Name:- Rushikesh Kazbhazi Palve
Roll NO. 31258

Date: 1/20 | Page No.:

Assignment NO. 6

DOP:- 23-09-2021 DOS:- 27-09-2021

Problem Definition :-

Implement pass-II of a two-pass macro processor in Java. The output of pass-I [Assignment 24] (MNT, MDT and file without any macro deffnition) should be input for this assignment.

Learning Objectives :-

(i) To understand Data structure pass-2 macro processor.
(ii) To understand pass-1 and pass-2 macroprocessor concept.
(iii) To understand Advanced macro facility.

learning Outcomes :-

After completion of this assignment students will be able to-
(i) Implement pass-2 macroprocessor.
(ii) Implement machine code from MDT and MNT table.
(iii) Understand concept pass-2 macroprocessor.

Theory concepts :-

Advanced Macro Facilities :-

1) AIF
2) AGO
3) Sequential symbol
4) Expansion time variable.

1] AIF :-

Use the AIF instruction to branch according to the results of a condition test. You can thus alter the sequence in which source program statements are processed by the assembler.
The AIF instruction also provides loop control for conditional assembly processing, which lets you control the sequence of statements to be generated.
It also lets you check for error conditions and thereby to branch to the appropriate MNOTE instruction to issue an error message.

2] AGO :-

The AGO instruction branches unconditionally. You can thus alter the sequence in which your assembly language statements are processed. This provides you with final exits from conditional assembly loops.

3] Sequence Symbols :-

You can use a sequence symbol in the name field of a statement ~~during~~ to branch to that statement during conditional assembly

processing, thus altering the sequence in which the assembler processes your conditional assembly and macro instructions. You can select the model statements from which the assembler generates assembler language statements for processing at assembly time.

A sequence symbol consists of a period (.) followed by an alphabetic character, followed by 0 to 61 alphanumeric characters.

Examples :-

```
.BRANCHING_LABEL#1
.A
```

Sequence symbols can be specified in the name field of assembler language statements and model statements; however, sequence symbols must not be used as name entries in the following assembler instructions :-

ALIAS	EQU	OPSYN	SETC
AREAD	ICTL	SETA	SETAF
CATTR	LOCTR	SETB	SETCF
DXD			

Also, sequence symbols cannot be used as name entries in macro prototype instructions, or in any instruction that already contains an ordinary or a variable symbol in the name field.

Sequence symbols can be specified in the operand field of an AIF or AGO instruction to branch to a statement with the same sequence symbol as a label.

4] Expansion Time Variables :-

① Expansion Time variables are variables which can only be used during the expansion of macro calls.
② Local EV is created for use only during a particular macro call.
③ Global EV exists across all macro calls situated in program and can be used in any macro which has a declaration for it.

LCL <EV specification>[,<EV specification>...]
GBL <EV specification>[,<EV specification>...]

<EV specification> has syntax &<EV name>, where EV name is ordinary string.

Initialize EV by preprocessor statement SET.
<EV specification> SET <SET-expression>

Data structures of Two pass Macros :-

1] Input source program for pass-II. It is produced by pass-I.

2] Macro Definition Table :
(MDT) produced by pass-I

- Location counter (where MDTP points to start position of macro)
- Opcode
- Rest (i.e. it will contain the other part than opcodes used in macro).

3] Macro Name Table :-
(MNT) produced by pass-I

- macro number (i.e. pointer referenced from MNTP)
- Name of macros
- MDTP (i.e. points to start position to MDT)

4] MNTP : (Macro name table pointer) gives number of enteies in MNT.

5] Argument List Array :-

- Index given to parameter
- Name of parameter-

which gives association between integer indices & actual parameters.

6] Source program with macro-calls expanded.
This is output of pass-II.

7] MDTP (macro definition table pointer)
gives the address of macro definition in macro definition table.

Algorithm :-

Take Input from Pass-I.
Examine all statements in the assembly source program to detect macro calls. For each Macro call :

Macro call:

a) Locate the macro name in MNT.
b) Establish correspondence between formal parameters & actual parameters.
c) Obtain information from MNT regarding position of the macro definition in MDT.
d) Expand the macro call by picking up model statements from MDT.

