Design Documentation

Designed and built by: Branson Petty, Connor Barry, Noah Potter, and Pedro Valente.

The purpose of this program is to create a functional assembly program that executes 6 bit instructions loaded from a file line by line or entered in the input box built-in the program. The program executes the instructions (known as words) in each register until the halt instruction is reached or the all of the registers have been executed.

The program was designed to only run .txt files other than entering instructions directly to the built-in input box. This decision was made to assure maximum accuracy and to prevent the user from crashing the program by running a file type that the program wouldn't be able to read accurately.

A past version of this program was made supporting 4 bit instructions, but it was decided to change the instruction size to 6 bits to allow for a larger register memory. The new version of the program will be able to convert the old 4 bit instructions to 6 bit instructions. Once the instructions are converted and successfully loaded, the program will be able to run them as if they were 6 bit instructions, and any further alterations to the instructions will be done to the newly converted 6 bit instructions.

A few things to keep in mind while using the program:

- You must use a halt instruction at the end of the program routine you mean to run. That will prevent unwanted instructions from being performed. If no halt instruction is present the program will run every register until it reaches register 249.
- If an invalid instruction is entered an error message will be displayed showing the location of the bad instruction, then the program will be automatically halted.
- Any empty register (+000000) will be skipped by the compiler until the end is reached.
- After you done running a program file, you will be given the option of running another one.
- Any time you open a new file all of the memory will be reset in order to load the new instructions.

How to run the program:

You may run the program by opening the command shell in the directory containing all of the python scripts, then running the command bellow:

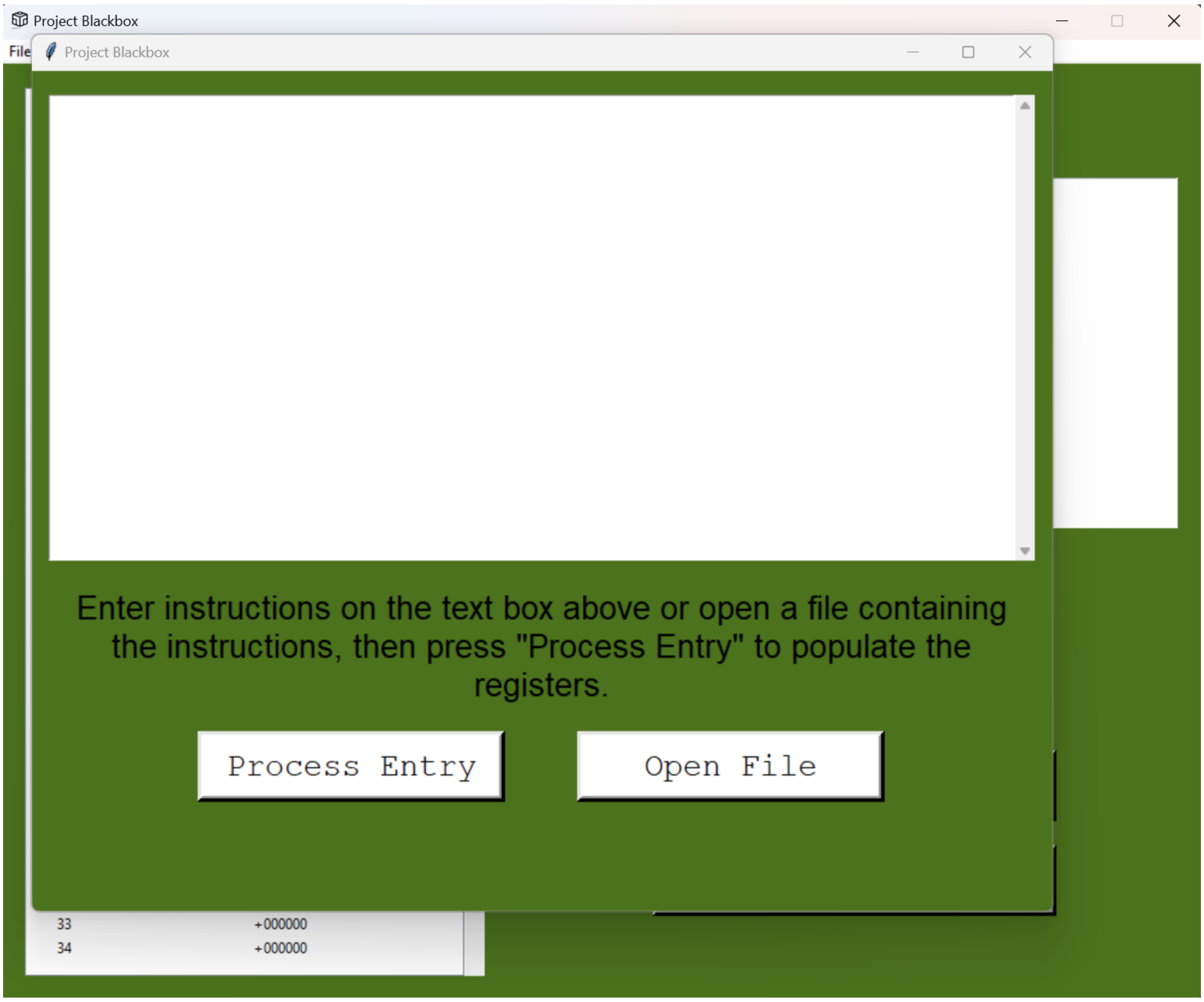
```
python gui_app.py
```

Once you execute the command above, a GUI will be open to the screen that can be used to execute any amount of BasicML scripts.

