DAYANANDA SAGAR COLLEGE OF ENGINEERING
COMPUTER SCIENCE & ENGINEERING

Minor Project- Report
Apr 2022-Jul 2022

Course Faculty: Prameetha Pai
Course Name & code: System Software 19CS6DCSSW
Semester: 6
Date: 31-08-2022

TITLE OF THE PROJECT	One Pass Assembler Implementation			
STUDENT NAME	Xitiz Verma	Vonkayala Ashwini	H Vishwarehalli	Vipul Tiwari
USN	**1DS19CS193**	**1DS19CS192**	**1DS19CS191**	**1DS19CS190**
INDIVIDUAL CONTRIBUTION	Designing and working of the assembler code	Designing and working of the assembler code	Designing and working of the assembler code	Designing and working of the assembler code
GUIDE	Prof. Prameetha Pai/ Prof. Dhara K N			
PROJECT ABSTRACT :	A single pass assembler scans the program only once and creates the equivalent binary program. The assembler substitutes all of the symbolic instruction with machine code in one pass. The difficult part is to resolve future label references and assembly code in one pass. This problem is known as forward referencing and can be solved without using two pass assemblers. We will be implementing this project using C programming.			
PLATFORM USED (H/W & S/W TOOLS TO BE USED	Python , Visual Studio Code, Pycharm			
INTRODUCTION	An assembler is a program that accepts an assembly language program (source) as input and produces its machine language equivalent (object code) along with the information for the loader. There are two types of assemblers, one pass and two pass. In this project we will be dealing with one pass assembler. Single Pass Assembler: –Performes and generates Object Code in single pass –Need to resolve the forward referencing The ability to compile in a single pass is often seen as a benefit because it simplifies			

the job of writing a compiler and one pass compilers generally compile faster than multi-pass compilers. Many languages were designed so that they could be compiled in a single pass .

DESIGN

The Main Criteria in designing the assembler using single pass was to resolve forward references. We can avoid to some extent the forward references by:

Eliminating forward reference to data items, by defining all the storage reservation statements at the beginning of the program rather at the end.

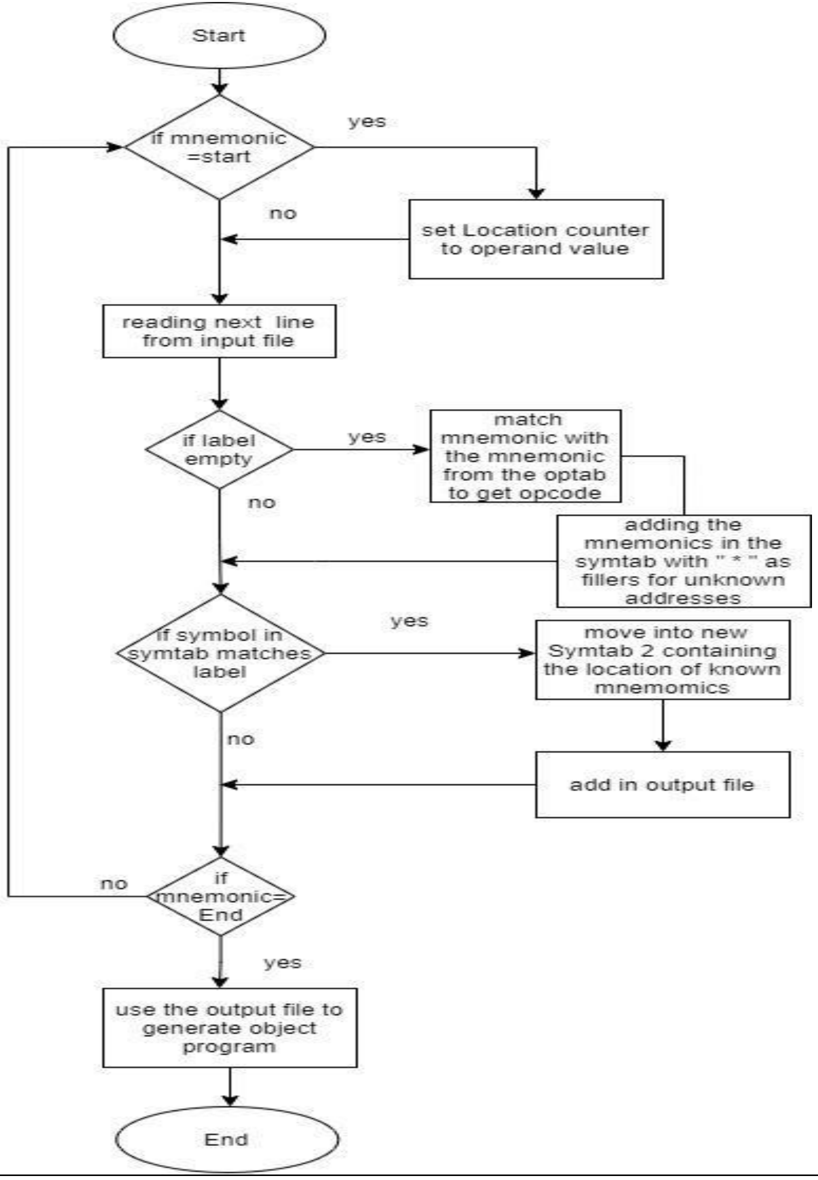
Unfortunately, forward reference to labels on the instructions cannot be avoided. (forward jumping)