Example for Pass-I and Pass-II of macroprocessor :-

Assembly Code segment :-

```
MACRO
INCR &X, &Y, &REG=AREG
MOVER &REG, &X
ADD &REG, &Y
MOVEM &REG, &X
MEND
MACRO
DECR &A, &B, &REG=BREG
MOVER &REG, &A
SUB &REG, &B
MOVEM &REG, &A
MEND
START
READ N1
READ N2
INCR N1, N2, REG=CREG
DECR N1, N2
STOP
```

```
N1  DS  1
N2  DS  2
END
```

Output of pass-I of Macroprocessor :-

#MNT

<#name>	<#PP>	<#KP>	<#MDTP>	<#KPDTP>
INCR	2	1	1	101
DECR	2	1	5	102

#PNTAB (INCR)

1.	X
2.	Y
3.	REG

#PNTAB (DECR)

1.	A
2.	B
3.	REG

#KPDTAB

	<name>	<value>
101.	REG	AREG
102.	REG	BREG

#MDT

1.	MOVER (P,3) (P,1)
2.	ADD (P,3) (P,2)
3.	MOVEM (P,3) (P,1)
4.	MEND
5.	MOVER (P,3) (P,1)
6	SUB (P,3) (P,2)
7.	MOVEM (P,3) (P,1)
8.	MEND

Output of pass-II of Macroprocessor :-

1) MACRO call :

(INCR FX, FY, FREG=AREG)

INCR N1, N2, REG = CREG

APTAB (INCR)

N1
N2
AREG

Expansion Code :-

```
MOVER AREG, N1
ADD   AREG, N2
MOVEM AREG, N1
```

2) Macro Call :-

```
DECR N1, N2
```

APTAB (DECR)

N1
N2
BREG

Expansion Code :-

```
MOVER BREG, N1
SUB   BREG, N2
MOVEM BREG, N1
```

Expanded Source file at the end of pass-II :-

```
START 200
READ N1
READ N2
MOVER AREG, N1
ADD   AREG, N2
MOVEM AREG, N1
```

```
MOVER  BREG, N1
 SUB   BREG, N2
MOVEM  BREG, N1
STOP
N1 DS 1
N2 DS 2
END
```

CONCLUSION :-

Thus, I have implemented pass-2 macro processor by taking input as output of Assignment 2A (i.e. MDT and MNT Table). Also, understood advanced macro facility.

