

Assembler GUI

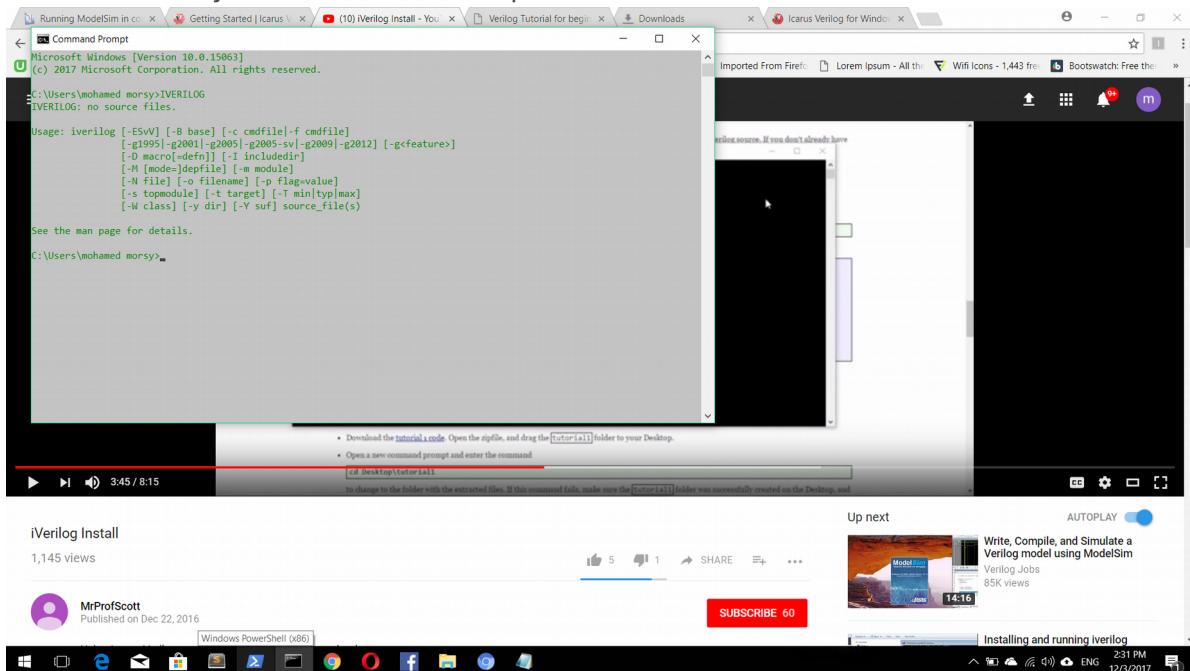
General Introduction:

Assuming having a verilog file NAME.v, This GUI is supposed to do the following:

- Provide text editor to write your assembly code in it
- run assembler code to turn input string into output binary file: BinaryFile.txt
- Provide display widget to see the assembly code you have written, binary code, and finally the output from the verilog NAME.v file
- You can open files, save files, and create new file using the GUI
- When saving an assembly code that you need to run on the GUI, it **MUST** be saved in the same directory as the GUI_assembler_Code.py code with the name: AssemblyFile.txt
- verilog file should read the BinaryFile.txt and produce OutputFile.txt for the GUI assembler to work correctly
- to use the GUI, you need to have python 3 installed. We use a built in library, called Tkinter to make the GUI widgets
- To compile and run verilog files:
 - You must download it through this link: <http://bleyer.org/icarus/>
 - after doing so, you need to add the path of bin file of iverilog (I.e: C:\iverilog\bin), and gtkwave (ie: C:\iverilog\gtkwave\bin) to your environment variables.

