Assembler Design
&
Symbol Table Construction
Using
C++

COMPUTER SOFTWARE LAB

Project Report

Sambhav R Jain – 107108103

S Vignesh – 107108102

Thanujan – 107108069

A single-pass assembler is designed for a specific instruction set. Hash table technique is employed to formulate the symbol table, which lists the various symbols used in the assembler code and the statement addresses referring to the symbol. The problem is implemented on a C++ platform.

PROBLEM STATEMENT:

In a single pass assembler, in resolving forward references of a symbol, the following procedure is followed:

The symbol table and the list of addresses of different statements referring to that symbol are noted in the symbol table. When the definition of the symbol is encountered, its value/address is noted in the symbol table and using the list of addresses, the symbol is replaced with its value/address in all the statements referring that symbol. It is required to design such an assembler for the assembly language whose instruction set is given below.

1. 3-address instructions: LDA, STA, JMP, JZ
2. 2-address instructions: MVI
3. 1-address instructions: ADD, INR, SUB, DCR, HLT
4. Allowable registers: A, B, C, D, E, H and L (all are in a 8-bit format)
5. Standard field structure of SOURCE.ASS:

   LABEL MNEMONIC/OPERATION OPERAND ; COMMENTS
6. Allowable pseudo operations: ORG, END, EQU, DB

Write a program in C++ to create the symbol table which can be used in the above single pass assembler. The symbol table should be of the form shown in the example below.

Note:

i. Use division method of hash table technique to construct the symbol table
ii. The source program must be read from a *source.ass* file
iii. Duplicate entries are not allowed in the symbol table
iv. Report of the source program and the symbol table
v. Addresses are assumed to be in hexadecimal form

Example:

Source Program:

source.ass

```
                ORG     1000H
                LDA     data1           ; loading acc with data1
                JZ      LOCNa
                LDA     data2           ; loading acc with data2
                JZ      LOCNa
LOCNa           LDA     data1           ; loading acc with data1
data2           EQU     100H
data1           EQU     800H
                END
```

Object Program:

object.txt

Address	*Bytes Reqd.*	*References*
1000	3	data1
1003	3	LOCNa
1006	3	data2
1009	3	LOCNa
100C	3	data1

Generated Symbol Table:

symbol	value/address	statement addresses referring to the symbol
data1	800	1000, 100C
LOCNa	100C	1003, 1009
data2	100	1006

