UVSim

By

Jake Hodgson, Ben Carroll,

Jarek Smith & Michael Page

Table of Contents

Page 1: Introduction / Executive Summary

Page 4-10: Use Cases & User Stories

Page 11: Class Diagram

Page 12: GUI Wireframe

Page 13-20: Application Instructions

Introduction/Executive Summary

The UVsim project that we created is a modern representation of a virtual machine designed to execute BasicML (Basic Machine Language). It’s a simple assembly language made for educational purposes. This software helps students understand and learn how to load, execute, and alter programs in BasicML. The UVSim is a hands-on learning tool that clearly shows the operations of a computer and its fundamental level.

Our virtual machine is built with Python, specifically Python version 3.8.10. We did this to ensure cross-platform compatibility and usability. Our virtual machine features a graphical user interface (GUI) that allows the user to interact with the software. Through the GUI one can load multiple program files, edit those programs, and execute them. Our latest update to our virtual machine allows programs with up to 250 lines and 3-digit memory addresses, handling of six-digit operations and compatibility with both our older style four-digit and new six-digit BasicML formats.

The UVSim serves as an essential educational tool for those starting out with BasicML. The UVSim gives the user the ability to alter and execute BasicML programs. With this ability, the user will be able to more fully understand how BasicML works and how we can use it.

This project is well-documented to ensure that users can easily set up and use the UVSim to its full potential. Look at the ‘Application Instructions’ section of this document for help setting up your UVSim.

Use Cases & User Stories

**Description:**

The UVSim is a simple virtual machine that executes programs written in BasicML machine language.

**User Stories:**

(User Story 1)

As a UVSim developer

I want to load and execute BasicML programs on the UVSim

So that I can learn how machine language works and become more familiar with computer architecture.

(User Story 2)

As a client

I want a UVSim for my students to learn machine language

So that they can gain a deeper understanding of low-level programming concepts.

(User Story 3)

As an end-user

I want to create a UVSim file that includes machine language instructions

So that functionality can be tested in a safe environment.

(User Story 4)

As a UVSim stakeholder

I want to verify that the program ignores incorrect files

So that the program succeeds with integrity and security.

**Use Cases:**

#1 Load Program

Description: This use case describes the process of loading a BasicML program into the UVSim memory starting at location 00

Actors: Student

System: UVSim

Goal: Load a BasicML program in the UVSim memory starting at location 00

Steps:

1. Student inputs the BasicML program into UVSim

2. UVSim reads the input (a signed four-digit or six-digit number)

3. UVSim stores each instruction into consecutive memory slots, storing up to 250 instructions.

4. UVSim validates that the program fits in the memory and that the instructions are valid.

5. UVSim confirms to the user that the program loaded successfully.

#2 I/O Read

Description: The program includes functionality to read a word from the keyboard and store it in a specified memory location.

Actors: I/O Module (UVsim subsystem)

System: UVSim

Goal: Read a word from the keyboard into a specific location in memory.

Steps:

1. I/O module initiates the I/O READ operation.

2. I/O module prompts the user to enter an integer.

3. User inputs an integer via the keyboard.

4. UVSim validates the input as a four or six-digit number.

5. The I/O module stores the input integer in the specified memory location.

#3 I/O Write

Description: The program can write a word from specific memory locations to the screen.

Actors: I/O Module (UVSim subsystem)

System: UVSim

Goal: Write a word from a specific location in memory to the screen.

Steps:

1. I/O module initiates the I/O WRITE operation

2. I/O module retrieves the number from the specified memory location.

3. I/O module outputs the number to the screen.

#4 Load Operations

Description: The program can load a word from a specific location in memory into the accumulator.

Actors: Memory Access Module

System: UVSim

Goal: Load a word from a specific location in memory into the accumulator.

Steps:

1. Memory access module initiates the LOAD operation

2. Memory access module retrieves the number from the specified memory location.

3. Memory access module loads the number into the accumulator.

#5 Addition Operation

Description: This use case describes how to perform addition in the UVSim, adding one number from a specific place in memory to another in the accumulator.

