

CPU Organization and Architecture

A comprehensive exploration of central processing unit structure, operation, and design principles for computer engineering students

Course Overview

O1

CPU Structure

Internal organization and functional components

O2

ALU Operations

Arithmetic and logical processing capabilities

O3

Memory Organization

Stack structures and register systems

O4

Instruction Design

Formats, addressing, and data manipulation

O5

Architecture Models

RISC versus CISC design philosophies

This presentation examines the fundamental concepts of computer organization, focusing on how the CPU orchestrates computation through its interconnected components. We'll explore both theoretical foundations and practical implementations that define modern processor design.

CPU Structure and Function

Core Components

The CPU consists of three primary functional units working in concert:

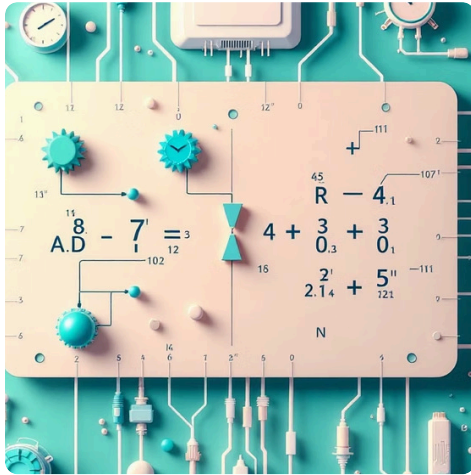
- Control Unit (CU): Orchestrates instruction execution
- Arithmetic Logic Unit (ALU): Performs computations
- Registers: High-speed temporary storage

Execution Cycle

The CPU operates through a continuous fetch-decode-execute cycle. During each cycle, instructions are retrieved from memory, interpreted by the control unit, and executed using appropriate functional units.

Internal buses facilitate communication between components, transferring data, addresses, and control signals. This interconnected architecture enables the processor to coordinate complex operations while maintaining synchronization through clock signals.

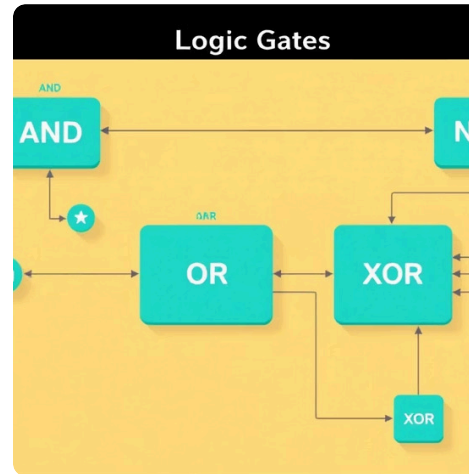
Arithmetic and Logic Unit



Arithmetic Operations

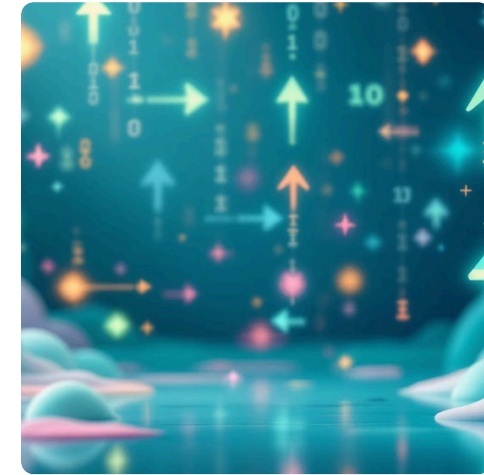
The ALU performs integer arithmetic including addition, subtraction, multiplication, and division. Fixed-point and floating-point circuits handle different numeric representations.

The ALU represents the computational heart of the processor, executing mathematical and logical operations on binary data. Status flags capture operation results, indicating conditions like zero, negative, overflow, and carry for subsequent decision logic.



Logical Operations

Boolean operations (AND, OR, NOT, XOR) enable bit manipulation and comparison operations. These form the foundation for decision-making in program execution.



Shift Operations

Logical and arithmetic shifts move bits left or right, enabling efficient multiplication, division, and bit-field manipulation without complex circuitry.

Stack Organization



Last-In-First-Out Structure

The stack implements a LIFO data structure crucial for managing subroutine calls, local variables, and expression evaluation. A dedicated stack pointer register tracks the current top position.

Key Operations:

- **PUSH:** Decrements stack pointer and stores data
- **POP:** Retrieves data and increments stack pointer
- **PEEK:** Reads top without modification

Stack-based architectures simplify instruction sets by using implicit operand locations. When a function calls another function, the return address is pushed onto the stack, enabling proper program flow upon completion. Local variables and saved registers also utilize stack storage, creating stack frames for each active subroutine.

Processor and Register Organization

General Purpose Registers

Fast storage for operands and intermediate results. Typically 8-32 registers providing quick access without memory latency.

Special Purpose Registers

- Program Counter (PC)
- Instruction Register (IR)
- Memory Address Register (MAR)
- Memory Data Register (MDR)

Status Registers

Flags register stores condition codes: zero, sign, carry, overflow, parity. Essential for conditional branching and error detection.

