﻿Milestone 5 Design Document

CS-2450

Group D

7/25/25

Project Overview

UVSim is a GUI-based Java program that behaves like a 250‑word, accumulator‑based machine. It loads a text file containing six digit BasicML instructions into main memory, then fetches, decodes, and executes each instruction.

Every subsystem (memory, CPU loop, loader, console) lives in its own class, and all numeric limits are checked with appropriate error messages.

The GUI itself is contained within its own class, with a couple of component classes to organize certain complex objects as parameters.

MemoryRegister Class

A wrapper around an integer “word” that reduces how you can interact with the integer to only getting and setting. You can “get” the opcode, value or entire word depending on the function call.

Memory Class

A thin wrapper around a fixed integer array of 100 words.

Holds int[] cells = new int[100].

Provides read(int addr) and write(int addr, int word) that first verify 0 ≤ addr ≤ 249. Otherwise throw an exception.

UVCpu Class

Holds the processor state and the fetch, decode, execute loop.

Fields: int pc (program counter), int acc (accumulator), boolean halted, and a reference to both a Memory and a Console instance.

void run() keeps looping until halted is true, calling fetch() and decode().

UVSimGUI Class

This is where all the items related to displaying information to the user are held.

The GUI is broken down into JPanels with buttons and text fields to display all needed information. The parsing of input, conversion of files, and any other work that doesn’t need the back-end simulated CPU/Memory is handled by this class.

FileTab Class

This is a specialized object that holds all the needed information for each individual tab. Each tab has its own CPU along with the panels that will be displayed by the GUI.

User Stories

As a CS student, I want to run my BasicML program inside UVSim so that I can debug my homework quickly by watching how each instruction affects the accumulator and memory.

As teacher, I want to get a clear descriptive error message when a student’s program crashes so that I can give equally clear and descriptive feedback on their assignments.

Use Cases

1. Load Program File

Actor: Student at the keyboard

System: ProgramLoader + Memory

Goal: Populate memory locations 000‑NNN with the words from the selected text file.

Prompt user for filename.

Open file and read each line.

Convert each line to an MemoryRegister word; ignore blank lines and comments.

Display to user what lines have been ignored.

Write the word into the next memory cell.

Close the file; return success or error (file not found etc.).

2. Validate Memory Address

Actor: Memory class internals

System: Memory bounds checker

Goal: Prevent illegal access outside 0‑99.

Receive a read(addr) or write(addr,val) request.

If addr < 0 or > 99, throw exception.

Otherwise continue with the read or write.

3. Fetch Instruction

Actor: CPU fetch‑decode‑execute loop

System: UVCpu

Goal: Retrieve the next instruction word for decoding.

Read memory at programCounter.

Split the four digit word into opcode (first 2) and operand (last 2).

Increment programCounter to point to the subsequent cell.

4. WRITE (11)

Actor: CPU

System: Console handler

Goal: Display a word from memory.

Fetch value from operand address.

Print value with sign and leading zeros.

5. LOAD (20)

Actor: CPU

System: Register logic

Goal: Move a word from memory into the accumulator.

Fetch value at operand address.

Copy value into accumulator register.

6. STORE (21)

Actor: CPU

System: Register logic

Goal: Save the accumulator back to memory.

Read current accumulator.

Write it to operand address in memory.

7. ADD (30)

Actor: CPU arithmetic unit

System: ALU emulation

Goal: Update accumulator with addition.

Fetch operand value.

Compute acc = acc + operand.

If |acc| > 999999, flag overflow, cut off highest digits until no longer overflowing.

Otherwise store the result in accumulator.

8. SUBTRACT (31)

Actor: CPU arithmetic unit

System: ALU emulation

Goal: Update accumulator with subtraction.

Fetch operand value.

Compute acc = acc - operand.

If |acc| > 999999, flag overflow, cut off highest digits until no longer overflowing.

Otherwise storethe result in accumulator.

9. DIVIDE (32)

Actor: CPU arithmetic unit

System: ALU emulation

Goal: Divide accumulator by operand.

Fetch operand value.

If operand == 0, print divide by zero error, HALT.

Else compute integer division, store in accumulator.

10. MULTIPLY (33)

Actor: CPU arithmetic unit

System: ALU emulation

Goal: Multiply accumulator by a word from memory.

Fetch operand value from memory.

Compute acc = acc \* operand.

If |acc| > 999999, flag overflow, cut off highest digits until no longer overflowing.

Otherwise, store the result in the accumulator.

11. BRANCH (40)

Actor: CPU control unit

System: Program counter manager

Goal: Unconditionally alter sequential flow.

Set programCounter = operand.

12. BRANCHNEG (41)

Actor: CPU control unit

System: Program counter manager

Goal: Alter sequential flow if accumulator is negative.

If acc < 0, set programCounter = operand; else continue sequentially.

13. BRANCHZERO (42)

Actor: CPU control unit

System: Program counter manager

Goal: Alter sequential flow if accumulator is zero.

If acc == 0, set programCounter = operand; else continue sequentially.

14. HALT (43)

Actor: CPU loop

System: UVCpu

Goal: End program execution cleanly.

Detect opcode 43.

Print "Program halted normally."

Break out of main loop and exit to OS.

15. Change GUI Colors

Actor: Student using program

System: UVSimGUI

Goal: Change background colors of GUI based on user’s preference when requested

Customize button is selected

Another menu displaying primary and secondary color choices is provided.

One of these buttons is selected

A display of all colors available is presented for the users choice.

A new color is chosen and confirmed

Panels flagged and stored as either “Primary” or “Secondary” will be iterated through to modify based on user’s choice.

16. Display User’s Code Visually

Actor: User opening a file

Goal: Display chosen files by the user in a readable format.

Selects “Open Program File”

GUI will open up a secondary window displaying possible selections

The user selects and confirms a file

GUI will send the file to the backend as well as display the contents on the screen.

17. Edit files before running code

Actor: User making edits

System: UVSimGUI

Goal: Change the inputted code while the software is running.

“Program” panel is selected

A cursor appears implying editability

A keystroke is inputted

The GUI displays keystrokes in the text box

18. Save edits to computer

Actor: User making edits

System: UVSimGUI

Goal: Maintain edits made to files between sessions in the software

“Save As” is clicked

A secondary window opens up

A file name is inputted

Select “OK” to confirm

When confirmed, the GUI will store the new file

19. Convert file format to 6-digit words

Actor: User with old file formats

System: UVSimGUI

Goal: Convert older version files into the newer format

4->6 button is clicked

The GUI displays a secondary window with available files for conversion.

The selected file is chosen

The window then prompts for the file to be saved as another name.

For conversion, the software will identify any 4-digit valid words, and convert them to 6-digit.

If there is an invalid line or one that is already 6-digit, conversion is skipped.

The file is saved and the secondary window is closed

20. Clear output from UI

Actor: User who has too many prompts in the output window

System: UVSimGUI

Goal: Clear output window for visibility

The “Clear” button is selected

The text field of the output object is cleared

21. View multiple tabs

Actor: User testing multiple programs

System: UVSimGUI / FileTab

Goal: Allow user to flip between files within the GUI

“Open Program File” is chosen

“Open Program File” is chosen for a second time

Labels identifying the file names are displayed in the “Program” window

The first file is selected by the user

The tab highlights and fills the “Program” window with the file content