $\mathbf{(3E)_{16}}$
9. $\mathbf{(1A5)_{16}}$
10. $\mathbf{(2F7)_{16}}$

⭐ Challenge Question

Convert:

$\mathbf{(10101101)_2}$**,** $\mathbf{(725)_8}$ **and** $\mathbf{(3A6)_{16}}$

into decimal and determine which one has the **largest decimal value**.

🎯 Lecture Summary

Other Base → Decimal

Binary → Decimal: Multiply each digit by the corresponding power of **2** and add.

Octal → Decimal: Multiply each digit by the corresponding power of **8** and add.

Hexadecimal → Decimal: Multiply each digit by the corresponding power of **16** and add.

The Golden Rule:

> **Digit × Base^Position → Add All → Decimal Answer**

📘 Chapter 2: Data Representation and Boolean Logic

Lecture 4 — One Base to Other Base System

🎯 Learning Objectives

By the end of this lecture, you will be able to:

- Convert Binary to Octal and Octal to Binary.
- Convert Binary to Hexadecimal and Hexadecimal to Binary.
- Convert Octal to Hexadecimal and Hexadecimal to Octal.
- Use the **grouping method** for Binary ↔ Octal and Binary ↔ Hexadecimal.
- Convert between any two number systems using the **Decimal intermediate method**.

---

1. 🔄 Methods of Conversion

There are two main ways to convert a number from one base to another.

Method 1: Using Decimal as an Intermediate

Base A → Decimal → Base B

For example:

Binary → Decimal → Octal

This method can be used for **any two number systems**.

Method 2: Direct Conversion

Some conversions can be performed directly using grouping:

- Binary ↔ Octal
- Binary ↔ Hexadecimal

> **Important:** Direct conversion is faster because both Octal and Hexadecimal have a direct relationship with Binary.

---

2. 💻 Binary ↔ Octal Conversion

Why do we group Binary digits in 3s?

Octal has base **8**.

Since:

$8 = 2^3$

every **3 binary bits** correspond to **1 octal digit**.

Binary to Octal Conversion

Steps:

1. Start grouping the binary digits from the **right side**.
2. Make groups of **3 bits**.
3. If the leftmost group has fewer than 3 bits, add zeros to the left.
4. Convert each group into its octal equivalent.

Binary–Octal Conversion Table

Binary	Octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Example 1: Convert $(101101)_2$ to Octal

Group the binary number from the right:

101 | 101

Now convert each group:

101 = 5

101 = 5

Therefore:

$(101101)_2 = (55)_8$

Example 2: Convert $(1101011)_2$ to Octal

Group from the right:

1 | 101 | 011

Add zeros to the first group:

001 | 101 | 011

Now:

001 = 1

101 = 5

011 = 3

Therefore:

$(1101011)_2 = (153)_8$

Octal to Binary

For Octal → Binary, replace every octal digit with its **3-bit binary equivalent**.

Example 3: Convert $(572)_8$ to Binary

5 → **101**

7 → **111**

2 → **010**

Therefore:

$(572)_8 = (101111010)_2$

---

3. 🔢 Binary ↔ Hexadecimal Conversion

Why do we group Binary digits in 4s?

Hexadecimal has base **16**.

Since:

$16 = 2^4$

every **4 binary bits** correspond to **1 hexadecimal digit**.

Binary–Hexadecimal Conversion Table

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Binary to Hexadecimal

Steps:

1. Start from the **right side**.
2. Divide the binary digits into groups of **4**.
3. Add zeros to the left if required.
4. Replace each group with its hexadecimal equivalent.

Example 1: Convert $(10110110)_2$ to Hexadecimal

Group into 4 bits:

1011 | 0110

Now:

1011 = B

0110 = 6

Therefore:

$(10110110)_2 = (B6)_{16}$

Example 2: Convert $(110101)_2$ to Hexadecimal

Group from the right:

11 | 0101

Add zeros to the left:

0011 | 0101

Now:

0011 = 3

0101 = 5

Therefore:

$(110101)_2 = (35)_{16}$

Hexadecimal to Binary

Replace every hexadecimal digit with its **4-bit binary equivalent**.

Example 3: Convert $(2F4)_{16}$ to Binary

2 → **0010**

F → **1111**

4 → **0100**

Therefore:

$(2F4)_{16} = (001011110100)_2$

Leading zeros may be removed:

$(2F4)_{16} = (1011110100)_2$

---

4. 🔁 Octal ↔ Hexadecimal Conversion

There is **no direct grouping relationship** between Octal and Hexadecimal.

Therefore, the easiest method is:

Octal → Binary → Hexadecimal

or

Hexadecimal → Binary → Octal

Example 1: Convert $(57)_8$ to Hexadecimal

First convert Octal → Binary:

5 → **101**

7 → **111**

So:

$(57)_8 = (101111)_2$

Now group the binary digits into 4:

0010 | 1111

0010 → 2

1111 → F

Therefore:

$(57)_8 = (2F)_{16}$

Example 2: Convert $(3A)_{16}$ to Octal

First convert Hexadecimal → Binary:

3 → **0011**

A → **1010**

So:

$(3A)_{16} = (00111010)_2$

Group into 3 from the right:

00 | 111 | 010

Add zeros to the left:

000 | 111 | 010

Now:

000 → 0

111 → 7

010 → 2

Therefore:

$(3A)_{16} = (72)_8$

---

5. 🧮 Conversion Through Decimal

If direct conversion is not convenient, use Decimal as an intermediate.

General Process

Number in Base A

↓

Convert to Decimal

↓

Convert Decimal to Base B

Example: Convert $(1011)_2$ to Octal

Step 1: Binary → Decimal

$(1011)_2$

$= 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$

= 8 + 0 + 2 + 1

= $\mathbf{11_{10}}$

Step 2: Decimal → Octal

Divide 11 by 8:

11 ÷ 8 = 1 remainder 3

1 ÷ 8 = 0 remainder 1

Read remainders from bottom to top:

13

Therefore:

$(1011)_2 = (13)_8$

> This method works for **any base-to-base conversion**.

---

6. ⭐ Quick Conversion Rules

Binary → Octal

Group 3 bits from the right.

$2^3 = 8$

Octal → Binary

Replace each octal digit with 3 bits.

Binary → Hexadecimal

Group 4 bits from the right.

$2^4 = 16$

Hexadecimal → Binary

Replace each hexadecimal digit with 4 bits.

