# UVSim Design Document

## High-Level Functionality

UVSim is a simple software simulator built to help computer science students learn machine language and computer architecture. It features a 100-word memory, the ability to load and execute programs written in Basic Machine Language (BasicML), and an accumulator register. Each word is a signed 4-digit number, where the first two digits represent the operation codes and the last two digits represent the memory address.

**UVSim can perform:**

- I/O: READ, WRITE
- Arithmetic: ADD, SUBTRACT, MULTIPLY, DIVIDE
- Load/Store: LOAD, STORE
- Control: BRANCH, BRANCHNEG, BRANCHZERO, HALT

Programs load into memory at address 00 and stop at a HALT instruction. The simulator currently runs in the command line or through a GUI (Tkinter), where users can open a program file, provide inputs, and view outputs.

## User Stories

- As a student, I want to run a BasicML program on UVSim so I can practice and see how the instructions work.
- As a professor, I want UVSim to show the result of the program so students can understand how the instructions change the memory and the accumulator.

## Use Cases

### Use Case #1: READ Input

- Actor: User
- System: UVSim I/O + Memory
- Goal: Store a user-entered number at the target memory location.
- Steps:
    1. Prompt user for input.
    2. Validate input (must be between −9999 and +9999).
    3. Store the value at memory[xx].
    4. Advance instruction pointer.

### Use Case #2: WRITE Output

- Actor: Processor
- System: UVSim I/O + Memory
- Goal: Display a value from memory.
- Steps:
  1. Fetch value from memory[xx].
  2. Send value to output (console or GUI output box).
  3. Advance instruction pointer.

**Use Case #3: LOAD**

- Actor: Processor
- System: Memory + Accumulator
- Goal: Load a value from memory into the accumulator.
- Steps:
  1. Parse instruction (20xx).
  2. Fetch value from memory[xx].
  3. Copy value into accumulator.
  4. Advance instruction pointer.

**Use Case #4: STORE**

- Actor: Processor
- System: Memory + Accumulator
- Goal: Save the accumulator into memory.
- Steps:
  1. Parse instruction (21xx).
  2. Copy accumulator value.
  3. Write value to memory[xx].
  4. Advance instruction pointer.

**Use Case #5: ADD**

- Actor: Processor
- System: ALU + Memory + Accumulator
- Goal: Add a value in memory to the accumulator.
- Steps:
  1. Parse instruction (30xx).
  2. Fetch value from memory[xx].
  3. Add value to accumulator.
  4. Truncate to 4 digits if overflow.
  5. Advance instruction pointer.

**Use Case #6: SUBTRACT**

- Actor: Processor
- System: ALU + Memory + Accumulator
- Goal: Subtract a value in memory from the accumulator.
- Steps:
  1. Parse instruction (31xx).
  2. Fetch value from memory[xx].
  3. Subtract value from accumulator.
  4. Truncate to 4 digits if overflow.
  5. Advance instruction pointer.

**Use Case #7: MULTIPLY**

- Actor: Processor
- System: ALU + Memory + Accumulator
- Goal: Multiply accumulator by a memory value.
- Steps:
  1. Parse instruction (33xx).
  2. Fetch value from memory[xx].
  3. Multiply accumulator by the value.
  4. Truncate to 4 digits if overflow.
  5. Advance instruction pointer.

**Use Case #8: DIVIDE**

- Actor: Processor
- System: ALU + Memory + Accumulator
- Goal: Divide accumulator by a memory value.
- Steps:
  1. Parse instruction (32xx).
  2. Fetch divisor from memory[xx].
  3. If divisor = 0 → raise error.
  4. Otherwise, accumulator = accumulator // divisor.
  5. Advance instruction pointer.

**Use Case #9: BRANCH**

- Actor: Processor
- System: Instruction Pointer
- Goal: Jump to a memory address unconditionally.
- Steps:
  1. Parse instruction (40xx).
  2. Set instruction pointer = operand.

3. Continue execution from that address.

**Use Case #10: BRANCHNEG**

- Actor: Processor
- System: Instruction Pointer + Accumulator
- Goal: Conditionally branch if accumulator is negative.
- Steps:
  1. Parse instruction (41xx).
  2. If accumulator < 0 → set instruction pointer = operand.
  3. Otherwise, increment instruction pointer.

**Use Case #11: BRANCHZERO**

- Actor: Processor
- System: Instruction Pointer + Accumulator
- Goal: Conditionally branch if accumulator = 0.
- Steps:
  1. Parse instruction (42xx).
  2. If accumulator == 0 → set instruction pointer = operand.
  3. Otherwise, increment instruction pointer.

**Use Case #12: HALT**

- Actor: Processor
- System: Execution Controller
- Goal: Stop program execution.
- Steps:
  1. Parse instruction (43xx).
  2. Set running = False.
  3. Program terminates.

**Use Case #13: Handle Invalid Opcode**

- Actor: Processor
- System: Error Handler
- Goal: Gracefully handle unknown instructions.
- Steps:
  1. Detect invalid opcode.
  2. Raise error message (console or GUI popup).
  3. Stop program execution.

**Use Case #14: Handle Out-of-Range Input**

- Actor: User
- System: Input Validator
- Goal: Ensure entered values are within range.
- Steps:
  1. Prompt user for input.
  2. If value < −9999 or > +9999 → show warning.
  3. Ask user to re-enter until valid.
  4. Store valid input in memory.

**Use Case #15: Execute Full Program**

- Actor: User (initiates), Processor (executes)
- System: Loader + Execution Loop
- Goal: Run a BasicML program to completion.
- Steps:
  1. Load program lines into memory[00–99].
  2. Fetch instruction at instruction pointer.
  3. Decode opcode and operand.
  4. Execute instruction.
  5. Repeat until HALT or error.