FRACTION CONVERSION

Decimal Fraction ↔ Binary Fraction

Decimal Fraction → Binary Fraction

Example: $(59.8125)_{10}$

1. Separate integer and fractional parts.
2. Integer part: $59_{10} = 111011_2$
3. Fractional part: 0.8125
4. Multiply the fraction repeatedly by 2:

   $0.8125 \times 2 = 1.625 \rightarrow 1$

   $0.625 \times 2 = 1.25 \rightarrow 1$

   $0.25 \times 2 = 0.5 \rightarrow 0$

   $0.5 \times 2 = 1.0 \rightarrow 1$

Therefore: $0.8125_{10} = 0.1101_2$

Final Answer: $(59.8125)_{10} = (111011.1101)_2$

Binary Fraction → Decimal Fraction

Example: $(0.0101)_2$

Use negative powers of 2 for the fractional positions:

$0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$

$= 0 + 0.25 + 0 + 0.0625$

$= 0.3125$

Final Answer: $(0.0101)_2 = (0.3125)_{10}$

Rule: digits after the binary point have weights 2^{-1}, 2^{-2}, 2^{-3}, ...

NUMBER SYSTEM & FRACTION CONVERSION

Octal Fraction → Binary • Bits & Bytes

Octal Fraction → Binary Fraction

Example: $(165.27)_8$ → $(?)_2$

Convert each octal digit into its 3-bit binary equivalent.

$1_8 = 001_2$ $6_8 = 110_2$ $5_8 = 101_2$

$2_8 = 010_2$ $7_8 = 111_2$

Integer part: $165_8 = 001\ 110\ 101_2$

Fractional part: $.27_8 = .010\ 111_2$

Final form: $(165.27)_8 = (001110101.010111)_2$

Rule: Every octal digit corresponds to exactly 3 binary bits.

Quick Octal → Binary Table

Octal digit Binary (3-bit)

0 → 000 1 → 001 2 → 010 3 → 011

4 → 100 5 → 101 6 → 110 7 → 111

Example: $27_8 \rightarrow 010\ 111_2$

For the fractional part, keep the binary point in the same position.

Bits, Nibbles & Bytes

Bit: Binary digit — 0 or 1.

Nibble: 4 bits.

Byte: 8 bits.

Therefore: 1 Byte = 2 Nibbles = 8 Bits.

Example of 1 byte: 10101100

A byte can represent one character in many common character encodings.

Storage Units

1 Byte = 8 Bits

1 KB = 1024 Bytes

1 MB = 1024 KB

1 GB = 1024 MB

1 TB = 1024 GB

1 PB = 1024 TB

1 EB = 1024 PB

BINARY OPERATIONS

Binary Addition • Rules • Examples

Binary Addition – Basic Rules

Binary addition uses only two digits: 0 and 1.

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 with Carry 1

Carry is transferred to the next higher bit.

Alternative method: convert binary numbers to decimal, add them, then convert the result back to binary.

Example 1: Binary Addition

Example: $(1011)_2 + (011)_2$

Align the numbers from the right:

```
  1011
+ 0011
------
  1110
```

Answer: $(1110)_2$

Example 2: Binary Addition

Example: $(110101)_2 + (100010)_2$

```
  110101
+ 100010
--------
 1010111
```

Answer: $(1010111)_2$

Example 3: Binary Addition

Example: $(101101)_2 + (110011)_2$

```
 101101
+ 110011
--------
 1100000
```

Answer: $(1100000)_2$

Binary Fraction Addition

Example: $(1011.01)_2 + (1100.11)_2$

Align the binary points:

```
  1011.01
+ 1100.11
---------
 11000.00
```

Answer: $(11000.00)_2$

BINARY SUBTRACTION & MULTIPLICATION

Rules • Borrow Method • Step-by-Step Examples

Binary Subtraction – Basic Rules

Binary subtraction uses only 0 and 1.

$0 - 0 = 0$

$1 - 0 = 1$

$1 - 1 = 0$

$0 - 1$ requires borrowing from the next higher bit.

After borrowing: $10_2 - 1_2 = 1_2$

Remember: borrowing 1 from the left gives 10_2 (decimal 2) in the current position.

Example: Binary Subtraction

Example: $(11010)_2 - (10001)_2$

```
  11010
- 10001
--------
  01001
```

Final Answer: $(11010)_2 - (10001)_2 = (01001)_2$

Binary Subtraction – Borrow Method

Example: $(10101)_2 - (00111)_2$

Align both numbers from the right.

When 0 − 1 occurs, borrow 1 from the next available higher bit.

```
  10101
− 00111
--------
  01110
```

Final Answer: $(10101)_2 - (00111)_2 = (01110)_2$

Binary Multiplication – Basic Rules

Binary multiplication follows simple rules:

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

For each next multiplier bit, shift the partial product one position to the left.

