Project : 1
Two Pass Assembler

Team Members:
Navneet Agarwal (2018348)
Sarthak Arora (2018307)

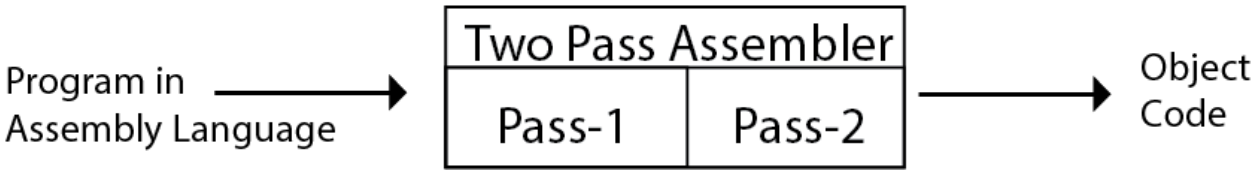
Documentation

The assembler is implemented in Python 3.7 and can be accessed by running the file "TwoPassAssembler.py"

Two Pass Assembler

An assembler translates an assembly program into equivalent object code.

Assembly language program has statements in the following format:
[label]<opcode><operand spec>[<operand spec>...]



In a Two pass assembler, the source file of assembly program is processed twice and then the final object code is received.

The processes involved are classified into two parts : Pass 1 and Pass2

During the first pass its collects all labels. During the second pass it produces the machine instruction and assigns address to each of them. It assigns addresses to labels by counting their position from the starting address.

Location Counter

Location counter, a variable that is used for providing addresses.
Length of each statement is added to LC after processing. While assigning the values, the present value at the time of processing gives the address associated with the corresponding label. The initial value of LC is specified in the START statement. In case, the initial value is not specified, LC is initialized to zero.

OPTAB

It contains mnemonic opcodes which shows the name of instruction along with its meaning and assembly opcode.

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 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 terminal and put in address	INP
1001	Display value in address on terminal	DSP
1010	Multiply accumulator and address contents	MUL
1011	Divide accumulator contents by address content. Quotient in R1 and remainder in R2	DIV
1100	Stop execution	STP

SYMTAB (symboltable.txt)

It contains symbols : labels and variables along with their addresses.

```
1    L1 109
2    L2 112
3    A 113
4    B 114
5
```

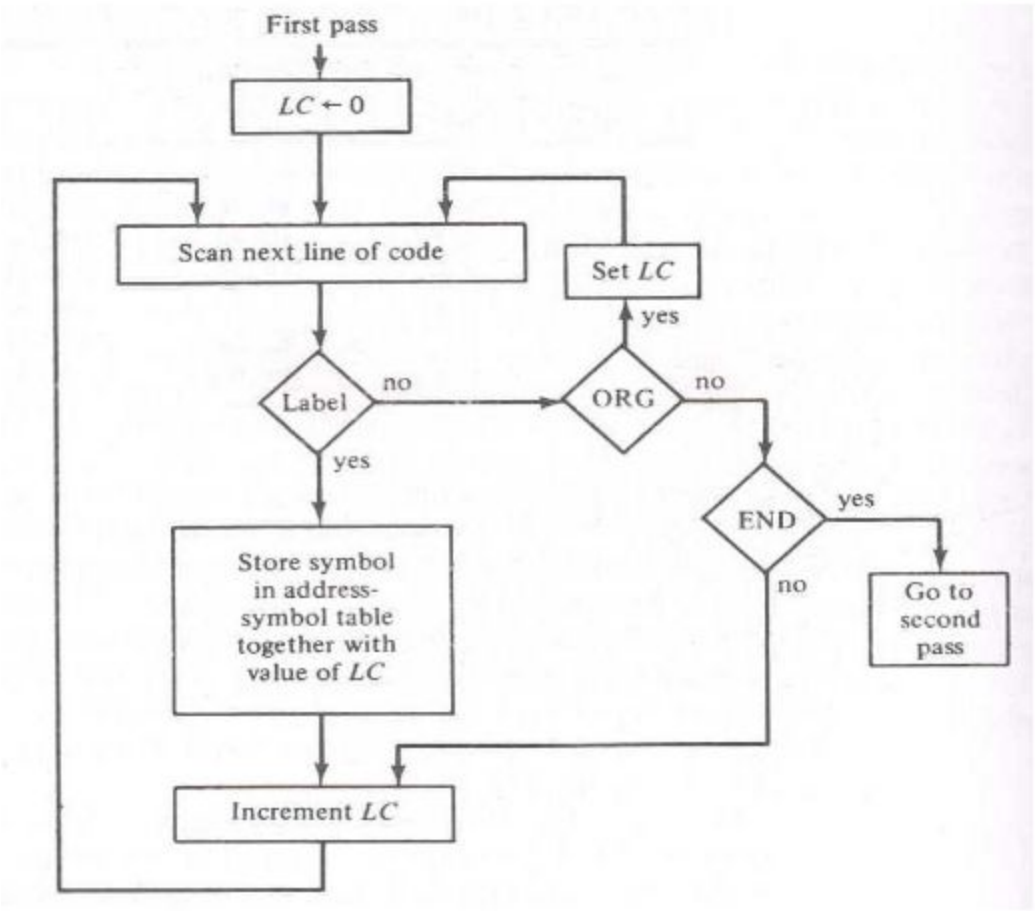
Symbol Table

```
1    START 100
2        CLA
3        INP A
4        INP B
5        LAC A
6        SUB B
7        BRN L1
8        DSP A
9        CLA
10       BRZ L2
11   L1: DSP B
12       CLA
13       BRZ L2
14   L2: STP
15   END
```

Sample Input

The First Pass

In the first pass, all statements in assembly language program are assigned addresses. These addresses are saved with labels and variables in symbol table for using them in the second pass.
Further, the length of machine instructions is determined and the location counter is kept updated accordingly.



Flow Chart for the first pass

Algorithm :

Step 1:

First of all, the first line is read,

If OPCODE = “START”, program begin with the following process

- Firstly, the address of START is stored. (It is the starting address of the program)
- LC is initialized the address passed with start.
- The line read is entered into an intermediate file which is used during the second pass.
- The next line of input is read.

Else

- The Location counter is initialized to zero.
- This suggests that starting address is not specified in the source program.

Step 2 :

The program proceed with a while loop,
While File has data:

- If Opcode is “END”
  >>Then the loop is broken then and there.

- If the line read is a comment line
  >>Ignore it and move to the next iteration of the loop.

StringToList() formats the data read from the file into a list.
Else

- If there is a symbol under the LABEL field then
  >>WriteLabelInSymbolTab() is called.
  >> Search the LABEL in the symboltable.txt
  >> If found, assign LC the address of the symbol.
  >>Else, Insert the address of the symbol to the symboltable.txt
- Now search the OPTAB for the OPCODE
  >>CheckForOptable() is called.
  >>If it is found and if the opcode is DIV then write the variable R1 and R2 in

symboltable.txt
>>Else If it is found and is any opcode other than CLA or STP then write the variable in symboltable.txt using function WriteInSymbolTab()
>>LocationCounter is incremented by 1.
>>Else, Error is returned.

The line read is written into intermediate line with the virtual address and the next line is read.
End of while.

The lines read are written into an intermediate file.
End of the First Pass.

Step 3 :

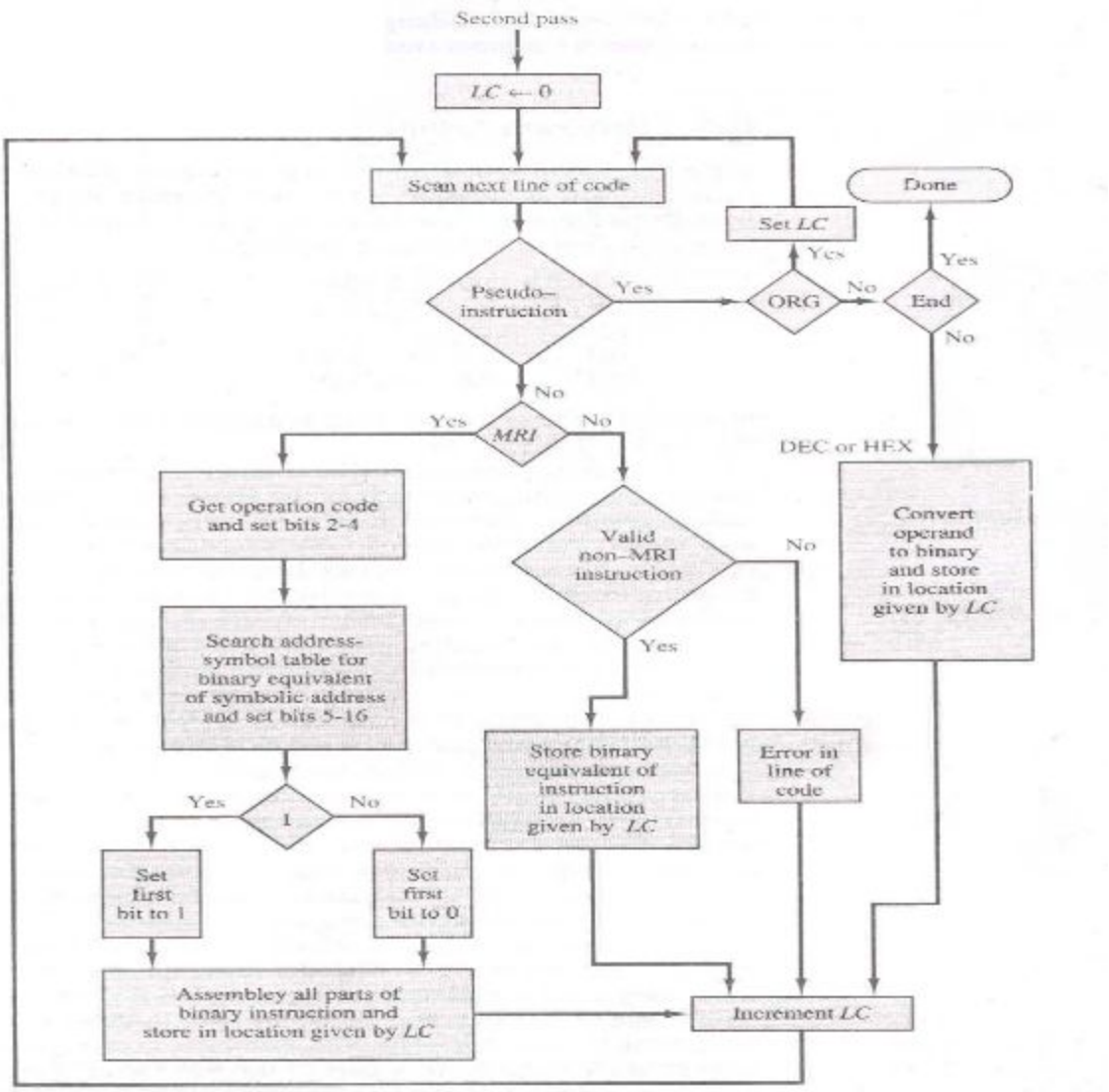
After the END has been reached we assign memory address to the variables through the function AssignMemoryAddressToVariables()

```
1     START 100
2   100 - CLA
3   101 - INP A
4   102 - INP B
5   103 - LAC A
6   104 - SUB B
7   105 - BRN L1
8   106 - DSP A
9   107 - CLA
10  108 - BRZ L2
11  109 L1 DSP B
12  110 - CLA
13  111 - BRZ L2
14  112 L2 STP
15    END
16
```

Intermediate result

The Second Pass

In the second pass, the processed intermediate result received from the first pass is further processed and converted to object code.



Flowchart of the second pass

Algorithm :

Step 1:

First of all, the first line is read,

If OPCODE = “START”, program begin with the following process

After this, the next input line is read.
Else
There was some error in Pass 1.

Step 2:

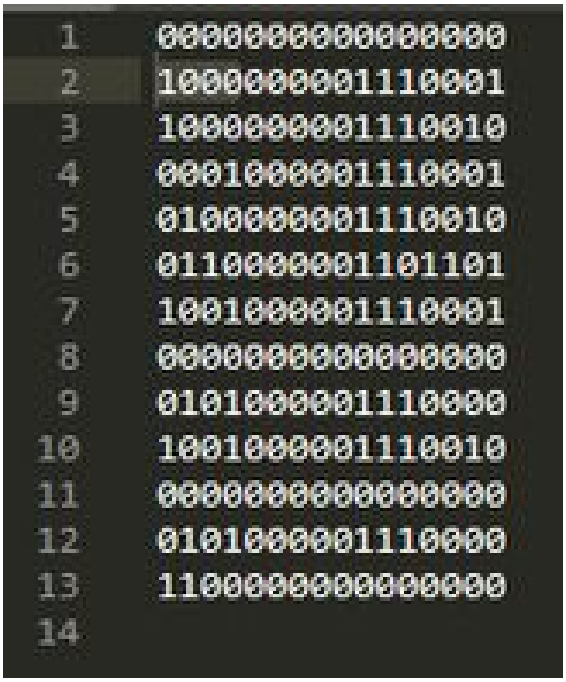
The program proceed with a while loop,

While File has data:

- If Opcode is “END”
  >>Then the loop is broken then and there.

- If the line read is a comment line
  >>Ignore it and move to the next iteration of the loop.

- IntermediateStringToList() is called and it formats the data read from the file into a list.
- After this, Opcode for the corresponding Assembly Opcode through the function GiveOpCode(), if the function returns -1 then it means its not a valid opcode and error is displayed.
- If the opcode is not CLA or STP, addresses of variable is fetched from symboltable.txt which are then converted to binary.
- Finally, the Instruction of 16 bits (opcode : 4 bits + address : 12 bits) is then written in the file objectcode.txt



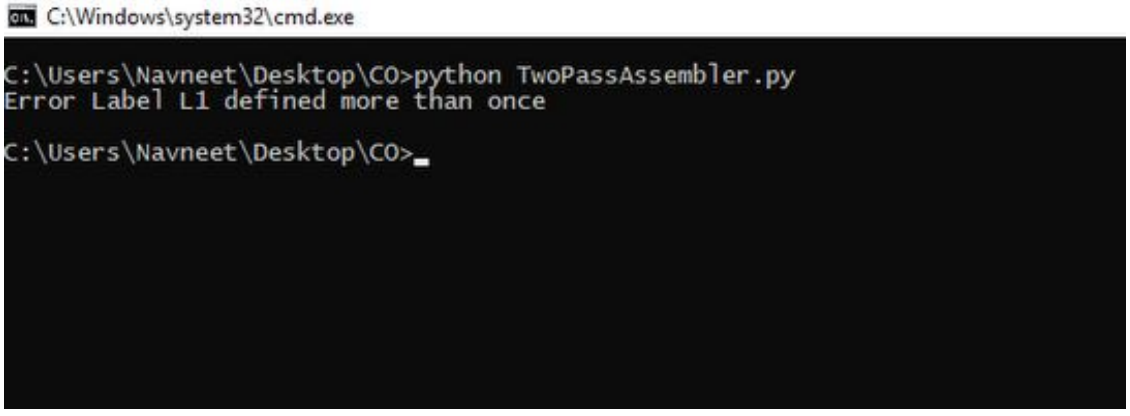
Result : Object Code

Error Reporting

Following types of errors are often encountered and are handled while making the two pass assembler. The assembler prints an error message along with the assembly line and continues with the assembly.

- More than one definition of label.

```
 1    START 100
 2        CLA
 3        INP A
 4        INP B
 5        LAC A
 6        SUB B
 7        BRN L1
 8        DSP A
 9        CLA
10        BRZ L2
11    L1: DSP B
12        CLA
13        BRZ L2
14    L2: STP
15    L1: CLA
16    END
17
```



- Not a legal OPCODE

```
1   START 100
2       CLA
3       INP A
4       INP B
5       LAC A
6       SUB B
7       BRN L1
8       DSP A
9       CLA
10      BRZ L2
11  L1: DSP B
12      CLA
13      BRZ L2
14  L2: STOPNOW
15  END
```

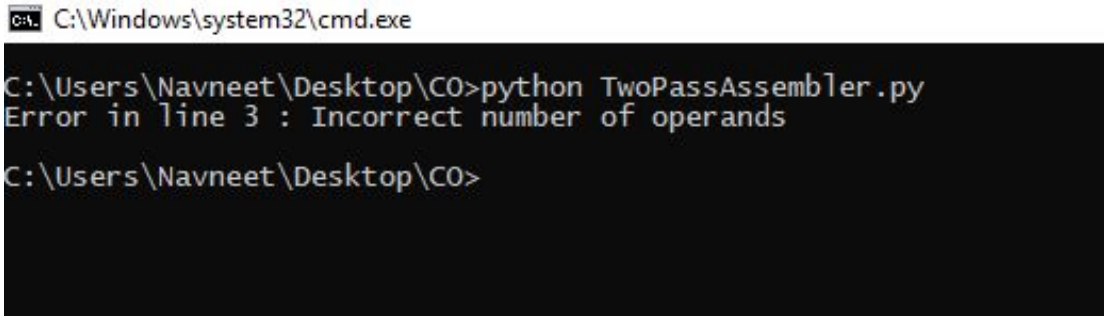
```
C:\Windows\system32\cmd.exe

C:\Users\Navneet\Desktop\CO>python TwoPassAssembler.py
Error in line 14:Not a Legal Opcode

C:\Users\Navneet\Desktop\CO>
```

- Less Operands supplied to Assembly OPCODE.

```
1   START 100
2       CLA
3       INP
4       INP B
5       LAC A
6       SUB B
7       BRN L1
8       DSP A
9       CLA
10      BRZ L2
11  L1: DSP B
12      CLA
13      BRZ L2
14  L2: STP
15  END
```



- More than required Operands supplied to Assembly OPCODE.

```
1    START 100
2        CLA
3        INP A B
4        INP B
5        LAC A
6        SUB B
7        BRN L1
8        DSP A
9        CLA
10       BRZ L2
11   L1: DSP B
12       CLA
13       BRZ L2
14   L2: STP
15   END
```

C:\Windows\system32\cmd.exe

```
C:\Users\Navneet\Desktop\CO>python TwoPassAssembler.py
Error in line 3 : Incorrect number of operands

C:\Users\Navneet\Desktop\CO>python TwoPassAssembler.py
Error in line 3 : Incorrect number of operands

C:\Users\Navneet\Desktop\CO>
```

- END statement is missing in the assembly program.

```
1    START 100
2        CLA
3        INP A
4        INP B
5        LAC A
6        SUB B
7        BRN L1
8        DSP A
9        CLA
10       BRZ L2
11   L1: DSP B
12       CLA
13       BRZ L2
14   L2: STP
15
```

C:\Windows\system32\cmd.exe

```
C:\Users\Navneet\Desktop\CO>python TwoPassAssembler.py
Error END statement missing

C:\Users\Navneet\Desktop\CO>_
```

- An empty Line encountered in assembly code.

```
1    START 100
2        CLA
3        INP A
4        INP B
5        LAC A
6        SUB B
7        BRN L1
8        DSP A
9        CLA
10       BRZ L2
11   L1: DSP B
12       CLA
13       BRZ L2
14   L2: STOPNOW
15
16
17   END
```

```
C:\Windows\system32\cmd.exe

C:\Users\Navneet\Desktop\CO>python TwoPassAssembler.py
Please remove empty line in between two instructions

C:\Users\Navneet\Desktop\CO>
```

- If no STOP opcode is found.

```
1    START 4095
2        CLA
3        INP A
4        INP B
5        LAC A
6        SUB B
7        BRN L1
8        DSP A
9        CLA
10       BRZ L2
11   L1: DSP B
12       CLA
13       BRZ L2
14   L2: CLA
15   END
```

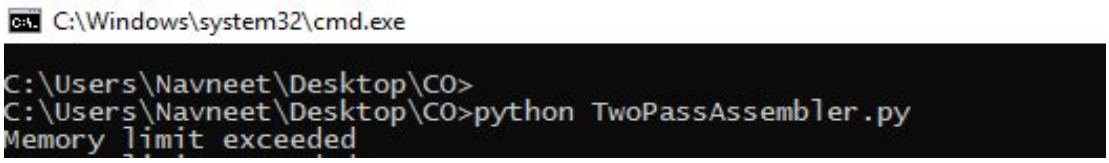
```
C:\Windows\system32\cmd.exe

C:\Users\Navneet\Desktop\CO>python TwoPassAssembler.py
Program length : 13
Error : No stop for stopping the execution

C:\Users\Navneet\Desktop\CO>_
```

- If the Virtual address exceeds 12 bits.

```
1    START 4095
2        CLA
3        INP A
4        INP B
5        LAC A
6        SUB B
7        BRN L1
8        DSP A
9        CLA
10       BRZ L2
11   L1: DSP B
12       CLA
13       BRZ L2
14   L2: STP
15   END
```



Sources Used

- http://www.wbuthelp.com/chapter_file/2677.pdf
- CSE 112 Slide.