**Design Document**

**User Stories**

1. As an educational client, I want the software to be easy to use, so that the students who are using it can more easily learn from it and will want to use my product.
2. As a computer science student, I want to be able to know why my BasicML isn’t working if there’s an error, so that I can still use the program and see at least some of my code working.

**Use Cases**

**Read**:

System will read a word from the keyboard into a location in memory.

1. Prompt the user for input.
2. If the user enters “Q” or “q”, exit the program.
3. System will verify that the input starts with + or -.
4. System will verify that the input is 5 characters long.
5. System will verify that the characters after the +/- are all integers.
6. Add the input into memory.

**Write:**

System will write a word from a location in memory to the screen.

1. System gets a word from given location in memory.
2. System will verify the word is a string.
3. System will verify the word is 5 characters long.
4. System will verify that the word starts with + or -.
5. System will verify that the characters after the +/- are all integers.
6. System will print the word to the user’s screen.

**Load**:

System will load a word from a location in memory into the accumulator.

1. System will retrieve a word from memory.
2. System will verify the word is a string.
3. System will verify the word is 5 characters long.
4. System will verify that the word starts with + or -.
5. System will verify that the characters after the +/- are all integers.
6. System will load this word into the accumulator.

**Store**:

System will store a word from the accumulator into a location in memory.

1. System will retrieve a word from the accumulator.
2. System will verify the word is a string.
3. System will verify the word is 5 characters long.
4. System will verify that the word starts with + or -.
5. System will verify that the characters after the +/- are all integers.
6. System will store the word from the accumulator into the specified location in memory.

**Add**:

System will add a word from a location in memory to the word in the accumulator.

1. System will retrieve a value from the accumulator.
2. System will parse the accumulator value into a integer.
3. System will retrieve a value from memory at the given location.
4. System will parse the memory value into a integer.
5. System will add the accumulator value to the value found at the given memory location.
6. System will make sure there is no overflow.
7. System will reformat the sum to be in the +/-#### format
8. System will update the accumulator with the formatted sum.

**Subtract**:

System will subtract a word from a location in memory from the word in the accumulator.

1. System will retrieve a value from the accumulator.
2. System will parse the accumulator value into a integer.
3. System will retrieve a value from memory at the given location.
4. System will parse the memory value into a integer.
5. System will subtract the stored value from the accumulator value.
6. System will make sure there is no overflow.
7. System will reformat the difference to be in the +/-#### format
8. System will update the accumulator with the formatted difference.

**Multiply**:

System will multiply the word in the accumulator and the word from a location in memory.

1. System will retrieve a value from the accumulator.
2. System will parse the accumulator value into a integer.
3. System will retrieve a value from memory at the given location.
4. System will parse the memory value into a integer.
5. System will multiply the accumulator value and the value found in memory.
6. System will make sure there is no overflow.
7. System will reformat the product to be in the +/-#### format
8. System will update the accumulator with the formatted product.

**Divide**:

System will divide the word in the accumulator by the word from a location in memory.

1. System will retrieve a value from the accumulator.
2. System will parse the accumulator value into a integer.
3. System will retrieve a value from memory at the given location.
4. System will parse the memory value into a integer.
5. System will divide the accumulator value by the value found in memory.
6. System will make sure there is no overflow.
7. System will reformat the quotient to be in the +/-#### format
8. System will update the accumulator with the formatted quotient.

**Branch**:

System will branch to a location in memory.

1. System will check that the location given to branch to isnt the current location.
2. System will update the PC location to the given location.

**BranchNeg**:

System will branch to a location in memory if the accumulator is negative.

1. System will check that the location given to branch to isnt the current location.
2. System will check that the word value starts with “-”.
3. System will update the PC location to the given location.

**BranchZero**:

System will branch to a location in memory if the accumulator is zero.

1. System will check that the location given to branch to isnt the current location.
2. System will check that the word value is “+0000” or “-0000”.
3. System will update the PC location to the given location.

**Halt**:

System will pause the program when Halt is called.

1. System will detect the halt command.
2. The program will be exited.

**No\_Infinite\_Branching**:

System will provide errors for infinite loops in branching.

1. System will take in the branch location and the current location
2. If the two locations are different, then there is no infinite branching
3. If the two locations are the same, then the program will print an error then halt

**Check\_No\_Overflow**:

System will provide errors for overflow.

1. System reads in the accumulator value as an integer.
2. If the number is more than 9999 or less than -9999, the program will Halt.
3. Otherwise, the function returns true and the program will continue.

**Main/Get\_Value:**

System will ensure that only inputs that are in the (+/-)#### format are accepted.

1. System gets a value from memory.
2. System checks that the value is 5 characters long.
3. System checks that the first symbol is a + or -.
4. System checks for an operation by getting the contents at its location using Get\_Value.
5. Get\_Value returns the value of a given key.
6. If there is a KeyError, Get\_Value returns an appropriate error message.

**Allocate\_Memory**:

System will allocate 99 locations in memory.

1. System creates a dictionary called MEM.
2. System populates the dictionary with 100 instances of “+0000” and numbers them 00 through 99
3. The program then takes the file contents and writes them into memory, replacing the filler content as needed.