**EXPERIMENT NO: 01**

**AIM:** To design and implement first pass of a two pass assembler for IBM 360/370 Processor.

**THEORY:**

An assembler performs the following functions

1. Generate Instructions

a. Evaluate the mnemonic in the operator field to produce its machine code.

b. Evaluate subfields- find value of each symbol, process literals & assign address.

2. Process pseudo-ops.

**PASS 1: DATABASE**

1. Source Program.

2. Location Counter (LC) which stores location of each instruction.

3. Machine Operation Table (MOT). This table indicates the symbolic mnemonic for each instructions and its length.

4. Pseudo Operation Table (POT). This table indicates the symbolic mnemonic and action taken for each pseudo-op in pass 1.

5. Symbol Table (ST) which stores each label along with its value.

6. Literal Table (LT) which stores each literal and its corresponding address.

7. A copy of input which will be used by pass 2.

**FORMAT OF DATABASES:**

The Machine Operation Table (MOT) and Pseudo Operation Table (POT) are examples of fixed tables. During the assembly process the contents of this table are not filled in or altered.

1. Machine Operation Table (MOT)

6-bytes per entry

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mnemonic  Op-codes  (4 bytes)  Characters | Binary  Op-codes  (1 byte)  hexadecimal | Instruction  Length  (2-bits)  Binary | Instruction  Format  (3-bits)  binary | Not used in this  Design  (3-bits) |
| “Abbb” | 5A | 10 | 001 |  |
| “AHbb” | 4A | 10 | 001 |  |
| “ALbb” | 5E | 10 | 001 |  |
| “ALRb” | 1E | 01 | 000 |  |
| …………… | ……………. | ……………. | …………… |  |

b: blank space

Codes: Instruction Length-01=1 half-words=2 bytes,10=2 half-words=4 bytes, 11=3 half-words=6 bytes

Instruction Format- 000=RR, 001=RX, 010=RS, 011=SI, 100=SS.

2. Pseudo Operation Table (POT)

8-bytes per entry

|  |  |
| --- | --- |
| Pseudo-op  (5-bytes)  Characters | Address of routine to process  Pseudo-op  (3-bytes=24 bits address) |
| “DROPb” | P1DROP |
| “ENDbb” | P1END |
| “EQUbb” | P1EQU |
| “START” | P1START |
| “USING” | P1USING |

Let us consider following source code and find the contents of symbol table and literal table.

|  |  |  |  |
| --- | --- | --- | --- |
| Stmt No. | Symbol | Op-code | Operands |
| 1 | SAMPLE | START | 0 |
| 2 |  | USING | \*,15 |
| 3 |  | A | 1,FOUR |
| 4 |  | A | 2,FIVE |
| 5 | TEMP | EQU | 10 |
| 6 |  | A | 3,=F’3’ |
| 7 |  | USING | TEMP,15 |
| 8 | FOUR | DC | F’4’ |
| 9 | FIVE | DC | F’5’ |
| 10 |  | END |  |

1. Symbol Table (ST)

14-bytes per entry

|  |  |  |  |
| --- | --- | --- | --- |
| Symbol  (8-bytes)  characters | Value  (4-bytes)  hexadecimal | Length  (1-byte)  hexadecimal | Relocation  (1-byte)  character |
| “SAMPLEbbb” | 0000 | 01 | “R” |
| “TEMPbbbb” | 0010 | 04 | “A” |
| “FOURbbbb” | 0012 | 04 | “R” |
| “FIVEbbbb” | 0016 | 04 | “R” |

1. Literal Table (LT)

7-bytes per entry

|  |  |  |  |
| --- | --- | --- | --- |
| Literal  (1-byte) | Value  (4-bytes)  hexadecimal | Length  (1-byte)  hexadecimal | Relocation  (1-byte)  character |
| F’3’ | 0020 | 04 | “R” |

5. Code after Pass 1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stmt No. | Relative  Address | Statement | |  |
| 1 |  | SAMPLE | START | 0 |
| 2 |  |  | USING | \*,15 |
| 3 | 0 |  | A | 1, \_ (0,15) |
| 4 | 4 |  | A | 2, \_ (0,15) |
| 5 |  | - | | |
| 6 | 8 |  | A | 3, \_ (0,15) |
| 7 |  | - | | |
| 8 | 12 | FOUR | 4 |  |
| 9 | 16 | FIVE | 5 |  |
| 10 |  | - | | |

**ALGORITHM:**

1. Initially location counter is set to relative address 0 i.e. LC=0.

2. Read the statement from source program.

3. Examine the op-code field: If match found in MOT then

a. From the MOT entry determine the length field i.e. L=Length.

b. Examine the operand field to check whether literal is present or not. If any new literal is found then corresponding entry is done in LT.

c. Examine the label field for the presence of symbol. If label is present then it is entered in ST and current value of location counter is assigned to symbol.

d. The current value of location counter is incremented by length of instruction (L).

