

Name: Digvijay Jondhale

Roll No: PC-82

PAN: 1032201770

Batch: C1

Page No.	1
Date	

SSC Lab Assignment No. 2

Title : Design of Pass 2 of Two Pass Assembler.

Aim : Design suitable data structures and implement Pass 2 of 2 Pass Assembler for pseudo machine.

Objective: Design suitable data structures and implement Pass 2 of 2 Pass Assembler for pseudo machine. Subset should consist of a few instructions from each category and few assembler directives.

Theory: Design Specification of an Assembler:

In addition to the analysis phase (Pass 1), as mentioned earlier, the design specification of an assembler also includes the synthesis phase, which covers the generation of machine code or object code from the processed source code. The synthesis phase typically includes:

Code Generation: In this phase, the assembler generates the actual machine code instructions using the information collected during Pass 1, such as the symbol table.

Object File Generation: It creates an object file or an executable file containing the machine code, instructions and possibly additional information needed for loading and execution.

Memory Allocation: Determines the memory addresses for the generated machine code instructions, especially for instructions that use labels and symbols.

Relocation: Handles relocatable code, adjusting memory addresses as needed for loading at different addresses.

Design of a Two-Pass Assembler - Algorithm for Pass 2:

Pass 2 in a two-pass assembler is responsible for generating machine code and performing additional tasks. The algorithm for Pass 2 typically involves:

- Initialize the location counter (LC) to the starting address, as specified in Pass 1.

- Read the intermediate representation (usually created during Pass 1), which includes the source code, labels, operation codes, and operands.

- Process each line of the intermediate representation.

- Translate each assembly instruction into its corresponding machine code representation using the information from the OP TAB and the symbol table.

- Handle addressing modes, immediate values, labels, and symbols, and generate the appropriate binary code for each instruction.

- Write the generated machine code to the object file or executable file.

- Increment the LC as instructions are processed.

- Continue until all instructions have been processed.

- Perform any necessary back-patching or relocation adjustments as specified during Pass 1.

- Error Listing and Error Handling:

Error listing and error handling are critical aspects of any assembler. Error handling in Pass 2 involves checking for errors in the assembly code, such as invalid instructions, undefined symbols, or addressing errors and providing meaningful error messages to aid in debugging. Here's how error listing and handling works:

Error Detection: During Pass 2, the assembler checks each instruction for syntax and semantic errors.

Error Detection: During Pass 2, the assembler checks each instruction for syntax and semantic errors.

Error Reporting: When an error is detected, the assembler generates an error message indicating the line number, type of error, and a description of the problem,

continued Processing: The assembler may continue processing the code to identify and report multiple errors in a single pass.

Error Listing: An error listing is a report generated by the assembler that lists all detected errors, their line numbers and descriptions.

Graceful Termination: In case of severe or fatal errors, the assembler should terminate gracefully to avoid generating incomplete or incorrect machine code.

CODE:

```
import java.util.*;
import java.io.*;
import java.text.DecimalFormat;

class As2
{
    public static void main(String[] args) throws IOException
    {
        ArrayList<String[]> sym_tab = new ArrayList<String[]>();
        ArrayList<String[]> lit_tab = new ArrayList<String[]>();

        File input = new File("intermediate.asm");
        input.createNewFile();
        File output = new File("output.asm");
        output.createNewFile();
        File tables = new File("tables.asm");
        tables.createNewFile();
        String[] tokens;

        //Reading tables from Pass 1
        Scanner fileReader = new Scanner(tables);
        String buffer="";
        int tableFlag = 0;
        int counter = 0;
        String[] a = new String[4];
        while(fileReader.hasNextLine())
        {
            String i_str = fileReader.nextLine();
            tokens = i_str.split("[ \\n]");
            counter = 0;
            for(String str : tokens)
            {
                if(str.equals("[SYMBOL_TABLE]"))
                {
                    tableFlag = 1;
                    break;
                }
                else if(str.equals("[LITERAL_TABLE]"))
                {
                    tableFlag = 2;
                    break;
                }
                switch(tableFlag)
                {
                    {
                        case 1:
                            a[counter++] = str;
                            if(counter == 4)
                            {
                                sym_tab.add(new String[] {a[0],a[1],a[2],a[3]});
                                counter = 0;
                            }
                            break;
                        case 2:
                            a[counter++] = str;
                            if(counter == 3)
                            {
                                lit_tab.add(new String[] {a[0],a[1],a[2]});
                                counter = 0;
                            }
                    }
                }
            }
        }
    }
}
```

```

        break;
    }
}
}
fileReader.close();

System.out.println("SYMBOL TABLE: ");
for(String[] arr : sym_tab)
    System.out.println(Arrays.toString(arr));
System.out.println("\nLITERAL TABLE: ");
for(String[] arr : lit_tab)
    System.out.println(Arrays.toString(arr));

//Tokenizer
fileReader = new Scanner(input);
String i_str = "";
FileWriter fw = new FileWriter("output.asm");
BufferedWriter bw = new BufferedWriter(fw);
DecimalFormat formater = new DecimalFormat("000");
int num;
String num_formated;
boolean skip_flag=false;
while(fileReader.hasNextLine())
{
    i_str = fileReader.nextLine();
    tokens = i_str.split("[ \\n\\t]");
    //Assembler Pass II
    counter = 0;
    buffer = buffer.concat(tokens[0]);
    if(tokens[0]!="")
        buffer = buffer+"\t";
    for(String str : tokens)
    {
        str = str.trim();
        skip_flag=false;
        if(str=="")
            continue;
        if(str.substring(1,3).equals("AD")|| str.equals("(DL,02)"))
        {
            skip_flag=true;
            buffer = "";
            break;
        }
        if(str.equals("(DL,01)")) //For LORG and it's operand 1
            buffer = buffer.concat("(00) ");
        else if(counter==2)
        {
            if(str.charAt(0)=='(' && str.charAt(2)=='') //If Register
                buffer = buffer.concat("(0"+str.charAt(1)+") ");
            else
                buffer = buffer.concat("(00) ");
        }
        if(str.charAt(1)=='S') //For Symbol
        {
            num =
Integer.parseInt(str.substring(str.indexOf(',')+1,str.indexOf(')')));
            buffer = buffer.concat("(" +sym_tab.get(num-1)[2]+") ");
        }
        else if(str.charAt(1)=='L') //For Literal
        {