Octal → Hexadecimal

Octal → Binary → Hexadecimal

Hexadecimal → Octal

Hexadecimal → Binary → Octal

Any Base → Any Other Base

Base A → Decimal → Base B

---

7. 📝 Practice Questions

A. Binary → Octal

1. Convert $(101010)_2$ to Octal.
2. Convert $(1110011)_2$ to Octal.
3. Convert $(10011101)_2$ to Octal.

B. Octal → Binary

4. Convert $(245)_8$ to Binary.
5. Convert $(731)_8$ to Binary.
6. Convert $(56)_8$ to Binary.

C. Binary → Hexadecimal

7. Convert $(10101111)_2$ to Hexadecimal.
8. Convert $(110101101)_2$ to Hexadecimal.
9. Convert $(11110010)_2$ to Hexadecimal.

D. Hexadecimal → Binary

10. Convert $(3C)_{16}$ to Binary.
11. Convert $(7A)_{16}$ to Binary.
12. Convert $(F25)_{16}$ to Binary.

E. Octal ↔ Hexadecimal

13. Convert $(75)_8$ to Hexadecimal.
14. Convert $(246)_8$ to Hexadecimal.
15. Convert $(3F)_{16}$ to Octal.
16. Convert $(A5)_{16}$ to Octal.

⭐ Challenge

Convert $(725)_8$ into Hexadecimal using:

Octal → Binary → Hexadecimal

---

8. 🧠 Important Points to Remember

- **3 Binary bits = 1 Octal digit**
- **4 Binary bits = 1 Hexadecimal digit**
- Binary and Octal have a direct grouping relationship because **$8 = 2^3$**.
- Binary and Hexadecimal have a direct grouping relationship because **$16 = 2^4$**.
- Octal and Hexadecimal do **not** have a direct grouping relationship.
- For Octal ↔ Hexadecimal, use **Binary as an intermediate**.
- For any difficult base conversion, use **Decimal as an intermediate**.
- When grouping Binary digits, always start from the **right side**.
- Add zeros only to the **leftmost group** when necessary.
- Leading zeros in the final answer can generally be removed.

🎯 Lecture Summary

The Most Important Shortcuts

Binary → Octal

```
3 bits → 1 octal digit
```

Octal → Binary

```
1 octal digit → 3 bits
```

Binary → Hexadecimal

```
4 bits → 1 hexadecimal digit
```

Hexadecimal → Binary

```
1 hexadecimal digit → 4 bits
```

Octal → Hexadecimal

```
Octal → Binary → Hexadecimal
```

Hexadecimal → Octal

```
Hexadecimal → Binary → Octal
```

Golden Rule

> **When converting between Binary and Octal, group in 3s.**

> **When converting between Binary and Hexadecimal, group in 4s.**

> **For any other conversion, use Decimal or Binary as an intermediate.**

📘 Chapter 2: Data Representation and Boolean Logic

Lecture 5 — Internal Storage Encoding of Characters

🎯 Learning Objectives

By the end of this lecture, you will be able to:

- Understand how computers store characters internally.
- Understand the meaning of **character encoding**.
- Explain **ASCII, ISCII and Unicode**.
- Differentiate between ASCII, ISCII and Unicode.
- Understand why Unicode is important in modern computing.

---

1. 💻 Character Encoding

Computers understand and process **binary data**, which consists of only:

0 and 1

However, humans use many different types of characters such as:

- Alphabets: `A, B, C, a, b, c`
- Numbers: `0, 1, 2, 3`
- Special symbols: `@, #, $, %, &`
- Characters from different languages: `अ, आ, क, م, 中`

Therefore, computers need a way to represent each character using a **numeric code**.

Character Encoding

Character encoding is a system that assigns a unique numerical value or code to each character so that the character can be stored and processed by a computer.

For example:

`Character → Numeric Code → Binary Representation`
A character such as **A** is assigned a particular code, which is stored inside the computer in binary form.

Why is Character Encoding Needed?

Without a common encoding system:

- Different computers might represent the same character differently.
- Text could become unreadable when transferred between systems.
- Different languages would be difficult to represent.
- Data exchange between computers would become complicated.

The major character encoding systems discussed in this chapter are:

1. **ASCII**
2. **ISCII**
3. **Unicode**

---

2. 🔤 ASCII — American Standard Code for Information Interchange

ASCII stands for:

American Standard Code for Information Interchange

ASCII was designed to represent characters commonly used in the English language.

ASCII Uses 7 Bits

Standard ASCII uses **7 bits** to represent a character.

Therefore, the total number of possible codes is:

$2^7 = 128$

So standard ASCII provides **128 different codes**, numbered from:

0 to 127

These codes represent:

- Uppercase letters
- Lowercase letters
- Digits

- Special symbols
- Control characters

Examples of ASCII Codes

Character	Decimal ASCII Code
A	65
B	66
C	67
a	97
b	98
c	99
0	48
1	49
Space	32

For example:

A → 65 → 1000001_2

Important ASCII Points

- Standard ASCII uses **7 bits**.
- It contains **128 codes**.
- It mainly supports the **English language**.
- It includes letters, digits, symbols and control characters.
- ASCII is still widely used as a foundation for modern computing.

ASCII Limitation

ASCII was mainly designed for English and therefore cannot adequately represent the characters of many other languages.

For example, characters such as:

अ, क, म, 中, م

cannot be represented using standard ASCII.

This created the need for more comprehensive character encoding systems.

---

3. IN ISCII — Indian Standard Code for Information Interchange

ISCII stands for:

Indian Standard Code for Information Interchange

It was developed to represent characters used in **Indian languages**.

ISCII was designed to support several Indian scripts, including scripts used for languages such as:

- Hindi
- Bengali
- Assamese
- Gujarati
- Punjabi
- Kannada
- Malayalam
- Tamil
- Telugu
- Odia

ISCII Uses 8 Bits

ISCII uses **8-bit encoding**.

Therefore:

$2^8 = 256$

possible code values can be represented.

Why Was ISCII Needed?

ASCII was mainly designed for English characters.

India has many languages and writing systems, so a character encoding system was needed to represent Indian scripts.

ISCII helped provide a common encoding approach for Indian-language text.

Important Features of ISCII

- ISCII stands for **Indian Standard Code for Information Interchange**.
- It uses **8-bit encoding**.
- It was designed primarily for **Indian scripts**.

- It supports multiple Indian languages/scripts.
- It was an important step toward multilingual computing in India.

Limitation of ISCII

ISCII was focused mainly on Indian scripts and was not designed to represent **all writing systems of the world**.

For worldwide multilingual computing, a more universal standard was needed.

---

4. 🌍 Unicode — Unicode for Multilingual Computing

Unicode is a universal character encoding standard designed to represent characters from **different languages and writing systems around the world**.

Unicode provides a unique code point for characters.

