UVSim

Group E
Nick Potter
Jarrett Nelson
Margo Kreutz
Katie Hancock

TABLE OF CONTENTS

Summary.....1

User Stories.....1

Use Cases.....2

Functional Specifications.....9

Class Diagram.....11

Wire Frame.....12

Unit Tests.....113

User Manual.....14

Creating A File.....14

Using a File.....14

Editing a File.....15

Changing Color Schemes.....16

Saving a File.....17

Opening a New Tab.....17

Copy / Paste / Delete.....18

Closing the Application.....18

Glossary.....18

Input/Output Operations.....18

Load/Store Operations.....18

Arithmetic Operations.....19

Control Operations.....19

Future Road Map.....19

SUMMARY

UVSimulator was created to help computer science students at UVU learn machine language. The simple virtual machine can interpret BasicML and is designed to teach students the basics of machine language.

Users can import a text file into the program, either in 4-digit format or 6-digit format. All 4-digit format files are automatically converted to 6-digit format upon opening. Each 6-digit word is made up of two parts. The first 3 digits contain an operation code, and the last 3 digits complete the instruction, based on what the instruction is. Potential instructions include input/output, load/store, arithmetic, and control operations. The 250 memory locations can either contain an instruction, contain a data value that can be used by the program, or be empty.

The program also has several features to improve usability for students. The default color scheme of the program is UVU colors, but users can also modify the primary and secondary color in the GUI. Users can also edit the function codes before executing the file. The ability to cut, copy, and paste adds more convenience for users, as does the ability to have more than one tab open and running simultaneously. We plan to continue to grow functionality for users, which will be outlined in a later section.

USER STORIES

A computer science student at UVU wants to be able to apply arithmetic operations to data in order to understand how the register works with operations on data.

A different computer science student wants to be able to read and write to a file, in order to load and store data into memory. Her goal is to understand how loading data into memory works.

USE CASES

Read from Keyboard:

- Actor: User (or another part of the program simulating user input for testing)
- System: UVSim (Universal Virtual Machine Simulator) 'IO' Class
- Goal: To read a word from the keyboard into a specific location in memory.
- Steps:
    1. The Actor invokes the 'read' method of the 'IO' class.
    2. The System prompts the Actor for a signed 6-digit number.
    3. The Actor inputs a signed 6-digit number.
    4. The System validates the input and stores it in the specified memory location.
- Extensions:
    - If the input is not a signed 6-digit number, the System prompts the Actor again until valid input is provided.

Write to Screen

- Actor: System/User (The system initiates the action, but it could be in response to a user command or program algorithm)
- System: UVSim 'IO' Class
- Goal: To write a word from a specific location in memory to the screen.
- Steps:
    1. The Actor requests the 'write' method of the 'IO' class with a specific memory location.
    2. The System retrieves the value from the specified memory location.
    3. The System displays the value on the screen.

Load

- Actor: System/User
- System: UVSim 'Memory' Class
- Goal: Load a word into the Accumulator from a memory location.
- Steps:
    1. Parse function code.
    2. Get desired memory address from the last 3 digits of the word.
    3. Go to that memory address and retrieve the value.

4. Copy retrieved value into the Accumulator register.

Store

- Actor: System/User
- System: UVSim 'Memory' Class
- Goal: Store a word from the Accumulator into a memory location.
- Steps:
    1. Parse function code.
    2. Copy value from the Accumulator register.
    3. Store value from the Accumulator into the specified memory location.

Add

- Actor: System/User
- System: UVSim 'Arithmetic' Class
- Goal: Successfully add the word in the Accumulator to a word from a specific memory location and load the sum into the Accumulator.
- Steps:
    1. Parse function code.
    2. Identify the target memory address from the last 3 digits of the word.
    3. Retrieve the value from the identified memory address.
    4. Add the memory value to the Accumulator value.
    5. Copy the sum value into the Accumulator register.

Subtract

- Actor: System/User
- System: UVSim 'Arithmetic' Class
- Goal: Successfully subtract the word from a specific memory location from the word in the Accumulator and load the difference into the Accumulator.
- Steps:
    1. Parse function code.
    2. Identify the target memory address from the last 3 digits of the word.
    3. Retrieve the value from the identified memory address.
    4. Subtract the memory value from the Accumulator value.
    5. Copy the difference into the Accumulator register.

Divide

- Actor: System/User
- System: UVSim 'Arithmetic' Class

- Goal: Successfully divide the word in the Accumulator by the word in a specific memory location and load the result into the Accumulator.
- Steps:
    1. Parse function code.
    2. Identify the target memory address from the last 3 digits of the word.
    3. Retrieve the value from the identified memory address.
    4. Divide the Accumulator value by the memory value.
    5. Copy the resulting value into the Accumulator register.

Multiply

- Actor: System/User
- System: UVSim 'Arithmetic' Class
- Goal: Successfully multiply the word in the Accumulator by the word in a specific memory location and load the result into the Accumulator.
- Steps:
    1. Parse function code.
    2. Identify the target memory address from the last 3 digits of the word.
    3. Retrieve the value from the identified memory address.
    4. Multiply the Accumulator value by the memory value.
    5. Copy the resulting value into the Accumulator register.

