UVSim

Machine Language Simulator

BasicML Documentation

Utah Valley University

CS 2450: Software Engineering

Product Version 1.0

Release Date

April 23, 2025

Development Team Members:

Chase Griffin, Sean Davis, Zach Ward, Elliot Damarjian

Table of Contents

Introduction

Welcome to the UVSim User Manual, a complete guide to understanding, installing, and operating the UVSim software. The UVSim is a simulation tool designed to create a simple interface to run and test Basic Machine Language (BasicML). It offers an easy way for students and educators to experiment with computer architecture and fundamental programming operations.

The UVSim has functions to write, load, and execute BasicML programs through a user-friendly graphical interface. There are displays for all necessary components of a basic CPU such as memory, accumulator, and program counter. This manual will explain all steps of the simulator from installation to running your own programs.

UVSim User Manual:

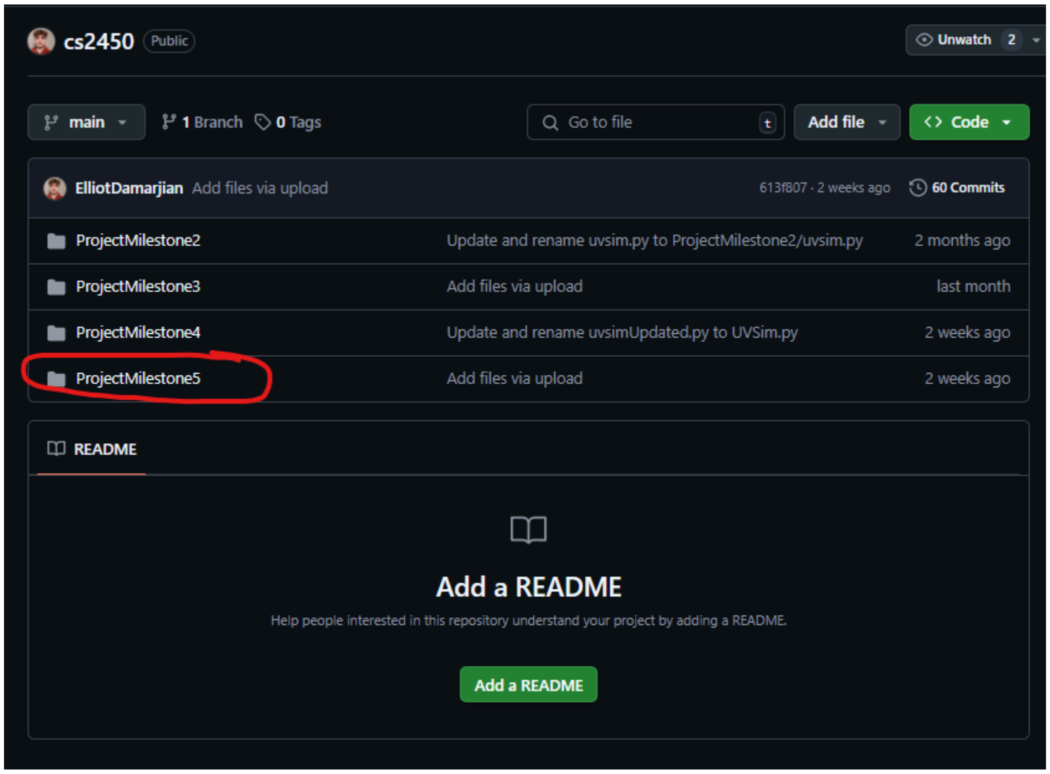
UVSim is a Basic Machine Language (BasicML) simulator. It simulates a computer CPU with memory, accumulator, and program counter. Allowing users to write and execute BasicML programs.

System Requirements Section

- **Operating System:** Windows 10/11, macOS, or Linux
- **Python Version:** 3.0 or later
- **RAM:** Minimum 2GB (4GB recommended)

Enhanced Installation Instructions

1. **Download the Software**
    - Download UVSim from [https://github.com/ElliotDamarjian/cs2450/ProjectMilestone5] *if necessary



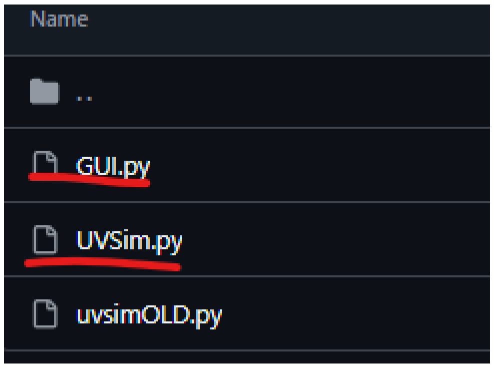
ElliotDamarjian Add files via upload 613f807 · 2 weeks ago 60 Commits

ProjectMilestone2	Update and rename uvsim.py to ProjectMilestone2/uvsim.py	2 months ago
ProjectMilestone3	Add files via upload	last month
ProjectMilestone4	Update and rename uvsimUpdated.py to UVSim.py	2 weeks ago
ProjectMilestone5	Add files via upload	2 weeks ago

Add a README

Help people interested in this repository understand your project by adding a README.

