Page :
Date : / /

## Assignment A2

**Title** : Pass II of a two pass assembler

**Date of Completion** : 16 / 01 / 2020

**Problem Statement** :
Implement Pass II of a two pass assembler for pseudo-machine in java ~~of~~ using object oriented features. The output of Assignment I (intermediate file & symbol table should be the input for this assignment.

**Objectives** :
i. Implement Pass II of a 2 pass assembler
ii. Use object oriented features for implementing pass II

**Outcome** :
Student should be able to :-
i. Understand & implement pass II of a 2 pass assembler
ii. Implement pass II using object oriented features.

**Software & Hardware Requirements** :
i. Fedora 64 bit OS
ii. Eclipse IDE for Java
iii. i5 processor
iv. 4 GB RAM
v. 500 GB HDD

14 | Implement algorithmic techniques by virtue ofadvance programming skill set and Free and Open Source Software (FOSS) tools.

Page:
Date:

## Theory:

An assembler is a program that accepts an assembly language program as input and produces its machine code as output.

Mostly assemblers are designed in two passes. The pass wise grouping of tasks is given below :-

Pass I
1. Separate labels, opcodes & operands
2. Determine storage requirement for every assembly language statement & update location counter
3. Build the Symbol Table

Pass II
1. Generate the machine code.

Unlike a one pass assembler, a two pass assembler can handle literals.

In the first pass, when the assembler encounters the use of literal in operand field of a statement, it allocates memory word to store the value of the literal. This is done with the help of literal table.

In the second pass, the assembler replaces every literal by its address.

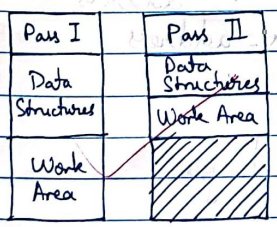
Page :
Date : / /

A one pass assembler will not be able to assign memory address to a literal. Hence, a one pass assembler cannot handle literals.

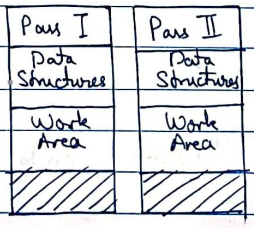
There are 2 variants of Intermediate Code :
Variant I - Operands are procesed in Pass I
Variant II - Operands are procesed in Pass II

Variant I
Variant I does more work in Pass I
Operand fields are completely procesed in Part I
Task of Pass II becomes quite simple
Intermediate Code becomes quite compact.
Memory & Processing requirements are higher in Pass I

Variant II
Variant II reduces the work of Pass I by making Pass II do the operand processing
Procesing Requirements are evenly distributed in both passes




Memory Utilization in Variant I




Memory Utilization in Variant II

Page:
Date:

## Algorithm for Pass II

Intermediate Code is stored in table IC[ ]
Target Code will be assembled in area named code_area

1. Code_area_address = address of Code Area
2. For each entry in IC[ ]
{
   a) If an imperative statement
      i) Read LC
      ii) Get opcode
      iii) Get operand address from Symbol table
      iv) Get literal address from Literal table
      v) Assemble instruct in machine_code_buffer
      vi) Move contents of machine_code_buffer in code_area at the address LC + code_area_address
   b) If DC statement then
      i) Read LC
      ii) Assemble constant in machine_code_buffer
      iii) Move contents of machine_code_buffer in code_area at the address LC + code_area_address
}
3. Write code_area into output file.

Page :
Date : / /

## Test Cases :

| Description | Input | Output | Result |
|---|---|---|---|
| Regular Command | (is,01) 1 (s,02) | 205 01 01 210 | Success |
| Stop Command | (is,00) | 215 00 00 000 | Success |
| DC Command | (ds,02) (s,01) (c,05) | 218 00 00 005 | Success |

## Conclusion :

The Pass II of a 2 pass assembler (Variant I) has been successfully implemented using concepts of Object Oriented Programming.

Sushrut
23/01/20

14 Implement algorithmic techniques by virtue ofadvance programming skill set and Free and Open Source Software (FOSS) tools.