It supports:

- English
- Hindi
- Arabic
- Chinese
- Japanese
- Korean
- Greek
- Cyrillic
- Many other scripts
- Mathematical symbols
- Emojis and many special characters

For example:

```
A  → Unicode character
अ  → Unicode character
ع  → Unicode character
中 → Unicode character
😀 → Unicode character
```

Why is Unicode Important?

Modern computers and the Internet need to handle text from many languages.

Unicode allows text from different writing systems to be represented consistently.

For example, a single document can contain:

Hello
नमस्ते
مرحبا
你好
Unicode makes multilingual communication possible.

Unicode Code Points

Each Unicode character has a unique **code point**.

A Unicode code point is commonly written in hexadecimal form.

For example:

A → U+0041

Here:

- `U+` indicates Unicode.
- `0041` is the hexadecimal code point.

Similarly:

अ → U+0905

Unicode Encoding Forms

Unicode can be implemented using different encoding formats, such as:

- **UTF-8**
- **UTF-16**
- **UTF-32**

Among these, **UTF-8** is extremely common on the web.

---

5. 📊 ASCII vs ISCII vs Unicode

Feature	ASCII	ISCII	Unicode
Full Form	American Standard Code for Information Interchange	Indian Standard Code for Information Interchange	Unicode
Main Purpose	English/basic characters	Indian scripts	Worldwide multilingual text

Feature	ASCII	ISCII	Unicode
Standard Size	7 bits	8 bits	Variable depending on encoding
Number of Standard Codes	128	256 possible 8-bit values	Very large character set
Language Coverage	Mainly English	Indian languages/scripts	Languages worldwide
Multilingual Support	Limited	Mainly Indian scripts	Extensive
Modern Importance	Still widely used	Limited/legacy	Widely used

Easy Way to Remember

ASCII → English

ISCII → Indian Scripts

Unicode → Universal/Multilingual

---

6. 🧠 Important Terms

Character

A single letter, number, symbol or other textual element.

Examples:

`A` , `7` , `@` , `अ`

Character Set

A collection of characters that a system can represent.

Character Encoding

A method of assigning numerical codes to characters so they can be stored and processed by computers.

Code Point

A numerical value assigned to a particular character in an encoding system such as Unicode.

ASCII

A character encoding standard mainly designed for English characters.

ISCII

A character encoding standard designed to represent Indian scripts.

Unicode

A universal standard designed to represent characters from many languages and writing systems.

---

7. 🖊 Quick Examples

Example 1

Question: How many codes are possible using 7 bits?

Solution:

Number of possible codes:

$2^7 = 128$

Answer: 128 codes

Example 2

Question: How many different values can be represented using 8 bits?

Solution:

$2^8 = 256$

Answer: 256 values

Example 3

Question: Which encoding standard was mainly designed for English characters?

Answer: ASCII

Example 4

Question: Which encoding system was designed to support Indian scripts?

Answer: ISCII

Example 5

Question: Which encoding standard is designed for multilingual computing?

Answer: Unicode

---

8. 📝 Practice Questions

A. Fill in the Blanks

1. ASCII stands for _____.
2. ISCII stands for _____.
3. Standard ASCII uses _____ bits.
4. Standard ASCII provides _____ codes.
5. ISCII uses _____ bits.
6. Unicode is designed for _____ computing.

B. One-Word Questions

1. Which encoding standard mainly represents English characters?
2. Which encoding standard was developed for Indian scripts?
3. Which encoding standard supports characters from languages around the world?
4. How many codes can be represented using 7 bits?
5. What is the numerical value assigned to a Unicode character called?

C. Short-Answer Questions

1. What is character encoding?
2. What is ASCII? Mention its number of bits and number of standard codes.
3. What is ISCII and why was it developed?
4. What is Unicode?
5. Why is Unicode preferred for multilingual computing?
6. Write two differences between ASCII and Unicode.
7. Differentiate between ASCII, ISCII and Unicode.

🎯 Lecture Summary

Character Encoding

Character → Code → Binary Representation

ASCII

- Mainly for English/basic characters
- Uses **7 bits**
- Provides **128 standard codes**

ISCII

- Designed for **Indian scripts**
- Uses **8 bits**
- Supports multiple Indian writing systems

Unicode

- Designed for **multilingual computing**
- Supports scripts from around the world
- Uses different encoding formats such as **UTF-8, UTF-16 and UTF-32**

⭐ Remember

> **ASCII = Mainly English**

> **ISCII = Indian Scripts**

> **Unicode = Multilingual/Universal**

The purpose of all these systems is the same at a basic level:

To represent characters in a form that computers can store, process and exchange.

📘 Chapter 2: Data Representation and Boolean Logic

Lecture 6 — Boolean Logic

🎯 Learning Objectives

By the end of this lecture, you will be able to:

- Understand the meaning of **Boolean Logic**.
- Understand Boolean values **True** and **False**.
- Represent Boolean values using **1 and 0**.
- Understand Boolean variables and expressions.
- Understand the basic principles of Boolean algebra.
- Relate Boolean Logic to computer decision-making.

---

1. 🧠 What is Boolean Logic?

Boolean Logic is a system of logic in which every statement or expression has one of only **two possible values**:

- **True**
- **False**

Boolean Logic is named after **George Boole**, an English mathematician who developed the mathematical system of logic on which digital computing is based.

In computers, Boolean values are commonly represented as:

Boolean Value	Binary Representation
True	1
False	0

Therefore:

True = 1

False = 0

Example

Consider the statement:

> "5 is greater than 3."

This statement is **True**.

Therefore:

5 > 3 → True → 1

Now consider:

> "2 is greater than 7."

This statement is **False**.

Therefore:

2 > 7 → False → 0

---

2. 🔢 Boolean Values and Boolean Variables

A **Boolean variable** is a variable that can have only two possible values:

True or **False**

In Python, Boolean values are written as:

```
True
False
```

Example

```
is_student = True
```
Here:

- `is_student` is a Boolean variable.
- Its value is `True`.

Another example:

```
is_raining = False
```
Here:

- `is_raining` is a Boolean variable.
- Its value is `False`.

Important

Python uses:

`True`
`False`
with a **capital T** and **capital F**.

---

3. 🔍 Boolean Expressions

A **Boolean expression** is an expression that produces either:

True or **False**

Comparison operations commonly produce Boolean results.

Examples

```
5 > 3
```

Output:

```
True
```

Because 5 is greater than 3.

```
10 < 4
```

Output:

```
False
```

Because 10 is not less than 4.

```
7 == 7
```

Output:

```
True
```

Because both values are equal.

More Examples

Expression	Result
`10 > 5`	True
`10 < 5`	False
`8 == 8`	True
`7 != 7`	False
`4 >= 4`	True
`3 <= 2`	False

Boolean expressions are extremely important because computers use them to **make decisions**.