You can load the instructions buy pressing the "Load Instruction" button as shown bellow or in the "File" menu on the top of the program:

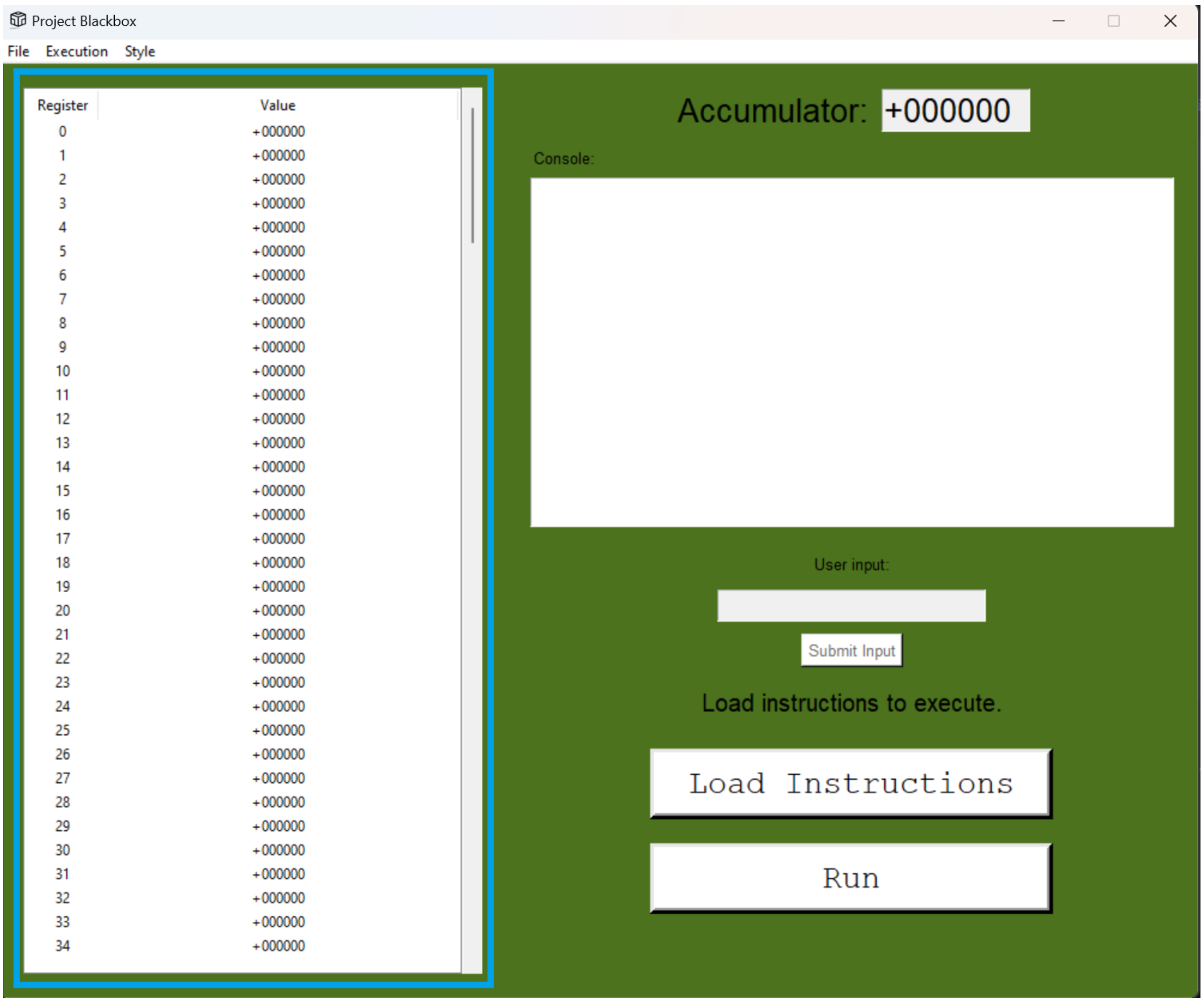
Load Instruction(Button)

Once you select "Load Instructions", a new window will appear as shown bellow. In this new window you may load the instructions from a file or enter them manually buy entering all of the instructions in the text box in the window. If you decide to open a file, you will be given the change to modify the instructions before loading them to the registers by pressing "Process Entry":

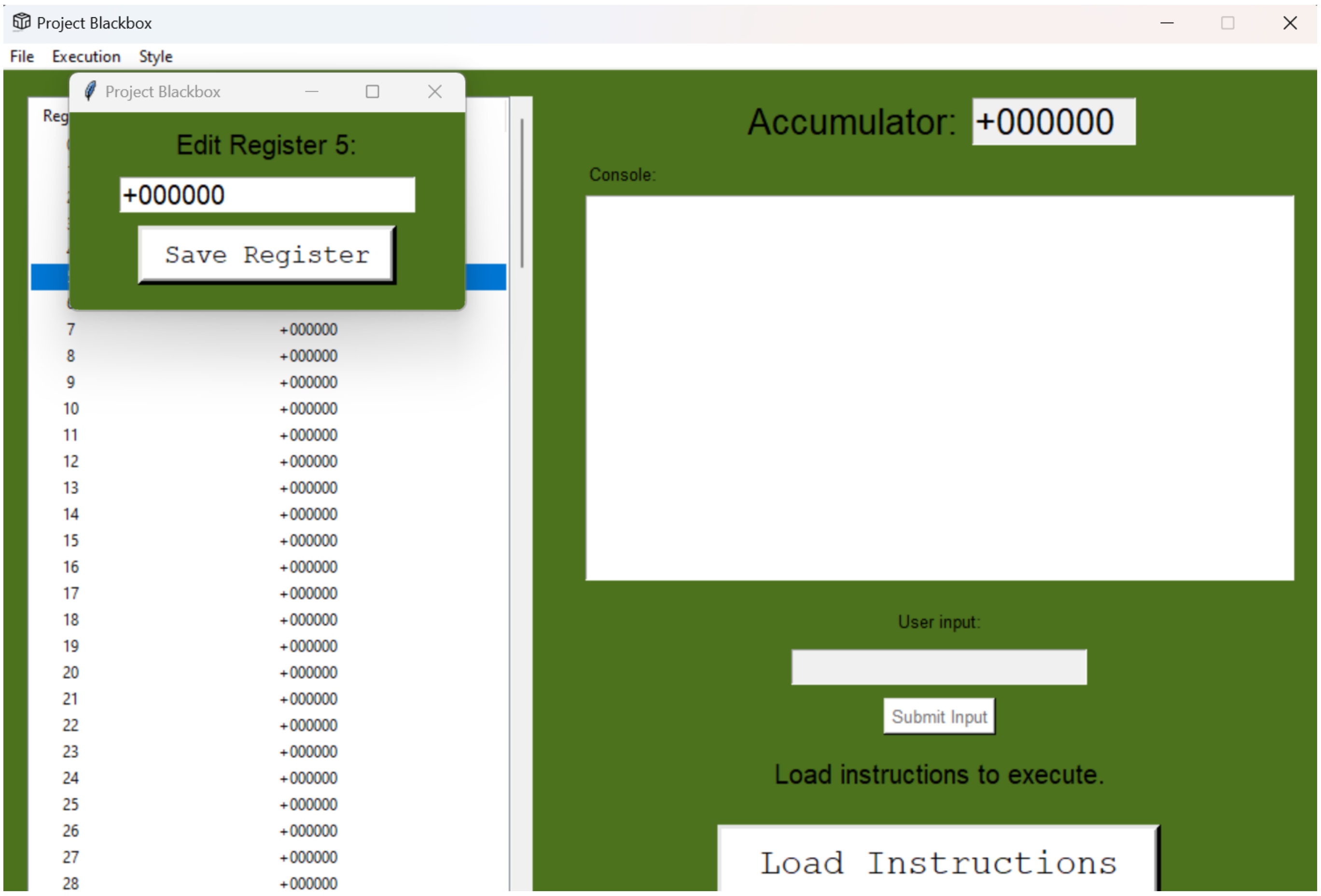


Enter instructions on the text box above or open a file containing the instructions, then press "Process Entry" to populate the registers.

