ASSIGNMENT A-2

TITLE: Pass II of a two pass assembler.

PROBLEM STATEMENT:

Implement Pass-II of two pass assembler for pseudo-machine in Java using object oriented features. The output of assignment-1 (intermediate file and symbol table) should be input for this assignment.

OBJECTIVE:

- Synthesis of the object code.
- Understand the use of data structures required in the design of assembler.

OUTCOME:

The students will be able to

- Parse and tokenize the intermediate code file
- Perform the LC processing
- Generate the target code file
- Demonstrate the use of symbol table, literal table, pooltab

S/W PACKAGES AND HARDWARE REQUIREMENTS:

- 64-bit open source Linux (Fedora 20)
- Eclipse IDE, JAVA
- 64-bit architecture I3 or 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 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 II Tasks:

- Assemble instructios(generate opcode and look up addresses)
- Generate data values defined by BYTE, WORD
- Perform processing of assembler directives(not done in pass I)
- Write the object program and the assembly listing

ALGORITHM:

1. *code_area_address* := address of *code_area*;
   pooltab_ptr := 1;
   loc_cntr := 0;
2. While next statement is not an END statement
   (a) Clear *machine_code_buffer*;
   (b) If an LTORG statement
      (i) Process literals in LITTAB [POOLTAB [*pooltab_ptr*]] ... LITTAB [POOLTAB [*pooltab_ptr*+1]]−1 similar to processing of constants in a DC statement, i.e. assemble the literals in *machine_code_buffer*.
      (ii) *size* := size of memory area required for literals;
      (iii) *pooltab_ptr* := *pooltab_ptr* + 1;
   (c) If a START or ORIGIN statement then
      (i) *loc_cntr* := value specified in operand field;
      (ii) *size* := 0;
   (d) If a declaration statement
      (i) If a DC statement then
         Assemble the constant in *machine_code_buffer*.
      (ii) *size* := size of memory area required by DC/DS;
   (e) If an imperative statement
      (i) Get operand address from SYMTAB or LITTAB.
      (ii) Assemble instruction in *machine_code_buffer*.
      (iii) *size* := size of instruction;
   (f) If *size* ≠ 0 then
      (i) Move contents of *machine_code_buffer* to the address *code_area-_address* + *loc_cntr*;
      (ii) *loc_cntr* := *loc_cntr* + *size*;
3. (Processing of END statement)
   (a) Perform steps 2(b) and 2(f).
   (b) Write *code_area* into output file.

TEST CASES:

Test case id	Test case	Expected Output	Actual Result
1	Input Intermediate code, symbol table, literal table	Generate Machine code	Succes

CONCLUSION:

Thus, we successfully implemented Pass II of two pass Assembler in JAVA.