Branch

- Actor: System/User
- System: UVSim 'Control' Class
- Goal: Successfully branch to a designated location in memory.
- Steps:
    1. User will input code 040 (memory location) into their text file.
    2. UVSim will read file that user has selected.
    3. UVSim will select the branch function.
    4. UVSim branch function with change the memory location.
    5. UVSim will verify that the memory location is correct and if it is not, output an error message to the console.
    6. UVSim will be successful if the memory location has changed.

Branch Neg

- Actor: System/User
- System: UVSim 'Control' Class
- Goal: Successfully branch to a designated location in memory if the Accumulator is negative.

- Steps:
    1. User will input code 041 (memory location) in their text file.
    2. UVSim will read the file that the user has selected.
    3. UVSim's 'BRANCHNEG' function will be called while iterating over memory.
    4. UVSim will then check the Accumulator to check if it is negative.
    5. If the Accumulator is negative it will branch to the new memory location.
    6. If the Accumulator is not negative it will continue to the next memory location.
    7. UVSim will be successful if the memory location has changed or continued according to the Accumulator.

Branch Zero

- Actor: System/User
- System: UVSim 'Control' Class
- Goal: Successfully branch to a designated location in memory if the Accumulator is 0.
- Steps:
    1. User will input code 042 (memory location) in their text file.
    2. UVSim will read the file that the user has selected.
    3. UVSim's 'BRANCHZERO' function will be called while iterating over memory.
    4. UVSim will then check the Accumulator to check if it is equal to zero.
    5. If the Accumulator is zero it will branch to the new memory location.
    6. If the Accumulator is not zero it will continue to the next memory location.
    7. UVSim will be successful if the memory location has changed or continued according to the Accumulator.

Halt Program Execution

- Actor: System (Part of the program/algorithm that determines execution should stop)
- System: UVSim 'Control' Class
- Goal: To stop the program execution.
- Steps:
    1. The Actor encounters a condition or instruction that requires halting the program (e.g., a specific instruction in the code or an external interrupt).

2. The Actor invokes the 'halt' method of the 'Control' class.
3. The System sets the program state to indicate that execution should stop.
4. The program ceases further instruction processing and exits the execution loop.

- Extensions:
    - The halt condition can be triggered manually by a user or automatically by reaching a predefined state in the program's execution.

Color Scheme

- Actor: User
- System: UVSim 'tkinterApp' Class
- Goal: To allow users to change the program's color scheme.
- Steps:
    1. The Actor clicks the "Options" button.
    2. The Actor chooses whether they will change the primary or secondary color.
    3. The Actor selects a color and presses "Ok."
    4. When they choose a file, the new colors will be displayed.

Select a File

- Actor: User
- System: UVSim 'Memory' Class
- Goal: To allow users to select a file from the computer directory.
- Steps:
    1. The Actor clicks the "Choose a File" button.
    2. The Actor browses through their directory to find a file to run through UVSim.
    3. The Actor selects a file and presses "Open."
    4. The program displays the file information.
    5. The Actor is able to select "Run File" in order to run the file.

Select a New File

- Actor: User
- System: UVSim 'Memory' Class
- Goal: To allow users to select a new file without reopening the program.
- Steps:
    1. The Actor clicks the "Load New" button.

2. The Actor browses through their computer directory to find a new file to run.
3. The Actor selects a file and presses "Open."
4. The program displays the file information, and the Actor can press "Run File."

Convert 4-Digit Format to 6-Digit Format

- Actor: System
- System: UVSim Application
- Goal: To load and differentiate between old (4-digit) and new (6-digit) file formats, converting old-format files to the new ones.
- Steps:

1. The user selects a file to load.
2. The system checks the word size of the file, identifying it as the old or new format.
3. If the file has been identified as being the old format, 2 leading 0's are added to convert it to the new file format.

Multi-File Management

- Actor: User
- System: UVSim Application
- Goal: To manage multiple files within a single application instance.
- Steps:

1. The user opens multiple files using the GUI, using a button to start a new tab.
2. The system provides tabs or sub-windows for each opened file.
3. The user switches between files as needed for editing and execution.

Editing a File

- Actor: User
- System: UVSim Application
- Goal: To view, add, modify, and delete function codes and data values within the file.

- Steps:
    1. The user loads a file into the GUI.
    2. The GUI displays the selected file.
    3. Upon pushing the edit file button, the file is populated into an entry widget.
    4. The user performs actions such as cut, copy, paste, add, and delete on the function codes and data values.
    5. The system ensures that the maximum data size entries is not exceeded during these operations.

Save

- Actor: User
- System: UVSim Application
- Goal: To save the current state of the file to the original location.
- Steps:
    1. The user selects the "Save" option in the GUI.
    2. The system writes the current state of the file to the disk, overwriting the previous contents.

Save As

- Actor: User
- System: UVSim Application
- Goal: To save the current state of the file under a new name or location.
- Steps:
    1. The user selects the "Save As" option in the GUI.
    2. The user specifies a new name and directory for the file.
    3. The system writes the current state of the file to the new location without altering the original file.

