1. Pass-1 Assembler

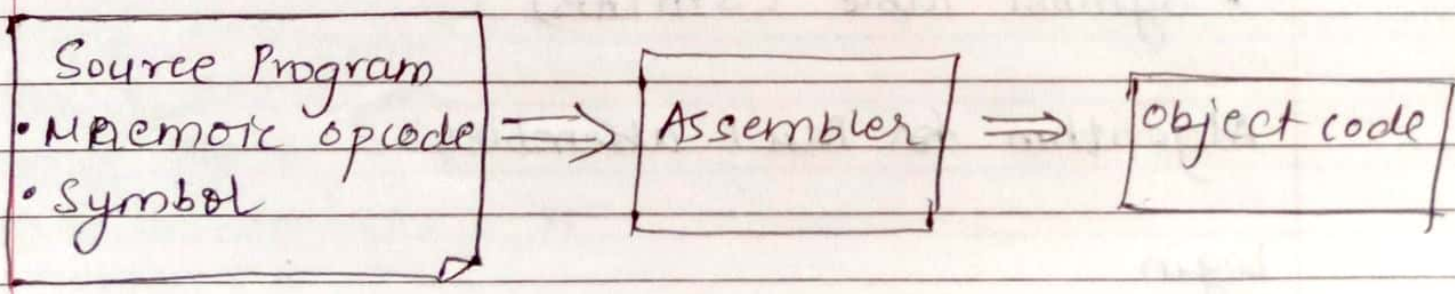
Aim: To implement Pass-1 Assembler

Problem Statement:-
Design suitable data structures & implement pass-I of a 2 pass assembler pseudo machine in Java using object oriented feature. Implementation should consist of a few instructions from each category and few assembler directives.

Theory:-

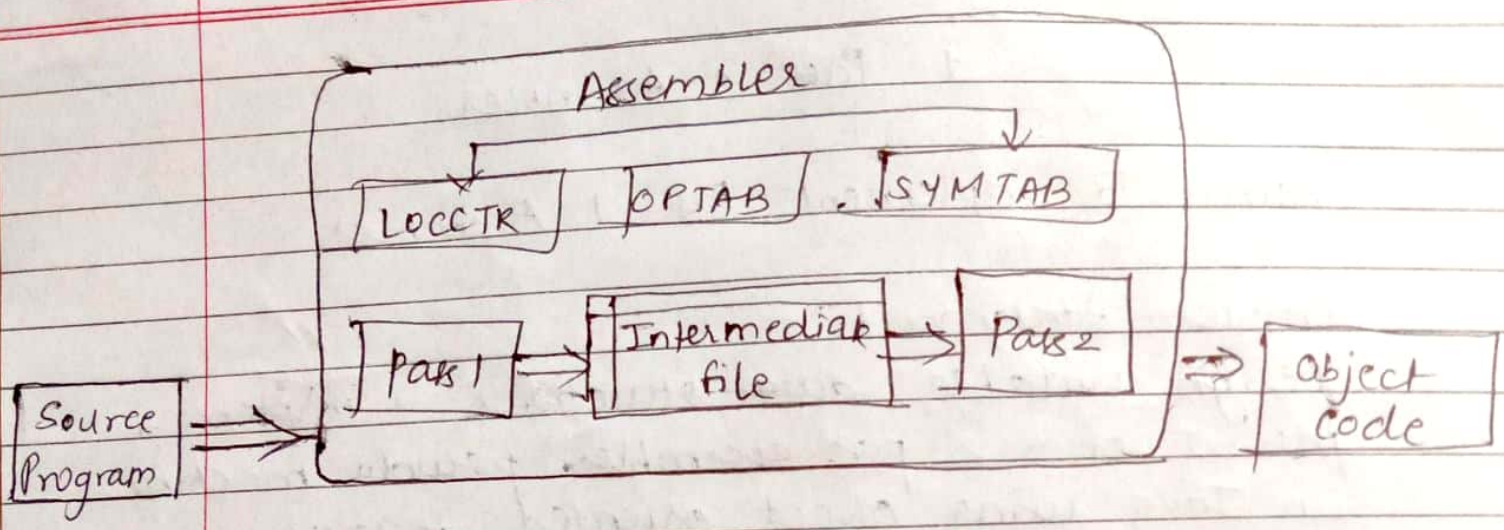
Assembly Language - It is a low-level programming language for a computer, or other programmable device, in which there is a very strong (one to one) correspondence.

