Two-Pass Assembler

Two-Pass Assembler performs two passes over the assembly language source code. In the First Pass, it iterates over all the lines and creates few essential tables, namely, Symbol Table, Literal Table, and Data Table. In the Second Pass, the Assembler translates the assembly code to machine language code.

1.1 Instruction Format

Instructions in the assembly language include Labels, Opcodes, Operands, and Comments. Opcodes and Operands are mandatory, while Comments and Labels are optional. The following instruction format should be used for the assembly language statements.

Label – Opcode – Operand – Comment

Machine language code includes binary or hexadecimal instructions. The following format will be used for machine language statements.

Label – Opcode – Target Label – Register – Operand

Note : If any instruction doesn't contain a Label or a Target Label or a Register or an Operand, then it will have '00' in its respective column.

For example, consider the following assembly instruction.

Addition ADD X

The machine code for the above instruction will be,

05 0011 00 00 25

Note : Addresses considered in the example are totally random.

The following table depicts the allocation of bits for the machine language statements.

	BITS
Label	2
Opcode	4
Target Label	2
Register	2
Operand	2
	12

2.1 Outputs

The Two-Pass Assembler outputs an Opcode Table, a Symbol Table, a Literal Table, a Data Table, and the Machine Code for the input Assembly Code. It will also create a text file by the name of "Machine Code.txt" that consists of the machine language code for the input assembly language code.

A Symbol Table includes all the variables and labels used in the assembly source code, their values ("None" for labels), their types ("Label" or "Variable"), and their addresses.

A Literal Table consists of all the literals used in the assembly source code and their respective address.

A Data Table consists of all the declared variables and the values assigned to them.

Opcode Table includes all the valid assembly opcodes and their respective machine opcodes. The following table shows all the valid opcodes.

OPCODES	MEANING	ASSEMBLY OPCODES
0000	Clear accumulator	CLA
0001	Load into accumulator from address	LAC
0010	Store accumulator contents into the address	SAC
0011	Add address contents to 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 a negative value	BRN
0111	Branch to address if accumulator contains a positive value	BRP
1000	Read from the terminal and put in the address	INP
1001	Display value in the 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

3.1 Functions

The program includes some functions that help in checking for literals, symbols, labels, and dealing with the instructions (Removing commas and spaces to get the final list of all the elements in an instruction).

Here are the functions used in the program.

i. **RemoveSpaces()** : As the name suggests, this function helps in removing the redundant spaces from a list that also contain elements of instruction such as opcode and operands.

ii. **RemoveCommas()** : This function helps in removing commas from a list that also contains elements of instruction.

Note : The instruction string is first read and then split by spaces(" ") that make a list of elements that contain not only the required elements but also some unnecessary spaces and commas.

iii. **CheckLiteral()** : This function returns 'True' is the element passed in it is of the format **='x'**, where x is a numeric value (If the element is a Literal). Otherwise, it returns 'False'.

iv. **CheckSymbol()** : This function checks for Symbols in a list of elements of instruction. If any symbol is found, it returns 'True'; otherwise, it just returns 'False'.

v. **CheckLabel()** : This function checks for Labels in a list of elements of instruction. It also returns 'True' is any Label is found; otherwise, it goes with 'False'.

4.1 How To Run The Program

Follow the steps given below to successfully run the program and convert the input assembly language code to its machine language code.

i. Firstly, make sure that the input assembly code file is in the correct format, and instructions are correctly written and in the required format.

ii. Once the input file is sorted and in the correct format, go to the line number 48 in the Two-Pass Assembler python program and change the name of the input file to the name of your input file. By default, it is "Assembly Code Input.txt".

iii. Go to line number 298 and, if required, change the name of the output file to the name of your choice. By default, it is "Machine Code.txt". It will be the name of your text document that consists of the machine code.

Note : The input file and the python program should be in the same folder. After execution, the output file that contains the machine code will also be created in this folder.

5.1 Input File Format

Below are some points one should keep in mind while making the input file for the Two-Pass Assembler.

i. The input file should be a text document.
ii. The program should have a START and an END statement.
iii. The opcodes used should be valid.
iv. Instructions should be in the correct format. (Refer to Section 1.1)
v. Comments should either be at the end of the instructions or as a separate line, and they should begin with "//".

6.1 Errors

The Two-Pass Assembler comprises an error reporting feature. If any error is found in the input assembly code, the Assembler outputs it in the console. The Assembler will output the error and end the program.