Actor: Arithmetic Logic Unit (ALU)

System: UVSim

Goal: Perform addition between two numbers, one from memory and the other in the accumulator.

Steps:

1. ALU initiates the Addition operation in the UVSim

2. ALU retrieves the number from the desired memory location

3. ALU adds the retrieved number to the number in the accumulator

4. ALU updates the accumulator with the result of the addition.

#6 Subtraction Operation

Description: This use case describes how to perform subtraction in the UVSim, subtracting one number from a specific place in memory to another in the accumulator.

Actor: Arithmetic Logic Unit (ALU)

System: UVSim

Goal: Perform subtraction between two numbers, one from memory and the other that's in the accumulator.

Steps:

1. ALU initiates the Subtraction operation in the UVSim

2. ALU retrieves the number from the desired memory location

3. ALU subtracts the retrieved number to the number in the accumulator

4. ALU updates the accumulator with the result of the subtraction.

#7 Multiplication Operation

Description: This use case describes how to perform multiplication in the UVSim, multiplying one number from a specific place in memory to another in the accumulator.

Actor: Arithmetic Logic Unit (ALU)

System: UVSim

Goal: Perform multiplication between two numbers, one from memory and the other that's in the accumulator.

Steps:

1. ALU initiates the Multiplication operation in the UVSim

2. ALU retrieves the number from the desired memory location

3. ALU multiplies the retrieved number and the number in the accumulator

4. UVSim updates the accumulator with the result of the multiplication.

#8 Division Operation

Description: This use case describes how to perform division in the UVSim, dividing the number in the accumulator by another number in a specific place in memory.

Actor: Arithmetic Logic Unit (ALU)

System: UVSim

Goal: Perform division between two numbers, one from memory (the divisor) and the other that's in the accumulator (the dividend).

Steps:

1. ALU initiates the Division operation in the UVSim

2. ALU retrieves the number from the desired memory location

3. ALU divides the number in the accumulator by the retrieved number

4. ALU updates the accumulator with the result of the division

#9 BRANCH Operation

Description: This use case describes how to perform a branch operation in the UVSim, branching to a specific place in memory.

Actor: Control Unit

System: UVSim

Goal: Branch to a location specified by the operand.

Steps:

1. Control unit initiates the BRANCH operation in the UVSim

2. Control unit sets the instruction counter to the desired memory location.

#10 BRANCHNEG Operation

Description: This use case describes how to perform a BRANCHNEG operation in the UVSim, branching to a specific place in memory only if the accumulator is negative.

Actor: Control Unit

System: UVSim

Goal: Branch to a location specified by the operand if and only if the current value stored in the accumulator is a negative number.

Steps:

1. Control unit initiates the BRANCHNEG operation in the UVSim

2. Control unit checks if the value in the accumulator is less than zero

3. If it is, UVSim branches to the desired memory location, otherwise it does not

#11 Editing Programs in GUI

Description: Edit BasicML program instructions within the UVSim GUI, including adding, modifying, deleting, cutting, copying, and pasting instructions.

Actor: User/Student

System: UVSim GUI

Goal: Allow users to edit BasicML program instructions before execution.

Steps:

1. User loads a program into the UVSim GUI.

2. User edits program instructions directly within the GUI interface.

3. UVSim GUI validates and updates the program instructions in memory.

4. UVSim GUI confirms to the user that edits are saved and ready for execution.

#12 Saving Edited Programs

Description: Save edited BasicML programs from the UVSim GUI to the file system under a user-chosen directory and filename.

Actor: Student

System: UVSim GUI

Goal: Allow users to save edited BasicML program instructions to a '.txt' file.

Steps:

1. User edits a program within the UVSim GUI from any user-specified folder/directory.

2. User clicks the 'Save Program' button in the UVSim GUI.

3. UVSim GUI prompts the user to choose any directory and enter a new filename.

4. UVSim GUI saves the edited program instructions as a '.txt' file in the user-specified folder/directory.

#13 Open Multiple Files

Description: The GUI opens multiple instruction files to be swapped between, edited, and executed in the app instance.

