Name :- Rushikesh Karbhari Palve
Roll No. 31258

Date :	Page No.
/ /20	

Assignment No. 4

Dop : 13-09-2021 Dos : 20-09-2021

Problem Definition :-

Design suitable data structures and implement pass-II of a two pass assembler for pseudo-machine in Java.
Implementation should consists of a few instructions from each category and few assembler directives. The output of pass-I (intermediate code file and symbol table) should be input for pass-II.

Learning Objectives :-

① To understand the concept of pass-II of two pass assembler.
② To implement the pass-II of two pass assembler.
③ To understand Advanced Assembler Directives.

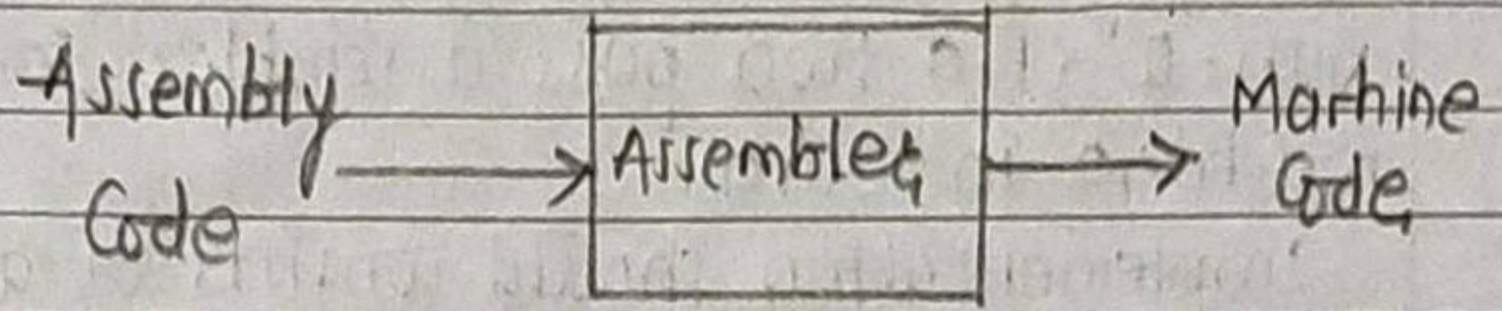
Outcomes :-

After completion of assignment, students will be able to -
① Implement pass-II of two pass assembler.
② Understand concept of Advanced Assembler Directive.

Theory concepts :-

Introduction of Assembler :-

Assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.



It generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code. Now, if assembler do all this work in one scan then it is called single pass assembler, otherwise if it does in multiple scans then called multiple pass assembler. Here assembler divide these tasks into two passes.

Pass 1 :-

1.) Define symbols and literals and remember them in symbol table and literal table respectively.
2.) Keep track of location counter.
3.) Process pseudo-operations.

Pass 2 :-

1.) Generate object code by converting symbolic op-code into respective numeric op-code.
2.) Generate data for literals and look for

values of symbols.

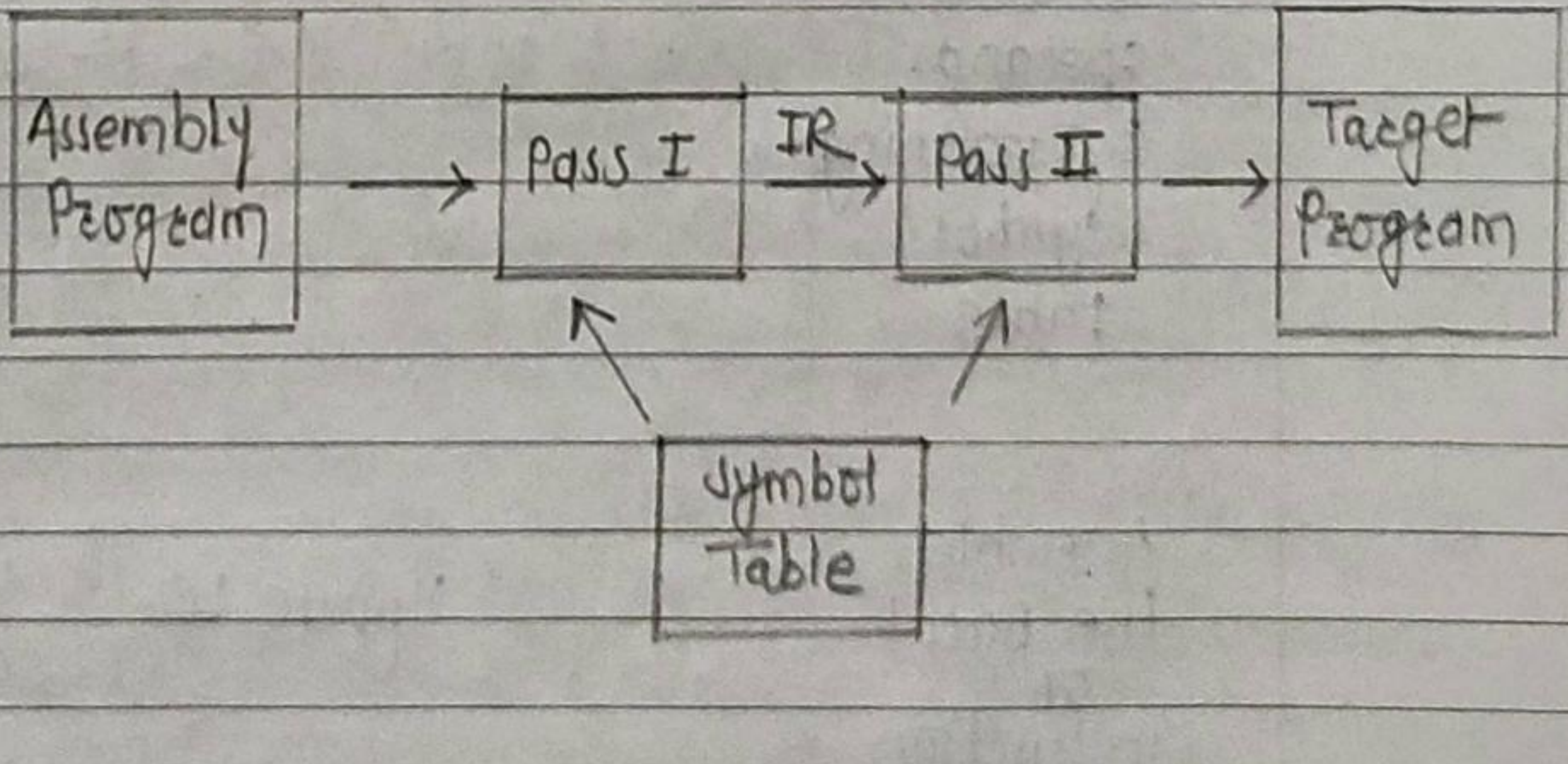
Working of Pass - 2 :-

Pass 2 of assembler generates machine code by converting symbolic machine-opcodes into their respective bit configuration (machine understandable form.). It stores all machine-opcodes in MOT table (opcode table) with symbolic code, their length and their bit configuration. It will also process pseudo-ops and will store them in POT table (pseudo-op table).

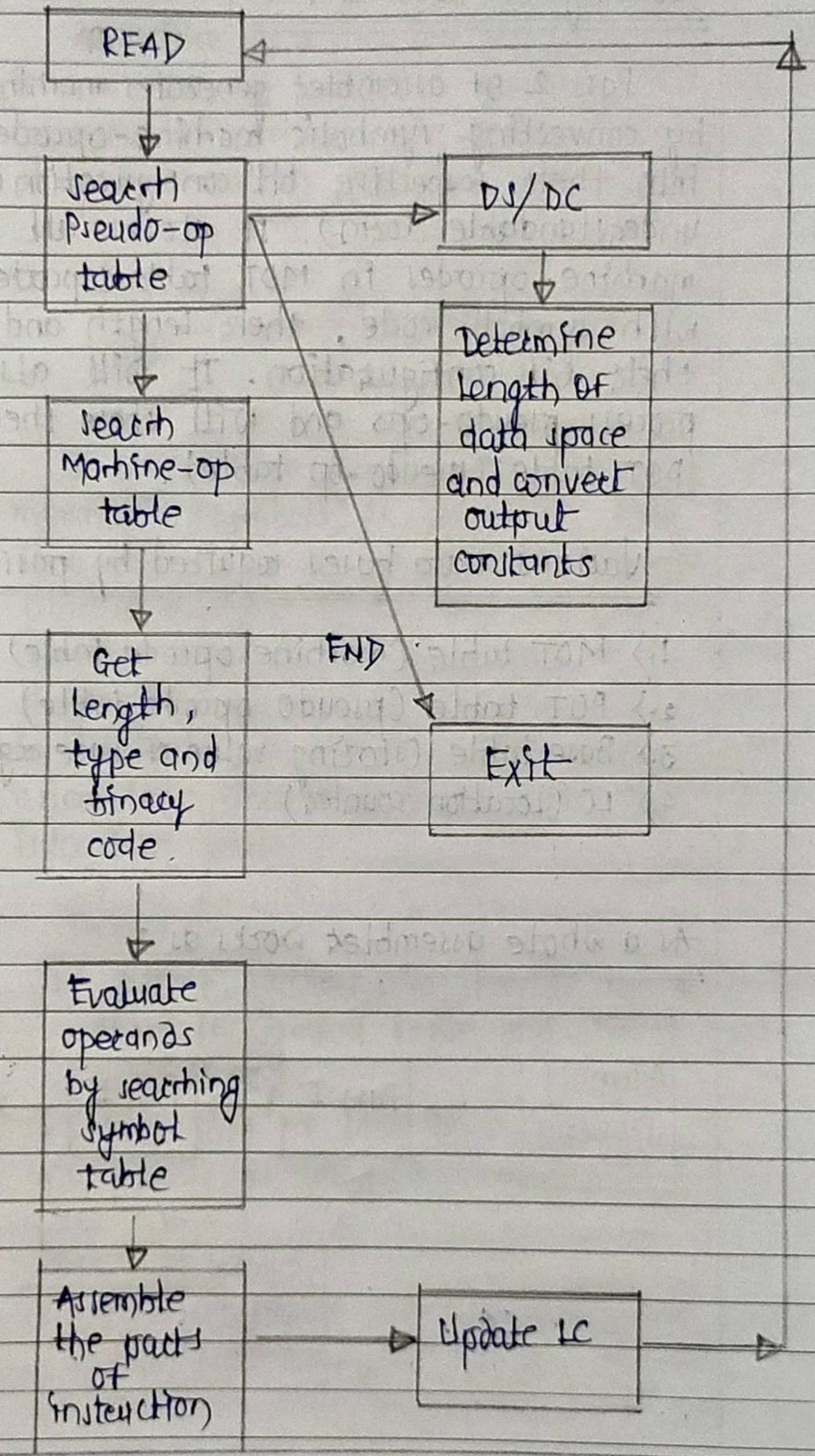
Various Data bases required by pass-2 :-

1.) MOT table (machine opcode table)
2.) POT table (pseudo opcode table)
3.) Base table (storing value of base register)
4.) LC (Location counter)

As a whole assembler works as :-



Flowchart :-


READ
Search Pseudo-op table
DS/DC
Determine length of data space and convert output constants
Search Machine-op table
END
Get length, type and binary code.
Exit
Evaluate operands by searching symbol table
Assemble the parts of instruction
Update LC

Example :-

For the following assembly languge code show the contents of symbol table, literal table and also generate intermediate and target code - (Assume suitable op-codes and instruction length and clearly indicate the assumptions made -

		Assembly Program	LC	Intermediate Code
1.		START 200		(AD, 01) (C, 200)
2.		MOVER AREG, ='4'	200	(IS, 04) (1) (L, 1)
3.		MOVEM AREG, A	201	(IS, 05) (1) (S, 1)
4.		MOVER BREG, ='1'	202	(IS, 04) (2) (L, 2)
5.	LOOP	MOVER CREG, B	203	(IS, 04) (3) (S, 3)
6.		LTORG	204 →	(AD, 05) (DL, 01) (C, 4)
7.		ADD CREG, ='6'	205 →	(DL, 01) (C, 1)
8.		STOP	206	(IS, 01) (3) (L, 3)
9.	A	DS 1	207	(IS, 00)
10.	B	DS 1	208	(DL, 02) (C, 1)
11.		END	209	(DL, 02) (C, 1)
				(AD, 02)
			210	(DL, 01) (C, 6)