CODE :-

```java
/*
 * Problem Statement :-
 *  Implement pass-II of a two-pass macro-processor in Java. The output of pass-I
 *  [Assignment 2A] (MNT, MDT and file without any macro definitions) should be input
 *  for this assignment.
 */


package assignmentNo_6;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.HashMap;
import java.util.Vector;

class Entry
{
    String name;
    int pp, kp, mdtp, kpdtp;

    public Entry(String name, int pp, int kp, int mdtp, int kpdtp)
    {
        super();
        this.name = name;
        this.pp = pp;
        this.kp = kp;
        this.mdtp = mdtp;
        this.kpdtp = kpdtp;
    }
    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
    public int getPp()
    {
        return pp;
    }
    public void setPp(int pp)
    {
        this.pp = pp;
    }
    public int getKp()
    {
        return kp;
    }
    public void setKp(int kp)
    {
        this.kp = kp;
    }
    public int getMdtp()
    {
```

```
        return mdtp;
    }
    public void setMdtp(int mdtp)
    {
        this.mdtp = mdtp;
    }
    public int getKpdtp()
    {
        return kpdtp;
    }
    public void setKpdtp(int kpdtp)
    {
        this.kpdtp = kpdtp;
    }
}

public class Assignment_No_6
{
    public static void main(String[] args) throws Exception
    {
        BufferedReader irb = new BufferedReader(new FileReader("intermediate.txt"));
        BufferedReader mdtb = new BufferedReader(new FileReader("mdt.txt"));
        BufferedReader kpdtb = new BufferedReader(new FileReader("kpdt.txt"));
        BufferedReader mntb = new BufferedReader(new FileReader("mnt.txt"));

        FileWriter fr = new FileWriter("pass2.txt");

        HashMap<String, Entry> mnt = new HashMap<>();
        HashMap<Integer, String> aptab = new HashMap<>();
        HashMap<String,Integer> aptabInverse = new HashMap<>();

        Vector<String>mdt = new Vector<String>();
        Vector<String>kpdt = new Vector<String>();

        int pp, kp, mdtp, kpdtp, paramNo;
        String line;
        while((line = mdtb.readLine()) != null)
        {
            mdt.addElement(line);
        }
        while((line = kpdtb.readLine()) !=  null)
        {
            kpdt.addElement(line);
        }
        while((line = mntb.readLine()) != null)
        {
            String parts[] = line.split("\\s+");
            mnt.put(parts[0], new Entry(parts[0], Integer.parseInt(parts[1]), Integer.
parseInt(parts[2]), Integer.parseInt(parts[3]), Integer.parseInt(parts[4])));

        }

        while((line = irb.readLine()) != null)
        {
            String []parts=line.split("\\s+");
            if(mnt.containsKey(parts[0]))
            {
                pp = mnt.get(parts[0]).getPp();
```

```
                kp = mnt.get(parts[0]).getKp();
                kpdtp = mnt.get(parts[0]).getKpdtp();
                mdtp = mnt.get(parts[0]).getMdtp();
                paramNo = 1;
                for(int i=0; i<pp; i++)
                {
                    parts[paramNo] = parts[paramNo].replace(",", "");
                    aptab.put(paramNo, parts[paramNo]);
                    aptabInverse.put(parts[paramNo], paramNo);
                    paramNo++;
                }
                int j = kpdtp-1;
                for(int i=0; i<kp; i++)
                {
                    String temp[] = kpdt.get(j).split("\t");
                    aptab.put(paramNo,temp[1]);
                    aptabInverse.put(temp[0],paramNo);
                    j++;
                    paramNo++;
                }

                for(int i=pp+1; i<parts.length; i++)
                {
                    parts[i] = parts[i].replace(",", "");
                    String splits[] = parts[i].split("=");
                    String name = splits[0].replaceAll("&", "");
                    aptab.put(aptabInverse.get(name),splits[1]);
                }
                int i = mdtp-1;
                while(!mdt.get(i).equalsIgnoreCase("MEND"))
                {
                    String splits[] = mdt.get(i).split("\\s+");
                    fr.write("+");
                    for(int k=0; k<splits.length; k++)
                    {
                        if(splits[k].contains("(P,"))
                        {
                            splits[k] = splits[k].replaceAll("[^0-9]", "");
                            String value = aptab.get(Integer.parseInt(splits[k]));
                            fr.write(value+"\t");
                        }
                        else
                        {
                            fr.write(splits[k]+"\t");
                        }
                    }
                    fr.write("\n");
                    i++;
                }

                aptab.clear();
                aptabInverse.clear();
            }
            else
            {
                fr.write(line+"\n");
            }
    }
```

```
    fr.close();
    mntb.close();
    mdtb.close();
    kpdtb.close();
    irb.close();
    System.out.println("\n\t Executed Successfully ...!!");
    }
}
```

INPUT :-

"intermediate.txt"

```
START 100
M1 10, 20, &B=CREG
M2 100, 200, &V=AREG, &U=BREG
END
```

"mdt.txt"

```
MOVER (P,3) (P,1)
ADD   (P,3) ='1'
MOVER (P,4) (P,2)
ADD   (P,4) ='5'
MEND
MOVER (P,3) (P,1)
MOVER (P,4) (P,2)
ADD   (P,3) ='15'
ADD   (P,4) ='10'
MEND
```

"kpdt.txt"

```
A  AREG
B  -
U  CREG
V  DREG
```

"mnt.txt"

```
M1 2  2  1  1
M2 2  2  6  3
```

OUTPUT :-

"IC_macroExpansion.txt"

```
START 100
+MOVER   AREG  10
+ADD  AREG  ='1'
+MOVER   CREG  20
+ADD  CREG  ='5'
+MOVER   BREG  100
+MOVER   AREG  200
+ADD  BREG  ='15'
+ADD  AREG  ='10'
END
```