System Programming and Operating Systems Lab

**ASSIGNMENT 6**

**Name:Maulick Metkar**

**Roll No: 323045**

**Batch: C2**

# Aim:

Design suitable data structures and implement pass-I of two-pass assembler for pseudo-machine in java using object-oriented features. Implementation should consist of few instructions from each category and few assembler directives.

# Objectives:

To implement pass-I of a two-pass assembler.

# Theory:

The pass-wise grouping of tasks in two pass assembler is given below:

Pass-I

* + Seperate the labels, mnemonic op-code and operand fields.

Determine the storage requirement for every assembly language statement and update the location counter.

•

Build the symbol table. Symbol table is used to store eeach label and each variable and its corresponding address.

•

Pass-II

* + Generate machine code.

Following are the data structures used in pass-I of a two-pass assembler:

* + Machine operation code table (MOT).
  + Symbol table (ST).
  + Literal table (LT).
  + Pool table (PT).

# Algorithm:

* + 1. loc-cntr := 0; (default value) pooltab-ptr := 1;

POOLTAB [1]:=1;

littab-ptr := 1;

* + 1. While next statement is not an END statement

1. If label is present then

this-label := symbol in label field;

Enter (this-label, loc-cntr) in SYMTAB.

1. If an LTORG statement then
2. Process literals LITTAB [POOLTAB [pooltab-ptr]]... LITTAB [lit- tab-ptr – 1] to allocate memory and put the address in the address field. Update loc-cntr accordingly.
3. pooltab-ptr := pooltab-ptr + 1;
4. POOLTAB [pooltab-ptr] := littab-ptr;
5. If a START or ORIGIN statement then loc-cntr := value specified in operand field;
6. If an EQU statement then
7. this-addr := value of <address spec>;
8. Correct the symtab entry for this-label to (this-label, this-addr).
9. If a declaration statement then
10. code := code of the declaration statement;
11. size := size of memory area required by DC/DS.
12. loc-cntr := loc-cntr + size;
13. Generate 1C ’(DL, code) ’.
14. If an imperative statement then
15. code := machine opcode from OPTAB;
16. loc-cntr := loc-cntr + instruction length from OPTAB;
17. If operand is a literal then

this-literal := literal in operand field; LITTAB [littab-ptr] := this-literal; littab-ptr := littab-ptr + 1;

else (i.e. operand is a symbol)

this-entry := SYMTAB entry number of operand; Generate 1C ’(IS, code)(S, this-entry)’;

* + 1. (Processing of END statement)

1. Perform step 2(b).
2. Generate 1C ’(AD.02)’.
3. Go to Pass II.

# Flowchart:

4

# Code:

**Pass1a.java :**

import java . i o . ; import java . u t i l . ; c l a s s AD {

∗

∗

S tr i n g s t a r t c l a s s ="IS "; i n t s ta r to p =1;

S tr i n g e n d c l a s s="IS "; i n t endop=2;

S tr i n g o r g c l a s s ="IS "; i n t orgop =3;

S tr i n g e q u c l a s s="IS "; i n t equop =4;

S tr i n g l t o r g c l a s s ="IS "; i n t l to r g o p =5;

}

c l a s s Dl {

S tr i n g d s c l a s s ="DL" ; i n t dspop =1;

i n t d s l e n =0;

S tr i n g d c c l a s s="DL" ; i n t dcop =2;

i n t dclen =1;

}

c l a s s IS {

S tr i n g s t p c l a s s ="IS "; i n t stopop =0;

i n t s to p l e n =1;

S tr i n g a d d c l a s s="IS "; i n t addop=1;

i n t addlen =1;

S tr i n g s u b c l a s s="IS "; i n t subop =2;

i n t sublen =1;

S tr i n g m u l tc l a s s="IS "; i n t multop =3;

i n t multlen =1;

S tr i n g moverclass="IS "; i n t moverop=4;

i n t moverlen =1;

S tr i n g movemclass="IS "; i n t movemop=5;

i n t movemlen=1;

S tr i n g compclass="IS "; i n t compop=6;

i n t complen =1;

S tr i n g b c c l a s s="IS "; i n t bcop =7;

i n t bclen =1;

S tr i n g d i v c l a s s ="IS "; i n t divop =8;

i n t d i v l e n =1;

