# Batch: B4 Roll No.: 1601012221

**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)

CO2- Understand the Central processing unit with addressing modes and working of control unit in depth

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Books/ Journals/ Websites referred:**

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “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 Set Instruction Set Architecture (RISC)

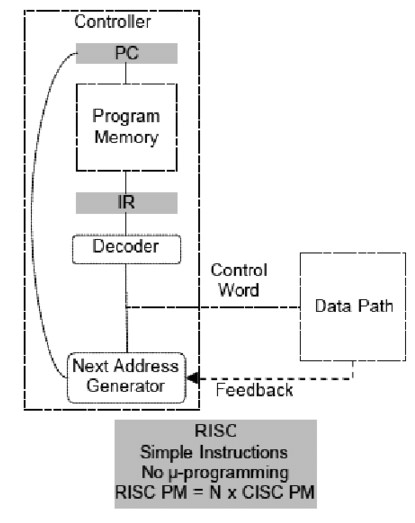
The main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating and storing operations just like a load command will load data, store command will store the data.

# 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. Controller:

- The controller is responsible for coordinating the various operations within the processor and controlling the execution of instructions.

2. Data Path:

- The data path is where actual data manipulation and computation take place. It consists of various functional units like registers, ALU (Arithmetic Logic Unit), and other components for data processing.

3. PC (Program Counter):

- The Program Counter is a register that keeps track of the memory address of the next instruction to be fetched and executed.

4. Program Memory:

- Program Memory is where the instructions of a program are stored. The PC fetches instructions from here.

5. IR (Instruction Register):

- The Instruction Register is a temporary storage location within the processor that holds the currently fetched instruction from memory.

6. Decoder:

- The Decoder decodes the instruction fetched from memory. It interprets the opcode (operation code) of the instruction to determine what operation needs to be performed.

7. Control Word:

- The Control Word is a set of control signals generated by the Decoder. It provides instructions to the various components of the data path to specify how the operation should be executed.

8. Next Address Generator:

- The Next Address Generator calculates the memory address of the next instruction to be fetched. It typically takes into account the current PC value and the control flow instructions in the program, like branches and jumps.

9. Feedback from Data Path:

- The Data Path may provide feedback signals to the Next Address Generator, especially in the case of branch instructions or other control flow changes. This feedback helps update the PC to ensure the correct execution sequence.

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

1. ADD (Addition) Instruction:

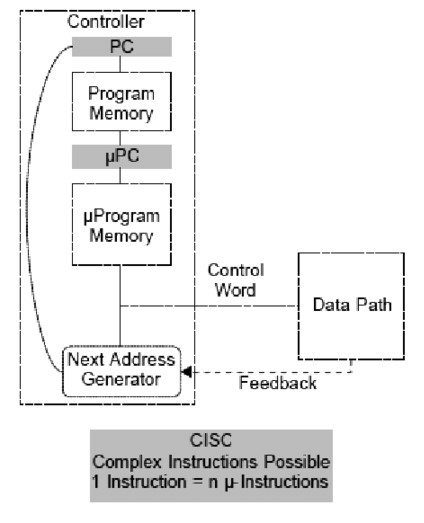
* Opcode: 0001 (an example binary representation)
* Format: ADD Rd, Rs1, Rs2
* Explanation: This instruction is used for performing integer addition. It adds the values stored in registers Rs1 and Rs2 and stores the result in register Rd. For example, if you have data in registers Rs1 and Rs2 like Rs1 = 10 and Rs2 = 20, executing the ADD instruction `ADD R3, R1, R2` would result in R3 containing the value 30 (10 + 20).

2. SUB (Subtraction) Instruction:

* Opcode: 0010 (an example binary representation)
* Format: SUB Rd, Rs1, Rs2
* Explanation: This instruction is used for performing integer subtraction. It subtracts the value stored in register Rs2 from the value in register Rs1 and stores the result in register Rd. For example, if you have data in registers Rs1 and Rs2 like Rs1 = 30 and Rs2 = 10, executing the SUB instruction `SUB R4, R1, R2` would result in R4 containing the value 20 (30 - 10).

# CISC Architecture

|  |  |
| --- | --- |
| 1. | Diagram of CISC Architecture: |



|  |  |
| --- | --- |
| 2. | Brief Explanation of each component |

1. Controller:

- The controller in a CISC architecture is responsible for managing the execution of complex instructions and coordinating various components within the processor.

