ASSIGNMENT NUMBER: A-01

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 a few instructions from each category and few assembler directives

OBJECTIVES:

- Analyze of source code to solve problem.
- Identify data structures required in the design of assembler.

OUTCOMES :

The students will be able to

- Parse and tokenize the assembly source code
- Perform the LC processing
- Generate the intermediate code file
- Design the symbol table, literal table, pooltab

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 machine code from assembly language program.

Assembly process consists of following activities:

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

Pass I Tasks:

- Assign addresses to all the statements in the program (address assignment)
- Save the values (addresses) assigned to all labels(including label and variable names) for use in pass II (Symbol Table creation)
- 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 S={I,O,T,D,Succ,Fail}

where,

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

I={Sf,Mf}

where,

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=

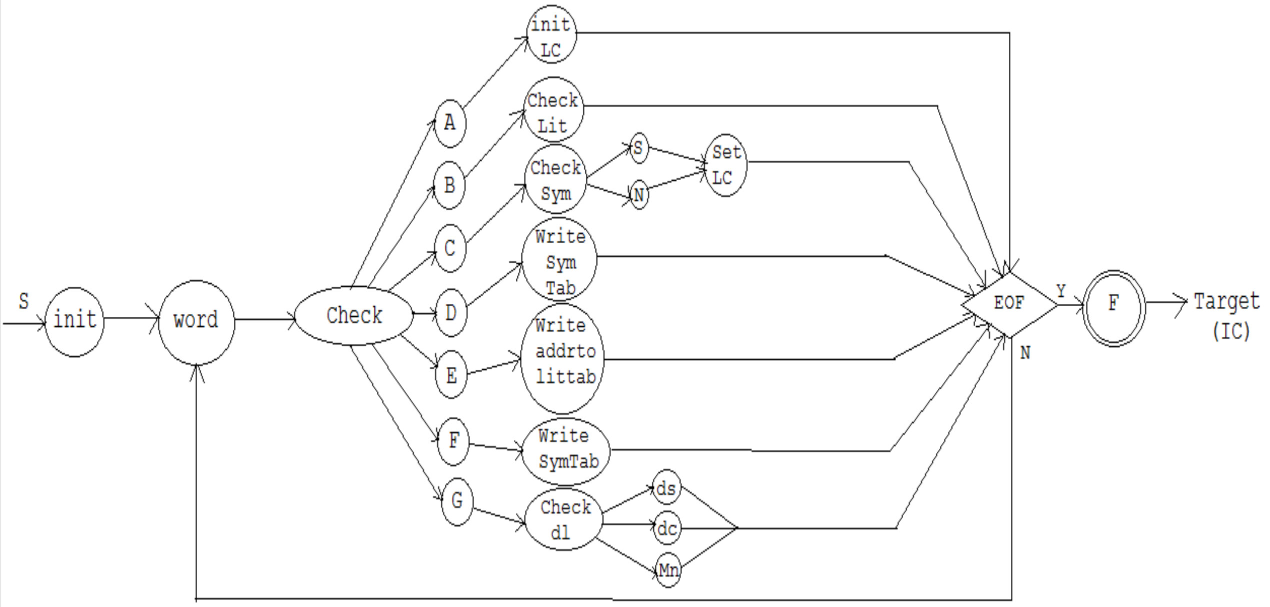
{Undefined Symbol (also label),
Duplicate Symbol,
Undefined Symbol in assembler directives,
}

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

Fail=

{Multiple statements in a line}

Turing machine/state diagram:



State Diagram
(Pass 1 of 2 Pass Assembler)

Steps to do /algorithm:

- Create MOT.
- Read the .asm file and tokenize it.
- Create symbol and literal tables.
- Generate intermediate code file.

Testing Method:

Use unit testing method for testing the functions. Test the functionalities using functional testing.

Sample Test cases

Test case id	Test case	Expected Output	Actual Result
1	Input all valid mnemonics	Replace the mnemonics with correct opcodes	Success
2	Input the instructions and operands in valid format	Generate valid intermediate code format	Success

CONCLUSION : By performing this experiment we have successfully learned about Pass-I assembler.