PROGRAM CODE:

```
/*
The "source.ass" should comply to the following specifications
1. Comments should follow after ';'
2. First line should be of the form "ORG 1000H" which would specify the starting
address
3. MVI command should be as "MVI A , 08" with at least one space between every word
4. All EQU commands should be placed at the end of the file
5. Keywords should be separated from symbol names by at least one space
*/

#include<iostream>
#include<fstream>       //For file handling
#include<cstring>       //To use strcmp(a,b), strlen(a)
#include<math.h>        //To use pow(a,b)
using namespace std;
char* p;       //To store the assembler-specific keywords like LDA, STA, JZ etc.
int adrs=0;    //Addresses of consecutive lines in the assembly file in hexadecimal
ifstream infile,intemp;          //Input stream objects
ofstream outfile;                //Output stream object
ostream& dispwidth(ostream& out) //User-defined manipulator (width and left-justify)
{
    out.width(15);
    out.setf(ios::left,ios::adjustfield);
    return out;
}
void conv()     //Removes comments and redundant spaces in the source file
{
    char* z=new char[50];
    while(!infile.eof())
    {
        infile.getline(z,50,'\n');
        for(int i=0;;i++)
        {
            if((z[i]==';')||(z[i]=='\0'))   //Comments start after ; (semi-colon)
            break;
            else
 {
if(((z[i]=='')||(z[i]=='\t'))&&((z[i+1]==' ')||
(z[i+1]=='\0')||(z[i+1]==';')||(z[i+1]=='\t')))
 continue;
 outfile<<z[i];
 }
        }
        outfile<<'\n';
    }
    delete z;
}
void value()    //To retrieve the value/address stored in a symbol
{
    char* z=new char[30];
    int i=0,s=0,n=0,m=0;    //s - no of spaces
    while(!infile.eof())
```

```
    {
        s=0;
        infile.getline(z,30,'\n');
        for(n=0;n<strlen(z);n++)    //To count no of spaces
        {
            if(z[n]==' ')
            s++;
        }
        if(s==2)    //Only statements with 2 spaces assign value/address to symbols
        {
            n=0;
            outfile<<'\n';
            while(z[n]!=' ')    //Output the symbol name
            {
                outfile<<z[n];
                n++;
            }
            outfile<<' ';
            if((z[n+1]=='E')&&(z[n+2]=='Q')&&(z[n+3]=='U'))
            {
                for(m=n+5;m<strlen(z);m++)
                outfile<<z[m];
            }
            else
            outfile<<"line"<<' '<<i;
        }
        i++;
    }
    delete z;
}
int tohex(char a)      //Returns an integer corresponding to each hexadecimal digit
{
    if(a>=48&&a<=57)        //If 'a' stores a digit (0-9)
    return (a-48);
    else if(a>=65&&a<=70)   //If 'a' stores an upper-case alphabet (A-F)
    return (a-55);
    else if(a>=97&&a<102)   //If 'a' stores a lower-case alphabet (a-f)
    return (a-87);
}
int hexadec(char ch[4])        //Converts hexadecimal to decimal
{
    int sum=0,rem=0;
    for(int i=0;i<=3;i++)
    {
        rem=tohex(ch[i]);
        sum+=rem*pow(16,i);
    }
    return sum;
}
int hash(char* a)   //This returns a unique hash value for each particular keyword
{
if((strcmp(a,"LDA")==0)||(strcmp(a,"STA")==0)||(strcmp(a,"JMP")==0)||(strcmp(a,"JZ")==
0))
    return 3;
    else if(strcmp(a,"MVI")==0)
```

```
    return 2;
    else
if((strcmp(a,"ADD")==0)||(strcmp(a,"INR")==0)||(strcmp(a,"SUB")==0)||(strcmp(a,"HLT")==0))
    return 1;
    else
    return 0;
}
int init()  //Initialization
{
    char hexadrs[4];    //To store the 4-bit starting address
    p=new char[3];
    infile>>p;
    if(strcmp(p,"ORG")!=0)
    {
        cout<<"origin of the code not defined! TERMINATING!!";
        return 0;
    }
    else
    infile>>hexadrs[3]>>hexadrs[2]>>hexadrs[1]>>hexadrs[0];
    adrs=hexadec(hexadrs);
    outfile.setf(ios::hex,ios::basefield);
    outfile<<adrs<<' ';
    infile>>p;  //Extract & discard the suffix 'H' (e.g. in ORG 1000H)
    delete p;
    return 1;
}
int main()
{
    cout<<"\nreading source file....'source.ass'\n";
    infile.open("source.ass");
    outfile.open("newsrc.txt");
    cout<<"\ncreating new source file....'newsrc.txt'\n";
    conv();         //To remove comments and redundant spaces from the source file
    infile.close();
    outfile.close();
    cout<<"\nformulating object file....'object.txt'\n";
    infile.open("newsrc.txt");
    outfile.open("object.txt");
    if(!init())
    return 0;
    while(!infile.eof())
    {
       p=new char[10];
       infile>>p;        //Extracts the assembler-specific keywords like LDA, STA, JZ
      int h=hash(p);     //Return a unique hash value for each particular keyword
       delete p;
       char* z;          //To retrieve and store symbols
       switch(h)
       {
           case 3:
                adrs+=3;    //Since 3-byte instructions
                z=new char[10];
                infile>>z;
                outfile<<z<<endl;
```

```
            outfile<<adrs<<' ';
            delete z;
            break;
        case 2:
            adrs+=2;    //Since 2-byte instructions
            z=new char[10];
            infile>>z;
            outfile<<z<<endl;
            outfile<<adrs<<' ';
            infile>>z>>z;        //To   discard   the   comma   and   value   moved
            delete z;
            break;
        case 1:
            adrs+=1;    //Since 1-byte instructions
            z=new char[10];
            infile>>z;
            outfile<<z<<endl;
            outfile<<adrs<<' ';
            delete z;
    }
}
outfile.unsetf(ios::hex);
infile.close();
outfile.close();
cout<<"\nprocuring values of symbols....'values.txt'\n";
infile.open("newsrc.txt");
outfile.open("values.txt");
value();        //To retrieve the value/address stored in a symbol
infile.close();
outfile.close();
infile.open("object.txt");
char* ad[20];   //To store line addresses
char* symb[20]; //To store symbols referenced at corresponding addresses
int i=0;
while(!infile.eof())
{
    ad[i]=new char;
    infile>>ad[i];
    symb[i]=new char;
    infile>>symb[i];
    i++;
}
infile.close();
infile.open("values.txt");
char* val[20];
char* z=new char[10];
while(!infile.eof())
{
    infile>>z;
    for(int r=0;r<i-1;r++)
    {
        if(!strcmp(z,symb[r]))
        {
            val[r]=new char;
            infile>>val[r];
```

```
                if(!strcmp(val[r],"line"))
                {
                    int lnum;       //Line number
                    infile>>lnum;
                    intemp.open("object.txt");
                    for(int m=1;m<lnum;m++)
                    intemp>>val[r]>>val[r];
                    intemp>>val[r];
                    intemp.close();
                }
                break;
            }
        }
    }
    delete z;
    infile.close();
    cout<<"\nstoring symbol table....'symbol.txt'\n";
    outfile.open("symbol.txt");
    outfile<<dispwidth<<"SYMBOL"<<dispwidth<<"VALUE/ADDRESS"<<dispwidth<<"\tREFERENCED
at"<<endl;
    int flag[100];
    for(int x=0;x<100;x++)
    flag[x]=0;
    for(int j=0;j<i-1;j++)
    {
        if(flag[j]!=1)
        {
            outfile<<'\n'<<dispwidth<<symb[j]<<dispwidth<<val[j]<<'\t'<<ad[j];
            flag[j]=1;
        }
        for(int k=j+1;k<i-1;k++)
        {
            if((strcmp(symb[j],symb[k])==0)&&(flag[k]!=1))
            {
                outfile<<','<<ad[k];
                flag[k]=1;
            }
        }
    }
    outfile.close();
    return 0;
}
```

