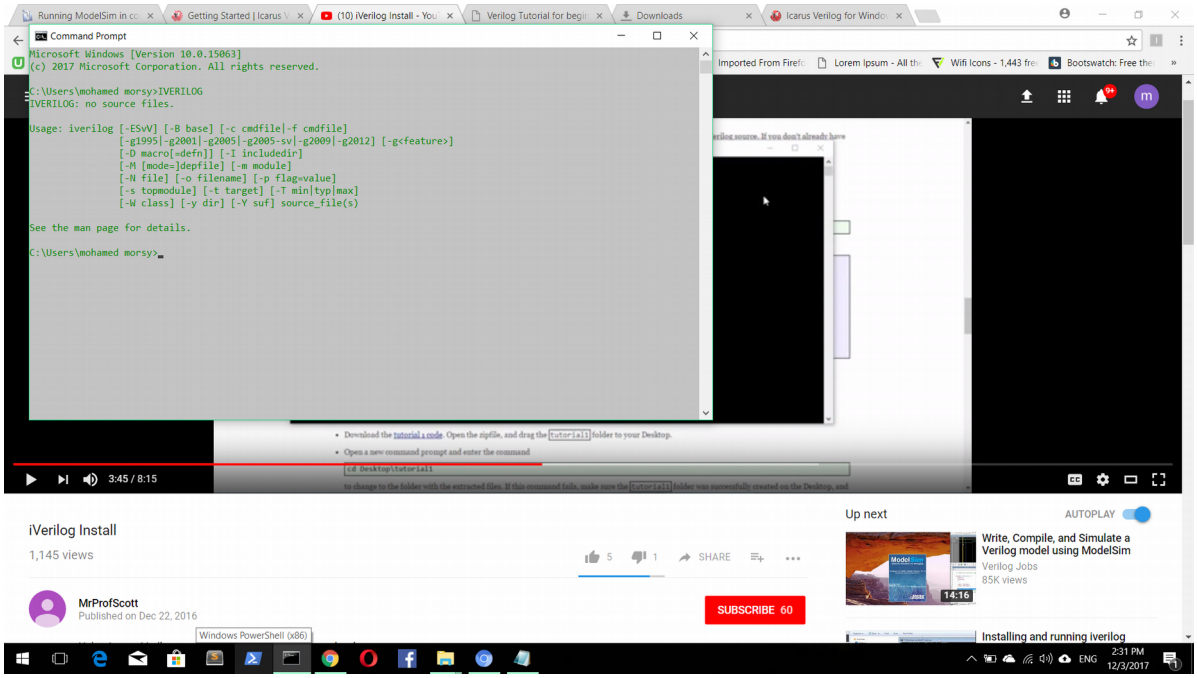
Assembler GUI

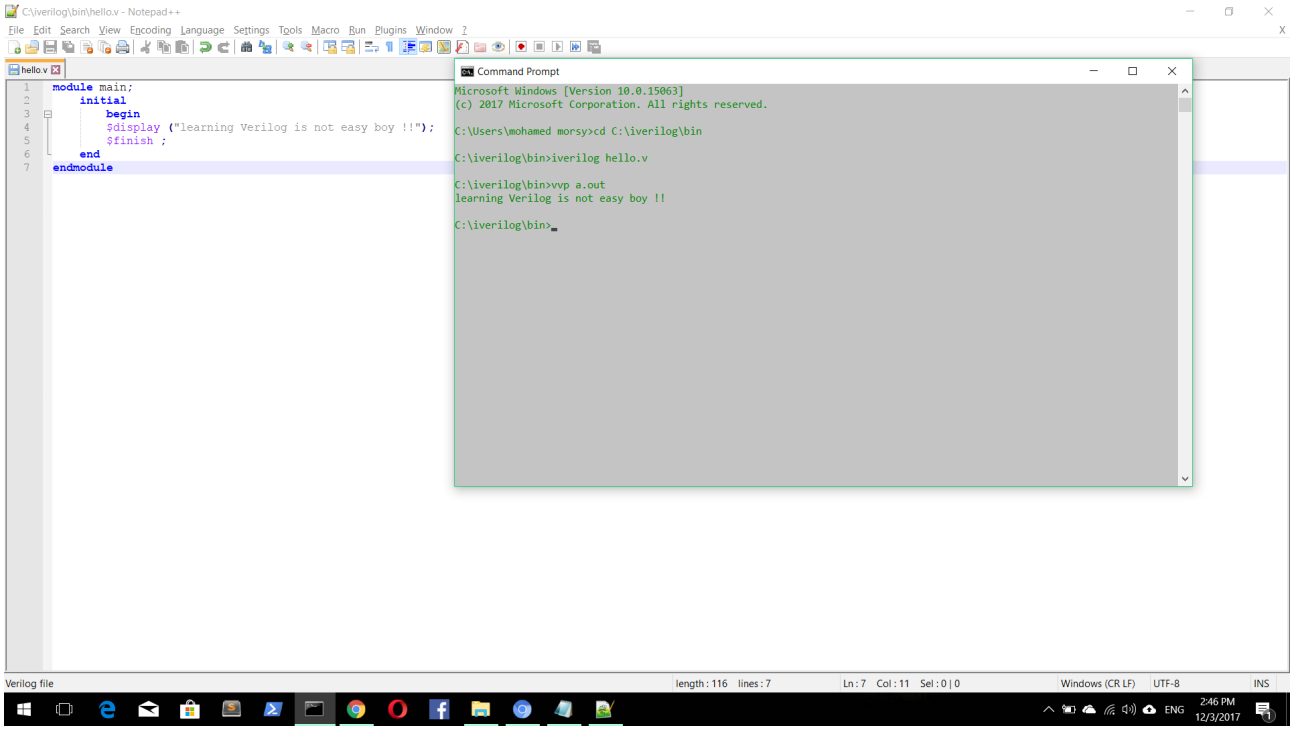
General Introduction:

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 thourgh this link: http://bleyer.org/icarus/
  - after doing so, you need to add the path of bin file of iverilog (I.e: C:\iverlilog\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

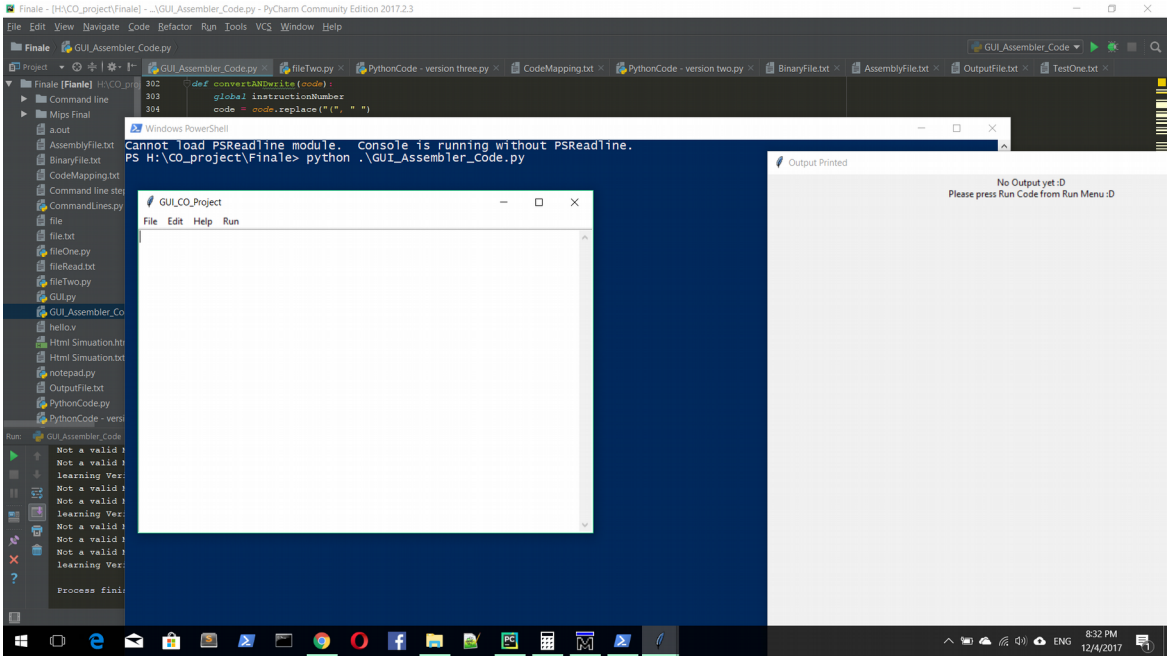


- use iverilog to use command lines in compiling & running
- To compile a file use command: iverilog file_name.v
  This makes a a.out file to run later
- To run the compiled file, use command: vvp a.out
- after doing this, provided that the verilog code works correctly, it will read BinaryFile.txt, and 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)



Overview on the Program:

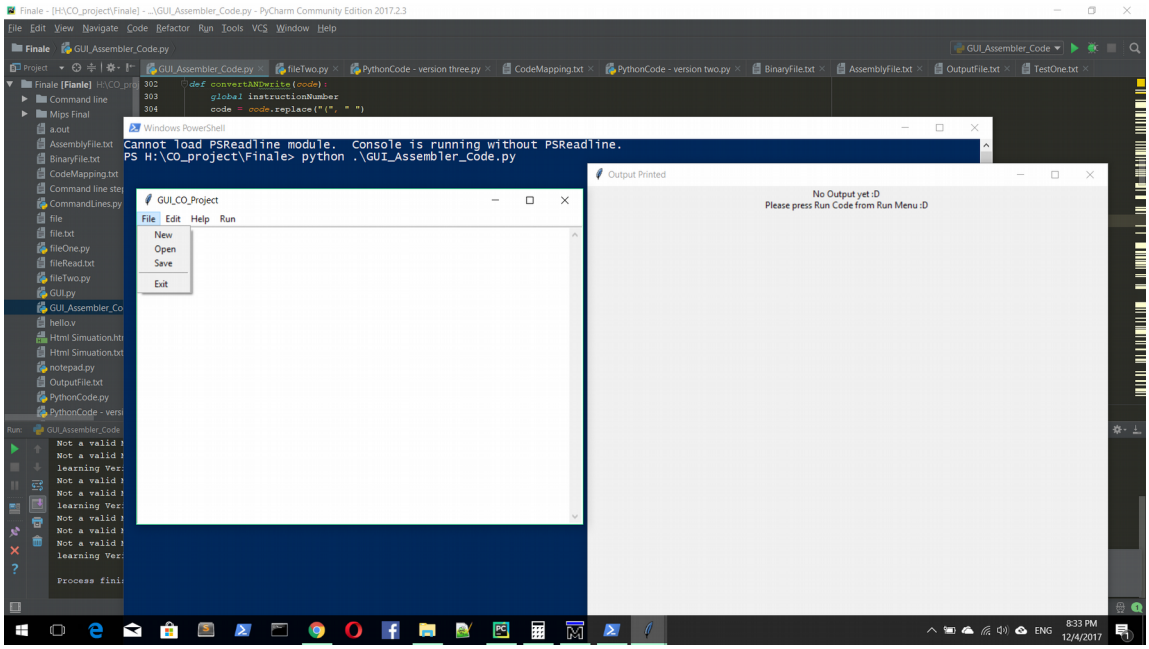
- To open the program:
  - navigate to the directory containing GUI_assembler_code.py
  - open your cmd usning shift + left mouse click >> open CMD
  - write command: python GUI_assembler_code.py
  - A picture to illustrate:



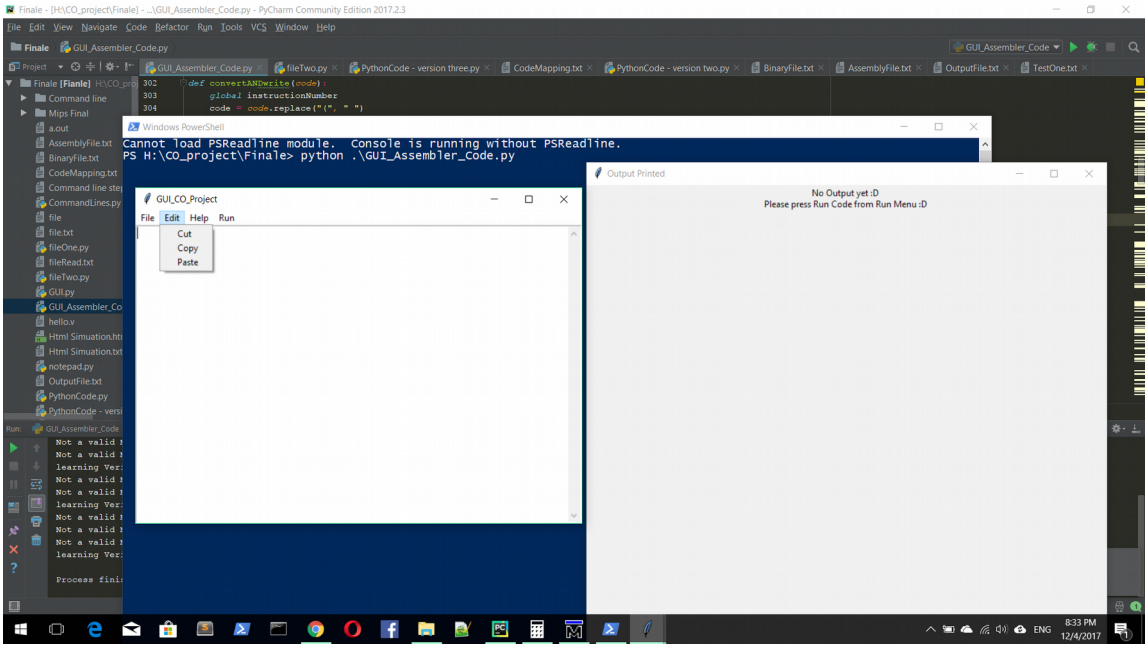
- Overview of the GUI:
  - Two windows are there:
    - GUI_CO_Project: your workspace
    - Output Printed: to see output

GUI_CO_Project

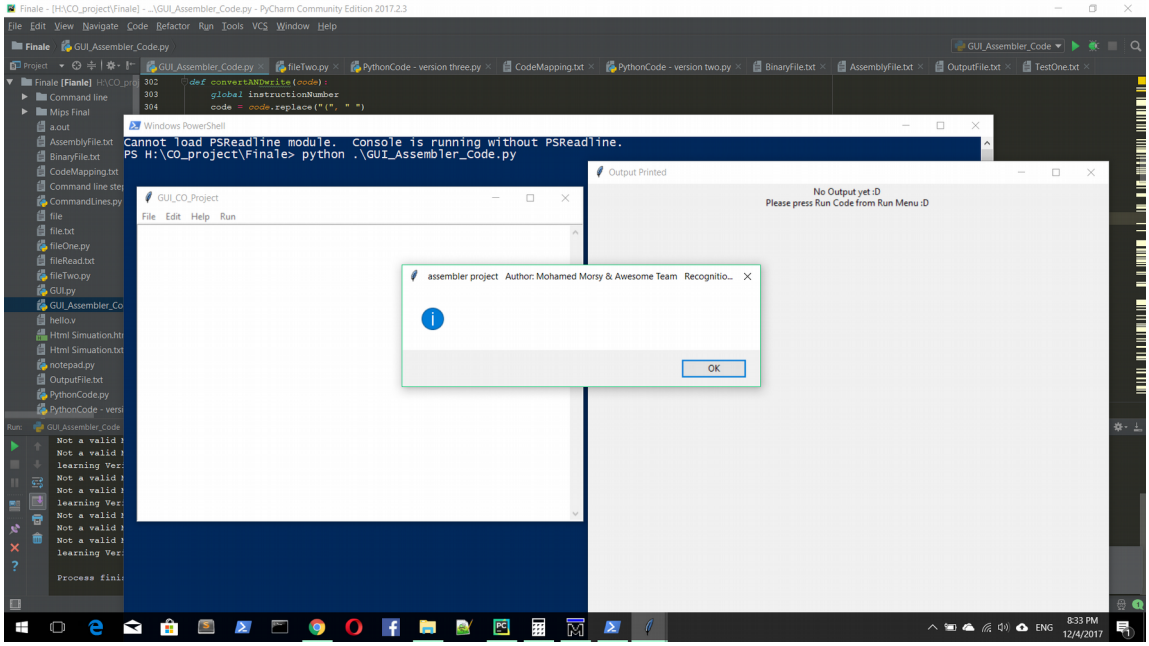
- you have 4 menus
- File menu to write new file, open one, or save:



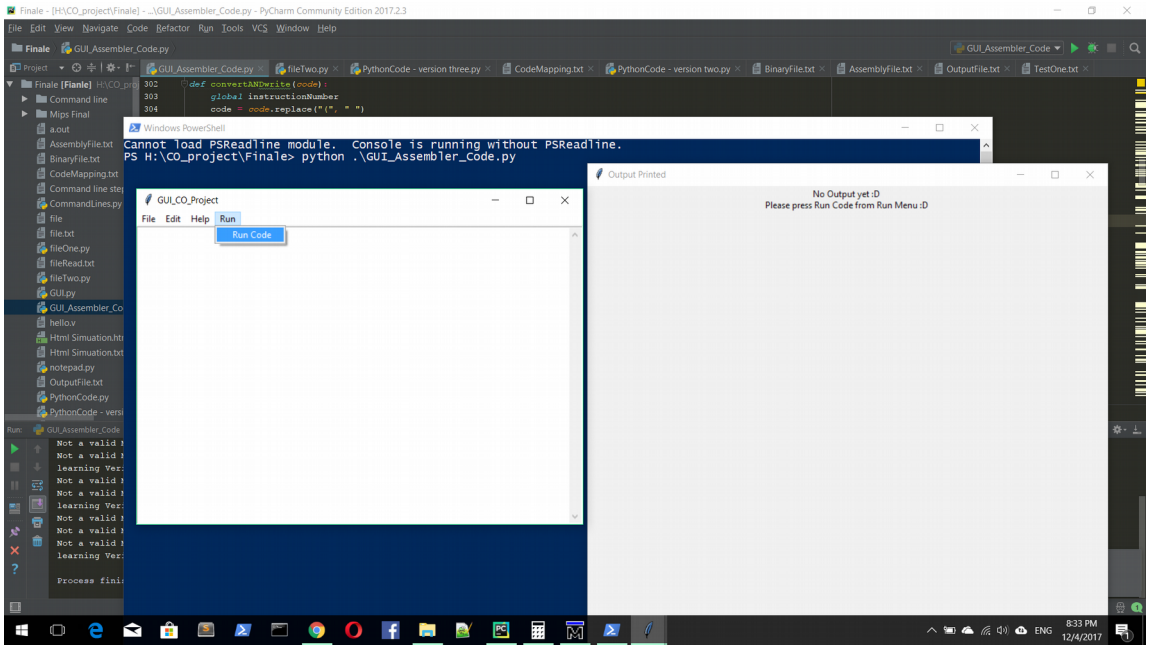
- Edit Menu to cut, copy, or paste (note if you write in the text editor, copy what you wrote, and close the program, you can't paste what you copied in the program when you open it one more time, so make sure to copy it first in another separate text file)



