**Design Document for Python Application**

**High-Level Functionality**

This Python application reads and processes instructions from a text file, simulating a simple machine language computer (UVSim). It supports various operations such as reading, writing, loading, storing, arithmetic operations, branching, and halting.

**User Stories**

1. As a Computer Science Student, I want to load a BasicML program into the UVSim so that I can execute machine language instructions and observe their results.
2. As an Instructor, I want to provide students with a reliable and easy-to-use simulator so they can learn about machine language and computer architecture effectively.

**Use Cases**

**1. Load Program**

* **Description**: The system loads a BasicML program from a file into memory.
* **Preconditions**: The program file exists and is accessible.
* **Postconditions**: The program is loaded into memory starting at address 0.
* **Actors**: System
* **Trigger**: User initiates the program loading process.
* **Flow**:
  1. User provides the file name.
  2. The system reads the instructions from the file.
  3. The system loads the instructions into memory starting at address 0.

**2. Read Data**

* **Description**: The system reads a number from the user and stores it in a specified memory location.
* **Preconditions**: The program is running and a read instruction is encountered.
* **Postconditions**: The number is stored in the specified memory location.
* **Actors**: System
* **Trigger**: A read instruction (10xx) is executed.
* **Flow**:
  1. The system prompts the user to enter a number.
  2. The user enters the number.
  3. The system stores the number in the specified memory location.

**3. Write Data**

* **Description**: The system writes a number from a specified memory location to the output.
* **Preconditions**: The program is running and a write instruction is encountered.
* **Postconditions**: The number is written to the output.
* **Actors**: System
* **Trigger**: A write instruction (11xx) is executed.
* **Flow**:
  1. The system retrieves the number from the specified memory location.
  2. The system writes the number to the output.

**4. Load Accumulator**

* **Description**: The system loads a number from a specified memory location into the accumulator.
* **Preconditions**: The program is running and a load instruction is encountered.
* **Postconditions**: The accumulator holds the number from the specified memory location.
* **Actors**: System
* **Trigger**: A load instruction (20xx) is executed.
* **Flow**:
  1. The system retrieves the number from the specified memory location.
  2. The system loads the number into the accumulator.

**5. Store Accumulator**

* **Description**: The system stores the number in the accumulator into a specified memory location.
* **Preconditions**: The program is running and a store instruction is encountered.
* **Postconditions**: The specified memory location holds the number from the accumulator.
* **Actors**: System
* **Trigger**: A store instruction (21xx) is executed.
* **Flow**:
  1. The system retrieves the number from the accumulator.
  2. The system stores the number in the specified memory location.

**6. Add to Accumulator**

* **Description**: The system adds a number from a specified memory location to the accumulator.
* **Preconditions**: The program is running and an add instruction is encountered.
* **Postconditions**: The accumulator holds the sum of its original value and the number from the specified memory location.
* **Actors**: System
* **Trigger**: An add instruction (30xx) is executed.
* **Flow**:
  1. The system retrieves the number from the specified memory location.
  2. The system adds the number to the accumulator.

**7. Subtract from Accumulator**

* **Description**: The system subtracts a number from a specified memory location from the accumulator.
* **Preconditions**: The program is running and a subtract instruction is encountered.
* **Postconditions**: The accumulator holds the result of the subtraction.
* **Actors**: System
* **Trigger**: A subtract instruction (31xx) is executed.
* **Flow**:
  1. The system retrieves the number from the specified memory location.
  2. The system subtracts the number from the accumulator.

**8. Divide Accumulator**

* **Description**: The system divides the accumulator by a number from a specified memory location.
* **Preconditions**: The program is running and a divide instruction is encountered.
* **Postconditions**: The accumulator holds the result of the division.
* **Actors**: System
* **Trigger**: A divide instruction (32xx) is executed.
* **Flow**:
  1. The system retrieves the number from the specified memory location.
  2. The system divides the accumulator by the retrieved number.
  3. The system handles division by zero by displaying an error message.

**9. Multiply Accumulator**

* **Description**: The system multiplies the accumulator by a number from a specified memory location.
* **Preconditions**: The program is running and a multiply instruction is encountered.
* **Postconditions**: The accumulator holds the result of the multiplication.
* **Actors**: System
* **Trigger**: A multiply instruction (33xx) is executed.
* **Flow**:
  1. The system retrieves the number from the specified memory location.
  2. The system multiplies the accumulator by the retrieved number.

**10. Branch**

* **Description**: The system branches to a specific memory location.
* **Preconditions**: The program is running and a branch instruction is encountered.
* **Postconditions**: The instruction counter is set to the specified memory location.
* **Actors**: System
* **Trigger**: A branch instruction (40xx) is executed.
* **Flow**:
  1. The system sets the instruction counter to the specified memory location.

**11. Branch if Negative**

* **Description**: The system branches to a specific memory location if the accumulator is negative.
* **Preconditions**: The program is running and a branch if negative instruction is encountered.
* **Postconditions**: If the accumulator is negative, the instruction counter is set to the specified memory location.
* **Actors**: System
* **Trigger**: A branch if negative instruction (41xx) is executed.
* **Flow**:
  1. The system checks if the accumulator is negative.
  2. If negative, the system sets the instruction counter to the specified memory location.

**12. Branch if Zero**

* **Description**: The system branches to a specific memory location if the accumulator is zero.
* **Preconditions**: The program is running and a branch if zero instruction is encountered.
* **Postconditions**: If the accumulator is zero, the instruction counter is set to the specified memory location.
* **Actors**: System
* **Trigger**: A branch if zero instruction (42xx) is executed.
* **Flow**:
  1. The system checks if the accumulator is zero.
  2. If zero, the system sets the instruction counter to the specified memory location.

**13. Halt Program**

* **Description**: The system stops the execution of the BasicML program.
* **Preconditions**: The program is running and a halt instruction is encountered.
* **Postconditions**: Program execution is halted.
* **Actors**: System
* **Trigger**: A halt instruction (43xx) is executed.
* **Flow**:
  1. The system stops the execution of the program.
  2. The system displays a message indicating that the program has halted.

**14. Execute Instruction**

* **Description**: The system decodes and executes a single instruction.
* **Preconditions**: The program is loaded into memory.
* **Postconditions**: The instruction is executed, and the system state is updated accordingly.
* **Actors**: System
* **Trigger**: An instruction is fetched from memory.
* **Flow**:
  1. The system decodes the instruction.
  2. The system executes the instruction.
  3. The system updates the state (e.g., accumulator, memory, instruction counter).

**15. Run Program**

* **Description**: The system runs a BasicML program from start to finish.
* **Preconditions**: The program is loaded into memory.
* **Postconditions**: The program is executed, and the final state is displayed.
* **Actors**: System
* **Trigger**: User initiates the program execution.
* **Flow**:
  1. User starts the program.
  2. The system fetches and executes instructions sequentially.
  3. The system displays the final state of the program.