OUTPUT:

source.ass

```
ORG   1000H       ; origin
LDA   data1       ; load
JZ    LOCNa       ; jump on zero
MVI   data3 , 05  ; move immediate
STA   data2
JZ    LOCNb       ; jump on zero
ADD   data2
INR   A
JMP   LOCNb       ; jump
HLT   C
LOCNa LDA   data3
LOCNb SUB   B
data1 EQU 12H
data2 EQU 24H
data3 EQU 56H
A     EQU 22
B     EQU 23
C     EQU 24
END
```

newsrc.txt

```
ORG 1000H
LDA data1
JZ LOCNa
MVI data3 , 05
STA data2
JZ LOCNb
ADD data2
INR A
JMP LOCNb
HLT C
LOCNa LDA data3
LOCNb SUB B
data1 EQU 12H
data2 EQU 24H
data3 EQU 56H
A EQU 22
B EQU 23
C EQU 24
END
```

object.txt

```
1000 data1
1003 LOCNa
1006 data3
1008 data2
100b LOCNb
100e data2
100f A
1010 LOCNb
1013 C
1014 data3
1017 B
```

values.txt

```
LOCNa line 10
LOCNb line 11
data1 12H
data2 24H
data3 56H
A 22
B 23
C 24
```

symbol.txt

SYMBOL	VALUE/ADDRESS	REFERENCED at
data1	12H	1000
LOCNa	1014	1003
data3	56H	1006,1014
data2	24H	1008,100e
LOCNb	1017	100b,1010
A	22	100f
C	24	1013
B	23	1017

RESULTS:

Hence the given assembly level statements are interpret into a symbol table with all the references to symbols used and their corresponding addresses. File handling feature of C++ has been extensively used to store data into files and allow their retrieval later. Hence a symbol table is successfully constructed.