S tr i n g r e a d c l a s s="IS "; i n t readop =9;

i n t r e a d l e n =1;

S tr i n g p r i n t c l a s s ="IS ";

i n t printop =10; i n t p r i n t l e n =1;

}

p u b l i c c l a s s Pass1a {

p u b l i c s t a t i c void main ( S t r i n g args [ ] ) throws IOException { IS i s=new IS ( ) ;

AD ad=new AD( ) ; Dl dl=new Dl ( ) ;

File Reader f r=new File Reader (" abc . txt " ) ; Buffered Reader br=new Buffered Reader ( f r ) ; S tr i n g s1 , s 7 = n u l l ;

Fi l e Wr i te r fw=new Fi l e Wr i te r (" PassOp . txt " , true ) ; while ( ( s 1=br . read Line ( ) ) ! = n u l l )

{

S tr i n g s 2 [ ]= s 1 . r e p l a c e A l l ("^ [ , \\ s ]+" , "") . s p l i t ( " [ , \ \ s ] +") ; S tr i n g s 4 [ ]= s 1 . s p l i t ( " , " ) ;

f o r ( S t r i n g w: s 4 ){

}

f o r ( S t r i n g w: s 2 ){

i f (w. e q ua l s ("START") | | w. e q u a l s ("END" ) ){ fw . w r i te ( " ( " ) ;

fw . w r i te ("AD"+" , " ) ;

i f (w. e q ua l s ("START") )

fw . w r i te ("0"+ ad . s ta r to p ) ; i f (w. e q ua l s ("END" ) )

fw . w r i te ("0"+ ad . endop ) ;

fw . w r i te ( " ) " ) ;

}

i f (w. e q ua l s ("MOVER") | | w. e q u a l s ("MOVEM" ) ){ fw . w r i te ("\ n " ) ;

fw . w r i te ( " ( " ) ;

fw . w r i te (" IS"+" , " ) ;

i f (w. e q ua l s ("MOVER" ) )

fw . w r i te ("0"+ i s . moverop ) ; i f (w. e q ua l s ("MOVEM" ) )

fw . w r i te ("0"+ i s . movemop ) ; fw . w r i te ( " ) " ) ;

}

i f (w. e q ua l s ("AREG") | | w. e q u a l s ("BREG" ) ){ s 7=w ;

fw . w r i te (" "+w+" " ) ;

}

i f ( (w. e q u a l s ("A" ) | | w. e q u a l s ("B") ) && ( s 7 . e q u a l s ("AREG" ) | | s 7 . e q u a fw . w r i te (" "+w+" " ) ;

s 7 ="";

}

i f ( (w. e q u a l s ( " ’ 5 ’ " ) | | w. e q u a l s ( " ’ 4 ’") ) && ( s 7 . e q ua l s ("AREG" ) | | s 7 . s 7 ="";

}

i f (w. e q ua l s ( " ’ 5 ’") | | w. e q u a l s ( " ’ 4 ’") ) { fw . w r i te ( " ( " ) ;

fw . w r i te ("L"+" , " ) ;

i f (w. e q ua l s ( " ’ 5 ’") ) { fw . w r i te ( " 0 1 ") ;

}

i f (w. e q ua l s ( " ’ 4 ’") ) { fw . w r i te ( " 0 2 ") ;

}

fw . w r i te ( " ) " ) ;

}

i f (w. e qu a l s ("DS")){

fw . w r i te ("\ n " ) ;

fw . w r i te ( " ( " ) ;

fw . w r i te ("DL"+" , " ) ; fw . w r i te ( " 0 1 ") ;

fw . w r i te ( ") " ) ;

fw . w r i te ( " ( " ) ;

fw . w r i te ("C, 0 1 ) " ) ;

}

}

fw . w r i te ("\ n " ) ;

}

f r . c l o s e ( ) ;

fw . c l o s e ( ) ;

}

}

Contents of the files used:

1. abc.txt (Assembly language code)

START

MOVER AREG,A MOVEM AREG, B A DS 1

B DS 1

END

# Output:

Generated symbol table 0 A 204

1 B 205

Generated literal table 0 ’5’ 208

1 ’4’ 209

# Conclusion:

In this assignment we understood in detail how to implement pass-I of a two-pass assembler and the data structures needed for it.