Check this link to know how to add environmental variables: <https://goo.gl/TWE8X1>

- To make sure you installed it write, open CMD and write IVERILOG



- use iverilog to use command lines in compiling & running
- To compile a file use command: iverilog file_name.v
This makes a a.out file to run later
- To run the compiled file, use command: vvp a.out
- after doing this, provided that the verilog code works correctly, it will read BinaryFile.txt, and produce OutputFile.txt, where I can read it from GUI
- Screenshot shows a simple hello.v file, and how to run it from command lines
Afterwards, we use python code to run command lines from python scripts using library: “subprocess” like this for example:
 - subprocess.call('iverilog hello.v', shell=True)
 - subprocess.call('vvp a.out', shell=True)

The screenshot shows a Windows desktop environment. In the foreground, there is a Notepad+ window titled "C:\iverilog\bin\hello.v" containing the following Verilog code:

```

1 module main;
2   initial
3     begin
4       $display ("learning Verilog is not easy boy !!");
5       $finish ;
6     end
7 endmodule

```

Below the Notepad+ window is a standard Windows taskbar with various icons. To the right of the Notepad+ window is a Command Prompt window titled "Command Prompt". The Command Prompt window shows the following terminal session:

```

Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\mohamed morsy\cd C:\iverilog\bin
C:\iverilog\bin>iverilog hello.v

C:\iverilog\bin>vvp a.out
learning Verilog is not easy boy !!

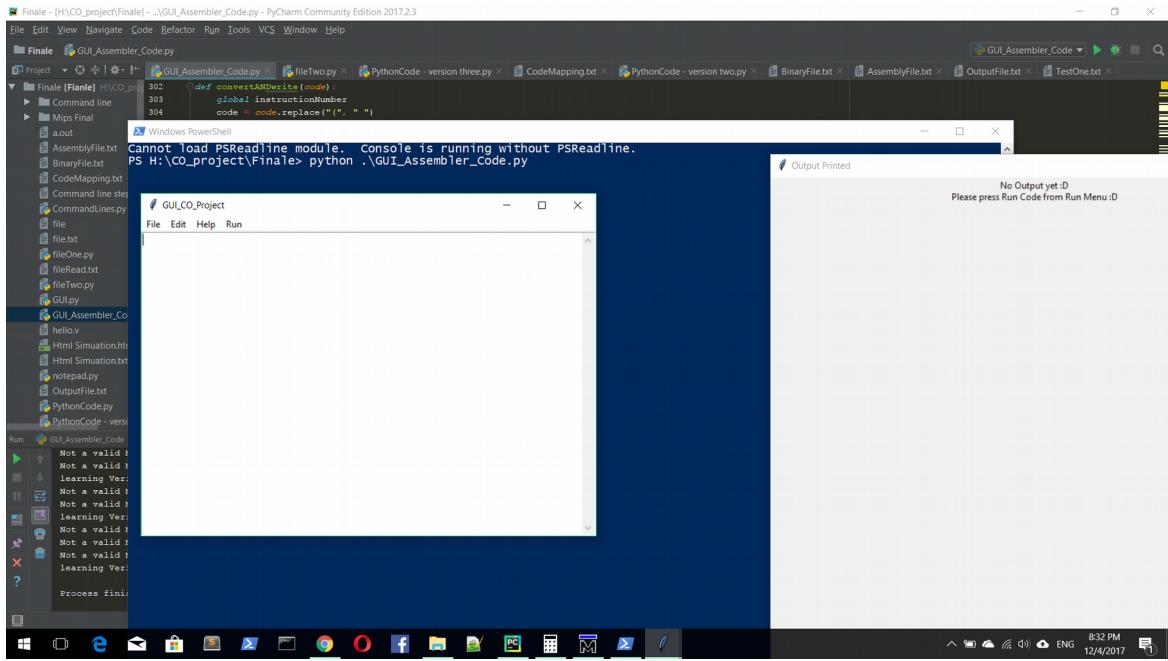
C:\iverilog\bin>_

```

The status bar at the bottom of the Command Prompt window provides system information such as length: 116, lines: 7, Ln:7 Col:11 Sel:0|0, and the date/time 12/3/2017 2:46 PM.