Symbol Table

	Symbol	Address
1)	A	208
2)	LOOP	203
3)	B	209

Literal Table

	Literal	Address
1)	='4'	204
2)	='1'	205
3)	='6'	210

Pool Table

Pool Table
1
3

Intermediate code	LC	Machine Code		
1. (AD, 01) (C, 200)				
2. (IS, 04)(1) (L, 1)	200	04	1	204
3. (IS, 05) (1)(S, 1)	201	05	1	208
4. (IS, 04) (2)(L, 2)	202	04	2	210
5. (IS, 04) (3) (S, 3)	203	04	3	209
6. (AD, 05)	204	00	0	004
7. (DL, 01) (C, 4)	205	00	0	006
8. (DL, 01) (C, 1)	206	01	3	205
9. (IS, 01) (3) (L, 3)	207	00	0	000
10. (IS, 00)	208			
11 (DL, 02) (C, 1)	209			
12. (DL, 02) (C, 1)	210	00	0	001
13. (AD, 02)				
14 (DL, 01) (C, 6)				

Algorithm :-

```
1. Start.
2. Code_area_address = address of Code_area
3. For each entry in IC[ ]
   {
       a) If an impecative statement
           (i) Read LC
           (ii) Get opcode
           (iii) Get operand / literal address from
                 the symbol / literal table.
           (iv) Assemble instruct in machine code_buffer
           (v) Move contents of machine code_buffer
               in code_area at the address LC +
               code_area_address.
```

Roll No. 31258

```
b) If a DC statement then
    (i) Read LC.
    (ii) Assemble the constant in machine_
         code_buffer.
    (iii) Move contents of machine_code_buffer
          in code_area at the address LC +
          code_area_address.

4. Write code_area into output file.

5. Stop. .
```

Test Cases :-

Test case Id	steps to be executed	Expected Output				Actual Output				Result
1.)	Intermediate Code :-	LC	Machine Code			LC	Machine Code			Pass
	(AD,01)(C,200)									
	(IS,04) (1) (L,1)	200	04	1	204	200	04	1	204	
	(IS,05) (1) (S,1)	201	05	1	208	201	05	1	208	
	(IS,04) (2) (L,2)	202	04	2	210	202	04	2	210	
	(IS,04) (3) (S,3)	203	04	3	209	203	04	3	209	
	(AD,05)	204	00	0	004	204	00	0	004	
	(DL,01) (C,4)	205	00	0	006	205	00	0	006	
	(DL,01) (C,1)	206	01	3	205	206	01	3	205	
	(IS,01) (3) (L,3)	207	00	0	000	207	00	0	000	
	(IS,00)	208				208				
	(DL,02) (C,1)	209				209				
	(DL,02) (C,1)	210	00	0	001	210	00	0	001	

Test case Id	Steps to be executed	Expected Output	Actual Output	Result
	(AD, 02)			
	(DL, 01) (C, 6)			

Symbol Table :

Symbol	Address
1) A	208
2.) LOOP	203
3.) B	209

Literal Table :-

Literal	Address
1.) ='4'	204
2.) ='1'	203
3.) ='6'	210

Pool Table

1
3

CONCLUSION :-

We understood the concept of pass-II of two pass assembler and ~~to~~ successfully implemented the pass-II of two pass assembler.