---

4. ⚙ Boolean Algebra

Boolean Algebra is a mathematical system used to work with Boolean values.

Unlike ordinary algebra, Boolean Algebra deals mainly with:

0 and 1

or:

False and True

Boolean Algebra is widely used in:

- Computer science
- Digital electronics
- Logic circuits
- Computer architecture
- Programming
- Decision-making systems

Basic Boolean Operations

The three fundamental Boolean operations are:

1. **AND**
2. **OR**
3. **NOT**

These operations will be studied in detail in the next lecture.

---

5. 🔗 Boolean Logic in Computers

Computers constantly make decisions based on conditions.

For example:

> If a student's marks are greater than or equal to 40, the student passes.

This condition can be represented as:

```
Marks >= 40
```

If the marks are 75:

```
75 >= 40 → True
```

If the marks are 25:

```
25 >= 40 → False
```

Therefore, Boolean Logic helps a computer determine **which condition is satisfied**.

Real-Life Examples

Example 1:

> Is the temperature greater than 30°C?

Answer:

True / False

Example 2:

> Is the password correct?

Answer:

True / False

Example 3:

> Is the user eligible to vote?

Answer:

True / False

All these questions can be represented using Boolean Logic.

---

6. 💡 Boolean Logic and Binary

Boolean Logic and the binary number system are closely related.

Boolean Logic	Binary
False	0
True	1

A computer's digital circuits work using binary signals.

A signal can generally be considered:

OFF → 0

ON → 1

This makes Boolean Logic suitable for designing digital computer systems.

Example

Suppose:

```
Switch A = ON
```
Then:

A = 1

If:

`Switch A = OFF`
Then:

A = 0

By combining such values using Boolean operations, computers can perform logical decisions.

---

7. 🖉 Solved Examples

Example 1

Determine whether the following expression is True or False:

`15 > 10`
Since 15 is greater than 10:

Answer: True

Example 2

`8 == 5`
8 is not equal to 5.

Answer: False

Example 3

`20 >= 20`
20 is equal to 20.

Therefore:

Answer: True

Example 4

If:

```
A = True
B = False
```
Then:

A = 1

B = 0

These values can be used with Boolean operators such as **AND, OR and NOT**.

---

8. 📝 Practice Questions

A. Fill in the Blanks

1. Boolean Logic was developed by _____.
2. Boolean Logic has _____ possible values.
3. True is represented by _____ in binary.
4. False is represented by _____ in binary.
5. Boolean Algebra mainly deals with _____ and _____.
6. The three basic Boolean operations are _____, _____ and _____.

B. True or False

1. Boolean Logic has three possible values.
2. True can be represented by 1.
3. False can be represented by 0.
4. Boolean expressions can produce True or False.
5. Boolean Logic is used in digital computer systems.

C. Find the Output

Determine whether each expression produces **True** or **False**:

```
12 > 7
4 == 9
15 >= 15
6 < 2
10 != 5
```

D. Short-Answer Questions

1. What is Boolean Logic?
2. What are the two values used in Boolean Logic?
3. What is a Boolean variable?
4. What is a Boolean expression?
5. What is Boolean Algebra?
6. Why is Boolean Logic important in computers?
7. Explain the relationship between Boolean Logic and binary numbers.

🎯 Lecture Summary

Boolean Logic

A system of logic based on two values:

True and False

Binary Representation

True = 1

False = 0

Boolean Expression

An expression that produces:

True or False

Boolean Algebra

A mathematical system that works with Boolean values, mainly:

0 and 1

Three Basic Boolean Operations

AND

OR

NOT

These three operations form the foundation of Boolean Logic and will be covered in detail in the **next lecture**.

> ⭐ **Remember: Boolean Logic → True/False → 1/0 → Computer Decisions**

📘 Chapter 2: Data Representation and Boolean Logic

Lecture 7 — Boolean Operators

🎯 Learning Objectives

By the end of this lecture, you will be able to:

- Understand the three basic Boolean operators.
- Explain **AND, OR and NOT** operations.
- Evaluate Boolean expressions.
- Understand how Boolean operators work with `True` and `False`.
- Use Boolean operators in Python.
- Relate Boolean operators to real-life conditions.

---

1. 🔗 What are Boolean Operators?

Boolean operators are operators used to perform logical operations on Boolean values.

The three basic Boolean operators are:

1. **AND**
2. **OR**
3. **NOT**

Boolean values are:

True = 1

False = 0

Boolean operators are used to combine or modify conditions and are very important in:

- Programming
- Decision making
- Digital electronics
- Logic circuits
- Computer systems

---

2. 🟢 AND Operator

The **AND** operation gives **True only when both inputs are True**.

Rule

> If **both** conditions are True → Result is True.

If even one condition is False, the result is False.

Truth Table

A	B	A AND B
False	False	False
False	True	False
True	False	False
True	True	True

In binary form:

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

Example

Suppose:

- A = "I have a valid ID."
- B = "I have a valid ticket."

To enter an event, **both** conditions must be satisfied.

```
A = True
B = True

A AND B = True
```

If either condition is False:

```
A = True
B = False

A AND B = False
```

Python

Python uses the keyword:

```
and
```

Example:

```
age = 20
has_id = True
```

```
print(age >= 18 and has_id)
```

Output:

```
True
```

---

3. 🔵 OR Operator

The **OR** operation gives **True if at least one input is True**.

Rule

> If **one or both** conditions are True → Result is True.

The result is False only when **both inputs are False**.

Truth Table

A	B	A OR B
False	False	False
False	True	True
True	False	True
True	True	True

In binary form:

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

Example

Suppose a student can enter a competition if:

- A = "Student has an invitation."
- B = "Student has a valid registration."

If either condition is satisfied, entry is allowed.

```
A = True
B = False

A OR B = True
```

Python

Python uses:

```
or
```

Example:

```
has_invitation = False
has_registration = True

print(has_invitation or has_registration)
```

Output:

```
True
```

---

4. 🔴 NOT Operator

The **NOT** operator reverses the Boolean value.

Rule

> True becomes False, and False becomes True.

