

**COMSATS UNIVERSITY ABBOTABAD DEPARTMENT OF**

**ELECTRICAL AND COMPUTER ENGINEERING**

Submitted By:

|  |  |
| --- | --- |
| NAME | Registration |
| Azhar Ali | FA22-BCE-088 |

[**Microprocessor Systems and Interfacing**](javascript:__doPostBack('ctl00$DataContent$gvCourseSummary$ctl05$lbCourse',''))

**ASSIGNMENT 1**

Course Instructor: Muhammad Naveed Shaikh

Submission Date: 14-04-2025

**Q.1)**

**A)** Describe the difference between RISC and CISC Architecture.

The main differences between **RISC (Reduced Instruction Set Computer)** and **CISC (Complex Instruction Set Computer)** architectures lie in their instruction set design, performance, and hardware complexity. Below is a comparison:

### **1. Instruction Set Complexity**

**RISC**:

* + Uses a **small, fixed set of simple instructions**.
  + Each instruction performs a single operation (e.g., load, store, add).
  + Requires more instructions to perform complex tasks.

**CISC**:

* + Uses a **large, variable set of complex instructions**.
  + A single instruction can perform multiple operations (e.g., a single instruction may load, compute, and store data).
  + Fewer instructions needed for complex tasks.

### **2. Execution Time**

**RISC**:

* + Most instructions execute in **one clock cycle** due to pipelining.
  + Faster for simple, repetitive tasks.

**CISC**:

* + Instructions take **multiple clock cycles** to execute.
  + Slower per instruction but may require fewer instructions overall.

### **3. Hardware Complexity**

**RISC**:

* + Simpler hardware design.
  + Fewer transistors, leading to lower power consumption.
  + Relies on software (compiler) for optimization.

**CISC**:

* + More complex hardware (microcode, dedicated circuits).
  + Higher transistor count, leading to higher power consumption.
  + Hardware handles instruction complexity.

### **4. Memory Access**

**RISC**:

* + Follows **load-store architecture** (only load/store instructions access memory).
  + Arithmetic/logic operations work only on registers.

**CISC**:

* + Allows **memory-to-memory operations** (arithmetic/logic operations can directly access memory).

### **5. Code Density**

**RISC**:

* + Lower code density (more instructions needed → larger program size).

**CISC**:

* + Higher code density (fewer instructions → smaller program size).

### **6. Examples**

**RISC Processors**: ARM, MIPS, RISC-V, Apple M-series, modern Intel/AMD (hybrid approach).

**CISC Processors**: x86 (Intel, AMD), older architectures like VAX, Motorola 68000.

### **7. Use Cases**

**RISC**:

* + Common in embedded systems, mobile devices (smartphones), and high-performance computing (due to efficiency).

**CISC**:

* + Dominates desktop/laptop CPUs (backward compatibility with x86).

### **Modern Trends**

* Many modern processors use a **hybrid approach** (e.g., x86 processors translate CISC instructions into RISC-like micro-ops internally).

**RISC-V** is gaining popularity as an open-source RISC architecture.

**B)** Describe the difference between Harvard and Von-Neuman Architecture.

The **Harvard** and **Von Neumann** architectures are two fundamental designs for computer systems, differing primarily in how they handle **memory organization** and **data/instruction access**. Below is a detailed comparison:

### **1. Memory Organization**

* **Von Neumann Architecture**:
  + **Single shared memory** for both **instructions (code)** and **data**.
  + Uses a **common bus** to fetch instructions and data.
  + Simpler design but can lead to bottlenecks (the "Von Neumann bottleneck").
* **Harvard Architecture**:
  + **Separate memories** for **instructions** and **data**.
  + Uses **two independent buses**, allowing simultaneous instruction and data access.
  + More complex but avoids bottlenecks.

### **2. Performance & Speed**

### **Von Neumann**:

* + Slower because instructions and data compete for the same bus (**sequential access**).
  + Example: A CPU must wait to fetch data after fetching an instruction.
* **Harvard**:
  + Faster due to **parallel access** (fetching an instruction while reading/writing data simultaneously).
  + Common in high-speed embedded systems (e.g., DSPs, microcontrollers).

### **3. Hardware Complexity**

* **Von Neumann**:
  + Simpler hardware (only one memory bus).
  + Cheaper to implement.
* **Harvard**:
  + More complex (dual memory buses, separate caches).
  + Higher cost but better performance.

### **4. Flexibility**

* **Von Neumann**:
  + More flexible; program code can be modified like data (useful for self-modifying code).
* **Harvard**:
  + Less flexible since instruction memory is separate (prevents self-modifying code unless explicitly supported).

### **5. Modified Harvard Architecture**

Modern processors (e.g., x86, ARM) often use a **modified Harvard architecture**:

* **Separate caches** for instructions (L1i) and data (L1d).
* **Unified main memory** (like Von Neumann).
* Combines speed benefits of Harvard with flexibility of Von Neumann.