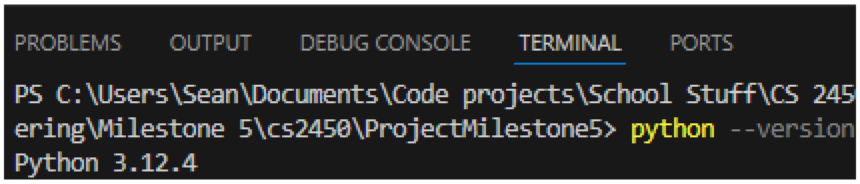
- Download the sources files to your preferred location on your computer [GUI.py, UVSim.py]



```
Name
..
GUI.py
UVSim.py
uvsimOLD.py
```

2. Python Installation

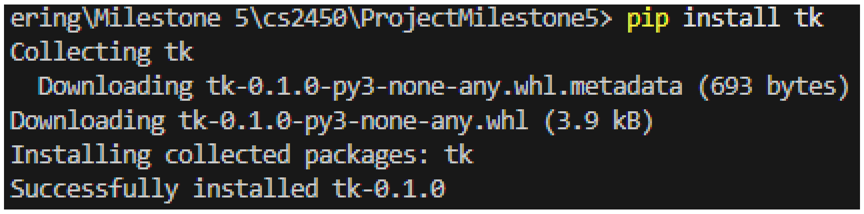
- If Python is not installed, download from python.org
- During installation, check "Add Python to PATH"
- Verify installation by opening a terminal/command prompt and typing: python --version



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
PS C:\Users\Sean\Documents\Code projects\School Stuff\CS 245
ering\Milestone 5\cs2450\ProjectMilestone5> python --version
Python 3.12.4
```

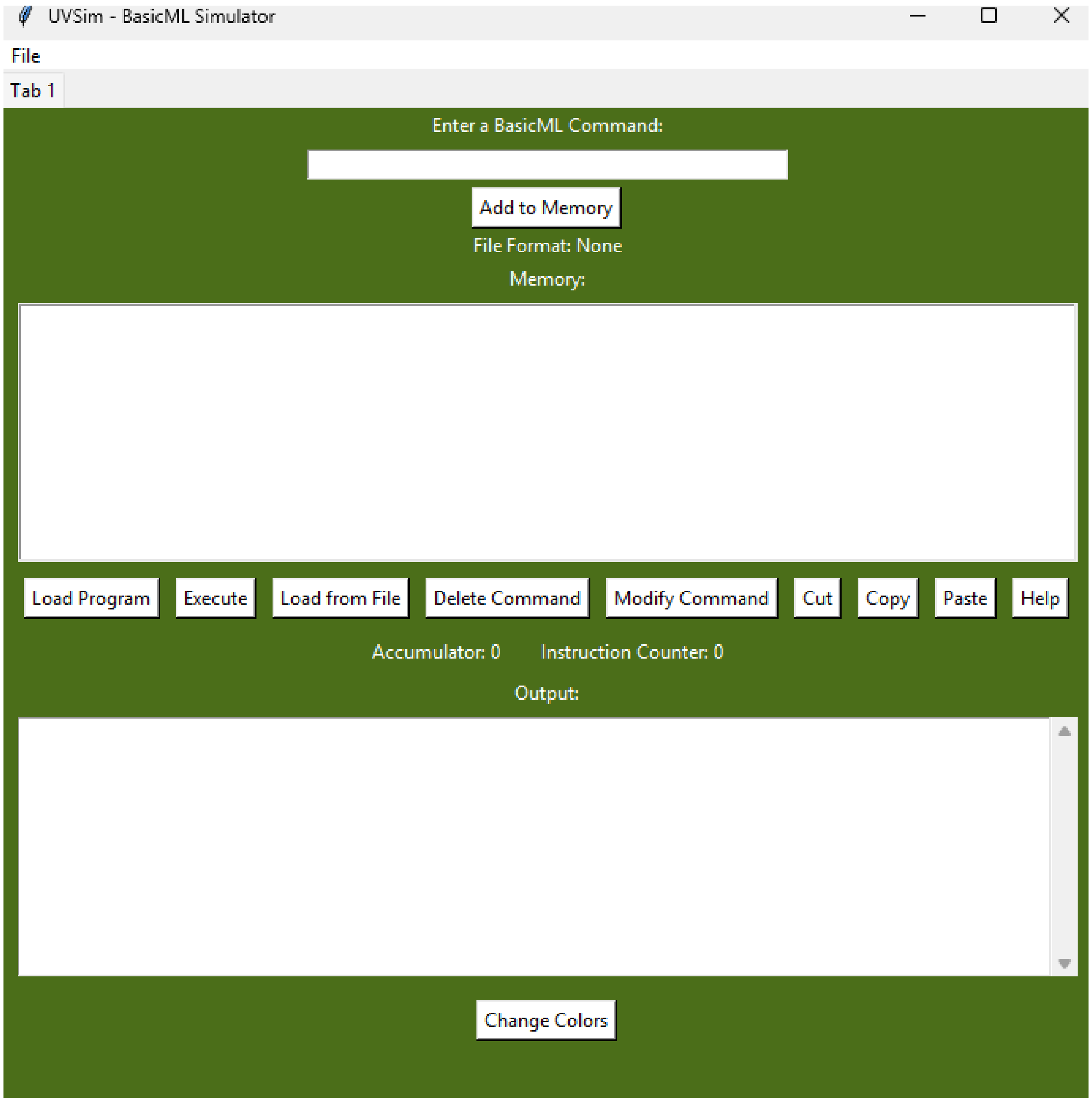
3. Dependencies Installation

- Open a command prompt
- Navigate to the UVSim folder
- Run: pip install tk to install pythons GUI software