Truth Table

A	NOT A
False	True
True	False

In binary form:

A	NOT A
0	1
1	0

Example

If:

```
A = True
```

Then:

```
NOT A = False
```

If:

```
A = False
```

Then:

```
NOT A = True
```

Python

Python uses:

```
not
```

Example:

```
is_raining = False

print(not is_raining)
```

Output:

```
True
```

---

5. 📊 Comparison of AND, OR and NOT

Operator	Meaning	Result
AND	Both conditions must be True	True only when both are True
OR	At least one condition must be True	False only when both are False
NOT	Reverses the condition	True ↔ False

Easy Way to Remember

AND → All must be True

OR → One is enough

NOT → Reverse

---

6. 💻 Boolean Operators in Python

Python provides three logical operators:

```
and
or
```

```
not
```

AND Example

```
a = True
b = False

print(a and b)
```

Output:

```
False
```

OR Example

```
a = True
b = False

print(a or b)
```

Output:

```
True
```

NOT Example

```
a = True

print(not a)
```

Output:

```
False
```

Using Comparisons

Boolean operators are often combined with comparison operators.

```
age = 20
marks = 75

print(age >= 18 and marks >= 40)
```

Output:

```
True
```

Both conditions are True.

---

7. 🧮 Solved Examples

Example 1

Find:

`True AND False`
Since both values are not True:

Answer: False

Example 2

Find:

`True OR False`
At least one value is True.

Answer: True

Example 3

Find:

`NOT False`
NOT reverses False.

Answer: True

Example 4

Evaluate:

`(True AND True) OR False`
First:

`True AND True = True`
Then:

`True OR False = True`
Answer: True

Example 5

Evaluate:

`NOT(True AND False)`
First:

`True AND False = False`

Then:

`NOT False = True`

Answer: True

---

8. 📝 Practice Questions

A. Fill in the Blanks

1. The three basic Boolean operators are _____, _____ and _____.
2. AND gives True when _____ inputs are True.
3. OR gives False only when _____ inputs are False.
4. NOT _____ the Boolean value.
5. In Python, AND is represented by the keyword _____.
6. In Python, OR is represented by the keyword _____.
7. In Python, NOT is represented by the keyword _____.

B. Find the Output

Determine the result:

1. `True and True`
2. `True and False`
3. `False or True`
4. `False or False`
5. `not True`
6. `not False`
7. `(True and False) or True`
8. `not(False or False)`

C. Python Practice

Predict the output:

```
a = 10
b = 20

print(a < b and b > 15)
x = 5
y = 10

print(x > y or x == 5)
is_holiday = False

print(not is_holiday)
```

D. Short-Answer Questions

1. What is a Boolean operator?
2. Explain the AND operator with its truth table.
3. Explain the OR operator with its truth table.
4. Explain the NOT operator with its truth table.
5. What are the Python keywords used for AND, OR and NOT?
6. Differentiate between AND and OR.
7. Why are Boolean operators important in programming?

🎯 Lecture Summary

AND

Both must be True.

```
1 AND 1 = 1
```

OR

At least one must be True.

```
1 OR 0 = 1
```

NOT

Reverses the value.

```
NOT 1 = 0
NOT 0 = 1
```

Python

Boolean Operation	Python Operator
AND	`and`
OR	`or`
NOT	`not`

⭐ Remember

> **AND → Both required**

> **OR → One is enough**

> **NOT → Reverse**

These three operators form the foundation for creating more complex Boolean expressions and will be used in the **truth tables and logic circuits** covered in the upcoming lectures.

📘 Chapter 2: Data Representation and Boolean Logic

Lecture 8 — Truth Tables

🎯 Learning Objectives

By the end of this lecture, you will be able to:

- Understand what a **Truth Table** is.
- Construct truth tables for Boolean expressions.
- Create truth tables for **AND, OR and NOT**.
- Understand the number of rows required in a truth table.
- Evaluate compound Boolean expressions using truth tables.

---

1. 📊 What is a Truth Table?

A **Truth Table** is a table used to show all possible combinations of input values and their corresponding output for a Boolean expression or logical operation.

Boolean values can be:

- **True (1)**
- **False (0)**

A truth table helps us understand exactly how a Boolean operator behaves for every possible input combination.

Example

For two Boolean variables:

```
A = 0 or 1
B = 0 or 1
```

There are four possible combinations:

```
00
01
10
11
```

Therefore, a truth table with **2 input variables has 4 rows**.

---

2. 🔢 Number of Rows in a Truth Table

The number of rows depends on the number of input variables.

Formula

Number of rows = 2^n

where:

n = number of input variables

Examples

Number of Inputs	Number of Rows
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^3 = 8$
4	$2^4 = 16$

Important Rule

- 1 input → **2 rows**
- 2 inputs → **4 rows**
- 3 inputs → **8 rows**
- 4 inputs → **16 rows**

---

3. 🟢 Truth Table for AND

The AND operation produces **1 only when both inputs are 1**.

AND Truth Table

A	B	A AND B
0	0	0

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

Remember

> **AND = Both must be 1**

Only the last row produces **1**.

Python Equivalent

```
A and B
```

---

4. 🔵 Truth Table for OR

The OR operation produces **1 when at least one input is 1**.

OR Truth Table

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

Remember

> **OR = At least one must be 1**

Only the first row produces **0**.

Python Equivalent

```
A or B
```

---

5. 🔴 Truth Table for NOT

NOT has only **one input** because it simply reverses the input value.

NOT Truth Table

A	NOT A
0	1
1	0

Remember

> **NOT = Reverse**

If:

A = 0 → NOT A = 1

If:

A = 1 → NOT A = 0

Python Equivalent

```
not A
```

---

6. 🧮 Truth Table for Compound Expressions

A **compound Boolean expression** contains more than one Boolean operation.

For example:

A AND B OR C

To construct its truth table, we must consider all possible combinations of A, B and C.

Since there are **3 variables**:

Number of rows = 2^3 = 8

Example

Construct the truth table for:

A AND B

A	B	A AND B
0	0	0
0	1	0

A	B	A AND B
1	0	0
1	1	1

Now consider:

(A AND B) OR C

We first calculate:

A AND B

and then use that result with C.

A	B	C	A AND B	(A AND B) OR C
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Important Technique

For complex expressions:

1. Write all input combinations.
2. Calculate the **first operation**.
3. Use that result for the next operation.
4. Continue until the final output is obtained.

---

7. 🖊 Solved Examples

Example 1

Construct the truth table for:

A OR B

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

Example 2

Construct the truth table for:

NOT A

A	NOT A
0	1
1	0

Example 3

Find the output of:

A AND (NOT B)

First calculate **NOT B**, then perform AND.

A	B	NOT B	A AND (NOT B)
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

Therefore, the expression is **True/1 only when A = 1 and B = 0**.

---

8. 📝 Practice Questions

A. Complete the Truth Tables

1. AND

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

2. OR

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

3. NOT

A	NOT A
0	?
1	?

