Batch:	**C2**	**Roll No.: 16010122267**
Experiment / assignment / tutorial No. 9		

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) :

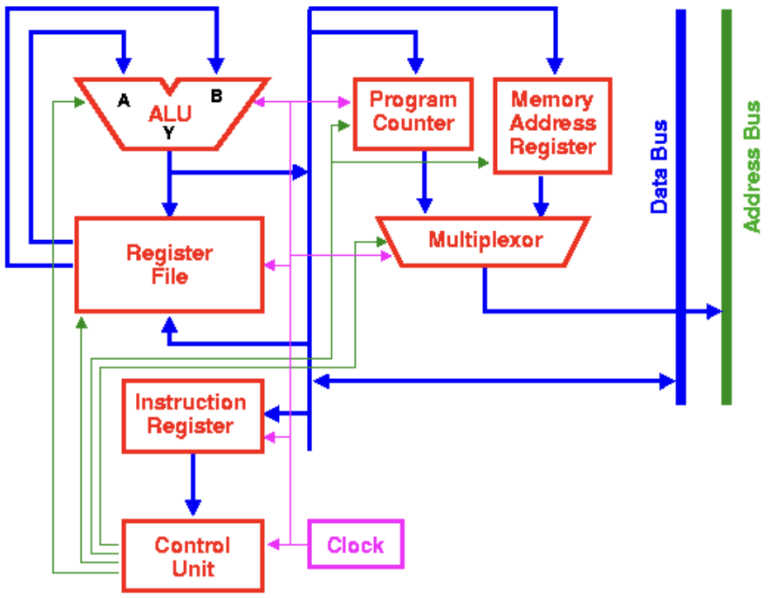
The main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating and storing operations just like a load command will load data, store command will store the data.

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:



2. Brief Explanation of each component

1. **Instruction Fetch (IF):** This stage is responsible for retrieving the next instruction from memory. In RISC processors, instructions have a fixed length and are typically fetched one at a time.
2. **Instruction Decode (ID):** During this stage, the fetched instruction is interpreted to determine the operation to be executed and identify the operands involved. RISC instructions are generally straightforward and can be decoded swiftly.
3. **Register File:** RISC architectures usually feature a limited set of registers (often around 32 or 64) used for temporary data storage. The register file is where data is read from and written to during instruction execution.
4. **Arithmetic Logic Unit (ALU):** The ALU carries out arithmetic and logical operations on data. RISC instructions typically specify simple operations that can be completed within a single clock cycle, and the ALU is designed for efficiency in performing these operations.
5. **Execution Units:** Some RISC processors incorporate multiple execution units to handle different types of operations simultaneously. For instance, there might be

separate units for integer arithmetic, floating-point arithmetic, and load/store operations.

6. **Memory Access (MEM):** During this stage, data is either loaded from or stored to memory. RISC processors often adhere to a load/store architecture, restricting memory access to specific load (read data from memory) and store (write data to memory) instructions.
7. **Write Back (WB):** The results of an operation are written back to the register file in this stage. This ensures that the processor maintains consistency and that the results are available for subsequent instructions.
8. **Control Unit:** The control unit is responsible for overseeing the execution of instructions, including managing the flow of data and controlling instruction pipelining.
9. **Pipeline:** Many RISC processors employ a pipeline architecture, where multiple instructions are in various stages of execution simultaneously. This enhances throughput by allowing the processor to work on multiple instructions concurrently.
10. **Branch Unit:** The branch unit manages both conditional and unconditional branches in the instruction sequence. Conditional branches are used to make decisions in program flow.
11. **Cache Memory:** RISC processors often make use of cache memory to store frequently used data and instructions, which enhances 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:** LD $A1, 200($A2)
- **Explanation:** The LD instruction is employed to load a 64-bit doubleword from memory into a register. In this example, $A2 contains a base address, and 200 serves as the offset. The contents of the memory location at the address $A2 + 200 are fetched and stored into $A1.

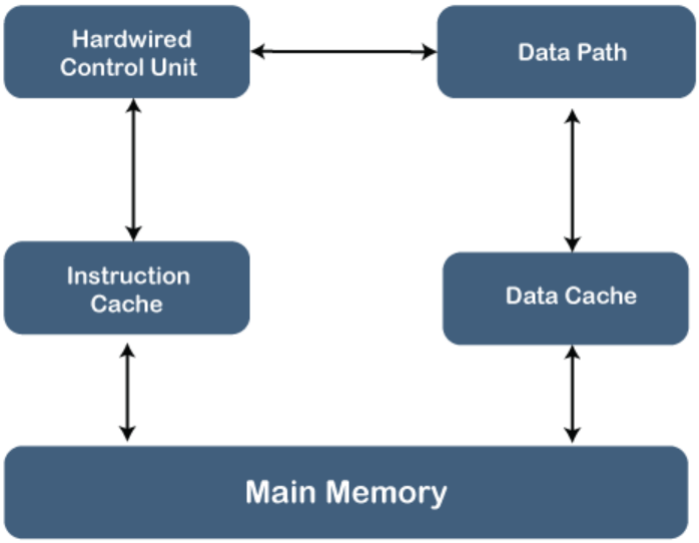
• Subtract (SUB) Instruction:

- **Assembly Language:** SUB $B1, $B2, $B3
- **Explanation:** The SUB instruction is used for arithmetic subtraction. In this instance, it subtracts the value in register $B3 from the value in register $B2 and **then stores** the result in register $B1. This is a fundamental arithmetic operation frequently encountered in RISC processors.