FUNCTIONAL SPECIFICATIONS

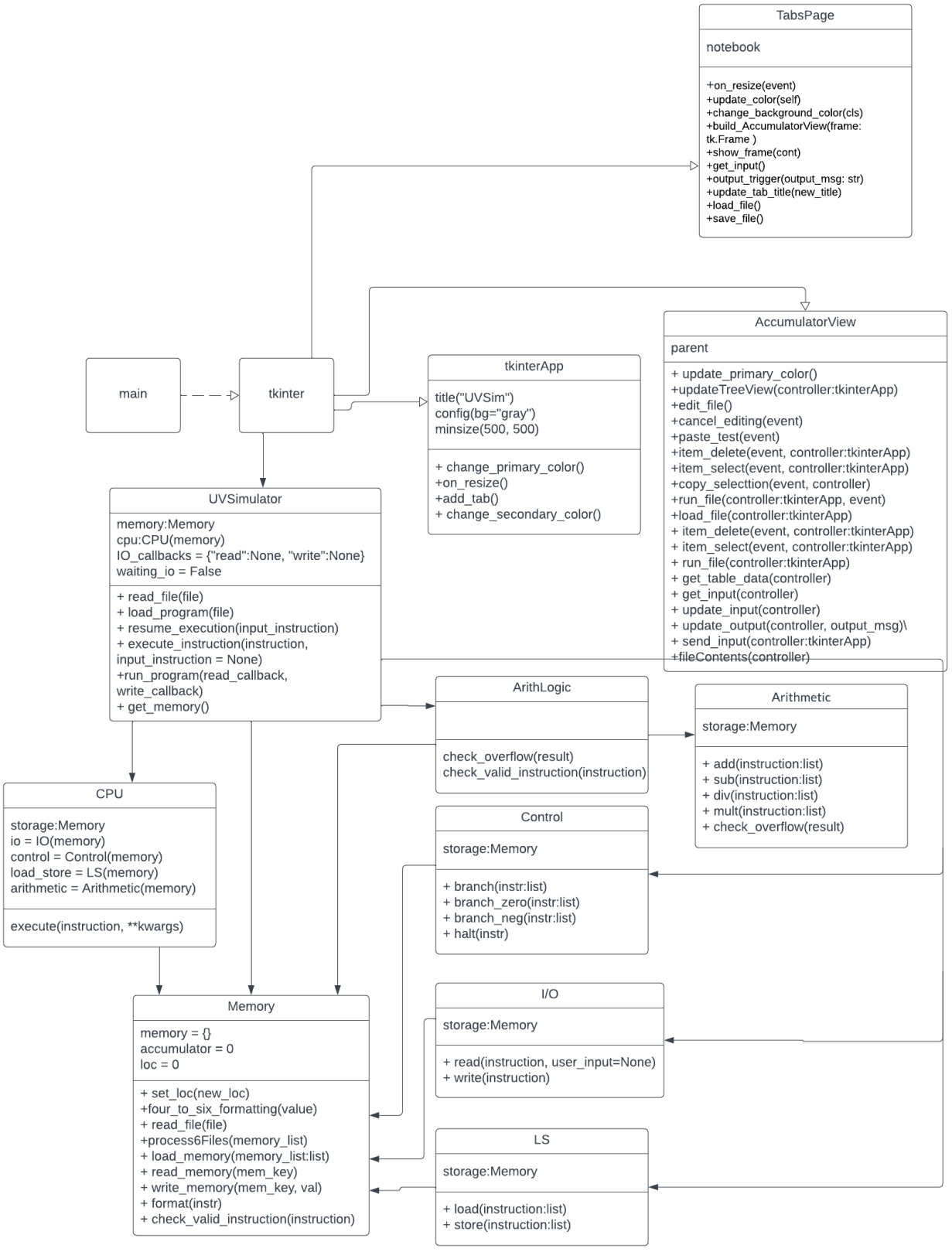
- The system shall prompt the user to select a .txt file that contains BasicML.
- The system shall load the user-selected file into memory starting from memory location 000 up to 249.
- The system shall read a word from the keyboard into a specific location in memory when the instruction code is '010'.
- The system shall write a word from a specific location in memory to the screen when the instruction code is '011'.
- The system shall load a word into the Accumulator from a memory location when the instruction code is '020'.
- The system shall store a word into a specific memory location from the Accumulator when the instruction code is '021'.
- When the instruction code is '030', the system shall add a word in the Accumulator to a word in a specific memory location, loading the sum into the Accumulator.
- When the instruction code is '031', the system shall subtract the word in a specific memory location from the word in the Accumulator, loading the difference into the Accumulator.
- When the instruction code is '032', the system shall divide the word in the Accumulator by the word in a specific memory location, loading the result into the Accumulator.
- When the instruction code is '033', the system shall multiply the word in a specific memory location by the word in the Accumulator, loading the result into the Accumulator.
- The system shall branch to a designated memory location when the instruction code is '040'.
- The system shall branch to a designated memory location when the instruction code is '041' and the Accumulator is negative.
- The system shall branch to a designated memory location when the instruction code is '042' and the Accumulator is '0'.
- The system shall stop program execution when the instruction code is '043', or when prompted by the user.