```

Mnemonic

```

        num =
Integer.parseInt(str.substring(str.indexOf(',')+1,str.indexOf(''))));
        buffer = buffer.concat("("+lit_tab.get(num-1)[2]+" ")");
    }
    else if(str.charAt(1)=='C') //For Constant
    {
        num =
Integer.parseInt(str.substring(str.indexOf(',')+1,str.indexOf(''))));
        num_formated = formater.format(num);
        buffer = buffer.concat("("+num_formated+" ")");
    }
    else if(str.substring(1,3).equals("IS")) //For Imperative Statements
        buffer = buffer.concat("("+str.substring(4,6)+" ")");
    counter++;
}
if(skip_flag==false)
{
    bw.write(buffer + "\n");
    buffer = "";
}
}
bw.close();
fw.close();
fileReader.close();

fileReader = new Scanner(output);
System.out.println("\n\nAssembly Code:");
while(fileReader.hasNextLine())
{
    System.out.println(fileReader.nextLine());
}
fileReader.close();
}
}

```

INPUT:

Intermediate.asm

```

        (AD,01) (C,100)
100)      (IS,04) (1) (S,1)
101)      (IS,01) (2) (S,1)
102)      (IS,04) (2) (S,3)
        (AD,03) (S,2)
101)      (IS,04) (2) (S,1)
102)      (DL,02) (C,5)
107)      (DL,01) (C,5)
        (AD,02)

```

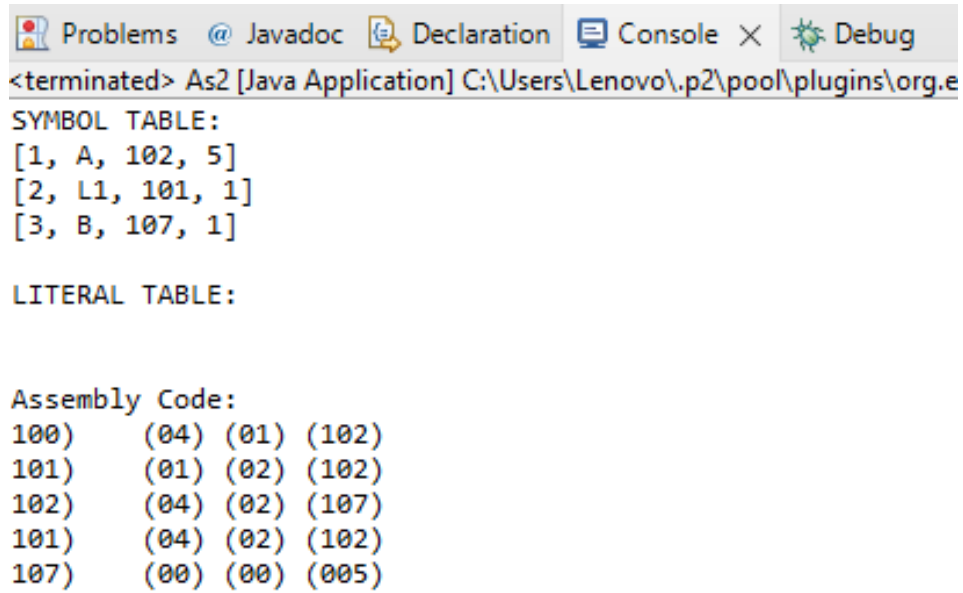
tables.asm

```

[SYMBOL_TABLE]
1 A 102 5
2 L1 101 1
3 B 107 1
[LITERAL_TABLE]

```

OUTPUT:



```
<terminated> As2 [Java Application] C:\Users\Lenovo\.p2\pool\plugins\org.e
SYMBOL TABLE:
[1, A, 102, 5]
[2, L1, 101, 1]
[3, B, 107, 1]

LITERAL TABLE:

Assembly Code:
100)      (04) (01) (102)
101)      (01) (02) (102)
102)      (04) (02) (107)
101)      (04) (02) (102)
107)      (00) (00) (005)
```