Actor: User/Student

System: UVSim GUI

Goal: Allows multiple files to be opened simultaneously in the app.

Steps:

1. User launches the UVSim GUI.

2. User hits the 'Load Program' button.

3. UVSim GUI will allow the user to browse their computer for a program file.

4. UVSim will then create a new tab file for the loaded file.

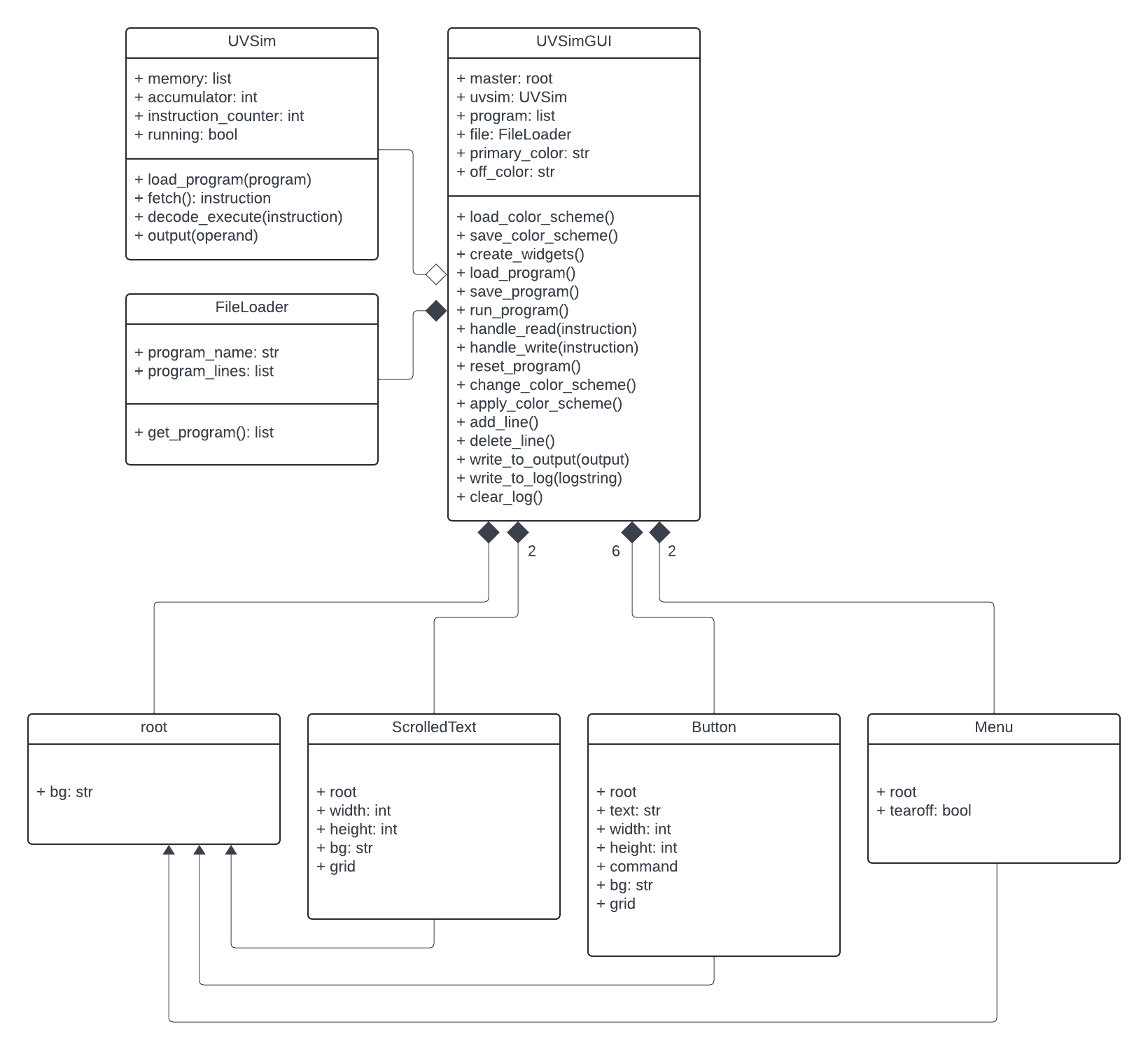
5. User repeats the steps above to open additional files.