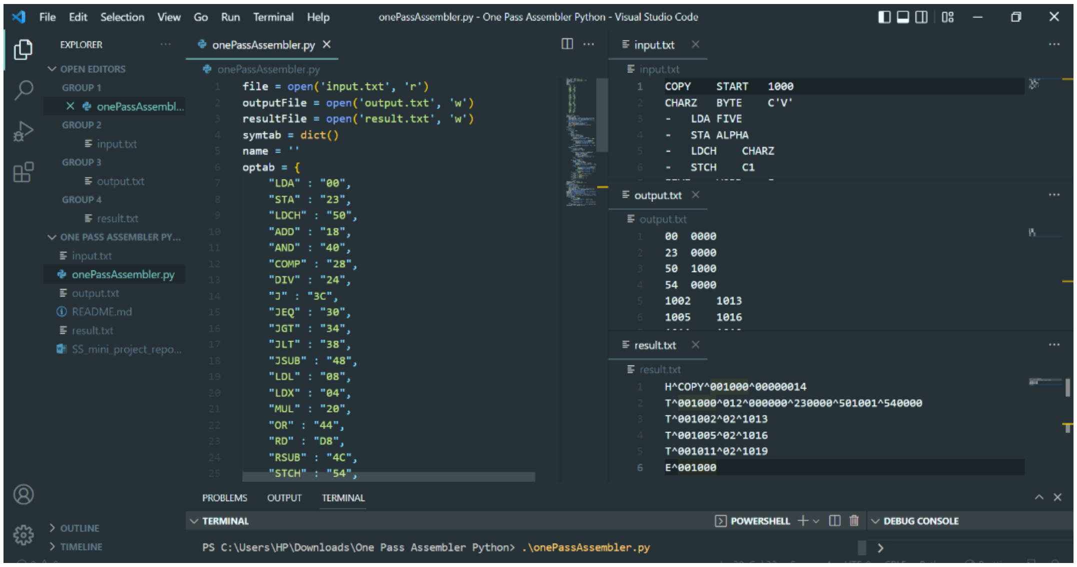
To provide some provision for handling forward references by prohibiting forward references to data items.

There are two types of one-pass assemblers:

One that produces object code directly in memory for immediate execution (Load-and-go assemblers).

The other type produces the usual kind of object code for later execution.


Start
if mnemonic =start
yes
no
set Location counter to operand value
reading next line from input file
if label empty
yes
match mnemonic with the mnemonic from the optab to get opcode
no
adding the mnemonics in the symtab with " * " as fillers for unknown addresses
if symbol in symtab matches label
yes
move into new Symtab 2 containing the location of known mnemomics
no
add in output file
no
if mnemonic= End
yes
use the output file to generate object program
End

	Load-and-go assembler generates their object code in memory for immediate execution. No object program is written out, no loader is needed. It is useful in a system with frequent program development and testing The efficiency of the assembly process is an important consideration. Programs are re-assembled nearly every time they are run; efficiency of the assembly process is an important consideration. Forward Reference in One-Pass Assemblers: In load-and-Go assemblers when a forward reference is encountered : Omits the operand address if the symbol has not yet been defined Enters this undefined symbol into SYMTAB and indicates that it is undefined Adds the address of this operand address to a list of forward references associated with the SYMTAB entry When the definition for the symbol is encountered, scans the reference list and inserts the address. At the end of the program, reports the error if there are still SYMTAB entries indicated undefined symbols. For Load-and-Go assembler Search SYMTAB for the symbol named in the END statement and jumps to this location to begin execution if there is no error.
PROJECT SOURCE CODE LINK (GITHUB/ GOOGLE DRIVE)	*https://github.com/XitizVerma/One-Pass-Assembler*
CONCLUSION /FUTURE ENHANCEMENT	We have successfully implemented the one pass assembler from scratch in python. We will work on implementing a GUI to make the assembler much more convenient to use. 

onePassAssembler.py

```python
file = open('input.txt', 'r')
outputFile = open('output.txt', 'w')
resultFile = open('result.txt', 'w')
symtab = dict()
name = ''
optab = {
    "LDA" : "00",
    "STA" : "23",
    "LDCH" : "50",
    "ADD" : "18",
    "AND" : "40",
    "COMP" : "28",
    "DIV" : "24",
    "J" : "3C",
    "JEQ" : "30",
    "JGT" : "34",
    "JLT" : "38",
    "JSUB" : "48",
    "LDL" : "08",
    "LDX" : "04",
    "MUL" : "20",
    "OR" : "44",
    "RD" : "D8",
    "RSUB" : "4C",
    "STCH" : "54",
```

input.txt

```
COPY    START   1000
CHARZ   BYTE    C'V'
-   LDA FIVE
-   STA ALPHA
-   LDCH    CHARZ
-   STCH    C1
```

output.txt

```
00  0000
23  0000
50  1000
54  0000
1002    1013
1005    1016
```

result.txt

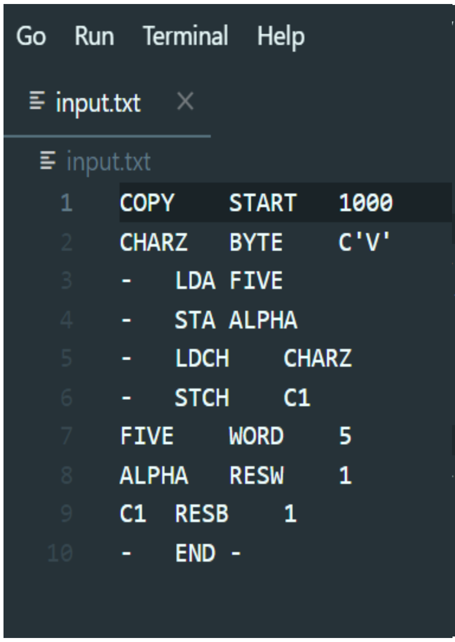
```
H^COPY^001000^00000014
T^001000^012^000000^230000^501001^540000
T^001002^02^1013
T^001005^02^1016
T^001011^02^1019
E^001000
```

TERMINAL

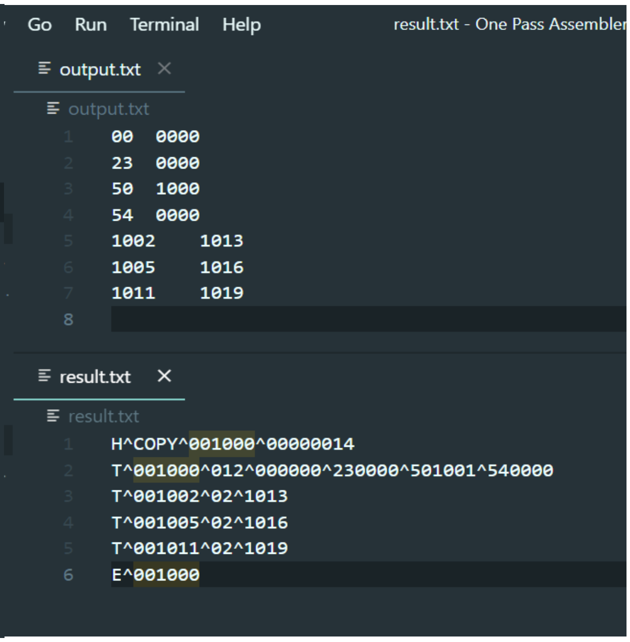
```
PS C:\Users\HP\Downloads\One Pass Assembler Python> .\onePassAssembler.py
```

UI SCREENSHOTS

Input Output and Result text file of the source program



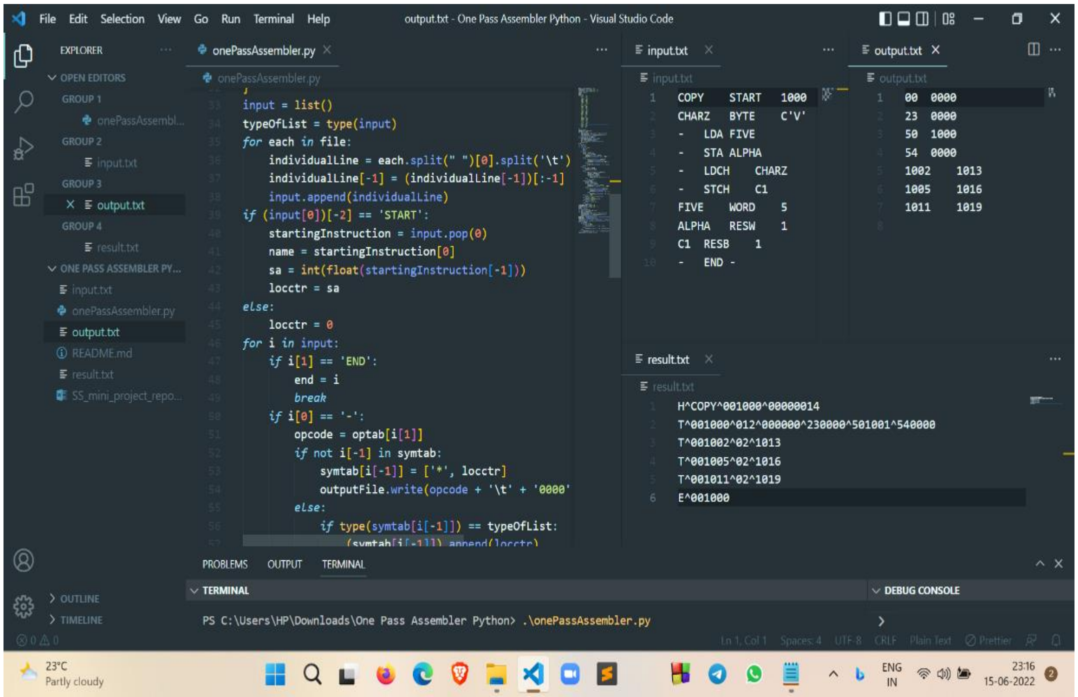
```
COPY    START   1000
CHARZ   BYTE    C'V'
-   LDA FIVE
-   STA ALPHA
-   LDCH    CHARZ
-   STCH    C1
FIVE    WORD    5
ALPHA   RESW    1
C1  RESB    1
-   END -
```



```
00  0000
23  0000
50  1000
54  0000
1002    1013
1005    1016
1011    1019
```

```
H^COPY^001000^00000014
T^001000^012^000000^230000^501001^540000
T^001002^02^1013
T^001005^02^1016
T^001011^02^1019
E^001000
```

Program Execution



```python
input = list()
typeOfList = type(input)
for each in file:
    individualLine = each.split(" ")[0].split('\t')
    individualLine[-1] = (individualLine[-1])[:-1]
    input.append(individualLine)
if (input[0])[-2] == 'START':
    startingInstruction = input.pop(0)
    name = startingInstruction[0]
    sa = int(float(startingInstruction[-1]))
    locctr = sa
else:
    locctr = 0
for i in input:
    if i[1] == 'END':
        end = i
        break
    if i[0] == '-':
        opcode = optab[i[1]]
        if not i[-1] in symtab:
            symtab[i[-1]] = ['*', locctr]
            outputFile.write(opcode + '\t' + '0000'
        else:
            if type(symtab[i[-1]]) == typeOfList:
                (symtab[i[-1]]).append(locctr)
```

```
PS C:\Users\HP\Downloads\One Pass Assembler Python> .\onePassAssembler.py
```

```
COPY    START   1000
CHARZ   BYTE    C'V'
-   LDA FIVE
-   STA ALPHA
-   LDCH    CHARZ
-   STCH    C1
FIVE    WORD    5
ALPHA   RESW    1
C1  RESB    1
-   END -
```

```
00  0000
23  0000
50  1000
54  0000
1002    1013
1005    1016
1011    1019
```

```
H^COPY^001000^00000014
T^001000^012^000000^230000^501001^540000
T^001002^02^1013
T^001005^02^1016
T^001011^02^1019
E^001000
```