Software Requirement Specification

Project Name: UVSim
Date: 11/23/2024
Version: 0.2
By: Caleb Beazel, Hope Fager, Thor Labrum, Conner Nunley

1 - Introduction

1.1 - Product Scope
UVSim is a program designed for the purpose of teaching computer science students the fundamentals of computer architecture. It simulates interacting directly with memory and a register, doing basic memory and arithmetic operations on data stored in memory addresses.

1.2 - Product Value
UVSim is a simplified assembly code simulator that allows students new to computer science and computer architecture to gain an understanding of the fundamentals in computer architecture and an assembly-like language.

1.3 - Intended Audience
Students and individuals new to computer science/computer architecture.

1.4 - Intended Use
UVSim provides a way to write simulated assembly code files that can be read into the system, and edited in a visually friendly way. UVSim incorporates a graphical user interface to simplify the process of writing to different locations in memory in a way that makes programming in an assembly-like code easier on new users.

1.5 - General Description
UVSim provides the option to read in a seed file with different memory or arithmetic commands to write simple assembly-like code. Each “word” written by the user uses the first two numbers to specify a given command/opcode with the following 4 values being used to specify a location in memory or integer value dependent on the opcode in use. The user will first be able to select a seed file from their computer in the GUI. Once a file has been chosen, the user has the option to look at their file loaded into the memory addresses and make edits to the file. Their program can then be run in the terminal window provided, to show the files output, error codes, or lack thereof.

2 - Functional Requirements

Input and Output

Input from Keyboard
A user must be able to input numerical data.
Purpose: To perform operations without modifying source code.

Output to Screen
A user must be able to view results of operations and instructions.
Purpose: To ensure users can see and interact with the program.

Memory Operations

Load from Memory
A user must be able to load a value from a specific memory address into the accumulator.
Purpose: To perform calculations on stored values.

Store to Memory
A user must be able to store the accumulator's current value to a specific memory address.
Purpose: To save the results of computations.

Arithmetic Operations

Addition
A user must be able to add a memory value to the accumulator, ensuring results remain within the range -999999 to 999999.
Purpose: To calculate sums without overflow.

Subtraction
A user must be able to subtract a memory value from the accumulator.
Purpose: To perform subtraction and compare values (negative result = less than, positive = greater than).

Division
A user must be able to divide the accumulator by a memory value, handling remainders appropriately.
Purpose: To enable division, the ability to determine even/odd numbers, modulus, etc.

Multiplication
A user must be able to multiply the accumulator by a memory value.
Purpose: To perform multiplication and operations like squaring numbers.

Control Flow

Unconditional Branch
A user must be able to jump to a specific memory address.
Purpose: To implement loops and alter program flow.

Branch on Negative
A user must be able to jump to a specific instruction address if the accumulator contains a negative value.
Purpose: To implement conditional loops or decisions like when A > B.

Branch on Zero
The system must jump to a specific instruction address if the accumulator contains zero.
Purpose: To iterate loops based on decrements to zero.

Program Halt
The system must provide an instruction to terminate execution.
Purpose: To prevent infinite loops.

Instruction Handling

Command vs. Data Differentiation
The system must distinguish between executable commands and data values in memory.
Purpose: To ensure instructions are processed correctly.

3 - Non-Functional Requirements

Performance
The program should execute instructions with minimal delay to provide a positive experience for the user.

The program should run effectively with up to 250 lines of memory or more, without a significant drop in performance.

Usability
The user interface should be sufficiently intuitive, such that a user can navigate the program and receive outputs and errors without confusion.

Reliability
The simulator must be able to handle edge cases, such as out-of-bounds memory access, without crashing.

The simulator should be able to handle calculations involving all 250 spaces in memory in under 1 second.

Compatibility
The simulator should be able to run on multiple platforms including Windows, MacOS, and Linux.

Maintainability
The code should be strongly rooted in object-oriented programming and well documented so as to make code modular, easy to update, and debug.

Security
The simulator should validate user input to ensure values are of the correct type and within the scope of the program, to prevent invalid commands and memory corruption.

Capacity
The simulator should allow for 250 memory addresses and be easily adjustable to add more at some future date.

Scalability
The simulator should be able to run effectively with double the amount of memory that it currently has.

4 - User Stories

1. As a user, I want to be able to read in a file into the simulator, so that I can work with an existing file.
2. As a user, I want to input data at runtime, so that I can work on my code within the simulator.
3. As a user, I want to view program results clearly on the screen, so that I can understand what my program is outputting and can view results/errors.
4. As a user, I want to be able to store and retrieve data, so that I can make multi-step operations/calculations.
5. As a user, I want to be able to make arithmetic operations on my data, so that I can make simple mathematical calculations.
6. As a user, I want to be able to make jumps in the program, so that I am able to implement loops and control the flow of the program.
7. As a user I want to be able to halt the program on command, so that my program does not run indefinitely.
8. As a user, I want to receive informative error messages, so that when an error occurs in my program I can react to it effectively.
9. As a developer, I want the program to differentiate between command and data values, so that the program reads word values correctly.
10. As a developer, I want the program to handle incorrect inputs from the user without interrupting the program, so that the user is informed about necessary issues and the program does not crash from user errors.

11. As a developer, I want distinct, well documented classes, so that the program is easy to understand and work with at any point in time.