Assignment No. 01

Part I: Systems Programming and Operating Systems, Group A, 1

Problem Statement: Design suitable data structures and implement pass1 and pass2 of a two pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives. The output of pass1 (intermediate code file and symbol table) should be input for pass2

Objectives:

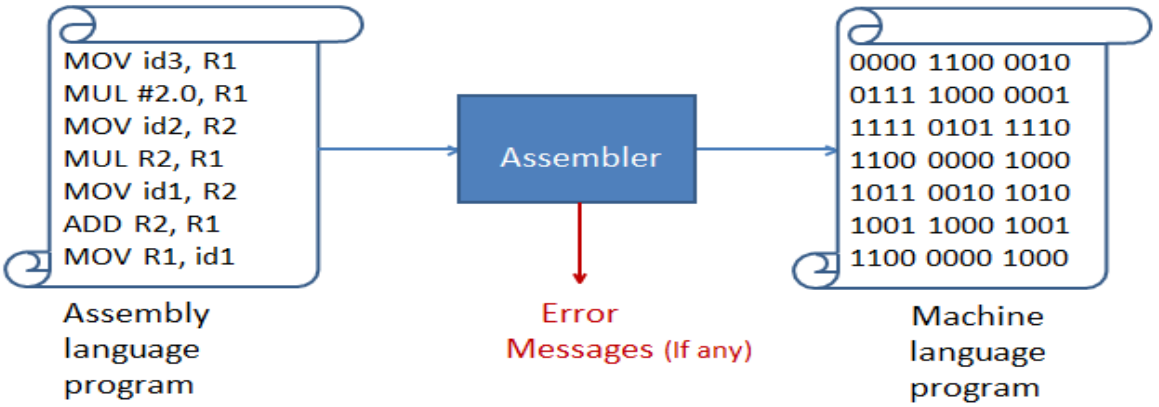
1. To learn systems programming tools
2. Implement language translator

Theory:

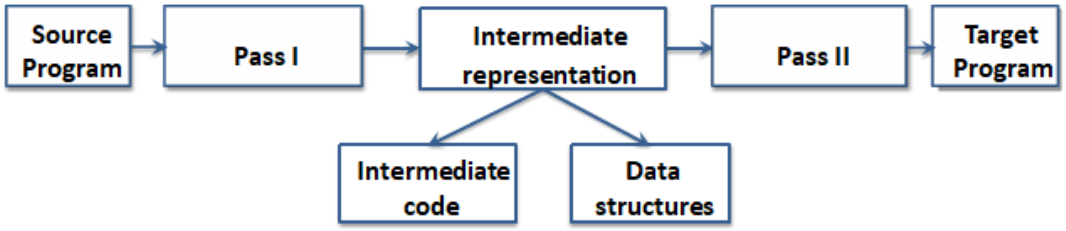
1. Assembler is a low level translator which translates source code to machine code
2. It works in two phases : analysis phase and synthesis phase
3. In analysis phase, source assembly code is analyzed to generate some intermediate datastructures
4. In synthesis phase, machine code is generated
5. There are well defined system level standard algorithms to design the assembler as a two pass assembly, namely Pass I and PassII algorithms of assembler
6. Pass I takes the source code in assembly language as input and generates intermediate datastructures.
7. PassII takes the intermediate data structures generated by Pass I as input and generates machinecode.

Assembler

- Assembler is a language processor that converts assembly language program to machine language program.

