

Experiment No. 04

Aim- Design an implementation of pass I of two pass assembler

Requirement- Java

Theory - The one-pass assembler cannot resolve forward references of data symbols. It requires all data symbols to be defined prior to being used. A two-pass assembler solves this dilemma by devoting one pass to exclusively resolve all (data/label) forward references and then generate object code with no hassles in the next pass. If a data symbol depends on another and this another depends on yet another, the assembler resolved this recursively.

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 in two passes:

Pass-1:

- Define symbols and literals and remember them in symbol table and literal table respectively.
- Keep track of location counter
- Process pseudo-operations
- Assign addresses to all statements in the program.
- Save the values (addresses) assigned to all labels (including label and variable names) for use in Pass 2 (deal with forward references).
- Perform some processing of assembler directives (e.g., BYTE, RESW, these can affect address assignment)

The Pseudo Code for Pass 1

```

Pass 1:
begin
  read first input line
  if OPCODE = 'START' then
    begin
      save #[OPERAND] as starting address
      initialize LOCCTR to starting address
      write line to intermediate file
      read next input line
    end {if START}
  else
    initialize LOCCTR to 0
  while OPCODE ≠ 'END' do
    begin
      if this is not a comment line then
        begin
          if there is a symbol in the LABEL field then
            begin

```

Code-

```

import java.util.*;
import java.lang.*;
import java.io.*;

class pass1
{
  public static void main(String []args)
  {
    BufferedReader reader;
    int lc=0,sti=0,di=0,i,j,li=0;
    int[][] symtab = new int[100][3];
    int[][] littab = new int[100][3];
    String[] sym = new String[100];
    String[] data = new String[100];
    try
    {
      reader = new BufferedReader(new FileReader("prg.txt"));
      String line = reader.readLine();
      String[] words = line.split("\\s+");
      //System.out.println(sym[0]+" "+symtab[0][0]);
      while (!words[1].equals("END"))
      {
        if(!words[0].equals(""))
        {
          if(words[1].equals("START"))
          {
            sym[sti] = words[0];

```

```

        symtab[sti][0] = 0;
        symtab[sti][1] = 1;
        symtab[sti][2] = 0;
        sti++;
        //System.out.println("in1");
    }
    else if(words[1].equals("EQU"))
    {
        sym[sti] = words[0];
        if(words[2].equals("*")) symtab[sti][0] = lc;
        else symtab[sti][0] = Integer.parseInt(words[2]);
        symtab[sti][1] = 1;
        symtab[sti][2] = 1;
        sti++;
        //System.out.println("in2");
    }
    else if(words[0].equals("SAVE"))
    {
        sym[sti] = words[0];
        symtab[sti][0] = lc;
        symtab[sti][1] = 4;
        symtab[sti][2] = 0;
        sti++;
        String[] lcw = words[2].split("F");
        int ds = Integer.parseInt(lcw[0]);
        lc+= (ds*4);
        //System.out.println("in3");
    }
    else
    {
        sym[sti] = words[0];
        symtab[sti][0] = lc;
        symtab[sti][1] = 4;
        symtab[sti][2] = 0;
        sti++;
        if(words[0].equals("LOOP")) lc+=4;
        //System.out.println("in4");
    }
}
else if(words[1].equals("USING"))
{
    line = reader.readLine();
    words = line.split("\\s+");
    //System.out.println("in11");
    //System.out.println(words[0]);
    continue;
}
else if(words[1].equals("LTORG"))
{
    //System.out.println(lc+" "+words[1]);

```

```

        while(lc%8!=0) lc++;
        for(i=0;i<di;i++)
        {
            littab[li][0] = lc;
            littab[li][1] = 4;
            littab[li][2] = 0;
            lc+=4;
            li++;
        }
        //System.out.println("in12");
    }
    else
    {
        String[] opr = words[2].split(",");
        //System.out.println(lc+" "+words[1]);
        //System.out.println(opr[0]+" "+opr[1]);
        for(i=0;i<opr.length;i++)
        {
            if(opr[i].charAt(0) == '=')
            {
                data[di] = opr[i];
                di++;
            }
        }
        if(words[1].charAt(words[1].length()-1) == 'R') lc+=2;
        else lc+=4;
        //System.out.println("in13");
    }
    //System.out.println(words[1]);
    line = reader.readLine();
    words = line.split("\\s+");
    //System.out.println(sym[sti-1]+" "+symtab[sti-1][0]+" "+symtab[sti-1][1]+"
"+symtab[sti-1][2]);
    //System.out.println(data[di-1]+" "+littab[di-1][0]+" "+littab[di-1][1]+" "+littab[di-
1][2]);
    //System.out.println(words[0]);
}
reader.close();
//System.out.println("-----");
//System.out.println("Symbol Table");
//System.out.println("Symbol   Value   Length Relocation(0-R;1-A)");
//System.out.println("-----");

try(OutputStream fw = new FileOutputStream("symboltable.txt"))
{
    for(i=0;i<sti;i++)
    {
        //BufferedWriter bw = new BufferedWriter(fw);
        String content = sym[i]+" "+symtab[i][0]+" "+symtab[i][1]+"
"+symtab[i][2]+System.getProperty("line.separator");