Once the instructions successfully loaded, they will be displayed the table shown in the first image bellow. At this point, you may double click any of the register entry in the table to edit it. Once you double click it, the window shown in the second image bellow will appear and you may enter the new value to be stored in the register:



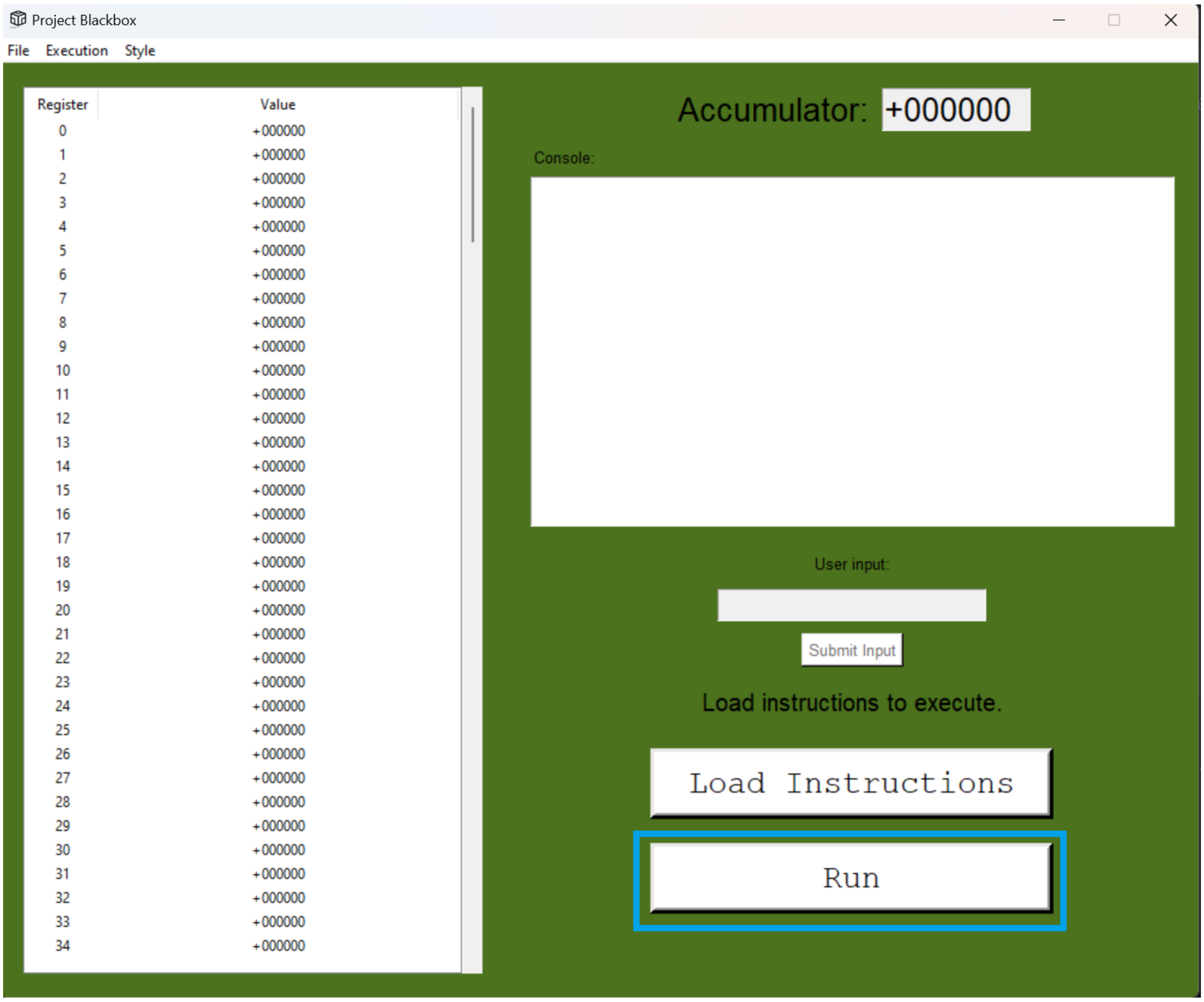
Register	Value
0	+000000
1	+000000
2	+000000
3	+000000
4	+000000
5	+000000
6	+000000
7	+000000
8	+000000
9	+000000
10	+000000
11	+000000
12	+000000
13	+000000
14	+000000
15	+000000
16	+000000
17	+000000
18	+000000
19	+000000
20	+000000
21	+000000
22	+000000
23	+000000
24	+000000
25	+000000
26	+000000
27	+000000
28	+000000
29	+000000
30	+000000
31	+000000
32	+000000
33	+000000
34	+000000



Reg	
7	+000000
8	+000000
9	+000000
10	+000000
11	+000000
12	+000000
13	+000000
14	+000000
15	+000000
16	+000000
17	+000000
18	+000000
19	+000000
20	+000000
21	+000000
22	+000000
23	+000000
24	+000000
25	+000000
26	+000000
27	+000000
28	+000000

29	+000000
30	+000000
31	+000000
32	+000000
33	+000000
34	+000000

Once the registers are loaded as desired you may execute the instructions by pressing the Run button as shown bellow:

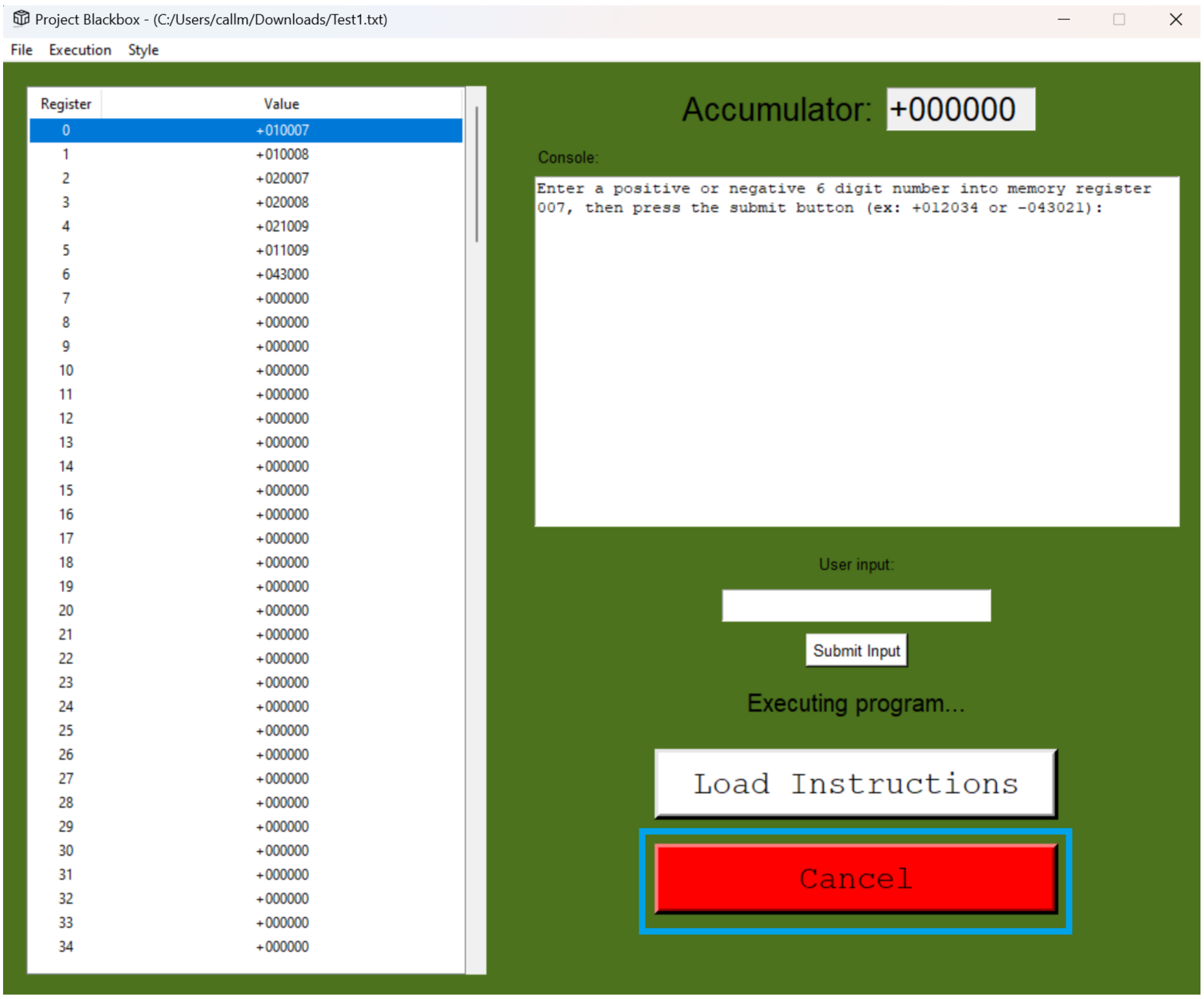


Register	Value
0	+000000
1	+000000
2	+000000
3	+000000
4	+000000
5	+000000
6	+000000
7	+000000
8	+000000
9	+000000
10	+000000
11	+000000
12	+000000
13	+000000
14	+000000
15	+000000
16	+000000
17	+000000
18	+000000
19	+000000
20	+000000
21	+000000
22	+000000
23	+000000
24	+000000
25	+000000
26	+000000
27	+000000
28	+000000
29	+000000
30	+000000
31	+000000
32	+000000
33	+000000
34	+000000

Accumulator: +000000

Load instructions to execute.

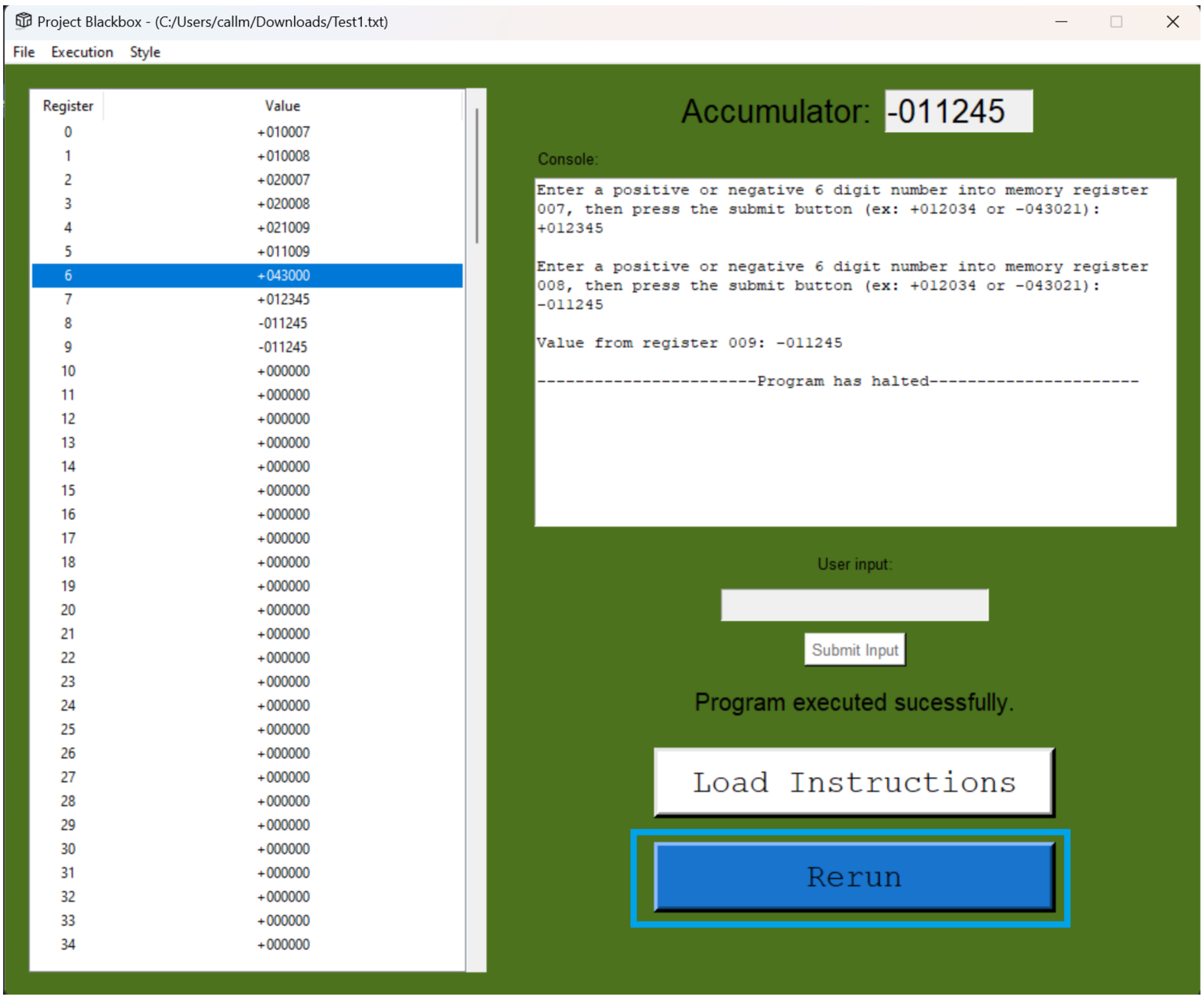
While the program is executing, a new button will appear that will allow you to abort the execution:



Register	Value
0	+010007
1	+010008
2	+020007
3	+020008
4	+021009
5	+011009
6	+043000
7	+000000
8	+000000
9	+000000
10	+000000
11	+000000
12	+000000
13	+000000
14	+000000
15	+000000
16	+000000
17	+000000
18	+000000
19	+000000
20	+000000
21	+000000
22	+000000
23	+000000
24	+000000
25	+000000
26	+000000
27	+000000
28	+000000
29	+000000
30	+000000
31	+000000
32	+000000
33	+000000
34	+000000

```
Enter a positive or negative 6 digit number into memory register
007, then press the submit button (ex: +012034 or -043021):
```

Once the program execution is completed, you will be given the option to rerun the program by pressing the button shown bellow (Just keep in mind that the program will run with all the modifications made during the previous execution):



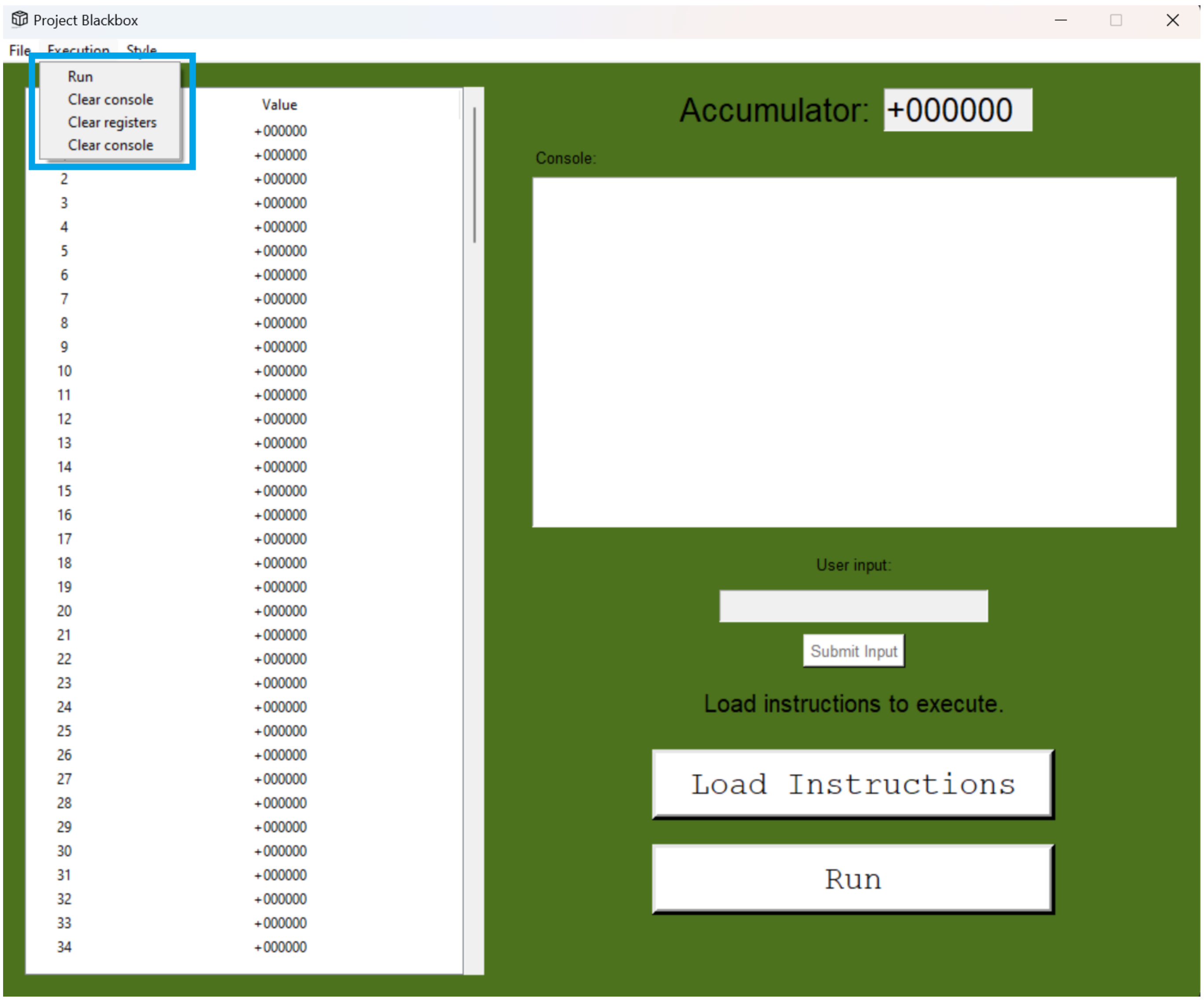
Register	Value
0	+010007
1	+010008
2	+020007
3	+020008
4	+021009
5	+011009
6	+043000
7	+012345
8	-011245
9	-011245
10	+000000
11	+000000
12	+000000
13	+000000
14	+000000
15	+000000
16	+000000
17	+000000
18	+000000
19	+000000
20	+000000
21	+000000
22	+000000
23	+000000
24	+000000
25	+000000
26	+000000
27	+000000
28	+000000
29	+000000
30	+000000
31	+000000
32	+000000
33	+000000
34	+000000

```
Enter a positive or negative 6 digit number into memory register
007, then press the submit button (ex: +012034 or -043021):
+012345

Enter a positive or negative 6 digit number into memory register
008, then press the submit button (ex: +012034 or -043021):
-011245

Value from register 009: -011245

-----------------------Program has halted----------------------
```