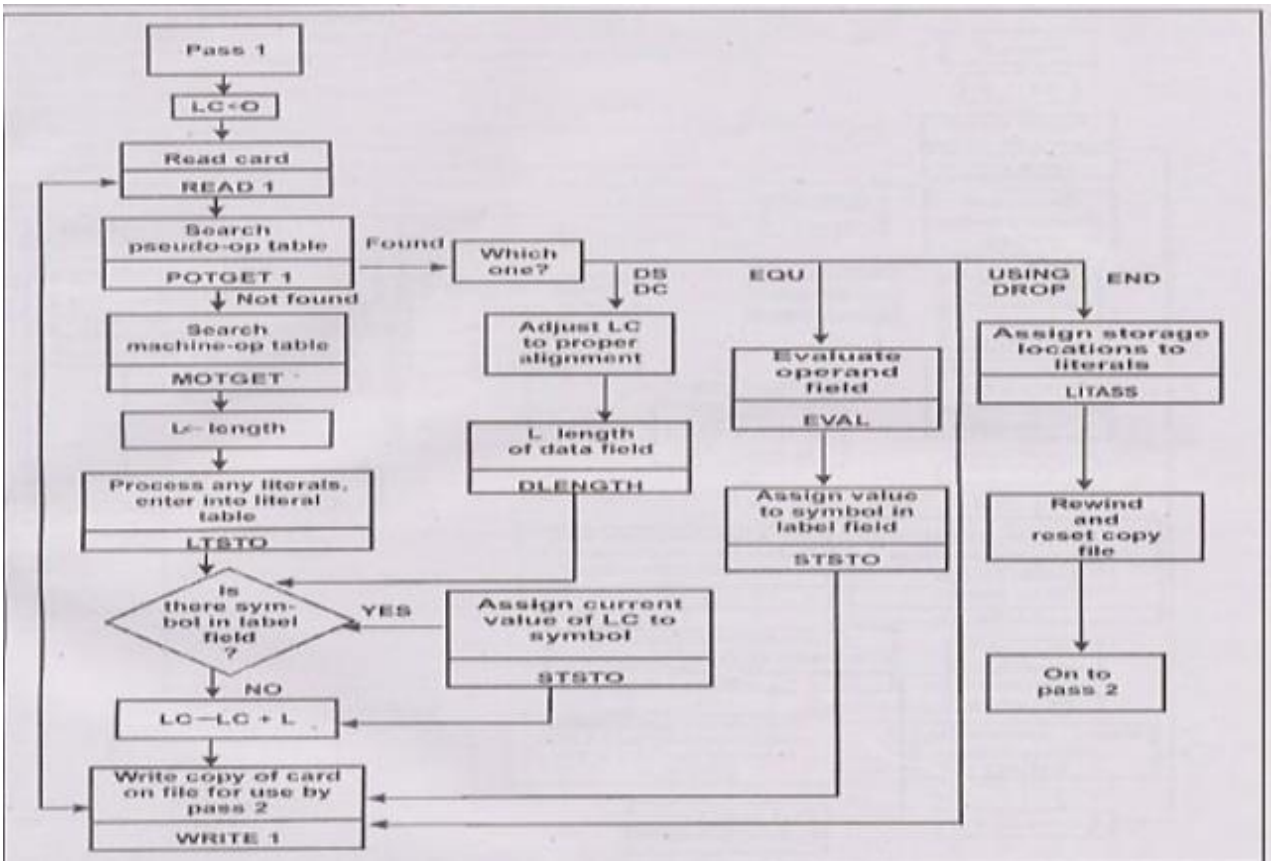


Two pass assembler (Two pass translation)

