Cloud Infrastructure

Instruction Set Architecture (ISA)

ISA: It is the interface between software and hardware. It tells exactly what instructions the processor understands and how it executes them.

What ISA specifies:

- The **data types** supported.
- The **registers** available.
- The **memory model** (addressing, virtual memory).
- The complete **instruction set**.
- The **I/O model**.

Instruction Set Architecture (ISA)

Instruction categories (8085 example):

- Data transfer
- Arithmetic
- Logical
- Branching
- Control

Key point:

- ISA defines **what operations are possible**.
- Microarchitecture defines **how they are carried out** in hardware.

Components of an ISA

An Instruction Set Architecture (ISA) defines how a processor looks to a programmer. Key parts include:

- **Storage cells**: general and special-purpose registers, memory cells, and I/O-related storage.
- **Machine instruction set**: all the operations the machine can perform, involving register transfers, fetch/execute cycle, etc.
- **Instruction format**: layout of bits within an instruction (opcode, operands).
- **Fetch–execute cycle**: the sequence of steps for executing instructions (fetch, decode, execute).

Every instruction must define four things:

1. **Operation** → opcode (e.g., add, load, branch).
2. **Operands** → where to find them (registers, memory, I/O).
3. **Result location** → where to store the output.
4. **Next instruction** → usually via program counter (PC), but branches can change this.

Instruction Types

Data movement (load, store)

Arithmetic and logic (ALU) (add, sub, shift, etc.)

Branch (control flow) (conditional/unconditional jumps, altering sequence).

Evolution of Instruction Sets

Three main architectures historically:

1. **Stack architecture** → operands come from a stack (push/pop).

2. **Accumulator architecture** → one special register (accumulator) holds results.

3. **General-purpose register (GPR) architecture** → multiple registers, most flexible; can be register–register, register–memory, or memory–memory.

Stack Architecture Example

A * B – (A + C * B)

- **Stack-based execution** means operands are pushed onto the stack, and operations (like `mul`, `add`, `sub`) use the top elements of the stack automatically.
- Steps for the expression **A * B – (A + C * B):**
  1. `push A, push B, mul` → computes **A * B**.
  2. `push A, push C, push B, mul, add` → computes **A + (C * B)**.
  3. `sub` → subtracts the second result from the first.

- No need to name registers explicitly.
- Operations always use the top of the stack.
- Instruction sequence is longer because of repeated `push/pop`.