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

User Story 1:
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.

User Story 2:
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.
I also want to be able to load, edit, and save multiple program files with extended memory support (up to 250 lines) and support both legacy (4-digit) and new (6-digit) file formats to manage more complex programs efficiently.

Use Cases

Use Case 1: File Input by User

- Actor: User
- System: File handling and loader subsystem
- Goal: Load a program file into memory for execution or editing
- Steps:
    1. Launch program.
    2. Prompt user to select a file from any directory.
    3. Receive file selection input.
    4. Validate file existence and format (4-digit or 6-digit words).
    5. Confirm file contains no more than 100 lines (old format) or 250 lines (new format).
    6. Read file contents word by word (5 characters for old, 6 for new format).
    7. Parse and store valid words into sequential memory addresses.
    8. If invalid format or too many lines, raise error and re-prompt.
    9. Display file contents in GUI for user inspection and editing.

Use Case 2: Execute BRANCH (Opcode 40 / 040 for new format)

- 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-digit or 3-digit opcode depending on format).
    2. Identify operand (target address, supporting 2- or 3-digit addresses).
    3. Verify target address within valid range (00-99 old, 000-249 new).

4. Set program counter to target address.
5. Continue execution from new memory location.

Use Case 3: Execute READ (Opcode 10 / 010 for new format)

- 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 / 043 for new format)

- 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 / 041 for new format)

- 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 / 042 for new format)

- 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 / 020 for new format)

- 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 / 021 for new format)

- 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 / 030 for new format)

- 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 / 031 for new format)

- 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 / 032 for new format)

- 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 / 033 for new format)

- 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.

Use Case 13: Convert 4-Digit File to 6-Digit File Format

- Actor: User
- System: File conversion utility within the application
- Goal: Convert an existing 4-digit word format file into the new 6-digit word format file for extended memory and functionality
- Steps:
    1. User opens the conversion tool from the application GUI or menu.
    2. Prompt user to select a 4-digit format file from any directory.
    3. Validate the selected file contains only valid 4-digit words and does not exceed 100 lines.

4. Read each line of the file sequentially.
5. For each line:
    - If the first two digits match a valid function opcode, convert using the 0XX0XX format (e.g., 1007 → 010007).
    - Otherwise, treat as numerical value and convert using the 00XXXX format (e.g., 5555 → 005555).
6. Reject the file if any invalid or ambiguous lines are detected.
7. Prompt user to save the new 6-digit file in a user-chosen directory with a new filename.
8. Save the converted file respecting the 250 line max and six-digit word format.
9. Confirm conversion success and optionally open the new file in the editor.