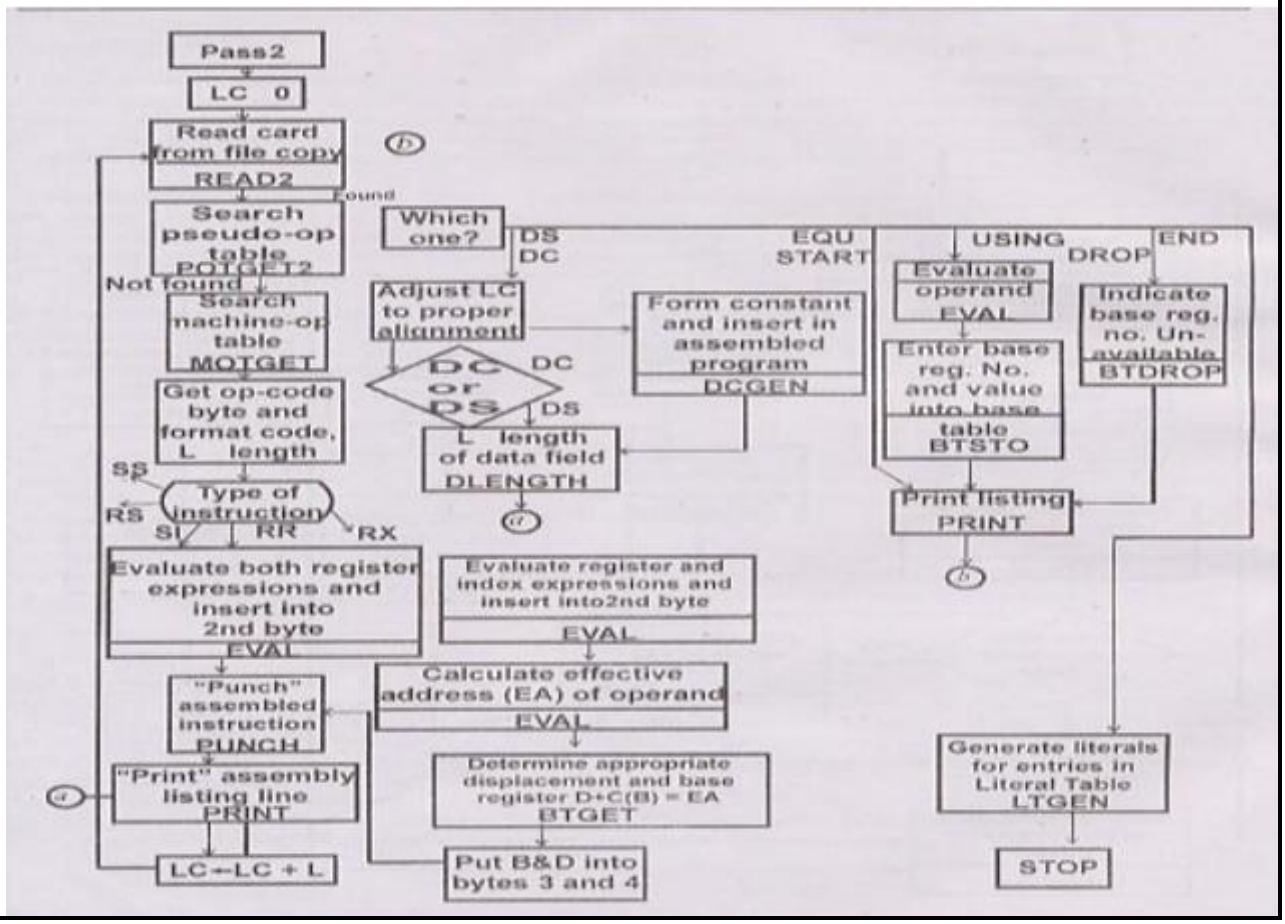


- The first pass performs analysis of the source program.
- The first pass performs Location Counter processing and records the addresses of symbols in the symbol table.
- It constructs intermediate representation of the source program.
- Intermediate representation consists of following two components:
  1. Intermediate code
  2. Data structures

Design diagrams (if any): **Flow chart for Pass-1**



Flow chart for Pass-2



Algorithms :

PASS I

• Initialize location counter, entries of all tables as zero.
• Read statements from input file one by one.
• While next statement is not END statement
I. Tokenize or separate out input statement as label,numonic,operand1,operand2
II. If label is present insert label into symbol table.
III. If the statement is LTORG statement processes it by making it's entry into literal table, pool table and allocate memory.
IV. If statement is START or ORIGEN Process location counter accordingly.
V. If an EQU statement, assign value to symbol by correcting entry in symbol table.
VI. For declarative statement update code, size and location counter.
VII. Generate intermediate code.
VIII. Pass this intermediate code to pass -2

PASS II

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 the machine_code_buffer

b) if an LTORG statement

I) process literals in LITTAB[POOLTAB[pooltab_ptr]]… LITTAB[POOLTAB[pooltab_ptr+1]] similar to processing of constants in a dc statement.

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 or DS:

e) if an imperative statement then

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)

Input for Pass I of assembler:
1. Assembly language program in hypothetical language as per the Author Dhamdhere
2. OPTB
3. Condition code table
4. Register table

Input for PassII of assembler:
1. IC
2. SYMTAB
3. LITTAB
4. POOLTAB
5. OPTAB

Output of PassI of assembler:
1. IC
2. SYMTAB
3. LITTAB
4. POOLTAB

Output of PassII of assembler:
Machine code

Software Requirement:
Operating System recommended :- 64-bit Open source Linux or its derivative

Programming tools recommended: - Eclipse IDE

Hardware Requirement:I3 and I5 machines

Conclusion:

1. Input assembly language program is processed by applying PassI algorithm of assembler and intermediate data structures, SYMTAB, LITTAB, POOLTAB, IC, are generated.
2. The intermediate data structures generated in PassI of assembler are given as input to the PassIIof assembler, processed by applying PassII algorithm of assembler and machine code is generated

Frequently Asked Questions:

1. What is two pass assembler?
2. What is the significance of symbol table?
3. Explain the assembler directives EQU, ORIGIN.
4. Explain the assembler directives START, END, LTORG.
5. What is the use of POOLTAB and LITTAB?
6. How literals are handled in pass I?
7. What are the tasks done in Pass I?
8. How error handling is done in pass I?
9. Which variant is used in implementation? Why?
10. Which intermediate data structures are designed and implemented in PassI?
11. What is the format of a machine code generated in PassII?
12. What is forward reference? How it is resolved by assembler?
13. How error handling is done in pass II?
14. What is the difference between IS, DL and AD.
15. What are the tasks done in Pass II?

Instructions :

Handwritten write-up as follows :

- Name of Student: Batch:T1/T2
- Subject:LP-1
- Assessment table
- Date of Performance:_____Date of Completion_____
- Title
- Objectives
- Problem statement
- Software and hardware requirements
- Theory –What is Assembler?
- Types of Statement format. and Explain each Statement.
- Flowchart of Pass-1 and Pass-2
- Conclusion

Attach Program code and Its Output