Software requirement specification (SRS) document template

Project name: Group Project Milestone 2

Date: 12/11/24

Version: 2

By: Chris Aldous, Elijah Rutherford, Sam Christmas

Revision history			
Version	**Author**	**Version description**	**Date completed**
1.0	All	4 Digit Input	11/22/24

Review history			
Approving party	**Version approved**	**Signature**	**Date**

Table of contents

Reviewer	Version reviewed	Signature	Date

Approval history

1 Introduction

Describe the purpose of the document.

The purpose of this document is to describe the high-level functionality and features of the UVSim application.

1.1 Product scope

List the benefits, objectives, and goals of the product.

UVSim is a software simulator designed to help computer science students learn machine language and computer architecture through practical experience. It allows students to write,

load, and execute BasicML programs in a controlled virtual environment. This simulator is aimed to help instructors provide an additional tool to aid in their students learning.

1.2 Product value

Describe how the audience will find value in the product.

UVSim is a useful tool that allows educators to provide their students an opportunity to gain valuable experience developing with BasicML in a low-risk environment where they can experiment and gain a better understanding of computer architecture.

1.3 Intended audience

Write who the product is intended to serve.

The intended audience for UVSim includes computer science students, educators and instructors, educational institutions, and self-learners.

1.4 Intended use

Describe how will the intended audience use this product.

UVSim is intended to serve as a supplemental resource for instructors, allowing their students to gain experience writing, loading, and executing BasicML programs in order to gain a better understanding of computer architecture.

1.5 General description

Give a summary of the functions the software would perform and the features to be included.

UVSim is a simple virtual machine designed to help computer science students learn machine language and computer architecture. It executes a machine language called BasicML. Users can either upload BasicML files from their computer or manually enter their programs. It will be able to execute BasicML instructions, perform operations such as Add, Subtract, Divide, and Multiply. It will provide branching controls that allow users to allow controlling the flow of executions based on the value in the accumulator. It will also include methods to manage memory. Users are able to modify the colors of the user interface and open multiple windows of UVSim simultaneously.

BasicML programs are loaded into memory starting at memory location 00. From here, the CPU fetches, decodes, and executes each BasicML instruction in sequence.

The accumulator is then used to perform calculations.

Results can be displayed using the WRITE command.

Users are also able to save and store the BasicML programs they build using UVSim.

2 Functional requirements

List the design requirements, graphics requirements, operating system requirements, and constraints of the product.

The design requirements for UVSim are the ability to load and execute BasicML programs through operations such as Read, Load, and Store, which will allow users to input their programs and store them in memory. It will allow the use of arithmetic operators such as Add, Subtract, Divide, and Multiply. The program will also support the use of output control through the Write operator. It will support the use of control operators such as Branch, branchNeg, branchZero, and Halt to dictate the flow of execution.

The user interface displays where words are stored in memory and which memory locations are open. This version of UVSim requires a Windows operating system.

The constraint of this version of UVSim is the simulated memory limit of 100- 250 words. This was done to force users to carefully manage memory in order to gain valuable memory management experience.

3 External interface requirements

3.1 User interface requirements	Describe the logic behind the interactions between the users and the software (screen layouts, style guides, etc).

The users are able to interact with the UVSim application via the user interface.

Users are able to specify the desired memory size, manually input BasicML instructions, and customize the colors in the user interface.

Users are also able to open multiple windows of UVSim simultaneously, as well as load and save files by interacting with the user interface.

3.2 Hardware interface requirements	List the supported devices the software is intended to run on, the network requirements, and the communication protocols to be used.

3.3 Software interface requirements	Include the connections between your product and other software components, including frontend/backend framework, libraries, etc.

This version of UVSim requires a Windows operating system to run.

3.4 Communication interface requirements	List any requirements for the communication programs your product will use, like emails or embedded forms.

4 Non-functional requirements

4.1 Security	Include any privacy and data protection regulations that should be adhered to.
4.2 Capacity	Describe the current and future storage needs of your software.
4.3 Compatibility	List the minimum hardware requirements for your software.
4.4 Reliability	Calculate what the critical failure time of your product would be under normal usage.
4.5 Scalability	Calculate the highest workloads under which your software will still perform as expected.

4.6 Maintainability

	Continuous integration should be used to deploy features and fix bugs quickly

4.7 Usability

	The UVSim application was designed to be very user friendly, allowing for inexperienced users to easily navigate through the program.

Our goal is to provide a user-friendly experience for students learning BasicML using UVSim

5 Definitions and acronyms

Appendix

User Stories:

As a student, I want to write and execute BasicML programs so that I can become familiar with machine language and gain a better understanding of computer architecture. By using the BasicML simulator, I will develop my coding skills as well as be more prepared for exams.

