**Batch: C1 Roll No.: 16010122236**

**Experiment / assignment / tutorial No.: 09**

**TITLE:** Study of RISC and CISC Architecture

**AIM:** Understanding RISC and CISC Architecture

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment: (Mentions the CO/CO’s attained)**

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, ZvonkoVranesic and SafwatZaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. William Stallings, “Computer Organization and Architecture: Designing for Performance”, Eighth Edition, Pearson.
3. Dr. M. Usha, T. S. Srikanth, “Computer System Architecture and Organization”, First Edition, Wiley-India.

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Pre Lab/ Prior Concepts:**

**Reduced Instruction Set Architecture ( RISC ) :**

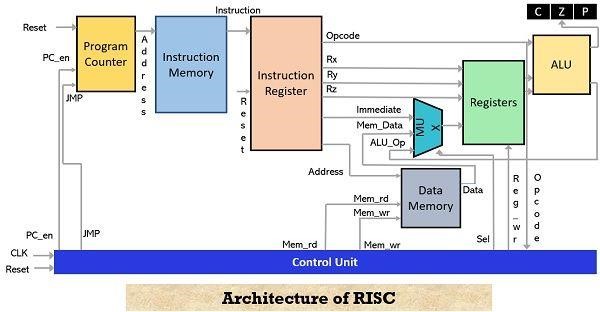
The main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating and storing operations just like a load command will load data, store command will store the data.

**Complex Instruction Set Architecture (CISC) :**

The main idea is that a single instruction will do all loading, evaluating and storing operations just like a multiplication command will do stuff like loading data, evaluating and storing it, hence it’s complex. Both approaches try to increase the CPU performance

**RISC Architecture:**

1. **Diagram of RISC Architecture:**



1. **Brief Explanation of each component**

**Instruction Fetch (IF):** This stage is responsible for fetching the next instruction from memory. In a RISC processor, instructions are of fixed length and are typically fetched one at a time.

**Instruction Decode (ID):** In this stage, the fetched instruction is decoded to determine the operation to be performed and the operands involved. RISC instructions are generally simple and can be decoded quickly.

**Register File:** RISC architectures typically have a small set of registers (often 32 or 64) that are used for temporary data storage. The register file is where data is read from and written to during instruction execution.

**Arithmetic Logic Unit (ALU):** The ALU performs arithmetic and logical operations on data.

RISC instructions often specify simple operations that can be executed in a single clock cycle, and the ALU is designed for efficiency in carrying out these operations.

**Execution Units:** Some RISC processors have multiple execution units to handle different types of operations simultaneously. For example, there may be separate units for integer arithmetic, floating-point arithmetic, and load/store operations.

**Memory Access (MEM):** In this stage, data can be loaded from or stored to memory. RISC processors typically use a load/store architecture, meaning that memory access is limited to specific load (read data from memory) and store (write data to memory) instructions.

**Write Back (WB):** The results of an operation are written back to the register file in this stage. This stage ensures that the processor maintains coherency and that the results are available for subsequent instructions.

**Control Unit:** The control unit is responsible for managing the execution of instructions, including controlling the flow of data and managing instruction pipelining.

**Pipeline:** Many RISC processors use a pipeline architecture, where multiple instructions are in different stages of execution at the same time. This improves throughput by allowing the processor to work on multiple instructions concurrently.

**Branch Unit:** The branch unit handles conditional and unconditional branches in the instruction stream. Conditional branches are used for decision-making in program flow.

**Cache Memory:** RISC processors often utilize cache memory to store frequently used data and instructions, which helps improve performance by reducing the need to access slower main memory.

3. RISC Processor Instruction Set Examples with explanation (Any 2)

* **Load (LW) Instruction:**

**Assembly Language:** LW $t1, 100($t2)

**Explanation:** This instruction is used to load a 32-bit word from memory into a register. In this example, $t2 contains a base address, and 100 is an offset. The contents of the memory location at the address $t2 + 100 are loaded into $t1.

* **Add (ADD) Instruction:**

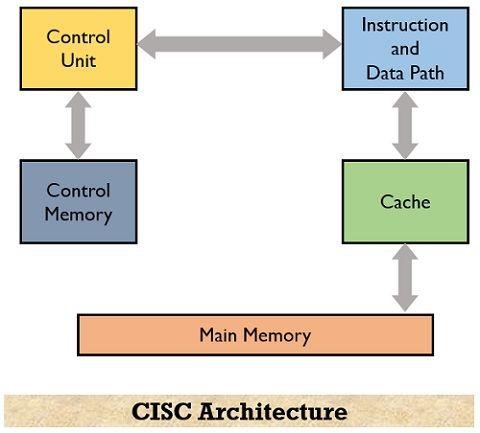
**Assembly Language:** ADD $s1, $s2, $s3

**Explanation:** The ADD instruction is used to perform arithmetic addition. In this example, the values in registers $s2 and $s3 are added together, and the result is stored in register $s1. This is a simple and common arithmetic operation in RISC processors.

RISC processors have a simplified and uniform instruction set, which means that instructions are typically of fixed length and are executed in a single clock cycle. These characteristics make RISC processors efficient and fast, but they may require more instructions to perform complex operations compared to CISC (Complex Instruction Set Computer) processors.