6. User will then be able to switch between files, to view and edit them.

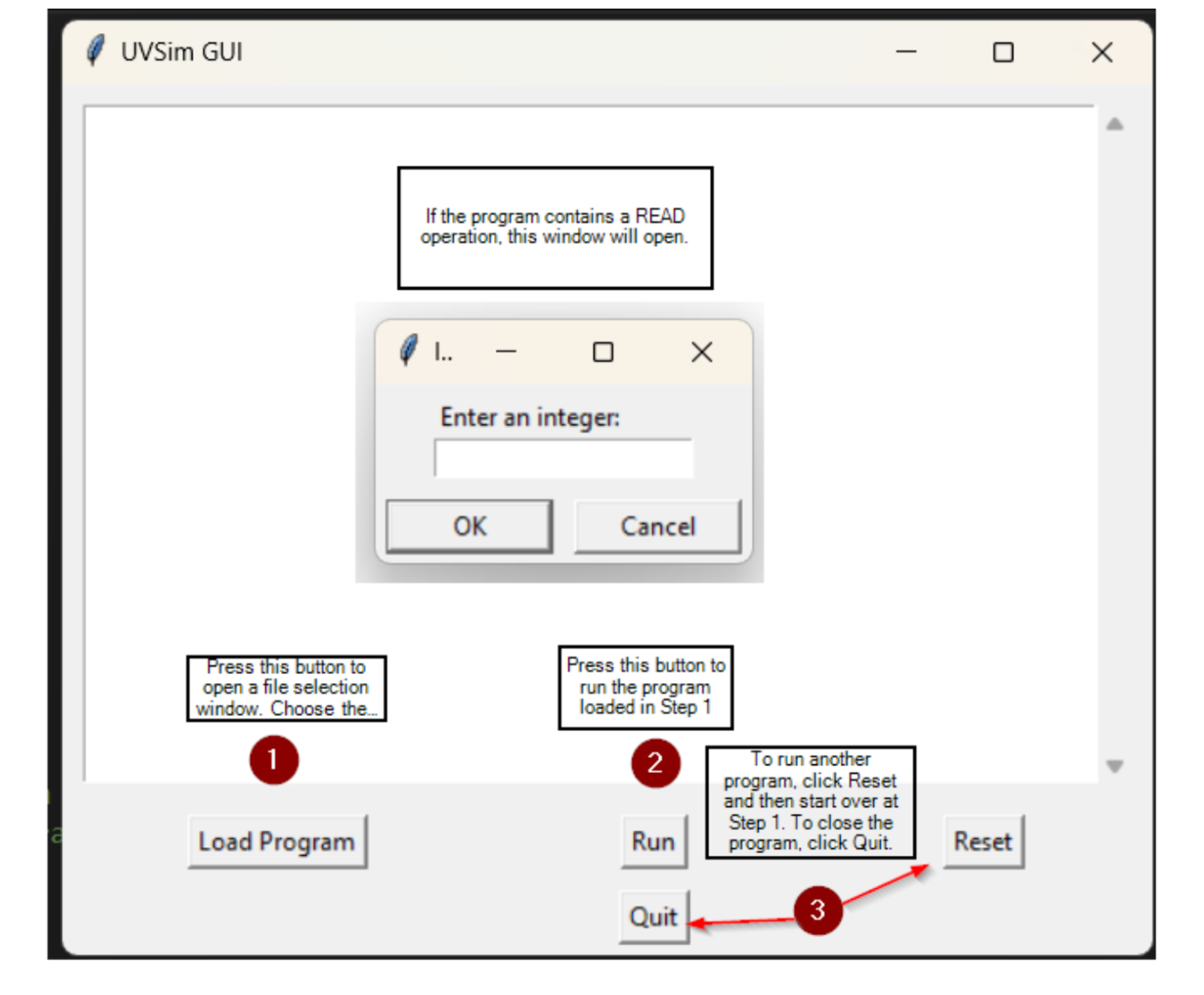
7. The user can open any file, save changes, or execute the program.

