Week 1: Project Planning & Setup

### **1. Define Project Scope**

**Purpose:** To clearly outline the boundaries and goals of the emulator, ensuring focus and clarity throughout development.

* **Objectives**:
  + Create a basic CPU emulator that supports essential CPU functions (arithmetic, logical, data movement, control flow).
  + Gain hands-on experience with CPU architecture concepts, including instruction execution and memory management.
  + Design a modular structure that can be expanded with additional instructions or features in the future.
* **Core Features**:
  + **Instruction Set**: Define and implement a simple instruction set with operations for arithmetic, logic, data movement, and control flow.
  + **Registers and Memory**: Simulate general-purpose registers, a program counter, and memory.
  + **Clock Cycle Simulation**: Implement a mechanism that simulates clock cycles to process instructions.
  + **Basic Error Handling**: Handle invalid instructions or memory access errors.
* **Constraints**:
  + Limited to essential CPU functions to keep development manageable.
  + Developed in C++ with standard libraries, avoiding external dependencies.
* **Deliverables**:
  + A functional CPU emulator hosted on GitHub.
  + Documentation, including a README and a guide to the instruction set.
  + Test cases and a demo script.

### **2. Gather Resources**

**Purpose:** To identify and collect the necessary resources for developing and understanding the project.

* **Technical Resources**:
  + **Programming**: C++ language documentation (for syntax, standard libraries).
  + **CPU Architecture References**: Online resources on basic CPU design and architecture, such as instruction sets, memory management, and registers.
  + **Emulation Concepts**: Articles or tutorials on basic CPU emulation for guidance on structuring the code.
* **Tools**:
  + **GitHub**: For version control and collaboration.
  + **Integrated Development Environment (IDE)**: Options include Visual Studio Code, a C++ compiler with debugging tools (GCC, GDB).
  + **Documentation Tools**: Markdown editors or tools like Doxygen for in-code documentation.
* **Learning Resources**:
  + Online tutorials on CPU architecture.
  + Books or online courses on C++ for any advanced concepts you may need.
  + Git tutorials to manage the version control aspect effectively.

### ​ **3. Set Up Development Environment**

**Purpose:** To establish a functional workspace where development and testing can proceed smoothly.

* **Install Required Software**:
  + Set up a C++ compiler (e.g., GCC or Clang) and debugger (e.g., GDB).
  + Install Git and create a GitHub repository for version control.
  + Choose and set up a suitable IDE (Visual Studio Code is popular and customizable for C++ projects).
* **Configure the GitHub Repository**:
  + Initialize a Git repository for version control and create a GitHub repository.
  + Plan a simple branching strategy for feature development and maintenance.
  + Create a README file with a project overview and setup instructions.
* **Prepare Project Directory Structure**:
  + Set up a basic directory structure with folders for source code (src), documentation (docs), and tests (tests).
  + Write a simple .gitignore file to exclude unnecessary files from version control.
* **Write Initial Code**:
  + Write basic C++ code for the main file, setting up the basic emulator class and testing setup.

### 4. Tasks and Details

1. **Outline the Features of the Virtual CPU:**
   * **Instruction Set**: Define the basic instructions your CPU will support (e.g., arithmetic, logic, load/store, and control flow).
   * **Registers**: Decide on the number and types of registers (general-purpose, special-purpose).
   * **Memory Management**: Determine how memory will be managed (e.g., stack, heap, memory-mapped I/O).
   * **Flags and Status Register**:Zero flag, Carry flag, Overflow flag, Sign flag.
   * **Error Handling**: Plan for any potential error handling, such as illegal instructions or memory access errors.
2. **Choose Programming Language and Tools:**
   * **Programming Language**: C++ is a good choice, as you've selected. It provides low-level control needed for emulation.
   * **Control**: Greater control over system resources and memory management.
   * **Extensive Libraries**: Access to numerous libraries for various functionalities.

### **Tools and Frameworks:**

* **Boost**: For a wide range of utility libraries.
* **Google Test**: For unit testing.
* **CMake**: For build automation.
* **GitHub/GitLab**: For version control.

​ This setup will help us stay organized and allow us to focus on coding the emulator’s core functions in the coming weeks.