**UVSim – SRS**

1. **Introduction**

* **System Name:** UVSim
* **Purpose:** UVSim is a simulator that executes programs written in BasicML. It helps students learn computer architecture by running simple instructions on a virtual CPU with 100-word memory and an accumulator.
* **Scope:** UVSim supports I/O, arithmetic, load/store, and control instructions. It can run from the command line or a GUI, validate input, and display outputs.

1. **Functional Requirements**

Each requirement must be written in “The system shall…” form, one idea per line.

Here are 15 sample functional requirements aligned with your code and README. Each teammate can tweak/add their own wording:

1. The system shall load a BasicML program into memory starting at address 00.
2. The system shall execute instructions sequentially until a HALT instruction is reached.
3. The system shall store user input into a specified memory location when a READ instruction is executed.
4. The system shall output the value stored in a specified memory location when a WRITE instruction is executed.
5. The system shall load a value from memory into the accumulator when a LOAD instruction is executed.
6. The system shall store the value in the accumulator into a memory location when a STORE instruction is executed.
7. The system shall add a memory value to the accumulator when an ADD instruction is executed.
8. The system shall subtract a memory value from the accumulator when a SUBTRACT instruction is executed.
9. The system shall multiply the accumulator by a memory value when a MULTIPLY instruction is executed.
10. The system shall divide the accumulator by a memory value when a DIVIDE instruction is executed, unless the divisor is zero.
11. The system shall truncate arithmetic results to the last 4 digits (e.g., 12345 → 2345).
12. The system shall update the instruction pointer according to BRANCH, BRANCHNEG, and BRANCHZERO instructions.
13. The system shall stop execution when a HALT instruction is encountered.
14. The system shall validate all input program lines as signed 4-digit numbers.
15. The system shall display an error if an invalid opcode or malformed file is detected.
16. **Non-Functional Requirements**

At least 3 are required. Here are examples you can use or adapt:

* **Usability:** The system shall provide clear error messages in the console or GUI for invalid inputs or operations.
* **Performance:** The system shall execute a 100-instruction program in under 1 second on a modern computer.
* **Portability:** The system shall run on Windows, macOS, and Linux with Python 3.8 or newer.

1. **Notes for Submission**

We each pick the tasks we can do, and we make sure no one’s work gets missed because another teammate is covering it. Our team communicates well—we keep each other updated during the week and ask everyone to review the tasks we’ve done. This helps us make sure everything matches the requirements for Milestone 3 and also follows the professor’s feedback so we can improve our paper.