FUNCTIONAL REQUIREMENTS DOC 1:

1. The program needs to load a basicML program file into memory
2. Which then is split into the 4 digit program words
3. The words are stored in 100 memory locations
4. The program will make sure the memory instruction exists and operand is 100 or less
5. The system will load the words starting at address 00
6. The program will then fetch the operations in order.
7. The first 2 digits in the word will be the operation code
8. The last 2 digits in the word will be the operand code
9. The program needs to handle READ which takes input from the user from the GUI
10. The program will also need to WRITE any outputs in the GUI
11. The accumulator will only have one word
12. LOAD and STORE will move information between memory and accumulator
13. The system will implement ADD, SUBTRACT, DIVIDE, and MULTIPLY
14. The system will have control operations like BRANCH, BRANCHNEG, BRANCHZERO, and HALT
15. The system will terminate execution when HALT is called

NonFunctional Requirements

1. The return should take less than 500 milliseconds to complete any file under 100 instructions
2. The program will have a simple and consistent GUI
3. The program will handle any errors gracefully

FUNCTIONAL REQUIREMENTS DOC 2:

1. The system needs to have a graphical user interface or GUI for all user interactions instead of a simple command line interface.

2. Also the system needs to allow users to select and load a correctly formatted BasicML program file from their local machine.

3. Also the system should validate that each line of the input file must have exactly four digits.

4. The system needs to parse each instruction into an opcode or first two digits and operand or last two digits.

5. The system needs to display the loaded program instructions within the GUI.

6. Also the system needs to allow the user to start program execution using simply the “Run” button.

7. The system need to allow the user to execute the program one instruction at a time using a “Step” button

8. Display the current value of the accumulation during execution.

9. Display contents of memory around 100 memory locations.

10. Need to support the following instructions: READ, WRITE, LOAD, STORE, ADD, SUBTRACT, DIVIDE, MULTIPLY, BRANCH, BRANCHNEG, BRANCHZERO, HALT and etc.

11. Display program output in a designated output are within the GUI

12. Display error messages when invalid opcodes or memory addresses are encountered

13. Prevent execution if no program file has been loaded

14. Allow user to reset the simulator to its original state

15. Notify users when program execution has been done.

Non-Functional Requirements:

1. Usability where the graphical interface of the user needs to have clearly labeled buttons, very organized layout sections, and constants to ensure easy use.

2. Performance where the system needs to execute instructions and update the GUI without a big waiting time.

3. Also systems need to maintain division between the GUI layers and the simulator logic to make sure of maintenance.

FUNCTIONAL REQUIREMENTS DOC 3:

Functional Requirements:

1.) The system should allow the user to input values again if they don't exist.

2.) The program should have a GUI that has an option to close the program.

3.) The program should stay running until the user quits the program.

4.) The program should keep track and display the actions the user takes.

5.) The program should have an instruction page of all the things the program can do.

6.) The system should have log in page with usernames and passwords.

7.) The program should be able to keep the value in its memory to reuse after its been halted.

8.) The program should have the ability to reset the entire simulator.

9.) The program should be able to do exponent mathematics

10.) The program should have the ability to ask the user if they want to input there operations from a file.

11.) The program should keep a total of the number of operations a user has done.

12.) If the user puts an incorrect operation or value the system will display an error message.

13.) The program should have a button that allows for users to input feedback they have.

14.) The GUI should display the name of the user logged in.

15.) The program should write the outputs of the simulator to a text file.

Non-Functional Requirements:

1.) When the system memory gets full it should be reset.

2.) The program should have as little bugs as possible.

3.) The programs GUI should be simple for anyone to understand.

DOC 4:

Merge 1:

1.) The program will have GUI for user interactions.

2.) The program will stop when HALT is called.

3.) The system will validate that inputs are only 4 digits long.

4.) The system will display an error message when an invalid input has been make.

5.) The program will handle READ(which takes inputs from the user), and WRITE(which outputs anything stored in the memory to the screen) within the GUI.

6.) The system will display the instructions that the system has to offer.

7.) There should be a QUIT button in the GUI.

8.) If there is no input file, the program will not work.

9.) The system will have operations like BRANCH, BRANCHNEG, BRANCHZERO, and HALT.

10.) The program should have a feature that allows for user feedback.

11.) The system will keep track of all the inputs that have been done.

12.) The system will check if the input file is formatted correctly.

13.) The system will execute the operations in the input files from top to bottom.

14.) The system needs to parse each instruction into an opcode or first two digits and operand or last two digits.

15.) The system will give you the option it input values again if they aren't valid.

Non-Functional Requirements:
1.) Usability where the graphical interface of the user needs to have clearly labeled buttons, very organized layout sections, and constants to ensure easy use.

2.) Performance where the system needs to execute instructions and update the GUI without a big waiting time.

3.) The return should take less than 500 milliseconds to complete any file under 100 instructions

DOC 5:

Merge 2:

1. The program needs to load a basicML program file into memory
2. Which then is split into the 4 digit program words
3. The words are stored in 100 memory locations
4. The program will make sure the memory instruction exists and operand is 100 or less
5. The system will load the words starting at address 00
6. The accumulator will only have one word
7. LOAD and STORE will move information between memory and accumulator
8. The system will implement ADD, SUBTRACT, DIVIDE, and MULTIPLY
9. Also the system needs to allow the user to start program execution using simply the “Run” button.
10. The system need to allow the user to execute the program one instruction at a time using a “Step” button
11. Display the current value of the accumulation during execution.
12. Display contents of memory around 100 memory locations.
13. Need to support the following instructions: READ, WRITE, LOAD, STORE, ADD, SUBTRACT, DIVIDE, MULTIPLY, BRANCH, BRANCHNEG, BRANCHZERO, HALT and etc.
14. Allow user to reset the simulator to its original state
15. Notify users when program execution has been done.

Non-Functional Requirements:

1. Also systems need to maintain division between the GUI layers and the simulator logic to make sure of maintenance.
2. The program will have a simple and consistent GUI
3. The program will handle any errors gracefully

DOC 6:

Final Merge:

Functional Requirements

1.) The program will tell you when the execution has finished.

2.) The program will allow you to reset it to it's original state.

3.) The system will implement ADD, SUBTRACT, DIVIDE, and MULTIPLY.

4.) The program will have a GUI for user interactions.

5.) The program will have Quit button in the GUI.

6.) The system will be able to use operations like BRANCH, BRANCHNEG, BRANCHZERO, and HALT.

7.) The program will be able to display what is inside all of the memory contents.

8.) The program will have a 'run' button which cause the program to run.

9.) The program will display an error message when an invalid input has been made.

10.) The system need to allow the user to execute the program one instruction at a time using a "Step" button

11.) The program will validate that inputs are only 4 digits long.

12.) The program will allow you to reinput values if they aren't valid.

13.) The program will display all of the operators the program can do.

14.) There will only be one accumulator.

15.) The system needs to parse each instruction into an opcode or first two digits and operand or last two digits.

Non-Functional Requirements:

1.) Usability where the graphical interface of the user needs to have clearly labeled buttons, very organized layout sections, and constants to ensure easy use.

2.) Also systems need to maintain division between the GUI layers and the simulator logic to make sure of maintenance.

3.) The return should take less than 500 milliseconds to complete any file under 100 instructions