Example: Binary Multiplication

Example: $(101)_2 \times (11)_2$

```
  101
×  11
-----
  101
 1010
-----
 1111
```

Final Answer: $(101)_2 \times (11)_2 = (1111)_2$

Binary Multiplication – Another Example

Example: $(1101)_2 \times (101)_2$

Partial products are written according to each multiplier bit.

```
  1101
×  101
------
  1101
 0000
110100
------
1000001
```

Final Answer: $(1101)_2 \times (101)_2 = (1000001)_2$

Quick Revision

Subtraction: 0−0=0, 1−0=1, 1−1=0, and 0−1 requires borrowing.

Multiplication: 1×1=1; all products involving 0 are 0.

Always align binary digits correctly.

In multiplication, shift partial products left for each new position.

Practice with step-by-step borrowing and carrying.

SIGNED BINARY REPRESENTATION & COMPLEMENTS

Sign-Magnitude • 1’s Complement • 2’s Complement • 9’s Complement

1. Sign-Magnitude Representation

Sign-magnitude representation uses the first (most significant) bit as the sign bit.

Sign bit = 0 → Positive (+) number

Sign bit = 1 → Negative (−) number

The remaining bits represent the magnitude (absolute value).

Example for 6-bit representation: 0 | 01010 → +10; 1 | 01010 → −10

Format: [Sign Bit] [Magnitude Bits]

2. 1’s Complement

1’s complement is obtained by changing every 0 to 1 and every 1 to 0.

Rule: 0 → 1

Rule: 1 → 0

Example: 101101 → 010010

Therefore, the 1’s complement of 101101 is 010010.

1’s complement is commonly used to represent negative binary numbers.

3. 2’s Complement

2’s complement is obtained by taking the 1’s complement and adding 1.

Step 1: Find the 1’s complement.

Step 2: Add 1 to the result.

Example: 010010 (1’s complement) + 1 = 010011

Therefore, the 2’s complement of 101101 is 010011.

Shortcut: From the right, keep bits unchanged up to the first 1; complement all bits to its left.

4. 8’s Complement

8’s complement is used with octal numbers.

Rule: Subtract each digit from 7. **after add 1.**

Example: Find the 8’s complement of 2107.

7 – 2 = 5, 7– 1 = 6, 7 – 0 = 7, 7 – 7 = 0

Therefore: 7’s complement of 2107 = 5671.

Use 9’s complement as a method for decimal subtraction.

4. 9’s Complement

9’s complement is used with decimal numbers.

Rule: Subtract each digit from 9.

Example: Find the 9’s complement of 2107.

9 − 2 = 7, 9 − 1 = 8, 9 − 0 = 9, 9 − 7 = 2

Therefore: 9’s complement of 2107 = 7892.

Use 9’s complement as a method for decimal subtraction.

Quick Comparison

Sign-Magnitude → Separate sign bit + magnitude bits.

1’s Complement → Invert every binary bit.

2’s Complement → 1’s complement + 1.

9’s Complement → Subtract each decimal digit from 9.

Key point: 1’s and 2’s complements are binary concepts; 9’s complement is a decimal complement method.

BINARY CODES

BCD • Excess-3 • Gray Code • Non-Binary • Alphanumeric

24. Logic Gates

A Logic Gate is an electronic circuit that performs a logical operation on binary inputs and produces a binary output.

- 0 = False / LOW
- 1 = True / HIGH

Main gates: AND, OR, NOT, NAND, NOR and XOR.

24. Logic Gates — AND Gate

Function: Output is 1 only when ALL inputs are 1.

Easy Rule: AND → All inputs must be 1

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

24. Logic Gates — OR Gate

Function: Output is 1 when AT LEAST ONE input is 1.

Easy Rule: OR → Any one input can be 1

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

24. Logic Gates — NOT Gate

Function: NOT Gate inverts (reverses) the input.

Easy Rule: NOT → Reverse

Input	Output
0	1
1	0

24. Logic Gates — NAND Gate

Function: NAND = NOT + AND. It is the inverse of AND.

Easy Rule: NAND → Opposite of AND

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

24. Logic Gates — NOR Gate

Function: NOR = NOT + OR. It is the inverse of OR.

Easy Rule: NOR → Opposite of OR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

24. Logic Gates — XOR Gate

Function: XOR = Exclusive OR. Output is 1 when inputs are DIFFERENT.

Easy Rule: XOR → Different = 1, Same = 0

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

25. Binary Codes

A Binary Code is a method of representing data or characters using binary digits (0 and 1).

Important binary codes:

- BCD (Binary Coded Decimal)
- Excess-3 Code
- Gray Code

25. Binary Codes — BCD

BCD = Binary Coded Decimal

• Each decimal digit is represented separately using 4 bits.

Example: 732_{10}

7 → 0111 3 → 0011 2 → 0010

Therefore: 732 → 0111 0011 0010

Important: BCD converts EACH decimal digit separately.

