## Week 1: Project Planning & Setup

The project begins with defining the core structure of a virtual CPU emulator. The `Memory` class is implemented to simulate RAM with read and write operations.

### Memory Class

- **`__init__(self, size=256)`**: Initializes memory with a fixed size (default 256 bytes).
- **`read(self, address)`**: Retrieves the value stored at a given memory address.
- **`write(self, address, value)`**: Stores a value at a given memory address.

## Week 2: Instruction Set Architecture (ISA)

A `Registers` class is introduced to store CPU registers and track execution.

### Registers Class

- **`__init__(self)`**: Initializes four general-purpose registers (`R0-R3`), a program counter (`pc`), and an instruction register (`ir`).

## Week 3: Basic CPU Components

The ALU (Arithmetic Logic Unit) is implemented to perform arithmetic operations.

### ALU Class

- **`execute(op, operand1, operand2)`**: Performs basic operations:
  - ADD: Adds two operands.
  - SUB: Subtracts the second operand from the first.

## Week 4: Instruction Execution

The CPU class is implemented, combining memory, registers, and an instruction execution model.

### CPU Class

- **`__init__(self)`**: Initializes memory, registers, and a dictionary of supported instructions.
- **`fetch(self)`**: Retrieves the next instruction from memory using the program counter.
- **`decode_execute(self)`**: Decodes and executes the instruction.

## Week 5: Memory Management

Basic memory handling operations are implemented.

- **`load(self, reg, addr)`**: Loads a value from memory into a register.
- **`store(self, reg, addr)`**: Stores a register's value into memory.

## Week 6: I/O Operations

Arithmetic operations are supported through the ALU.

- **`add(self, reg1, reg2)`**: Adds values from two registers and stores the result in the first register.
- **`sub(self, reg1, reg2)`**: Subtracts the value in the second register from the first.

## Week 7: Advanced Features

Control flow features such as jumps and printing are added.

- **`jmp(self, addr)`**: Sets the program counter to a new address.
- **`print_reg(self, reg)`**: Prints the value of a given register.

### Instruction Execution

The `run(self, program)` method:

1. Stores the program in memory.
2. Executes each instruction sequentially.

## Week 8: Performance Optimization

The program execution flow is optimized for efficiency.

## Week 9: Final Testing & Debugging

A simple test program is provided.

### Test Program

```
assembly
CopyEdit
LOAD R0 10
LOAD R1 20
ADD R0 R1
PRINT R0
```

- Loads values into registers.
- Performs addition.
- Prints the result (`30`).