Software Requirement Specification (SRS) Document

Functional:

1. The user is able to submit a .txt file to the program.
2. The user may submit a different .txt file after the first one has been supplied.
3. Students can execute their machine language programs on the simulator.
4. The user is able to enter input from the keyboard.
5. All the information in the UVSim is handled in terms of words.
6. UVSim is equipped with a 100-word memory.
7. The user can view the register/accumulator.
8. A BasicML program must be loaded into the main memory starting at location 00 before executing.
9. The user may clear the memory of the simulation back to its default.
10. Each instruction written in BasicML occupies one word of the UVSim memory.
11. The program can read, write, load and store in memory locations.
12. The user is able to step through the program, running one instruction at a time.
13. UVSim can handle comments in the program file.
14. The user is able to see the stack or memory locations.
15. The UVSim can interpret a machine language called BasicML.
16. The user is able to run the program, executing each instruction consecutively.

Non-Functional:

1. The program runs with less than 1s delay.
2. The stack has 2 columns, address and value.
3. Buttons must display a shade of blue to indicate that the mouse is successfully hovering over it.

GROUP 1

Functional:

1. Users have the capability to run their machine language scripts on the simulator.
2. The user can browse the local memory.
3. It is understood that the sign of a BasicML instruction is plus sign.
4. Within UVSim, each valid slot in the memory may hold an instruction.
5. The UVSim is equipped with a memory of 100 words.
6. The BasicML program is duly loaded into the primary memory commencing at location 00 prior to execution.
7. The first two digits in each BasicML instruction have the operation code, the intended operation.
8. UVSim is capable of interpreting BasicML.
9. Instructions in BasicML take up a word of the memory.
10. The accumulator is employed in calculations or examined in different ways.
11. UVSim handles all information in a manner with everything dealt with in terms of words.
12. The program can adeptly read, write, load, and store in memory locations.
13. Words within this system are referenced by their location numbers, from 00 to 99.
14. Essential arithmetic operations such as Add, Subtract, Divide, and Multiply are integrated.
15. UVSim's ability to process comments within the program file allows for a more user-friendly experience.

Non-Functional:

1. There are four operational buttons on the GUI.
2. The stack displays all 100 memory slots.
3. The GUI is divided into four separate areas: stack, register, console, and buttons.

GROUP 2

Functional:

1. Users can adjust the position of the iterator as needed.
2. It employs an accumulator for performing calculations or for various inspections.
3. After the initial file has been uploaded, they may submit another .txt file.
4. Each BasicML command occupies a single word in UVSim's memory, and it is assumed that the sign for a BasicML command is always positive.
5. Users can progress through the program incrementally, executing instructions one at a time, or run the program continuously, executing instructions in succession.
6. UVSim manages all data in the form of words and comes with a memory capacity of 100 words.
7. Memory locations in UVSim can each hold a command, with the first two digits of every BasicML command serving as the operation code that dictates the action to be executed.
8. It also allows users to input data via the keyboard.
9. Users have the ability to inspect the stack or memory addresses and can view the contents of the register/accumulator.
10. The program is capable of performing read, write, load, and store operations on memory addresses.
11. UVSim supports the inclusion of comments within the program file.
12. The simulator, known as UVSim, is capable of interpreting a machine language referred to as BasicML.
13. To execute a BasicML program, it must first be loaded into the simulator's main memory.
14. They have the option to reset the simulation's memory to its original state.
15. Learners can run their machine language codes using the simulator.
16. Users have the option to upload a .txt file to the simulator.

Non-Functional:

1. The program operates with a delay of under one second.
2. The stack is organized into two columns: address and value.
3. Arithmetics must be performed in 2s

Individual 1

Functional:

1. The register/accumulator is viewable, providing users with a window into the machine's current state.
2. Input from the keyboard can be entered by the user.
3. Accepts mouse clicks on buttons.
4. The facility to submit a .txt file to the programme is needed.
5. The user is empowered to step through the programme, running one instruction at a time
6. Executing each instruction consecutively enabling users to run the program in a continuous stream.
7. The memory of the simulation may be cleared back to its default state by the user.
8. The ability to manipulate the position of the iterator is provided, offering users control over their navigation through the program.
9. The simulator must show the current position of the program counter.
10. The simulator should include a help section or documentation to aid new users in understanding assembly language and how to use the simulator.
11. Users can add commentary to files.
12. The program is capable of read, write, load, and store operations on memory.
13. Memory locations in UVSim can each hold a command.
14. First two digits of every BasicML command serving as the operation code that dictates the action to be executed.
15. Has an exit so it doesn't overload 100 addresses.