CODE :-

```
/*
 * Problem Statement :-
 *  Design suitable data structures and implement pass-II of a two-pass assembler for
 *  Pseudo-machine in Java.
 *  Implementation should consists of a few instructions from each category and few
 *  assembler directives. The output of Pass-
I (intermediate code file and symbol table)
 *  should be input for pass-II.
 */


#include<bits/stdc++.h>
using namespace std;

class Entry
{
    public:
        string symbol;
        int address, index;
        Entry()
        {
            symbol = " ";
            address = index = 0;
        }
        Entry(string s, int add, int idx = 0)
        {
            symbol = s;
            address = add;
            index = idx;
        }

        void setSymbol(string sym)
        {
            symbol = sym;
        }
        void setAddress(int add)
        {
            address = add;
        }
        void setIndex(int idx)
        {
            index = idx;
        }

        string getSymbol()
        {
            return symbol;
        }
        int getAddress()
        {
            return address;
        }
        int getIndex()
        {
            return index;
        }
};

void tokenize(string s, vector<string> &res, string del = "\t")
{
    int start = 0;
    int end = s.find(del);
    if(end == 0) res.push_back("EMPTY");
    while (end != -1)
    {
        if(end-start > 0) res.push_back(s.substr(start, end - start));
        start = end + del.size();
        end = s.find(del, start);
    }
    if(s.substr(start, end - start) != "") res.push_back(s.substr(start, end -
 start));
}

class Pass2
{
    private:
        vector<Entry> SYMTAB,LITTAB;
    public:
        void readTables();
        void generateCode(string filename);
};

void Pass2::readTables()
{
    string line;
    ifstream fin1("SYMTAB.txt");
    while(getline(fin1, line))
    {
        vector<string> parts;
        tokenize(line, parts);
        Entry e(parts[1], stoi(parts[2]), stoi(parts[0]));
        SYMTAB.push_back(e);
    }
    ifstream fin2("LITTAB.txt");
    while(getline(fin2, line))
    {
        vector<string> parts;
        tokenize(line, parts);
        string temp = "";
        for(unsigned int i=0; i<parts[1].size(); i++)
        {
            if(parts[1][i] != '\'' && parts[1][i] != '=')
            {
                temp += parts[1][i];
            }
        }
        parts[1] = temp;
        Entry e(parts[1], stoi(parts[2]), stoi(parts[0]));
        LITTAB.push_back(e);
    }
}

void Pass2::generateCode(string filename)
```

```
{
    readTables();
    ifstream fin(filename);
    ofstream fout("MachineCode.txt");
    string line,code;

    while(getline(fin, line))
    {
        vector<string> parts;
        tokenize(line, parts);

        if( (parts[0].find("AD") != string::npos) || (parts[0].find("DL,02") != string
::npos) )
        {
            fout<<"\n";
        }
        else if(parts.size() == 2)
        {
            if(parts[0].find("DL") != string::npos)
            {
                string temp = "";
                for(unsigned int i=0; i<parts[0].length(); i++)
                {
                    if(isdigit(parts[0][i]))
                    {
                        temp += parts[0][i];
                    }
                }
                parts[0] = temp;

                if(stoi(parts[0]) == 1)
                {
                    temp = "";
                    for(unsigned int i=0; i<parts[1].length(); i++)
                    {
                        if(isdigit(parts[1][i]))
                        {
                            temp += parts[1][i];
                        }
                    }
                    parts[1] = temp;

                    int cons = stoi(parts[1]);
                    fout.fill('0');
                    fout<<"00\t0\t"<<setw(3)<<cons<<"\n";
                }
            }

            else if(parts[0].find("IS") != string::npos)
            {
                string temp = "";
                for(unsigned int i=0; i<parts[0].length(); i++)
                {
                    if(isdigit(parts[0][i]))
                    {
                        temp += parts[0][i];
                    }
                }
                parts[0] = temp;
                int opcode = stoi(parts[0]);

                if(opcode == 10)
                {
                    if(parts[1].find("S") != string::npos)
                    {
                        temp = "";
                        for(unsigned int i=0; i<parts[1].length(); i++)
                        {
                            if(isdigit(parts[1][i]))
                            {
                                temp += parts[1][i];
                            }
                        }
                        parts[1] = temp;

                        int symIndex = stoi(parts[1]);
                        fout.fill('0');
                        fout<<setw(2)<<opcode<<"\t0\t"<<setw(3)<<SYMTAB[symIndex-
1].getAddress()<<"\n";
                    }
                    else if(parts[1].find("L") != string::npos)
                    {
                        temp = "";
                        for(unsigned int i=0; i<parts[1].length(); i++)
                        {
                            if(isdigit(parts[1][i]))
                            {
                                temp += parts[1][i];
                            }
                        }
                        parts[1] = temp;

                        int litIndex = stoi(parts[1]);
                        fout.fill('0');
                        fout<<setw(2)<<opcode<<"\t0\t"<<setw(3)<<LITTAB[litIndex-
1].getAddress()<<"\n";
                    }
                }
            }
        }

        else if(parts.size() == 1 && parts[0].find("IS") != string::npos)
        {
            string temp = "";
            for(unsigned int i=0; i<parts[0].length(); i++)
            {
                if(isdigit(parts[0][i]))
                {
                    temp += parts[0][i];
                }
            }
            parts[0] = temp;

            int opcode = stoi(parts[0]);
            fout.fill('0');
            fout<<setw(2)<<opcode<<"\t0\t000\n";
        }

        else if(parts[0].find("IS") != string::npos && parts.size() == 3)
        {
            string temp = "";
            for(unsigned int i=0; i<parts[0].length(); i++)
            {
                if(isdigit(parts[0][i]))
                {
                    temp += parts[0][i];
                }
            }
            parts[0] = temp;
            temp = "";
            for(unsigned int i=0; i<parts[1].size(); i++)
            {
                if(parts[1][i] != '(' && parts[1][i] != ')')
                {
                    temp += parts[1][i];
                }
            }
            parts[1] = temp;
            int opcode = stoi(parts[0]);
            int regcode = stoi(parts[1]);

            if(parts[2].find("S") != string::npos)
            {
                temp = "";
                for(unsigned int i=0; i<parts[2].length(); i++)
                {
                    if(isdigit(parts[2][i]))
                    {
                        temp += parts[2][i];
                    }
                }
                parts[2] = temp;

                int symIndex = stoi(parts[2]);
                fout.fill('0');
                fout<<setw(2)<<opcode<<"\t"<<regcode<<"\t"<<setw(3)<<SYMTAB[symIndex-
1].getAddress()<<"\n";
            }
            else if(parts[2].find("L") != string::npos)
            {
                temp = "";
                for(unsigned int i=0; i<parts[2].length(); i++)
                {
                    if(isdigit(parts[2][i]))
                    {
                        temp += parts[2][i];
                    }
                }
                parts[2] = temp;

                int litIndex = stoi(parts[2]);
                fout.fill('0');
                fout<<setw(2)<<opcode<<"\t"<<regcode<<"\t"<<setw(3)<<LITTAB[litIndex-
1].getAddress()<<"\n";
```

```
            }
        }

    }
}

int main()
{
    Pass2 p1;
    p1.generateCode("Intermediate_Code.txt");
}
```

