EXPERIMENT NO: 11

1. Title:

Design suitable data structures and implement pass-I of a two-pass assembler for pseudo-machine in Java using object oriented feature. Implementation should consist of a few instructions from each category and few assembler directives.

2. Objectives:

- To understand data structures to be used in pass I of an assembler.
- To implement pass I of an assembler

3. Problem Statement:

Write a program to create pass-I Assembler

4. Outcomes:

After completion of this assignment students will be able to:

- Understand the concept of Pass-I Assembler
- Understand the Programming language of Java

5. Software Requirements:

- Linux OS, JDK1.7

6. Hardware Requirement:

- 4GB RAM ,500GB HDD

7. Theory Concepts:

A language translator bridges an execution gap to machine language of computer system. An assembler is a language translator whose source language is assembly language.

An Assembler is a program that accepts as input an Assembly language program and converts it into machine language.

Language processing activity consists of two phases, Analysis phase and synthesis phase. Analysis of source program consists of three components, Lexical rules, syntax rules and semantic rules. Lexical rules govern the formation of valid statements in source language. Semantic rules associate the formation meaning with valid statements of language. Synthesis phase is concerned with construction of target language statements, which have the same meaning as source language statements. This consists of memory allocation and code generation.

TWO PASS TRANSLATION SCHEME:

In a 2-pass assembler, the first pass constructs an intermediate representation of the source program for use by the second pass. This representation consists of two main components - data structures like Symbol table, Literal table and processed form of the source program called as intermediate code(IC). This intermediate code is represented by the syntax of Variant –I.

Analysis of source program statements may not be immediately followed by synthesis of equivalent target statements. This is due to forward references issue concerning memory requirements and organization of Language Processor (LP).

Forward reference of a program entity is a reference to the entity, which precedes its definition in the program. While processing a statement containing a forward reference, language processor does not posses all relevant information concerning referenced entity. This creates difficulties in synthesizing the equivalent target statements. This problem can be solved by postponing the generation of target code until more information concerning the entity is available. This also reduces memory requirements of LP and simplifies its organization. This leads to multi-pass model of language processing.

Language Processor Pass: -

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

Assembly Language statements: -

There are three types of statements Imperative, Declarative, Assembly directives. An imperative statement indicates an action to be performed during the execution of assembled program. Each imperative statement usually translates into one machine instruction. Declarative statement e.g. DS reserves areas of memory and associates names with them. DC constructs memory word containing constants. Assembler directives instruct the assembler to perform certain actions during assembly of a program,

e.g. START<constant> directive indicates that the first word of the target program generated by assembler should be placed at memory word with address <constant>

Function Of Analysis And Synthesis Phase:

Analysis Phase: -

Isolate the label operation code and operand fields of a statement.

Enter the symbol found in label field (if any) and address of next available machine word into symbol table.

Validate the mnemonic operation code by looking it up in the mnemonics table.Determine the machine storage requirements of the statement by considering the mnemonic operation code and operand fields of the statement.

Calculate the address of the address of the first machine word following the target code generated for this statement (Location Counter Processing)

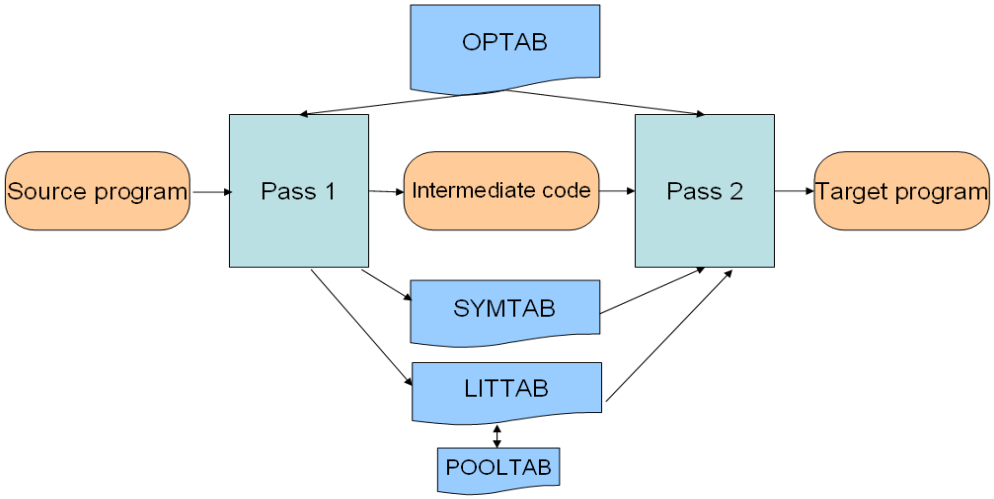
Synthesis Phase:

Obtain the machine operation code corresponding to the mnemonic operation code by searching the mnemonic table.

Obtain the address of the operand from the symbol table.

Synthesize the machine instruction or the machine form of the constant as the case may be.

DATA STRUCTURES OF A TWO PASS ASSEMBLER:



Data Structure of Assembler:

a) Operation code table (OPTAB) :This is used for storing mnemonic,

operation code and class of instruction

Structure of OPTAB is as follows

b) Data structure updated during translation: Also called as translation time data

structure. They are

I. SYMBOL TABLE (SYMTAB) : Ii contains entries such as

symbol,it's address and value.

SYMBOL TABLE have following fields :

Name of symbol	Symbol Address	Value

II. LITERAL TABLE (LITTAB) : it contains entries such as literal

and it's value.

Literal Table has following fields :

literal	Address of Literal

III . POOL TABLE (POOLTAB): Contains literal number of the

starting literal of each literal pool.

Pool TABLE (pooltab) have following fields.

LITERAL_NO

IV: Location Counter which contains address of next instruction by

calculating length of each instruction.

Design of a Two Pass Assembler: -

Tasks performed by the passes of two-pass assembler are as follows:

Pass I: -

Separate the symbol, mnemonic opcode and operand fields.

Determine the storage-required foe every assembly language statement and update the location counter.

Build the symbol table and the literal table.

Construct the intermediate code for every assembly language statement.

Pass II: -

Synthesize the target code by processing the intermediate code generated during pass1

INTERMEDIATE CODE REPRESENTATION

The intermediate code consists of a set of IC units, each IC unit consisting of the following three fields

1. Address
2. Representation of the mnemonic opcode
3. Representation of operands

Mnemonic field

The mnemonic field contains a pair of the form: (statement class, code)

Where statement class can be one of IS,DL and AD standing for imperative statement, declaration statement and assembler directive , respectively.

For imperative statement, code is the instruction opcode in the machine language

For declaration and assembler directives , following are the codes

Declaration Statements		Assembler directives	
DC	01	START	01
DS	02	END	02
		ORIGIN	03
		EQU	04
		LTORG	05

Mnemonic Operation Codes :

Instruction Opcode	Assembly Mnemonic	Remarks
00	STOP	STOP EXECUTION
01	ADD	FIRST OPERAND IS MODIFIED CONDITION CODE IS SET
02	SUB	FIRST OPERAND IS MODIFIED CONDITION

		CODE IS SET
03	MULT	FIRST OPERAND IS MODIFIED CONDITION CODE IS SET
04	MOVER	REGISTER ← MEMORY MOVE
05	MOVEM	MEMORY MOVE → REGISTER MOVE
06	COMP	SETS CONDITION CODE
07	BC	BRANCH ON CONDITION
08	DIV	ANALOGOUS TO SUB
09	READ	FIRST OPERAND IS NOT USED
10	PRINT	FIRST OPERAND IS NOT USED

Branch On Condition :

BC <condition code > <memory address>

Transfer Control to the Memory word With Address < memory address>

Condition code	Opcode
LT	01
LE	02
EQ	03
GT	04
GE	05
ANY	06

8. Algorithms(procedure) :

PASS 1

• Initialize location counter, entries of all tables as zero.

• Read statements from input file one by one.

• While next statement is not END statement

I. Tokenize or separate out input statement as label,numonic,operand1,operand2

II. If label is present insert label into symbol table.

III. If the statement is LTORG statement processes it by making it's entry into literal table, pool table and allocate memory.

IV. If statement is START or ORIGEN Process location counter accordingly.

V. If an EQU statement, assign value to symbol by correcting entry in symbol table.

VI. For declarative statement update code, size and location counter.

VII. Generate intermediate code.

VIII. Pass this intermediate code to pass -2.

10. Conclusion:

Thus, I have studied visual programming and implemented dynamic link library application for arithmetic operation

References:

J. J. Donovan, "Systems Programming", McGraw Hill.[chapter 3 topic 3.0,3.1, 3.2.1 in brief , figure 3.3 &3.5]

Oral Questions: [Write short answer]

1. Explain what is meant by pass of an assembler.
2. Explain the need for two pass assembler.
3. Explain terms such as Forward Reference and backward reference.
4. Explain various types of errors that are handled in passé-I.
5. Explain the need of Intermediate Code generation and the variants used.
6. State various tables used and their significance in the design of two pass Assembler.
7. What is the job of assembler?
8. What are the various data structures used for implementing Pass-I of a two-pass assembler.
9. What feature of assembly language makes it mandatory to design a two pass assembler?
10. How are literals handled in an assembler?
11. How assembler directives are handled in pass I of assembler?