**UVSim User Stories and Use Cases**

**User Stories**

As a computer science student,   
I want to load and execute BasicML programs on the UVSim simulator,   
so that I can observe how low-level machine operations affect memory and computation.

As a professor of Computer Organization and Architecture at UVU,   
I want to use a simple computer language that can run I/O and Load/Store operations,   
so that I can teach my students basic memory allocation on a computer.

**Use Cases**

Actor: UVSim Execution

System: Parses the operation 30 from a memory instruction. Gets the memory address from the same line of the operation instruction. Fetches the value from the memory location. Adds the value to the accumulator. Checks for overflow. Updates the accumulator with the result.

Goal: Perform addition operation

Actor: UVSim Execution

System: Parses the operation 31 from a text file containing BasicML instruction. Gets the memory address from the same line of the operation instruction. Fetches the value from the memory location. Subtracts the value from the accumulator. Checks for overflow. Updates the accumulator with the result.

Goal: Perform subtraction operation

Actor: UVSim Execution

System: Parses the operation 32 from a text file containing BasicML instruction. Gets the memory address from the same line of the operation instruction from the same line of the operation instruction. Fetches the value from the memory location. Divides the accumulator by the value. Handles division by zero. Stores the result in the accumulator.

Goal: Perform division operation

Actor: UVSim Execution

System: Parses the operation 33 from a text file containing BasicML instruction. Gets the memory address from the same line of the operation instruction from the same line of the operation instruction. Fetches the value from the memory location. Multiplies it with the value in the accumulator. Checks for overflow or underflow. Updates the accumulator with the result.

Goal: Perform multiplication operation

Actor: UVSim Loader

System: Accepts a user-provided text file containing BasicML. Parses instructions from the file. Loads instructions and data into main memory sequentially starting at location 00.

Goal: Load a BasicML program into UVSim memory

Actor: User

System: Provides keyboard input during program execution. UVSim stores the input into the specified memory location.

Goal: Read input into memory (BasicML operation 10)

Actor: UVSim Execution

System: Parses the operation 11 from a text file containing BasicML instruction. Gets the memory address from the same line of the operation instruction from the same line of the operation instruction. Outputs the value stored in the memory address to screen.

Goal: Write output from memory to screen

Actor: UVSim Execution

System: Parses the operation 20 from a text file containing BasicML instruction. Gets the memory address from the same line of the operation instruction from the same line of the operation instruction. Loads the value from the memory location into the accumulator.

Goal: Load value into accumulator

Actor: UVSim Execution

System: Parses the operation 21 from a text file containing BasicML instruction. Gets the memory address from the same line of the operation instruction from the same line of the operation instruction. Stores the value from the accumulator into the memory location.

Goal: Store accumulator value into memory

Actor: UVSim Execution

System: Parses the operation 40 from a text file containing BasicML instruction. Jumps to the specified memory address.

Goal: branch

Actor: UVSim Execution

System: Parses the operation 41 from a text file containing BasicML instruction. If the accumulator is negative, jumps to the specified memory address.

Goal: branch on negative

Actor: UVSim Execution

System: Parses the operation 42 from a text file containing BasicML instruction. If the accumulator is zero, jumps to the specified memory address.

Goal: branch on zero

Actor: UVSim Execution

System: Parses the operation 43 from a text file containing BasicML instruction. Halts the execution of the program.

Goal: Stop the program execution