Design Document

1. Introduction

The UVSim is a simple virtual machine, but powerful. The UVSim can only interpret a machine language called BasicML.

2. User Stories

2.1 User Story 1: Learning Machine Language

As a computer science student, I want to learn and understand machine language, so I can execute BasicML programs on the UVSim simulator and enhance my skills.

2.2 User Story 2: Experimenting with Code

As a computer science student, I want to experiment with code in a safe and controlled environment, so that I can make mistakes, learn from them, and improve my coding skills.

3. Use Cases

3.1 Use Case 1: Load Program into Memory

Actor: User

System Goal: Load a BasicML program into the UVSim memory starting at location 00.

Steps:

1. Ask user for BasicML program file location.
2. Open the specified file.
3. Parse each line of the file into a memory register starting at location 00.

3.2 Use Case 2: Input/Output Operations

Actor: User

System Goal: User utilizes READ & WRITE operations for input and output to and from specific memory locations.

Steps:

1. Parse the function code for READ or WRITE operation.
2. Identify the memory address from the last two digits of the function code.

3. If READ operation, fetch input from the user and store it in the identified memory address. If WRITE operation, retrieve data from the identified memory address and display it as output.

3.3 Use Case 3: Load and Store Operations

Actor: UVSim

System Goal: Manage data within the memory and accumulator using LOAD & STORE operations.

Steps:

1. Parse the function code for LOAD or STORE operation.
2. Identify the memory address from the last two digits of the function code.
3. If LOAD operation, move data from the identified memory address into the accumulator. If STORE operation, move data from the accumulator into the identified memory address.

3.4 Use Case 4: Perform Arithmetic Operations

Actor: UVSim

System Goal: Perform arithmetic operations (ADD, SUBTRACT, DIVIDE, and MULTIPLY) on words in the memory and accumulator.

Steps:

1. Parse the function code for ADD, SUBTRACT, DIVIDE, or MULTIPLY operation.
2. Identify the memory address from the last two digits of the function code.
3. Fetch value from the identified memory address.
4. Perform the specified arithmetic operation using the fetched value and the current value in the accumulator. Store the result in the accumulator.

3.5 Use Case 5: Control Operations

Actor: UVSim

System Goal: Use control operations (BRANCH, BRANCHNEG, BRANCHZERO, and HALT) to control the flow of the program.

Steps:

1. Parse the function code for BRANCH, BRANCHNEG, BRANCHZERO, or HALT operation.
2. Depending on the operation, jump to a specific memory address, halt the program, or branch based on the value in the accumulator.

3.6 Use Case 6: Debugging

Actor: UVSim

System Goal: Display specific errors based on user input (ex. division by zero, invalid operation, invalid memory address).

Steps:

1. Catch exceptions and errors during the execution of the program.
2. Display specific, meaningful error messages to the user when exceptions or errors occur.

3.7 Use Case 7: Launch Simulator from Command Line

Actor: User

System Goal: Launch UVSim from the command line.

Steps:

1. User enters the command to run the UVSim in the command line.
2. UVSim launches and prompts the user for the next action.

3.8 Use Case 8: Interpret BasicML Instructions

Actor: UVSim

System Goal: Interpret BasicML instructions and execute them accordingly.

Steps:

1. Fetch the next instruction from memory.
2. Parse and understand the function code from the instruction.
3. Execute the operation specified by the function code.

3.9 Use Case 9: Fetch Instructions From Memory

Actor: UVSim

System Goal: Fetch the next instructions from memory to be executed.

Steps:

1. Identify the memory address of the next instruction.
2. Retrieve the instruction from the identified memory address.
3. Load the instruction into a register for execution.

3.10 Use Case 10: Pause & Resume Execution

Actor: User

System Goal: The user can halt and resume the program execution.

Steps:

1. User inputs a command to pause the execution of the program.
2. UVSim halts execution and maintains current state.
3. Upon user command, UVSim resumes execution from the point where it was halted.

4. Architecture and Components

The UVSim will contain the following main components:

- **CPU**: This will be responsible for interpreting and executing BasicML instructions.
- **Register**: This will hold information before it is used in calculations or examined in various ways.
- **Main Memory**: Equipped with a 100-word memory, and these words are referenced by their location numbers 00, 01, ..., 99. The BasicML program must be loaded into the main memory starting at location 00 before executing. Each instruction written in BasicML occupies one word of the UVSim memory (instruction are signed four-digit decimal number). We shall assume that the sign of a BasicML instruction is always plus, but the sign of a data word may be either plus or minus. Each location in the UVSim memory may contain an instruction, a data value used by a program or an unused area of memory. The first two digits of each BasicML instruction are the operation code specifying the operation to be performed.
- **Accumulator**: A register into which information is put before the UVSim uses it in calculations or examines it in various ways.

5. Technologies and Tools

- **Programming Language**: Python
- **Version Control System**: Git