```

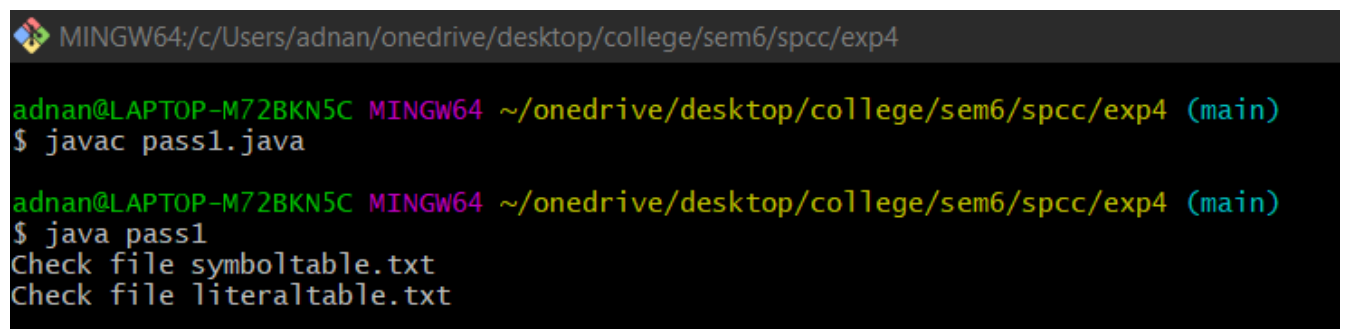
```

        //System.out.println(content);
        fw.write(content.getBytes(),0,content.length());
        //System.out.println(sym[i]+" "+symtab[i][0]+" "+symtab[i][1]+"
"+symtab[i][2]);
    }
}
catch (IOException e) { e.printStackTrace(); }
System.out.println("Check file symboltable.txt");
//System.out.println("-----");
//System.out.println("Literal Table");
//System.out.println("Literal Value Length Relocation(0-R;1-A)");
//System.out.println("-----");
try(OutputStream fw = new FileOutputStream("literaltable.txt"))
{
    for(i=0;i<di;i++)
    {
        //BufferedWriter bw = new BufferedWriter(fw);
        String content = data[i]+" "+littab[i][0]+" "+littab[i][1]+"
"+littab[i][2]+System.getProperty("line.separator");
        //System.out.println(content);
        fw.write(content.getBytes(),0,content.length());
        //System.out.println(sym[i]+" "+symtab[i][0]+" "+symtab[i][1]+"
"+symtab[i][2]);
    }
}
catch (IOException e) { e.printStackTrace(); }
System.out.println("Check file literaltable.txt");
}
catch (IOException e) { e.printStackTrace(); }
}
}

```

Output-

Execution:



```

MINGW64: c:/Users/adnan/onedrive/desktop/college/sem6/spcc/exp4
adnan@LAPTOP-M72BKN5C MINGW64 ~/onedrive/desktop/college/sem6/spcc/exp4 (main)
$ javac pass1.java
adnan@LAPTOP-M72BKN5C MINGW64 ~/onedrive/desktop/college/sem6/spcc/exp4 (main)
$ java pass1
Check file symboltable.txt
Check file literaltable.txt

```

Input File:

```

prg - Notepad
File Edit View

PRGAM2 START 0
        USING *,15
        LA 15, SETUP
        SR TOTAL, TOTAL
AC EQU 2
INDEX EQU 3
TOTAL EQU 4
DATABASE EQU 13
SETUP EQU *
        USING SETUP,15
        L DATABASE, =A(DATA1)
        USING DATAAREA, DATABASE
        SR INDEX, INDEX
LOOP L AC, DATA1(INDEX)
    AR TOTAL, AC
    A AC, =F'6'
    ST AC, SAVE(INDEX)
    A INDEX, =F'4'
    C INDEX, =F'8000'
    BNE LOOP
    LR 1, TOTAL
    BR 14
    LTORG
SAVE DS 2000F
DATAAREA EQU *
DATA1 DC F'01'
END

```

Symbol Table and Literal Table:

```

symboltable - Notepad...
File Edit View

PRGAM2 0 1 0
AC 2 1 1
INDEX 3 1 1
TOTAL 4 1 1
DATABASE 13 1 1
SETUP 6 1 1
LOOP 12 4 0
SAVE 64 4 0
DATAAREA 8064 1 1
DATA1 8064 4 0

Ln 1, Col 1 | 100% | Wi

```

```

literals - Notepad
File Edit View

=A(DATA1) 48 4 0
=F'6' 52 4 0
=F'4' 56 4 0
=F'8000' 60 4 0

Ln 1, Col 1 | 100% | Wind

```

Conclusion- We have successfully implemented Pass 1 of two pass Assembler