

UVSim Design Document

CS 2450 - Group C

Updated Apr. 8, 2023

Introduction

The UVSim is a program to simulate running a low-level computer using basic machine language instructions. The instructions will be written in a type of machine language called BasicML. The user will be able to load BasicML instructions from text files into the machine and then the machine will run the instructions. The UVSim will support basic, but powerful instruction types that will allow for reading and writing data to memory, getting input and sending output, performing basic arithmetic operations, and controlling the program flow. The BasicML instructions will be stored as six-digit words prepended either by a plus or minus sign. Words can either be instructions or simple data for instructions to manipulate. The intended users for UVSim will be individuals in education. Two examples of individuals who might use the UVSim are teachers and students. Both types of individuals will use the UVSim for specific purposes and two user stories will be used to illustrate those purposes.

User Stories

User Story 1:

“As a teacher, I want to be able to run programs in the UVSim that my students have written as text files so that I can find out where to focus my teaching and improve their learning.”

User Story 2:

“As a student, I want to be able to write a program using BasicML, load that program into memory, and run that program so that I can learn about how computer memory and low-level programs work.”

Use Cases

Use Case 1:

Program reaches a READ instruction:

Goal: Read a word input from the user's keyboard into a specific location in memory. Process:

1. Program calls a function to grab a valid keyboard input.
2. The function validates the input to ensure it has exactly one sign and is six digits long.

- a. If validation is successful, program writes the user input input directly into the memory address given in the READ instruction.
 - b. If validation is unsuccessful the function asks the user to input a valid instruction again until the validation is successful.
3. After the word is stored in memory, the program has accomplished its goal and it continues to the next instruction.

Use Case 2:

Program reaches a WRITE instruction:

Goal: Write a word from a specific location in memory to the screen.

Process:

1. Program accesses memory and obtains the word at the memory address specified in the instruction.
2. Program outputs that word directly to the screen.
3. Once the word is output to the screen, the program has accomplished its goal and it continues to the next instruction.

Use Case 3:

Program reaches a LOAD instruction:

Goal: Load a word from memory into the program's accumulator.

Process:

1. Program accesses the data value at the memory location specified in the instruction.
2. Program stores that data value in the UVSIM object's accumulator.
3. Once the accumulator has been set, the program has accomplished its goal and it continues to the next instruction.

Use Case 4:

Program reaches a STORE instruction:

Goal: Store a word from the accumulator into a specific location in memory. Process:

1. Program accesses its accumulator and retrieves the word stored there.
2. Program takes the word from the accumulator and stores it at the memory address specified by the STORE instruction.
3. Once the word is stored, the program has accomplished its goal and it continues to the next instruction.

Use Case 5:

Program reaches an ADD instruction:

Goal: Add a word from a specific location in memory to the word in the accumulator and store

the result in the accumulator.

Process:

1. Program retrieves the word at the location in memory specified by the instruction. 2. Program retrieves the word in the accumulator and adds that to the word from memory. A) If the result is greater than six digits, the program will produce an error. B) If the value can be contained in six digits, the program will store the result in the accumulator.
3. Once the result is stored in the accumulator, the program has accomplished its goal and it continues to the next instruction.

Use Case 6:

Program reaches a SUBTRACT instruction:

Goal: Subtract a word from a specific location in memory from the word in the accumulator and store the result in the accumulator.

Process:

1. Program retrieves the word at the location in memory specified by the instruction.
2. Program subtracts that word from the word in the accumulator.
 - A) If the result is greater than six digits, the program will produce an error. B) If the value can be contained in six digits, the program will store the result in the accumulator.
3. Once the result is stored in the accumulator, the program has accomplished its goal and it continues to the next instruction.

Use Case 7:

Program reaches a DIVIDE instruction:

Goal: Divide the word in the accumulator by a word in memory and store the result in the accumulator.

Process:

1. Program retrieves the word at the location in memory specified by the instruction. 2. Program divides the value of the word in the accumulator by the value of the word from memory.
 - A) If the result is greater than six digits, the program will produce an error. B) If the value can be contained in six digits, the program will store the result in the accumulator.
3. Once the result is stored in the accumulator, the program has accomplished its goal and it continues to the next instruction.

Use Case 8:

Program reaches a MULTIPLY instruction:

Goal: Multiply the word in the accumulator and a word in memory and store the result in the accumulator.

Process:

1. Program retrieves the word at the location in memory specified by the instruction.
2. Program multiplies the value of the word in the accumulator and the value of the word from memory.

A) If the result is greater than six digits, the program will produce an error. B) If the value can be contained in six digits, the program will store the result in the accumulator.

3. Once the result is stored in the accumulator, the program has accomplished its goal and it continues to the next instruction.

Use Case 9:

Program reaches a BRANCH instruction:

Goal: Set the value of its program counter to the memory address specified by the instruction and then continue executing instructions from that new memory address. Process:

1. The program will have a function called tick that will check the value of the program counter and send the word at the memory location specified by the program counter to another function that will interpret it.
2. To accomplish its goal of branching, the program will access its program counter and set the program counter to the value specified by the BRANCH instruction.
3. Once the program counter is set, the program has accomplished its goal and its tick function will access the program counter at the new memory location and start executing instructions from there.