Non-Functional:

1. The interface is divided into four specific areas.
2. Buttons should alter their color intensity upon being pressed to indicate a successful click.
3. The program should feature the ability to halt or pause within three seconds.

Individual 2

Functional:

1. The program supports the submission of a .txt file by the user.
2. Students are enabled to execute their machine language programs using the simulator.
3. The program takes keyboard input entry by the user.
4. All data within UVSim is treated as words.
5. Users can submit a new .txt file once a previous file has been provided.
6. UVSim is designed with a 100-word capacity memory.
7. The user can see the contents of the register or accumulator.
8. BasicML programs must be loaded into UVSim's main memory beginning at location 00 to start execution.
9. Users are granted the functionality to reset the simulator memory to its initial state thus clearing the memory.
10. Instructions in BasicML take up a single word in UVSim's memory.
11. The program is capable of executing read, write, load, and store operations in memory.
12. The user can sequentially process the program, executing one instruction at a time.
13. UVSim can distinguish and handle comments within the program file.
14. The user can inspect the stack or specific memory locations.
15. UVSim is capable of interpreting into machine language.

Non-Functional:

1. Execution of the program occurs with a delay of less than 2 seconds.
2. Buttons are visually designed to change in color when clicked.
3. The stack is presented with 2 columns.

Individual 3

Functional:

1. The program should handle a read and write instruction in the handle.
2. The stack visualizer needs to be user-friendly and sizable.
3. Each memory location can be viewed in the stack.
4. The user can see the contents of the register or accumulator.
5. BasicML programs must be loaded into UVSim's main memory beginning at location 00 to start execution.
6. Users are granted the functionality to reset the simulator memory to its initial state thus clearing the memory.
7. UVSim manages all data in the form of words and comes with a memory capacity of 100 words.
8. Memory locations in UVSim can each hold a command, with the first two digits of every BasicML command serving as the operation code that dictates the action to be executed.
9. It also allows users to input data via the keyboard.
10. UVSim is equipped with a 100-word memory.
11. The user can view the console.
12. BasicML program must be loaded into the main memory starting at location 00 before executing.
13. The user is able to submit a .txt file to the program.
14. The user may submit changed or edited text files.
15. Students can execute their machine language programs on the simulator.

Non-Functional:

1. All 100 memory locations are listed in the stack.
2. There are four functional buttons incorporated into the GUI.
3. The GUI has 4 areas for displaying information/interacting with the simulation.

Individual 4

Functional:

1. The simulator must provide a predefined memory space for program execution.
2. Users must have the option to execute their code one instruction at a time for debugging purposes.
3. There should be functionality to run the entire program from start to finish without halting.
4. Users should have the option to execute the program until a stop input is encountered.
5. The simulator must display the current value of all registers during and after execution.
6. A memory view that shows the content of memory locations is essential.
7. A console for displaying program output, such as print statements and execution results, is required.
8. The simulator needs to have robust error handling to report syntax and runtime errors to the user.
9. Provision to accept user input, both at the beginning and during the execution of the program.
10. The assembler within the simulator should resolve labels and calculate the appropriate addresses.
11. The simulator should support a comprehensive set of assembly instructions.
12. The register/accumulator is viewable, providing users with a window into the machine's current state.
13. Input from the keyboard can be entered by the user.
14. UVSim is capable of interpreting BasicML.
15. Every instruction written in BasicML occupies a single word of the UVSim memory.

Non-Functional:

1. The program includes a feature to halt or pause within 7 seconds.
2. Arithmetic operations within the program are performed in under 0.5 seconds.
3. The stack holds a column for both the address and corresponding value.