As an educator, I want to analyze the BasicML programs submitted by my students so that I can assess their understanding of key concepts in computer architecture and provide useful and constructive feedback. This will help to make the understanding of this abstract subject easier to grasp and establish a good foundation of computer architecture knowledge.

Use Cases

The system is UVSim for all use cases

Use Case 1: Read Input from Keyboard

Primary Actor: Student

Scenario: During program execution, the program requests user input. The user enters the required data using the keyboard, and the system stores this data in the specified memory location.

Use Case 2: Write Output to Screen

Primary Actor: Student

Scenario: Upon completing the program execution, the user can request output to be displayed using the Write command. The system retrieves the specified memory content and presents it on the screen.

Use Case 3: Load a word into accumulator

Actor: Student

Scenario: The student will be able to successfully load a word from memory into the accumulator.

Use Case 4: Store a word from the accumulator into memory

Actor: Student

Scenario: The student will be able to store the contents of the accumulator into a memory location.

Use Case 5: Add

Primary Actor: Student

Scenario: The user writes a BasicML program that includes the add operator. This operator correctly adds the word in the accumulator to a word from memory and the results are stored in the accumulator.

Use Case 6: Subtract

Primary Actor: Student

Scenario: The user writes a BasicML program that includes the subtract operator. This operator correctly subtracts the word in the accumulator from a word in memory and the results are stored in the accumulator.

Use Case 7: Divide

Primary Actor: Student

Scenario: The user writes a BasicML program that includes the divide operator. This operator correctly divides the word in the accumulator by a word from memory and the results are stored in the accumulator.

Use Case 8: Multiply

Primary Actor: Student

Scenario: The user writes a BasicML program that includes the subtract operator. This operator correctly subtracts the value in the accumulator from a value stored in memory and the results are stored in the accumulator.

Use Case 9: Halt Program Execution

Primary Actor: Student

Scenario: While the program is running, the user chooses to halt execution. The system stops processing the program.

Use Case 10: Handle Invalid Instructions

Primary Actor: System

Scenario: The user runs a program that contains invalid BasicML instructions. The system detects the error, displays an appropriate error message, and halts execution.

Use Case 11: Load Instructions from Text File

Actor: Student

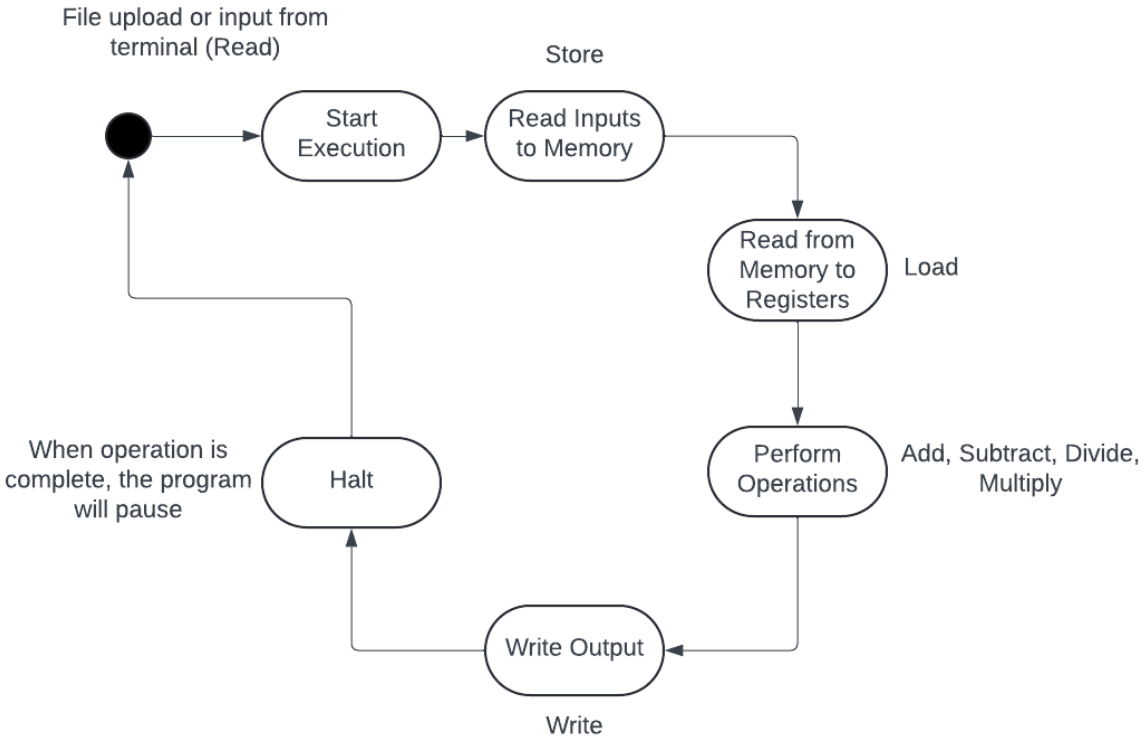
Scenario: The student selects a valid text file. The instructions from the file are successfully loaded into the memory registers, starting at the specified memory location.