B. Construct Truth Tables

Construct complete truth tables for:

1. **A AND B**
2. **A OR B**
3. **NOT A**
4. **NOT B**
5. **A AND (NOT B)**
6. **(A OR B) AND C**
7. **(A AND B) OR C**
8. **NOT(A AND B)**

C. Short-Answer Questions

1. What is a truth table?
2. How many rows are required for 2 Boolean variables?
3. How many rows are required for 3 Boolean variables?
4. Write the formula for determining the number of rows.
5. Why does a NOT operation require only one input?

6. What is the output of AND when both inputs are 1?
7. What is the output of OR when both inputs are 0?

🎯 Lecture Summary

Truth Table

A truth table shows **all possible input combinations and their corresponding outputs**.

Number of Rows

2^n

where **n = number of input variables**.

AND

1 AND 1 = 1

All other combinations produce 0.

OR

Only:

0 OR 0 = 0

All other combinations produce 1.

NOT

0 → 1

1 → 0

⭐ Remember

> **AND → 1 only when all inputs are 1**

> **OR → 1 when at least one input is 1**

> **NOT → reverses the input**

> **n inputs → 2^n rows**

In the next lecture, we will move from **truth tables to Logic Circuits**, where we will see how these Boolean operations are represented using actual logic gates.

📘 Chapter 2: Data Representation and Boolean Logic

Lecture 9 — Logic Circuits and Logic Gates

🎯 Learning Objectives

By the end of this lecture, you will be able to:

- Understand what a **logic circuit** is.
- Understand the meaning of a **logic gate**.
- Explain **AND, OR and NOT gates**.
- Understand the relationship between **Boolean expressions, truth tables and logic gates**.
- Determine the output of basic logic circuits.
- Draw simple logic circuits from Boolean expressions.

---

1. ⚡ Introduction to Logic Circuits

Computers work internally using two binary values:

0 and 1

These values can represent:

Binary Value	Boolean Value	Signal
0	False	OFF / LOW
1	True	ON / HIGH

A computer performs logical operations using electronic circuits called **Logic Circuits**.

Logic Circuit

A **logic circuit** is an electronic circuit that performs logical operations on one or more binary inputs to produce a binary output.

A logic circuit contains one or more **logic gates**.

Logic Gate

A **logic gate** is the basic building block of a digital circuit that performs a specific Boolean operation.

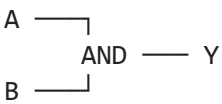
The three basic logic gates are:

1. **AND Gate**
2. **OR Gate**
3. **NOT Gate**

Basic Flow

Binary Inputs → Logic Gate → Binary Output

Example:



Here:

- A and B are inputs.
- AND is the logic gate.
- Y is the output.

---

2. 🟢 AND Gate

An **AND Gate** performs the Boolean **AND operation**.

It usually has:

- Two or more inputs
- One output

Rule

> The output of an AND gate is **1 only when ALL inputs are 1**.

Boolean Expression

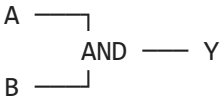
For inputs A and B:

Y = A AND B

It can also be written as:

Y = A · B

Conceptual Diagram



Truth Table

A	B	Y = A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Example

Suppose a machine starts only when:

- Switch A is ON
- Switch B is ON

If:

A = 1

and

B = 1

then:

Y = 1

But if either switch is OFF:

Y = 0

Easy Trick

> **AND = ALL inputs must be 1**

---

3. 🔵 OR Gate

An **OR Gate** performs the Boolean **OR operation**.

It usually has:

- Two or more inputs
- One output

Rule

> The output of an OR gate is **1 if at least one input is 1**.

Boolean Expression

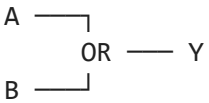
Y = A OR B

It can also be written as:

Y = A + B

Here, `+` represents Boolean OR, not ordinary arithmetic addition.

Conceptual Diagram



Truth Table

A	B	Y = A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Example

Suppose an alarm activates when:

- Sensor A detects movement

OR

- Sensor B detects movement

If either sensor produces `1`, the alarm activates.

Easy Trick

> **OR = ONE input being 1 is enough**

---

4. 🔴 NOT Gate

A **NOT Gate** performs the Boolean **NOT operation**.

Unlike AND and OR gates, a NOT gate has:

- **One input**
- **One output**

It is also known as an:

Inverter

because it reverses the input.

Rule

> The output is always the **opposite of the input**.

Boolean Expression

Y = NOT A

It may also be represented as:

Y = $\bar{A}$

Conceptual Diagram

A ── NOT ── Y

Truth Table

A	Y = NOT A
0	1
1	0

Therefore:

Input 0 → Output 1

Input 1 → Output 0

Easy Trick

> **NOT = Opposite / Reverse**

---

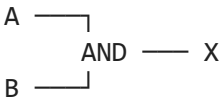
5. 🔗 Boolean Expression → Logic Circuit

Multiple logic gates can be connected together to create a **logic circuit**.

Consider the Boolean expression:

Y = (A AND B) OR C

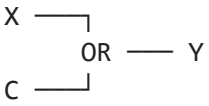
Step 1 — Perform A AND B



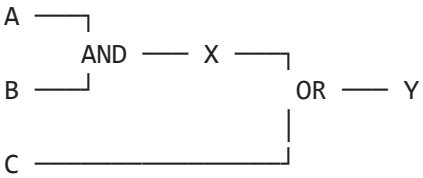
Here:

X = A AND B

Step 2 — Perform X OR C



Complete Conceptual Circuit



Therefore:

Y = (A AND B) OR C

Example

Suppose:

A = 1

B = 0

C = 1

First:

A AND B

= 1 AND 0

= **0**

Now:

0 OR C

= 0 OR 1

= **1**

Therefore:

Y = 1

---

6. 🧮 Solving Logic Circuits

When a circuit contains multiple gates, solve it **gate by gate**.

Example 1

Find the output of:

Y = (A OR B) AND C

when:

A = 0

B = 1

C = 1

Step 1

Calculate:

A OR B

= 0 OR 1

= **1**

Step 2

Now:

1 AND C

= 1 AND 1

= **1**

Therefore:

Y = 1

Example 2

Find:

Y = NOT(A AND B)

when:

A = 1

B = 1

First:

A AND B

= 1 AND 1

= **1**

Then:

NOT 1

= **0**

Therefore:

Y = 0

Example 3

Find:

Y = (NOT A) OR B

when:

A = 1

B = 0

First:

NOT A

= NOT 1

= **0**

Then:

0 OR 0

= **0**

Therefore:

Y = 0

⭐ Rule for Complex Circuits

> **Solve the circuit from input side to output side, one gate at a time.**

---

7. 📊 Boolean Operator vs Logic Gate

Boolean operators and logic gates represent the same logical concepts in different ways.