8. User hits 'Run' to execute the active file in the selected tab.

Class Diagram



GUI Wireframe



Application Instructions

**Overview:**

The UVSim is a simple virtual machine desgined to execture programs written in BasicML machine language. This README file will provide the reader with all the instructions needed to use the UVSim from the command line.

**Prerequisites:**

Before running the UVSim, make sure that you have Python Installed on your system. The UVSim that we created was created using Python Version 3.8.10

**Installation:**

1. Clone the UVSim repository from GitHub

2. Open your computer's command line interface

3. Type in the following: git clone https://github.com/JHodgson21/CS-2450-Group-Project.git

4. This link is important because here you will be able to obtain the UVSim codebase from our GitHub. By cloning the repository you’ll be able to access the source code and run the program on your own computer.

5. Navigate to the project directory if needed (cd CS-2450-Group-Project)

**Usage:**

To run the UVSim follow these steps:

1. Open a “Command Prompt” on Windows or a “Terminal” on macOS/Linux.

2. Once the command prompt is open, go to the directory where you saved the UVSim repository using the ‘cd’ command.

Example:

cd \Users\Jakob\Desktop\2024 Software Engineering Project\CS-2450-Group-Project

3. Make a txt file for commands to be used in the UVSIM. If you are on Windows search for notepad. If you are on macOS you will be using TextEdit, or you can use any other text editor you like.

4. Writing to your program (to the UVSIM): In the text editor write your program using the commands supported by the UVSIM. Each command should be written on a separate line.

5. Save the file in the same location where the UVSIM is saved. (also make sure its a .txt file).

**Example of a .txt file**

+010007

+010008

+020007

+030008

+021009

+011009

+043000

This txt file explained

‘+010007’: This is a READ operation. It prompts the user to input an integer and stores it in memory to location 07

‘+010008’: This is a READ operation. It prompts the user to input an integer and stores it in memory to location 08

‘+020007’: This is a LOAD operation. It loads the value stored in memory location 07 into the accumulator

‘+030008’: This instruction is an ADD operation. It adds the value stored in memory location 08 to the value currently in the accumulator

‘+021009’: This instruction is a STORE operation. It stores the value currently in the accumulator into memory location 09.

‘+011009’: This instruction is a WRITE operation. It will write or output the value stored in memory location 09.

‘+043000’: This instruction is a HALT operation. It will halt the program, indicating to the computer that the program has finished running.

Run the UVSim application using Python, providing the name of the python file (UVSimGUI.py), example below.

python UVSimGUI.py

After inputting the line above the GUI will pop up on your screen.

Follow the instructions below to use the GUI

**Loading the program:**

1. Click the "Load Program" button.

2. A file dialog will appear. Navigate to your '.txt' file that contains the program instructions and select it.

3. The program will be loaded, and you will see a confirmation message appear in the output text area.

4. On the second half of the screen you will also see the contents or 'instructions' in your .txt file.

**Editing the program:**

1. You can edit the program instructions directly within the GUI on the lower half of the screen.

2. You can add, modify, delete, cut, copy, and paste within the GUI.

3. After you've finished editing you can either hit 'Run' or 'Save Program'.

**Saving Edited Programs:**

1. After editing, you can click "Save Program" to save it as a new file.

2. Choose a directory and enter a filename to save the edited program as a '.txt' file.

**Running the program:**

1. After loading the program, click the "Run" button.

2. If the program requires any input, a prompt will appear in the GUI for you to enter the required values.

3. The output of the program, including any WRITE operations and the final accumulator value, will be displayed in the output text area.