- In the case of arithmetic overflow, the system shall truncate the value to the final 6 digits.
- The system shall have a maximum of 250 memory locations available.
- The GUI shall display error messages for invalid operations.
- The system shall prompt the user to either run a new file or quit after a halt command.
- The system shall prompt the user to either run a new file or quit after a program has passed memory location 249.
- The system shall read and display the value at a specified memory location when prompted.
- The system shall have a default color scheme.
- The system shall have the ability to create a custom color scheme.
- The system shall have the ability to Copy text.
- The system shall have the ability to Paste text.
- The system shall support 4-digit words and 6-digits words.
- When a user uploads a file containing 4-digit words, the system shall convert this file to 6-digit format upon opening the file.
- The system shall have the ability to edit files.
- The system shall have the ability to save edited files.
- The system shall have the ability to open new tabs that can run simultaneously.

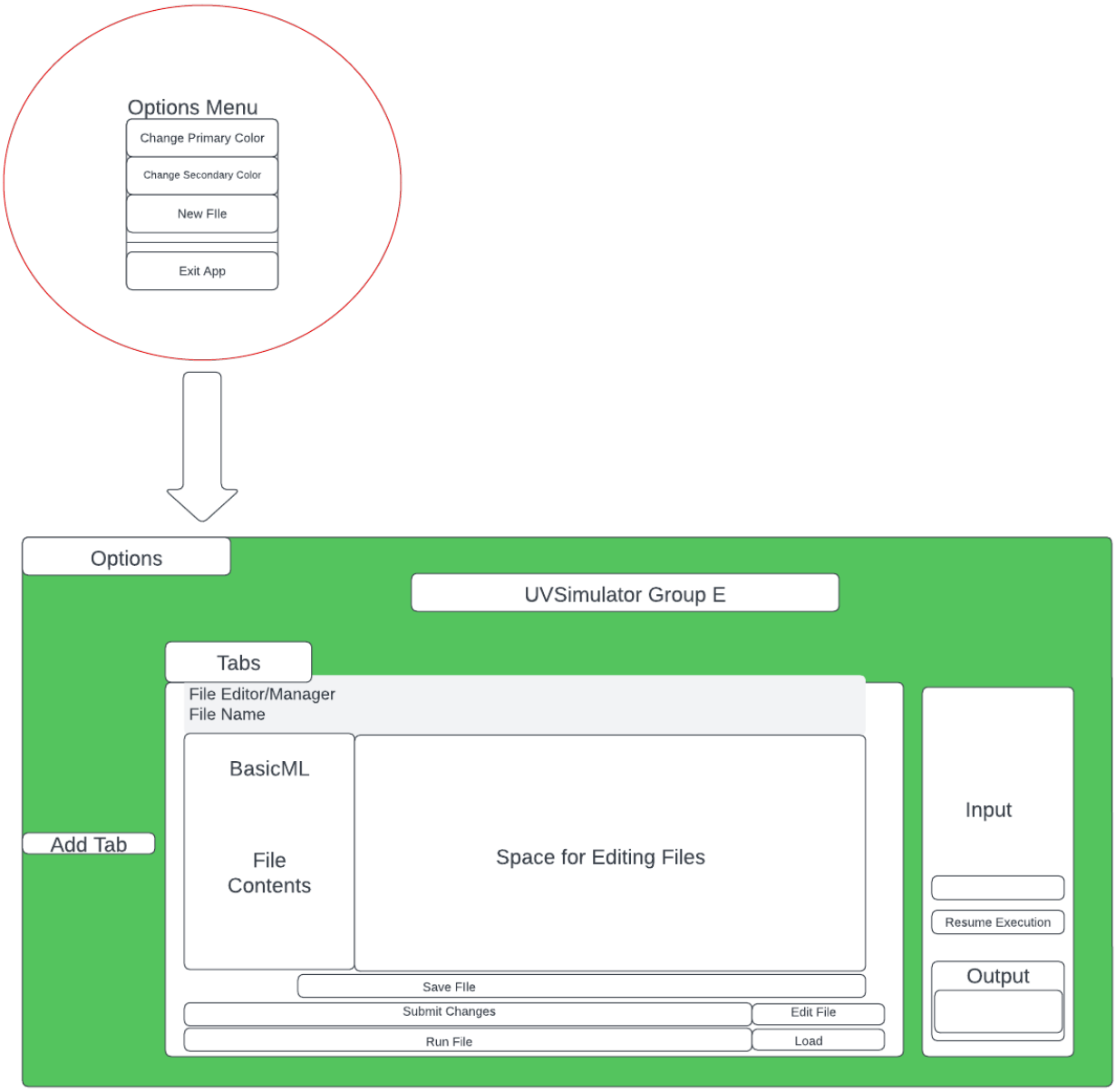
NON-FUNCTIONAL SPECIFICATIONS

- GUI shall have default high contrast colors.
- GUI must have a defined window size to make sure all elements are visible.
- The application must execute on any PC or Mac computer operating system.