The Assembler can hunt for the following errors:

i. **STARTError** : If the START statement is missing.
ii. **TooManyOperandsError** : If the number of operands provided for an opcode exceeds the required number of operands.
iii. **LessOperandsError** : If the number of operands provided for an opcode is less than the required number of operands.
iv. **InvalidOpcodeError** : If an invalid opcode is used in the assembly language code.
v. **DefinationError** : If a variable is defined multiple times.
vi. **UndefinedVariableError** : If a variable is used in the program but not defined.
vii. **ENDError** : If the END statement is missing.
viii. **RedundantDeclarationError** : If a variable is declared but not used anywhere in the program.

7.1 Example

To show how the Two-Pass Assembler works, we will consider a sample assembly language code and translate it to the machine language code.

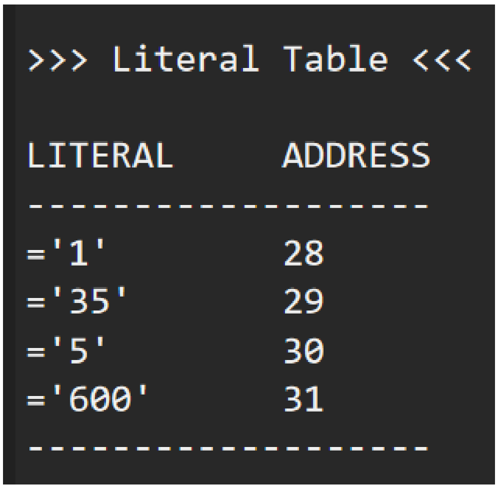
```
              START
// Comment Number One
// Comment Number Two
LoopOne       CLA
              LAC      A
              ADD      ='1'
              SUB      ='35'
Loop          BRP      Subtraction      // Comment Number Three
Subtraction   SUB      ='5'
              ADD      B                // Comment Number Four
              MUL      C
              SUB      D
              MUL      ='600'
              BRZ      Zero             // Comment Number Five
Division      DIV      E
              CLA
              LAC      REG1
              BRP      Positive
Zero          SAC      X
              DSP      X
              STP
Positive      CLA
              DSP      REG1
              DSP      REG2
        A     DATA     250
        B     DATA     125
        C     DATA     90
        D     DATA     88
        E     DATA     5
        X     DATA     0
              END
```

Assembly Language Code

The above assembly code will be read by the Two-Pass Assembler and converted to the machine language code.

In the first pass, the Two-Pass Assembler will create various tables, namely, Symbol Table, Literal Table, Data Table, and Opcode Table, and display them in the console.

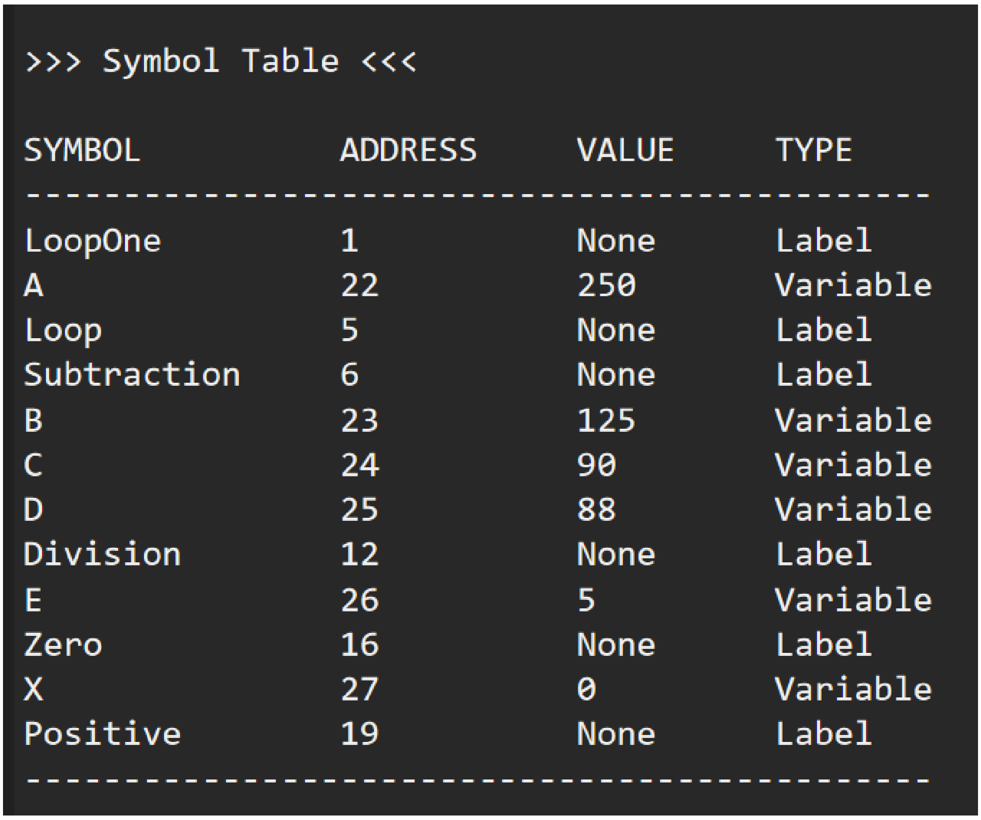
For the assembly code above, here are the tables.



```
>>> Literal Table <<<

LITERAL     ADDRESS
-------------------
='1'        28
='35'       29
='5'        30
='600'      31
-------------------
```

