**Documentation**

**ASSUMPTIONS**

Following assumptions are used for the better working of the code:

1. We have used case sensitive input.
2. The input is indentation sensitive.
3. Comments will not be multiple lines.
4. Variables are declared in the format: X 10

(a tab will be in between the variable and its corresponding value.)5.

5. There will not be any blank line in the input code given by the user.

6. There are no constants (i.e. literals) in the input (as announced in class).

7. A label can be declared only once.

8. No space/lines must be left after ‘END’ statement.

9. Each instruction would cost 1 bit so we have 256 bits of storage.

10. Comments must start with "/”.

11. The comments must be 3 tabs away from the starting of the line.

12. Users cannot add comments while providing value to variables.

**README**

We have used Python language (python 3) for writing the code.

To run the program, run assembler\_code.py file and write the name of the text file (which is saved in the same folder in which assembler\_code.py is saved.)

**Working of the code:**

OPCODE TABLE

Opcode                                   Meaning           Assembly Opcode

0000 Clear accumulator CLA

0001 Load into accumulator from address LAC

0010 Store accumulator contents into address SAC

0011 Add address contents to the accumulator contents ADD

0100 Subtract address contents from accumulator contents SUB

0101 Branch to address if accumulator contains zero BRZ

0110 Branch to address if accumulator contains negative value   BRN

0111 Branch to address if accumulator contains positive value     BRP

1000 Read from the terminal and put in address INP

1001 Display value in an address on terminal DSP

1010 Multiply accumulator and address contents MUL

1011 Divide accumulator contents by address content.

The quotient in R1 and remainder in R2               DIV

1100 Stop execution STP

The code consists of the following functions:

* passOne: It will first read the file provided by the user. The variable "location\_counter" is initialised with 0 if input doesn't provide address location with START; else it will start with the address location provided; i.e. if input says: START 100, location\_counter is initialised to 100. We are building an opcode table, label table and symbol table with the help of dictionaries which have been initialized in the beginning: opcodeTable = {}, symbolTable = {} and lableTable = {}. We have considered forward referencing. If the program reads the ‘END’ statement, the first pass is finished. The description of the dictionaries are:

* opcodeTable = {}: The key of the dictionary is location counter and the value is a list where first index is assembly opcode symbol; second is opcode of the symbol; third is the operand.
* symbolTable = {}: The key is the variable/label and the value is a list where first index is the value stored in variable and second index is memory allocated to variable.
* lableTable = {} : The key of the dictionary is label and the value is location counter
* isVariableValid: This function is used to check whether the variables provided by the user are valid or not. If the variable used by the user is present in opcodeTable or is an assembly directive (‘START ‘or ‘END’), then it will show an error. Also, if the variable used is ‘R1’ or ‘R2’ (used for DIV and DIV is not used before the variable in the code), then it will show an error. We cannot assign the same name to a variable and label.
* isLabelValid: This function is used to check if the label is valid or not. It will show an error if the label is the same as opcode or assembly directive or the same as a variable.
* readVariables:  It allocates memory to the variables and checks whether the variables initiated by the user is used by the user in the code or not (when declared) and checks whether a value is assigned to the variable or not (otherwise shows an error). The function displays a warning if a variable is assigned a value but is not used in the code.
* secondPass: It is used to generate the object code. The input in decimal format is converted to Binary and is generating the machine code for the given instruction.
* isSymbolValid: It checks whether the given opcode is valid or not.
* decimalToBinary: It converts a decimal to Binary.
* printDict: It prints all the tables which are stored in the form of dictionaries in our code.
* printCool: It is printing the sub-heading (initialising assembler, initialising first pass, initialising second pass) in typing effect, on the screen.

When the user will run the program, the program will ask the user to enter the name of the file. If the user will enter the valid file, then will print the following: ‘Given opcode’, ‘Label Table’, ‘Symbol Table’, ‘Opcode Table’ and ‘Object Code’. If the user enters an invalid text file, it will show an error and will ask for a valid file.

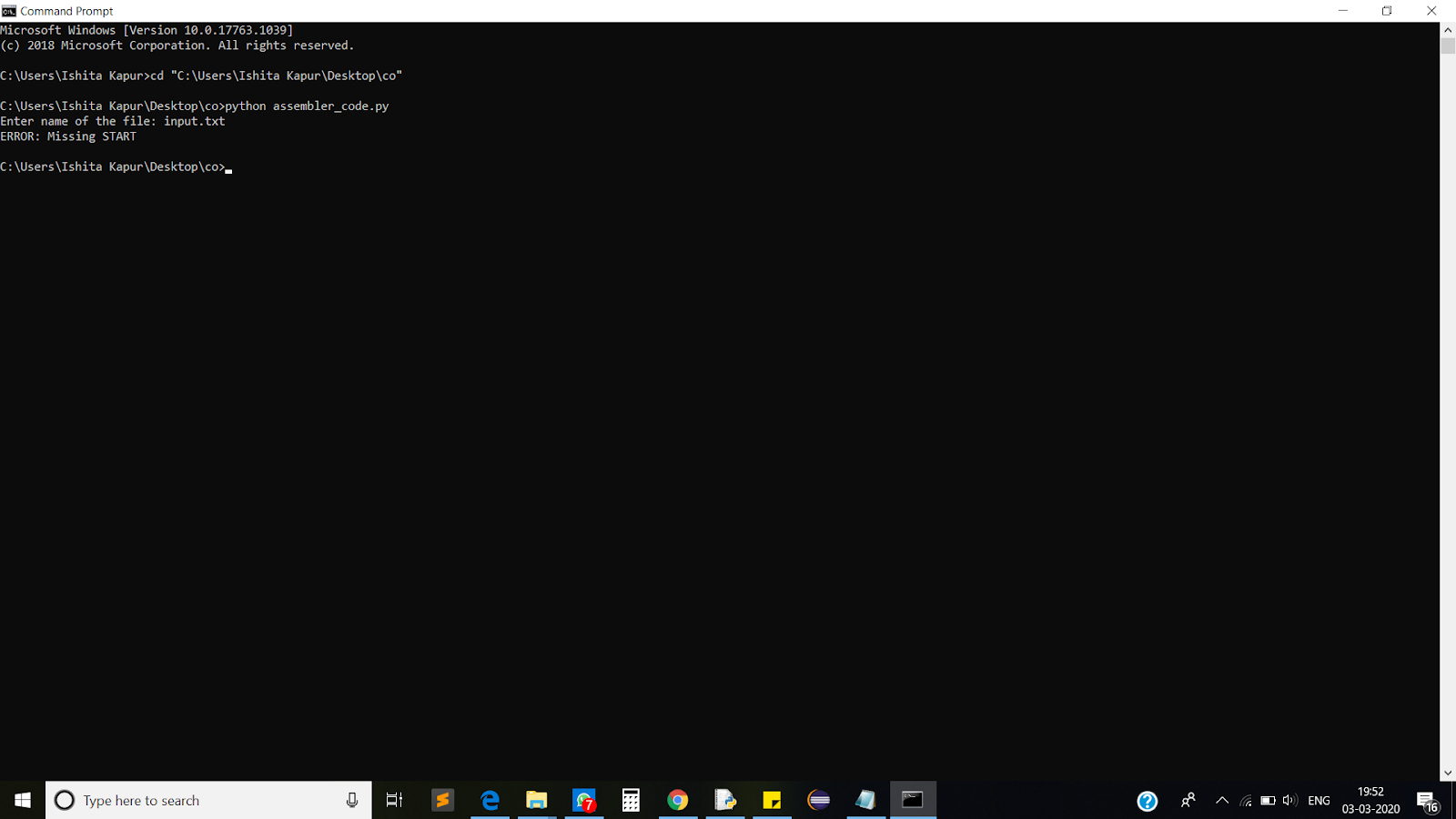
**ERROR REPORTING AND WARNING**

The program will terminate when an error will be displayed on the screen. It will display the error on the screen and the program will terminate. Warning (check point number 11) doesn’t affect the functioning of the program.

Following errors are handled by the program:

1.Missing of “START” statement:

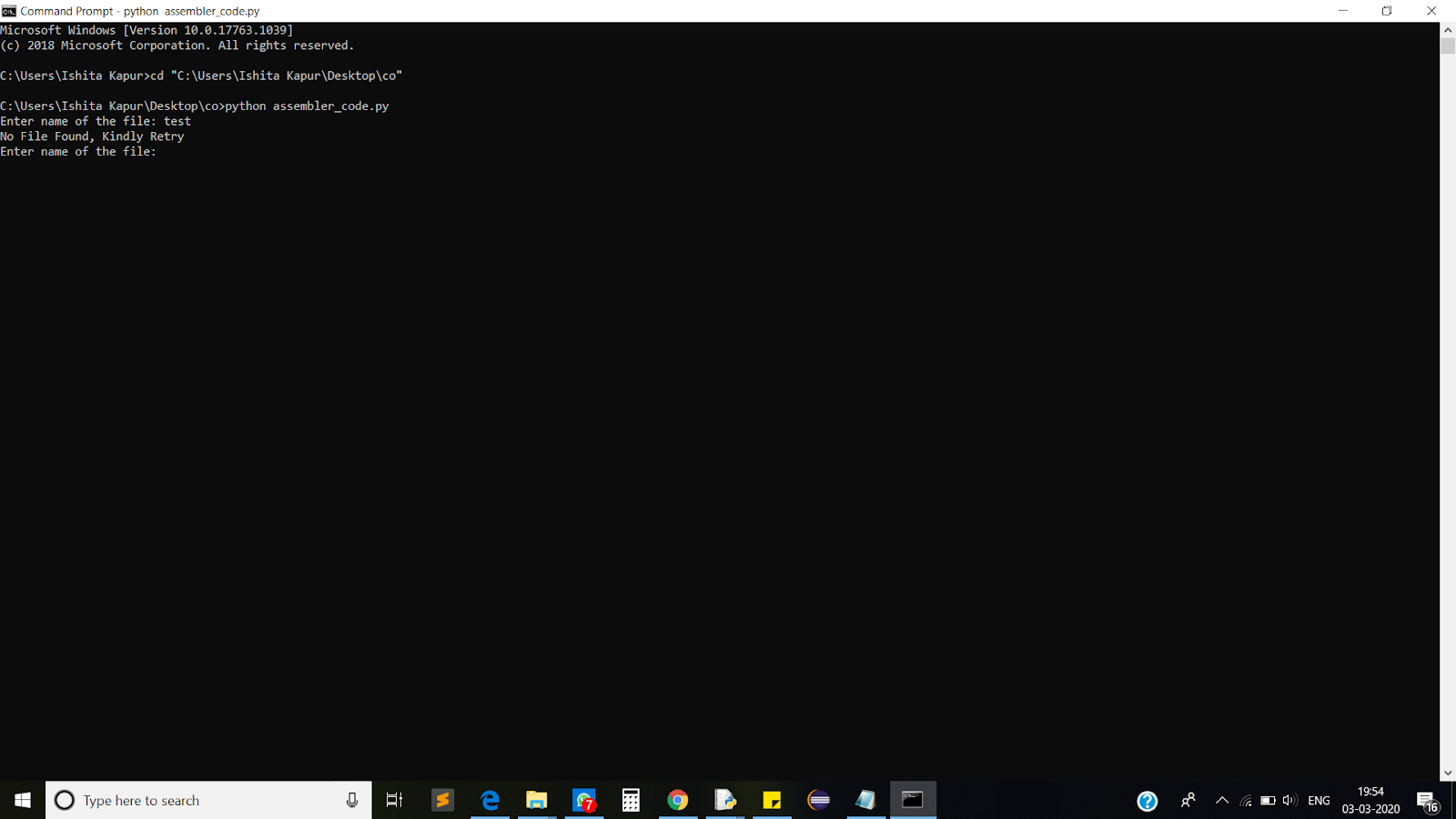
   If the program does not have a “START” statement, then it will show the following error.



2. File not Found:

    If the user does not enter a valid text file, then it will show the following error-

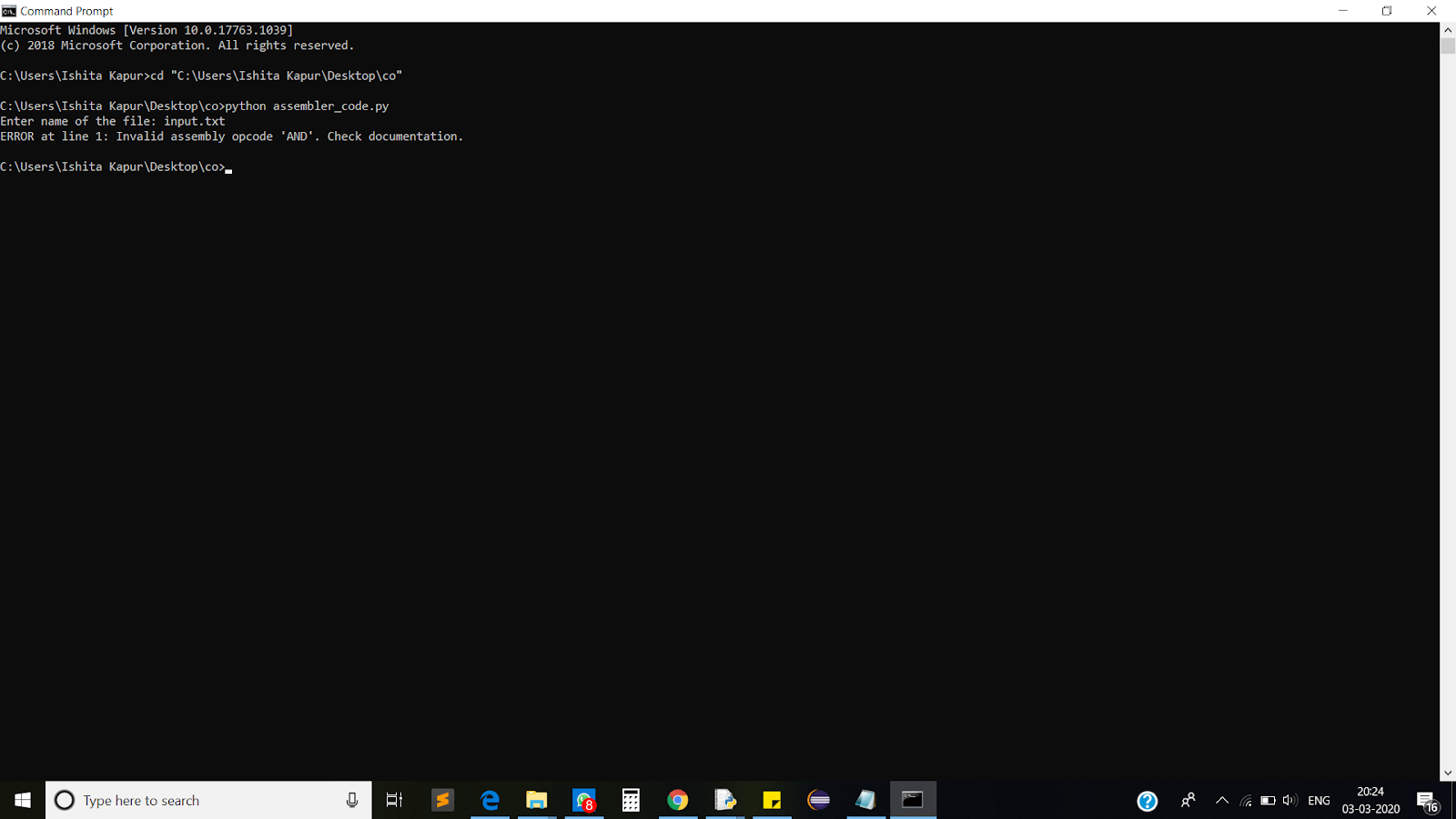
    ” No file Found,  Kindly Retry”



3.Invalid Opcode:

    If the user uses an invalid opcode (opcode which is not mentioned in the opcode table), then it will show the following error:

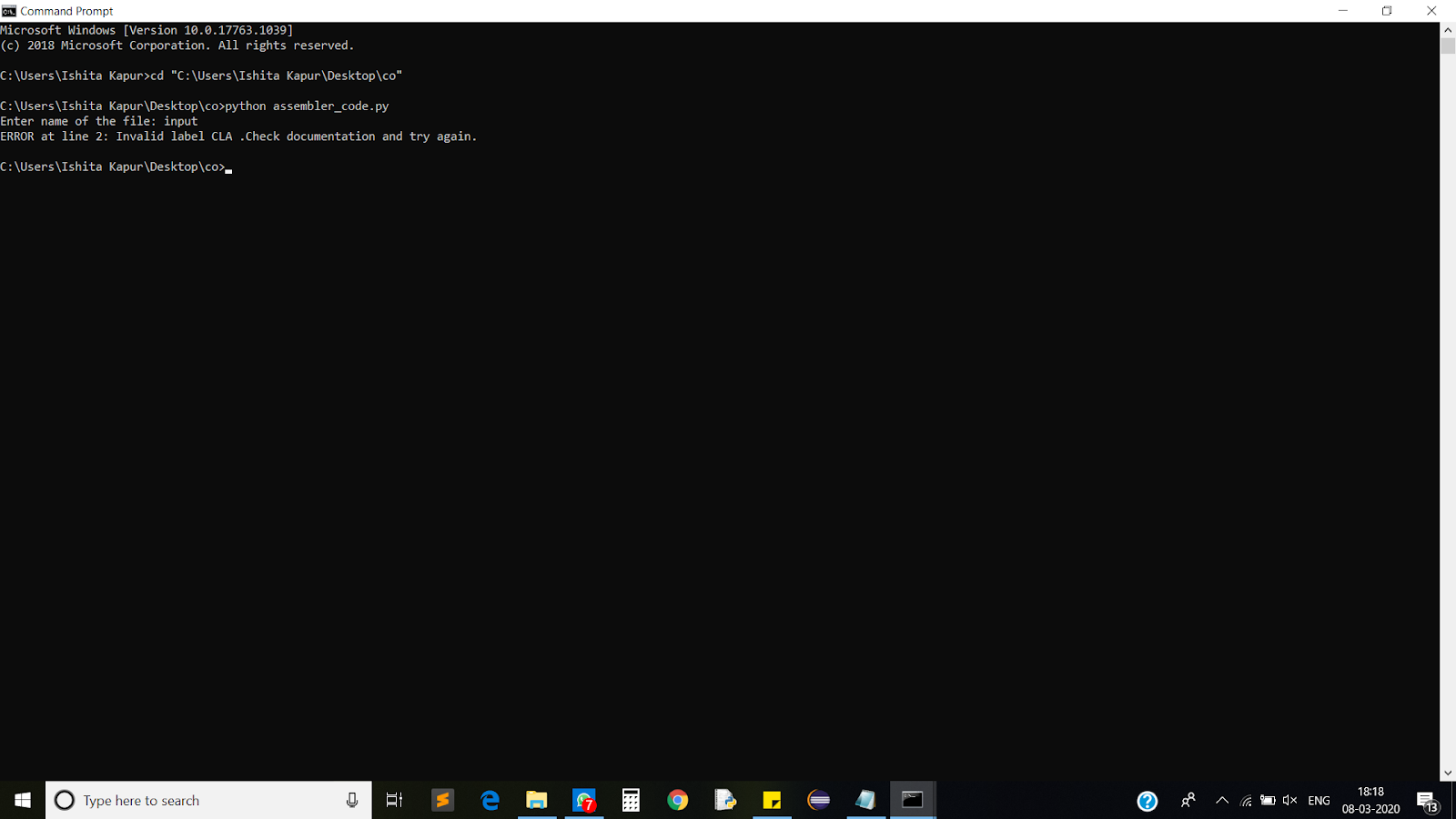
“ERROR at the line(number): Invalid assembly opcode ‘(invalid opcode)’. Check documentation”

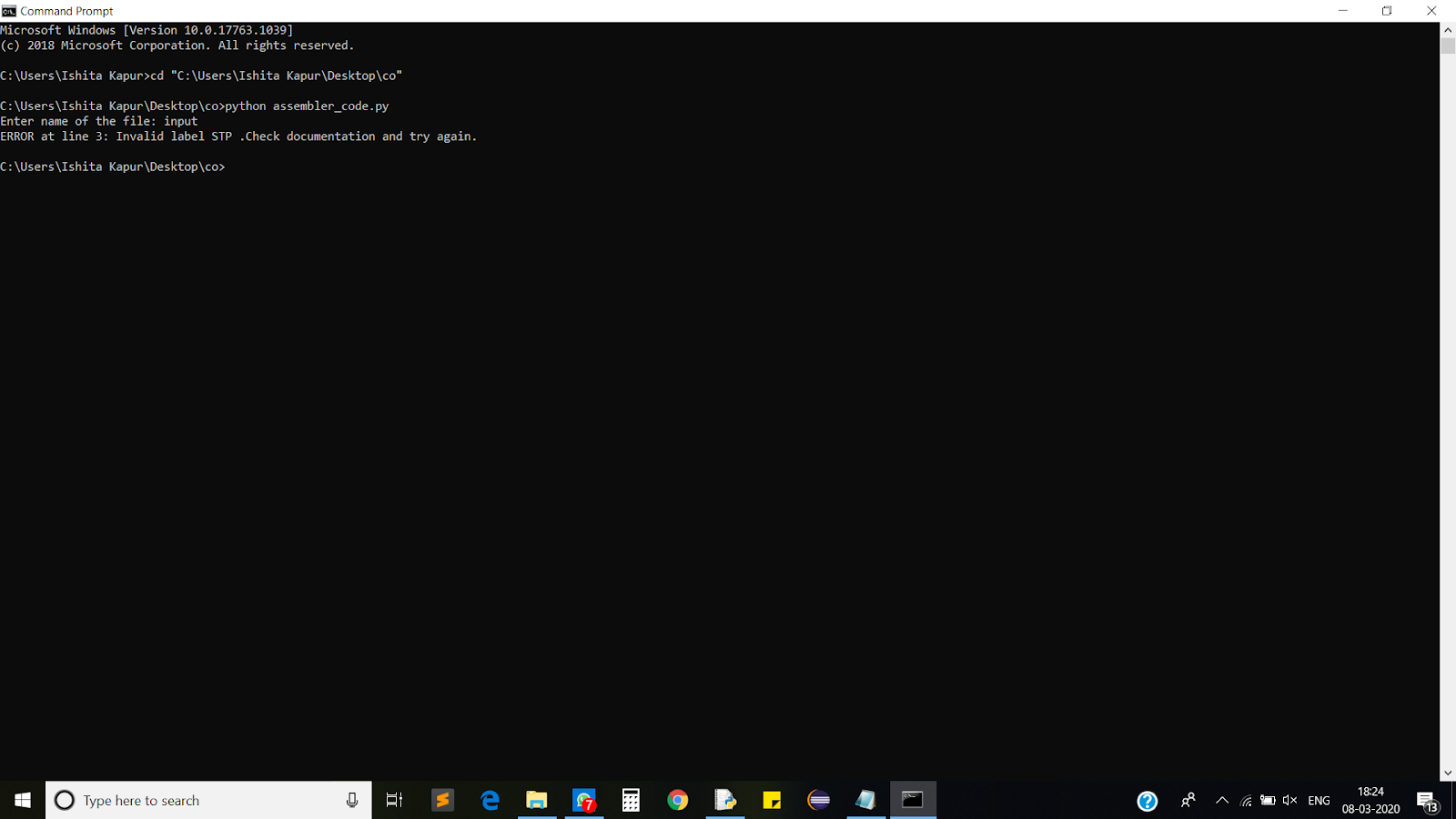


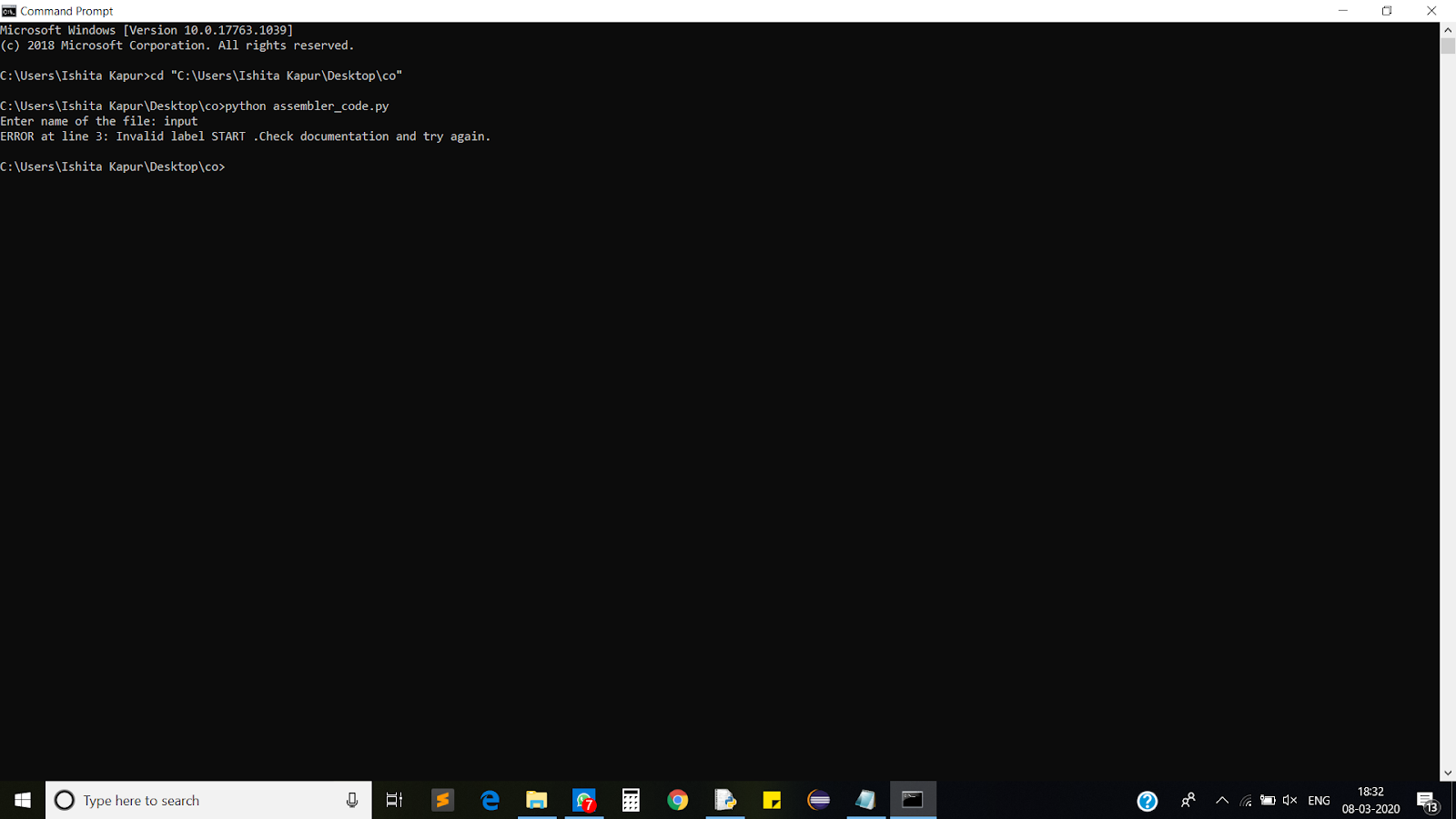
4. Invalid Label:

    If the user uses a label which is an assembly directive or an opcode or the same as a variable, then an error will be shown-

“ERROR at line (line number): Invalid label (label). Check documentation and try again.”



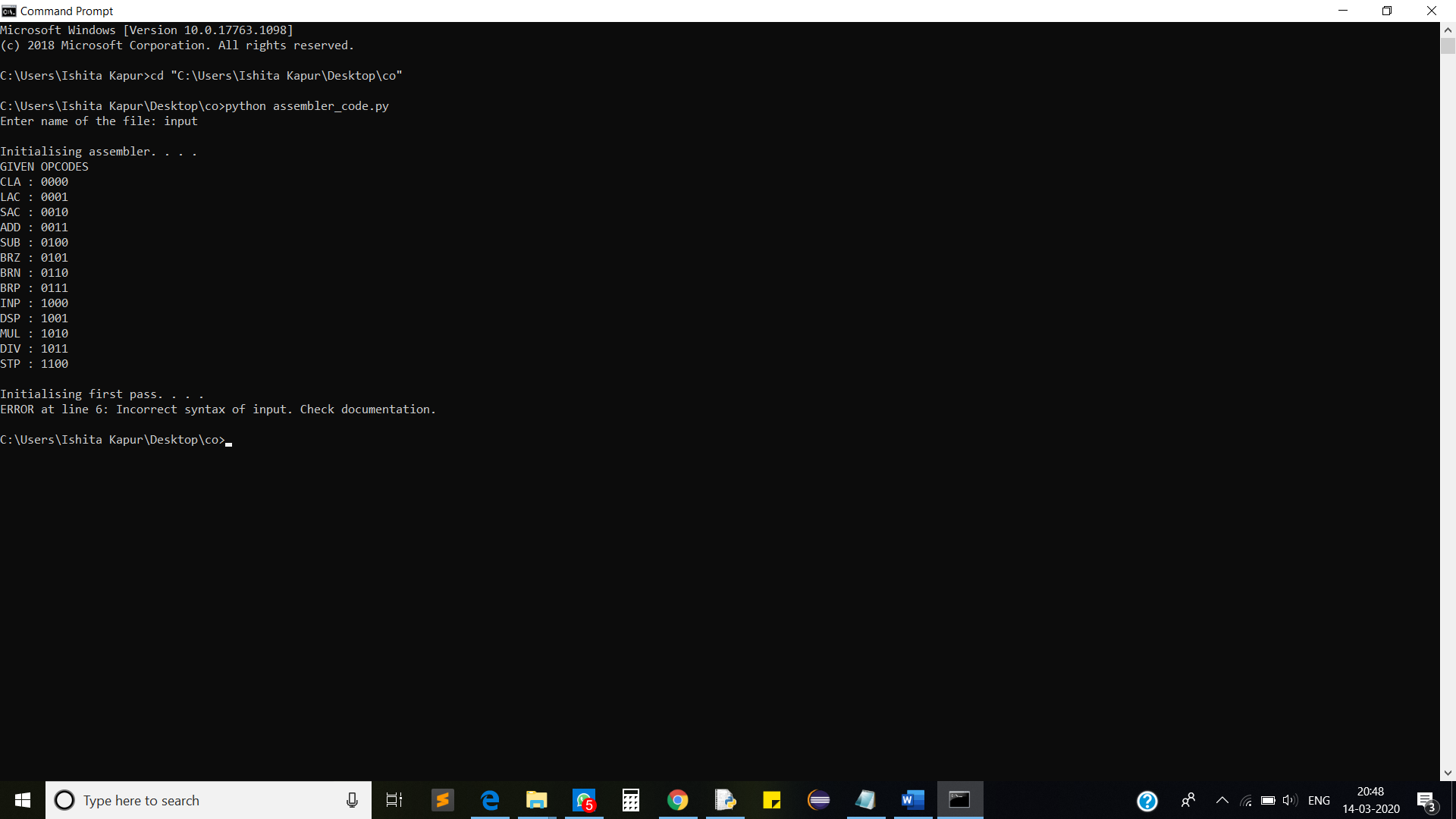




5. If ‘STP’ is not present in the program:

   If ‘STP’ is not present, then it will show the error:

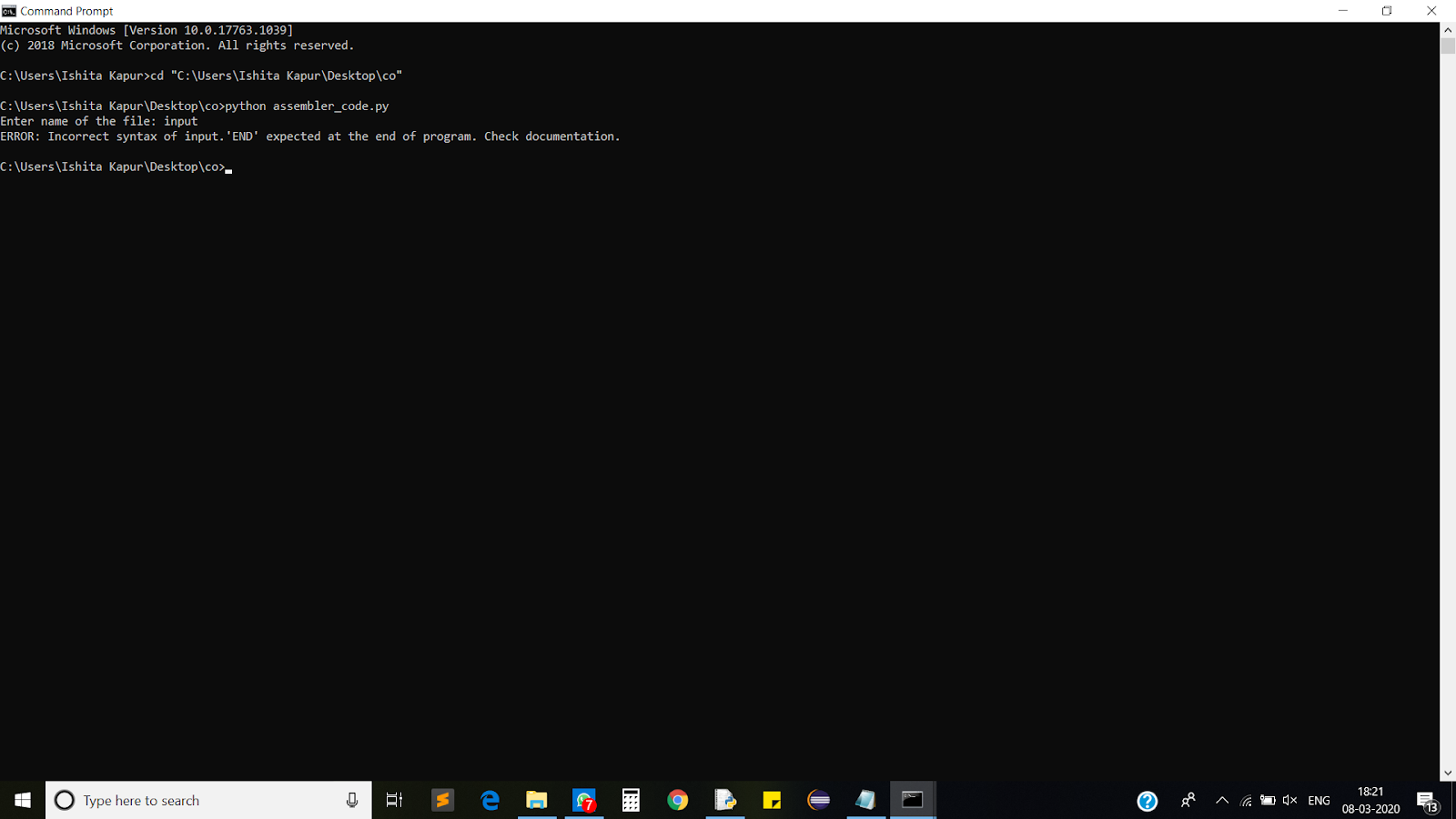
   “ERROR at line (line number): Incorrect syntax of input. Check documentation.”



6. Missing of “END” statement:

    If the “END” statement is missing in the program, then it will show the following error-

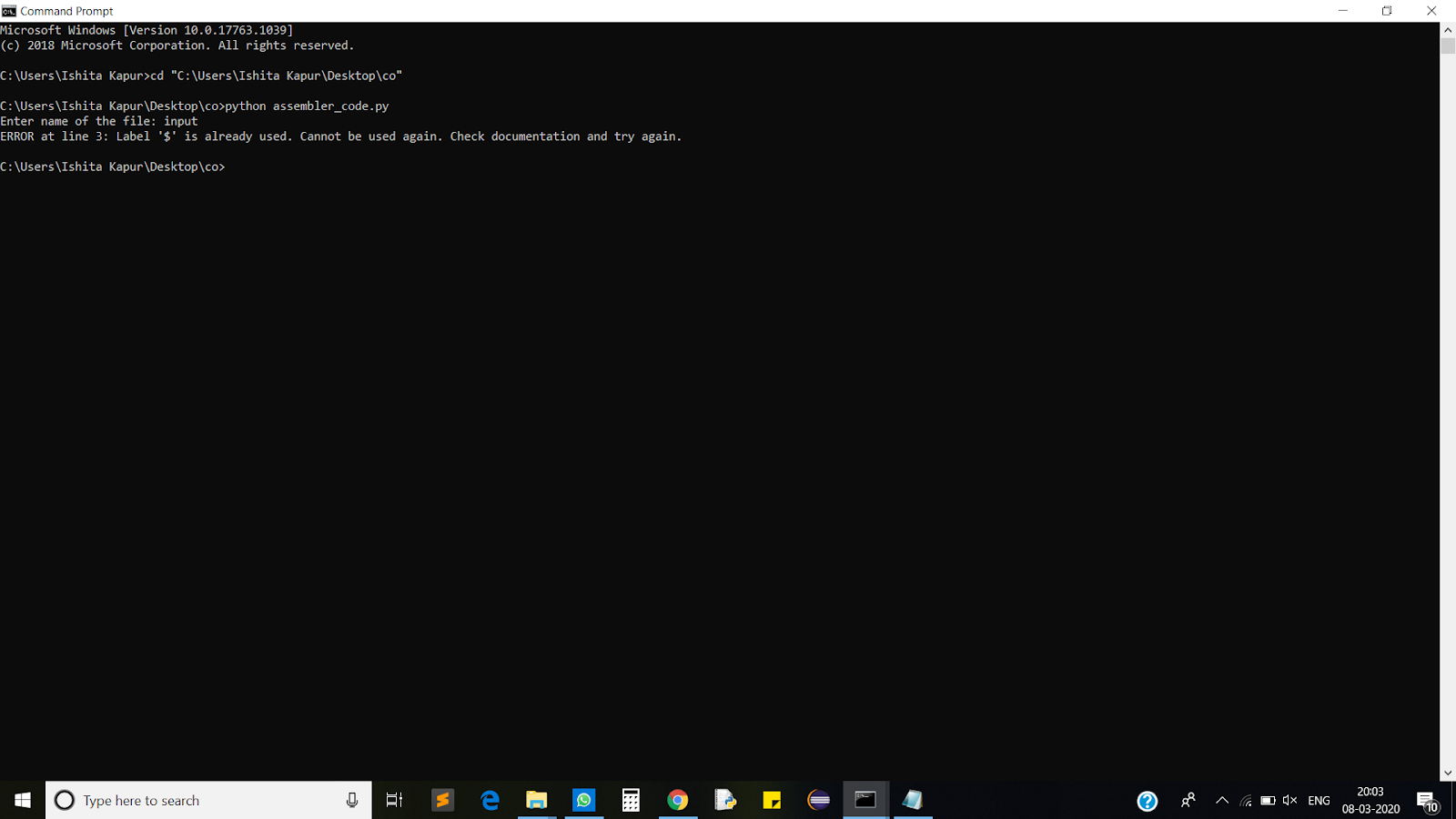
    “ERROR: Incorrect syntax of input. 'END' expected at the end of the program. Check documentation."



7.A label is declared more than once:

   If a label is declared more than one time, then it will show the following error:

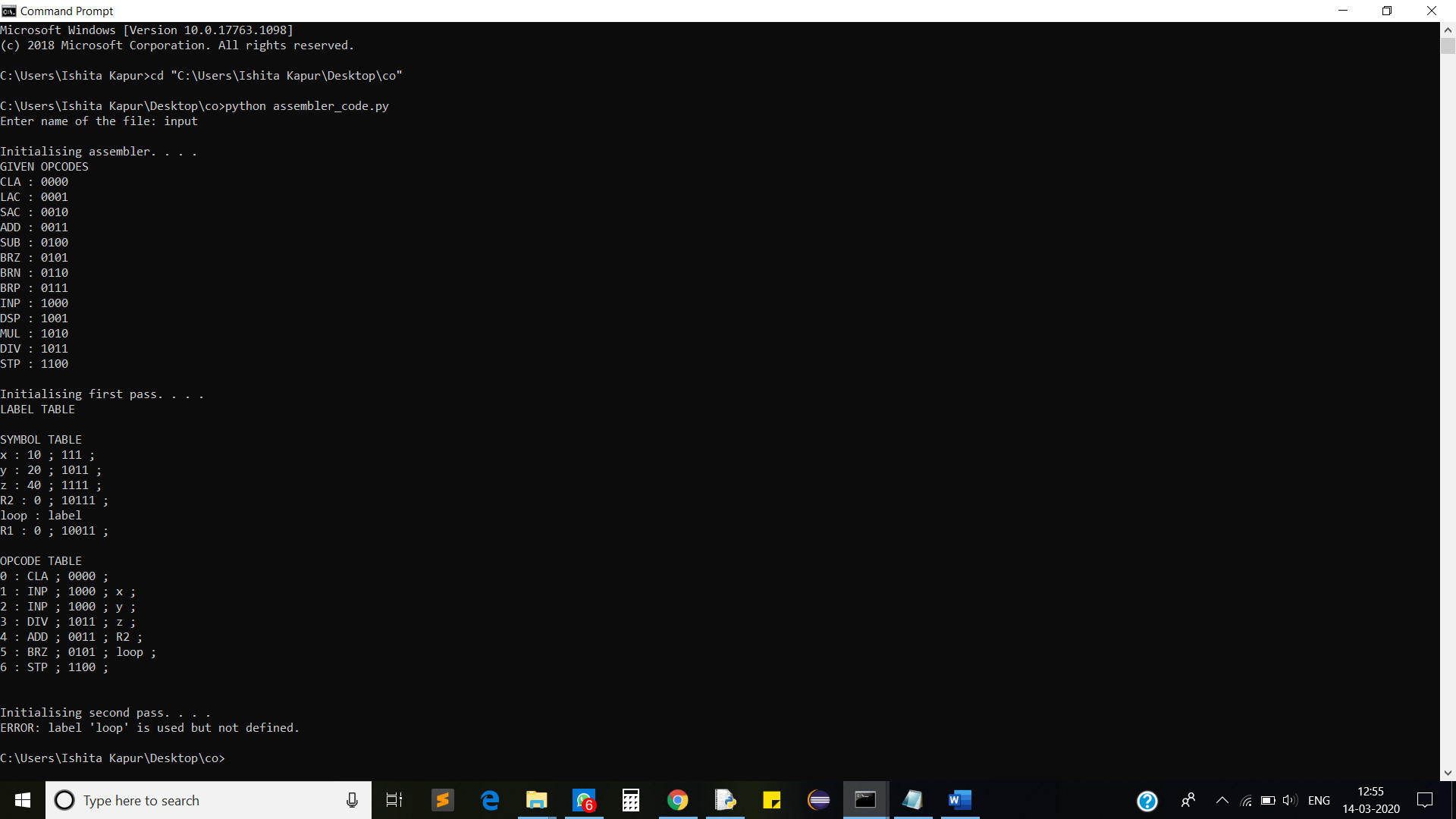
   "ERROR at line (line number): Label (label used) is already used. Cannot be used again. Check documentation and try again."



8. The label is not defined:

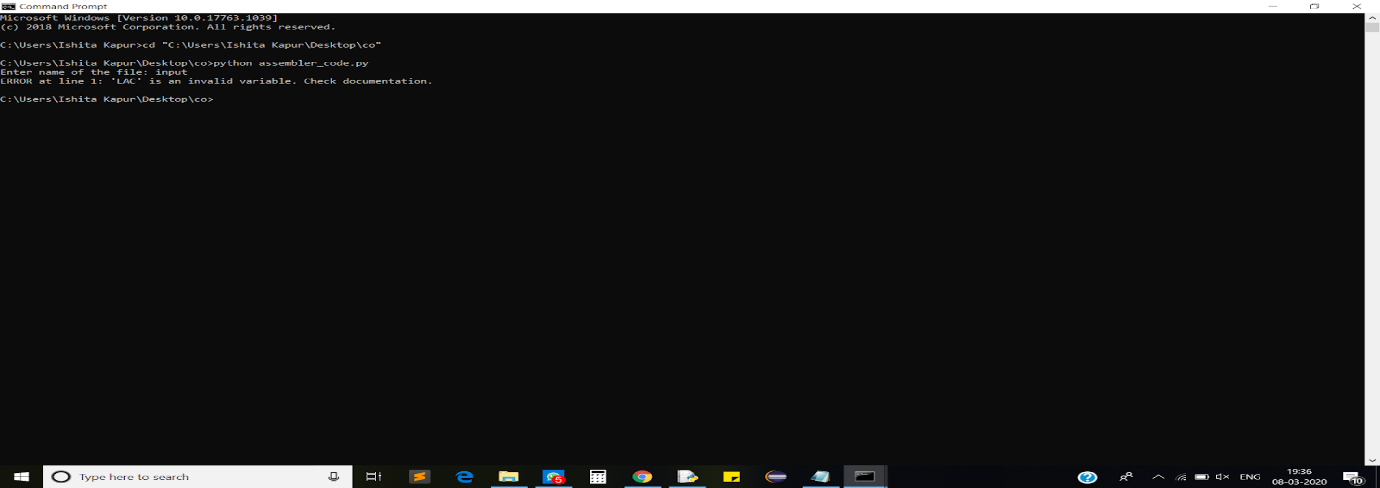
    When we use BRZ/BRN/BRP, it is necessary to define a label (as shown in the picture below, where the loop is a label). If the user does not provide a label, then it will show an error:

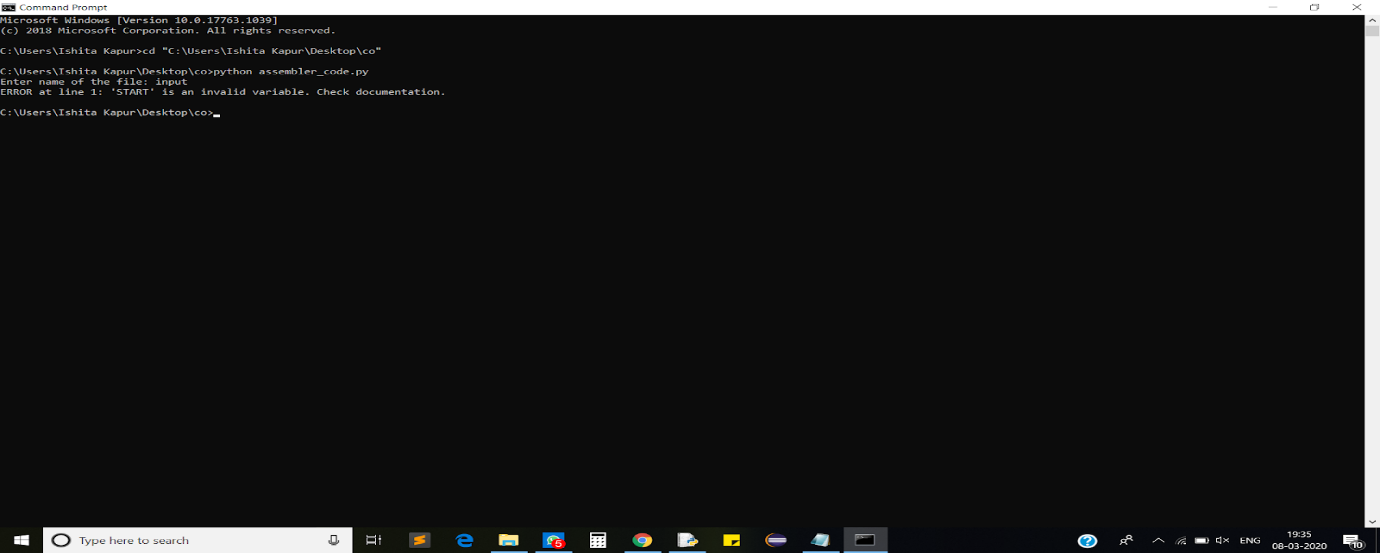
“ERROR : Label ‘loop’ is used but not defined."



9. Invalid Variable:

    If the user uses a variable which is already used in the opcode table or if the variable is “START” Or “END”, then it will show the following error:

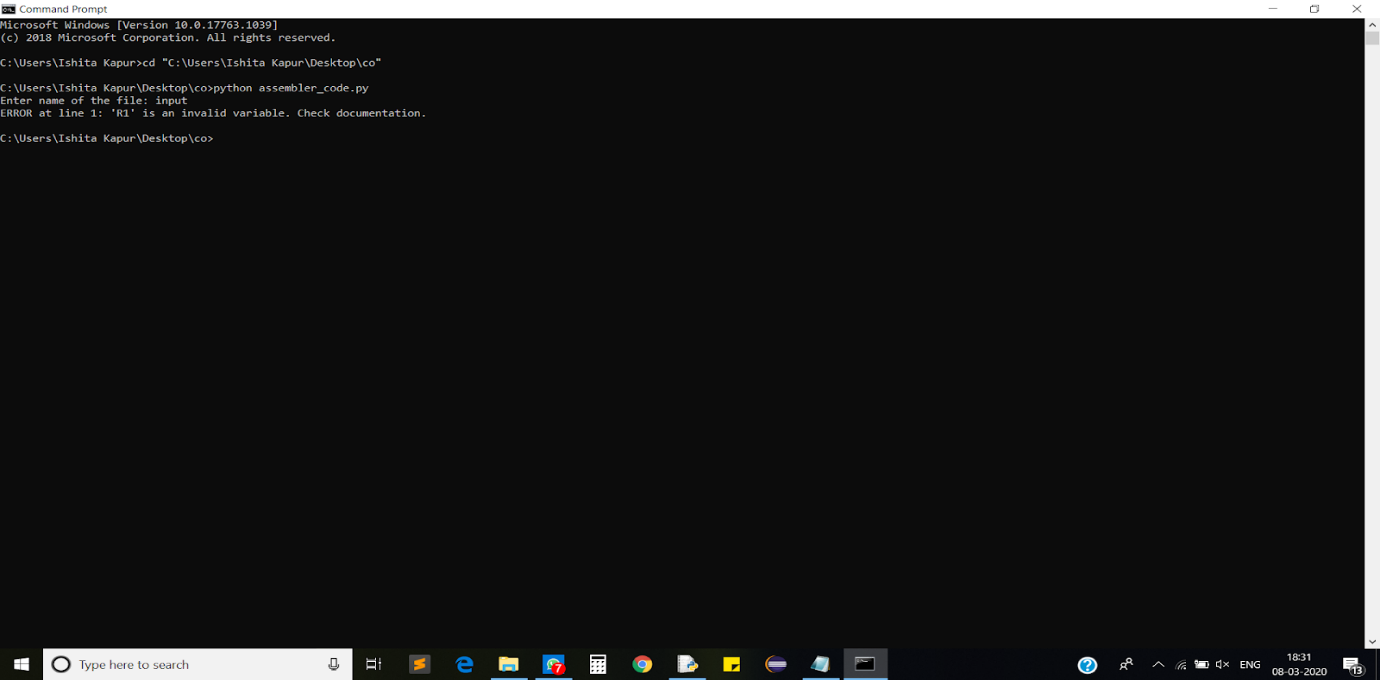




10. ERROR if R1 and R2 are used as variables without using DIV

    If R1 and R2 are used as variables without using DIV, then it will show an error. (R1 and R2 are used only when DIV(division) is used.)

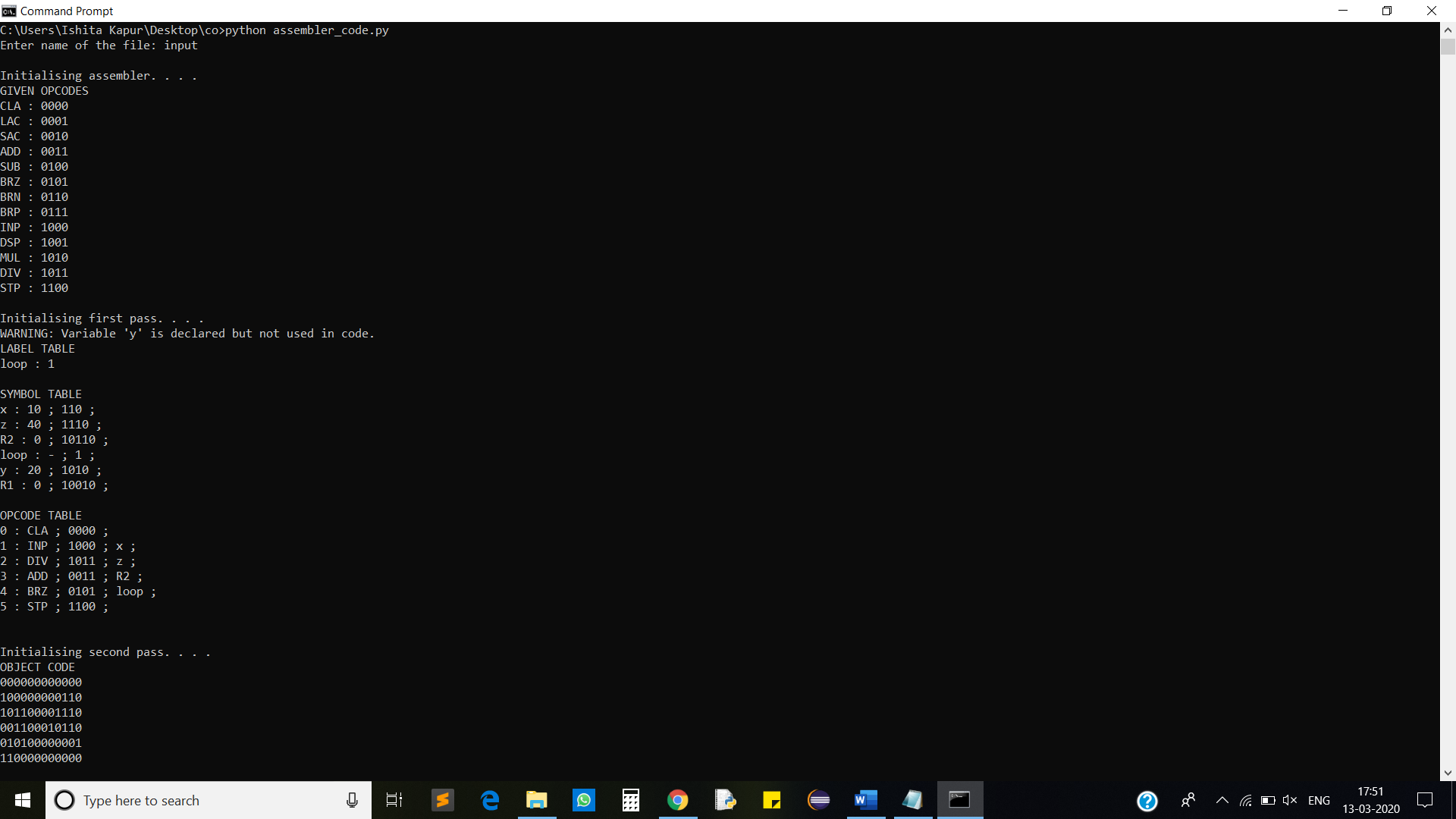
“ERROR ta line(line number) : ‘R1’/’R2’ is an invalid variable. Check documentation”



11. Variable is declared but not used:

    If a variable is declared in the program but it is not used in the program, then it will show the following warning- “WARNING: Variable(variable name) is declared but not used in code”

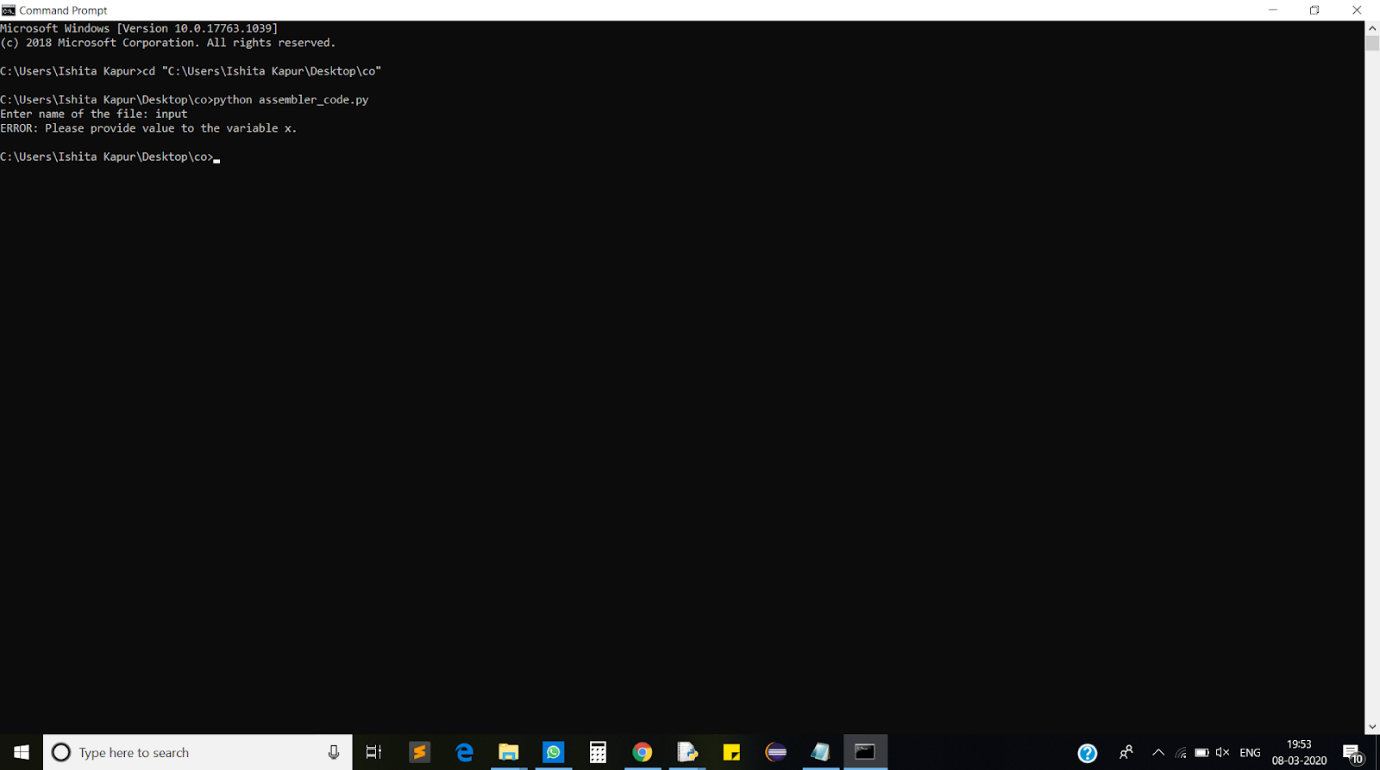
.



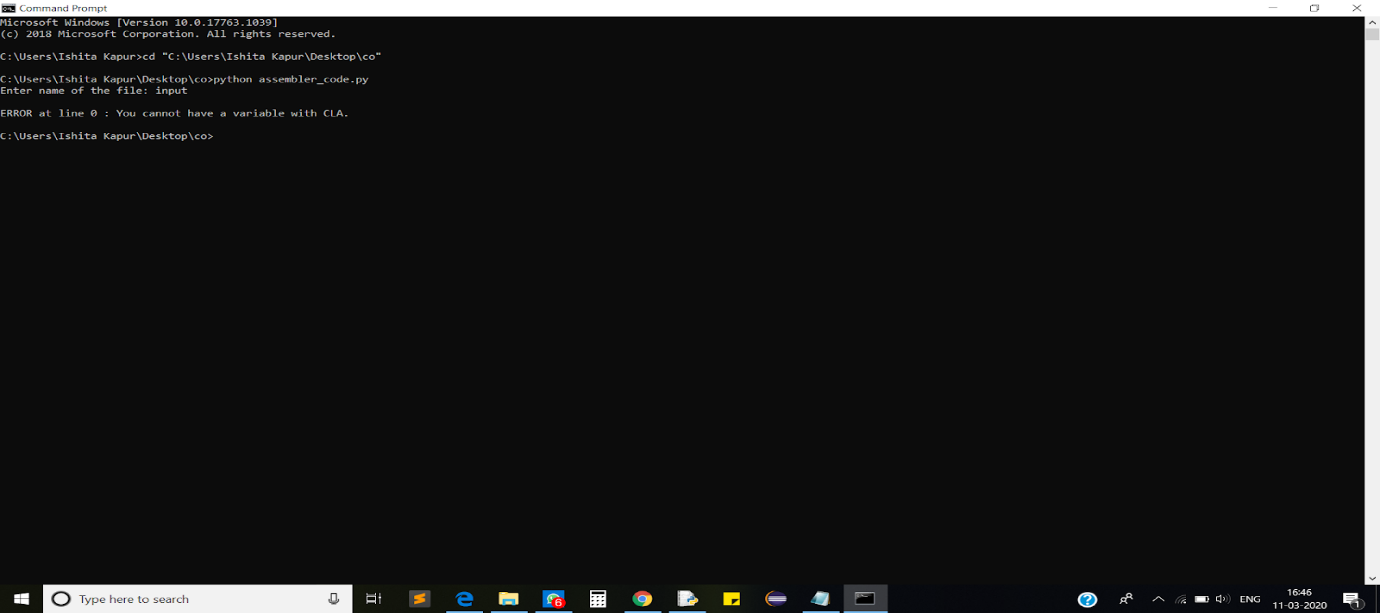
12. Variable is not provided with a value:

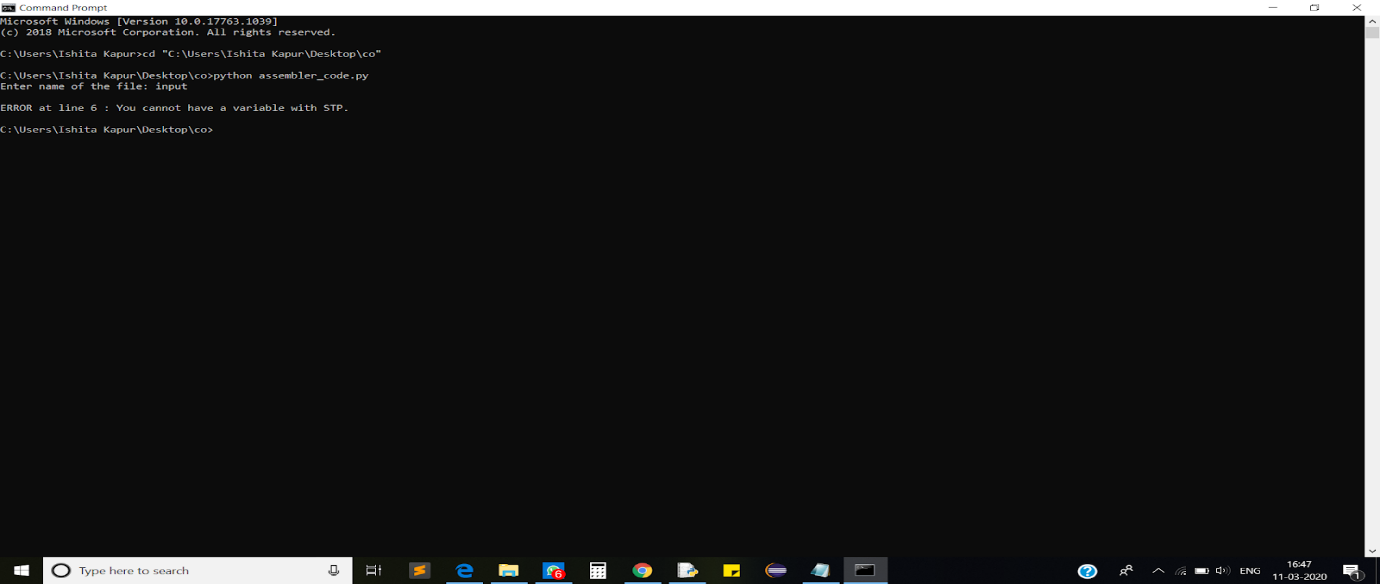
    If a variable is used in the program, then a value must be assigned to it, otherwise, it will

show the following error-” ERROR: Please provide value to the variable (variable name)."



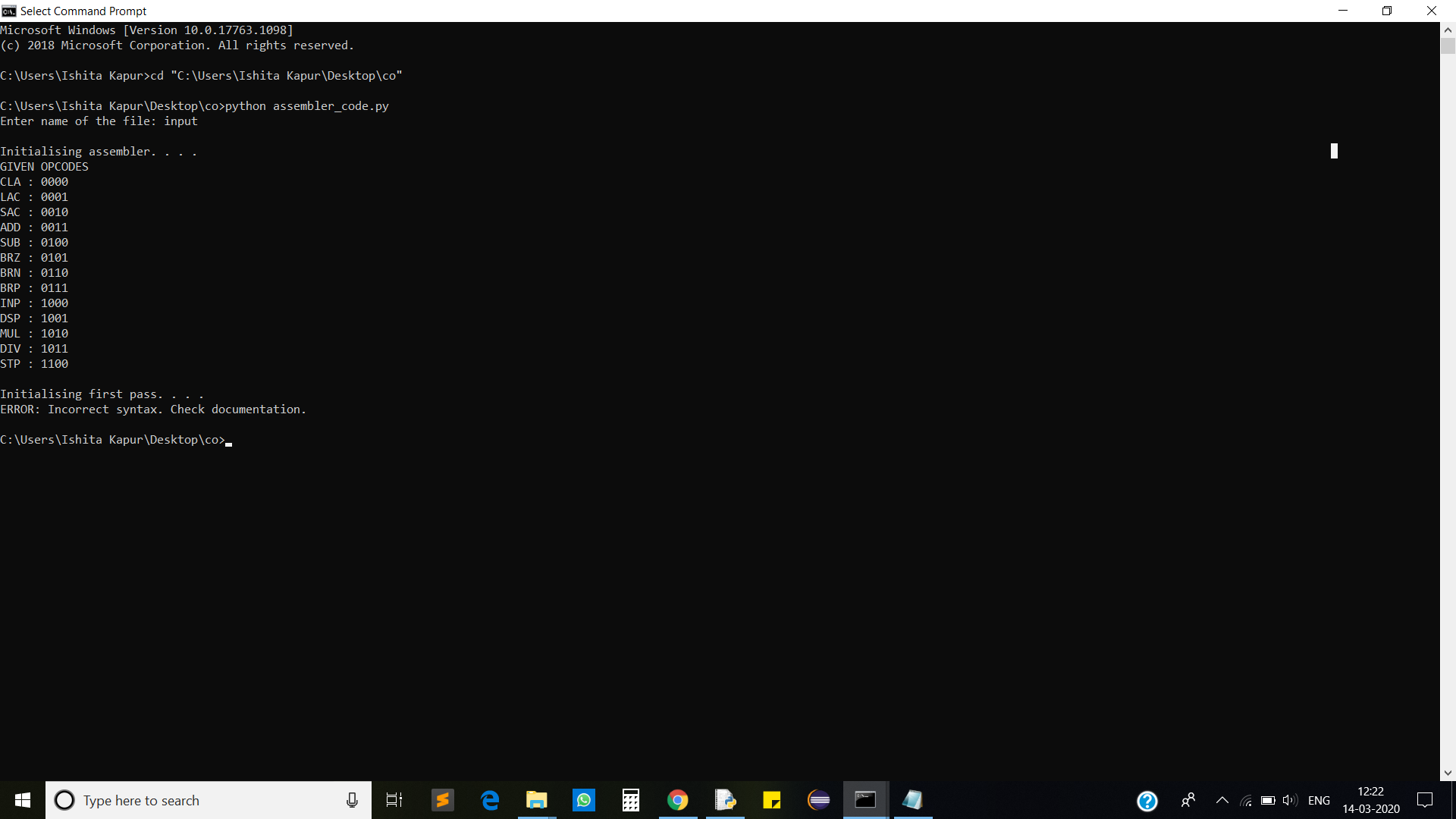
13.Variable with CLA and STP:  
       If the user uses a variable with ‘CLA’ or ‘STP’, then it will show an error.





14. Variable with ‘END’ statement:

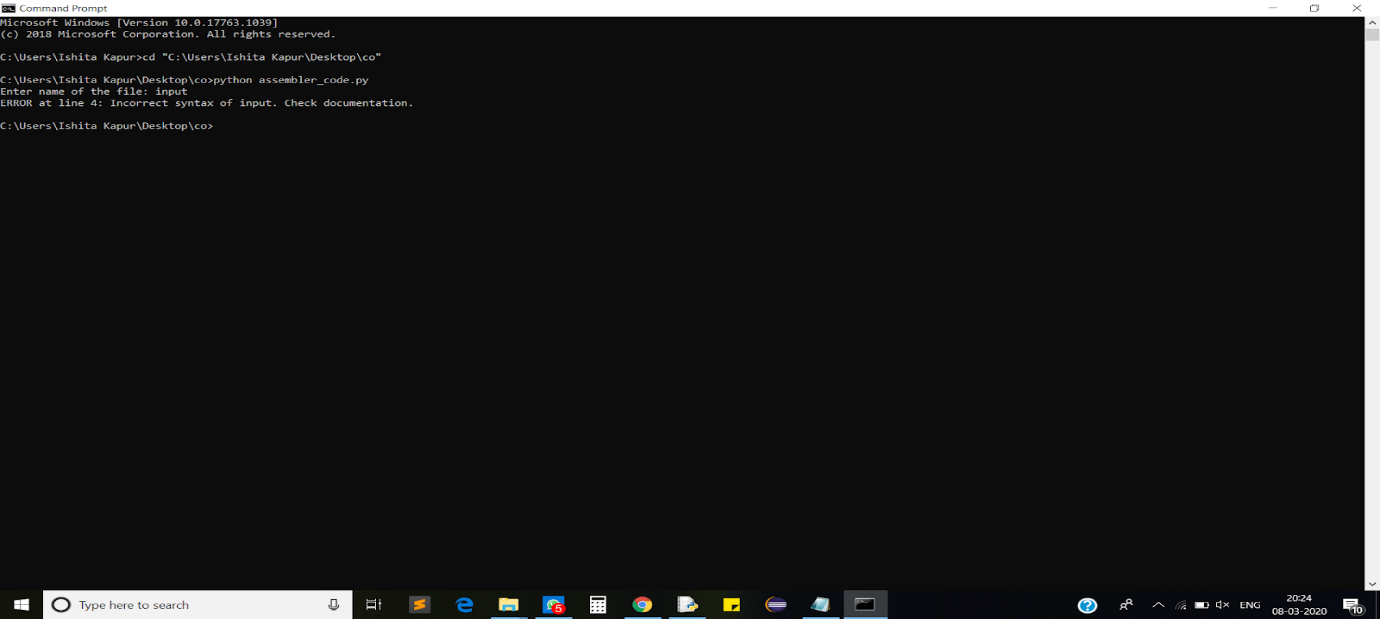
      If the user uses a variable with END statement or after the END statement, then it will show   the following error: “ERROR: Incorrect syntax. Check documentation.”



15. Variable is not provided to the Opcode:

      If the user does not provide any variable to an opcode which expects a variable, then it will show the following error:

      “ERROR at line (line number): Incorrect syntax of input. Check documentation.")

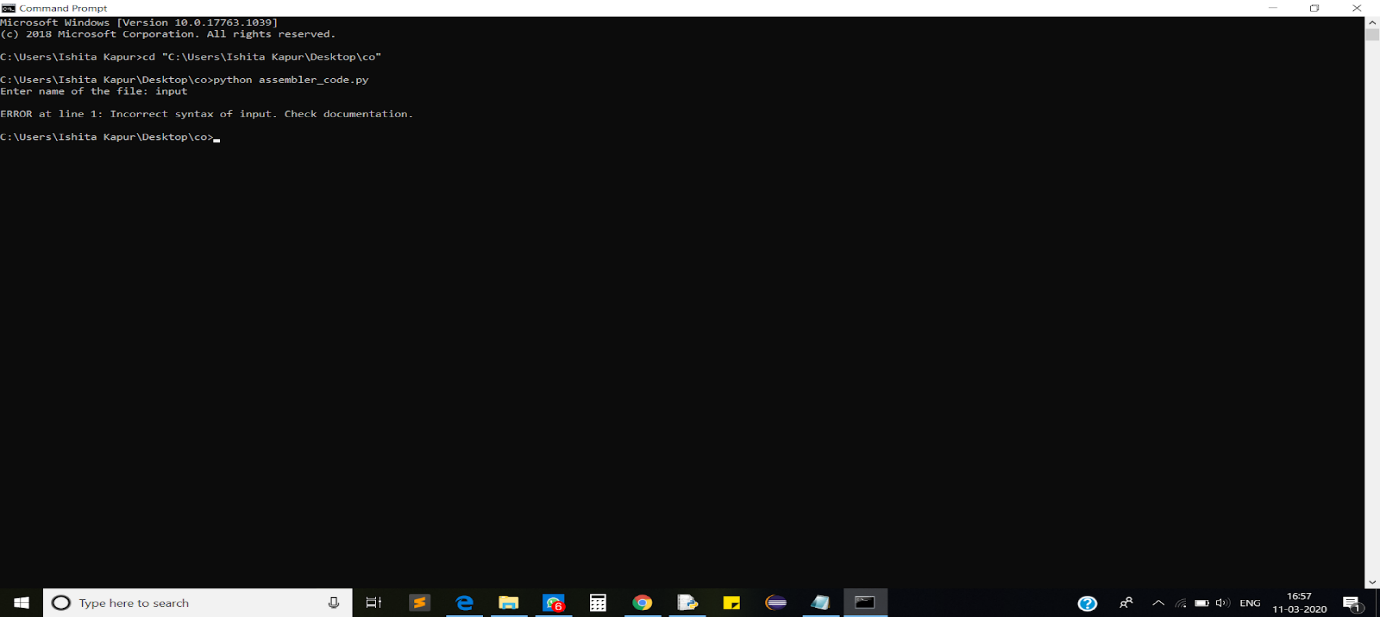


16. More than one variable used in one instruction

      If more than one variable is used in one instruction, then it will show an error:

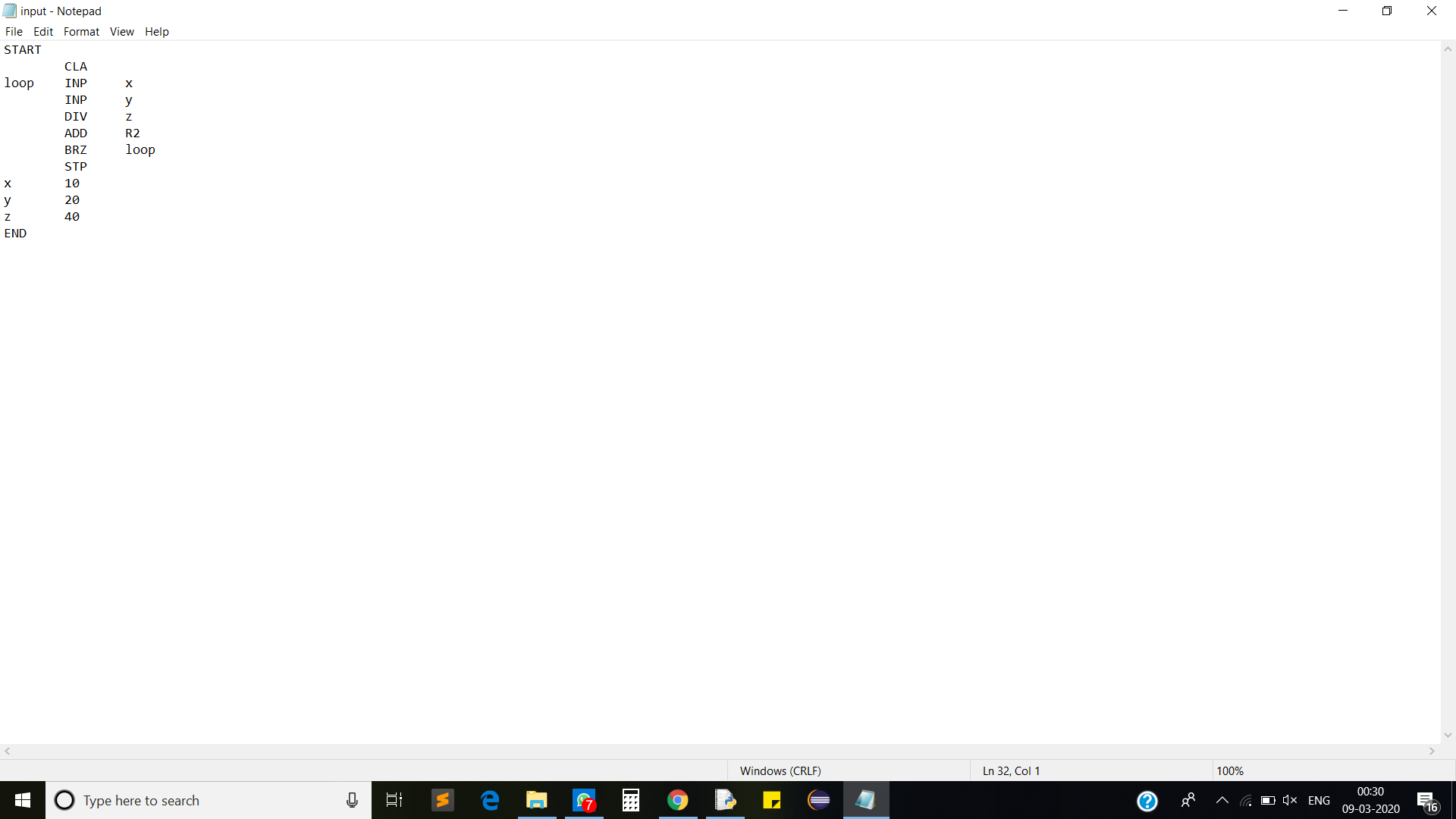
      "ERROR at line (line number) : Incorrect syntax of input. Check documentation."

(ASSUMPTION: There must be a tab between two variables)



**INPUT FILE:**

This is the sample input file.



Corresponding output:

