“Module 3: In-Class Assignment - Little Man Computer”

Harsh Siddhapura

“Dr. Dinesh Sthapit”

“September 11, 2023”

“Little Man Computer”

Limitations of Code

The Little Man Computer (LMC) implemented is a simplified model of a computer and has several limitations, as it is intentionally designed to be a simple educational tool. Here are some of its limitations:

- Limited Instruction Set: The LMC has a very basic instruction set with only four instructions: LDA (Load), ADD (Add), OUT (Output), and HLT (Halt). This limited set of instructions makes it impractical for more complex tasks and real-world applications.
- Fixed Memory Size: The LMC has a fixed memory size of 16 locations, which is very small compared to modern computers. This limits the amount of data and instructions that can be stored and processed.
- No Support for Variables: The LMC does not support variables or symbolic names for memory locations. All memory addresses are specified directly in the instructions, which can make programs less readable and harder to maintain.
- No Arithmetic Operations: The LMC only supports addition (ADD) as an arithmetic operation. It lacks other basic arithmetic and logical operations like subtraction, multiplication, division, and comparison.
- Limited Input/Output: The LMC only supports output through the OUT instruction. It lacks any input capability, which means it cannot interact with the user or read data from external sources.
- No Conditionals or Loops: The LMC lacks conditional statements (e.g., if-else) and loops (e.g., for, while), making it challenging to implement complex algorithms or decision-making processes.
- Limited Data Representation: The LMC uses simple integer values for data representation, which limits its ability to work with different data types and precision.
- Single Accumulator: The LMC has only one general-purpose register (accumulator), which can be a limitation when performing complex computations that require multiple registers.

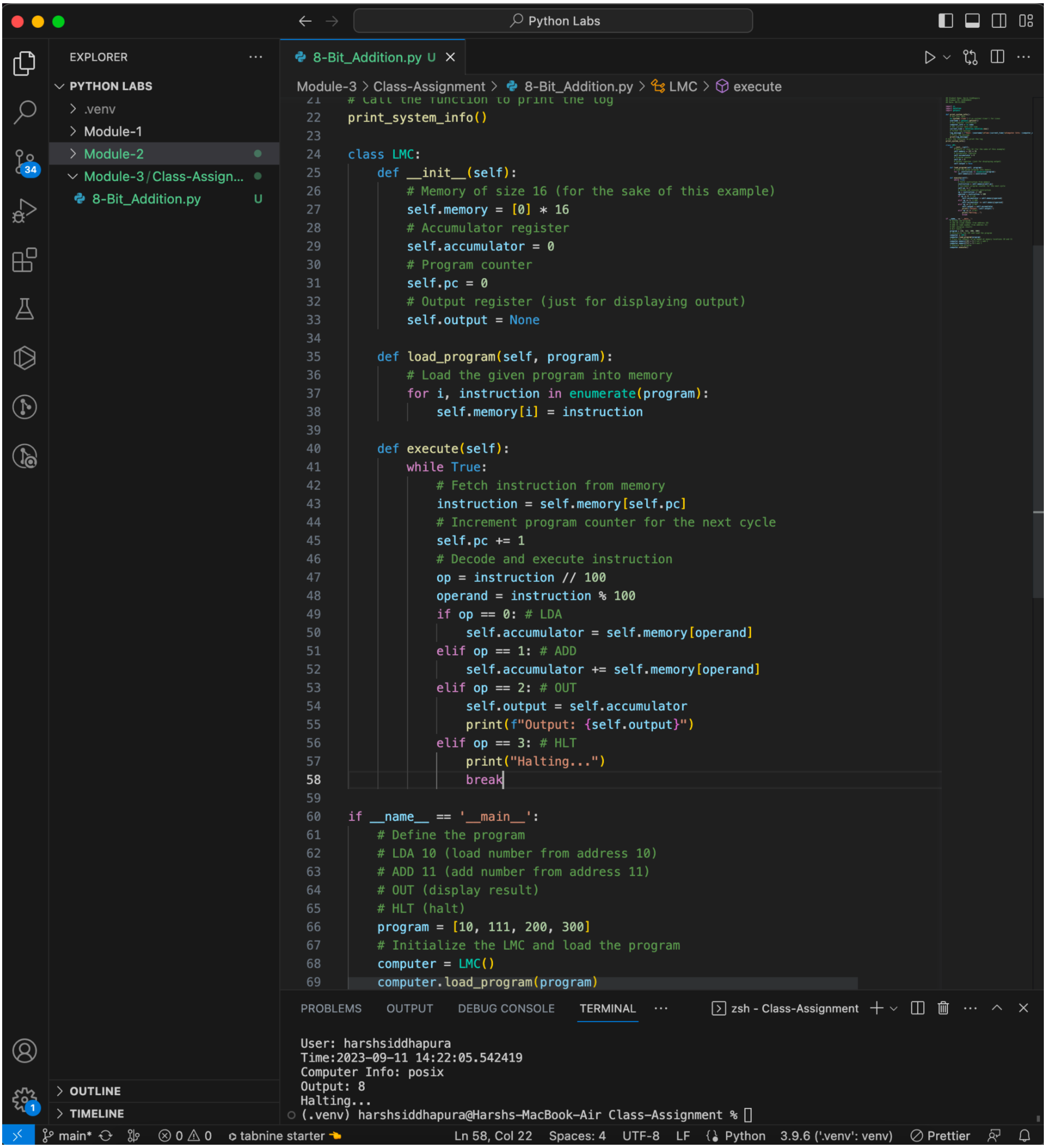
In summary, the LMC is a very simplified and constrained model of a computer, and while it is useful for educational purposes to understand fundamental concepts, it is not suitable for practical programming or solving real-world problems due to its limitations.

