Introduction to the One Address Machine (OAM)

- The course emphasizes practical, hands-on understanding of low-level machine operations.
- A simulator for the OAM will be built in Python to understand how the machine operates.
- The OAM is based on the **von Neumann architecture**:
    - Components: Input/Output, Processor, Memory, Storage.
    - Key simplifications:
        - Infinite memory.
        - No distinct storage—data persists in infinite memory.
        - I/O operations occur via memory-mapped I/O (e.g., Memory Location 0 serves as a proxy for keyboard/screen).

OAM Architecture

1. **Processor and Components:**
    - **Arithmetic Logic Unit (ALU):** Handles calculations and logic operations.
    - **Bus:** Transfers data between memory and processor.
    - **Registers:**
        - **Program Counter (PC):** Tracks the memory location of the next instruction.
        - **Instruction Register (IR):** Stores the current instruction being executed.
        - **Address Register (AR):** Interacts with memory via the bus.
        - **Accumulator (ACC) & B Register:** Store operands for operations, with results stored in the ACC.
2. **Memory:**
    - Conceptualized as infinitely large, storing:
        - Data (e.g., variables).
        - Program instructions.
    - Memory locations store a single unit of data (e.g., integer, character).
    - **Memory-Mapped I/O:**
        - Location 0 is reserved for keyboard (input) and screen (output).

Von Neumann Architecture

- Characterized by:
    - Unified memory for instructions and data.
    - Sequential instruction execution by default.
    - Key limitation: **von Neumann bottleneck** (slow interaction between processor and memory).

Instruction Cycle in OAM

The machine operates through a **three-phase instruction cycle**:

1. **Fetch Phase:**
    - PC value is copied to AR.
    - Memory retrieves the instruction at the address in AR.
    - Instruction is placed in IR.
2. **Increment Phase:**
    - PC is incremented by 1 to point to the next instruction.
    - Ensures sequential execution unless explicitly modified.
3. **Execute Phase:**
    - The instruction in IR is interpreted and executed.
    - Actions depend on the operation specified by the instruction.

Instruction Set of the OAM

- Total: **17 instructions**.
- **Arithmetic Instructions:**
    - `ADD A`: Add the value at address A to ACC.
    - `SUB A`: Subtract the value at address A from ACC.
    - `MUL A`: Multiply ACC by the value at address A.
    - `DIV A`: Divide ACC by the value at address A.
- **Direct Register Operations:**
    - `SET V`: Sets ACC to a specific value V.
    - `NEG`: Negates the value in ACC.
    - `INC`: Increments ACC by 1.
    - `DEC`: Decrements ACC by 1.
- **Memory Transfer Instructions:**
    - `LOAD A`: Load the value at address A into ACC.
    - `STORE A`: Store the value in ACC to address A.

Key Design Principles

- The OAM is a **one-address machine**:
    - Most instructions reference at most **one memory address**.

    - ACC holds one operand; additional operands (if needed) are fetched from memory.
- Trade-offs in instruction set design:
    - Smaller sets are efficient but may require multiple instructions for certain operations.
    - Larger sets increase complexity and hardware requirements.

Simplified OAM Behaviors

- Sequential Execution:
    - Default behavior due to the increment phase.
    - Explicit instructions (e.g., `JUMP`) are needed to alter this sequence for loops, recursion, etc.
- Registers and Memory Interaction:
    - Memory operations are slower than processor operations.
    - **Cache Memory:** Modern systems mitigate this bottleneck by using fast, localized memory.

Applications and Examples

- Real-world parallels include early Macintosh computers, simple calculators, and modern reduced instruction set computers (**RISC**).
- Use of the OAM in Python simulation:
    - Build and visualize low-level operations such as addition, memory transfer, and loops.

Upcoming Topics

- Detailed implementation of the remaining 7 instructions.
- Writing and testing a program using the OAM simulator.
- Exploring advanced instructions and their use cases.

Summary

- The OAM provides foundational insights into computer architecture and instruction cycles.

- Practical programming exercises (e.g., building a simulator) are critical to understanding these concepts.
- Efficiency and trade-offs are central to instruction set and hardware design.