Use Case 10:

Program reaches a BRANCHNEG instruction:

Goal: Set the value of its program counter to the memory address specified by the instruction, if the accumulator contains a negative number and then continue executing instructions from that new memory address.

Process:

1. The program will have a function called tick that will check the value of the program counter and send the word at the memory location specified by the program counter to another function that will interpret it.
2. To accomplish its goal of branching, the program will access the accumulator and determine if the accumulator's value is negative.

A) If it is not negative, the program will continue to the next instruction.

B) If the accumulator is negative, the program will set the program counter to the value specified by the BRANCHNEG instruction.

3. Once the program counter is set, the program has accomplished its goal and its tick function will access the program counter at the new memory location and start executing instructions from there.

Use Case 11:

Program reaches a BRANCHZERO instruction:

Goal: Set the value of its program counter to the memory address specified by the instruction, if the accumulator contains zero. The program will then continue executing instructions from that new memory address.

Process:

1. The program will have a function called tick that will check the value of the program counter and send the word at the memory location specified by the program counter to another function that will interpret it.
2. To accomplish its goal of branching, the program will access the accumulator and determine if the accumulator's value is zero.
 - A) If it is not zero, the program will continue to the next instruction.
 - B) If the accumulator is zero, the program will set the program counter to the value specified by the BRANCHZERO instruction.
3. Once the program counter is set, the program has accomplished its goal and its tick function will access the program counter at the new memory location and start executing instructions from there.

Use Case 12:

Program reaches a HALT instruction:

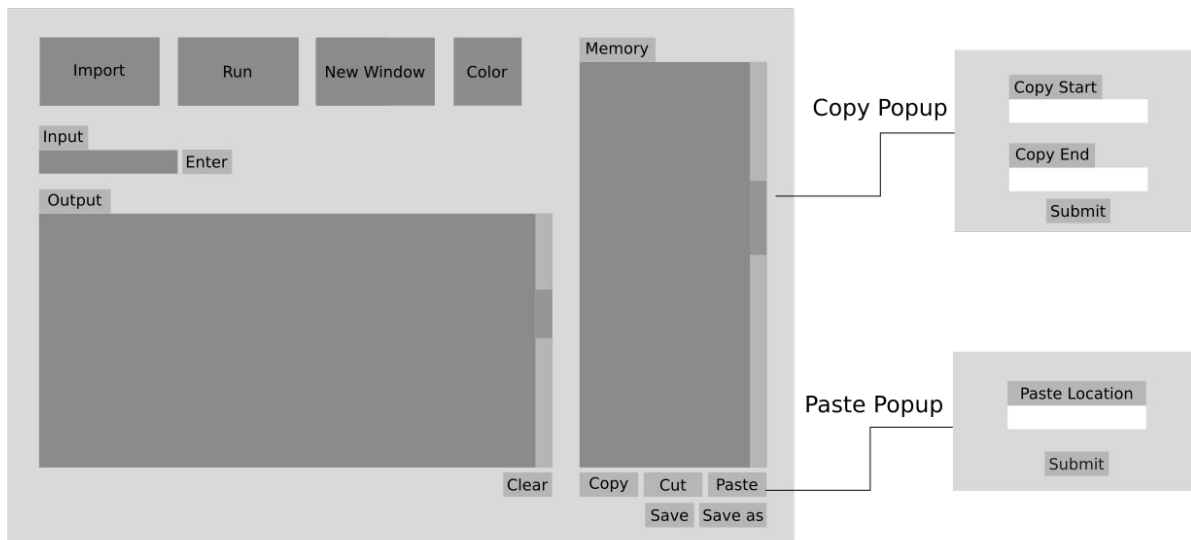
Goal: Stop program execution.

Process:

1. The program will have a function called tick that will check the value of the program counter and send the word at the memory location specified by the program counter to another function that will interpret it.
2. When the program is running, it checks a boolean value to determine if it should keep running.
 - A) If the boolean is true, the program calls its tick function.
 - B) If the boolean value is false, the program stops.
3. When executing a HALT instruction, the program will set its running boolean to false which will stop program execution.
4. Once program execution has ceased, the program has accomplished its goal.

Program Design

Wireframe:



Wireframe Description:

Element	Function
Memory Display	Show the contents of the UVSim's memory. Section will be scrollable so the user can see all the contents of memory.
Import Button	Opens up a file picker window when clicked. Allows the user to load a program into memory.
Run Button	Creates a UVSim machine and immediately runs the program in the memory display.
New Window Button	Creates a new UVSim window so that more than one program can be edited at a time by the user.
Color Button	Allows the user to customize the UVSim experience by changing theme colors.
Input Text Field	Text field for the user to input a word that the UVSim can store.
Output Text Field	Text field that displays the output from the UVSim. The user cannot interact with it directly.

Copy / Cut / Paste Buttons	Opens relevant windows that allow the user to manipulate ranges of the program.
Save / Save-as Buttons	Allows the user to save the currently open program as a file that can be opened again by UVSim later.