Experiment No 7

Aim: Write a program to implement Two Pass Assembler

Learning Objective: Translating mnemonic operation codes to their machine language equivalents. Assigning machine addresses to symbolic labels used by the programmer. Lastly to convert assembly language to binary.

Algorithm:

Pass 1:

```
1. Start
2.Intialize location counter to zero
3.Read opcode field of next instruction.
4.search opcode in pseudo opcode Table(POT)
5.If opcode is found in POT
        5.1 If it is 'DS' or 'DC'
              Adjust location counter to proper alignment.
              Assign    length of data field to 'L'
             Go to  step 9
        5.2 If it is 'EQU'
                        Evaluate operand field
                        Assign values to symbol in label field
                        Go to step 3
        5.3 If it is 'USING' or 'DROP' Go to step 3
        5.4 If it is 'END'
```

```
            Assign value to symbol in label field
            Go to step 3
6. Search  opcode in Machine Opcode Table.
7. Assign its length to ‘L’.
8. Process any literals and enter them into literal Table.
9.If symbol is there in the label field
        Assign current value of Location Counter to symbol
10. Location Counter= Location Counter +L.
11.Go to step 3.
12. Stop.
```

Pass2:

```
1.Start
2.Intialize location counter to zero.
3. Read opcode field of next instruction.
4.Search opcode in pseudo opcode table.
5.If  opcode is found in pseudo opcode Table
    5.1 If it is ‘DS’ or ‘DC’
            Adjust location counter to proper alignment.
            If it is ‘DC’ opcode form constant and insert in assembled program
            Assign length of data field to ‘L’
            Go to step 6.4
    5.2 If it is ‘EQU’ or ‘START’ ignore it. Go to step 3
    5.3 If it is ‘USING’
        Evaluate operand and enter base reg no. and value into base table
```

```
            Go to step 3
        5.4 If it is ‘DROP’
                Indicate base reg no . available in base table . Go to step 3
        5.5 If it is ‘END’
                Generate literals for entries in Literal Table
                Go to step 12
6 Search opcode in MOT
7. Get opcode byte and format code
8. Assign its  length to ‘L’.
9. Check type of instruction.
10.If it is type ‘RR’ type
      10.1 Evaluate both register expressions and insert into second byte.
      10.2 Assemble instruction
      10.3 Location Counter= Location Counter +L.
      10.4.Go to step 3.
11. If it is ‘RX’ type
   11.1 Evaluate register and index expressions and insert into second byte.
   11.2 Calculate effective address of operand.
   11.3 Determine appropriate displacement and base register
   11.4 Put base and displacement into bytes 3 and 4
   11.5 Location Counter= Location Counter +L.
   11.6 Go to step 11.2
13 Stop.
```

Implementation Details

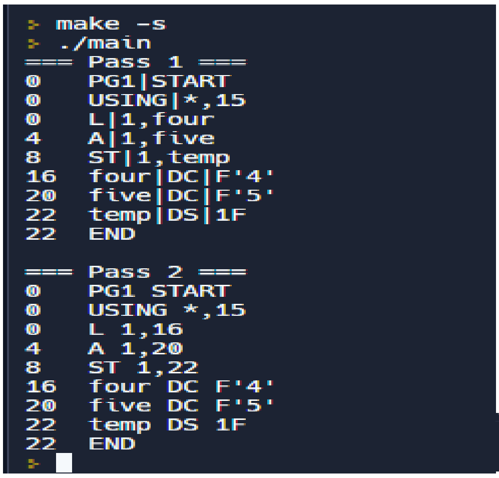
1. Read Assembly language input file
2. Display output of Pass1 as the output file with Op-code Table, Symbol Table
3. Display output of pass2 as the Op-code Table, Symbol Table , Copy file

Test Cases:

1 Input symbol which is not defined

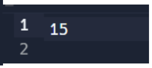
2 Input Opcode which is not entered in MOT

Output :



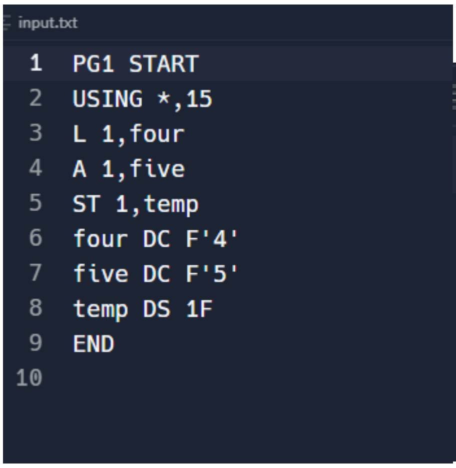
```
> make -s
> ./main
=== Pass 1 ===
0    PG1|START
0    USING|*,15
0    L|1,four
4    A|1,five
8    ST|1,temp
16   four|DC|F'4'
20   five|DC|F'5'
22   temp|DS|1F
22   END

=== Pass 2 ===
0    PG1 START
0    USING *,15
0    L 1,16
4    A 1,20
8    ST 1,22
16   four DC F'4'
20   five DC F'5'
22   temp DS 1F
22   END
>
```

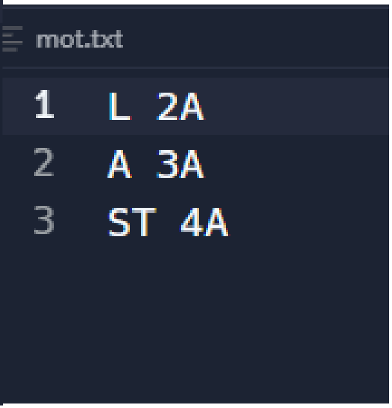


```
15
```

Input.txt **mot.txt**

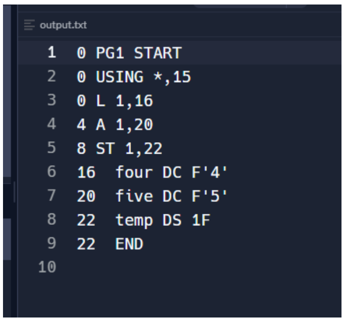


```
PG1 START
USING *,15
L 1,four
A 1,five
ST 1,temp
four DC F'4'
five DC F'5'
temp DS 1F
END
```

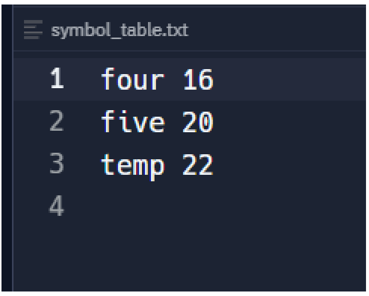


```
L 2A
A 3A
ST 4A
```

Output.txt **symbol_table.txt**



```
0 PG1 START
0 USING *,15
0 L 1,16
4 A 1,20
8 ST 1,22
16  four DC F'4'
20  five DC F'5'
22  temp DS 1F
22  END
```



```
four 16
five 20
temp 22
```

Post Lab Questions:

1. **Define the basic functions of assembler.**
2. **What is the need of SYMTAB (symbol table) in assembler?**
3. **What is the need of MOT in assembler**
4. **What is meant by one pass assembler?**