

UVSim Final Project Documentation

CS-2450: Computer Organization

Team Members:

- Daniel Urling
- Michael Findlay
- Traedon Harris

Utah Valley University

August 2025

Table of Contents

1. Introduction / Executive Summary
2. User Stories and Use Cases
3. Functional Specifications (Requirements)
4. Class Diagrams, Descriptions, and GUI Wireframes
5. Unit Test Descriptions
6. Application Instructions (User Manual)
7. Future Road Map

1. Introduction / Executive Summary

UVSim is a virtual machine simulator designed for educational use at Utah Valley University. Its purpose is to allow students to load, edit, and execute Basic Machine Language (BasicML) programs while observing how low-level machine operations interact with memory and CPU registers.

This documentation summarizes the full development lifecycle of UVSim. It includes user stories, use cases, functional requirements, architecture diagrams, wireframes, test plans, a user manual, and future development recommendations. The goal of this project is to provide a teaching and learning tool that bridges theoretical computer organization concepts with hands-on practice in a safe and visual environment.

2.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

Actor: UVSIM UI

System:

Change hex value of primary and secondary colors when button is pressed

Based on user input.

Goal: dynamically alter colors

Actor: UI Editor

System:

Truncate any file loaded over 250 lines of instructions

Goal: Keep instruction size down

Actor: UI ConsoleManager

System:

Update the labels in the console whenever there is a message to display

Goal: display necessary information

Actor: UVSIM SolutionExplorer

System:

Change the editor text based on the file that is selected

Goal: Allow multiple files to be worked on at once

3.Functional Requirements

1. The system shall allow loading BasicML programs from .txt files from any directory.
2. The system shall allow saving BasicML programs .txt files into any directory.
3. The system shall have a solution explorer to select txt files that have been created or loaded in.
4. The system shall load text from selected .txt file into instructions editor
5. The system shall allow editing of loaded programs.
6. The system shall validate code from instructions editor by ensuring it matched BasicML syntax.
7. The system shall provide a Run button to execute BasicML code from editor
8. The system shall display system output such as errors or confirming operations
9. They system shall display selected text file in instructions editor
10. The system shall visually indicate execution status in console (running/waiting for input).
11. The system shall execute all BasicML operations:
 - a. READ (10) for input from the user to specified location
 - b. WRITE (11) for output to the user from specified location
 - c. LOAD (20) for loading specified location value into the accumulator
 - d. STORE (21) for storing accumulator value to specified location
 - e. ADD (30) for adding specified location to accumulator
 - f. SUBTRACT (31) for subtracting specified location from accumulator