26. Excess-3 Code

- Add 3 to each decimal digit and then convert the result into 4-bit binary.

Example: 732

7 + 3 = 10 → 1010

3 + 3 = 6 → 0110

2 + 3 = 5 → 0101

Therefore: 732 → 1010 0110 0101

Easy Rule: Add 3 to every decimal digit → convert to 4-bit binary.

27. Gray Code

- Gray Code is a non-weighted binary code.
- Also called Reflected Binary Code.
- XOR operation is used in Binary → Gray conversion.

27. Gray → Binary Code — Example 11101_2

Rule 1: First Binary bit = First Gray bit.

Rule 2: Every next Binary bit = Previous Binary bit XOR Current Gray bit.

Gray: 1 1 1 0 1

Binary: 1 0 1 1 0

Calculation: 1 | (1⊕1) | (0⊕1) | (1⊕0) | (1⊕1)

Answer: (11101)Gray = $(10110)_2$

27. Binary → Gray Code — Example 10110_2

Rule 1: First Gray bit = First Binary bit.

Rule 2: Every next Gray bit = XOR of two adjacent Binary bits.

Binary: 1 0 1 1 0

Gray: 1 1 1 0 1

Calculation: 1 | (1⊕0) | (0⊕1) | (1⊕1) | (1⊕0)

Answer: $(10110)_2$ = (11101)Gray

28. Classification of Binary Codes

1. Binary Codes: BCD, Excess-3, Gray Code
2. Non-Binary Codes: Morse Code
3. Alphanumeric Codes: ASCII, EBCDIC, International Cod

Characters → Letters + Numbers + Symbols

⬇️

Character Coding System → ASCII / EBCDIC etc.

⬇️

Computer representation → Binary

29. Morse Code

- Morse Code is a communication coding system.
- It represents letters, numbers and symbols using dots (.) and dashes (−).

Example: A = .−

Example: SOS = ... --- ...

Easy Definition: Morse Code represents information using dots and dashes.

- It is listed as a Non-Binary Code in the given classification.
- pre-defined coding system.
- international distress/emergency signal

Character	Morse Code	Small	Morse Code	Number	Morse Code
A	.−	**a**	.−	**0**	−−−−−
B	−...	**b**	−...	**1**	.−−−−
C	−.−.	**c**	−.−.	**2**	..−−−
D	−..	**d**	−..	**3**	...−−
E	.	**e**	.	**4**	....−
				5	.....

30. Alphanumeric Codes

Alphanumeric = Alpha + Numeric

- Alpha = Alphabet (A, B, C, ...)
- Numeric = Numbers (0, 1, 2, ...)
- Alphanumeric codes represent letters, numbers and commonly used symbols.

Important coding systems: ASCII, EBCDIC and International Code.

30. ASCII

ASCII = American Standard Code for Information Interchange

- Character coding system.
- Original ASCII uses 7 bits.
- 7 bits → 2^7 = 128 possible code combinations.

Important values: A = 65, B = 66, a = 97, 0 = 48, 1 = 49 , Space = 32

Example: A → 65 → 1000001 (binary)

Important: 7-bit tells the code length; ASCII defines the character-coding rules.

31. EBCDIC

EBCDIC = Extended Binary Coded Decimal Interchange Code

- Character coding system.
- Uses 8 bits.
- 8 bits → 2^8 = 256 possible code combinations.
- Associated with IBM mainframe systems.

Important: 8-bit tells the code length; EBCDIC defines the coding rules used to represent characters.

31. International Code

According to the provided material:

- International Code is listed as an 8-bit alphanumeric code.
- 8 bits → 2^8 = 256 possible combinations.
- The provided material does not give a full form or detailed character table for this item.

31. ASCII vs EBCDIC

ASCII → 7-bit → 128 possible combinations

EBCDIC → 8-bit → 256 possible combinations → IBM mainframe systems

Both are character coding systems, but they use different coding standards.

31. Bits and Possible Codes

1 bit → 2^1 = 2 combinations

2 bits → 2^2 = 4 combinations

3 bits → 2^3 = 8 combinations

4 bits → 2^4 = 16 combinations

7 bits → 2^7 = 128 combinations

8 bits → 2^8 = 256 combinations

Formula: n bits → 2^n possible combinations

24–31. Quick Revision

Logic Gates: AND → All 1 | OR → Any 1 | NOT → Reverse | NAND → Opposite of AND | NOR → Opposite of OR | XOR → Different = 1

Binary Codes: BCD → each decimal digit uses 4 bits | Excess-3 → add 3 to each digit | Gray → consecutive values differ by 1 bit

Classification: Binary → BCD, Excess-3, Gray | Non-Binary → Morse | Alphanumeric → ASCII, EBCDIC, International Code

ASCII → 7-bit → 128

EBCDIC → 8-bit → 256 → IBM

International Code → 8-bit (as listed in the provided material)

Formula: n bits → 2^n possible combinations