Instruction Set Architectures: Accumulator and Stack-Based Machines

1. Accumulator Architecture

Concept

An **accumulator architecture** uses a single register called the **accumulator (AC)** to hold intermediate results. All operations are performed between AC and a memory operand.

Instruction Set Examples

- `ADD A` $\Rightarrow$ AC := AC + M[A]
- `SUB A` $\Rightarrow$ AC := AC - M[A]
- `MUL A` $\Rightarrow$ AC := AC $\times$ M[A]
- `LOAD A` $\Rightarrow$ AC := M[A]
- `STORE A` $\Rightarrow$ M[A] := AC

Example: $A \times B - (A + B \times C)$

```
LOAD B        ; AC := B
MUL C         ; AC := B * C
ADD A         ; AC := A + B*C
STORE D       ; D := A + B*C
LOAD A        ; AC := A
MUL B         ; AC := A * B
SUB D         ; AC := A*B - (A + B*C)
```

Pros and Cons

Pros

- Very low hardware requirements
- Easy to design and understand

Cons

- Accumulator becomes the bottleneck

- High memory traffic
- Little parallelism or pipelining

2. Stack-Based Architecture (Zero-Address Machine)

Concept

This architecture uses a **stack** for all computations. Operands are implicitly at the top of the stack, and instructions like `ADD`, `MUL` operate on them.

Instruction Set Examples

- `PUSH A` ⇒ Push A onto stack
- `POP` ⇒ Remove top of stack
- `ADD` ⇒ Pop two operands, add, push result
- `MUL` ⇒ Pop two operands, multiply, push result

Example: $A \times B - (A + C \times B)$

```
PUSH A
PUSH B
MUL         ; A*B
PUSH A
PUSH C
PUSH B
MUL         ; C*B
ADD         ; A + C*B
SUB         ; A*B - (A + C*B)
```

Pros and Cons

Pros

- Implicit operand addressing
- Low hardware complexity

Cons

- Stack can be a bottleneck
- Difficult to access random data
- Extra instructions like `SWAP`, `POP` needed

3. Summary Table

Feature	Accumulator-Based	Stack-Based (Zero Address)
Operand Source	AC and Memory	Top of Stack
Instruction Format	One explicit address	No explicit address
Temp Storage	Accumulator	Stack
Hardware Complexity	Low	Very Low
Instruction Count	Moderate	High
Parallelism	Low	Very Low
Efficiency	Medium	Low

Table 1: Comparison: Accumulator vs Stack Architectures

4. Instruction Set Requirements

A well-designed instruction set should be:

- **Complete:** Should be able to compute any function.
- **Efficient:** Frequently used operations must be fast.
- **Regular:** Instruction formats should be predictable.
- **Compatible:** Should support legacy or existing codebase.

5. Classification of Instructions

Type	Purpose	Examples
Data Transfer	Move/copy data	`LOAD`, `STORE`, `MOVE`
Arithmetic	Numeric computation	`ADD`, `SUB`, `MUL`
Logical	Bitwise operations	`AND`, `NOT`
Program Control	Change execution flow	`JUMP`, `BRANCH`
Input/Output	Device communication	`READ`, `PRINT`

Table 2: Types of Instructions in ISAs