```
ering\Milestone 5\cs2450\ProjectMilestone5> pip install tk
Collecting tk
  Downloading tk-0.1.0-py3-none-any.whl.metadata (693 bytes)
Downloading tk-0.1.0-py3-none-any.whl (3.9 kB)
Installing collected packages: tk
Successfully installed tk-0.1.0
```

Graphical User Interface Overview



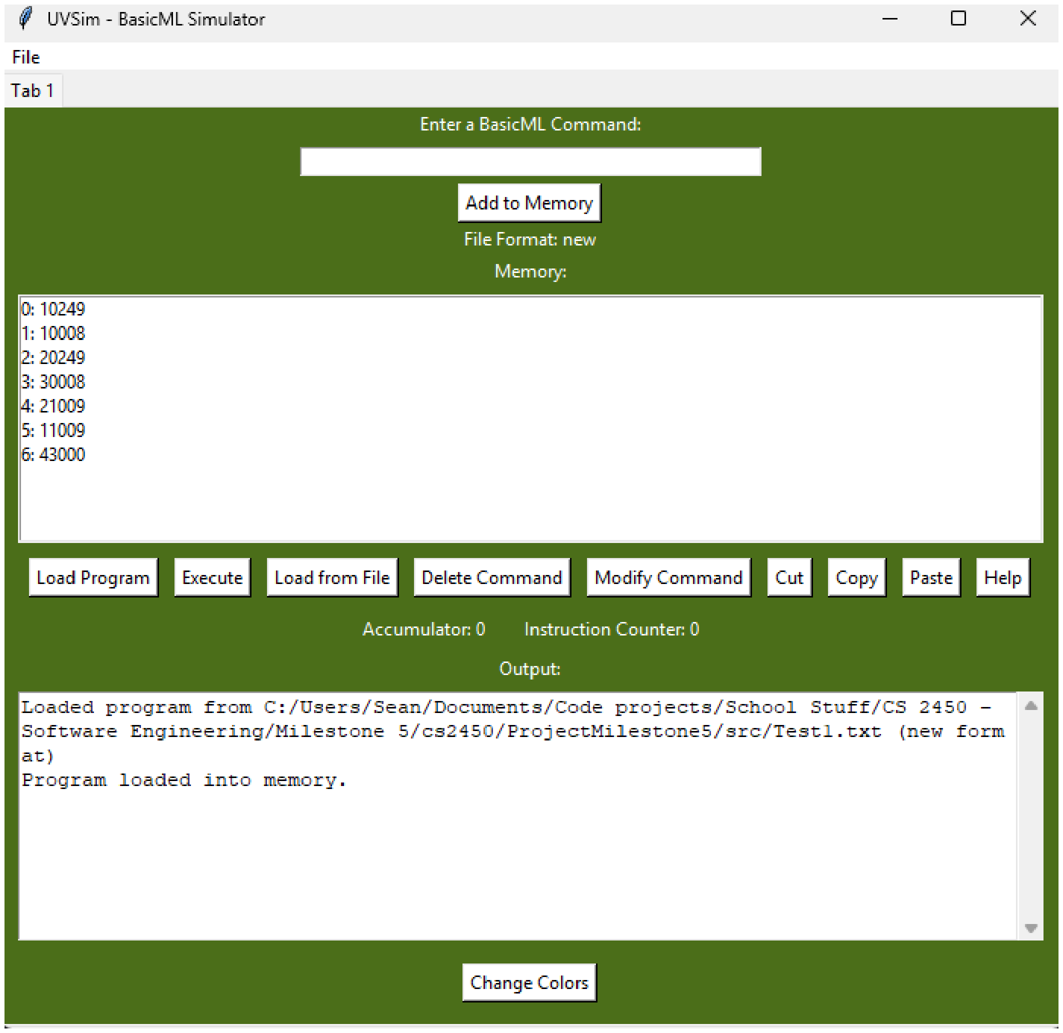
Each Component:

1. **Memory Display:** Shows the current state of the simulated memory
2. **Accumulator Counter:** Shows the current value in the accumulator
3. **Instruction Counter:** Displays the current instruction being executed

4. **Instruction Input Field:** Area to manually enter BasicML instructions
5. **Control Buttons:** Execute, Reset, Load Program, Load from File, Modify, Delete, Cut, Copy, Paste
6. **Output Display:** Shows program output and error messages
7. **Help Button:** Displays BasicML Commands
8. **Tab button:** Allows for multiple tabs to be opened - located top left
9. **Change Colors:** Select two different colors to change the GUI

Program Execution Flow

1. **Loading a Program:**
    - Instructions are loaded into memory
    - Program counter is set to 0
    - Accumulator is reset to 0
2. **Execution Cycle:**
    - Fetch: Instruction at program counter is retrieved
    - Decode: Instruction is parsed into operation and operand
    - Execute: Operation is performed
    - Increment: Program counter advances to next instruction
    - Repeat until HALT instruction or error



File Format: new

Memory:

```
0: 10249
1: 10008
2: 20249
3: 30008
4: 21009
5: 11009
6: 43000
```

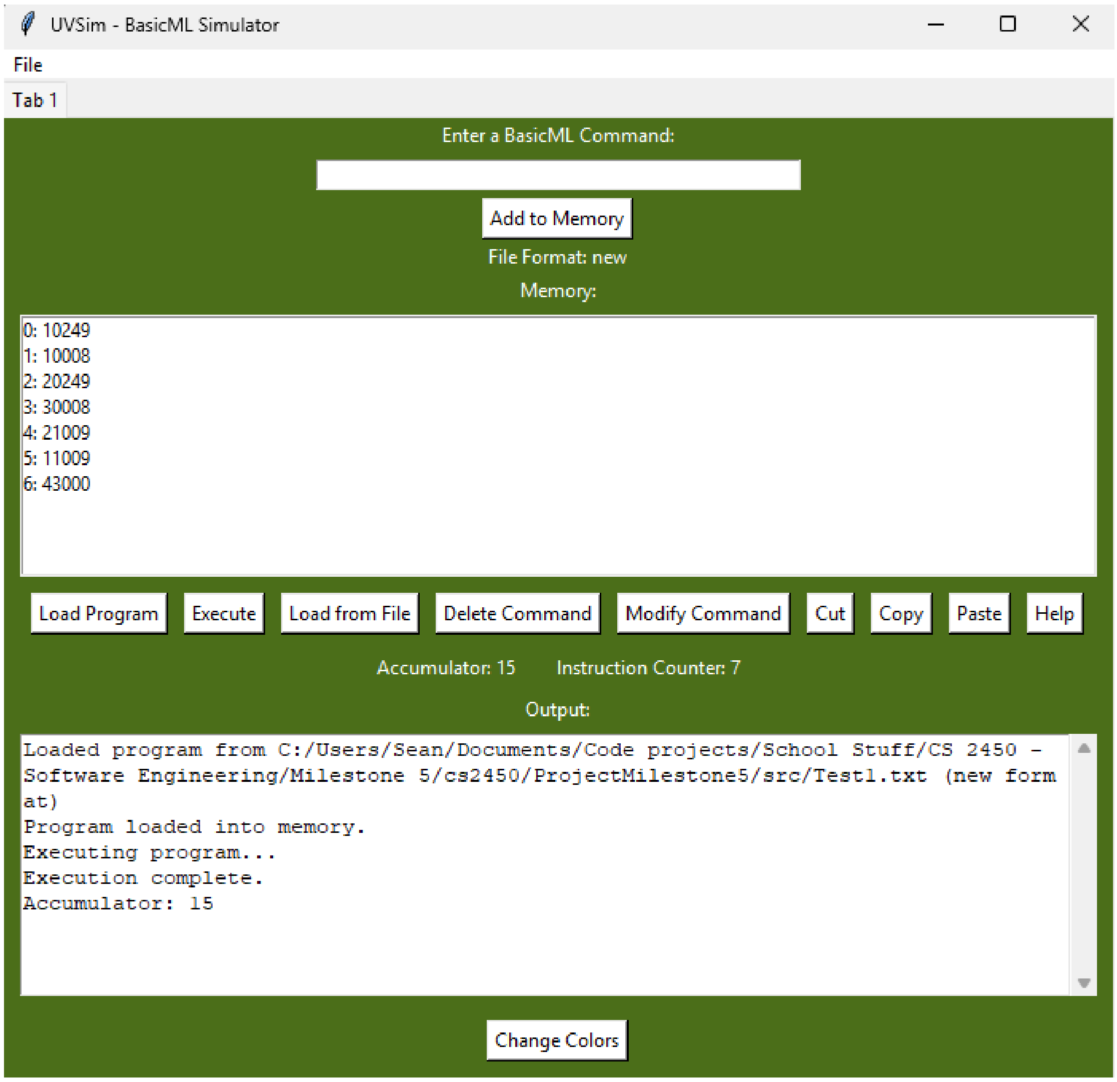
Accumulator: 0   Instruction Counter: 0