Summary

The provided Python code defines a simple emulation of the Little Man Computer (LMC), a simplified educational model of a computer. Here's a summary of the code's functionality and structure:

- Class Definition (LMC): The code defines a class named `LMC` that represents the Little Man Computer. It includes attributes such as memory, an accumulator register, a program counter, and an output register for displaying results.
- Memory Initialization: The memory of LMC is initialized to have 16 locations, each initially set to 0.
- Loading Programs: The `load_program` method allows you to load a program (a list of instructions) into the LMC's memory. The instructions are stored in memory starting from the first location (location 0).
- Execution Loop: The `execute` method is responsible for executing the loaded program. It enters an infinite loop where it repeatedly fetches instructions from memory and processes them.
- Instruction Set: The LMC supports a simple instruction set with the following operations:
    - LDA (Load Accumulator): Load a value from a specified memory location into the accumulator.
    - ADD (Add to Accumulator): Add a value from a specified memory location to the accumulator.
    - SUB (Subtract from Accumulator): Subtract a value from a specified memory location from the accumulator (added in the extended code).
    - MUL (Multiply Accumulator): Multiply the accumulator by a value from a specified memory location (added in the extended code).
    - OUT (Output): Display the value in the accumulator.
    - HLT (Halt): Terminate program execution.
- Execution Flow: The program counter (PC) is incremented after each instruction execution, allowing the LMC to proceed to the next instruction in memory.
- Output Display: When the `OUT` instruction is encountered, the LMC prints the value stored in the accumulator as output.
- Initialization and Program Execution: In the `__main__` section, a sample program is defined, which loads values from specific memory locations, performs arithmetic operations (addition, subtraction, and multiplication), displays the results, and halts the execution.