Overview on the Program:

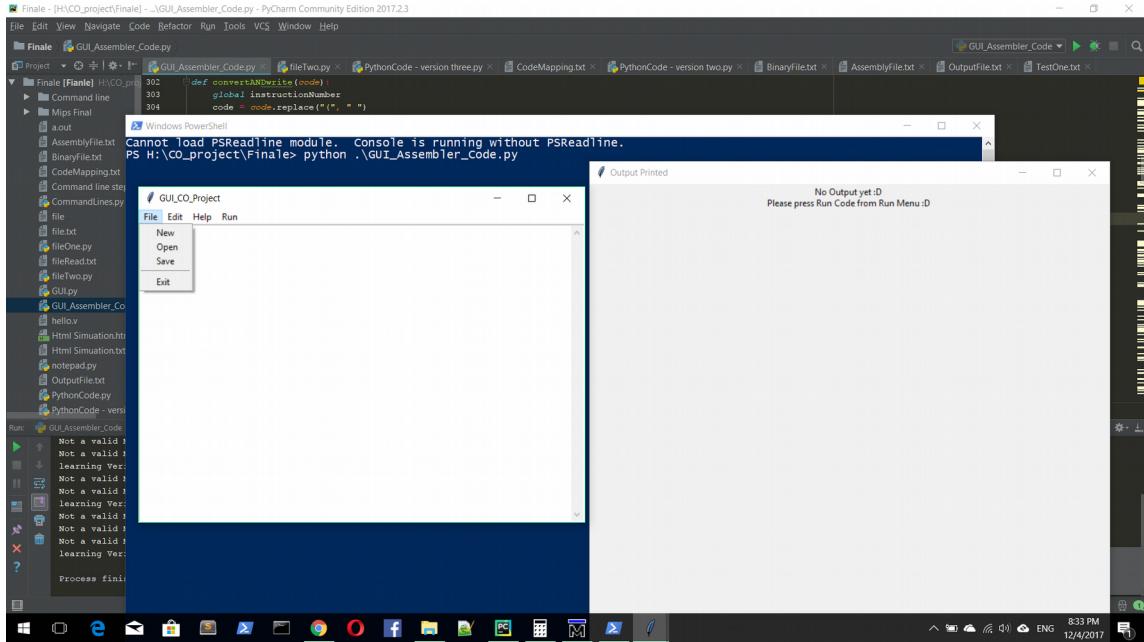
- To open the program:
 - navigate to the directory containing GUI_assembler_code.py
 - open your cmd usning shift + left mouse click >> open CMD
 - write command: python GUI_assembler_code.py
 - A picture to illustrate:



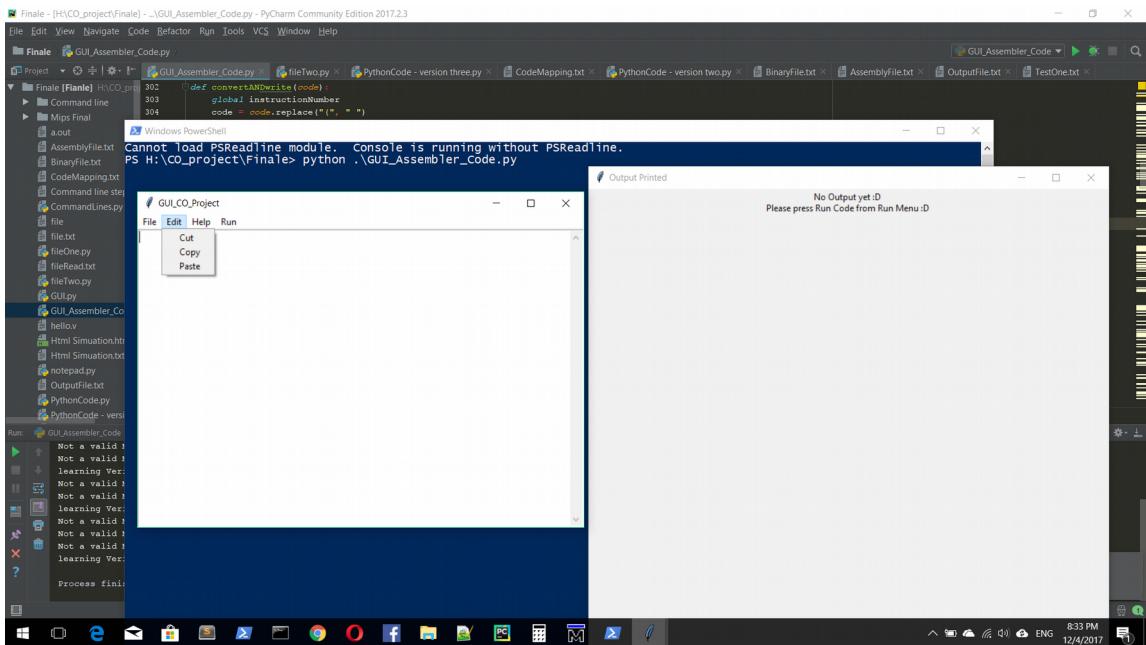
- Overview of the GUI:
 - Two windows are there:
 - GUI_CO_Project: your workspace
 - Output Printed: to see output