Use Case 12: Execute Program Primary

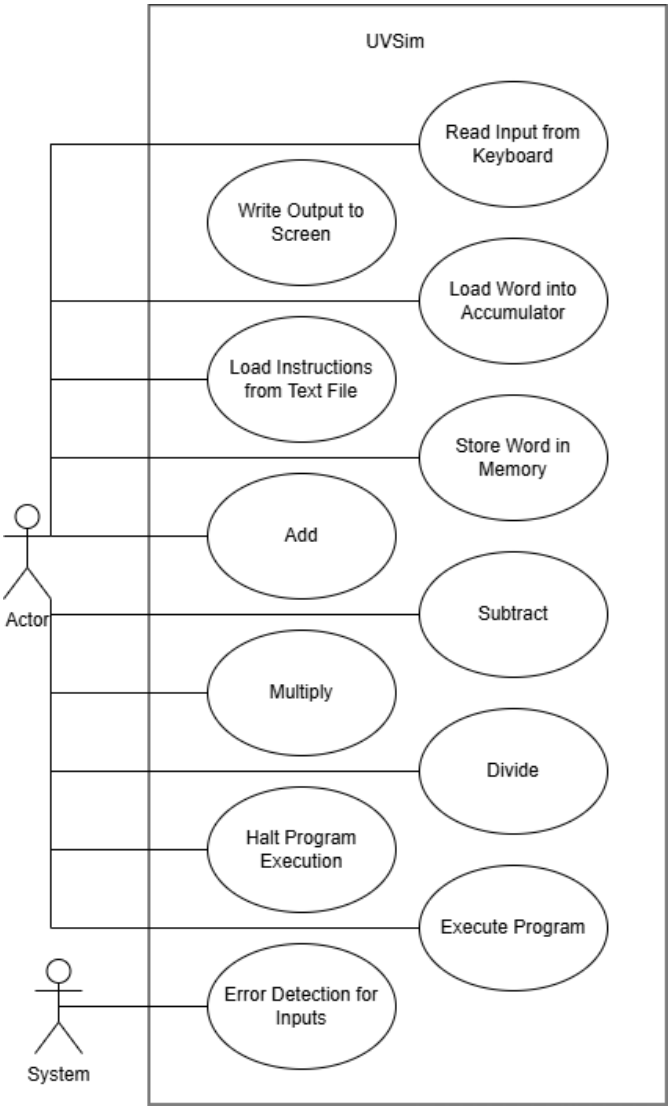
Actor: Student

Scenario: The student executes the program they have loaded into UVSim using the user interface.

State Transition Diagram



Use Case Diagram



Test Case Spreadsheet

Test cases:	Description:	Input:	Expected Out:	Pass or Fail
TestWritePos	Testing the write function with a Positive Number.	10		Pass
TestWriteNeg	Testing the write function with a Negative Number.	-10		Pass
TestAddPos	Testing the Add function with 2 positive numbers.	10 and 10	20	Pass
TestAddNegToPos	Testing the Add function with a positive number and a negative number.	10 and -10	0	Pass
TestSubPos	Testing the Subtract function with 2 positive numbers.	10 and 5	5	Pass
TestSubNegFromPos	Testing the Subtract function with a positive number and a negative number.	10 and -5	15	Pass
TestDividePos	Testing the Divide function with 2 positive numbers	10 and 5	2	Pass
TestDivideNegFromPos	Testing the Divide function with a positive number and a negative number.	10 and -5	-2	Pass
TestMultPos	Testing the Multiply function with 2 positive numbers	10 and 5	50	Pass
TestMultNeg	Testing the Multiply function with 2 Negative numbers	-5 and -4	20	Pass
TestBranch	Testing the Branch function to jump to another location in memory	Initial location = 3 Branch location = 25	location = 25	Pass
TestBranchNeg	Testing the BranchNeg function to jump to another location in memory if conditions are met	Initial location = 0 Branch location = 34 accumulator = -20	location = 34	Pass
TestBranchNeg2	Testing the BranchNeg function to not jump to another location in memory if conditions are not met.	Initial location = 0 Branch location = 34 accumulator = 17	location = 0	Pass

TestBranchZero	Testing the BranchZero function to jump to another location in memory if conditions are met	Initial location = 0 Branch location = 21 accumulator = 0	location = 21	Pass
TestHalt	Testing the Halt function to stop executing code and jump out of memory			Pass