CLASS DIAGRAM



WIRE FRAME


Options Menu
Change Primary Color
Change Secondary Color
New FIle
Exit App
Options
UVSimulator Group E
Tabs
File Editor/Manager
File Name
BasicML
File
Contents
Space for Editing Files
Add Tab
Input
Resume Execution
Output
Save FIle
Submit Changes
Edit File
Run File
Load

UNIT TESTS

Our team performed numerous unit tests to ensure that each program operation is fully functional. Below are the operations that we tested:

BRANCH testing: Verify correct branching to several different locations by testing that the location pointer updates to the correct value.

BRANCHNEG testing: Verify that branching only occurs if the Accumulator contains a negative value.

BRANCHZERO test: Verify that branching only occurs if the Accumulator equals 0.

LOAD testing: Verify the LOAD operation, sequentially loading values into the Accumulator from memory. Verify that the Accumulator correctly loads and updates for each instruction, in the correct order.

READ testing: Tests reading an input into a specific memory location to ensure the specified memory location contains the correct value.

WRITE testing: Tests writing content in a specified memory location is written to output, by verifying that the correct value is output. Also verifies that outputted values are correctly positive or negative when necessary.

HALT testing: Verifies that execution halts correctly, by checking that program execution is halted when required or instructed.

ADD testing: Tests the addition operation by verifying that the Accumulator updates to the sum of specified values.

SUBTRACT testing: Tests the subtraction operation by verifying that the Accumulator updates to the difference of specified values.

DIVIDE testing: Tests the division operation by verifying that the Accumulator updates to quotient of specified values.

MULTIPLY testing: Tests the multiplication operation by verifying that the Accumulator updates to the product of specified values.

USER MANUAL

UVSim is a virtual machine designed for interpreting BasicML.

Creating A File

```
+020014
+030015
+021021
+011021
+031017
+021021
+011021
+033019
+021021
+011021
+032020
+021021
+011021
+043000
+011011
+022022
+033033
+044044
+055055
+066066
+000003
```

An example .txt file in 6-digit format.

- Files can be in 4-digit or 6-digit format. All files in 4-digit format will be converted to 6-digit format automatically upon opening.
- Files must only contain one format, either 4-digit or 6-digit.
- For 6-digit words, the first 3 digits are the OpCode, and the last 3 digits are the memory address of the word that the OpCode will be performed on.
- All input-values must have "+" or "-" before the input value.
- When editing a file, you must use the 6-digit format.
- UVSim supports up to 250 memory locations, numbered 000 – 249. Each location can contain an instruction or a data value.
- Each word in a file is read into a memory location when the file is opened, starting at location 000, incrementing for each word.

Using a File

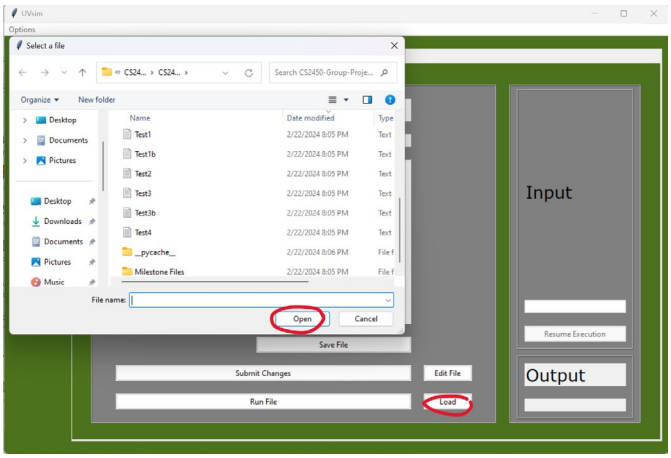
To open and run a file in the application, follow these simple steps:

1. Open a terminal.
2. Type the following command:

```
bash python main.py
```

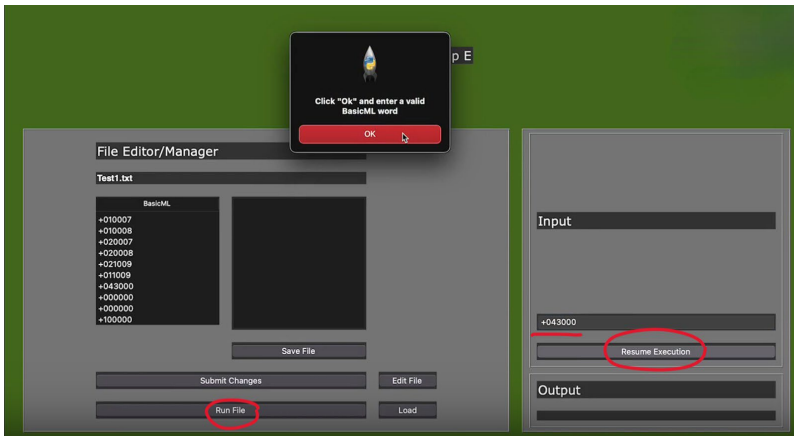
3. The system will open a GUI.
4. You will select a file by clicking the "Load" button. The application will open a file explorer.

5. Choose your desired file and click the "Open" button.
   NOTE: Files must be in .txt format.
6. Run the file by clicking the "Run File button." The system will prompt you to enter a valid BasicML word.
7. Enter a valid BasicML word by typing a valid 6-digit word into the Input Box, then press "Resume Execution."
   NOTE: Input words must begin with a "+" or "-" sign or the application will not run the file.
8. The application will run the file up until another Input value is required, then prompt the user for another input.
   NOTE: The amount of required input values depends on which commands are within the .txt file being used.
9. When prompted to enter a valid BasicML Word, click "Ok" and repeat Step 6, changing the Input value each time until the application has finished running the entire .txt file.
10. The application will then display the Output value.