GUI_CO_Project

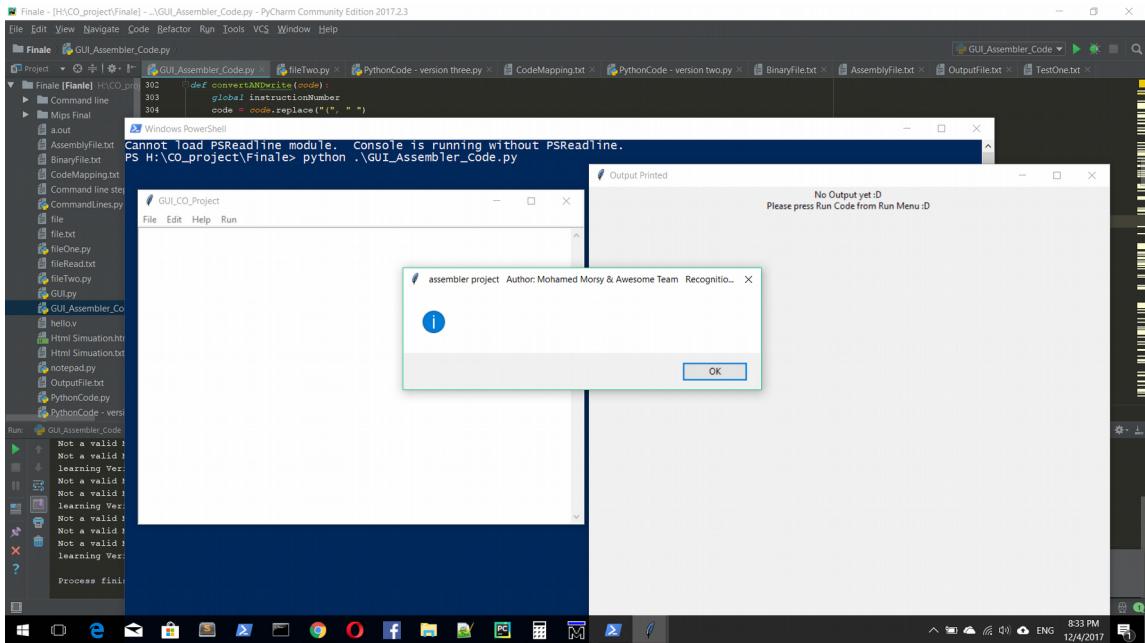
- you have 4 menus
- File menu to write new file, open one, or save:



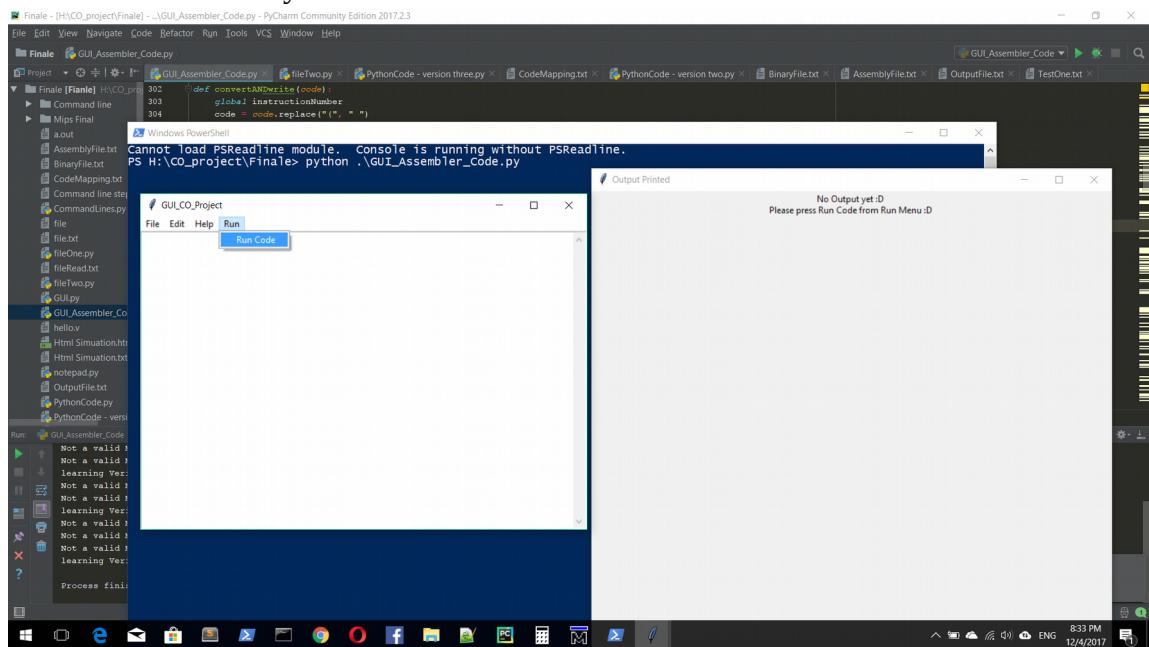
- Edit Menu to cut, copy, or paste (note if you write in the text editor, copy what you wrote, and close the program, you can't paste what you copied in the program when you open it one more time, so make sure to copy it first in another separate text file)



