1. Each team member (without consulting with other team members) will create their own SRS document containing 15 functional requirements and 3 non-functional requirements.

2. Submit these documents WITHOUT NAMES to the scrum master assigned to this sprint.

3. The scrum master will divide the team into two groups and assign half of the documents to each section.

4. Each sub-team will discuss the documents they've been given and "merge" them into one document. (The merged documents should still have at least 15 functional requirements and 3 non-functional requirements in the end but can have more if you find the submitted requirements distinct and valuable enough apart from each other. Don't just copy and paste one into another, though -- you'll need to discuss how to add/combine/rewrite the contents of both files to create the new one so it is internally consistent without repeats, redundancy, or contradiction between all the requirements.)

5. Each sub-team submits its merged document back to the scrum master.

6. As a team, meet and examine the resulting documents, and create one final SRS document based on your group's discussion and feelings about the results of the previous exercises. (The final document should have at least the same 15/3 content as before after any merging/rewrites, but it can have more. As before, make sure your final document is consistent after the merge with no repeats, redundancy, or contradictions)

7. Submit the final document and the 6-7 working documents created in the earlier steps. (One per team member, plus one per subgroup, in addition to the final document)

Functional Requirements:

1. Load BasicML program: The UVSim should allow users to load a BasicML program into the main memory, starting at location 00, before executing it.
2. Execute BasicML program: The UVSim should interpret and execute the loaded BasicML program step by step.
3. Read input from keyboard: The UVSim should provide instruction to read a word from the keyboard and store it in a specific location in memory.
4. Write output to the screen: The UVSim should provide an instruction to write a word from a specific location in memory to the screen.
5. Load word from memory: The UVSim should provide instructions to load a word from a specific location in memory into the accumulator.
6. Store word to memory: The UVSim should provide instructions to store a word from the accumulator in a specific location in memory.
7. Add word from memory: The UVSim should provide an instruction to add a word from a specific location in memory to the word in the accumulator.
8. Subtract word from memory: The UVSim should provide an instruction to subtract a word from a specific location in memory from the word in the accumulator.
9. Divide the word stored in the accumulator by a word in memory: The UVSim should provide instructions to divide the word stored in the accumulator by a word from a specific location in memory.
10. Multiply word from memory: The UVSim should provide an instruction to multiply a word from a specific location in memory by the word in the accumulator.
11. Branch to a specific memory location: The UVSim should provide an instruction to branch to a specific location in memory.
12. Branch if the accumulator is negative: The UVSim should provide an instruction to branch to a specific location in memory if the accumulator is negative.
13. Branch if the accumulator is zero: The UVSim should provide an instruction to branch to a specific location in memory if the accumulator is zero.
14. Pause program execution: The UVSim should provide an instruction to pause the execution of the program.
15. Memory management: The UVSim should manage the main memory of 100 words, ensuring proper allocation and deallocation of memory for instructions and data.

Non-Functional Requirements:

1. Performance: The UVSim should execute BasicML programs efficiently, with minimal processing and response times.
2. Usability: The UVSim should have a user-friendly interface, allowing students to interact with the simulator easily and understand the execution of BasicML programs.
3. Reliability: The UVSim should be reliable, providing accurate results and handling errors or invalid instructions gracefully, with appropriate error messages and behavior.