**CISC Architecture**

1. **Diagram of CISC Architecture:**



1. **Brief Explanation of each component**

In a Complex Instruction Set Computer (CISC) architecture, the central processing unit (CPU) is designed to execute a wide variety of complex instructions. Each instruction typically corresponds to a single high-level operation, and these instructions are stored in memory. Here's a brief explanation of each major component in a CISC architecture:

**Control Unit:** The control unit is responsible for managing the execution of instructions. It decodes the instructions fetched from memory and generates the necessary control signals to coordinate the various components of the CPU.

**Instruction Set:** In CISC architectures, the instruction set is rich and complex. It includes a wide range of instructions that can perform various tasks, such as arithmetic operations, data movement, and control flow instructions. CISC processors have a diverse set of instructions that can operate on operands in memory or registers.

**Register File:** CISC architectures typically have a set of general-purpose registers and specialized registers that are used to store data and perform operations. These registers are used to hold intermediate values and are a key component for executing CISC instructions efficiently.

**Memory:** Memory is used to store both data and instructions. CISC processors can operate on data in memory directly, which allows for more complex addressing modes. Memory access in CISC architectures can be both explicit (with load and store instructions) and implicit (with instructions that reference memory directly).

**ALU (Arithmetic Logic Unit):** The ALU is responsible for performing arithmetic and logical operations on data stored in registers or memory. CISC processors typically have a rich set of operations that can be performed, including both simple arithmetic (addition, subtraction) and more complex operations (e.g., multiplication and division).

**FPU (Floating-Point Unit):** CISC architectures often include a dedicated Floating-Point Unit for handling floating-point arithmetic operations. This unit is designed to efficiently perform operations on real numbers, which are used in scientific and engineering calculations.

**Addressing Modes:** CISC architectures offer a wide range of addressing modes, which specify how operands are located in memory. Common addressing modes include immediate, register direct, memory indirect, and scaled-index addressing.

**Pipelining:** Some CISC processors may implement pipelining to improve instruction throughput. Pipelining divides the instruction execution into multiple stages, allowing different instructions to be processed simultaneously, improving overall performance.

**Microcode:** CISC processors may use microcode to execute complex instructions. Microcode is a lower-level set of instructions that control the individual operations of the CPU. Microcode can simplify the design of the CPU by breaking down complex CISC instructions into smaller, simpler operations.

**Control Unit:** The control unit is responsible for managing the execution of instructions. It decodes the instructions fetched from memory and generates the necessary control signals to coordinate the various components of the CPU.

In contrast to CISC, RISC (Reduced Instruction Set Computer) architectures have a smaller, simpler instruction set with a focus on executing instructions quickly, often by making extensive use of registers and optimizing pipelining. CISC architectures, on the other hand, aim to provide more versatile instructions, even if they require more complex hardware to execute.

**3. CISC Processor Instruction Set Examples with explanation (Any 2)**

* **Load and Add (LDA):**

**Operation:** This instruction loads a value from memory into a register and then adds another value to it.

**Example Assembly Code:**

LDA R1, [A] ; Load the value at memory address A into register R1

ADD R1, R2 ; Add the value in R1 to the value in register R2

**Explanation:** In this example, the first instruction loads the content of memory location A into register R1. The second instruction then adds the value in R1 to the value in register R2. This combination of load and add operations can be useful when working with data from memory and performing immediate calculations.

* **Multiply and Store (MULS):**

**Operation:** This instruction multiplies two values in registers and stores the result in memory.

**Example Assembly Code:**

MULS R1, R2, [B] ; Multiply the values in R1 and R2 and store the result in memory location B

**Explanation:** The MULS instruction takes the values from registers R1 and R2, multiplies them together, and then stores the result in memory at location B. This operation is a typical example of a CISC instruction, as it combines multiple steps (loading, multiplication, and storing) into a single instruction. It simplifies complex calculations and reduces the number of instructions required.

CISC processors aim to provide a wide range of powerful instructions, which can lead to reduced code size and more efficient execution of certain tasks. However, they can also be more challenging to design and optimize than RISC (Reduced Instruction Set Computer) processors, which have a simpler and more uniform instruction set.

**Post Lab Descriptive Questions :**

# Write a tabular comparative analysis of RISC v/s CISC

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **RISC** | **CISC** |  |
| **Instructions** | **Few, simple instructions** | **Many, complex instructions** |  |
| **Complexity** | **Low** | **High** |  |
| **Pipeline** | **Short, fewer stages** | **Long, more stages** |  |
| **Clock Cycle** | **Consistent** | **Variable** |  |
| **Performance** | **Often better for specific tasks** | **More versatile but slightly performance** | **lower** |
| **Memory Access** | **Separate load/store** | **Embedded in complex instructions** |  |
| **Code Size** | **Larger** | **Smaller** |  |
| **Power Efficiency** | **Generally more efficient** | **Can be less efficient** |  |
| **Example**  **Architectures** | **ARM, RISC-V** | **x86 (Intel, AMD)** |  |
| **Common Usage** | **Mobile, embedded, HPC** | **Desktop, server** |  |
| **Ease of Programming** | **Simpler** | **More complex** |  |

**Conclusion:**

In conclusion, our experiment on RISC and CISC architectures highlighted their distinct characteristics and performance differences. RISC processors excelled in power efficiency and adaptability, while CISC processors demonstrated superiority in handling complex workloads. The choice between these architectures should be driven by specific application requirements. This study deepened our understanding of computer architecture, aiding informed decision-making in selecting the most suitable platform for diverse computing needs.

# Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge