Session - 18

Bits

CONTENTS

01

Introduction to Bits

Introduction to Bits

What is Bit Manipulation?

- Performing operations directly on bits of numbers
- Works at the **lowest level** of data (binary level)
- Used for **optimization**, **encoding**, **flags**, etc.

Introduction to Bits

Why Bit Manipulation?

- Super fast (direct hardware support)
- Saves memory (can store multiple states in 1 integer)
- Essential in:
  - Competitive programming
  - Embedded systems
  - Cryptography
  - Game dev, OS internals

02

Bitwise Operators and Mask

Bitwise Operators

- **Operators**
  - OR (|)
  - AND (&)
  - XOR (^)
  - Complement (~)
  - Right Shift (>>)
  - Left Shift (<<)

Bitwise Operators

Bitwise AND (&) :

- Returns **1 only if both bits are 1**
- Used for **masking bits**, clearing flags

Truth Table:

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Example :- 5 & 3 = 0101 & 0011 = 0001 → 1

Bitwise Operators

Bitwise OR (|) :

- Returns **1 if atleast one bit is 1**
- Used for **setting bits**

Truth Table:

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Example :- 5 | 3 = 0101 | 0011 = 0111 → 7

Bitwise Operators

Bitwise XOR (^) :

- Returns **1 only if the bits are different**
- Used for **toggle** or finding differences

Truth Table:

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Example :- 5 ^ 3 = 0101 ^ 0011 = 0110 → 6

Bitwise Operators

Bitwise NOT (~) :

- **Inverts all bits**
- Unary operator

Truth Table:

A	~A
0	1

A (binary)	~A (binary)	A (decimal)	~A (decimal)
0000	1111	0	-1
0001	1110	1	-2
0010	1101	2	-3
0101	1010	5	-6

Bitwise Operators

Bitwise NOT (~) :

- **Inverts all bits**
- Unary operator

Truth Table:

A	~A
0	1

A (binary)	~A (binary)	A (decimal)	~A (decimal)
0000	1111	0	-1
0001	1110	1	-2
0010	1101	2	-3
0101	1010	5	-6

Bitwise Operators

Left Shift (<<)

- Shifts bits **left**, fills with 0s on the right
- Equivalent to num × 2^n

Shift Table:

Input	Shift	Output
0001	<< 1	0010
0011	<< 1	0110
0101	<< 2	10100

Example :- 5 << 1 = 0101 << 1 = 1010 → 10

Bitwise Operators

Right Shift (>>)

- Shifts bits **right**, discards rightmost bits
- Signed: fills with sign bit (Java/C++)

Shift Table:

Input	Shift	Output
0100	>> 1	0010
0110	>> 1	0011
1010	>> 1	1101 (signed shift)

Example :- 5 >> 1 = 0101 >> 1 = 0010 → 2

Masks

Masks (particular combination of bits) can be created and used in conjunction with the Bitwise Operators to:

- Set a particular bit
- Get a particular bit
- Clear a particular bit

✅ Set a particular bit

java — Copy — Edit

```java
int num = 5;         // 0101
int pos = 1;         // Set 1st bit
int mask = 1 << pos;
num = num | mask;    // result: 0101 | 0010 = 0111 → 7
```

🔍 Get a particular bit

java — Copy — Edit

```java
int num = 5;         // 0101
int pos = 2;
int mask = 1 << pos;
int bit = (num & mask) >> pos;  // result: 0101 & 0100 = 0100 → bit = 1
```

❌ Clear a particular bit

java — Copy — Edit

```java
int num = 5;         // 0101
int pos = 0;
int mask = ~(1 << pos);
num = num & mask;    // result: 0101 & 1110 = 0100 → 4
```

03

Applications of Bits

Applications of Bits

- Media Files (e.g., MP3, MP4, JPG use bit-level encoding)
- Networking (e.g., MP3, MP4, JPG use bit-level encoding)
- Security (e.g., AES, RSA, hashing)
- Gaming (e.g., Collision maps, status toggles, and item collections are stored as bits)
- Flag handling (Use bit masks to set, clear, and toggle flags efficiently)
- Compression (e.g., compressing large arrays)
- Optimization (Fast math operations like multiplication/division by powers of 2 (`x << n`, `x >> n`))
- Searching / Algorithms (e.g., Used in Bitmask DP, Trie-based search, and subset generation)

04

Activity

Activity Graph

- Single Number
- Find the Duplicate Number
- Counting Bits
- Number of 1 bits
- Missing Number
- Reverse Bits
- Add Binary