TECHNO MAIN SALT LAKE

Name : **Shreyash Lal**

Dept: **CSE(AI & ML) - A**

University roll: **13030824012**

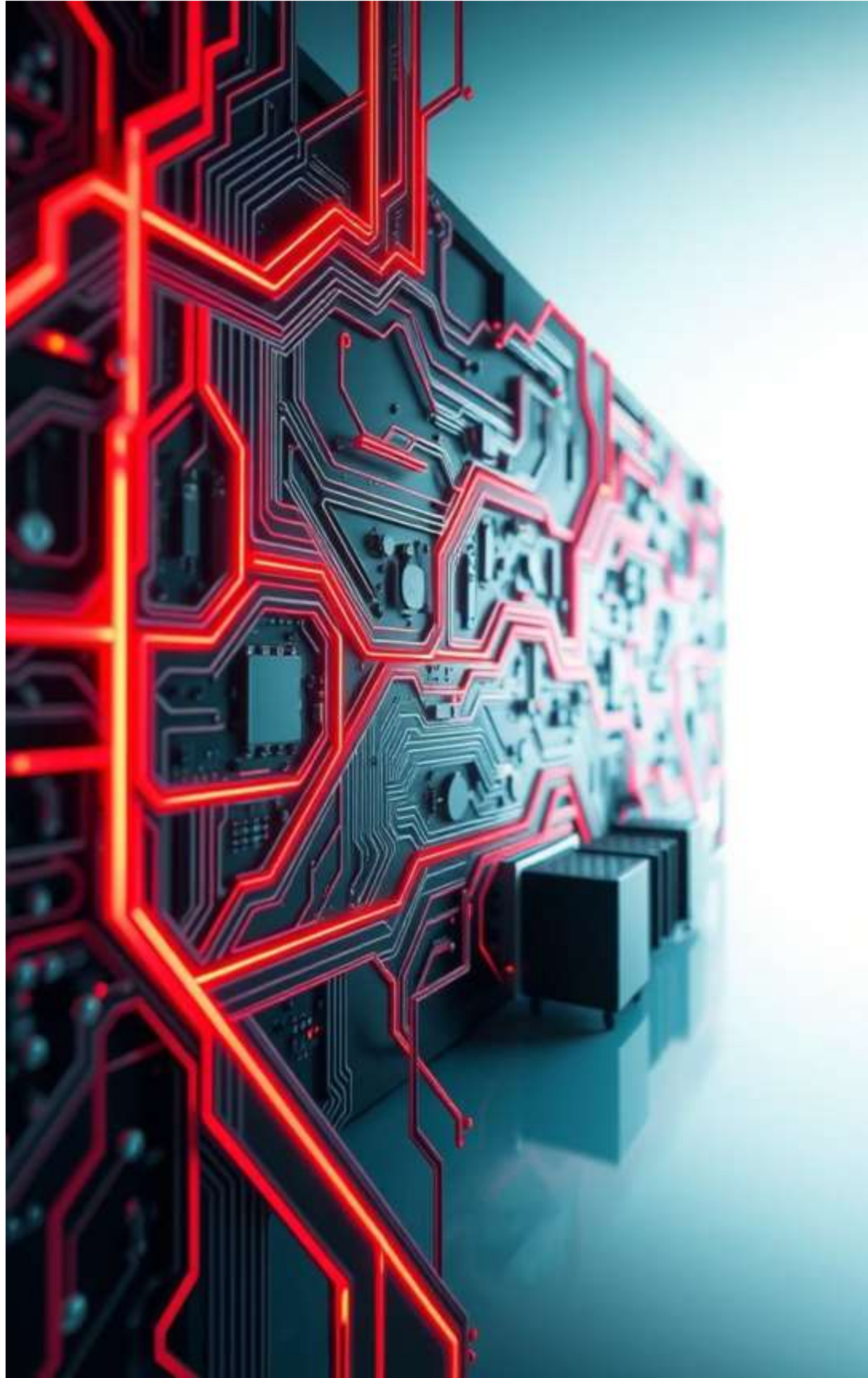
Paper Name: **Computer Organization and Architecture**

Paper Code: **PCC-CS301**

Sem: **3rd**

Unveiling the Architecture: Instruction Formats & Number Systems

Welcome to an exploration of the fundamental building blocks of computing: instruction formats and number systems. In this presentation, we will demystify how computers understand and execute commands, and how different number bases underpin all digital operations.