```
Loaded program from C:/Users/Sean/Documents/Code projects/School Stuff/CS 2450 -
Software Engineering/Milestone 5/cs2450/ProjectMilestone5/src/Test1.txt (new form
at)
Program loaded into memory.
```

- Step Execution: Execute one instruction at a time
- Memory Inspection: View and modify memory contents during execution
- Breakpoints: Set points where execution will pause
- Register Monitoring: Watch accumulator and program counter values in real-time

Example 1: Addition

- 010249 // READ value into memory location 249
- 010008 // READ another value into memory location 8
- 020249 // LOAD the first value from memory location 249 into accumulator
- 030008 // ADD the second value from memory location 8 to accumulator
- 021009 // STORE result in memory location 9
- 011009 // WRITE the result from memory location 9
- 043000 // HALT
- 000000 // Unused memory location
- 000000 // Unused memory location
- 000000 // Unused memory location
- -99999 // End of program marker



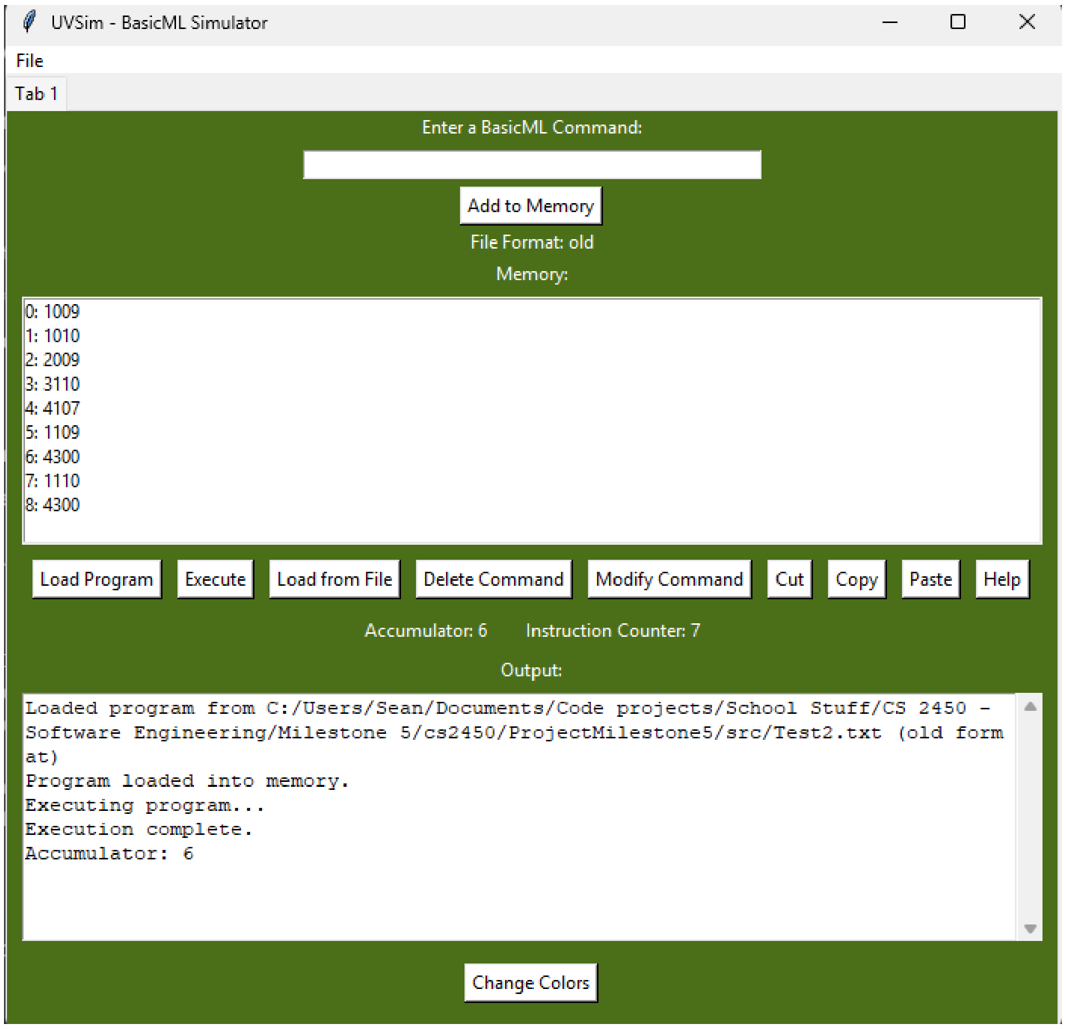
```
0: 10249
1: 10008
2: 20249
3: 30008
4: 21009
5: 11009
6: 43000
```