Click "Load" to open the file explorer.

Select a .txt file and click "Open."



After clicking "Run File," the application will prompt for an input as needed. After typing your input, click "Resume Execution" to continue."

Note: This application is compatible with Windows (above left) and iOS (above right).

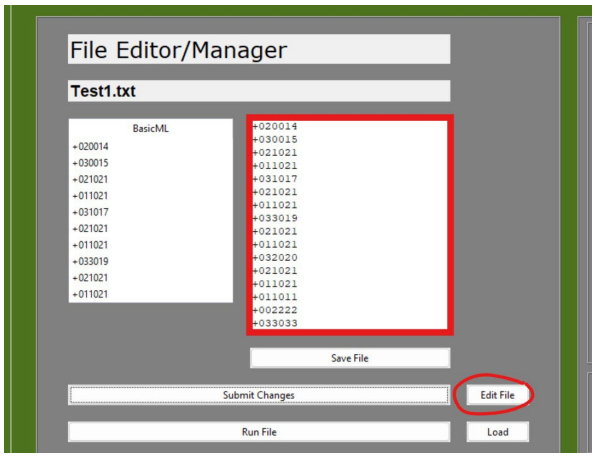
Editing a File

To make edits to any line in your .txt file prior to running it:

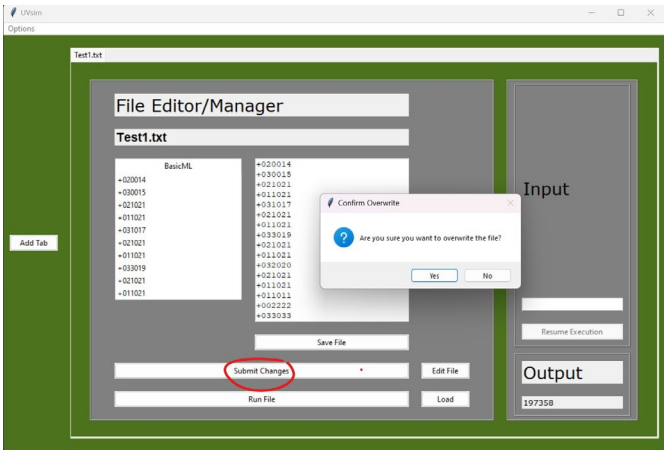
1. Select a file by clicking the "Load" button.
2. Click the "Edit File" button.
3. All contents of the file will load in the display box on the right.
4. Click anywhere inside the display box on the right to edit any line of code.

NOTE: Each line of code must begin with a "+" or "-" sign, followed by 6-digits.

5. When you have finished editing your code, click the "Submit Changes" button.
6. The application will ask "Are you sure you want to overwrite the file?" Click "Yes" to submit the changes to the file.
7. Any changes to the file will now appear in the left display box.



To edit, click "Edit File" and the file contents will appear in the display box on the right for you to edit.

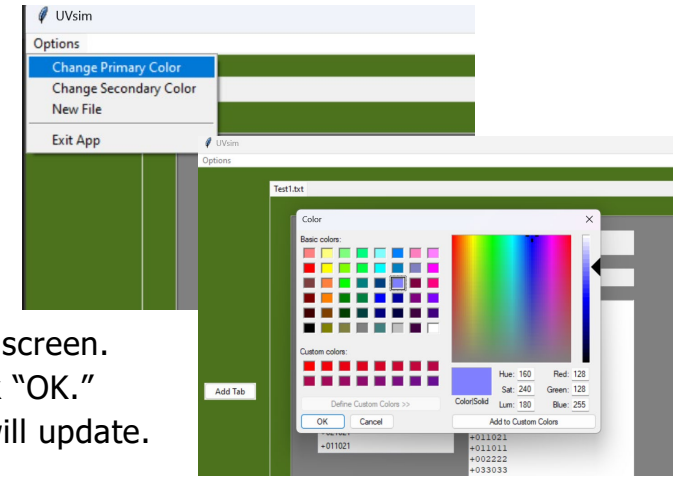


After editing, click "Submit Changes" and then "Yes" to implement any changes.

Changing Color Schemes

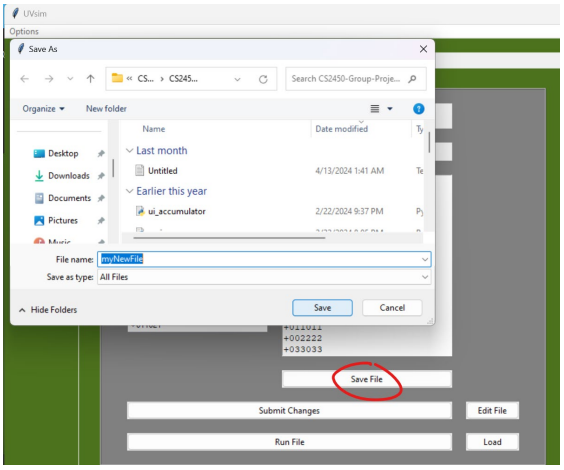
To change the primary and secondary colors of the application:

1. Click on the "Options" tab in the top-left corner of the application.
2. Click "Change Primary Color" to change the background color. Click "Change Secondary Color" to change the accent color.
3. A color wheel will appear on the screen.
4. Select the desired color and click "OK."
5. The application's color scheme will update.