2. Data Path:

- The data path is where the actual data manipulation and computation take place, similar to what we discussed for the RISC architecture.

3. PC (Program Counter):

- The Program Counter is a register that keeps track of the memory address of the next instruction to be fetched and executed.

4. Program Memory:

- Program Memory is where the instructions of a program are stored. The PC fetches instructions from here.

5. μPC (Microprogram Counter):

- In CISC architectures, particularly those with microprogramming, there is often a microprogram counter (μPC). Microprogramming is a technique used to execute complex instructions by breaking them down into a series of simpler microoperations, each controlled by a microprogram. The μPC keeps track of the microinstruction currently being executed.

6. μProgram Memory (Microprogram Memory):

- Microprogram Memory stores microinstructions, which are used to control the execution of complex instructions. Each complex instruction is broken down into a sequence of microinstructions, and the μPC is used to fetch and execute these microinstructions.

7. Next Address Generator:

- The Next Address Generator is responsible for determining the memory address of the next instruction to be fetched. It takes into account the current state of the program, including the PC and the μPC, to determine the next instruction's address.

8. Decoder:

- The Decoder decodes the complex instruction fetched from Program Memory. In CISC architectures, complex instructions can include various operations and addressing modes. The Decoder interprets the complex instruction and generates control signals for the data path and microprogram memory.

9. Control Word:

- Similar to the RISC architecture you mentioned earlier, the Control Word is a set of control signals generated by the Decoder. It provides instructions to various components of the data path and microprogram memory, specifying how the complex instruction should be executed.

10. Feedback from Data Path:

- As in the RISC architecture, the Data Path in CISC architectures may provide feedback signals to the Next Address Generator, especially in the case of control flow instructions or exceptions. This feedback helps update the PC and μPC to ensure the correct execution sequence.

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

1. LOAD (Load Data from Memory) Instruction:

* Opcode: 0010 (an example binary representation)
* Format: LOAD Rd, [Rb + offset]
* Explanation: This instruction is used to load data from memory into a register. It takes two operands: Rd (destination register) and Rb (base register), along with an offset. The offset is added to the value in the base register (Rb) to calculate the memory address from which data is loaded. The loaded data is then stored in the destination register (Rd). For example, if you have data at memory address R2 + 100, executing the LOAD instruction `LOAD R1, [R2 + 100]` would load the data from that memory location into register R1.

2. MULTIPLY (Multiply) Instruction:

* Opcode: 0111 (an example binary representation)
* Format: MULTIPLY Rd, Rs1, Rs2
* Explanation: This instruction is used to perform multiplication of two values and store the result in a destination register (Rd). It takes two source registers (Rs1 and Rs2), multiplies their values, and stores the result in Rd. For example, if you have values in registers Rs1 = 10 and Rs2 = 5, executing the MULTIPLY instruction `MULTIPLY R3, R1, R2` would result in R3 containing the value 50 (10 \* 5).

**Post Lab Descriptive Questions**

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

1. Instruction Set Complexity:

- RISC: Features a simple and small instruction set with basic instructions. - CISC: Utilizes a complex and large instruction set with diverse and multi-step instructions.

2. Instruction Execution:

* RISC: Typically executes most instructions in a single clock cycle, leading to fast and predictable performance.
* CISC: Allows variable-clock-cycle execution, with some instructions requiring multiple cycles, potentially leading to varying execution times.

3. Memory Access:

* RISC: Utilizes load and store instructions for memory access, promoting a uniform approach.
* CISC: Supports direct memory access operations using complex addressing modes, allowing for more flexible memory interactions.

4. Hardware Complexity:

- RISC: Employs relatively simple hardware components, making it power-efficient. - CISC: Features more complex hardware components, including microprogramming, which can consume more power.

5. Typical Use Cases:

* RISC: Commonly found in modern embedded systems, mobile devices, and generalpurpose computing platforms.
* CISC: Historically prevalent in mainframes and older computing systems; still used in some high-performance computing applications.

**Conclusion:**

In conclusion, this experiment highlighted the key differences between RISC and CISC architectures. RISC prioritizes simplicity and efficiency and is common in modern computing. CISC offers complex instructions and has historical relevance in older systems. The choice depends on system needs, with modern processors often blending aspects of both.

## Date: 21th October, 2023 Signature of faculty in-charge