Boolean Operation	Logic Gate	Basic Rule
AND	AND Gate	Output 1 only if all inputs are 1
OR	OR Gate	Output 1 if at least one input is 1
NOT	NOT Gate	Reverses the input

Relationship

```
Boolean Logic
      ↓
Boolean Expression
      ↓
Truth Table
      ↓
Logic Gates
```

```
      ↓
Logic Circuit
```

Example

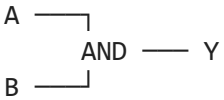
Boolean Expression:

Y = A AND B

Truth Table:

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

Logic Circuit:



All three represent the **same logical operation**.

---

8. 📝 Practice Questions

A. Fill in the Blanks

1. A _____ performs a logical operation on binary inputs.
2. The three basic logic gates are _____, _____ and _____.
3. An AND gate gives output 1 only when _____ inputs are 1.
4. An OR gate gives output 1 when _____ input is 1.
5. A NOT gate is also called an _____.
6. A NOT gate has _____ input.
7. Logic circuits work using binary values _____ and _____.

B. Identify the Gate

1. Output is 1 only when both inputs are 1.
2. Output is 1 when at least one input is 1.
3. Output is the opposite of the input.
4. The gate is also known as an inverter.

C. Find the Output

Evaluate:

1. **1 AND 1**
2. **1 AND 0**
3. **0 OR 1**
4. **0 OR 0**
5. **NOT 1**
6. **NOT 0**

D. Solve the Boolean Expressions

Find the output for the given values:

Question 1

Y = (A AND B) OR C

where:

A = 1, B = 1, C = 0

Question 2

Y = (A OR B) AND C

where:

A = 0, B = 1, C = 1

Question 3

Y = NOT(A AND B)

where:

A = 1, B = 0

Question 4

Y = (NOT A) OR B

where:

A = 0, B = 0

E. Draw Using tldraw

Draw the logic circuits for:

1. **Y = A AND B**
2. **Y = A OR B**
3. **Y = NOT A**
4. **Y = (A AND B) OR C**
5. **Y = (A OR B) AND C**
6. **Y = NOT(A AND B)**

F. Short-Answer Questions

1. What is a logic circuit?
2. What is a logic gate?
3. Explain the AND gate with its truth table.
4. Explain the OR gate with its truth table.
5. Why is a NOT gate called an inverter?
6. What is the difference between an AND gate and an OR gate?
7. How are Boolean expressions related to logic circuits?

🎯 Lecture Summary

AND Gate

Both/All inputs must be 1

```
1 AND 1 = 1
```

OR Gate

At least one input must be 1

```
1 OR 0 = 1
```

NOT Gate

Reverses the input

```
NOT 1 = 0
NOT 0 = 1
```

Boolean Representation

AND → A · B

OR → A + B

NOT → $\bar{A}$

⭐ Golden Rules

> **AND → ALL must be 1**

> **OR → ONE is enough**

> **NOT → OPPOSITE**

> **For a complex logic circuit, solve one gate at a time from input to output.**

🔜 Next Lecture

Lecture 10 — De Morgan's Laws + Chapter Revision

We will learn:

- De Morgan's First Law
- De Morgan's Second Law
- Verification using Truth Tables
- Important chapter concepts
- Chapter revision and practice

📘 Chapter 2: Data Representation and Boolean Logic

Lecture 10 — De Morgan's Laws + Chapter Revision

🎯 Learning Objectives

By the end of this lecture, you will be able to:

- Understand **De Morgan's Laws**.
- State the two laws correctly.
- Apply De Morgan's Laws to Boolean expressions.
- Verify De Morgan's Laws using truth tables.
- Simplify Boolean expressions using De Morgan's Laws.
- Revise the complete chapter.

---

1. 🧠 Introduction to De Morgan's Laws

De Morgan's Laws are important rules in Boolean Algebra that describe how the **NOT** operation behaves when applied to **AND** and **OR** expressions.

There are **two De Morgan's Laws**.

First Law

NOT (A AND B) = (NOT A) OR (NOT B)

In Boolean notation:

$$\overline{(A \cdot B)} = \bar{A} + \bar{B}$$

Second Law

NOT (A OR B) = (NOT A) AND (NOT B)

In Boolean notation:

$$\overline{(A + B)} = \bar{A} \cdot \bar{B}$$

Easy Way to Remember

When NOT is applied to a group:

> **AND changes to OR**

and

> **OR changes to AND**

Also, every variable is complemented.

Example

NOT (A AND B)

becomes:

(NOT A) OR (NOT B)

Similarly:

NOT (A OR B)

becomes:

(NOT A) AND (NOT B)

---

2. 🔴 De Morgan's First Law

Statement

NOT (A AND B) = (NOT A) OR (NOT B)

Or:

$(A \cdot B)' = \bar{A} + \bar{B}$

This means that when we apply NOT to an AND expression:

AND → OR

and each input is also complemented.

Example

Suppose:

A = 1

B = 1

Left side:

NOT (A AND B)

= NOT (1 AND 1)

= NOT 1

= **0**

Right side:

(NOT A) OR (NOT B)

= (NOT 1) OR (NOT 1)

= 0 OR 0

= **0**

Both sides are equal.

Therefore:

NOT (A AND B) = (NOT A) OR (NOT B)

---

3. 🔵 Verification of First Law Using Truth Table

We need to verify:

NOT (A AND B) = (NOT A) OR (NOT B)

A	B	A AND B	NOT(A AND B)	NOT A	NOT B	(NOT A) OR (NOT B)
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Compare the two final columns:

NOT(A AND B)

and

(NOT A) OR (NOT B)

They are identical.

Therefore, the **First De Morgan's Law is verified**.

⭐ Remember

NOT(AND) = NOT A OR NOT B

---

4. 🟢 De Morgan's Second Law

Statement

NOT (A OR B) = (NOT A) AND (NOT B)

Or:

$\overline{(A + B)} = \bar{A} \cdot \bar{B}$

This means that when we apply NOT to an OR expression:

OR → AND

and each input is also complemented.

Example

Suppose:

A = 0

B = 0

Left side:

NOT (A OR B)

= NOT (0 OR 0)

= NOT 0

= **1**

Right side:

(NOT A) AND (NOT B)

= (NOT 0) AND (NOT 0)

= 1 AND 1

= **1**

Both sides are equal.

Therefore:

NOT (A OR B) = (NOT A) AND (NOT B)

---

5. 🔵 Verification of Second Law Using Truth Table

We need to verify:

NOT (A OR B) = (NOT A) AND (NOT B)

A	B	A OR B	NOT(A OR B)	NOT A	NOT B	(NOT A) AND (NOT B)
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Compare:

NOT(A OR B)

and

(NOT A) AND (NOT B)

Both columns are identical.

