**TITLE: Implement Pass-I of a Two-pass assembler**

**PROBLEM STATEMENT:**

Design suitable data structures and implement pass-I of a two-pass assembler.

Implementation should consist of a few instructions from each category and few assembler directives.

**THEORY:**

* Assembler: An assembler is a program that takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations. Some people call these instructions assembler language and others use the term assembly language.
* Data Structures: The data structure name indicates itself that organizing the data in memory. There are many ways of organizing the data in the memory as we have already seen one of the data structures, i.e., array in C language. Array is a collection of memory elements in which data is stored sequentially, i.e., one after another. In other words, we can say that array stores the elements in a continuous manner. This organization of data is done with the help of an array of data structures.
* Pass-I of two pass assembler: Pass 1 of the assembler scans the source, determining the size and address of all data and instructions; then pass 2 scans the source again, outputting the binary object code.

**PROGRAM:**

#include<stdio.h>

#include<string.h> #include<stdlib.h> void main() { int lit, count=0;

char opcode[10], operand[10], label[10], code[10], mnemonic[10];

int locctr, start, length;

char \*tmp;

FILE \*fp1, \*fp2, \*fp3, \*fp4, \*fp5, \*fp6;

fp1 = fopen("input.txt", "r"); fp2 = fopen("optab.txt", "r"); fp3 = fopen("symtbl.txt", "w"); fp4 = fopen("out.txt", "w"); fp5 = fopen("lit.txt", "w");

fp6 = fopen("litpool.txt", "w");

fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);

fprintf(fp6, "%d\n", count);

if(strcmp(opcode, "START")==0) {

start = atoi(operand);

locctr = start; fprintf(fp4, "\t%s\t%s\t%s\n", label, opcode, operand);

fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);

} else { locctr = 0;

} while(strcmp(opcode, "END")!=0) {

fprintf(fp4, "%d\t", locctr); tmp = strstr(operand,"="); if (tmp != NULL) { count++; fprintf(fp5, "%s\n", operand);

}

if(strcmp(label, "\*\*")!=0) {

fprintf(fp3, "%s\t%d\n", label, locctr);

}

fscanf(fp2, "%s\t%s", code, mnemonic);

while(strcmp(code, "END")!=0) {

if(strcmp(opcode, code)==0) {

locctr+=1; break;

}

fscanf(fp2, "%s\t%s", code, mnemonic);

}

if(strcmp(opcode, "ORIGIN")==0) {

locctr+=3;

}

else if(strcmp(opcode, "DS")==0) {

locctr+=1;

}

else if(strcmp(opcode, "LTORG")==0) {

for (lit=0; lit<count; lit++) {

fprintf(fp5, "%d\n", locctr + lit);

}

locctr+=count;

fprintf(fp6, "%d\n", count);

}

fprintf(fp4, "%s\t%s\t%s\t\n", label, opcode, operand);

fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);

}

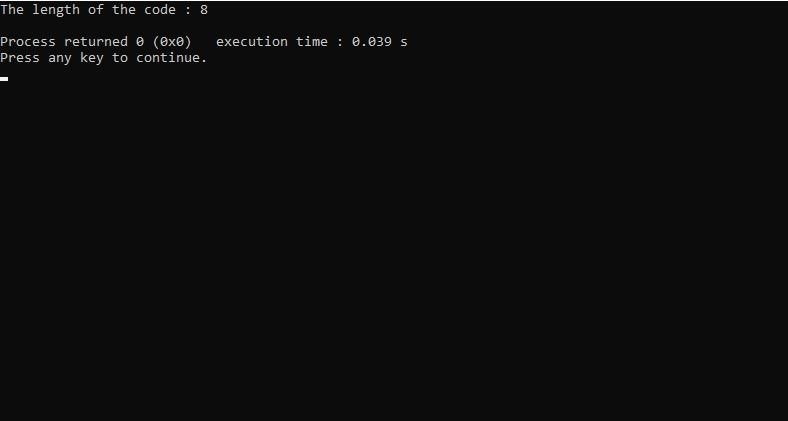
fprintf(fp4, "%d\t%s\t%s\t%s\n", locctr, label, opcode, operand);

length = locctr-start; printf("The length of the code : %d\n", length); fclose(fp1);

fclose(fp2); fclose(fp3); fclose(fp4);

}

**OUTPUT:**



**CONCLUSION:**

We have successfully learnt the implementation and the theory of pass-I of a two pass assembler