■ Help Menu

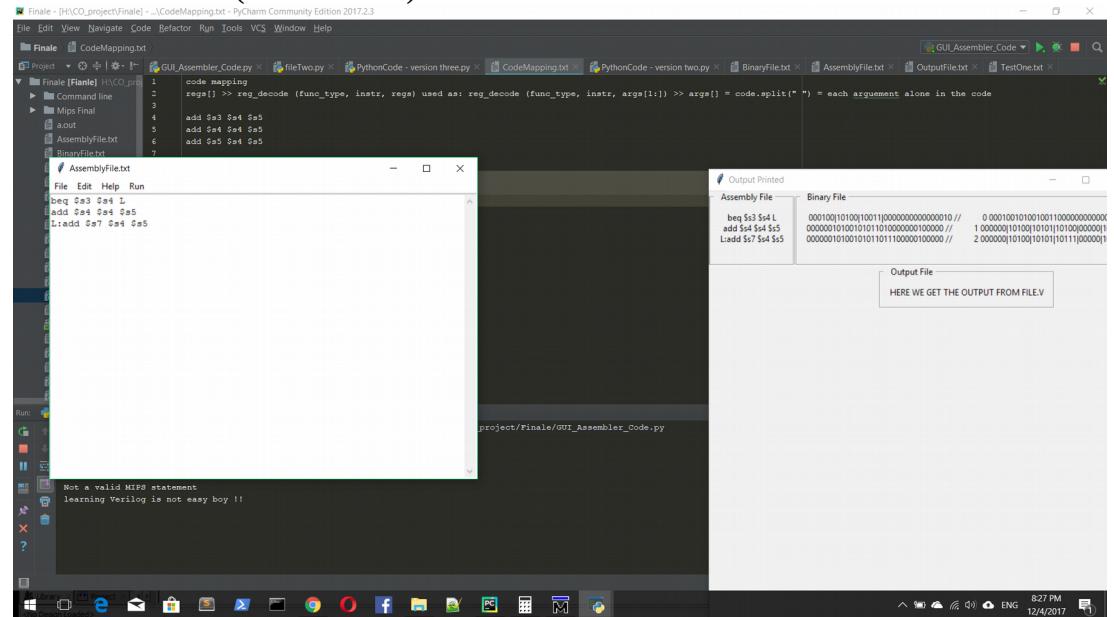


■ Run Menu to run your code:

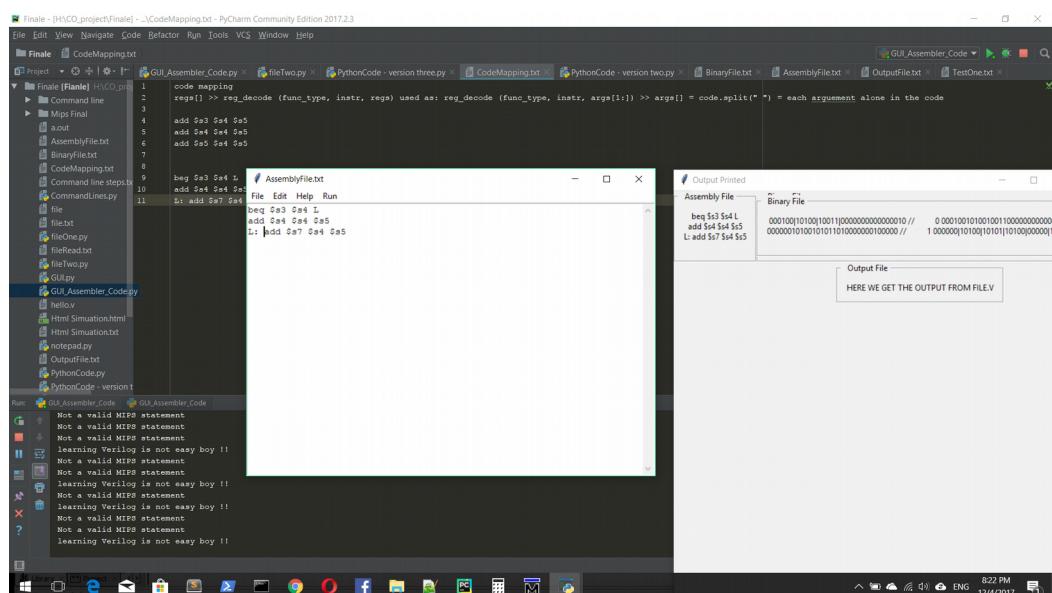


- You need to be careful with beq

- CORRECT:(L:Command)



- False: (L:SPACE)



- False:

beq \$s3 \$s5 Label >> use only letter L for label

Examples for code:

The screenshot shows the PyCharm IDE interface. On the left, the project tree displays files like 'AssemblyFile.txt' and 'BinaryFile.txt'. In the center, a code editor window shows assembly code:

```
1 code mapping
2 reg[] >> req_decode (func_type, instr, regs) used as: req_decode (func_type, instr, args[1:]) >> args[] = code.split(" ") = each argument alone in the code
3
4 add $s3 $s4 $s5
5 add $s4 $s4 $s5
6 add $s5 $s4 $s5
7 lw    $s4 50($s0)
8
9 beg $s3 $s4 L
10 add $s4 $s4 $s5
11 L: add $s7 $s4 $s5
12
13
```

To the right, a 'Binary File' window shows the assembly code:

```
Assembly File
lw    $s4 50($s0)
```

A callout box points to the 'Binary File' window with the text 'HERE WE GET THE OUTPUT FROM FILE.V'.