8-Bit Addition

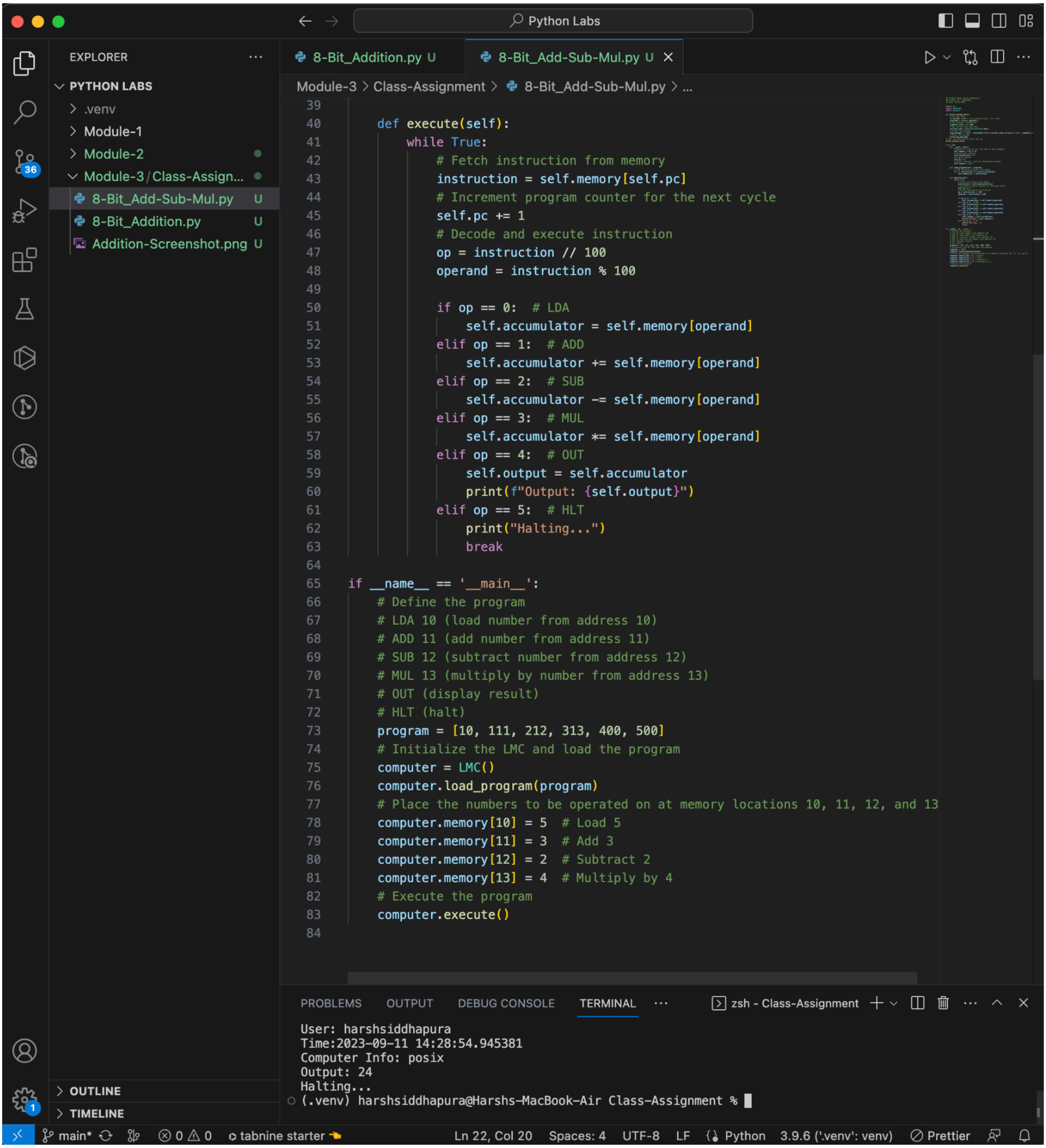


Python Labs
EXPLORER
PYTHON LABS
.venv
Module-1
Module-2
Module-3 / Class-Assign...
8-Bit_Addition.py U
8-Bit_Addition.py U
Module-3 > Class-Assignment > 8-Bit_Addition.py > LMC > execute
OUTLINE
TIMELINE
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
zsh - Class-Assignment
main* ⊗ 0 ⚠ 0 tabnine starter
Ln 58, Col 22 Spaces: 4 UTF-8 LF Python 3.9.6 ('.venv': venv) Prettier

```python
# Call the function to print the log
print_system_info()

class LMC:
    def __init__(self):
        # Memory of size 16 (for the sake of this example)
        self.memory = [0] * 16
        # Accumulator register
        self.accumulator = 0
        # Program counter
        self.pc = 0
        # Output register (just for displaying output)
        self.output = None

    def load_program(self, program):
        # Load the given program into memory
        for i, instruction in enumerate(program):
            self.memory[i] = instruction

    def execute(self):
        while True:
            # Fetch instruction from memory
            instruction = self.memory[self.pc]
            # Increment program counter for the next cycle
            self.pc += 1
            # Decode and execute instruction
            op = instruction // 100
            operand = instruction % 100
            if op == 0: # LDA
                self.accumulator = self.memory[operand]
            elif op == 1: # ADD
                self.accumulator += self.memory[operand]
            elif op == 2: # OUT
                self.output = self.accumulator
                print(f"Output: {self.output}")
            elif op == 3: # HLT
                print("Halting...")
                break

if __name__ == '__main__':
    # Define the program
    # LDA 10 (load number from address 10)
    # ADD 11 (add number from address 11)
    # OUT (display result)
    # HLT (halt)
    program = [10, 111, 200, 300]
    # Initialize the LMC and load the program
    computer = LMC()
    computer.load_program(program)
```

```
User: harshsiddhapura
Time:2023-09-11 14:22:05.542419
Computer Info: posix
Output: 8
Halting...
(.venv) harshsiddhapura@Harshs-MacBook-Air Class-Assignment %
```

8-Bit Addition, Subtraction and Multiplication



```python

    def execute(self):
        while True:
            # Fetch instruction from memory
            instruction = self.memory[self.pc]
            # Increment program counter for the next cycle
            self.pc += 1
            # Decode and execute instruction
            op = instruction // 100
            operand = instruction % 100

            if op == 0:  # LDA
                self.accumulator = self.memory[operand]
            elif op == 1:  # ADD
                self.accumulator += self.memory[operand]
            elif op == 2:  # SUB
                self.accumulator -= self.memory[operand]
            elif op == 3:  # MUL
                self.accumulator *= self.memory[operand]
            elif op == 4:  # OUT
                self.output = self.accumulator
                print(f"Output: {self.output}")
            elif op == 5:  # HLT
                print("Halting...")
                break

if __name__ == '__main__':
    # Define the program
    # LDA 10 (load number from address 10)
    # ADD 11 (add number from address 11)
    # SUB 12 (subtract number from address 12)
    # MUL 13 (multiply by number from address 13)
    # OUT (display result)
    # HLT (halt)
    program = [10, 111, 212, 313, 400, 500]
    # Initialize the LMC and load the program
    computer = LMC()
    computer.load_program(program)
    # Place the numbers to be operated on at memory locations 10, 11, 12, and 13
    computer.memory[10] = 5  # Load 5
    computer.memory[11] = 3  # Add 3
    computer.memory[12] = 2  # Subtract 2
    computer.memory[13] = 4  # Multiply by 4
    # Execute the program
    computer.execute()

```

```
User: harshsiddhapura
Time:2023-09-11 14:28:54.945381
Computer Info: posix
Output: 24
Halting...
(.venv) harshsiddhapura@Harshs-MacBook-Air Class-Assignment %
```

The code provides output for 5 + 3 - 2 * 4 = 24