4. If match found in POT then

a. If it is USING or DROP pseudo-op then first pass do nothing. It just writes a copy of these cards for pass 2.

b. If it is EQU pseudo-op then evaluate expression in operand field and assign value to the symbol present in label field.

c. If it is DS or DC pseudo-op then by examining the operand field find out number of bytes of storage required. Adjust the location counter for proper alignment.

d. If it is END pseudo-op then pass 1 is terminated and control is passed to pass 2. Before transferring the control it assigns location to literals.

5. A copy of source card is saved for pass 2.

6. Go to step 2.

**CONCLUSION:** Thus, we have studied and implemented the functionality of first pass of a two pass assembler. The symbol table, literal table and intermediate code are generated for the given program.

**Program:**

import java.util.\*;

import java.io.\*;

class Pass1

{

static int lc=0,index=0;

static String litrl[][]=new String[10][4]; //assumning 10 literals

static int basetable[][]=new int[10][2]; //assuming 10 entries

public static void main(String args[])

{

pass1();

}

static void pass1()

{

try

{

int val=0,potflag=0,i;

String inpt,strng=null,mot;

String lit[]=new String[1];

BufferedReader inp = new BufferedReader(new FileReader("input.txt"));

File syt = new File("symbol\_table.txt");

if (!syt.exists())

syt.createNewFile();

BufferedWriter sy = new BufferedWriter(new FileWriter(syt.getAbsoluteFile()));

File ltt = new File("literal\_table.txt");

if (!ltt.exists())

ltt.createNewFile();

BufferedWriter lt = new BufferedWriter(new FileWriter(ltt.getAbsoluteFile()));

File p1op = new File("pass1output.txt");

if (!p1op.exists())

p1op.createNewFile();

BufferedWriter op = new BufferedWriter(new FileWriter(p1op.getAbsoluteFile()));

for(;(inpt=inp.readLine())!=null;val=potflag=0)

{

StringTokenizer st=new StringTokenizer(inpt);

String str[]=new String[st.countTokens()];

for(i=0;i<str.length;i++)

str[i]=st.nextToken();

if(str.length==3)

val=1;

if(str.length!=1)

{

StringTokenizer stkn=new StringTokenizer(str[val+1],","); //delimiter is comma

lit=new String[stkn.countTokens()];

for(i=0;i<lit.length;i++)

lit[i]=stkn.nextToken();

}

if(str[val].equalsIgnoreCase("DS") || str[val].equalsIgnoreCase("DC")) //checking whether it is in pot

{

int l=0;

if (val==1)

strng=str[0]+"\t"+lc+"\t4\tR";

if(lit[0].indexOf("F")!=0)

{

l=Integer.parseInt(lit[0].substring(0,lit[0].length()-1));

l\*=4;

}

else

for(i=0;i<lit.length;i++)

l+=4;

lc+=l;

}

else

{

if(str[val].equalsIgnoreCase("EQU"))

{

if(str[2].equals("\*"))

strng=str[0]+"\t"+lc+"\t1\tR";

else

strng=str[0]+"\t"+str[2]+"\t1\tA";

}

else

{

if(str[val].equalsIgnoreCase("START"))

strng=str[0]+"\t"+str[2]+"\t1\tR";

else

{

if(str[val].equalsIgnoreCase("LTORG"))

ltorg(true);

else

{

if(str[val].equalsIgnoreCase("END"))

ltorg(false);

else potflag=1;

}

}

}

}

if(potflag==1) //mot search

{

if(str.length!=1)

{

for(i=0;i<lit.length;i++)

if(lit[i].charAt(0)=='=')

{

litrl[index][0]=lit[i].substring(1,lit[i].length());

litrl[index][1]="-1";

litrl[index][2]="4";

litrl[index++][3]="R";

}

}

BufferedReader mt = new BufferedReader(new FileReader("mot.txt"));

while((mot=mt.readLine())!=null)

{

StringTokenizer stk=new StringTokenizer(mot);

String s[]=new String[stk.countTokens()];

for(i=0;i<s.length;i++)

s[i]=stk.nextToken();

if(str[val].equalsIgnoreCase(s[0]))