Saving a File

To save any changes to a file:



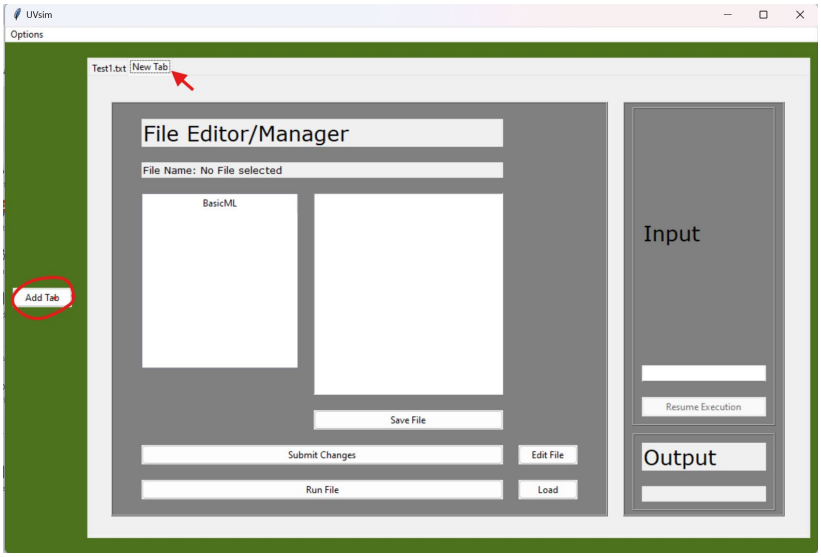
To save a file, click "Save File," choose a location, type the desired file name, and click "Save."

1. Click "Save File."
2. The system will open a file explorer.
3. Browse to the desired folder you want to save your file in.
4. Type the desired name of your file in the "Save as:" box.
5. Click the "Save" button to save your file.
6. A prompt will appear to verify that your file was successfully saved. Click "OK" to exit this box.

Opening a New Tab

To open a new tab to run another file:

1. Click "Add Tab" in the center left of the application.
2. A new tab will appear to the right of the current tab at the top of the application. Click "New Tab" to view the new tab.
3. Follow the steps above to open and use a file in the new tab.
4. The tab name will update when a file is loaded.
5. Click on the desired tab name to switch between tabs.



To add a tab, click "Add Tab." Then click on the new tab to open it.

Copy / Paste / Delete

To copy and paste lines of code into the display box on the right:

1. Select the text you want to copy, either from within the application or from another location. To select multiple lines of code in the application, hold CTRL while selecting.
2. Press CTRL+C to copy the desired code.
3. Click inside the display box on the right.
4. Press CRTL+V to paste desired code.

To delete lines of code in the display box on the left:

1. Select the desired line(s) of code.
2. Press the DELETE key on your keyboard.
3. The selected line(s) have now been removed.

NOTE: Remember to save any changes you have made to your file.

Closing the Application

To exit the application:

1. Click on the "Options" tab in the top left corner of the application.
2. Click on the "Exit App" button to close the program.

Glossary

Accumulator: A register that holds information before UVSim uses it, used for calculation or other temporary uses.

Input/Output Operations

READ = 010: Read a word from the keyboard into a specific location in memory.

WRITE = 011: Write a word from a specific location in memory to the screen.

Load/Store Operations

LOAD = 020: Load a word from a specific location in memory into the Accumulator.

STORE = 021: Store a word from the Accumulator to a specific location in memory.

Arithmetic Operations

ADD = 030: Add a word from a specific location in memory to the word in the Accumulator, leaving the result in the Accumulator).

SUBTRACT = 031: Subtract a word from a specific location in memory from the word in the Accumulator, leaving the result in the Accumulator.

DIVIDE = 032: Divide the word in the Accumulator by a word from a specific location in memory, leaving the result in the Accumulator.

MULTIPLY = 033: Multiply a word from a specific location in memory to the word in the Accumulator, leaving the result in the Accumulator.

Control Operations

BRANCH = 040: Branch to a specific location in memory.

BRANCHNEG = 041: Branch to a specific location in memory if the Accumulator is negative.

BRANCHZERO = 042: Branch to a specific location in memory if the Accumulator is zero.

