Fokane Sakshi Anil
TE-A-42

Page No.
Date

Assignment No : 1 (1)

Pass - 1 Assembler

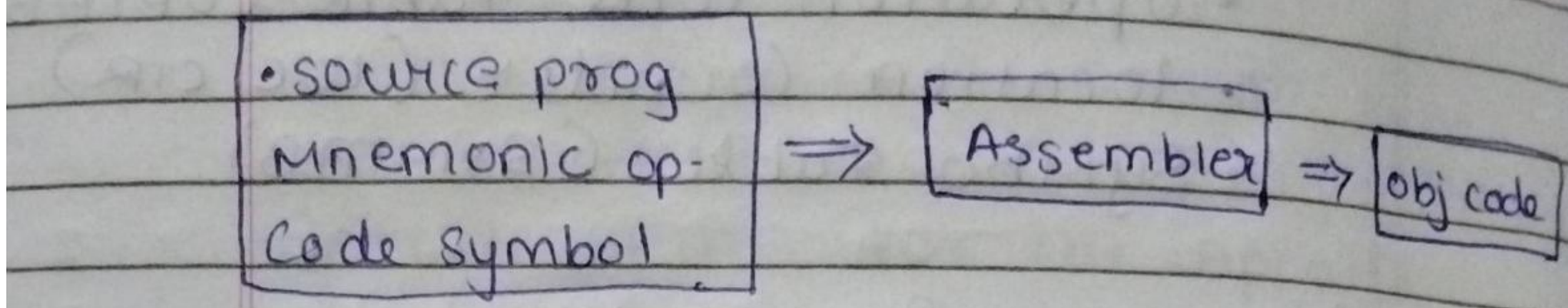
* Aim :- To implement pass - 1 assembler.

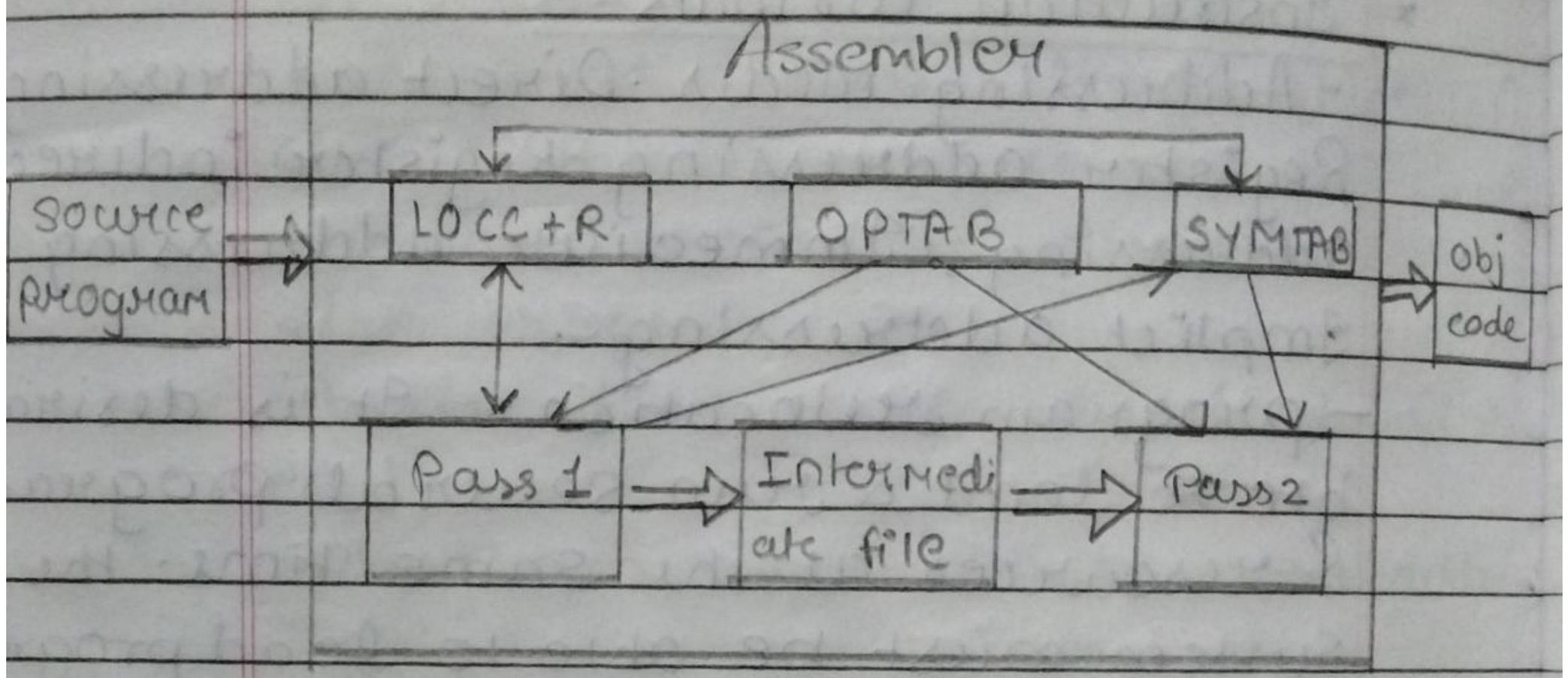
* Problem Statement :- Design Suitable data structure & implement pass 1 of two pass assembler feature. for pseudo-machine in java using object oriented feature. Implementation should consist of a few instruction from each category & few directives.