Assembler :- It is converted into executable machine code by a utility program.



Scanned with CamScanner

Rainbow
PAGE :
DATE : / /



Assembler Directives :-

- Assembler directives are pseudo instructions
  - They will not be translated into machine instructions.
  - They only provide instruction to assembler.

Assembler directives :-

1. START
2. END
3. EQU

Three Main Data Structures

- Operation Code Table (OPTAB)
- Location Counter (LOCCTR)
- Symbol Table (SYMTAB)

Algorithm for Pass I assembler :-

```
begin
    if starting address is given
        LOCCTR = starting address
    else
        LOCCTR = 0;
```

Scanned with CamScanner

PAGE:
DATE: / /

```
while OPCODE != END do  ;; or EOF
    begin
    read a line from the code
    if there is a label
        if this label is in SYMTAB, then error
        else insert (label, LOCCTR) into SYMTAB
    search OPTAB for the op code
    if found
        LOCCTR += N  ;; N is the length of
                     this instruction (4 for MIPS)
    else if this is an assembly directive
        update LOCCTR as directed
    else error
    write line to intermediate file.
    end
program size = LOCCTR - starting address;
```

INPUT:

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

Scanned with CamScanner

PAGE:
DATE: / /

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	8	L	3
IS	00			
DL	02	C	1	
DL	02	C	1	
AD	02			

Conclusion :-

Thus, we have implemented PASS-I Assembler using Object oriented features.

Scanned with CamScanner

Assignment No. 01 [Pass 1 Assembler]

Problem Satement: Design suitable data structures and implement pass-I of a twopass assembler for pseudo-machine in Java using object oriented feature. Implementation should consist of a few instructions from each category and few assembler directives

---

1. Pass 1 Program:

```
import
java.io.BufferedReader;
import java.io.*; import
java.io.IOException; import
java.util.*;

public class Pass1 { public static void
        main(String[] args) {

                BufferedReader br = null;
                FileReader fr = null;

                FileWriter fw = null;
                BufferedWriter bw = null;

                try {
                        String inputfilename = "/home/sagar-ravan/Desktop/Input.txt";
                        fr = new FileReader(inputfilename); br = new
                        BufferedReader(fr);

                        String OUTPUTFILENAME = "/home/sagar-ravan/Desktop/IC.txt";
                        fw = new FileWriter(OUTPUTFILENAME);
                        bw = new BufferedWriter(fw);

                        Hashtable<String, String> is = new Hashtable<String, String>();
                        is.put("STOP", "00"); is.put("ADD", "01"); is.put("SUB",
                        "02"); is.put("MULT", "03"); is.put("MOVER", "04");
                        is.put("MOVEM", "05"); is.put("COMP", "06"); is.put("BC",
                        "07"); is.put("DIV", "08"); is.put("READ", "09");
                        is.put("PRINT", "10");

                        Hashtable<String, String> dl = new Hashtable<String, String>();
                        dl.put("DC", "01"); dl.put("DS", "02");
                        Hashtable<String, String> ad = new Hashtable<String, String>();

                        ad.put("START", "01");
                        ad.put("END", "02");
                        ad.put("ORIGIN", "03");
```