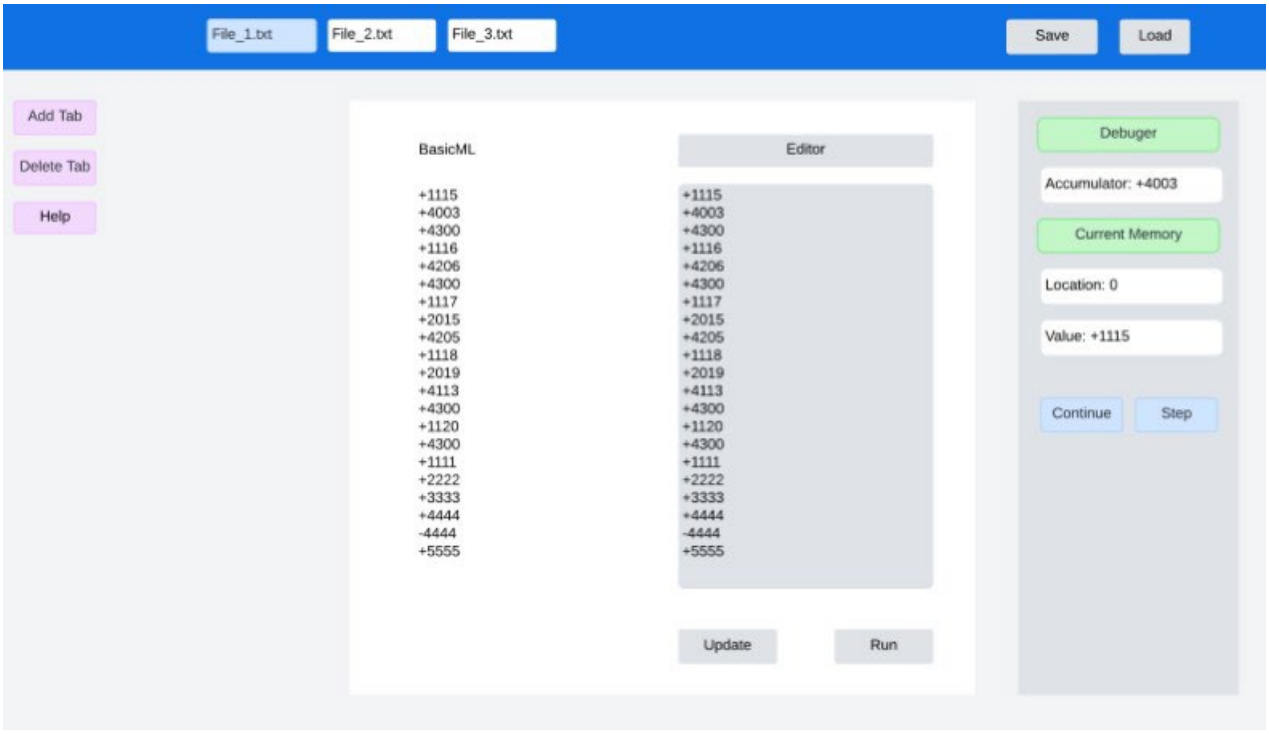
HALT = 043: Stop the program.

FUTURE ROAD MAP

UVSim will only get better as time goes on. There are several features that we plan to introduce that will help to grow and improve the application.

DEBUGGER

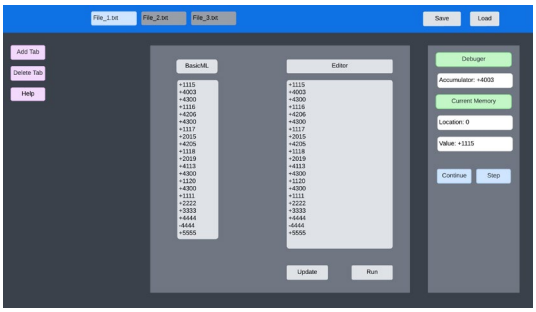
A real-time debugging feature will help users to monitor memory locations and see them update in real-time while the application runs. This will allow users to monitor the current value of the Accumulator. The debugger will also allow users to step through the program one line of code at a time, not only allowing students to pinpoint any errors, but also helping them visualize and learn exactly what is happening within each line of a .txt file.



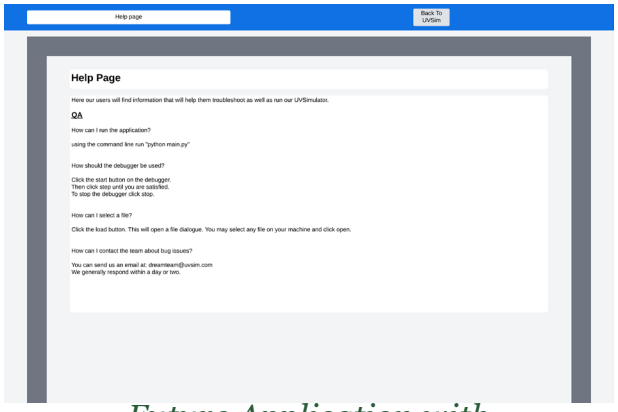
Future Application with a Built-In Debugging Feature

INTERFACE IMPROVEMENTS

In order to improve accessibility for all students, we will transition UVSim to a more modern, clean interface design. We plan to introduce high-contrast themes for those with visual impairments and dark mode themes for better nighttime use. The future of UVSim is bright! Or dark! Or both!



Future Application with Dark Mode Capability



Future Application with Help Page for students

Built-in tutorials will enhance the learning experience for students, allowing them to learn at their own pace. An integrated help section in the menu toolbar will be added to answer any commonly asked questions. Detailed descriptions and examples of each OpCode and function will be readily available to provide support for new users.