In the top of the window you will find a "File" menu that can be used to load the instructions or save the register instructions to a txt file:

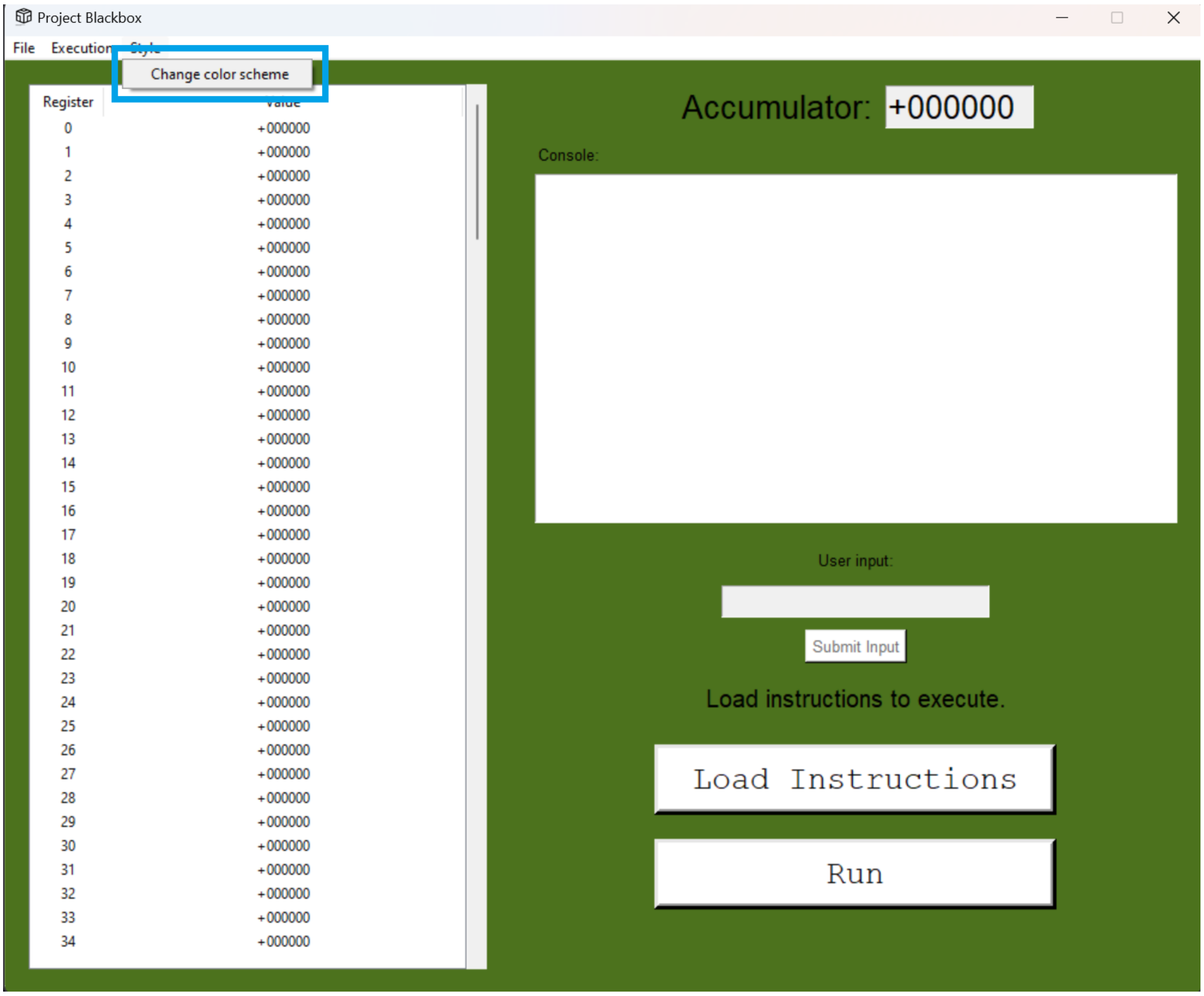
File Menu

Next to the "File" menu you will find the "Execution menu" that can be used to run/cancel/rerun program, reset all the register to 0, and clear the console:



	Value
2	+000000
3	+000000
4	+000000
5	+000000
6	+000000
7	+000000
8	+000000
9	+000000
10	+000000
11	+000000
12	+000000
13	+000000
14	+000000
15	+000000
16	+000000
17	+000000
18	+000000
19	+000000
20	+000000
21	+000000
22	+000000
23	+000000
24	+000000
25	+000000
26	+000000
27	+000000
28	+000000
29	+000000
30	+000000
31	+000000
32	+000000
33	+000000
34	+000000

If you wish to change the color scheme of the program, you may click on the "Change Color Scheme" under the Style menu.