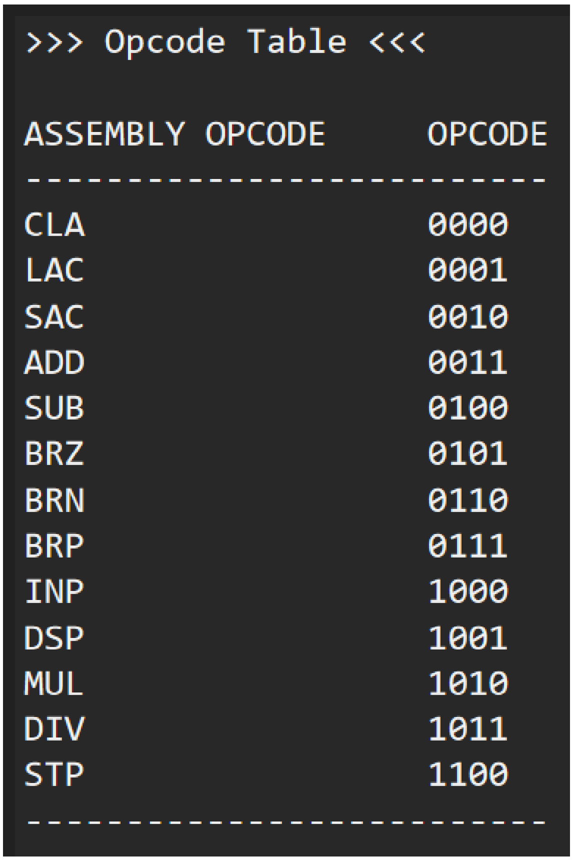
Literal Table



```
>>> Symbol Table <<<

SYMBOL          ADDRESS     VALUE     TYPE
----------------------------------------------
LoopOne         1           None      Label
A               22          250       Variable
Loop            5           None      Label
Subtraction     6           None      Label
B               23          125       Variable
C               24          90        Variable
D               25          88        Variable
Division        12          None      Label
E               26          5         Variable
Zero            16          None      Label
X               27          0         Variable
Positive        19          None      Label
----------------------------------------------
```

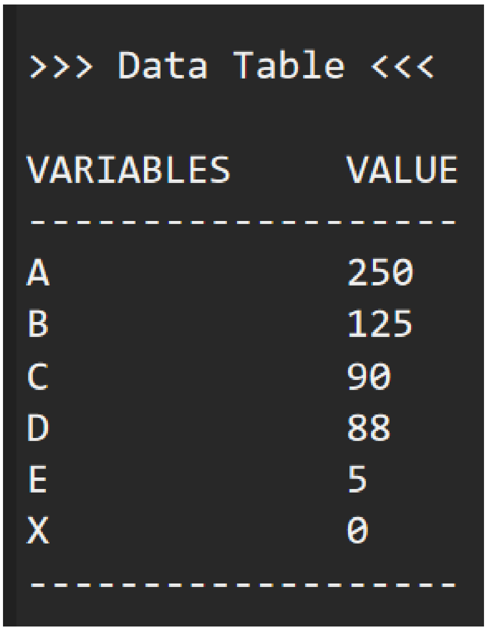
Symbol Table



```
>>> Opcode Table <<<

ASSEMBLY OPCODE     OPCODE
--------------------------
CLA                 0000
LAC                 0001
SAC                 0010
ADD                 0011
SUB                 0100
BRZ                 0101
BRN                 0110
BRP                 0111
INP                 1000
DSP                 1001
MUL                 1010
DIV                 1011
STP                 1100
--------------------------
```