Topic 1: Instruction Formats

The Language of Computers: Instruction Formats

Instruction formats define the structure of the machine code instructions that a computer's CPU can understand and execute. They specify the opcode (operation code) and the operands (data or addresses) needed for that operation. Understanding these formats is crucial for efficient computer architecture design.

Opcode

Specifies the operation to be performed (e.g., ADD, LOAD, STORE).

Operands

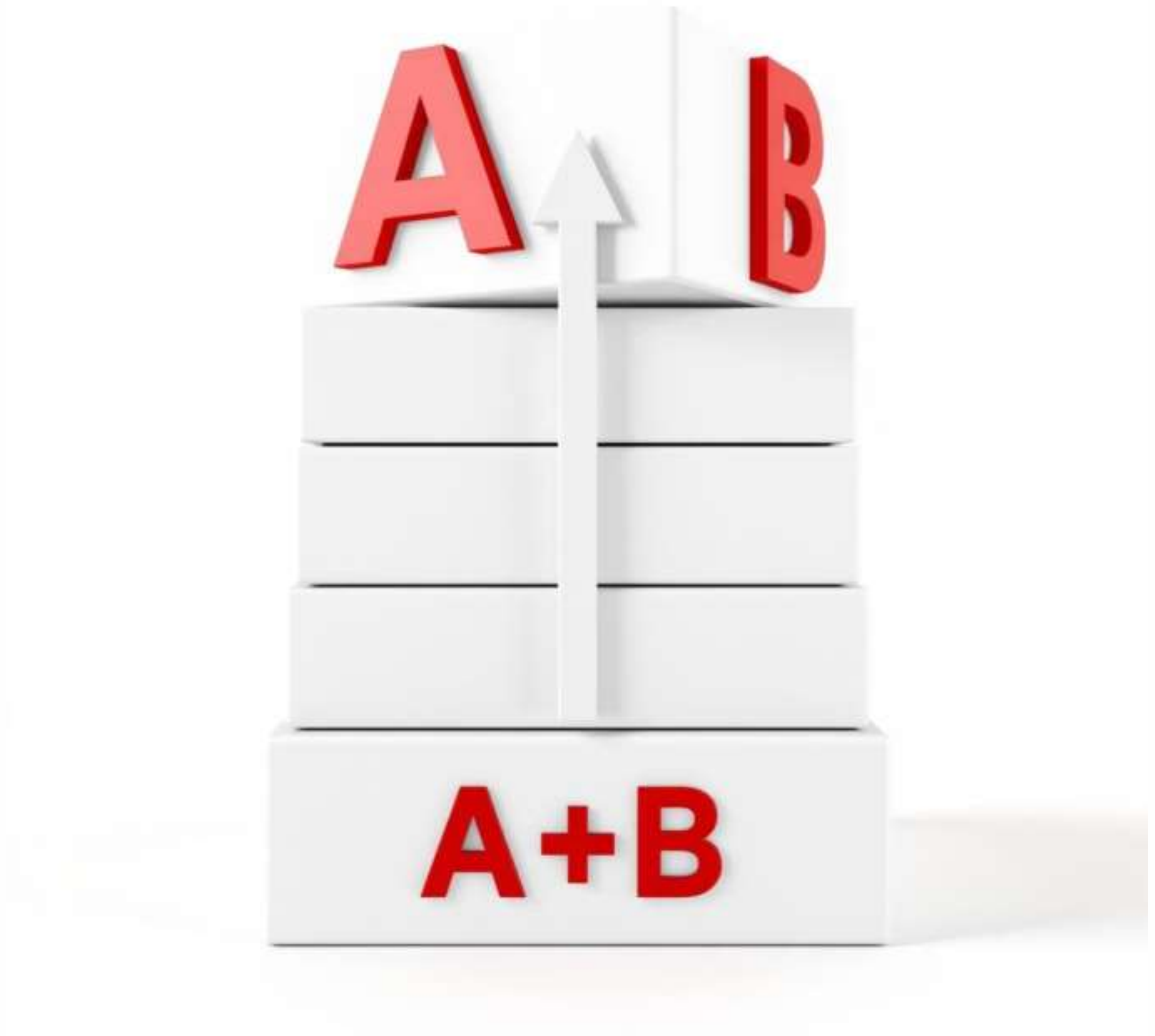
The data or memory addresses on which the operation will act.