```
Loaded program from C:/Users/Sean/Documents/Code projects/School Stuff/CS 2450 -
Software Engineering/Milestone 5/cs2450/ProjectMilestone5/src/Test1.txt (new form
at)
Program loaded into memory.
Executing program...
Execution complete.
Accumulator: 15
```

10 + 5 = 15

Example 2: Subtraction

- 1009 // READ value into memory location 09
- 1010 // READ another value into memory location 10
- 2009 // LOAD the first value from memory location 09 into accumulator
- 3110 // SUBTRACT the second value (memory location 10) from accumulator
- 4107 // BRANCHNEG to memory location 07 if result is negative
- 1109 // WRITE the result from memory location 09 (if result is positive/zero)
- 4300 // HALT (if result is positive/zero)
- 1110 // WRITE the second value from memory location 10 (if result is negative)
- 4300 // HALT (if result is negative)
- 0000 // Unused memory location
- 0000 // Unused memory location
- -99999 // End of program marker



```
0: 1009
1: 1010
2: 2009
3: 3110
4: 4107
5: 1109
6: 4300
7: 1110
8: 4300
```

```
Loaded program from C:/Users/Sean/Documents/Code projects/School Stuff/CS 2450 -
Software Engineering/Milestone 5/cs2450/ProjectMilestone5/src/Test2.txt (old form
at)
Program loaded into memory.
Executing program...
Execution complete.
Accumulator: 6
```

18 - 12 = 6

Error Messages and Troubleshooting

Error Message	Possible Cause	Solution
"Invalid opcode"	Instruction doesn't start with a valid operation code	Check instruction format and refer to opcode table
"Memory out of bounds"	Trying to access memory location beyond 99	Ensure all memory references are between 00-99
"Division by zero"	Attempting to divide by zero	Check your logic to prevent division by zero
"File not found"	Specified program file doesn't exist	Verify file path and name

User Stories / Use Cases

Overview: The UVSim is a program designed to simulate a virtual machine. The main purpose is to educate computer science students on machine language and computer architecture. The simulator will interpret lines of machine language in BasicML. Instructions are read from a file into the program.

User Stories:

As a computer science student I want to learn computer architecture and get an understanding of machine language.
As a computer science teacher I want to educate my students on machine language.

Use Cases:

1. Student enters BasicML codes into a notepad
   then they execute the program
   instructions given are done.

2. A user wants to load data from a point in memory
   they enter the LOAD code 20 with the location in memory
   program is executed and the data is loaded into the accumulator

3. John needs to add a number with one stored in memory location 21
   he adds his new number to memory with read 1043
   after loading that number into the accumulator with 2043
   the numbers are added with code 3021

4. A teacher wants to demonstrate writing words from memory to screen
   first they read a word from the keyboard to memory with 1002
   then they write it to screen with 1102

5. Emily needs to divide from a number stored at 08
   she enters the number to divide with 1009
   then loads it into accumulator with 2009
   using code 3208 she divides them

6. A student wants to read keyboard input to memory
   they give the sim code 1001 and execute
   the input is stored in memory

7. A tester wants to ensure the code stops properly
   they enter code 4300 at the end of their program
   it halts and exits.

8. A programmer wants to branch to a different section in memory
   they input the code 4026 into the UVSim
   the program executes and goes to memory location 26

9. David wants to store a word into memory
   they input the code 1001 to store it into 01 in memory
   after executing the word is stored

10. Sarah wants to subtract two numbers with the program
    he first she reads into memory slot 01 with 1001
    the second she puts into 02 with 1002
    then she loads up the second number with 2002
    finally she uses 3101 to subtract the first number from the second

Functional Specifications:
Software Requirements Specification

Functional Requirements

1. The program shall execute BasicML instructions as defined in the specification.
2. The program shall provide a GUI that allows users to navigate available functions.