OUTPUT:-

```
"Intermediate_Code.txt"
```

```
(AD,01) (C,200)
(IS,04) 1  (L,01)
(IS,05) 1  (S,01)
(IS,04) 1  (S,01)
(IS,04) 3  (S,03)
(IS,01) 3  (L,02)
(IS,04) 1  (S,01)
(IS,04) 3  (S,03)
(IS,04) 1  (S,01)
(IS,04) 3  (S,03)
(IS,04) 1  (S,01)
(IS,07) 6  (S,04)
(DL,01) (C,5)
(DL,01) (C,1)
(IS,04) 1  (S,01)
(IS,02) 1  (L,03)
(IS,07) 1  (S,05)
(IS,00)
(AD,03) (S,2)+2
(IS,03) 3  (S,03)
(AD,03) (S,6)+1
(DL,02) (C,1)
(AD,04) (C,LOOP)
(DL,02) (C,1)
(AD,02)
(DL,01) (C,1)
```

```
"SYMTAB.txt"
```

```
1  A  215
2  LOOP 202
3  B  216
4  NEXT 212
5  BACK 202
6  LAST 214
```

```
"LITTAB.txt"
```

```
1  ='5' 217
2  ='1' 218
```

```
MachineCode.txt
```

```
04 1  217
05 1  215
04 1  215
04 3  216
01 3  218
04 1  215
04 3  216
04 1  215
04 3  216
04 1  215
07 6  212
00 0  005
00 0  001
04 1  215
02 1  000
07 1  202
00 0  000

03 3  216





00 0  001
```