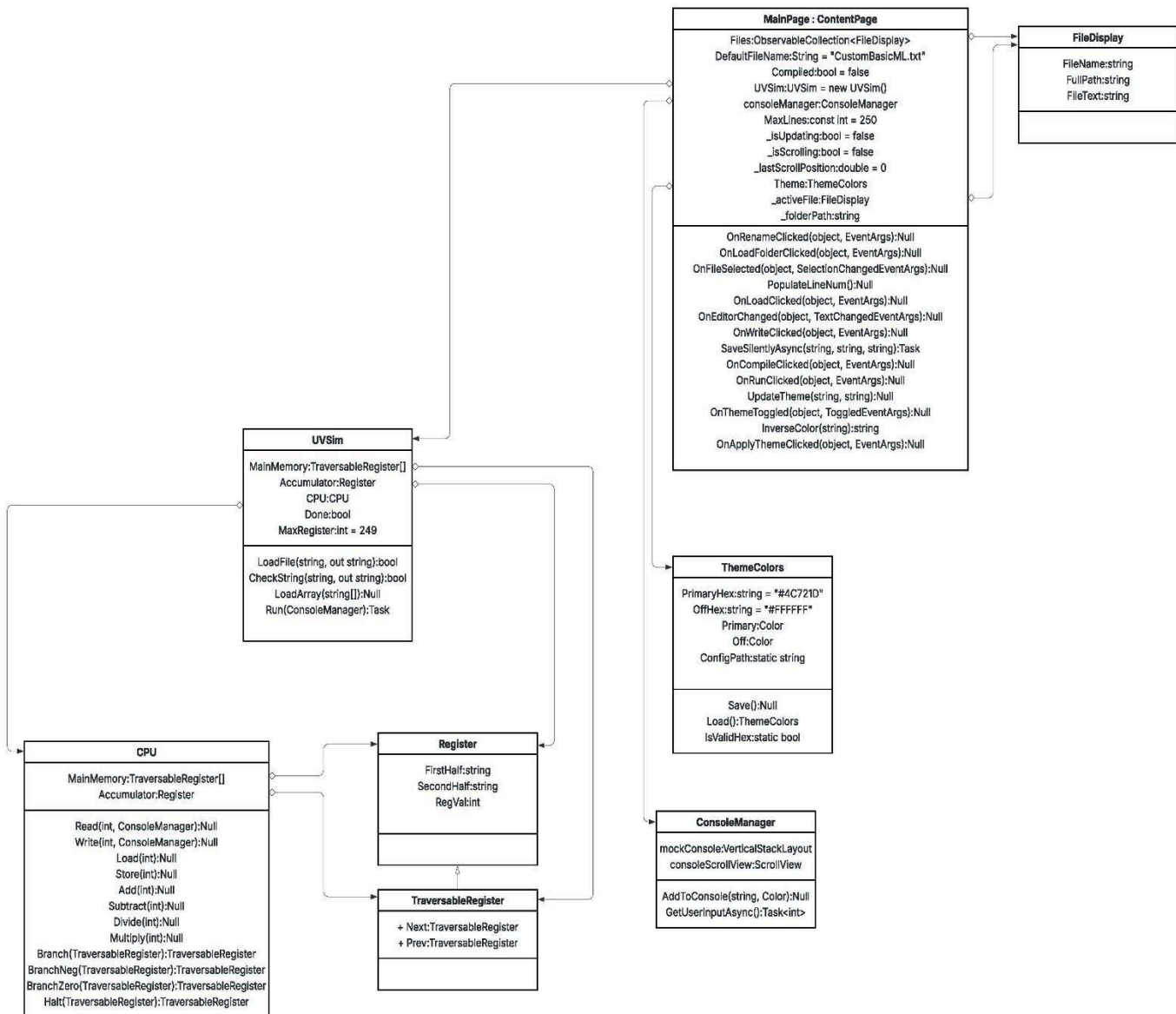
- g. MULTIPLY (32) for multiplying specified location by accumulator
 - h. DIVIDE (33) for dividing accumulator by specified location
 - i. BRANCH (40) for branching to specified location while running
 - j. BRANCHNEG (41) for branching to specified location while running if accumulator is negative
 - k. BRANCHZERO (42) for branching to specified location while running if accumulator is 0
 - l. HALT (43) for stopping the run of the program
12. The system shall handle user input via console for READ operations.
 13. The system shall display output from WRITE operations in the console.
 14. The system shall maintain and display accumulator state during execution.
 15. The system shall highlight the currently executing instruction.
 16. The system shall report all errors without crashing.
 17. The system shall display memory contents in a table format that updates during execution.
 18. The system shall handle 6 digit instructions
 19. The system shall handle up to 250 lines of memory
 20. The system shall convert any 4 digit format file into 6 digit format
 21. The system shall truncate anything above 250 lines of text in the instructions editor
 22. The system shall allow user to enter primary and off colors in hex
 23. The system shall allow user to invert colors with toggle switch.