3. The program shall validate user input, ensuring only integer values are accepted where required.
4. The program shall store up to 100 words in memory at a time.
5. The program shall include an accumulator to store intermediate results of operations.
6. The program shall allow users to input data via keyboard or file upload.
7. The simulator shall detect and display the following errors:
    - Invalid Instructions (Unrecognized opcodes)
    - Memory Overflow (Exceeding 100 memory slots)
    - Divide-by-Zero Errors (Prevent division by zero operations)
8. The program shall display execution results through the GUI output window.
9. The simulator shall allow users to search for specific values in memory.
10. The program's GUI shall display the contents of memory in real-time.
11. The GUI shall include tooltips or a Help menu explaining available commands and functionalities.
12. The program shall support execution of the following arithmetic operations:
    - ADD (Addition)
    - SUBTRACT (Subtraction)
    - MULTIPLY (Multiplication)

- DIVIDE (Division)

13. The simulator shall maintain a log of executed commands that users can view.

14. The program shall halt execution when encountering the HALT (4300) instruction.

15. The program shall update and display the accumulator and instruction counter in real-time.

Non-Functional Requirements

1. The program's GUI layout shall follow standard usability principles, ensuring clear navigation with labeled buttons and tooltips.

2. The simulator shall handle invalid input by:
    - Preventing crashes through error handling.
    - Displaying clear, descriptive error messages.

3. Each instruction shall be executed in under 100 ms to ensure smooth user interaction.

Class Diagrams and GUI wireframes

Class: UVSim (GUI)

Purpose: This class creates a BasicML graphical interface for users to enter opcodes and execute them. There are buttons for load program, execute, load from file, and help.

Attributes:

- memory: A 250 word memory list.
- accumulator: Integer for current accumulator value.
- accumulator_label: A Tkinter Label widget to show the current value of the accumulator.
- memory_label: A Tkinter Label widget for memory display.
- memory_listbox: A Tkinter Listbox widget to display memory contents.
- instruction_counter: Integer for current instruction pointer.
- instruction_label: A Tkinter Label to display instruction input prompts.
- instruction_input: A Tkinter ScrolledText widget for user instruction entry.
- load_button: A Tkinter Button widget to load instructions into memory.
- execute_button: A Tkinter Button widget to execute loaded instructions.
- file_button: A Tkinter Button widget to load instructions from a file.
- status_frame: A Tkinter Frame widget for accumulator and instruction counter display.
- instruction_counter_label: A Tkinter Label widget to show the current instruction counter.
- output_label: A Tkinter Label widget for output display.
- output_text: A Tkinter ScrolledText widget for output messages.
- help_button: A Tkinter Button widget that opens a help dialog.

Methods: __init__(self, root)
Purpose: Initializes the UVSim GUI.

Class: UVSim (Processor)

Purpose: This class checks for correct output from UVSim through unit tests.

Attributes:

- memory: A 250 word memory list.
- accumulator: Holds integers for operations.
- program_counter: Keeps track of the current instruction being executed.
- running: Boolean to control program execution.

Methods:

__init__()

Purpose: Initializes memory, accumulator, program counter, and execution state.

Parameters: None.

Post-conditions: Initializes memory, sets accumulator and program_counter to zero, and sets running to True.

load_program_from_file(filename)

Purpose: Reads a program file and loads it into memory.

Parameters: filename (str): Name of the file containing BasicML instructions.

Return Value: True if the program loads successfully. False if an error occurs.

Pre-conditions: File must exist and contain valid instructions.

Post-conditions: Memory is filled with instructions, up to a maximum of 250.

get_input()

Purpose: Handles user input, allowing for easy overriding in tests.

Return Value: Integer input from the user.

Pre-conditions: User must provide a valid integer input.

Post-conditions: Returns a valid integer.

execute()

