You explain the points you have added and I'll add somethinf about opcodes and how to use them.

CO PROJECT 1: DOCUMENTATION

This is a simple two pass assembler that takes in an Assembly level code written using basic opcodes and produces an object code in two passes. It is written in python.

Restrictions:

1. The opcode can not have more than one operand. Some opcodes like CLA has no operand at all, but Assembler throws an error when more than one operand is present in any case.
2. Each memory address can hold 12-bit data. Out of the 12 bits, 4 bits are reserved for opcodes and remaining 8 bits for operands.
3.

Errors handled:

- Duplicate label: This error is thrown when same label is defined more than once.
- Too many arguments: This error is thrown when an opcode is given more arguments than
- Not enough arguments: Thrown when an opcode is given lesser number of argument than that it was expecting.
- STP not found: Thrown when the end of file is reached in the first pass without the encounter of the 'STP' keyword.
- Label not found: Thrown when a label is used but has not been defined.
- Opcode not found: Thrown when opcode that is out of the scope of the assembler is used in the assembly code.

Assumptions

1. Label definitions are always followed by a colon to differentiate between labels and opcodes.
2. Comments can start with either a semicolon (;) or a hashtag(#). There can be inline comments too, but multiline comments are not handled.
3. If the instruction has a variable or a label instead of an actual memory location, it is assumed that the opcodes like "BRP", "BRN", "BRZ" are always followed by labels whereas rest of the opcodes are followed by variables.
4. A random number is generated that is assumed to be the starting physical address for the block of memory assigned to the object code generated by the assembler. The physical memory mapping is displayed on the terminal.

Data Structures Used:

1. Symbol Table:
   All the labels and symbols used in the assembly language is stored in the symbol table in the first pass.
   For labels, once the label definition is encountered, the location counter denoting the location of the label is saved to the symbol table in the first pass so that the assembler can refer to it in the second pass.
   For variables, the variable is stored in the table with NULL address. At the end of the first pass, the virtual location starting from the end of the assembly program is assigned to all the variables in the table. They are then used in the same way as labels in the second pass.

   Example:

Symbol	Value	Type
x	14	Variable
L1	5	Label

2. Opcode Table:

All the opcodes that are encountered during first pass is put into opcode table along with its binary code, Address and instruction length, which is considered to be one. The opcodes with no address are given “----” (NULL) in the address field.

opcode	Binary code	operandAddress	instruction length
CLA	0000	----	1

Sample input/ intermediate file/ object code

Sample input	Intermediate File [virtualAdd, Label, opcode, BinOpcode]	Sample Object code [virtualAdd, instruction]
START 100 CLA	0 ---- CLA ---- 1 ---- INP 157	0000000000 0000-------- 0000000001 100010011101

INP 157 INP 158 LAC 157 SUB x BRN L1 DSP x CLA BRZ L2 L2: DSP 158 CLA BRZ L2 L1: STP END	2 ---- INP 158 3 ---- LAC 157 4 ---- SUB x 5 ---- BRN L1 6 ---- DSP x 7 ---- CLA ---- 8 ---- BRZ L2 9 L2 DSP 158 10 ---- CLA ---- 11 ---- BRZ L2 12 L1 STP ----	0000000010 100010011110 0000000011 000110011101 0000000100 010000001110 0000000101 011000001100 0000000110 100100001110 0000000111 0000-------- 0000001000 010100001001 0000001001 100110011110 0000001010 0000-------- 0000001011 010100001001 0000001100 1100--------