Non-Functional Requirements

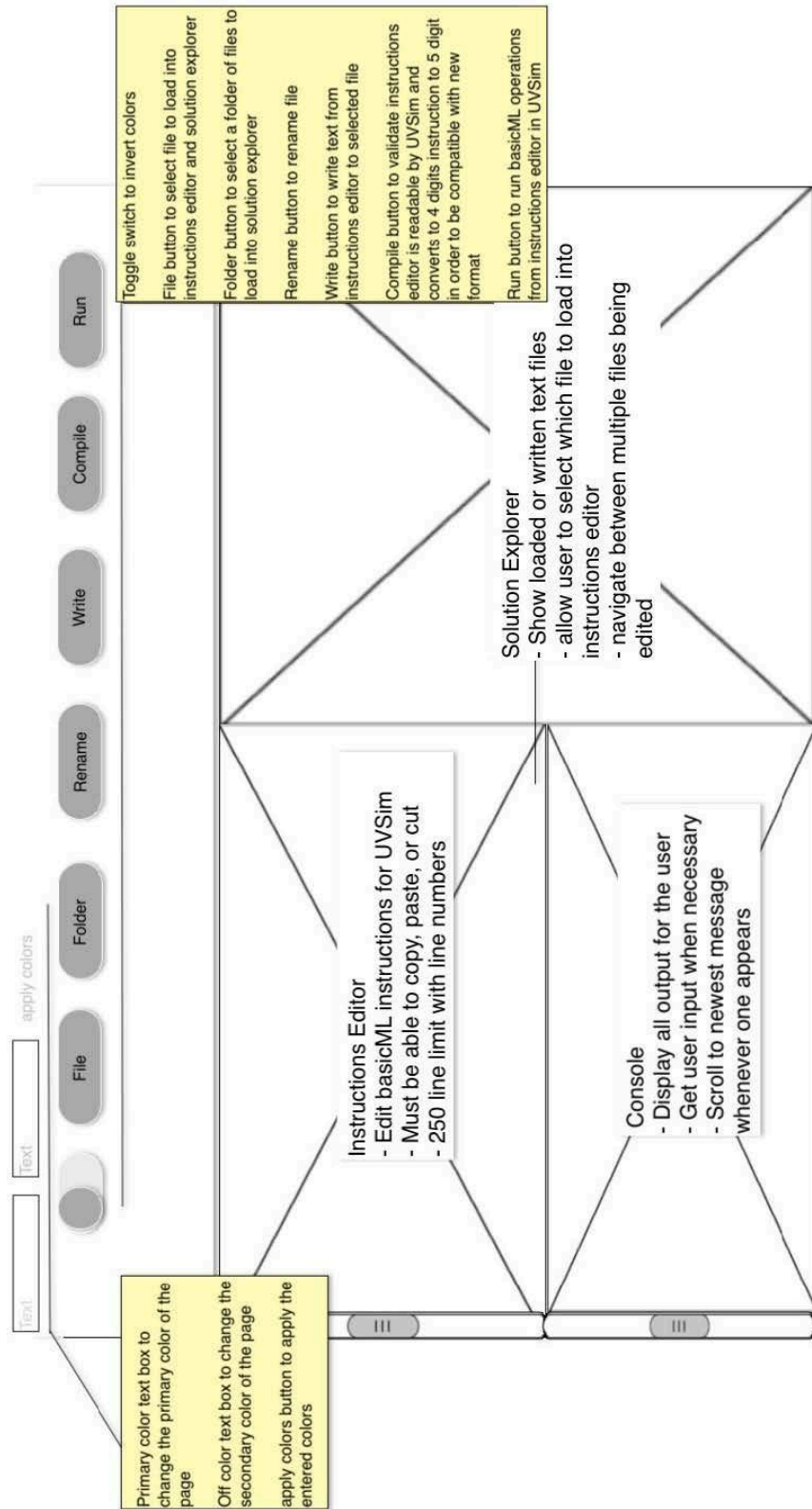
1. The system shall execute programs in less than 10 seconds.
2. The system shall have an intuitive user interface
3. The system shall be compatible with Windows (.NET 8.0+).
4. The system shall respond to user input within 1 second.

4. Class Diagrams, Descriptions, and GUI Wireframes

UML Class Diagram:



GUI wire diagram:



5. Unit Test Descriptions

Test Name	Description	Use Case	Inputs	Expected Output	Success Criteria	
AddTest_Success	Tests addition of two values from memory.	Perform addition operation	1050 (10), 1051 (3), 2050, 3051	Accumulator = 13	Accumulator value equals 13	
AddOverflowTest_Success	Tests addition with large values to check overflow behavior.	Perform addition operation	1050 (9999), 1051 (9999), 2050, 3051	Accumulator = 1999	Accumulator value equals 1999	
SubtractTest_Success	Tests subtraction of two values from memory.	Perform subtraction operation	1050 (10), 1051 (3), 2050, 3151	Accumulator = 7	Accumulator value equals 7	
SubtractOverflowTest_Success	Tests subtraction with large magnitude values.	Perform subtraction operation	1050 (9999), 1051 (-9999), 2050, 3151	Accumulator = 1999	Accumulator value equals 1999	
MultiplyTest_Success	Tests multiplication of two values from memory.	Perform multiplication operation	1050 (10), 1051 (3), 2050, 3351	Accumulator = 30	Accumulator value equals 30	