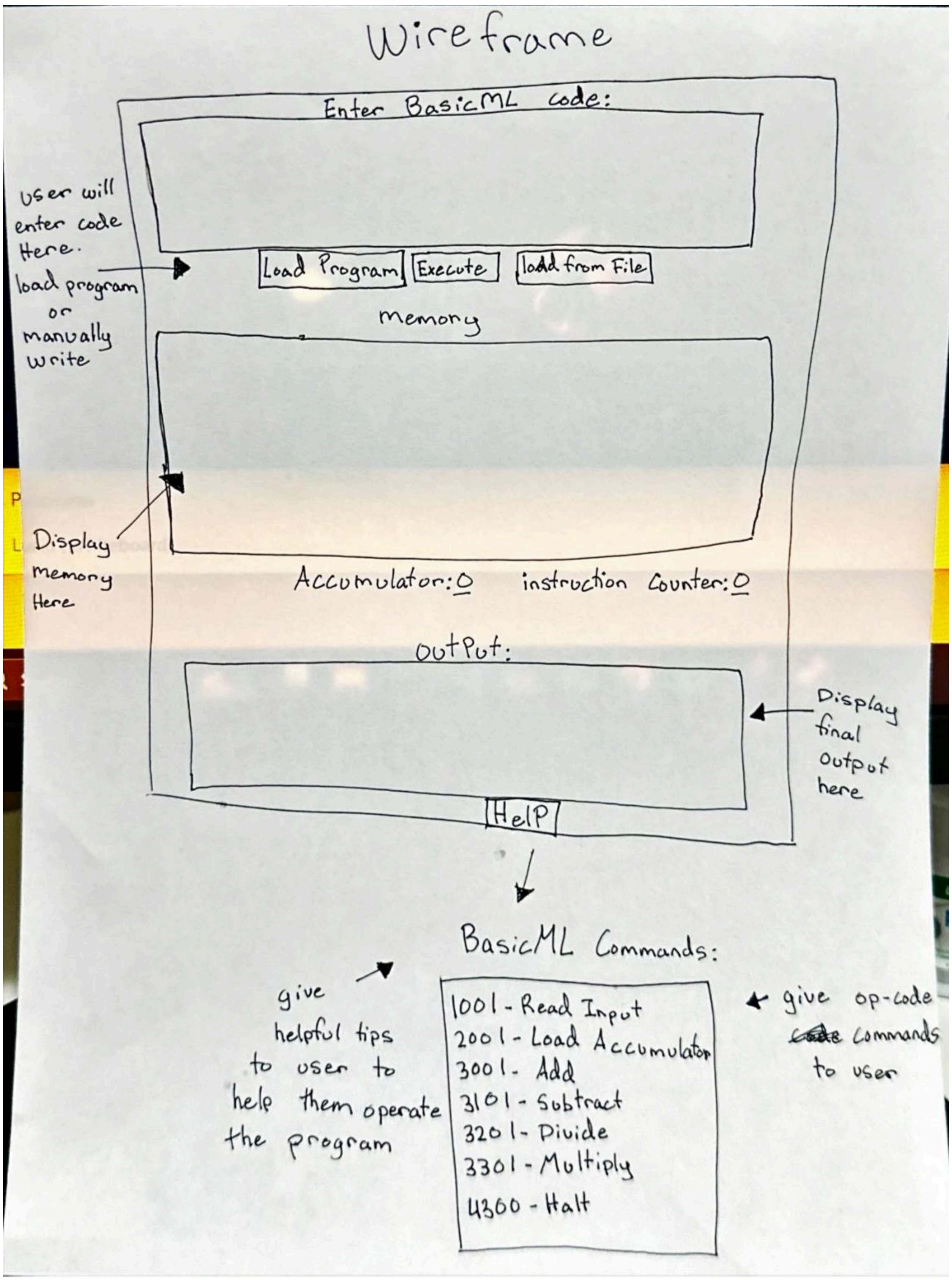
Purpose: Executes the loaded program based on given BasicML instructions.

Pre-conditions: Program must be loaded into memory.

Post-conditions: Executes instructions until a halt command is encountered or an error occurs.

Instruction Set:

- 10XX: Read input into memory location XX.
- 11XX: Write value from memory location XX.
- 20XX: Load value from memory location XX into the accumulator.
- 21XX: Store accumulator value into memory location XX.
- 30XX: Add value from memory location XX to the accumulator.
- 31XX: Subtract value from memory location XX from the accumulator.
- 32XX: Divide accumulator by value from memory location XX.
- 33XX: Multiply accumulator by value from memory location XX.
- 40XX: Unconditional branch to memory location XX.
- 41XX: Branch to XX if accumulator is negative.
- 42XX: Branch to XX if accumulator is zero.
- 43XX: Halt execution.


Wireframe
Enter BasicML code:
user will enter code Here.
load program or manually write
Load Program
Execute
load from File
memory
Display memory Here
Accumulator: 0
instruction Counter: 0
output:
Display final output here
HelP
BasicML Commands:
give helpful tips to user to help them operate the program
1001 - Read Input
2001 - Load Accumulator
3001 - Add
3101 - Subtract
3201 - Divide
3301 - Multiply
4300 - Halt
give op-code ~~code~~ commands to user

Unit Test Descriptions

Class: TestUVSim

Purpose: This class checks for correct output from UVSim through unit tests.

Test Cases:

test_read_valid_input()

Purpose: Tests reading valid user input into memory.
Setup: Stores a read instruction (1005) in memory.
Execution: Mocks user input to simulate reading 1234.
Expected Result: Memory at index 5 stores 1234.

test_read_invalid_input()

Purpose: Tests handling of invalid input.
Setup: Stores a read instruction (1005) in memory.
Execution: Simulates non-numeric input ('abc').
Expected Result: Raises a ValueError.

test_write_output()

Purpose: Tests writing memory value to output.
Setup: Stores 5678 in memory at index 10.
Execution: Runs the write instruction (1110).
Expected Result: Output displays 5678.

test_addition_valid()

Purpose: Tests accumulator addition.
Setup: Accumulator contains 10, memory at index 5 contains 15.
Execution: Runs add instruction (3005).
Expected Result: Accumulator holds 25.

test_divide_by_zero()

Purpose: Tests division by zero error handling.
Setup: Accumulator contains 10, memory at index 5 contains 0.
Execution: Runs divide instruction (3205).
Expected Result: Prints "Error: Division by zero".

test_branch_valid()

Purpose: Tests unconditional branch instruction.
Setup: Stores branch instruction (4020) in memory.
Execution: Runs execute().
Expected Result: program_counter is 20.

test_halt_execution()

Purpose: Tests program halt instruction.
Setup: Stores halt instruction (4300) in memory.
Execution: Runs execute().
Expected Result: Prints "*** Program terminated normally ***"

Future Road Map

Feature: Convert program functionality to a React Web App

Description:

To improve usability and user experience we will convert the UVU sim into a responsive web application using the React framework. The web app will feature an intuitive user interface, fast simulation, and tools to easily run basicML.

Feature: Create a mobile app from the React Web App

Description:

To further increase accessibility to the program we will develop a mobile application by converting our React Web App to function on mobile. This will provide users with easy access to the simulation allowing them to get work done from anywhere.

Feature: Keyboard Shortcuts

Description:

For easier use of the program we will implement keyboard shortcuts for commonly used actions. This will let users work faster as they get more familiar with the program.

Feature: Cloud Sync

Description:

Save, edit, and delete programs from inside the cloud so they can be modified anywhere.

Feature: Visual representation - Color coded

Description:

Have a detailed visual representation that is color coded to show the flow of data between components. The registers will be one color and the executing commands another.

Feature: Tutorial Videos

Description:

Have detailed tutorial videos that explain the software to new users to help them learn the basic functionalities of the program.