**UVSim**

Group C - Alexander Nin, David De Leo, Jess Smith, Zhihui Chen

Table of Contents

[Introduction 3](#_Toc173416529)

[User Stories & Use Cases 4](#_Toc173416530)

[User Stories 4](#_Toc173416531)

[Use Cases 4](#_Toc173416532)

[Functional Specifications 13](#_Toc173416533)

[Class Diagrams & Descriptions 14](#_Toc173416534)

[GUI Wireframes 15](#_Toc173416535)

[Unit Test Descriptions 16](#_Toc173416536)

[Application Instructions 19](#_Toc173416537)

[Future Road Map 20](#_Toc173416538)

# Introduction

The UVSim project delivers a robust virtual machine environment designed to interpret BasicML, a straightforward machine language tailored for educational purposes. This simulator not only mimics a basic computer’s memory architecture with a 250-word memory simulation but also enhances user interaction through a comprehensive graphical user interface (GUI). The UVSim supports a variety of operations including arithmetic calculations, control flow commands, and input/output processes, making it an ideal tool for learning and experimenting with the fundamentals of machine language programming. Integrated features such as multi-file support, color theme customization, and an interactive editor further enrich the user experience, offering both novices and experienced programmers a versatile platform to develop and test machine language programs.

# User Stories & Use Cases

## User Stories

## User Story 1:

## As a computer science student,

## I want to input two numbers, perform arithmetic operations (addition, subtraction, multiplication, division), and see the results,

## so that I can understand how basic machine language operations manipulate data and apply these concepts to learn about computer architecture.

## User Story 2:

## As a computer science student,

## I want to input a number, check if it is positive, negative, or zero, and then direct the flow of the program based on these conditions,

## So that I can understand how conditional branching works in machine language.

## Use Cases

Use case 1: Adding numbers

Actor: UVSim system

System: Arithmetic unit

Goal: To add two numbers and update the accumulator with the result.

Steps:

1. System identifies the 'add' operation code.
2. System retrieves the number from the specified memory address.
3. System adds the number to the accumulator.
4. Result is stored in the accumulator.

Use case 2: Subtract numbers

Actor: UVSim system

System: Arithmetic unit

Goal: To subtract one number from another and update the accumulator with the result.

Steps:

1. System identifies the 'subtract' operation code.
2. System retrieves the number from the specified memory address.
3. System subtracts the number from the accumulator.
4. Result is stored in the accumulator.

Use case 3: Divide numbers

Actor: UVSim system

System: Arithmetic unit

Goal: To divide one number by another and update the accumulator with the result.

Steps:

1. System identifies the 'divide' operation code.
2. System retrieves the number from the specified memory address.
3. System divides the accumulator by the number.
4. Result is stored in the accumulator.

Use case 4: Multiply numbers

Actor: UVSim system

System: Arithmetic unit

Goal: To multiply two numbers and update the accumulator with the result.

Steps:

1. System identifies the 'multiply' operation code.
2. System retrieves the number from the specified memory address.
3. System multiplies the number by the accumulator.
4. Result is stored in the accumulator.

Use case 5: Read

Actor: User

System: Input/Output handler

Goal: To read input from the user and store it in memory.

Steps:

1. System prompts the user for input.
2. User enters data.
3. System stores the input in the specified memory location.

Use case 6: Write

Actor: UVSim system

System: Input/Output handler

Goal: To display a value from memory to the screen.

Steps:

1. System retrieves a value from a specified memory address.
2. System outputs the value to the screen.

Use case 7: Load

Actor: UVSim system

System: Memory management unit

Goal: To load a word from a specific memory location into the accumulator.

Steps:

1. System identifies the 'load' operation code.
2. System retrieves a word from the specified memory address.
3. System updates the accumulator with the retrieved value.

Use case 8: Store

Actor: UVSim system

System: Memory management unit

Goal: To store the value from the accumulator into a specified memory location.

Steps:

1. System identifies the 'store' operation code.
2. System writes the value of the accumulator to the specified memory address.

Use Case 9: Branch Actor: UVSim system

System: Control unit

Goal: To change the instruction pointer based on a specified condition or unconditionally.

Steps:

1. System identifies the 'branch' operation code.
2. The instruction pointer is updated to the specified memory address.
3. The system continues execution from the new instruction pointer location.

Use case 10: Halt

Actor: UVSim system

System: Control unit

Goal: To halt the virtual machine process and end program execution.

Steps:

1. Load program
2. Find address
3. Halt program process and exits program

Use Case 11: Edit Instruction

Actor: User

System: Instruction Editor

Goal: To modify an existing instruction in the program's memory.

Steps:

1. User selects the instruction to be edited.
2. User modified instruction.
3. System updates the instruction in the memory.

Use Case 12: Save Edited Instructions

Actor: User

System: File Management System

Goal: To save the current set of instructions in the memory to a new file.

