**Non-Class Functions 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 is there’s an error so that I can still use the program and see at least some of my code working.

**Use Cases**

**Read**:

UVSim 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. UVSim will verify that the input is 6 or 7 characters long.
4. If it is 7 characters long, UVSim will verify that the input starts with + or -.
5. If it is 7 characters long, UVSim will verify that the characters after the +/- are all integers.
6. If it is 6 characters long, UVSim will verify that the characters are all integers.
7. UVSim will check that the user input is not empty before continuing.
8. If any of these tests fail, UVSim will output the corresponding error message and then prompt the user to enter a new input value until they enter a correct one or quit.
9. Upon correct entry, the user input will be read into the specified location in memory.

**Write:**

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

1. UVSim gets a word from a given location in memory.
2. UVSim will verify the word is a valid BasicML word using the Memory class’s validate\_value function.
3. If it is a valid BasicML word, UVSim will print the word to the user’s screen and the function will return true.
4. If the word is not a valid BasicML word, UVSim will Halt.

**Load**:

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

1. UVSim will retrieve a word from memory.
2. UVSim will verify the word is a valid BasicML word using the Memory class’s validate\_value function.
3. UVSim will load this word into the accumulator.

**Store**:

UVSim will store a word from the accumulator in a location in memory.

1. UVSim will retrieve a word from the accumulator.
2. UVSim will verify the word is a valid BasicML word using the Memory class’s validate\_value function.
3. UVSim will store the word from the accumulator in the specified location in memory.

**Add**:

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

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

**Subtract**:

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

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

**Multiply**:

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

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

**Divide**:

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

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

**Check\_No\_Overflow**:

UVSim will provide errors for overflow.

1. UVSim reads in the accumulator value as an integer.
2. If the number is more than 999999 or less than -999999, the program will Halt.
3. Otherwise, the function returns true and the program will continue.

**Branch**:

UVSim will branch to a location in memory.

1. UVSim will check that the location for branching to isn't the current location.
2. UVSim will update the PC location to the given location.

**BranchNeg**:

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

1. UVSim will check that the location for branching to isn't the current location.
2. UVSim will check that the word value starts with “-”.
3. UVSim will update the PC location to the given location.

**BranchZero**:

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

1. UVSim will check that the location for branching to isn't the current location.
2. UVSim will check that the word value is “+000000” or “-000000”.
3. UVSim will update the PC location to the given location.

**Halt**:

UVSim will pause the program when Halt is called.

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

**No\_Infinite\_Branching**:

UVSim will provide errors for infinite loops in branching.

1. UVSim 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\_Branch\_In\_Range:**

UVSim will take determine if the branch destination is within range of the memory.

1. UVSim takes the branch destination value and the memory as arguments.
2. UVSim will determine if the branch destination is within range of the memory size.
3. The function returns true if the branch destination is in range and halts if it is not.

**Main:**

UVSim will read and execute the BasicML file step by step.

1. UVSim gets the memory containing the file and the window that is currently active.
2. UVSim checks that the current position is within the range of the file size.
3. UVSim sets the current location in the file to a variable called pc\_location.
4. If the value at the current location is 0, UVSim increments the location and begins to check the next line in the BasicML code.
5. If the value at the current location isn’t zero, UVSim checks for and executes an operation in correspondence to the memory value at the current location.
6. If there is a branch, UVSim updates the position and repeats the cycle based on the newly set position.
7. If UVSim detects a HALT command in the BasicML code, it will stop analyzing any remaining BasicML code.

**Allocate\_Memory**:

UVSim will allocate 250 locations in memory.

1. UVSim creates a dictionary called MEM.
2. UVSim populates the dictionary with 250 instances of “+000000” and numbers them from 0 through 249.
3. The program then takes the file contents and writes them into memory, replacing the filler content as needed.

**Check\_for\_Operation:**

UVSim will interpret a line in the code to determine and run the specified operation.

1. UVSim checks the first 3 characters to see if they match an operation’s ID.
2. UVSim executes the operation using the remaining numbers given in the line.
3. UVSim will output messages to the console upon completing any arithmetic or IO operations.
4. UVSim will skip over any lines that are not actual BasicML codes.

**is\_valid\_hex:**

UVSim will verify that the input given by the user in the color picker is in hex format.

1. UVSim checks that the color picker input is 6 characters in length.
2. UVSim verifies that the input can be converted to a hex value.
3. UVSim will output an error message if the input given is not a valid hex value.