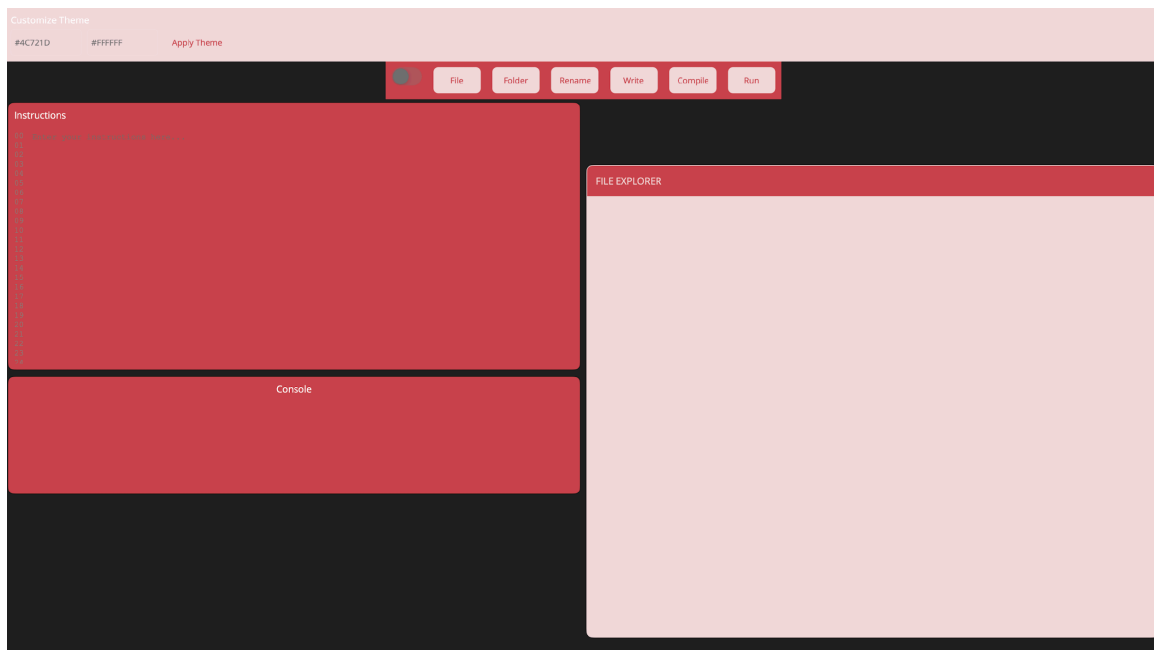
MultiplyOverFlowTest_Success	Tests multiplication with large values for overflow behavior.	Perform multiplication operation	1050 (9999), 1051 (9999), 2050, 3351	Accumulator = 9998	Accumulator value equals 9998	
DivideTest_Success	Tests division of accumulator value by memory value.	Perform division operation	1050 (10), 1051 (3), 2050, 3251	Accumulator = 3	Accumulator value equals 3	
ReadTest_Success	Tests reading input into memory at given location.	Read input into memory (BasicML operation 10)	User input = 7777, instruction = 1008	Memory[8] = 7777	Memory value equals 7777	
WriteTest_Success	Tests output from memory to console.	Write output from memory to screen	Memory[4] = 7777, instruction = 1104	Console output contains 7777	Console output contains 7777	
LoadTest_Success	Tests loading value from memory into the accumulator.	Load value into accumulator	Memory[4] = 7777, instruction = 2004	Accumulator = 7777	Accumulator value matches memory[4]	
StoreTest_Success	Tests storing accumulator value into memory.	Store accumulator value into memory	Accumulator = 7777, instruction = 2104	Memory[4] = 7777	Memory[4] value matches accumulator	