Scanned with CamScanner

```
ad.put("EQU", "04");
ad.put("LTORG", "05");

Hashtable<String, String> symtab = new Hashtable<String, String>();
Hashtable<String, String> littab = new Hashtable<String, String>();
ArrayList<Integer> pooltab = new ArrayList<Integer>();

String sCurrentLine; int
locptr = 0; int litptr = 1; int
symptr = 1; int pooltabptr =
1; sCurrentLine =
br.readLine();

String s1 = sCurrentLine.split(" ")[1];
if (s1.equals("START")) {
        bw.write("AD \t 01 \t");
        String s2 = sCurrentLine.split(" ")[2];
        bw.write("C \t" + s2 + "\n");
        locptr = Integer.parseInt(s2);
}

while ((sCurrentLine = br.readLine()) != null) { int mind_the_LC = 0;
        String type = null; int flag2 = 0; // checks whether addr is
        assigned to current symbol

        String s = sCurrentLine.split(" |\\,")[0]; // consider the first word in the
line

        for (Map.Entry m : symtab.entrySet()) { // allocating addr to arrived
symbols if (s.equals(m.getKey())) {
                m.setValue(locptr);
                flag2 = 1;
            }
        }
        if (s.length() != 0 && flag2 == 0) { // if current string is not " " or
addr is not assigned,

    // then the current string must be a new symbol.
          symtab.put(s, String.valueOf(locptr));
              symptr++;
        }
        int isOpcode = 0; // checks whether current word is an opcode or not

        s = sCurrentLine.split(" |\\,")[1]; // consider the second word in the
line

        for (Map.Entry m : is.entrySet()) { if (s.equals(m.getKey())) {
                bw.write("IS\t" + m.getValue() + "\t"); // if match found
in imperative stmt
```

Scanned with CamScanner

```
                                        type = "is";
                                        isOpcode = 1;
                                }
                        }

                        for (Map.Entry m : ad.entrySet()) { if (s.equals(m.getKey())) {
                                bw.write("AD\t" + m.getValue() + "\t"); // if match
found in Assembler Directive type = "ad"; isOpcode = 1;
                                }
                        }
                        for (Map.Entry m : dl.entrySet()) { if (s.equals(m.getKey())) {
                                bw.write("DL\t" + m.getValue() + "\t"); // if match
found in declarative stmt type = "dl"; isOpcode = 1;
                                }
                        }

                        if (s.equals("LTORG")) {
                                pooltab.add(pooltabptr);
                                for (Map.Entry m : littab.entrySet()) { if (m.getValue() == "")
                                        { // if addr is not assigned to the
literal
                                                m.setValue(locptr);
                                                locptr++;
                                                pooltabptr++;
                                                mind_the_LC = 1;
                                                isOpcode = 1;
                                        }
                                }
                        }

                        if (s.equals("END")) {
                                pooltab.add(pooltabptr);
                                for (Map.Entry m : littab.entrySet()) {
                                        if (m.getValue() == "") {
                                                m.setValue(locptr);
                                                locptr++; mind_the_LC =
                                                1;
                                        }
                                }
                        }

                        if (s.equals("EQU")) { symtab.put("equ",
                                String.valueOf(locptr));
                        }

                        if (sCurrentLine.split(" |\\,").length > 2) { // if there are 3 words
                                s = sCurrentLine.split(" |\\,")[2]; // consider the 3rd word

                                // this is our first operand.
```

