High Level Summary: A simulator that supports I/O, load and store, arithmetic, and control operations.

User Stories:

1) As a CS student I want to be able to apply arithmetic operations to data to understand how the register works with operations on data.
2) As a CS student I want to be able to read and write to a file to load and store data in memory to understand how loading data into memory works.

Use Cases:

1. Read from Keyboard
   - Actor: User (or another part of the program simulating user input for testing)
   - System: UVSim (Universal Virtual Machine Simulator) `IO` Class
   - Goal: To read a word from the keyboard into a specific location in memory.
   - Steps:
     1. The Actor invokes the `read` method of the `IO` class.
     2. The System prompts the Actor for a signed four-digit number.
     3. The Actor inputs a signed four-digit number.
     4. The System validates the input and stores it in the specified memory location.
   - Extensions:
     - If the input is not a signed four-digit number, the System prompts the Actor again until valid input is provided.
2. Write to Screen
   - Actor: System/User (The system initiates the action, but it could be in response to a user command or program algorithm)
   - System: UVSim `IO` Class
   - Goal: To write a word from a specific location in memory to the screen.
   - Steps:
     1. The Actor requests the `write` method of the `IO` class with a specific memory location.
     2. The System retrieves the value from the specified memory location.
     3. The System displays the value on the screen.
3. Load
   - Actor: System/User
   - System: UVSim ‘Memory’ Class
   - Goal: Load a word into the accumulator from a memory location.
   - Steps:
     1. Parse function code.
     2. Get desired memory address from the last 2 digits of the word.
     3. Go to that memory address and retrieve the value.
     4. Copy retrieved value into the accumulator register.
4. Store
   - Actor: System/User
   - System: UVSim ‘Memory’ Class

- Goal: Store a word from the accumulator into a memory location.
- Steps:
    1. Parse function code.
    2. Copy value from the accumulator register.
    3. Store value from the accumulator into the specified memory location.

5. Add
    - Actor: system / user
    - System: UVSim ‘Arithmetic’ Class
    - Goal: Successfully add the word in the accumulator to a word from a specific memory location, and load the sum into the accumulator.
    - Steps:
        1. Parse function code.
        2. Identify the target memory address from the last 2 digits of the word.
        3. Retrieve the value from the identified memory address.
        4. Add the memory value to the accumulator value.
        5. Copy the sum value into the accumulator register.
6. Subtract
    - Actor: program / user
    - System: UVSim ‘Arithmetic’ Class
    - Goal: Successfully subtract the word from a specific memory location from the word in the accumulator, and load the difference into the accumulator.
    - Steps:
        1. Parse function code.
        2. Identify the target memory address from the last 2 digits of the word.
        3. Retrieve the value from the identified memory address.
        4. Subtract the memory value from the accumulator value.
        5. Copy the difference into the accumulator register.
7. Divide
    - Actor: program / user
    - System: UVSim ‘Arithmetic’ Class
    - Goal: Successfully divide the word in the accumulator by the word in a specific memory location, and load the result into the accumulator.
    - Steps:
        1. Parse function code.
        2. Identify the target memory address from the last 2 digits of the word.
        3. Retrieve the value from the identified memory address.
        4. Divide the accumulator value by the memory value.
        5. Copy the resulting value into the accumulator register.
8. Multiply
    - Actor: program / user
    - System: UVSim ‘Arithmetic’ Class
    - Goal: Successfully multiply the word in the accumulator by the word in a specific memory location, and load the result into the accumulator.
    - Steps:

    1. Parse function code.
    2. Identify the target memory address from the last 2 digits of the word.
    3. Retrieve the value from the identified memory address.
    4. Multiply the accumulator value by the memory value.
    5. Copy the resulting value into the accumulator register.
9. Branch
    - Actor: program / user
    - System: UVSim ‘Control’ Class
    - Goal: Successfully branch to a designated location in memory.
    - Steps:
        1. User will input code 40(memory location) in to their text file
        2. uvSim will read file that user has selected
        3. uvSim will select the branch function
        4. uvSim branch function with change the memory location
        5. uvSim will verify that the memory location is correct and if it is not, output an error message to the console.
        6. uvSim will be successful if the memory location has changed.
10. Branch Neg
    - Actor: program / user
    - System: UVSim ‘Control’ Class
    - Goal: Successfully branch to a designated location in memory if the accumulator is negative.
    - Steps:
        1. User will input code 41(memory location) in their text file.
        2. uvSim will read the file that the user has selected.
        3. uvSim’s branchneg function will be called while iterating over memory
        4. uvSim will then check the accumulator to check if it negative
        5. If the accumulator is negative it will branch to the new memory location
        6. If the accumulator is not negative it will continue to the next memory location
        7. uvSim will be successful if the memory location has changed or continued according to the accumulator.
11. Branch Zero
    - Actor: program / user
    - System: UVSim ‘Control’ Class
    - Goal: Successfully branch to a designated location in memory if the accumulator is 0.
    - Steps:
        1. User will input code 42(memory location) in their text file.
        2. uvSim will read the file that the user has selected.
        3. uvSim’s branchzero function will be called while iterating over memory
        4. uvSim will then check the accumulator to check if it is equal to zero
        5. If the accumulator is zero it will branch to the new memory location
        6. If the accumulator is not zero it will continue to the next memory location
        7. uvSim will be successful if the memory location has changed or continued according to the accumulator.

12. Halt Program Execution
    - Actor: System (Part of the program/algorithm that determines execution should stop)
    - System: UVSim `Control` Class
    - Goal: To stop the program execution.
    - Steps:
        1. The Actor encounters a condition or instruction that requires halting the program (e.g., a specific instruction in the code or an external interrupt).
        2. The Actor invokes the `halt` method of the `Control` class.
        3. The System sets the program state to indicate that execution should stop.
        4. The program ceases further instruction processing and exits the execution loop.
    - Extensions:
        - The halt condition can be triggered manually by a user or automatically by reaching a predefined state in the program's execution.
13. Color Scheme
    - Actor: User
    - System: UVSim tkinterApp Class
    - Goal: To allow users to change the program’s color scheme.
    - Steps:
        1. The Actor clicks the “Options” button.
        2. The Actor chooses whether they will change the primary or secondary color.
        3. The Actor selects a color and presses “Ok.”
        4. When they choose a file, the new colors will be displayed.

14. Select a File
    - Actor: User
    - System: UVSim Memory Class
    - Goal: To allow users to select a file from the computer directory.
    - Steps:
        1. The Actor clicks the “Choose a File” button.
        2. The Actor browses through their directory to find a file to run through UVSim.
        3. The Actor selects a file and presses “Open”.
        4. The program displays the file information.
        5. The Actor is able to select “Run File” in order to run the file.

15. Select a New File
    - Actor: User
    - System: UVSim Memory Class
    - Goal: To allow users to select a new file without reopening the program.
    - Steps:
        1. The Actor clicks the “Load New” button.
        2. The Actor browses through their computer directory to find a new file to run.
        3. The Actor selects a file and presses “Open”.
        4. The program displays the file information, and is able to press “Run File”.

Naming Conventions:

- Functions:
    - Define all functions with snake_case
- Classes:
    - Start with Capital letters.
- Docstrings
    - Clearly describe what the function is doing inside of the function.