Branch_Success	Tests unconditional branching.	Branch	4002, 4300, 2004, 4300, Memory[4] = 5	Accumulator = 5	Accumulator value equals 5 after branch	
BranchNeg_Success	Tests conditional branch on negative accumulator.	Branch on negative	Accumulator = -1000, 4103, Memory[5] = -1000	Accumulator = -1000	Branch occurs only if accumulator is negative	
BranchZero_Success	Tests conditional branch on zero accumulator.	Branch on zero	Accumulator = 0, 4203, Memory[5] = 0	Accumulator = 0	Branch occurs only if accumulator is zero	
Halt_Success	Tests program halting behavior.	Stop the program execution	4300, Memory[3] = 2003	Accumulator = 2003	Program halts and accumulator is unchanged	
ForLoopTest_Success	Tests a simulated for-loop using read, write, and branch operations.	Perform combination of execution tasks	1050, 1150, 2050, 3108, 2150, 4207, 4001, 4300	Looped values output: 4, 3, 2, 1	Output contains all decremented values	
Coordinated OperationsOrderedTest_Success	Tests combined read, write, load, and store in order.	Combined read, write, load, store	1005, 1105, 2005, 2106, 4300	Memory[5] and [6] = 7777, console output 7777	All steps executed correctly and output verified	

Coordinated Operations Reversed Order Test_Success	Tests combined store, load, write, and read in reverse.	Combined store, load, write, read	2189, 2088, 1189, 1000, 4300	Memory[0] cleared, console output contains 7777	Each instruction executes in correct order with expected effects
--	---	-----------------------------------	------------------------------	---	--

6. Application Instructions (User Manual)

Front Page:



UVSim is a virtual machine simulator built for educational use. It executes programs written in BasicML (Basic Machine Language) and visually simulates memory, instructions, and CPU execution via a clean, modern GUI built using .NET MAUI.

This version includes:

- Graphical editor for BasicML programs
- Real-time console for simulated I/O

- Visual file explorer for .txt programs
- File loading, editing, renaming, and saving
- Customizable theme support with persistent config
- Support for both 6 digit BasicML instruction and conversion from 4 digit to 6 digit instruction

IMPORTANT:

This application is optimized and tested for ****Windows only****.

While MAUI supports multiple platforms, the current implementation and features have been tailored specifically for Windows performance, styling, and file handling.

How to Run the Program:

Requirements:

- .NET 8.0 SDK (<https://dotnet.microsoft.com/download>)
- Visual Studio 2022 or newer with the .NET MAUI workload installed

Supported Platform:

- Windows 10 or later

Steps to Run:

1. Clone the repository:

```
git clone https://github.com/YOUR-REPO/UVSim.git
```

2. Open the solution (UVSim.sln) in Visual Studio.
3. Select "Windows Machine" as the target platform.
4. Build and run the project.

GUI Workflow and Controls:

Instructions Editor:

```
Instructions
00 +1115
01 +4003
02 +4300
03 +1116
04 +4206
05 +4300
06 +1117
07 +2015
08 +4205
09 +1118
10 +2019
11 +4113
12 +4300
13 +1120
14 +4300
15 +1111
16 +2222
17 +3333
18 +4444
19 -4444
20 +5555
21
```

- Enter or paste up to 250 BasicML instructions.
- Editor automatically shows line numbers for clarity.
- Instructions are validated and compiled into memory.

File Explorer:



- Located on the right side of the GUI.
- Lists all `.txt` files in a selected folder.
- Click a file to load its contents into the editor.
- Any changes made in a file editor will be temporarily saved while switching between files.

If you need the changes to be saved permanently, just be sure to click 'write'.

- Files can also be renamed (feature available if wired).

Buttons and Their Functions:



Apply Theme: Sets new theme colors based on two HEX input fields.

Toggle: Toggles the theme primary and secondary colors by inverting their hex.

File: Opens a file picker to load a `.txt` file into the editor.

Folder: Opens a file picker to load a folder with '.txt' files autoloading into the editor.

Rename: Rename the file that you have selected.

Write: Saves all currently loaded and edited files back to disk.

Compile: Parses and validates instructions, loading them into memory.