Steps:

1. User edits the instructions.
2. User initiates the Save As operation.
3. System prompts the user to enter a filename for the new file.
4. User enters the filename and confirms the save operation.
5. System writes the instructions from memory to the new file.

Use Case 13: Change Color Theme

Actor: User

System: User Interface

Goal: To change the color theme of the program's GUI.

Steps:

1. User clicks the Change Color Theme option in the menu.
2. System displays the color palette for the user to choose a new color theme.
3. User selects a desired color theme.
4. System applies the new color theme to the GUI.

Use Case 14: Clear Output

Actor: User

System: User Interface

Goal: To clear the previous output in console.

Steps:

1. User clicks the Clear Output option.
2. System clears all text and data previously displayed in the console.

Use Case 15: Support GUI

Actor: User

System: Graphical User Interface

Goal: To interact with the program through a GUI.

Steps:

1. User launches the program, which starts with the GUI.
2. User interacts with the GUI elements such as buttons and input fields to perform operations (e.g., load, edit, save instructions).
3. System responds to each interaction by updating the GUI accordingly (e.g., displaying data, updating status).
4. User closes the GUI to end the session.

Use Case 16: Manage Multiple Files Simultaneously

Actor: User

System: User Interface and File Management System

Goal: To open, edit, and manage multiple files simultaneously within a single application instance.

Steps:

1. User opens multiple files using the menu.
2. System displays each file in a separate tab within the main application window.
3. User switches between tabs or windows to edit different files.
4. System keeps track of changes in each file independently.
5. User executes or saves files individually.

Use Case 17: Support Different Format

Actor: System

System: File and Memory Management System

Goal: To handle data files and internal memory with a new six-digit word format.

Steps:

1. System detects the format of the file being loaded (four or six digits) based on file metadata or initial inspection.
2. System processes file operations, ensuring all memory addresses and operations codes are valid and conform to the six-digit requirement.
3. System checks for overflow conditions and handles them as per the six-digit operation limits.

# Functional Specifications

TODO

# Class Diagrams & Descriptions

TODO

# GUI Wireframes

A screenshot of a computer

Description automatically generated

# Unit Test Descriptions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Name | Reference | Description | Inputs | Expected Outputs | PASSED? |
| Set Memory | TestSetMemory | Test if memory can be set | register 0 = 1001 | register 0 = 1001 | PASS |
| Load Program | TestLoadProgram | Test if the file can be loaded successfully | test1.txt | register 0 = 1007 ... register 7 = 0000 | PASS |
| Read File Fail | TestLoadProgramFail | Test if a non-existing file reports an error | FAKE\_FILE.txt | error message | PASS |
| Accumulator | TestAccumulator | Test whether the accumulator can be set to a value | accumulator = 1002 | accumulator = 1002 | PASS |
| Halt | TestHalt | Test to see if halt works | N/A | halt program | PASS |
| Instruction Pointer | TestInstructionPointer | Test to see if the instruction pointer's value can be obtained | N/A | instructionPointer = 0 | PASS |
| Instruction Pointer Increment | TestInstructionPointerIncrement | Tests whether the value of the instruction pointer can be modified | instructionPointer = 1 | instructionPointer = 1 | PASS |
| Read | TestRead | Test that the input values are stored correctly | input = 7 | memory 10 = 7 | PASS |
| Read Fail | TestReadFail | Test to see if an error can be thrown if the input value is abnormal | input = a | error message | PASS |
| Write | TestWrite | Test to see if the value in memory can be read correctly | N/A | memory 10 = 9 | PASS |
| Load | TestLoad | Tests if the value can be read and added to the accumulator | N/A | accumulator = 9 | PASS |
| Store | TestStore | Test to see if the value in the accumulator can be saved to memory | N/A | memory 10 = 9 | PASS |
| Branch | TestBranch | Test to see if function can jump to the correct address | N/A | accumulator = 0 | PASS |
| Add | TestAdd | Test that the addition works and gives the correct answer | memory 10 = 9 | accumulator = 9 | PASS |
| Subtract | TestSubtract | Test that subtraction works and gives the correct answer | memory 10 = 9 | accumulator = -9 | PASS |
| Multiply | TestMultiply | Test that the multiplication works and gives the correct answer | memory 10 = 9; accumulator = 10 | accumulator = 90 | PASS |
| Divide | TestDivide | Test that the division works and gives the correct answer | memory 10 = 5; accumulator = 10 | accumulator = 2 | PASS |
| Branch Negative | TestBranchNeg | Tests if the function can be executed when the accumulator is negative | accumulator = -10 | accumulator = -10 | PASS |
| Branch Zero | TestBranchZero | Tests if the function can be executed when the accumulator is 0 | accumulator = 0 | accumulator = 0 | PASS |
| Divide Fail | TestDivideFail | Tests whether an error can be thrown when calculating division if the value is illegal | memory 10 = 0; accumulator = 0 | error message | PASS |

# Application Instructions

TODO

# Future Road Map

TODO