Assignment A1

Title : Pass I of a two pass assembler

Problem Statement :

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 few instructions from each category and few assembler directives.

Objectives:

(i) Understand the internals of language translators.
(ii) Handle tools like LEX and YACC.
(iii) Understand the operating system internals and functionalities with implementation point of view.

Software and Hardware Requirements:

64-bit open source Linux (Fedora 20)
Eclipse IDE, JAVA
I3 and I5 machines.

Theory:

Assembler is a program which converts assembly language instructions into machine language form.

A two pass assembler takes two scans of source code to produce the machines code from assembly language program.

Assembly process consists of following activities-

(i) Convert mnemonics to their machine language opcode equivalents.
(ii) Convert symbolic (i.e. variables, jump labels) operands to their machine addresses.
(iii) Translate data constants into internal machine representations.
(iv) Output the object program and provide other information required for linker and loader

Pass I Tasks:

(i) Assign addresses to all the statements in the program (address assignment)
(ii) Save the values (addresses) assigned to all labels (including label and variable names) for use in pass II (Symbol Table creation)
(iii) Perform processing of assembler directives (e.g. BYTE, RESW Directives can affect address assignment)

Description using set theory:

Let 'S' be set which represents a system where,

I = Input
O = Output
T = Type (Variant I or II)
D = Data Structure

$S = \{I, O, T, D, Succ, Fail\}$

$I = \{SF, MF\}$

SF = Source Code File
MF = Mnemonic Table

O = {St, Lt, IC}.

where

St = Symbol
Lt = Literal
IC = Intermediate Code File

St = {N, A}.

where

N = Name of Symbol
A = Address of Symbol

Lt = {N, A}

where

N = Name of Literal.
A = Address of Literal

T = Variant II
D = {Ar, Fl, Sr}.

where,

Ar : Array
Fl = File
Sr : Structure

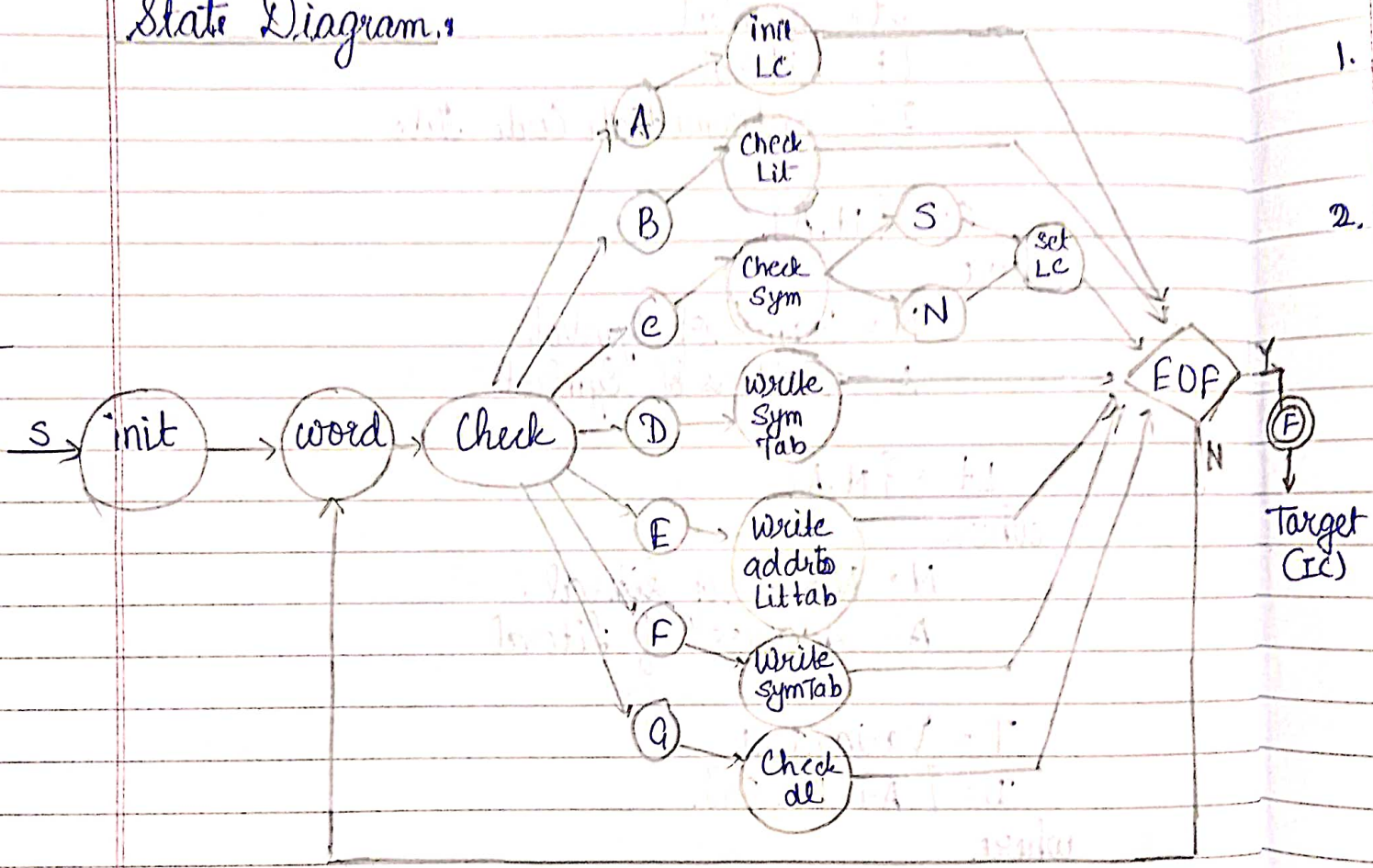
Success Succ = {x | x is set of all cases that are handled in program}

Succ = { Undefined Symbol (also label),
Duplicate Symbol,
Undefined Symbol in assembler directives,
}

Failure Fail = {x | x is set of all cases that are not handled in program}

Fail = {Multiple statements in a line}

State Diagram:



Algorithm:

(i) Create MOT
(ii) Read the .asm file and tokenise it.
(iii) Create symbol and literal tables
(iv) Generate intermediate code files.

Testing Method:

Use unit testing method for testing the functions.

	Test base	Expected Output	Result
1.	Input all valid mnemonics	Replace the mnemonics with correct options	Success.
2.	Input the instructions and operands in valid format	Generate valid intermediate code format	Success.

Conclusion:

We successfully understood and implemented pass I of a two pass assembler.