SPOS

Assignment - A1

Title : Pass I of a 2 pass assembler

Problem Statement :

Design suitable data structures & implement pass I of a 2-pass assembler for pseudo-machine in Java using OOP feature

Objectives :

1. Understand the internals of language translators.
2. Handle tools like LEX and YACC
3. Understand the OS internals & functionalities with implementation pt of view.

Outcomes :

I'll be able to :

1. Parse & tokenize the assembly src code.
2. Perform the 2C processing.
3. Generate the intermediate code file.
4. Design the SYMTAB, LITTAB, POOLTAB

S/W & H/W Requirements :

64-bit open src Linux (fedora 20)
Eclipse IDE, JAVA I3 & I5 mc

PAGE NO.:
DATE: / /

Theory:

1. Assembler is a program which converts assembly language instructions into m/c language form.
2. A 2 pass assembler takes 2 scans of src code to produce the m/c code from a/p.

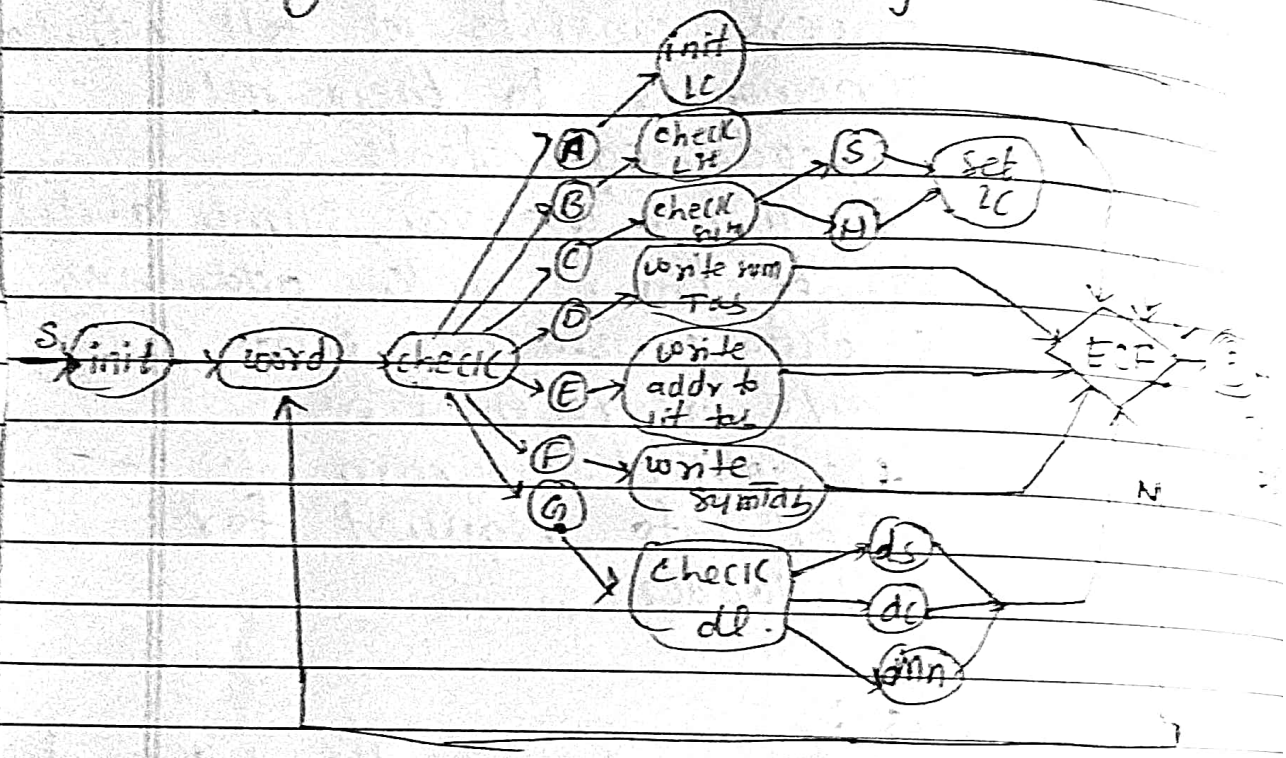
Assembly process consists of foll activities
- Convert mnemonics to their m/c language opcode equivalents.
- Convert Symbolic (ie. var, jump labels) operands to their m/c addresses.
- Translate data constants into internal m/c representation.
- Output the object program & provide other info required 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 & var names) for use in pass II (sym table creation)
- Perform processing of assembler directives (eg. BYTE, RESW directives can affect addr assignment)

Data structures used:

1. Symbol Table (SYMTAB)
2. LITTABLE (LITTAB)
3. Pool Table (POOLTAB)
4. Hashmap (for mnemonics storage)

Turing machine / State diagram



State diagram

(Pass 1 of 2 pass assembler)

Algorithm:

1. Create MOT
2. Read the .asm file & tokenize it
3. Create sym & lit tables
4. Generate IC code file.

PAGE NO.:
DATE: / /

Testing method:

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

Sample Test cases:

1.	Input all valid mnemonics	Replace the mnemonics with correct opcodes	Success
2.	I/p the ins & operands in valid format	Generate valid intermediate code format	Success.

Conclusion :-

Hence we successfully implemented pass 1 of two pass assembler.