ASSIGNMENT A-1

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.

OBJECTIVE:

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

OUTCOME:

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

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 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)

ALGORITHM:

1. *loc_cntr* := 0; (default value)
   pooltab_ptr := 1; POOLTAB [1] := 1;
   littab_ptr := 1;
2. While next statement is not an END statement
   (a) If label is present then
   this_label := symbol in label field;
   Enter (*this_label, loc_cntr*) in SYMTAB.
   (b) If an LTORG statement then
   (i) Process literals LITTAB [POOLTAB [*pooltab_ptr*]] ... LITTAB [*littab_ptr*−1] to allocate memory and put the address in the *address* field. Update *loc_cntr* accordingly.
   (ii) *pooltab_ptr* := *pooltab_ptr* + 1;
   (iii) POOLTAB [*pooltab_ptr*] := *littab_ptr*;
   (c) If a START or ORIGIN statement then
   loc_cntr := value specified in operand field;
   (d) If an EQU statement then
   (i) *this_addr* := value of <*address spec*>;
   (ii) Correct the symtab entry for *this_label* to (*this_label, this_addr*).
   (e) If a declaration statement then
   (i) *code* := code of the declaration statement;
   (ii) *size* := size of memory area required by DC/DS.
   (iii) *loc_cntr* := *loc_cntr* + *size*;
   (iv) Generate IC '(DL, *code*) ⋯'.
   (f) If an imperative statement then
   (i) *code* := machine opcode from OPTAB;
   (ii) *loc_cntr* := *loc_cntr* + instruction length from OPTAB;
   (iii) If operand is a literal then
   this_literal := literal in operand field;
   LITTAB [*littab_ptr*] := *this_literal*;
   littab_ptr := *littab_ptr* + 1;
   else (i.e. operand is a symbol)
   this_entry := SYMTAB entry number of operand;
   Generate IC '(IS, *code*)(S, *this_entry*)';
3. (Processing of END statement)
   (a) Perform step 2(b).
   (b) Generate IC '(AD,02)'.
   (c) Go to Pass II.

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:

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