(User Story) “As a <role>, I want to <do something>, So that <big picture need or problem is solved>”

As an instructor, I want to provide CS students with a machine language simulation tool, so that they can experiment with low-level programming concepts without needing physical hardware.

As a CS student, I want to run my BasicML program in the UVSim so I can understand how assembly-level instructions interact with CPU and memory.

Use Case 1: File Input by User

Actor: User
System: File handling and loader subsystem
Goal: Load a program file into memory for execution
Steps:

1. Launch program
2. Prompt user to enter file name
3. Receive file name input
4. Validate file existence
5. Read contents of file one word at a time
6. Check each word is 5 characters long
7. Parse and store each valid word into sequential memory addresses
8. Raise an error and re-prompt if file is invalid or word format is incorrect

Use Case 2: Execute BRANCH (Opcode 40)

Actor: Instruction execution unit
System: Program counter and control flow logic
Goal: Jump unconditionally to a specific memory address
Steps:

1. Parse opcode from instruction
2. Identify operand (target address)
3. Set program counter to target address
4. Continue execution from new memory location

Use Case 3: Execute READ (Opcode 10)

Actor: Instruction execution unit
System: Input handling and memory writing logic
Goal: Store user input into specified memory location
Steps:

1. Parse opcode from instruction
2. Identify target memory address from operand
3. Prompt user for input
4. Receive input from console
5. Store input value in identified memory address
6. Increment program counter

Use Case 4: Execute HALT (Opcode 43)

Actor: Instruction execution unit
System: Program state and control logic
Goal: End program execution
Steps:

1. Parse opcode from instruction
2. Set halted flag to True
3. Output message to console showing halt location and accumulator value
4. Stop instruction cycle

Use Case 5: Execute BRANCHNEG (Opcode 41)

Actor: Instruction execution unit
System: Program counter and accumulator logic
Goal: Branch to a new address if accumulator is negative
Steps:

1. Parse opcode from instruction
2. Identify operand (target address)

3. Check if accumulator < 0
4. If true, set program counter to operand
5. Else, increment program counter

Use Case 6: Execute BRANCHZERO (Opcode 42)

Actor: Instruction execution unit

System: Program counter and accumulator logic

Goal: Branch to a new address if accumulator is zero

Steps:

1. Parse opcode from instruction
2. Identify operand (target address)
3. Check if accumulator == 0
4. If true, set program counter to operand
5. Else, increment program counter

Use Case 7: Execute LOAD (Opcode 20)

Actor: Instruction execution unit

System: Memory management and code processor

Goal: Successfully load a value into the accumulator

Steps:

1. Parse function code
2. Identify target memory address from operand
3. Fetch value from identified memory address
4. Copy fetched value into accumulator register
5. Increment program counter

Use Case 8: Execute STORE (Opcode 21)

Actor: Instruction execution unit
System: Memory management subsystem
Goal: Store the value in the accumulator into memory
Steps:

1. Parse function code
2. Identify target memory address from operand
3. Copy value from accumulator to memory at identified address
4. Increment program counter

Use Case 9: Execute ADD (Opcode 30)

Actor: Instruction execution unit
System: Arithmetic logic unit (ALU)
Goal: Add a value from memory to the accumulator
Steps:

1. Parse function code
2. Identify memory address from operand
3. Fetch value from memory
4. Add value to accumulator
5. Store result in accumulator
6. Increment program counter

Use Case 10: Execute SUBTRACT (Opcode 31)

Actor: Instruction execution unit
System: Arithmetic logic unit (ALU)
Goal: Subtract a memory value from the accumulator
Steps:

1. Parse function code
2. Identify memory address from operand
3. Fetch value from memory

4. Subtract value from accumulator
5. Store result in accumulator
6. Increment program counter

Use Case 11: Execute DIVIDE (Opcode 32)

Actor: Instruction execution unit

System: Arithmetic logic unit (ALU)

Goal: Divide the accumulator by a value from memory

Steps:

1. Parse function code
2. Identify memory address from operand
3. Fetch value from memory
4. If value ≠ 0, divide accumulator by value
5. Store result in accumulator
6. If value == 0, raise divide-by-zero error and halt
7. Increment program counter unless halted

Use Case 12: Execute MULTIPLY (Opcode 33)

Actor: Instruction execution unit

System: Arithmetic logic unit (ALU)

Goal: Multiply accumulator by a value from memory

Steps:

1. Parse function code
2. Identify memory address from operand
3. Fetch value from memory
4. Multiply value by accumulator
5. Store result in accumulator
6. Increment program counter