At the bottom left, the terminal window shows the command and its output:

```
*C:\Users\mohamed morsey\AppData\Local\Programs\Python\Python36-32
Not a valid MIF8 statement
Not a valid MIF8 statement
learning Verilog is not easy boy !!
```

The status bar at the bottom right indicates the date and time: 12/4/2017 8:29 PM.

The screenshot shows the PyCharm IDE interface. On the left, the project tree displays files like 'AssemblyFile.txt' and 'BinaryFile.txt'. In the center, a code editor window shows assembly code:

```
1 0000001010010101101100000100000 // 0 00000|10100|10101|10011|00000|100000
2 0000001010010101101000000100000 // 1 00000|10100|10101|10100|00000|100000
3 000000101001010110101000000100000 // 2 00000|10100|10101|10101|00000|100000
4
```

To the right, a 'Binary File' window shows the assembly code:

```
Assembly File
Binary File
add $s3 $s4 $s5
0000010100101011001100000100000 //
add $s4 $s4 $s5
1000001010010101101000000100000 //
add $s5 $s4 $s5
20000010100101011010100000100000 //
```

A callout box points to the 'Binary File' window with the text 'HERE WE GET THE OUTPUT FROM FILE.V'.

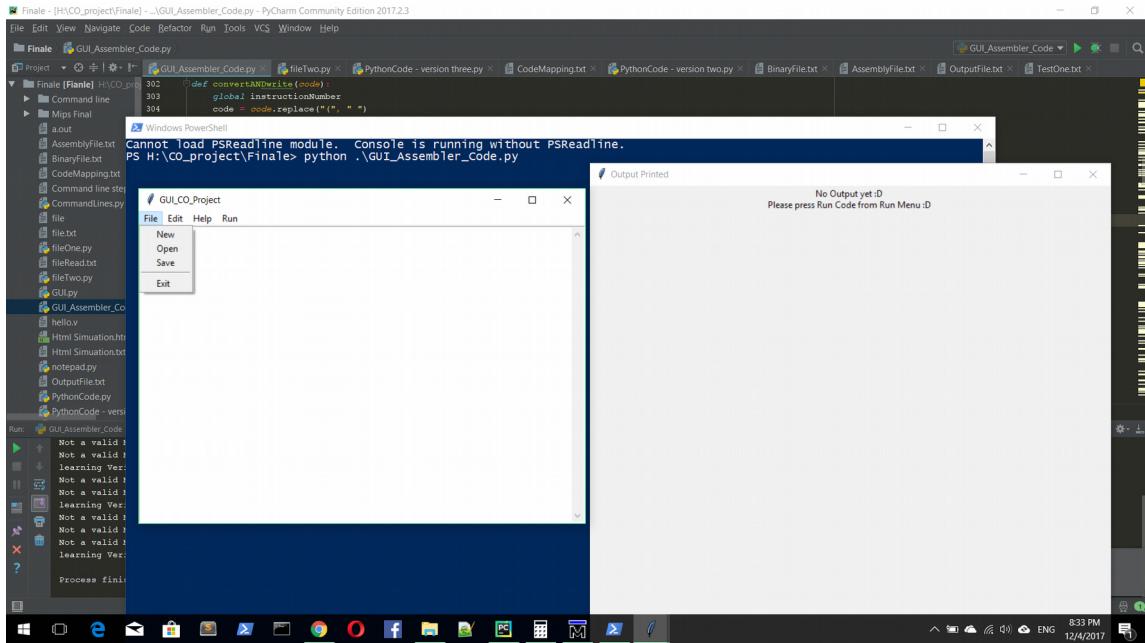
At the bottom left, the terminal window shows the command and its output:

```
*C:\Users\mohamed morsey\AppData\Local\Programs\Python\Python36-32
Not a valid MIF8 statement
Not a valid MIF8 statement
learning Verilog is not easy boy !!
```

