CISC AND RISC

Here's a simplified comparison between **CISC** and **RISC** architectures:

1. **CISC (Complex Instruction Set Computer)**:
   - Large, complex instruction set.
   - Instructions vary in size.
   - Needs a microcode interpreter to translate instructions.
   - Instructions are broken down into smaller steps (micro-operations).
   - Example: **Intel x86 family**.
2. **RISC (Reduced Instruction Set Computer)**:
   - Small, simple instruction set.
   - All instructions are the same size.
   - Instructions are simpler and easier to decode.
   - Executed directly by hardware.
   - Examples: **ARM, MIPS, PowerPC, SPARC**.

Width of buses in MIPS:

In **MIPS architecture**, the **bus width** typically depends on the specific implementation of MIPS, but in general:

- For **32-bit MIPS processors**, the bus width is **32 bits**. This means the data, addresses, and instructions are transferred in 32-bit chunks.
- For **64-bit MIPS processors**, the bus width is **64 bits**, allowing larger data transfers and handling of 64-bit addresses.

Components of MIPS:

In the **MIPS architecture**, the following are the key components and their roles:

1. **Program Counter (PC)**: Holds the address of the next instruction to be executed.
2. **Memory**: Stores instructions and data.
3. **Instruction Register (IR)**: Holds the current instruction fetched from memory.
4. **Register File**: A collection of registers that store temporary data for fast access by the processor.
5. **Arithmetic and Logic Unit (ALU)**: Performs arithmetic operations (addition, subtraction) and logical operations (AND, OR).
6. **Control Unit (CU)**: Manages the execution of instructions by directing other components.
7. **Buses**: These are the pathways that interconnect all components except the Control Unit, allowing data and instructions to flow between them.

MIPS assembly programming:

1. **Data Types and Literals**: In MIPS, data types represent how data is stored (e.g., integers, characters). Literals are constant values (like numbers or characters) directly used in

instructions.

2. **Registers**: MIPS has 32 general-purpose registers (e.g., $t0, $s1, etc.), each holding 32-bit data for quick access during operations.
3. **Program Structure**: MIPS programs consist of sections like:
   - **.data**: To declare data (variables, strings).
   - **.text**: Contains the instructions (code) for execution.
4. **Data Declarations**: In the .data section, data is declared using labels. For example:
   - myVar: .word 5 declares a word (32-bit) variable myVar initialized to 5.
5. **Arithmetic Instructions**: Basic operations like addition, subtraction, and multiplication:
   - add $t0, $t1, $t2: Adds values in $t1 and $t2 and stores the result in $t0.
6. **Load/Store Instructions**: MIPS uses load/store architecture:
   - lw $t0, 0($s0): Loads a word from memory address in $s0 into $t0.
   - sw $t0, 4($s1): Stores the value in $t0 into memory at the address in $s1 with an offset of 4.
7. **Indirect and Based Addressing**:
   - **Indirect Addressing**: Uses a register to point to a memory location, like lw $t0, 0($t1) where the address is stored in $t1.
   - **Based Addressing**: Adds an offset to the base register address to calculate the effective address. Example: lw $t0, 4($s1) adds 4 to the address in $s1.