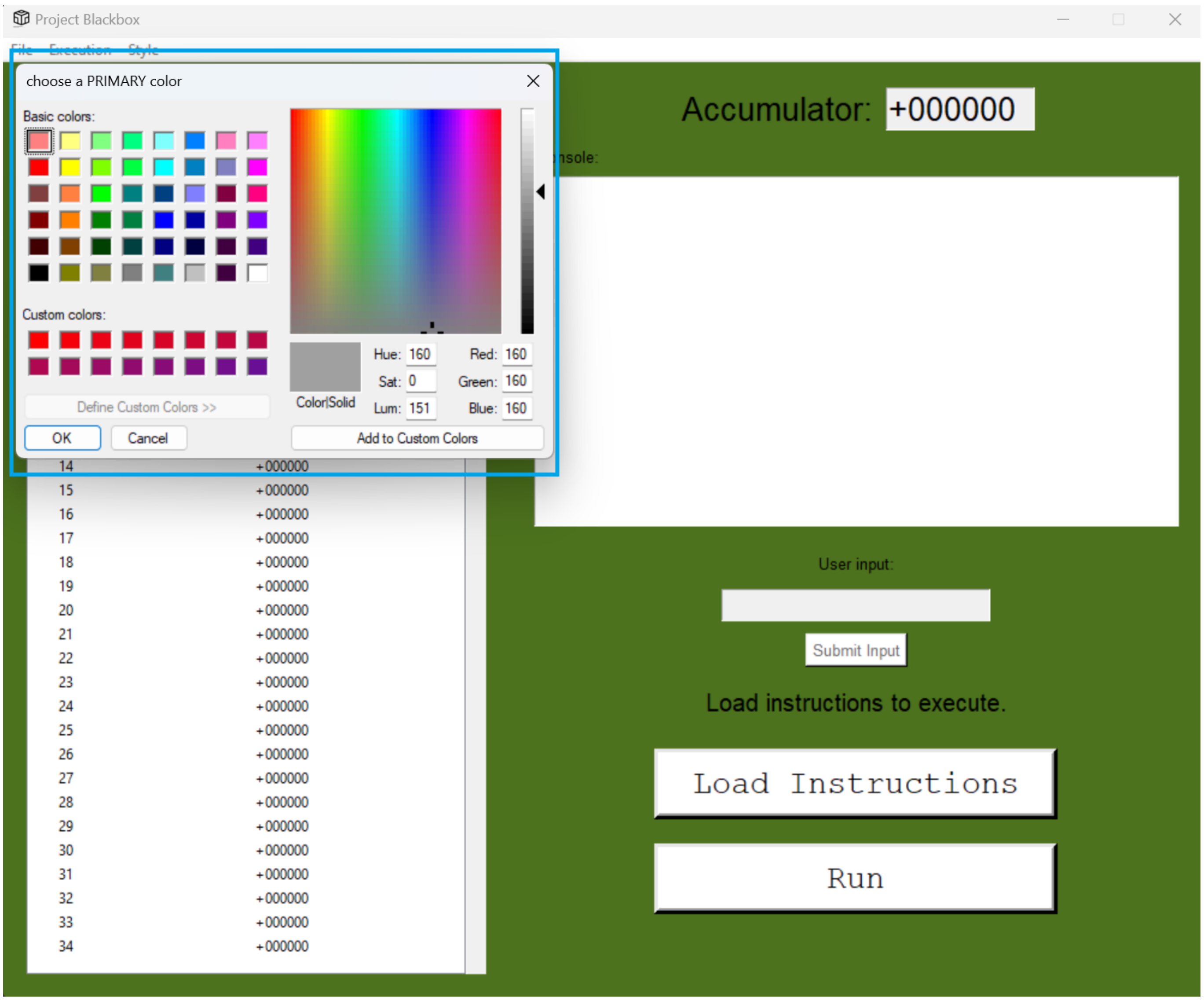


Accumulator: +000000

Register	Value
0	+000000
1	+000000
2	+000000
3	+000000
4	+000000
5	+000000
6	+000000
7	+000000
8	+000000
9	+000000
10	+000000
11	+000000
12	+000000
13	+000000
14	+000000
15	+000000
16	+000000
17	+000000
18	+000000
19	+000000
20	+000000
21	+000000
22	+000000
23	+000000
24	+000000
25	+000000
26	+000000
27	+000000
28	+000000
29	+000000
30	+000000
31	+000000
32	+000000
33	+000000
34	+000000

Load instructions to execute.

Once you select the option, you will be able to choose a primary color (color of the background) as shown bellow:



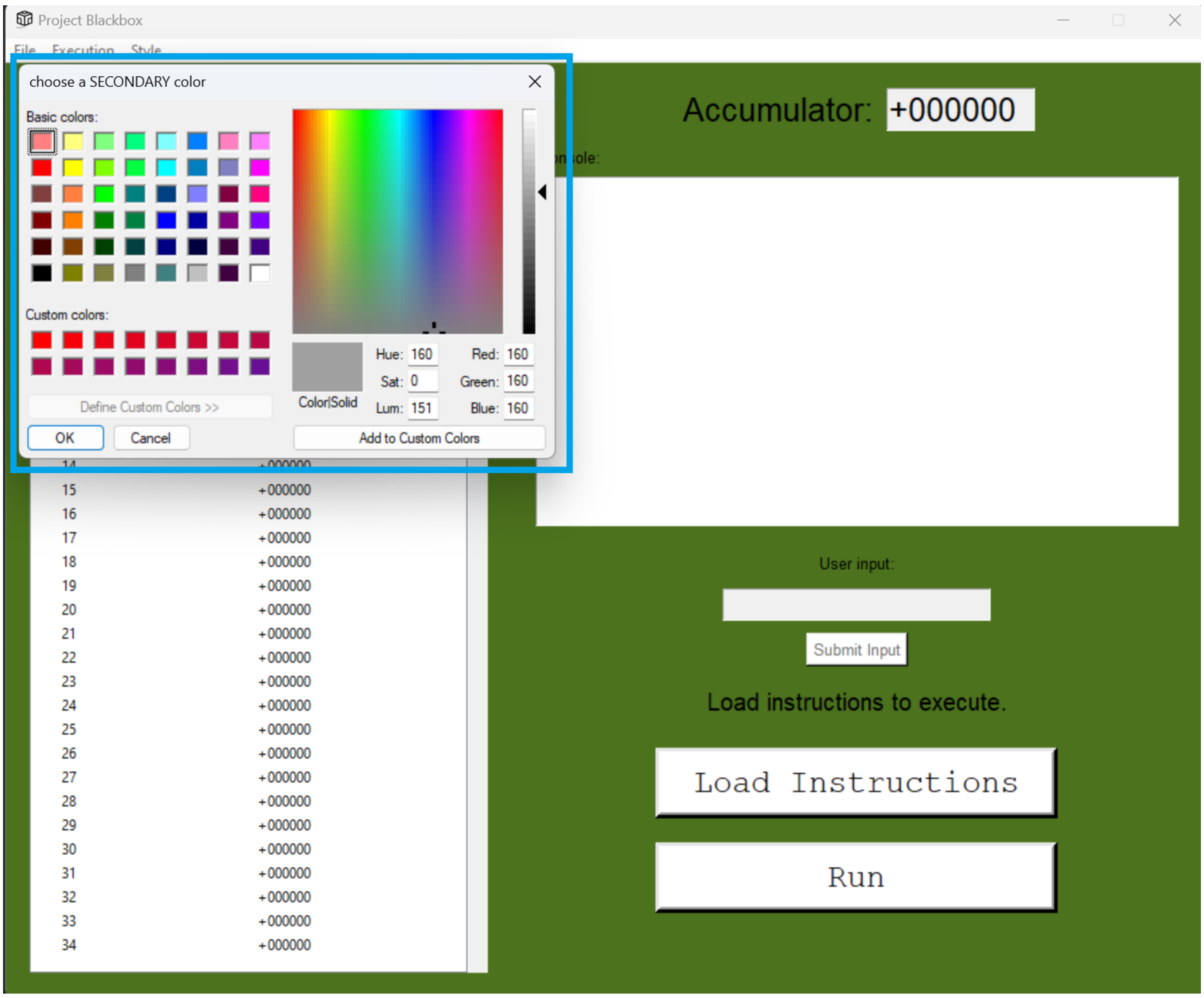
Hue: 160 Sat: 0 Lum: 151
Red: 160 Green: 160 Blue: 160