Therefore, the **Second De Morgan's Law is verified**.

⭐ Remember

NOT(OR) = NOT A AND NOT B

---

6. 🖊 Applying De Morgan's Laws

De Morgan's Laws can be used to simplify or transform Boolean expressions.

Example 1

Simplify:

NOT(A AND B)

Using First Law:

NOT(A AND B) = (NOT A) OR (NOT B)

Answer:

$\bar{A} + \bar{B}$

Example 2

Simplify:

NOT(A OR B)

Using Second Law:

NOT(A OR B) = (NOT A) AND (NOT B)

Answer:

$\bar{A} \cdot \bar{B}$

Example 3

Simplify:

NOT(A AND B AND C)

Apply the same rule:

AND → OR

and complement every variable.

Therefore:

NOT(A AND B AND C)

= **(NOT A) OR (NOT B) OR (NOT C)**

Example 4

Simplify:

NOT(A OR B OR C)

Apply:

OR → AND

and complement every variable.

Therefore:

NOT(A OR B OR C)

= **(NOT A) AND (NOT B) AND (NOT C)**

---

7. 📑 Complete Chapter Revision

2.1 Introduction

Data inside computers is represented using suitable forms that computers can store and process.

The chapter focuses on:

- Number systems
- Character encoding
- Boolean Logic
- Boolean operators
- Truth tables
- Logic circuits

- De Morgan's Laws

2.2 Number System

A **number system** is a method of representing numbers using a specific set of digits and a base.

Number System	Base	Digits
Binary	2	0, 1
Octal	8	0–7
Decimal	10	0–9
Hexadecimal	16	0–9, A–F

2.3 Number System Conversions

Decimal → Other Base

Use the **repeated division method**.

Other Base → Decimal

Use the **positional value method**:

Digit × Base^Position

and add all values.

Binary ↔ Octal

Group Binary digits in **3s**.

3 bits = 1 Octal digit

Binary ↔ Hexadecimal

Group Binary digits in **4s**.

4 bits = 1 Hexadecimal digit

Octal ↔ Hexadecimal

Use Binary as an intermediate:

Octal → Binary → Hexadecimal

or:

Hexadecimal → Binary → Octal

2.4 Internal Storage Encoding of Characters

Computers need character encoding to represent text.

ASCII

- Mainly designed for English/basic characters.
- Standard ASCII uses **7 bits**.
- Provides **128 standard codes**.

ISCII

- Indian Standard Code for Information Interchange.
- Designed for Indian scripts.
- Uses **8-bit encoding**.

Unicode

- Designed for multilingual computing.
- Supports characters from many languages and writing systems.
- Common encoding forms include **UTF-8, UTF-16 and UTF-32**.

Easy Memory Trick

ASCII → Mainly English

ISCII → Indian Scripts

Unicode → Worldwide/Multilingual

2.5 Boolean Logic

Boolean Logic works with two values:

True = 1

False = 0

A Boolean expression produces either **True or False**.

2.6 Boolean Operators

The three basic Boolean operators are:

Operator	Meaning	Basic Rule
AND	Both conditions	Both must be 1
OR	At least one condition	One or more can be 1
NOT	Reversal	0 ↔ 1

Python:

```
and
or
not
```

2.7 Truth Table

A truth table shows all possible combinations of inputs and their corresponding outputs.

Formula

Number of rows = 2^n

where **n = number of input variables**.

Examples:

2 inputs → 2^2 = 4 rows

3 inputs → 2^3 = 8 rows

2.8 Logic Circuit

A logic circuit is an electronic circuit that performs logical operations using logic gates.

Basic gates:

- AND
- OR
- NOT

Relationship:

Boolean Expression → Truth Table → Logic Circuit

2.9 De Morgan's Laws

First Law

NOT(A AND B) = (NOT A) OR (NOT B)

$\overline{(A \cdot B)} = \bar{A} + \bar{B}$

Second Law

NOT(A OR B) = (NOT A) AND (NOT B)

$\overline{(A + B)} = \bar{A} \cdot \bar{B}$

Golden Rule

> **When NOT crosses a bracket: AND changes to OR, OR changes to AND, and every variable is complemented.**

---

8. 📝 Final Practice Questions

A. Number System

1. Convert **$(101101)_2$** to Decimal.
2. Convert **$(347)_8$** to Decimal.
3. Convert **$(2AF)_{16}$** to Decimal.
4. Convert **$(110101)_2$** to Octal.
5. Convert **$(10101111)_2$** to Hexadecimal.
6. Convert **$(75)_8$** to Hexadecimal.

B. Character Encoding

7. What is ASCII?
8. How many standard codes are provided by 7-bit ASCII?
9. What is ISCII?
10. Why is Unicode important?

C. Boolean Logic

11. What is Boolean Logic?
12. What are the three basic Boolean operators?
13. What is the output of `True AND False`?
14. What is the output of `True OR False`?
15. What is `NOT True`?

D. Truth Tables and Logic Circuits

16. How many rows are required for 3 Boolean variables?

17. Draw the truth table of AND.
18. Draw the truth table of OR.
19. Draw the truth table of NOT.
20. Draw a logic circuit for **Y = (A AND B) OR C**.

E. De Morgan's Laws

21. State De Morgan's First Law.
22. State De Morgan's Second Law.
23. Simplify **NOT(A AND B)**.
24. Simplify **NOT(A OR B)**.
25. Verify one De Morgan's Law using a truth table.

🎯 Chapter 2 — Final Quick Revision

Number Systems

Binary → Base 2

Octal → Base 8

Decimal → Base 10

Hexadecimal → Base 16

Conversion Shortcuts

Binary ↔ Octal → Groups of 3

Binary ↔ Hexadecimal → Groups of 4

Character Encoding

ASCII → English/basic characters

ISCII → Indian scripts

Unicode → Multilingual

Boolean Logic

True = 1

False = 0

Operators

AND → Both

OR → At least one

NOT → Reverse

Truth Tables

n inputs → 2^n rows

Logic Gates

AND, OR, NOT

De Morgan's Laws

NOT(AND) → OR + NOT

NOT(OR) → AND + NOT

⭐ Chapter Completion

You have now completed:

2.1 Introduction

→ 2.2 Number System

→ 2.3 Number System Conversions

→ 2.4 Internal Storage Encoding of Characters

→ 2.5 Boolean Logic

→ 2.6 Boolean Operators

→ 2.7 Truth Table

→ 2.8 Logic Circuit

→ 2.9 De Morgan's Law

🎉 Chapter 2 Complete!

Next step: Chapter 2 — 15 Marks Test

In [ ]: