Assignment 1

Pass 1 of Two Pass Assemblers

→ **Problem Statement :**

Write a program to implement Pass-1 of Two Pass Assembler for Symbols & Literal Processing

Consider the following cases :

1. Forward References
2. DS & DC Statements
3. START, EQU, LTORG, END
4. Error Handling : Symbol used but not defined, invalid instruction/register etc.

→ **Theory :**

- A language translator bridges the gap to machine language of computer system. An assembler is a language translator whose source language is assembly language
- language processing activity consists of 2 phases, Analysis phase & Synthesis phase. Analysis of source program consists of 3 components Lexical Rules, Syntax rules & Semantic rules.

- Because of forward references issues concerning memory requirements & organization of language Processor, the Analysis of source code may not be immediately followed by synthesis.

- To solve the problem, multi-pass language processing is preferred as it postpones the generation of target code until more information about the concerning entity is acquired.

- Language processor Pass

It is the processing of every statement in a source code or its equivalent to perform language-processing function.

- Function of Analysis & Synthesis Phase

1. Isolate the label operation code & operands field of a statement.
2. Enter the symbol found in label & address of next available machine word into symbol table.
3. Validate the mnemonic operation code by looking it up in the mnemonic table
4. Determine the machine storage requirements of the statement by considering the mnemonic operation code & operand fields of the statement.

- Assembly language Statements :

There are 3 type of Statements :

1. Imperative : Indicates an action
2. Declarative : Reserves memory & associates names with them.
3. Assembly Directives : Indicates certain specific Actions.

- Pass 1

It separates the symbols, mnemonic codes & operand fields. Determines the storage requirements for every statements. Builds the symbol table & literal table. Construct the intermediate code for every assembly language statement.

Data Structures Required :

1. Source file containing the assembly program.
2. OPTAB : A table of mnemonics for the various Assembly Language Statements
3. SYMTAB : A table containing information of symbols/variables like value, location/memaddr & position.
4. LITTAB : Similar table but for literals.
5. POOLTAB : Stores pointers to the literals in the current literal pool.

- Algorithm :

1. Open the source file in read mode.
2. If end of file go to step 8
3. Read next line of source code.
4. Separate line into words.
5. Search for 1st word is mnemonic in table, if not then its is label & add this in SYMTAB with current LC.
6. If instruction found
   case 1 : imperative statement
   case 2 : declarative statement
   case 3 : Assembler directive.
   Generate intermediate code & write to file.
7. go to step 2.
8. close source file & open {intermediate code}
9. If end of file intermediate code, go to step 13
10. Read next line from intermediate code
11. Write opcode register code & address of memory on target code.
12. go to step 9.
13. close all files.
14. Display symbol table, literal table & target file.

→ Conclusion :

Topics Covered :

1. Basics of Assembler
2. Passes in an Assembler
3. Forward Referencing
4. Pass 1 of Two Pass Assembler.