Accumulator-Based CPU Architecture

Key Components of an Accumulator-based CPU

Accumulator (ACC)

- A special-purpose register used to store intermediate results during computations.
- The CPU performs most operations between the accumulator and other registers or memory locations.

Arithmetic Logic Unit (ALU)

- Performs arithmetic (addition, subtraction, etc.) and logical (AND, OR, NOT, etc.) operations on the values in the accumulator.
- The result is typically stored back into the accumulator.

Registers

- Apart from the accumulator, there might be general-purpose or special-purpose registers, but the accumulator is the primary one used for computation.

Memory

- Holds data and instructions.
- The CPU fetches instructions from memory, and the data for operations is often loaded into the accumulator.

Control Unit

- Decodes and executes instructions by sending control signals to the ALU, registers, and memory.

Working of an Accumulator-based CPU

Loading Data

- The CPU loads data from memory into the accumulator using instructions like `LOAD`.

Performing Operations

- The ALU performs operations with the data in the accumulator.
- For example, an `ADD` instruction would add the contents of the accumulator with another operand (from memory or another register).

Storing Results

- After the operation, the result is usually stored back into the accumulator.
- If needed, the result can be moved to memory or another register.

Example

- `LOAD A` : Load the value at memory location A into the accumulator.
- `ADD B` : Add the value at memory location B to the accumulator.
- `STORE C` : Store the value in the accumulator into memory location C.

Pros and Cons of Accumulator-based Architecture

Pros

- **Simplicity:** Fewer registers, with operations centralized around the accumulator.
- **Reduced Instruction Set:** Simpler instructions focused on the accumulator.

Cons

- **Limited Parallelism:** Centralized use of the accumulator limits parallel execution.
- **Speed Bottlenecks:** Heavy use of the accumulator may delay execution.

Example Instruction Set (Hypothetical)

- `LOAD R` : Load value from register R into the accumulator.
- `ADD R` : Add the value in register R to the accumulator.
- `SUB R` : Subtract the value in register R from the accumulator.
- `STORE R` : Store the accumulator's value into register R.
- `JUMP` : Jump to another instruction based on the program counter.

Accumulator-Based Architecture (1-Operand Machine)

Key Characteristics

- Accumulator (ACC) holds all intermediate results.
- Instruction format: $I = \text{op.adr}$ where:
  - `op` = operation code (e.g., `LOAD`, `ADD`)
  - `adr` = memory address
- Only one operand (in memory); the second implicit operand is the accumulator.

Instruction Cycle (Step-by-Step)

Notation

- `PC` = Program Counter (address of the next instruction)
- `IR` = Instruction Register (holds fetched instruction)
- `AR` = Address Register (holds address part of instruction)
- `ACC` = Accumulator
- $M[\,]$ = Memory

Step 1: Fetch Instruction

$$
\begin{aligned}
&\texttt{IR} \leftarrow M[\texttt{PC}] \quad \text{(Fetch instruction)} \\
&\texttt{PC} \leftarrow \texttt{PC} + 1 \quad \text{(Increment PC)} \\
&\texttt{op} \leftarrow \texttt{IR[opcode]} \\
&\texttt{AR} \leftarrow \texttt{IR[address]}
\end{aligned}
$$

Step 2: Decode and Execute

Common Instructions:

$$
\begin{aligned}
\texttt{LOAD adr:} \quad & \texttt{ACC} \leftarrow M[\texttt{adr}] \\
\texttt{ADD adr:} \quad & \texttt{ACC} \leftarrow \texttt{ACC} + M[\texttt{adr}] \\
\texttt{SUB adr:} \quad & \texttt{ACC} \leftarrow \texttt{ACC} - M[\texttt{adr}] \\
\texttt{STORE adr:} \quad & M[\texttt{adr}] \leftarrow \texttt{ACC} \\
\texttt{JUMP adr:} \quad & \texttt{PC} \leftarrow \texttt{adr}
\end{aligned}
$$

Example Execution

Initial Memory State

Address	Contents
100	`LOAD 200`
101	`ADD 201`
102	`STORE 202`
200	5
201	3
202	0

Execution Steps

- `PC = 100` $\rightarrow$ Fetch `LOAD 200`, set `AR = 200`, `IR = LOAD`
- `ACC` $\leftarrow M[200] = 5$
- `PC = 101` $\rightarrow$ Fetch `ADD 201`, set `AR = 201`, `IR = ADD`
- `ACC` $\leftarrow$ `ACC` $+ M[201] = 5 + 3 = 8$
- `PC = 102` $\rightarrow$ Fetch `STORE 202`, set `AR = 202`, `IR = STORE`
- `M[202]` $\leftarrow$ `ACC` $= 8$

Summary

- Instruction format: `op.adr`
- Execution focuses on the accumulator for all calculations.
- Efficient for simple systems, but limits flexibility due to single operand.