Accumulator: +000000

Load instructions to execute.

```
14    +000000
15    +000000
16    +000000
17    +000000
18    +000000
19    +000000
20    +000000
21    +000000
22    +000000
23    +000000
24    +000000
25    +000000
26    +000000
27    +000000
28    +000000
29    +000000
30    +000000
31    +000000
32    +000000
33    +000000
34    +000000
```

Then you will be able to choose a secondary color (color of the buttons) as shown bellow:



Hue: 160 Red: 160
Sat: 0 Green: 160
Lum: 151 Blue: 160

Accumulator: +000000

Load instructions to execute.

```
14  +000000
15  +000000
16  +000000
17  +000000
18  +000000
19  +000000
20  +000000
21  +000000
22  +000000
23  +000000
24  +000000
25  +000000
26  +000000
27  +000000
28  +000000
29  +000000
30  +000000
31  +000000
32  +000000
33  +000000
34  +000000
```

After selecting both colors, the color scheme of the program will be modified as shown bellow:

New Style

Tecnical implementation:

More information on how the program was implemented, please refer to the Class definition document.

Instructions available:

I/O operations:

- READ = 010 Read a word from the keyboard into a specific location in memory.
- WRITE = 011 Write a word from a specific location in memory to screen.

Load/store operations:

- LOAD = 020 Load a word from a specific location in memory into the accumulator.
- STORE = 021 Store a word from the accumulator into a specific location in memory.

Arithmetic operations:

- ADD = 030 Add a word from a specific location in memory to the word in the accumulator (leave the result in the accumulator)
- SUBTRACT = 031 Subtract a word from a specific location in memory from the word in the accumulator (leave the result in the accumulator)
- DIVIDE = 032 Divide the word in the accumulator by a word from a specific location in memory (leave the result in the accumulator).
- MULTIPLY = 033 multiply a word from a specific location in memory to the word in the accumulator (leave the result in the accumulator).

Control operatios:

- BRANCH = 040 Branch to a specific location in memory
- BRANCHNEG = 041 Branch to a specific location in memory if the accumulator is negative.
- BRANCHZERO = 042 Branch to a specific location in memory if the accumulator is zero.
- HALT = 043 Pause the program

Example of an valid instruction file:

The following is and example of the correct format of a file to be run through the program. Make sure all the instructions are stored on a *.txt file.

```
+010007
+010008
+020007
+020008
+021009
+011009
+043000
+000000
+000000
+000000
-99999
```

User Stories:

- Brian has taken an interest in computer science and wants to understand at a base level how assembly instructions work. He is able to write short instruction files to learn how each of them works and eventually how to perform multiple successive functions.

- Justin is designing a program on his raspberry pi. Because the raspberry pi doesn't have much in the way of memory he needs to design a program at the lowest level to improve memory and performance. For easier testing than running his long instruction programs on his raspberry pi, he is able to run this program from his computer and use the debugger to watch each register as a line of instruction is read.

Use Cases:

- Actor: User
- Goal: Write a word from memory to the screen (WRITE)

- System:
  - Checks validity of address.
  - Retrieves the word from the register.
  - Displays word to user.

- Actor: User
- Goal: Read a user-input word into memory (READ)
- System:
  - Get user-input word
  - Checks validity of address to ensure register exists.
  - Stores formatted input into desired register.

- Actor: System
- Goal: Add a word in memory to the word in the accumulator (ADD)
- System:
  - Checks validity of address.
  - Perform addition arithmetic.
  - Check validity of result (verify no overflow occured).
  - Store result in accumulator.

- Actor: System
- Goal: Convert 4 bit instructions to 6 bit equivalents
- System:
  - The previous process provides the conversion function with a list of instructions.
  - The function checks all instructions in search of 4-bit instructions
  - Add 0's accordingly.
  - Store new instruction.

- Actor: User
- Goal: Change color scheme of program
- System:
  - User selects "change color scheme" from the Style dropdown menu to begin the process.
  - User chooses a primary color (used for background).
  - User chooses a secondary color (used for buttons).
  - Program updates color scheme to reflect user-choices.

- Actor: System
- Goal: Validate user inputs and populate registers.
- System:
  - User presses the Process Entry Button to start the process.
  - Program gathers all of the user input.
  - Program validates the entry for size (verifies that it only contains a maximum of 250 instructions)
  - Program loads every instruction into a list.
  - Program converts any 4 bit instructions to 6 bit.

  - Program validates if all the user inputs are valid.
  - If all inputs are valid, the program load all the instructions to the simulator register.
  - Program refreshes the register display.
  - Program is now ready to be executed.

- Actor: User
- Goal: Open a file containing instructions
- System:
  - User selects "load instructions" on primary page to begin process.
  - User selects "open file" on secondary tab to open file browser.
  - User selects proper .txt file and contents are loaded into text-box.
  - File is now loaded and ready to be processed.

- Actor: User
- Goal: Open two files containing different instructions at once
- System:
  - User selects "load instructions" on primary page to begin process.
  - User selects "open file" on secondary tab to open file browser.
  - User selects proper .txt file and contents are loaded into text-box.
  - File is now loaded and ready to be processed.
  - User selects "open new window" from the file-drowndown menu.
  - Repeat steps 1-4 above.

- Actor: System
- Goal: Multiply a word from a register by the word in the accumulator.
- System:
  - Verifies address validity
  - Performs multiplication arithmetic.
  - Validates result of multiplication (no overflow)
  - Stores valid result in accumulator

- Actor: System
- Goal: Branch to a specific location in memory(register)
- System:
  - Checks address validity (between 0-249)
  - Programs current address is set to the specific location chosen in the instruction.