Register organization significantly impacts performance. More registers reduce memory access frequency, while specialized registers optimize specific operations. The register file connects directly to the ALU through dedicated paths, enabling single-cycle operations.

Instruction Formats



Instruction encoding balances expressiveness against code density. Different formats trade off operand specification flexibility for instruction length efficiency.

Fixed-Length

All instructions occupy the same number of bits. Simplifies fetch and decode logic but may waste space or limit operand fields.

Variable-Length

Instructions span different byte counts based on complexity. Maximizes code density but requires more complex decoding circuitry.

Each instruction contains an opcode specifying the operation plus fields for operands and addressing modes. The format determines how many explicit operands can be specified versus implicit registers like accumulators.

Addressing Modes



Immediate Addressing

Operand value contained directly in instruction. Fast but limited to constants. Example: `MOV R1, #25`



Direct/Absolute

Instruction contains memory address of operand. Simple but inflexible. Example: `LOAD R1, [1000]`



Register Indirect

Register holds address of operand. Enables pointer manipulation. Example: `LOAD R1, [R2]`



Indexed/Displacement

Combines base address with offset for array access. Example: `LOAD R1, [R2+4]`

Addressing modes provide flexibility in how operands are located, enabling efficient implementation of data structures like arrays, stacks, and linked lists.

Data Transfer and Manipulation

Transfer Instructions

Data movement forms the foundation of program execution, copying information between registers, memory, and I/O devices.

- **Register-to-Register:** Fastest, single cycle
- **Memory-to-Register:** Load operations (LOAD, LDR)
- **Register-to-Memory:** Store operations (STORE, STR)
- **I/O Transfers:** IN and OUT instructions

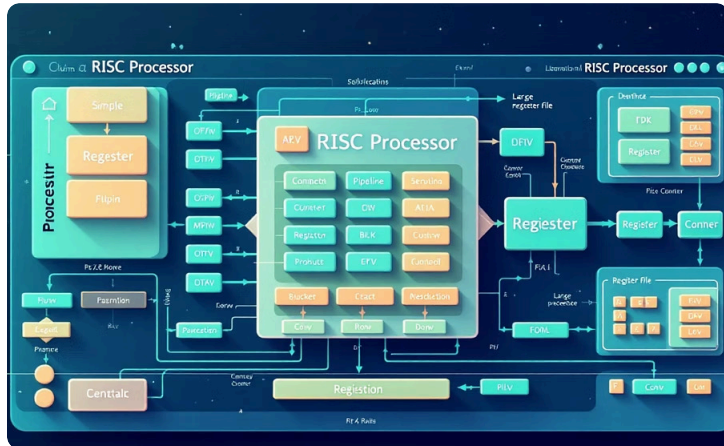
Efficient data manipulation requires minimizing memory accesses by keeping frequently used values in registers. The instruction set architecture defines available operations and their encoding.

Manipulation Operations

Beyond transfers, CPUs provide instructions for transforming data:

- Arithmetic: ADD, SUB, MUL, DIV
- Logical: AND, OR, XOR, NOT
- Shift/Rotate: SHL, SHR, ROL, ROR
- Comparison: CMP, TEST

RISC vs CISC Architectures

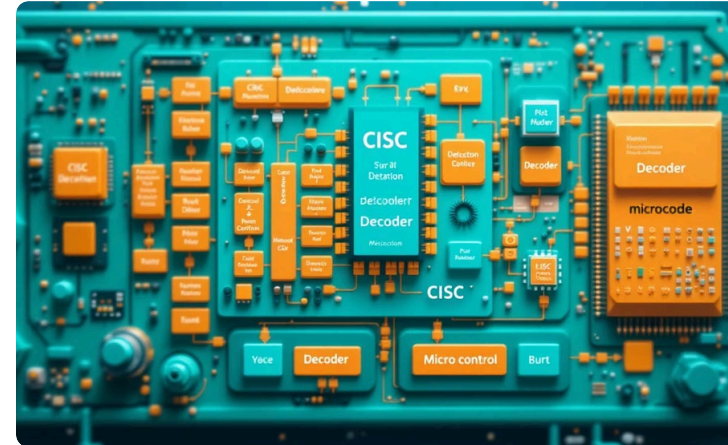


RISC Design Philosophy

Reduced Instruction Set Computing emphasizes simplicity and speed through uniform instruction length, load-store architecture, and extensive pipelining.

- Simple, fixed-length instructions
- Large register file (32+ registers)
- Single-cycle execution
- Example: ARM, MIPS, RISC-V

Modern processors blur these distinctions, with CISC architectures using RISC-like execution cores and microcode translation. The choice impacts compiler design, power consumption, and performance characteristics for specific workloads.



CISC Design Philosophy

Complex Instruction Set Computing provides rich instructions that accomplish more per instruction, reducing program size at the cost of implementation complexity.

- Variable-length instructions
- Memory operands in instructions
- Multi-cycle operations
- Example: x86, VAX