Opcode Table



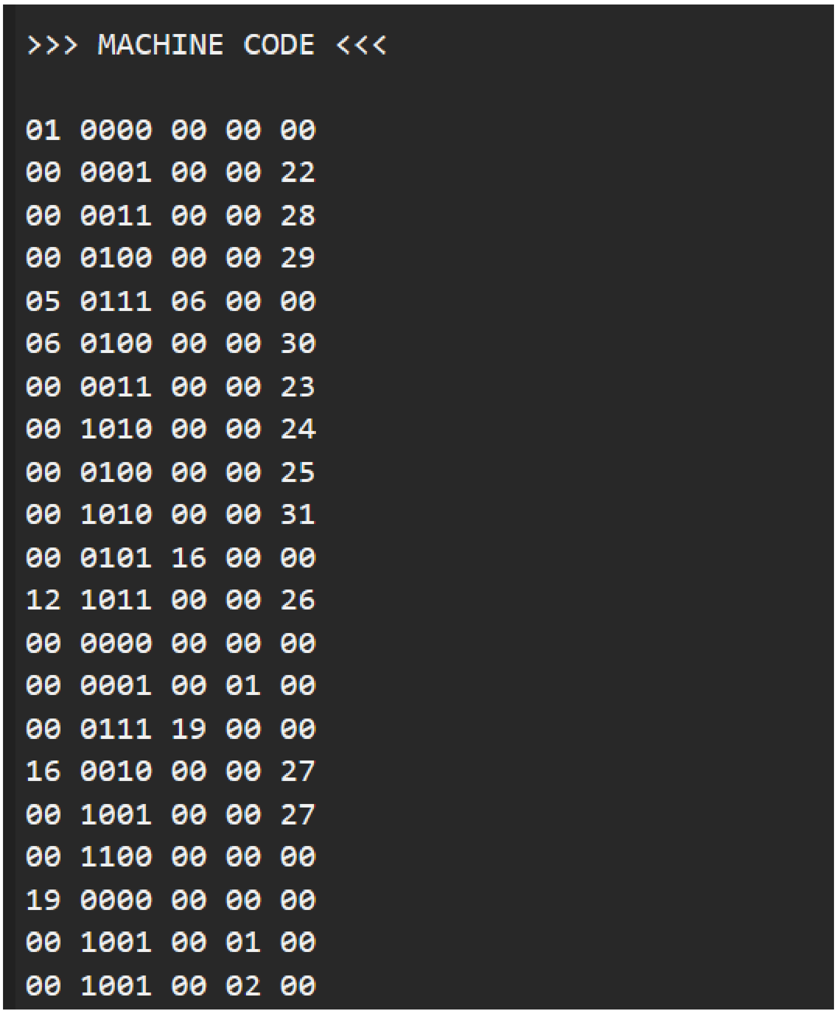
```
>>> Data Table <<<

VARIABLES     VALUE
-------------------
A             250
B             125
C             90
D             88
E             5
X             0
-------------------
```

Data Table

In the second pass, with the help of these tables, the assembly code will be converted into the machine code. The Two-Pass Assembler will not only output the machine code in the console but also create a text file that has the machine code.

Here is the machine code for the above assembly language code.



```
>>> MACHINE CODE <<<

01 0000 00 00 00
00 0001 00 00 22
00 0011 00 00 28
00 0100 00 00 29
05 0111 06 00 00
06 0100 00 00 30
00 0011 00 00 23
00 1010 00 00 24
00 0100 00 00 25
00 1010 00 00 31
00 0101 16 00 00
12 1011 00 00 26
00 0000 00 00 00
00 0001 00 01 00
00 0111 19 00 00
16 0010 00 00 27
00 1001 00 00 27
00 1100 00 00 00
19 0000 00 00 00
00 1001 00 01 00
00 1001 00 02 00
```

Machine Language Code

8.1 Assumptions

The following are the assumptions taken under consideration while making this Two-Pass Assembler.

i. The Assembly Language code always starts with a START statement, ends with the END statement, and the initial value of the location counter is always zero.

ii. All the operands are declared at the end of the assembly instructions, sequentially and in the order of their appearance in the code.

iii. There should be no Macros and Procs in the assembly language code.

iv. Literals are of the format **=’x’**, where x is a numeric value. For example, =’1’, =’45’, =’12345’, etc.

v. The programming language used for making this Two-Pass Assembler is python.