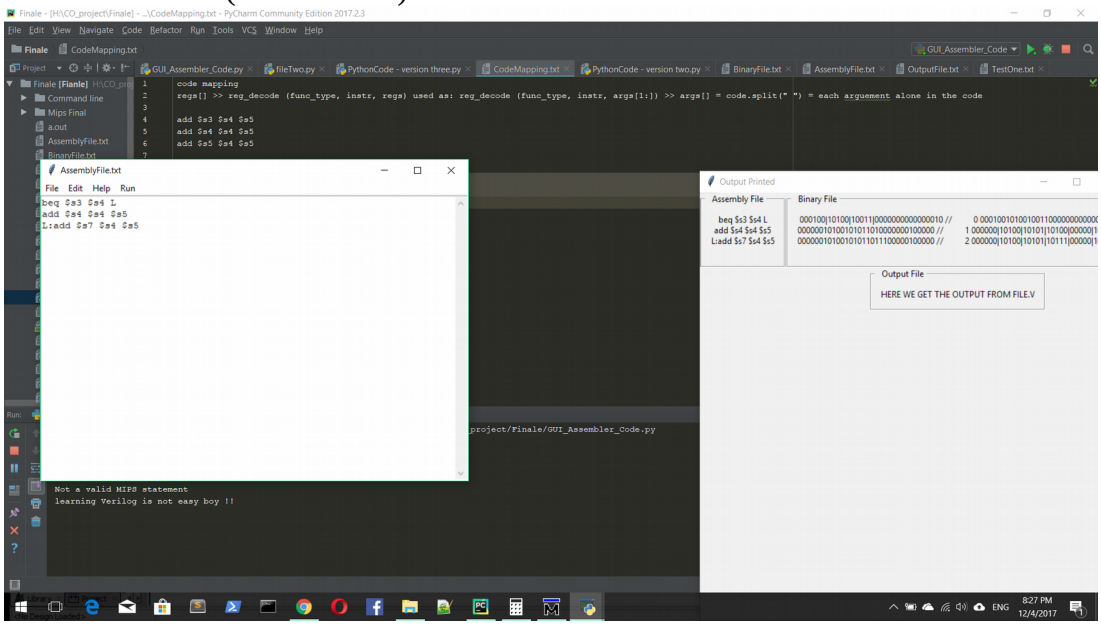
- Help Menu



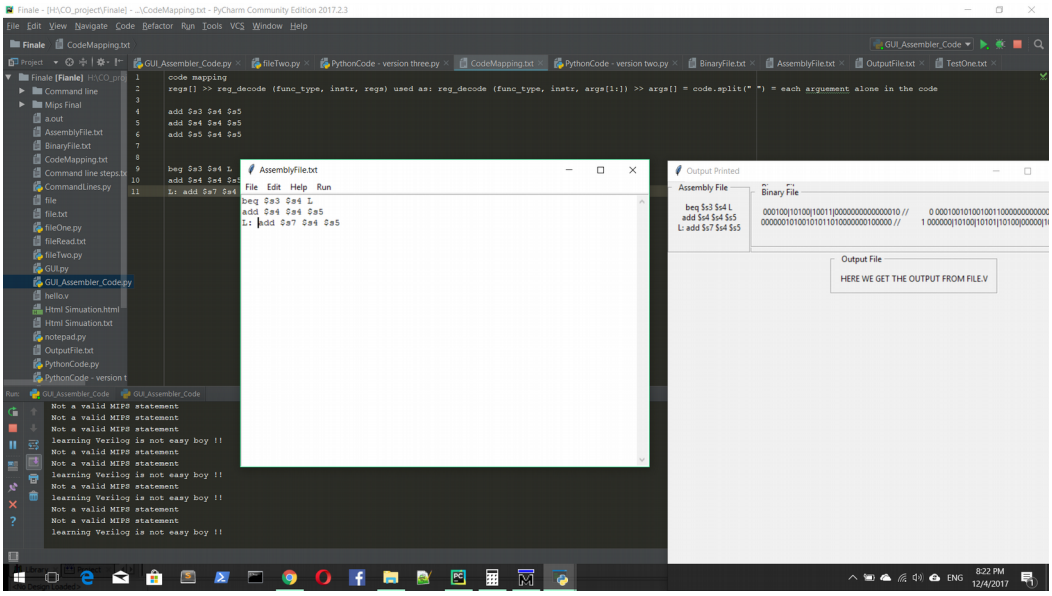
- Run Menu to run your code:



- You need to be careful with beq
  - CORRECT:(L:Command)

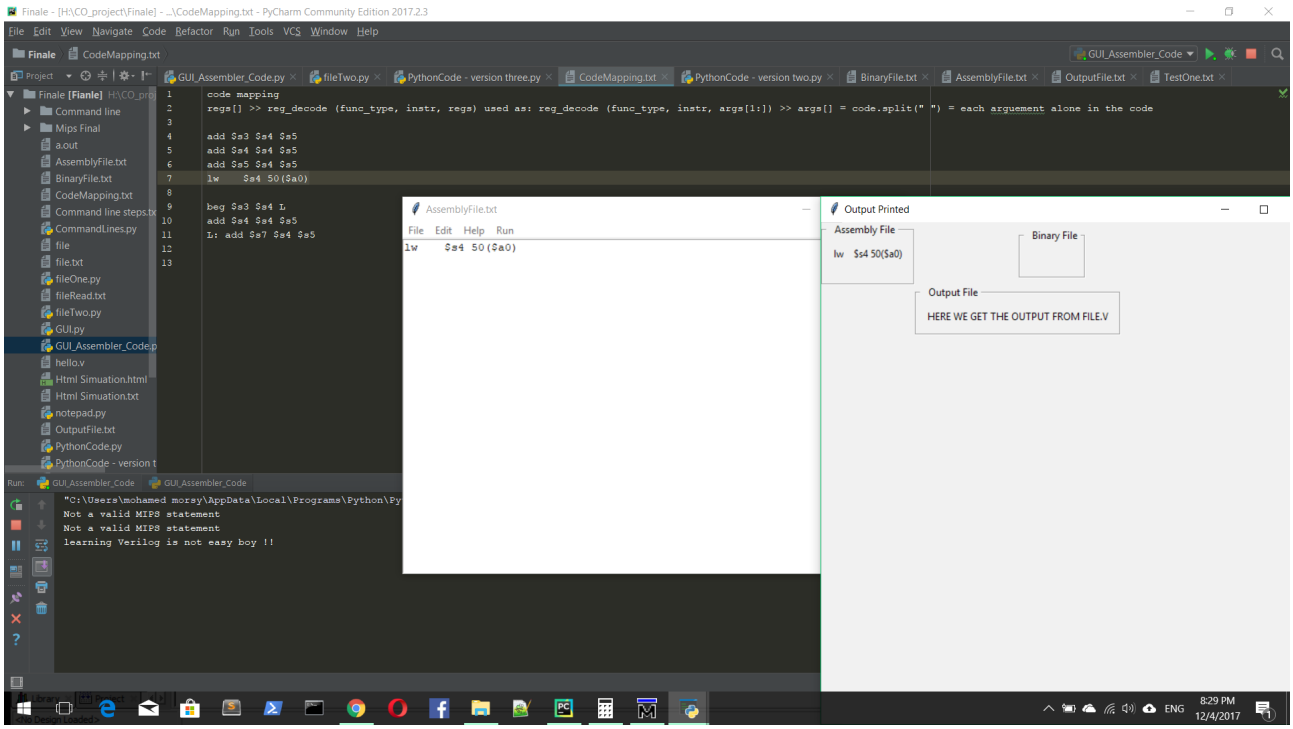


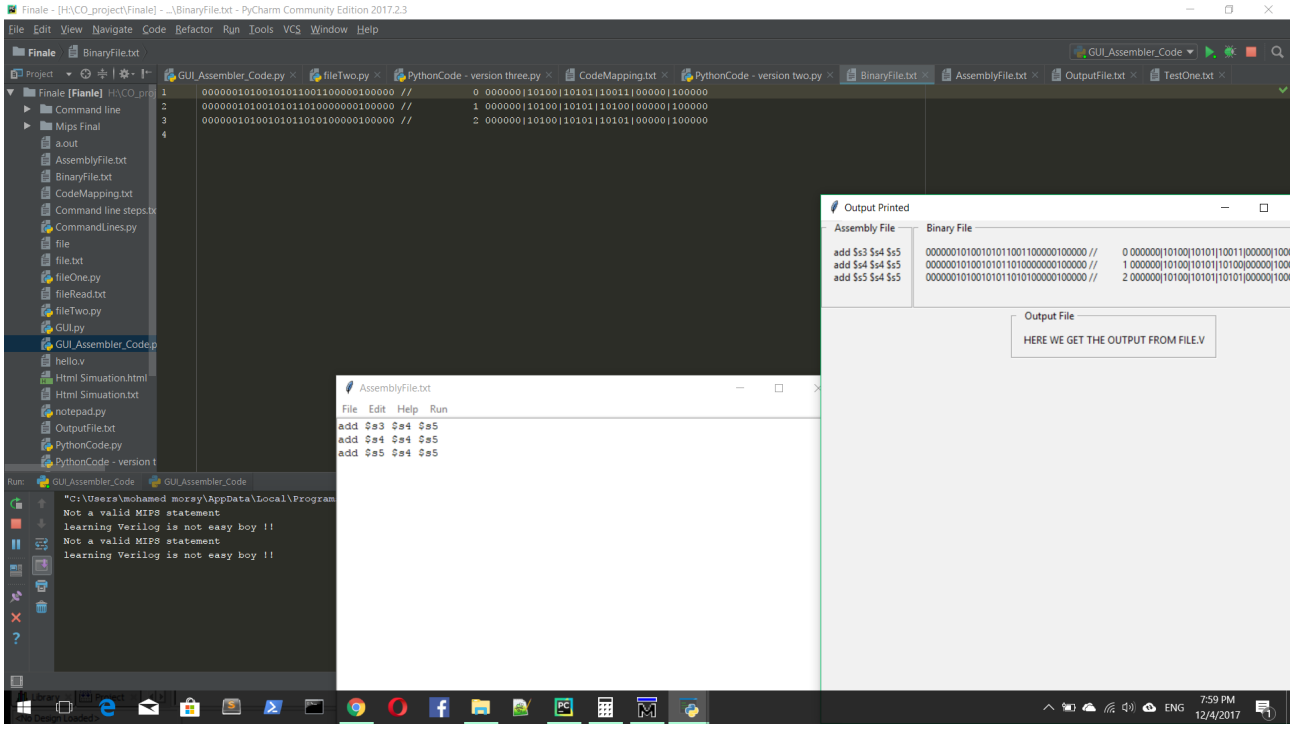
  - False: (L:SPACE)



  - False:
    beq $s3 $s5 Label >> use only letter L for label

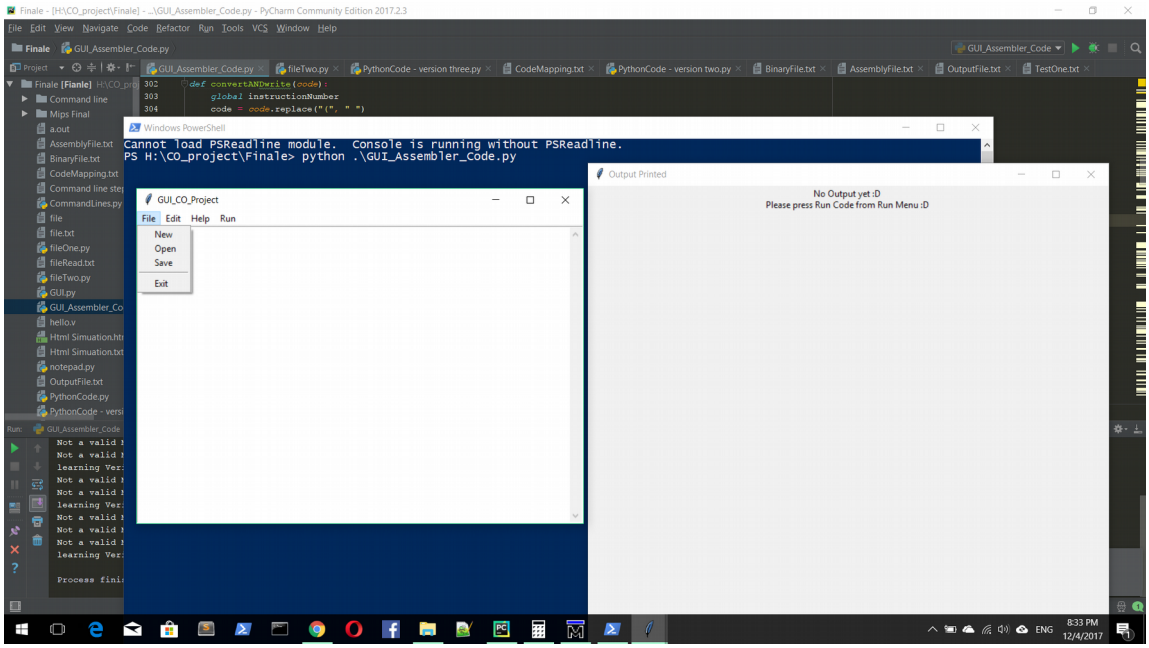
Examples for code:



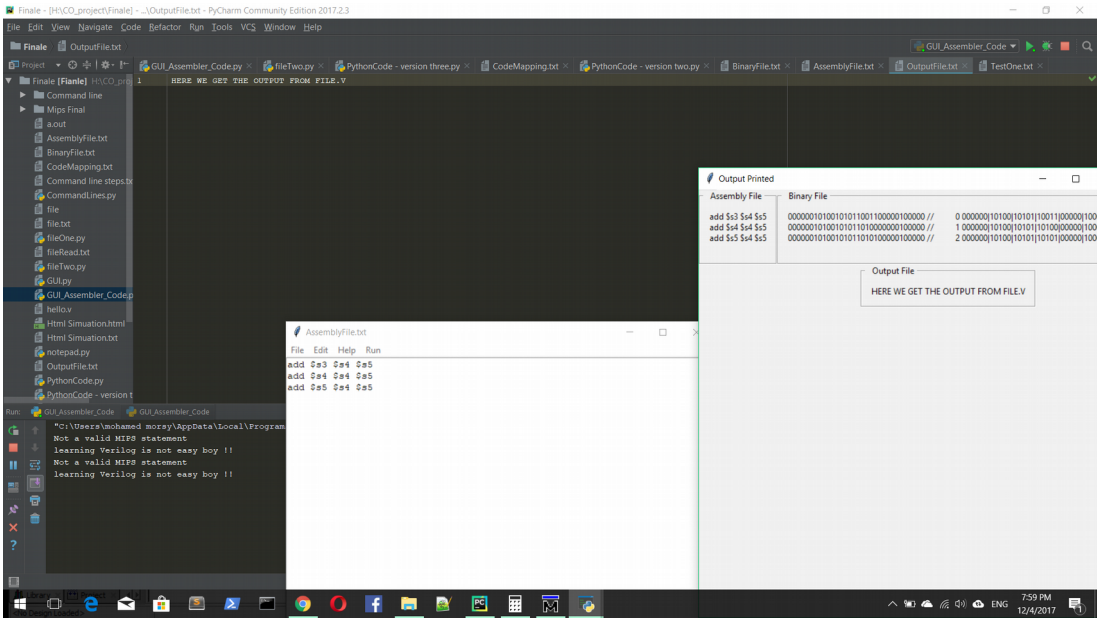


Output Printed:

- 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 affext the functionality of the program, espcially if you run it from command lines directly, you wont 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 ststement 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