Lesson 3: Data Representation & Logic

1. Number Representation Concepts

Core Ideas

- **Why different systems?** Binary is for computers; Octal & Hex are human-friendly shortcuts for long binary strings.
- **MSB/LSB:** Most/Least Significant Bit. The bit with the highest/lowest place value.

Signed Integers (Representing +/-)

- **Sign-Magnitude:** Left-most bit is the sign (0=+, 1=-). Simple, but has two zeros (+0, -0) and makes arithmetic complex.
- **One's Complement:** Negate by flipping all bits (0s to 1s, 1s to 0s). Still has two zeros.
- **Two's Complement:** Flip all bits + 1. The standard for computers because it has only one zero and makes subtraction hardware simple (subtraction becomes addition).

Decimal Number Representation

- **Fixed-point:** For numbers with a fixed number of decimal places (e.g., currency $123.45). Simple & fast. Limited range.
- **Floating-point:** For scientific numbers (very large/small). More flexible range, but more complex.
  - **Structure:** Sign bit | Exponent | Mantissa.
  - **IEEE 754 Standard:**
    - → **Single Precision (32-bit):** 1 Sign, 8 Exp, 23 Man.
    - → **Double Precision (64-bit):** 1 Sign, 11 Exp, 52 Man.

2. Character Representation

Character Codes & Comparison

- **BCD (Binary Coded Decimal):**
  - Represents only numbers 0-9 (4-bit).
  - *Pro: Easy decimal conversion. Con: Wastes space.*
- **ASCII (American Standard...):**
  - 7-bit (128 chars). Standard for English & PCs.
  - *Pro: Widely compatible. Con: Limited characters.*
- **EBCDIC (Extended BCD...):**
  - 8-bit (256 chars). Used mainly by IBM Mainframes. Not compatible with ASCII's letter ordering.
- **Unicode:**
  - 16/32-bit. Represents all world languages.
  - *Pro: Universal standard. Con: Uses more memory than ASCII.*

3. Arithmetic & Logic

Binary Arithmetic

- **Addition Rules:** `0+0=0, 0+1=1, 1+1=0 carry 1.`
- **Subtraction Rules:** `1-1=0, 1-0=1, 0-0=0, 0-1=1 borrow 1.`

Bitwise Logic Operations

- **NOT:** Inverts bits (`NOT 1100 -> 0011`).
- **AND:** Masks bits (keeps bits set in **both**).
- **OR:** Sets bits (keeps bits set in **either**).
- **XOR:** Toggles bits (keeps bits that **differ**).

4. Number Conversion Examples

Decimal → Binary (Integer)

Rule: Divide by 2, read remainders up.
Ex: Convert 43_{10} **to Binary**

```
43 / 2 = 21 R 1   ^
21 / 2 = 10 R 1   |
10 / 2 =  5 R 0   |
 5 / 2 =  2 R 1   |
 2 / 2 =  1 R 0   |
 1 / 2 =  0 R 1   |
Ans: 101011_2
```

Binary → Decimal

Rule: Use place values ($...16, 8, 4, 2, 1$).
Ex: Convert 101011_2 **to Decimal**

```
  1   0   1   0   1   1
  x   x   x   x   x   x
 32 + 0 + 8 + 0 + 2 + 1 = 43_10
```

Binary ↔ Octal & Hex

Rule: Group bits from right (3 for Oct, 4 for Hex).

- **Binary to Octal:** 101110_2
  - `101 | 110` → `5 | 6` → 56_8
- **Binary to Hex:** 10111110_2
  - `1011 | 1110` → `B | E` → BE_{16}
- **Hex to Binary:** $2\mathbf{A}_16$
  - `2 | A` → `0010 | 1010` → 00101010_2

Decimal → Binary (Fraction)

Rule: Multiply fraction by 2, read integer parts down.
Ex: Convert 0.8125_{10} **to Binary**

```
0.8125 * 2 = 1.625  (1) |
0.625  * 2 = 1.25   (1) |
0.25   * 2 = 0.5    (0) v
0.5    * 2 = 1.0    (1)
Ans: 0.1101_2
```

2's Complement Example: -45 in 8-bit

1. **Positive (+45):** `00101101`
2. **One's Complement:** `11010010` (Flip all bits)
3. **Two's Complement:** `11010011` (Add 1 to result)