{

if(val==1)

strng=str[0]+"\t"+lc+"\t"+s[2]+"\tR"; //formation of symbol table

lc+=Integer.parseInt(s[2]);

break;

}

}

mt.close();

op.write(inpt); //input to pass 2

op.newLine();

}

if(val==1)

{

sy.write(strng);

sy.newLine();

}

}

for(i=0;i<index;i++)

{

lt.write(litrl[i][0]+"\t"+litrl[i][1]+"\t"+litrl[i][2]+"\t"+litrl[i][3]);

lt.newLine();

}

inp.close();

sy.close();

lt.close();

op.close();

}

catch(FileNotFoundException ex)

{

System.out.println("Unable to find file ");

}

catch(IOException e)

{

e.printStackTrace();

}

}

static void ltorg(boolean flag)

{

int i,l=0;

if(flag)

{

l=lc+8;

lc=l-(l%8);

}

for(i=0;i<index;i++)

if(litrl[i][1].equals("-1"))

{

litrl[i][1]=""+lc;

lc+=4;

}

}

static String offset(String s)

{

int value,indx,i,ind=0,offst,new\_offst,indx\_reg=0;

String string=s;

if(s.charAt(0)=='=')

value=getValue(s.substring(1,s.length()),1); //0=symbol table & 1=literal table value

else

{

indx=s.indexOf("(");

if(indx!=-1)

{

s=s.substring(0,indx);

indx\_reg=getValue(string.substring(string.indexOf("(")+1,string.indexOf(")")),0);

}

value=getValue(s,0);

}

offst=Math.abs(value - basetable[ind][1]);

for(i=1 ; i<index ; i++)

{

new\_offst = Math.abs(value - basetable[i][1]);

if(new\_offst < offst)

{

offst = new\_offst;

ind = i;

}

}

String result = ","+offst + "(" + indx\_reg + ", " + basetable[ind][0] + ")";

return result;

}

static int getValue(String s,int flag)

{

try

{

String sym,file\_name;

if(flag==0)

file\_name="symbol\_table.txt";

else

file\_name="literal\_table.txt";

BufferedReader br = new BufferedReader(new FileReader(file\_name));

while((sym=br.readLine())!=null)

{

StringTokenizer st=new StringTokenizer(sym);

String str[]=new String[st.countTokens()];

for(int i=0;i<str.length;i++)

str[i]=st.nextToken();

if(str[0].equalsIgnoreCase(s))

return Integer.parseInt(str[1]);

}

}

catch(FileNotFoundException ex)

{

System.out.println("Unable to find file ");

}

catch(IOException e)

{

e.printStackTrace();

}

return -1;

}

}

**Input:**

**1. Input.txt**

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'5'

ST AC,SAVE(INDEX)

A INDEX,=F'4'

C INDEX,=F'8000'

BNE LOOP

LR 1,TOTAL

BR 14

LTORG

SAVE DS 3F

DATAAREA EQU \*

DATA1 DC F'25,26,27'

END

**2. Mot.txt**

LA 01h 4 RX

SR 02h 2 RR

L 03h 4 RX

AR 04h 2 RR

A 05h 4 RX

C 06h 4 RX

BNE 07h 4 RX

LR 08h 2 RR

ST 09h 4 RX

BR 15h 2 RR

**OUTPUT:**

**1. literal\_table.txt**

Literal Value Length Relocation

(1-byte) (4-bytes) (1-byte) (1-byte)

A(DATA1) 48 4 R

F'5' 52 4 R

F'4' 56 4 R

F'8000' 60 4 R

**2. pass1output.txt**

USING \*,15

LA 15,SETUP

SR TOTAL,TOTAL

USING SETUP,15

L DATABASE,=A(DATA1)

USING DATAAREA,DATABASE

SR INDEX,INDEX

LOOP L AC,DATA1(INDEX)

AR TOTAL,AC

A AC,=F'5'

ST AC,SAVE(INDEX)

A INDEX,=F'4'

C INDEX,=F'8000'

BNE LOOP

LR 1,TOTAL

BR 14

**3. symbol\_table.txt**

Symbol value Length Relocation

(8-bytes) (4-bytes) (1-byte) (1-byte)

PRGAM2 0 1 R

AC 2 1 A

INDEX 3 1 A

TOTAL 4 1 A

DATABASE 13 1 A

SETUP 6 1 R

LOOP 12 4 R

SAVE 64 4 R

DATAAREA 76 1 R

DATA1 76 4 R

**CONCLUSION:**

Thus, we have studied and implemented the functionality of first pass of a two pass assembler. The symbol table, literal table and intermediate code are generated for the given program.