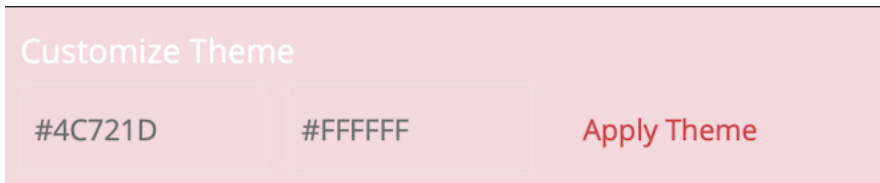
Run: Executes instructions line by line. Output appears in the console.

User Input:

When a READ instruction is executed:

- A text field appears in the console.
- Type an integer and press ENTER to submit.
- Invalid input will show an error message below the input field.

Theme Customization:



Customize Theme

#4C721D #FFFFFF Apply Theme

You can customize the appearance of the simulator by setting two colors:

Primary Color: Affects header bars, console background, and major highlights.

Off Color: Affects button backgrounds and supporting areas.

To apply a theme:

1. Enter valid HEX codes into the theme fields (e.g., #4C721D, #FFFFFF).
2. Click the "Apply Theme" button.
3. Optionally use the toggle to invert the theme.

Themes are saved to disk and persist between sessions.

RGB Hex values to use:

Primary	Secondary
Blue - #007BFF	Light Gray - #F8F9FA
Red - #DC3545	Light Red - #F8D7DA
Green - #28A745	Light Green - #D3F9D8
Yellow - #FFC107	Light Yellow - #FFF3CD
Orange - #FD7E14	Light Teal - #C3FAE8
Purple - #6F42C1	Light Purple - #E0BBE4
Teal - #20C997	Gray - #6C757D

4 digit file support

We do support 4 digit files. When you compile your program, the editor will update the 4 digit instructions to 0XXXX format. This way we can still expect the same 3 digit instructions as when we have six digits without changing the numeric value.

This means 1002 will be changed to 01002 so that '010' can be seen as a command, but still allow 1002 to be the numeric value in any mathematical operations.

Project Structure:

UVSimClassLib/

- CPU.cs: Handles execution of each instruction.
- Register.cs: Stores and formats values using a four-digit word model.
- TraversableRegister.cs: Enables linked navigation of memory.

- UVSim.cs: Runs the fetch-decode-execute loop and manages memory setup.

UVSimGUI/

- MainPage.xaml: Defines the visual layout of the GUI.

- MainPage.xaml.cs: Handles user interaction, file operations, execution control.

- ThemeColors.cs: Loads and saves persistent theme settings.

- App.xaml, AppShell.xaml: Set up the MAUI application shell and routes.

Example Instruction File (example.txt):

1007 ; READ a value into memory address 07

2007 ; LOAD value from address 07 into accumulator

3008 ; ADD value at address 08

2109 ; STORE result into address 09

1109 ; PRINT value at address 09

4300 ; HALT program

7. Future Road Map

Future Roadmap

The UVSim project has successfully achieved its initial milestones of creating a functional machine language simulator with a graphical interface. As the project continues to mature, several enhancements can be introduced to extend its educational value, usability, and technical capabilities.

Expanded Instruction Set

Add support for additional machine-level instructions such as modulus, shift, and bitwise operations.

Implement floating-point arithmetic for more complex computational programs.

Advanced Debugging Tools

Step-by-step execution mode with the ability to set breakpoints.

Memory and register inspection during execution.

Call stack visualization for branching instructions.

Improved User Interface

Dark/light mode theme options in addition to current customizable color schemes.

Drag-and-drop file loading for BasicML programs.

Enhanced memory table visualization with filtering and searching capabilities.

Cross-Platform Support

Extend the application beyond Windows to include macOS and Linux support.

Provide limited functionality builds for mobile (Android/iOS) to support learning on tablets.

Classroom Integration

Multi-user mode for professors to monitor student activity in real time.

Assignment mode: load curated BasicML problems and track student progress.

Cloud-based save/load functionality for collaborative work.

Extended File Handling

Support for exporting memory dumps in CSV or JSON format.

Automated program conversion between different instruction formats.

Version control integration for student projects.

Performance and Optimization

Faster execution engine to handle larger instruction sets (>250 lines).

Parallel instruction simulation for future machine learning experiments.

Documentation and Learning Resources

In-app tutorials to teach BasicML step-by-step.

Interactive exercises with guided feedback for new users.

Video demonstrations and sample problem sets integrated directly in the GUI.