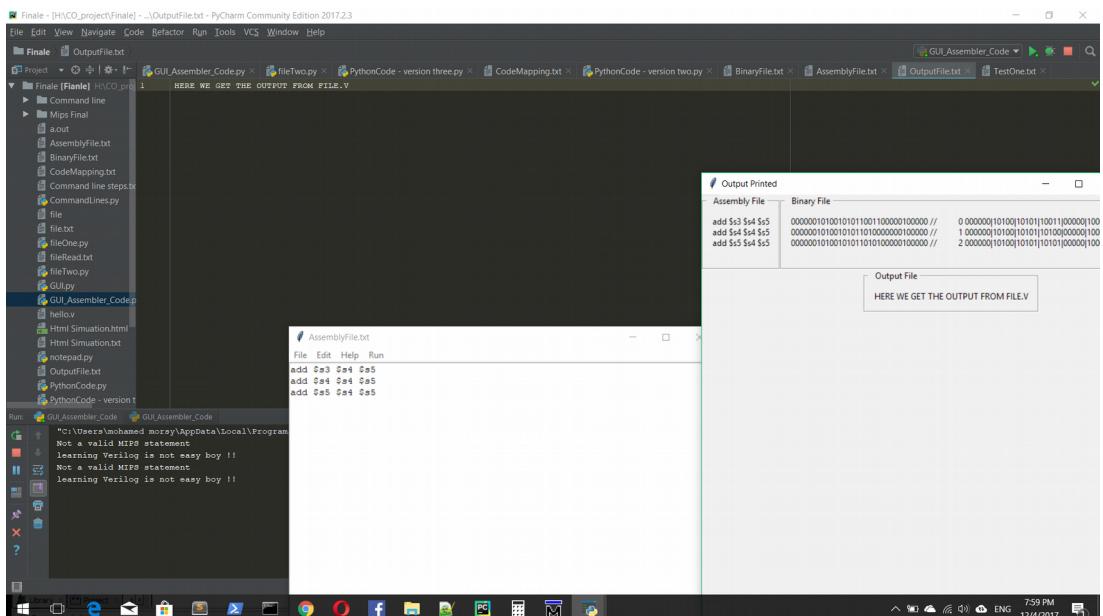
The status bar at the bottom right indicates the date and time: 12/4/2017 7:59 PM.

Output Printed:

- You have plain window with a text explaining you didn't run code yet



- You have three frames after clicking “Run code” in “Run” menu
 - Assembly file frame: redisplays what you wrote in the GUI text editor
 - Binary file frame: displays certain format for the assembled code “binary”
 - Output file frame: displays what is meant to be shown from the \$display or \$monitor command in verilog >> we write it into a new file named outputFile.txt, and read it into the Output file frame on the GUI check screenshot:



Bugs I couldn't fix due to lack of time:

1. When checking console of any IDE like pycharm, you find this statement when error occurs:
“Not a valid MIPS statement !!”
But it shows up even when there is no problem, though the assembly code is fully assembled into binary with no mistakes.
So it doesn’t affect the functionality of the program, especially if you run it from command lines directly, you won’t even notice the problem
2. You can’t select text using shift + up/down, then copy and paste it into the GUI text editor, this makes various bugs I couldn’t fix due to lack of time
3. I couldn’t test the actual verilog code, and combine it to the GUI code, meaning I couldn’t test whether the output file is generated correctly or not
[NOTE] This is out of the GUI scope, as it only reads the OutputFile.txt
4. You can’t use any label name rather than “L” in beq
5. You can’t write label like this:
L: VERILOG STATEMENT
example: “L:**|**add \$s7 \$s4 \$s5”
You MUST attach the colon to the verilog statement like this:
L:VERILOG STATEMENT
example: “L:**:**add \$s7 \$s4 \$s5”
6. You can’t leave more than one extra space like this:
lw \$s4 50(\$a0)
7. Three file names can’t change, and must be in the same directory as GUI_Assembler_Code.py
 1. AssemblyFile.txt >> saves the code you wrote in the GUI text editor
 2. BinaryFile.txt >> output from the GUI, and displayed in the display widget called: “Output Printed”
 3. OutputFile.txt >> output from verilog code, displays what’s usually written in the test bench display command