Scanned with CamScanner

```
                        // it must be either a
                        Register/Declaration/Symbol if
                        (s.equals("AREG")) { bw.write("1\t"); isOpcode
                        = 1;
                        } else if (s.equals("BREG")) {
                                bw.write("2\t");
                                isOpcode = 1;
                        } else if (s.equals("CREG")) {
                                bw.write("3\t");
                                isOpcode = 1;
                        } else if (s.equals("DREG")) {
                                bw.write("4\t");
                                isOpcode = 1;
                        } else if (type == "dl") {
                                bw.write("C\t" + s + "\t");
                        } else { symtab.put(s, ""); // forward referenced
                        symbol }
                }

                if (sCurrentLine.split(" |\\,").length > 3) { // if there are 4 words

                        s = sCurrentLine.split(" |\\,")[3]; // consider 4th word.


        // this is our 2nd operand

        // it is either a literal, or a symbol if
                        (s.contains("=")) {
                        littab.put(s, "");
                       bw.write("L\t" + litptr + "\t");
                                isOpcode = 1;
                                litptr++;
                        } else { symtab.put(s, ""); // Doubt : what if the current
                                symbol
is already present in SYMTAB?
                                                                        // Overwrite?
                                bw.write("S\t" + symptr + "\t");
                                symptr++;
                                        }
                }

                    bw.write("\n"); // done with a line.

                if (mind_the_LC == 0)
                        locptr++;
        }

        String f1 = "/home/sagar-ravan/Desktop/SYMTAB.txt";
                    FileWriter fw1 = new FileWriter(f1);
        BufferedWriter bw1 = new BufferedWriter(fw1); for
        (Map.Entry m : symtab.entrySet()) { bw1.write(m.getKey()
```

Scanned with CamScanner

```
                + "\t" + m.getValue() + "\n");
                System.out.println(m.getKey() + " " + m.getValue());
                }

                String f2 = "/home/sagar-ravan/Desktop/LITTAB.txt";
                        FileWriter fw2 = new FileWriter(f2);
                BufferedWriter bw2 = new BufferedWriter(fw2); for
                (Map.Entry m : littab.entrySet()) { bw2.write(m.getKey() +
                "\t" + m.getValue() + "\n"); System.out.println(m.getKey()
                + " " + m.getValue());
                }

                String f3 = "/home/sagar-ravan/Desktop/POOLTAB.txt";
                        FileWriter fw3 = new FileWriter(f3);
                BufferedWriter bw3 = new BufferedWriter(fw3);
                for (Integer item : pooltab) {
                        bw3.write(item + "\n");
                        System.out.println(item);
                }

                bw.close();
                bw1.close();
                bw2.close();
                bw3.close();

        } catch (IOException e) {
                e.printStackTrace();
        }

    }

}
```

PASS 1 - ASSEMBLER OUTPUT:

```
Pritam-spos@pritam-HP:~/Desktop$ javac Pass1.java

Note: Pass1.java uses unchecked or unsafe operations.

Note: Recompile with –Xlint:unchecked for details.

Pritam-spos@-HP:~/Desktop$ java Pass1 Input.txt

A  8

LOOP 3
```

Scanned with CamScanner

B 9

= ‘4’ 4

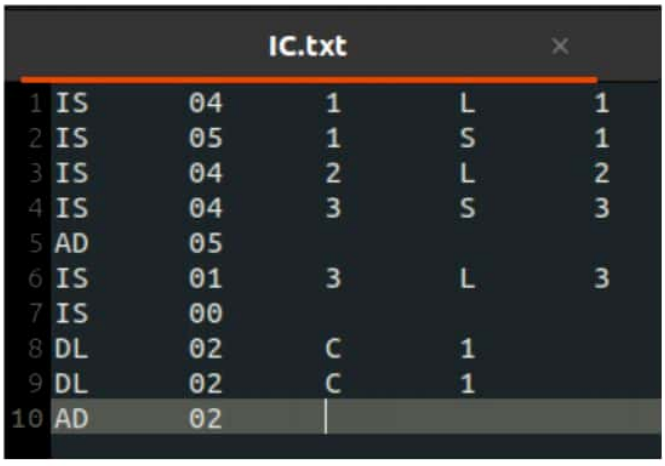
= ‘6’ 10

= ‘1’ 5

1

3

IC.txt



SYMTAB.txt

Scanned with CamScanner

SYMTAB.txt

```
1 A         8
2 LOOP      3
3 B         9
```

LITTAB.txt
POOLTAB.txt

POOLTAB.txt

```
1 1
2 3
```

LITTAB.txt

```
1 ='4'    4
2 ='6'    10
3 ='1'    5
```

Scanned with CamScanner