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. UVSim should provide instructions to read a word from the keyboard and store it in memory.
2. UVSim should provide an instruction to write a word from a specific location in memory to the screen.
3. UVSim should provide instructions to load a word from a specific location in memory into the accumulator.
4. UVSim should provide instructions to store a word from the accumulator in a specific location in memory.
5. UVSim should provide instructions to add a word from a specific location in memory to the word in the accumulator.
6. UVSim should provide instructions to subtract a word from a specific location in memory from the word in the accumulator.
7. UVSim should provide instructions to divide the word stored in the accumulator by a word from a specific location in memory.
8. UVSim should multiply a word from a specific location in memory by the word in the accumulator.
9. UVSim should manage the main memory of 100 words, ensuring proper allocation and deallocation of memory for instructions and data.
10. UVSim will use a file picker pop-up in order to select the file they would like to run in the program and retrieve input numbers.
11. UVSim should provide an instruction to branch to a specific location in memory normally and in specific cases of negative or zero values.
12. UVSim will have a GUI interface that will have two options for the user to input a word when prompted. Either using a visual 10-keypad and mouse or just tying the word with their keyboard into an input box.
13. UVSim should provide a way for the user to run, pause and quit the program.
14. UVSim should keep track of the accumulator values, memory locations, and positive/negative signs in the code.
15. UVSim should correctly read the user’s file and dissect BasicML code.
16. UVSim should accept numbers without signs and provide an assumed positive sign.
17. The program will output any errors during the run of the program to the user's GUI so that they can be aware of what failed.

Non-Functional Requirements:

1. The UVSim should have a user-friendly, colorblind friendly interface, allowing students to interact with the simulator easily; including separate information sections, a display of the accumulator value, and provide test files for the user.
2. The UVSim should execute BasicML programs efficiently, with minimal processing and response times on Windows and Apple operating systems.
3. The UVSim should be reliable, providing accurate results and gracefully handling errors or invalid instructions, with appropriate error messages and behavior.