**Resetting the program:**

1. Click the "Reset" button to clear the current program and reset the UVSim instance.

2. This will clear the output text area and reset the internal state of the UVSim.

**Configuring the Color Scheme:**

1. The color sceheme is set to a default UVU Green.

2. To change the color scheme hit the button 'Change Color Scheme'.

3. The GUI will show some default colors as well as allowing the user to choose any color they'd like.

4. The first color you pick will be for the outline of the GUI and the second color will be for the button color as well as the background color.

5. Once you find the first color you like select the color or type in the RGB values of it on the right hand side of the screen and click 'ok'.

- If you'd like to save that specific color hit 'Add To Custom Colors'.

6. After clicking 'ok' you'll choose your second color by selecting it or manually typing in the RGB values of it on the right hand side of the screen. Then once you're finished click 'ok'.

7. The color you chose will carry over throughout your sessions!

**Quitting the application:**

1. Click the "Quit" button to close the UVSim GUI.

How the BasicML vocabulary works within the UVSim.

BasicML vocabulary defined as follows:

I/O operation:

READ = 10 Read a word from the keyboard into a specific location in memory.

WRITE = 11 Write a word from a specific location in memory to screen.

Load/store operations:

LOAD = 20 Load a word from a specific location in memory into the accumulator.

STORE = 21 Store a word from the accumulator into a specific location in memory.

Arithmetic operation:

ADD = 30 Add a word from a specific location in memory to the word in the accumulator (leave the result in the accumulator)

SUBTRACT = 31 Subtract a word from a specific location in memory from the word in the accumulator (leave the result in the accumulator)

DIVIDE = 32 Divide the word in the accumulator by a word from a specific location in memory (leave the result in the accumulator).

MULTIPLY = 33 multiply a word from a specific location in memory to the word in the accumulator (leave the result in the accumulator).

Control operation:

BRANCH = 40 Branch to a specific location in memory

BRANCHNEG = 41 Branch to a specific location in memory if the accumulator is negative.

BRANCHZERO = 42 Branch to a specific location in memory if the accumulator is zero.

HALT = 43 Stop the program

**New Features:**

UVSim Supports 250 Lines and Three-Digit Memory Addresses:

The UVSim supports data files containing up to 250 lines, with internal memory registers going from 000 to 249.

Commands trying to reference a number outside the range of 000-249 will result in an error.

Handling Six-Digit Math Operations:

The UVSim can handle six-digit math operations with proper overflow handling.

Supporting Old and New File Formats:

The UVSim supports both old (four-digit) and new (six-digit) file formats.

Conversion From Four-Digit to Six-Digit Format:

THe UVSim includes a new conversion feature to convert four-digit files to six-digit form.

Multiple Files Open Simultaneously:

The UVSim GUI now allows multiple files to be opened, edited, and saved within the application.

Only one file can be executed at a time.

Other txt files (ideas)

Subtracting two numbers:

+010007

+010008

+020007

+031008

+021009

+011009

+043000

Multiplying two numbers:

+010007

+010008

+020007

+033008

+021009

+011009

+043000

**Other Files That Are Needed to Run the Software:**

Configuration File (config.txt)

The config.txt file is used to store the default color scheme for the UVSIM. It contains two lines, the first is the primary color and the second line is for the off color.

The default colors are set to be UVU's colors which is a darkish green and white.

You don't need to mess with this config.txt file. If you'd like to change the color you can do it through the GUI. ( SEE Configuring The Color Scheme ABOVE ).

FL.py

The FL.py file contains the 'FileLoader' class which is responsible for loading programs from text files into the UVSIM.

IH.py

The IH.py file contains the InstructionHandler4 and InstructionHandler6 classes for parsing instructions.

The main purpose of the InstructionHandler4 function is to handle the instructions in the old four-digit format.

The main purpose of the InstructionHandler6 function is to handle the instructions for the new six-digit format.