* Theory :- Assembly language :-
An assembly language is a low level programming language for a computer. or other programmable device in which there is a very strong (generally one to one) corresponding between the language and the architectures machine code instruction each assembly language is specific to particular computer architecture in contract to most high level language.

* Assembler :-
Assembly language is converted into executable machine code by utility program referred to as an assable the conversion process is refered to

assembly or assembling the code.





Translate assembly language program to object programs or machine code is called as Assembler.

* Assembler Directives :-

- Assembler directive are pseudo instructions
  - They will not be translate into M/c instruction
  - They only provide instruction/direction information to the assembler.
- Basic assembler directives
  - START: specify name & starting address
  - END :- Indicate the END of source program
  - EQU :- Is used to replace number by a Symbol

* Three main DataStructure :~
  - operation code Table (OPTAB)
  - location counter (LOC CTR)
  - Symbol Table (SYMTAB)

* Instruction formats :-
- Addressing modes : Direct addressing. Register addressing Register indirect addressing, Immediate addressing. Implict addressing.
- program relocation :- It is desirable to load & run several program & resources at the same time. the system must be able to load progra into memory wherever there is room.

* DataStructure for assembler :-
Op-code table :~
looked up for the translation of mnemonic code.
Key - mnemonic code.
Algorithm for pass 1 - Assembler.

begin :-
if starting address is given
LOCCTR = starting address;
else
LOCCTR = 0
while OPCODE != END do ;; OR EOF
begin

read a line from the code
if there is label
if this label is in SYMTAB then
else insert (label, LOCCTR) into SYMTAB
Search OPTAB for the opcode
if found
LOCCTR += N ;; N is length of instruct
else if this is an assembly directive
update LOCCTR is directed
else error.
write line to intermediate file.
end
program size = LOCCTR - Start address;
end.

Input :-

```
START 200
MOVER AREG ='4'
MOVEM AREG,,A
MOVER BREG,='1'
LOOP MOVER CREG,B
LTORG
ADD CREG,='6'
STOP
A DS1
B DS 1
END
```

Expected Output : Symbol table

A	208
LOOP	203
B	209

Intermediate code.

AD	01	C	200	
IS	04	1	L	1
IS	05	1	S	1
IS	04	2	L	2
IS	04	3	S	3
AD	05			
IS	01	3	L	3
IS	00			
DL	02	C	1	
DL	02	C	1	
AD	02			

Conclusion :-

Thus, we have implemented PASS-1 Assembler using Object oriented feature.

```
//Name: Fokane Sakshi Anil
// TE-A 42
//  ASSINGNMENT:GROUP_A_1

/*
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.
*/
import java.io.*;
class SymTab
{
	public static void main(String args[])throws Exception
	{
		FileReader FP=new FileReader(args[0]);
		BufferedReader bufferedReader = new BufferedReader(FP);

		String line=null;
		int line_count=0,LC=0,symTabLine=0,opTabLine=0,litTabLine=0,poolTabLine=0;

		 //Data Structures
		 final int MAX=100;
		 String SymbolTab[][]=new String[MAX][3];
		 String OpTab[][]=new String[MAX][3];
		 String LitTab[][]=new String[MAX][2];
		 int PoolTab[]=new int[MAX];
		 int litTabAddress=0;
/*-------------------------------------------------------------------------------------------------------*/

		 System.out.println("_____________________________________________________________");
		   while((line = bufferedReader.readLine()) != null)
		   {
			 String[] tokens = line.split("\t");
			if(line_count==0)
			{
				LC=Integer.parseInt(tokens[2]);
				//set LC to operand of START
				for(int i=0;i<tokens.length;i++)		//for printing the input program
					System.out.print(tokens[i]+"\t");
				System.out.println("");
			}
			else
```

```
                {
                        for(int i=0;i<tokens.length;i++) //for printing the input program
                                System.out.print(tokens[i]+"\t");
                        System.out.println("");
                        if(!tokens[0].equals(""))
                        {

                                //Inserting into Symbol Table
                                SymbolTab[symTabLine][0]=tokens[0];
                                SymbolTab[symTabLine][1]=Integer.toString(LC);
                                SymbolTab[symTabLine][2]=Integer.toString(1);
                                symTabLine++;
                        }
                        else
if(tokens[1].equalsIgnoreCase("DS")||tokens[1].equalsIgnoreCase("DC"))
                        {
                                //Entry into symbol table for declarative statements
                                SymbolTab[symTabLine][0]=tokens[0];
                                SymbolTab[symTabLine][1]=Integer.toString(LC);
                                SymbolTab[symTabLine][2]=Integer.toString(1);
                                symTabLine++;
                        }

                        if(tokens.length==3 && tokens[2].charAt(0)=='=')
                        {
                                //Entry of literals into literal table
                                LitTab[litTabLine][0]=tokens[2];
                                LitTab[litTabLine][1]=Integer.toString(LC);
                                litTabLine++;
                        }

                        else if(tokens[1]!=null)
                        {
                                        //Entry of Mnemonic in opcode table
                                OpTab[opTabLine][0]=tokens[1];


        if(tokens[1].equalsIgnoreCase("START")||tokens[1].equalsIgnoreCase("END")||tokens[1].equals
IgnoreCase("ORIGIN")||tokens[1].equalsIgnoreCase("EQU")||tokens[1].equalsIgnoreCase("LTORG"))
        //if Assembler Directive
                                {
                                        OpTab[opTabLine][1]="AD";
```

```
                                        OpTab[opTabLine][2]="R11";

                                    }
                                    else
if(tokens[1].equalsIgnoreCase("DS")||tokens[1].equalsIgnoreCase("DC"))
                                    {
                                        OpTab[opTabLine][1]="DL";
                                        OpTab[opTabLine][2]="R7";

                                    }
                                    else
                                    {
                                        OpTab[opTabLine][1]="IS";
                                        OpTab[opTabLine][2]="(04,1)";
                                    }
                                opTabLine++;
                                }
                        }
                        line_count++;
                        LC++;
                    }


        System.out.println("_____");

                        //print symbol table
                        System.out.println("\n\n          SYMBOL TABLE          ");
                        System.out.println(" ------------------------ ");
                        System.out.println("SYMBOL\tADDRESS\tLENGTH");
                        System.out.println(" ------------------------ ");
                        for(int i=0;i<symTabLine;i++)

        System.out.println(SymbolTab[i][0]+"\t"+SymbolTab[i][1]+"\t"+SymbolTab[i][2]);
                        System.out.println(" ------------------------ ");


                        //print opcode table
                        System.out.println("\n\n          OPCODE TABLE          ");
                        System.out.println(" ------------------------- ");
                        System.out.println("MNEMONIC\tCLASS\tINFO");
                        System.out.println(" ------------------------- ");
                        for(int i=0;i<opTabLine;i++)
```

```
                System.out.println(OpTab[i][0]+"\t\t"+OpTab[i][1]+"\t"+OpTab[i][2]);
            System.out.println(" ---------------------------- ");

            //print literal table
            System.out.println("\n\n   LITERAL TABLE                        ");
            System.out.println(" ---------------- ");
            System.out.println("LITERAL\tADDRESS");
            System.out.println(" ---------------- ");
            for(int i=0;i<litTabLine;i++)
                System.out.println(LitTab[i][0]+"\t"+LitTab[i][1]);
            System.out.println(" ----------------- ");


            //intialization of POOLTAB
            for(int i=0;i<litTabLine;i++)
            {
                if(LitTab[i][0]!=null && LitTab[i+1][0]!=null ) //if literals are present
                {
                    if(i==0)
                    {
                        PoolTab[poolTabLine]=i+1;
                        poolTabLine++;
                    }
                    else
if(Integer.parseInt(LitTab[i][1])<(Integer.parseInt(LitTab[i+1][1]))-1)
                    {
                        PoolTab[poolTabLine]=i+2;
                        poolTabLine++;
                    }
                }
            }
            //print pool table
            System.out.println("\n\n   POOL TABLE              ");
            System.out.println(" ---------------- ");
            System.out.println("LITERAL NUMBER");
            System.out.println(" ---------------- ");
            for(int i=0;i<poolTabLine;i++)
                System.out.println(PoolTab[i]);
            System.out.println(" ---------------- ");


        // Always close files.
```

```
                    bufferedReader.close();
        }
}

/*
OUTPUT-
_____________________________________________________
        START 100
        READ    A
LABLE   MOVER A,B
        LTORG
                ='5'
                ='1'
                ='6'
                ='7'
        MOVEM           A,B
        LTORG
                ='2'
LOOP    READ    B
A       DS      1
B       DC      '1'
                ='1'
        END
_____________________________________________________


        SYMBOL TABLE
- - - - - - - - - - - - - - - -
SYMBOL          ADDRESS         LENGTH
--------------------------------
LABLE   102     1
LOOP    111     1
A       112     1
B       113     1
--------------------------------


        OPCODE TABLE
- - - - - - - - - - - - - - - - -
MNEMONIC    CLASS   INFO
- - - - - - - - - - - - - - - - -
READ            IS      (04,1)
```

```
MOVER           IS      (04,1)
LTORG           AD      R11
MOVEM                   IS      (04,1)
LTORG           AD      R11
READ            IS      (04,1)
DS              DL      R7
DC              DL      R7
END             AD      R11
---------------------------------


  LITERAL TABLE
-----------
LITERALADDRESS
--------------------
='5'      104
='1'      105
='6'      106
='7'      107
='2'      110
='1'      114
----------------------


  POOL TABLE
-----------
LITERAL NUMBER
-----------
1
5
6
------------


*/
```