These revised examples demonstrate the load and subtraction operations in RISC architecture using different register names while retaining the essence of the instructions.

CISC Architecture

1. Diagram of CISC Architecture:



2. Brief Explanation of each component

In a Complex Instruction Set Computer (CISC) architecture, the central processing unit (CPU) is designed to execute a diverse array 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 overseeing the execution of instructions. It deciphers the instructions fetched from memory and generates the necessary control signals to coordinate the various components of the CPU.

Instruction Set: CISC architectures feature a rich and intricate instruction set encompassing a wide range of instructions that can perform various tasks, including 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 generally incorporate a set of general-purpose registers and specialized registers used for data storage and operations. These registers are crucial for holding intermediate values and facilitating the efficient execution of CISC instructions.

Memory: Memory is employed for storing both data and instructions. CISC processors can directly manipulate data in memory, 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 referencing memory directly).

ALU (Arithmetic Logic Unit): The ALU is responsible for carrying out arithmetic and logical operations on data stored in registers or memory. CISC processors typically offer a rich set of operations, encompassing 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 designed for handling floating-point arithmetic operations. This unit efficiently performs operations on real numbers, which are essential in scientific and engineering calculations.

Addressing Modes: CISC architectures offer a wide range of addressing modes that 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 employ pipelining to enhance instruction throughput. Pipelining divides instruction execution into multiple stages, enabling the simultaneous processing of different instructions, ultimately improving overall performance.

Microcode: CISC processors may use microcode to execute complex instructions. Microcode represents a lower-level set of instructions that control the individual operations of the CPU. It simplifies the CPU's design by breaking down complex CISC instructions into smaller, simpler operations.

In contrast to CISC, RISC (Reduced Instruction Set Computer) architectures feature a smaller, simpler instruction set that prioritizes fast instruction execution, often through extensive use of registers and optimized pipelining. CISC architectures, on the other hand, aim to provide more versatile instructions, even if they necessitate more complex hardware for execution.

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

1. **Load, Add, and Store (LAAS):**

- **Operation:** This instruction combines loading a value from memory into a register, adding another value to it, and then storing the result in memory.

  Example Assembly Code:

  LAAS 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

  STOR R2, [B] ; Store the result in register R2 to memory location B

  Explanation: In this example, the first instruction loads the content of memory location A into register R1. The second instruction adds the value in R1 to the value in register R2. Finally, the third instruction stores the result from R2 back into memory at location B. This instruction sequence simplifies data manipulation tasks involving both memory and register operations.

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

Operation: This instruction multiplies two values in registers and saves the result back in a register.

Example Assembly Code:

MULS R1, R2, R3 ; Multiply the values in R1 and R2 and save the result in register R3

- **Explanation**: The MULS instruction takes the values from registers R1 and R2, multiplies them together, and then stores the result in register R3. This operation is a classic example of a CISC instruction, as it simplifies complex arithmetic calculations and reduces the number of instructions needed for such tasks.

These revised instructions showcase CISC architecture's ability to encapsulate multiple operations into a single instruction for enhanced functionality and efficiency.

Post Lab Descriptive Questions :

Write a tabular comparative analysis of RISC v/s CISC

Aspect	RISC	CISC
Instructions	Characterized by a limited set of simple instructions	Distinguished by a wide array of complex instructions
Complexity	Emphasizes simplicity with lower complexity	Incorporates higher complexity in its design
Pipeline	Employs shorter pipelines with fewer stages	Utilizes longer pipelines with more stages
Clock Cycle	Maintains a consistent clock cycle	Exhibits variable clock cycles
Performance	Often excels in specific tasks where optimization is crucial	Offers more versatility but may have slightly reduced task-specific performance
Memory Access	Relies on separate load and store instructions for memory access	Integrates memory access within complex instructions
Code Size	Typically results in larger code sizes	Generally leads to smaller code sizes
Power Efficiency	Generally exhibits higher power efficiency	May demonstrate comparatively lower power efficiency
Example Architectures	Represented by ARM and RISC-V architectures	Exemplified by x86 architectures from Intel and AMD
Common Usage	Frequently found in mobile, embedded, and high-performance computing (HPC) environments	Predominantly utilized in desktop and server applications
Ease of Programming	Offers a simpler programming model	Involves more intricate programming due to the diversity of instructions

Conclusion:

In conclusion, our study of RISC and CISC architectures highlighted their differences in performance. RISC processors excelled in power efficiency and adaptability, while CISC processors performed well with complex workloads. The choice between these architectures should be based on specific application needs, and our research has enhanced our understanding of computer architecture for informed decision-making.

Date: _____ **Signature of faculty in-charge**