Zero-Address Instruction Format

In a zero-address instruction format, the locations of operands are implied, typically by a stack-based architecture. Operations are performed on the top elements of the stack. This design minimizes instruction length but requires more operations for complex tasks.

Example (Pseudo-code):

```
PUSH APUSH BADD  // Operates on B and A, result stored on stackPOP C
```

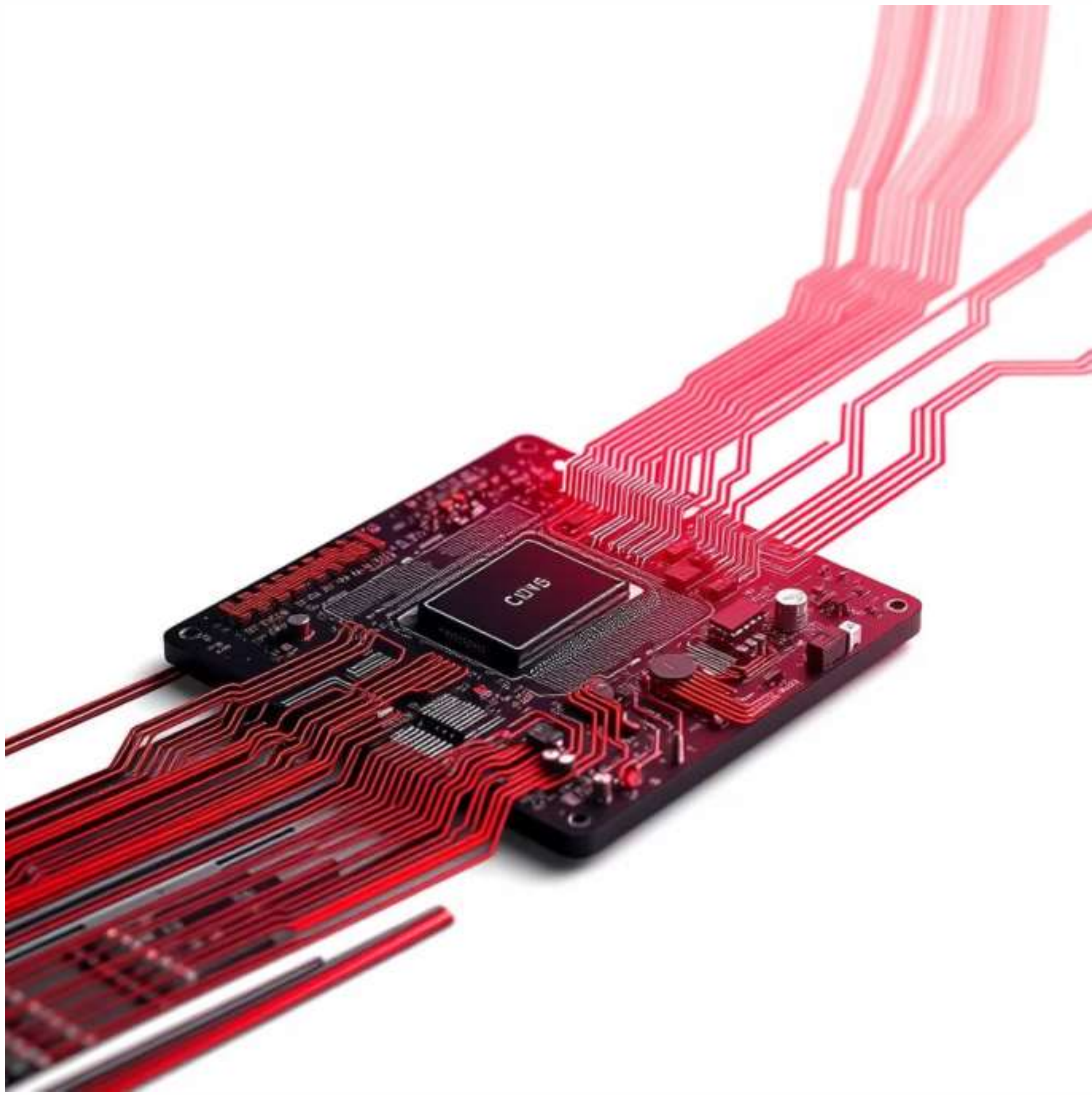


One-Address Instruction Format

The one-address instruction format, also known as accumulator-based, uses an implicit accumulator register for one of the operands and to store the result. This simplifies instruction decoding but can be inefficient for complex expressions as intermediate results must be constantly moved to and from memory.

Example (Pseudo-code):

```
LOAD A    // ACC = AADD B     // ACC = ACC + BSTORE C   // C = ACC
```



Two and Three-Address Instruction Formats

Two-Address Format:

One of the operands also serves as the destination for the result, reducing instruction length compared to three-address. Common in RISC architectures.

```
ADD R1, R2   // R1 = R1 + R2
```

Three-Address Format:

All operands (two source, one destination) are explicitly specified. This results in longer instructions but offers greater flexibility and reduces the need for temporary storage.

```
ADD R1, R2, R3 // R1 = R2 + R3
```

Two-Address

Topic 2: Number Systems

The Foundation: Base (Radix) in Number Systems

In any positional numeral system, the base or radix defines the number of unique digits (including zero) used to represent numbers. It's the core of how we count and perform arithmetic. Different bases are optimized for various applications, especially in digital electronics.

Common Number Systems and Their Bases

Binary (Base 2)

Uses only two digits: 0 and 1. This is the native language of computers because it directly maps to electrical states (on/off, high/low voltage). All digital logic is built upon binary.

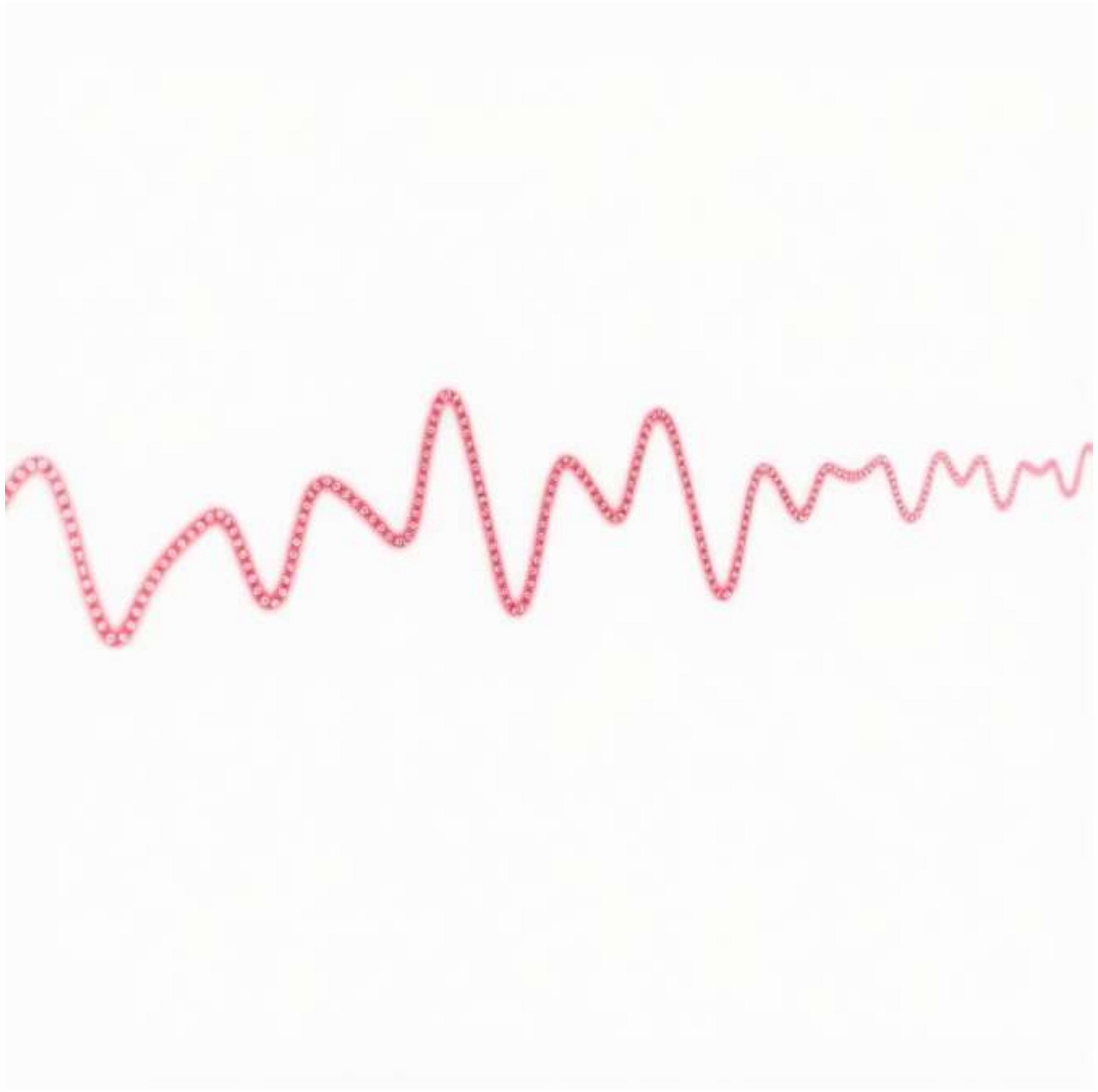
Decimal (Base 10)

Our everyday number system, using digits 0-9. It's intuitive for humans due to our ten fingers. While not directly used by hardware, it's how we interpret computer outputs.

Hexadecimal (Base 16)

Uses digits 0-9 and letters A-F (representing 10-15). It's a compact way to represent binary data, as one hex digit corresponds to four binary digits (a nibble). Crucial for memory addressing and color codes.

Why Base Matters in Digital Systems



Hardware Implementation

Binary (base 2) directly translates to the two stable states (voltage levels) of electronic switches, making it the most efficient for digital circuit design.

Data Representation

Understanding bases allows conversion between human-readable (decimal) and machine-readable (binary, hexadecimal) formats, essential for programming and debugging.

Efficiency and Abstraction

Higher bases like hexadecimal provide a more concise and less error-prone way to represent large binary numbers, improving code readability for engineers.

Conclusion & Next Steps

We've explored how instruction formats dictate the fundamental operations of a CPU, from the concise stack-based approach to the explicit three-address system. We also highlighted the crucial role of number bases, particularly binary and hexadecimal, in the very fabric of digital electronics.

> Understanding these foundational concepts is key to designing, programming, and troubleshooting modern computer systems.

Further Exploration:

- Investigate different CPU architectures (e.g., RISC vs. CISC).
- Practice number base conversions for complex values.
- Explore assembly language programming to see instruction formats in action.

Thank You!

We appreciate your time and attention today. We hope this presentation has provided valuable insights into the fascinating world of computer architecture, from instruction formats to the fundamental number systems that underpin all digital operations.

Feel free to reach out with any further questions.