**Develop pass-1 of two-pass assembler.**

//

// This code is written by ESHWAR M S

//

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

#include <sstream>

#include <map>

using namespace std;

map<string, int> opcodes;

map<string, int> symbols;

int main()

{

string label, opcode, operand, data;

int startAddress, endAddress, locctr;

ifstream input, optab;

ofstream output, symtab;

string nmu;

int code;

input.open("input.txt");

optab.open("optab.txt");

output.open("output.txt");

symtab.open("symtab.txt");

while(!optab.eof()){

optab>>nmu>>hex>>code;

opcodes.insert(pair<string, int>(nmu,code));

}

getline(input, data);

std::stringstream iss(data);

getline(iss, label ,'\t');

getline(iss, opcode ,'\t');

getline(iss, operand ,'\t');

if(!opcode.compare("START")){

stringstream convert(operand);

convert>>hex>>startAddress;

locctr = startAddress;

if(!label.empty()){

symbols.insert(pair<string, int>(label, locctr));

symtab<<label<<"\t"<<hex<<locctr<<endl;

}

output<<hex<<locctr<<"\t"<<label<<"\t"<<opcode<<"\t"<<operand<<endl;

}

while(opcode.compare("END")){

getline(input, data);

std::stringstream iss(data);

getline(iss, label ,'\t');

//comment

if(!label.compare(".")){

output<<'\t'<<data<<endl;

continue;

}

getline(iss, opcode ,'\t');

operand = "";

getline(iss, operand ,'\t');

// write the line to intermediate file

output<<hex<<locctr<<"\t"<<label<<"\t"<<opcode<<"\t"<<operand<<endl;

// Add label to symtab if it exists

if(!label.empty()){

map<string,int>::iterator it = symbols.find(label);

if(it!=symbols.end()){

cout<<"Symbol already exist"<<endl;

break;

}

else{

symbols.insert(pair<string, int>(label, locctr));

symtab<<label<<"\t"<<hex<<locctr<<endl;

}

}

// find if opcode exists in optab else suitable assembly directive

map<string,int>::iterator it = opcodes.find(opcode);

if(it!=opcodes.end()){

locctr += 3;

}

else if(!opcode.compare("WORD")){

locctr += 3;

}

else if(!opcode.compare("RESW")){

int temp;

stringstream convert(operand);

convert>>temp;

locctr += 3 \* temp;

}

else if(!opcode.compare("RESB")){

int temp;

stringstream convert(operand);

convert>>temp;

locctr += temp;

}

else if(!opcode.compare("BYTE")){

if (operand.at(0) == 'C')

{

int temp = operand.length()-3;

locctr += temp;

}

else if (operand.at(0) == 'X')

{

int temp = operand.length()-3;

locctr += temp/2;

}

}

else{

if(!opcode.compare("END")){

break;

}

cout<<"Invalid opcode"<<endl;

exit(1);

}

}

input.close();

optab.close();

symtab.close();

output.close();

endAddress = locctr;

cout<<"Length is: "<<endAddress - startAddress<<endl;

}

input.txt

COPY START 2000

LDA FIVE

STA ALPHA

LDCH CHARZ

STCH C1

ALPHA RESW 1

FIVE WORD 5

CHARZ BYTE C'Z'

C1 RESB 1

END

output.txt

2000 COPY START 2000

2000 LDA FIVE

2003 STA ALPHA

2006 LDCH CHARZ

2009 STCH C1

200c ALPHA RESW 1

200f FIVE WORD 5

2012 CHARZ BYTE C'Z'

2013 C1 RESB 1

2014 END

**Develop pass-2 of two-pass assembler.**

//

// This code is written by ESHWAR M S

//

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

#include <sstream>

#include <map>

#include <iomanip>

using namespace std;

map<string, int> opcodes;

map<string, int> symbols;

class head{

public:

string name;

int startAddress, length;

head(string n,int s,int l){

name = n;

startAddress = s;

length = l;

}

string getRecord(){

stringstream s;

s<<"H"

<<left<<setw(6)<<std::setfill(' ')<<name

<<right<<setw(6)<<setfill('0')<<hex<<startAddress

<<setw(6)<<setfill ('0')<<hex<<length;

return s.str();

}

};

class text{

public:

int startAddress, length;

string opcodes;

// bool isFull;

int TOTAL;

text(int s){

startAddress = s;

opcodes ="";

length = 0;

TOTAL = 30;

// isFull = false;

}

string getRecord(){

stringstream s;

s<<"T"

<<setw(6)<<setfill('0')<<hex<<startAddress

<<setw(2)<<hex<<length

<<opcodes;

return s.str();

}

void add(string opcode){

if(canAdd(opcode)){

length += opcode.length()/2;

opcodes.append(opcode);

}

}

bool canAdd(string opcode){

if(length + opcode.length()/2 <= TOTAL) return true;

else return false;

}

//TODO - fix the bug

void fill(){

while(canAdd(" ")){

opcodes.append(" ");

TOTAL--;

}

// isFull = true;

}

};

class end1{

public:

int startAddress;

end1(int s){

startAddress = s;

}

string getRecord(){

stringstream s;

s<<"E"<<setw(6)<<setfill('0')<<hex<<startAddress;

return s.str();

}

};

string assemble(int code, int address, bool x){

stringstream s;

s<<setfill('0')<<setw(2)<<hex<<code;

if(x){

address += 0x8000;

}

s<<setfill('0')<<setw(4)<<hex<<address;

return s.str();

}

int main()

{

string label, mnumonic, operand, data, locstr, location;

int startAddress, endAddress, locctr;

ifstream input, optab, symtab;

ofstream output, listing;

int code;

input.open("input.txt");

optab.open("optab.txt");

output.open("output.txt");

symtab.open("symtab.txt");

listing.open("listing.txt");

while(!optab.eof()){

optab>>mnumonic>>hex>>code;

opcodes.insert(pair<string, int>(mnumonic,code));

}

while(!symtab.eof()){

symtab>>mnumonic>>hex>>code;

symbols.insert(pair<string, int>(mnumonic,code));

}

getline(input, data);

std::stringstream iss(data);

getline(iss, locstr ,'\t');

getline(iss, label ,'\t');

getline(iss, mnumonic ,'\t');

getline(iss, operand ,'\t');

if(!mnumonic.compare("START")){

stringstream convert(operand);

convert>>hex>>startAddress;

locctr = startAddress;

}

int length = 20;

// Write head record

head h(label, startAddress, length);

output<<h.getRecord()<<endl;

// Initialize text record

text \*t = new text(startAddress);

while(mnumonic.compare("END")){

getline(input, data);

std::stringstream iss(data);

getline(iss, location ,'\t');

getline(iss, label ,'\t');

// if comment then continue

if(!label.compare(".")){

listing<<data<<endl;

continue;

}

getline(iss, mnumonic ,'\t');

getline(iss, operand ,'\t');

int address = 0;

bool isIndex = false;

string objectcode = "";

// search optab for mnumonic

map<string,int>::iterator it = opcodes.find(mnumonic);

// if mnumonic found

if(it!=opcodes.end()){

// if there is symbol in OPERAND field

if(operand.compare("")){

std::size\_t found = operand.find(",X");

string op = operand;

if(found!=std::string::npos){

op.replace(found,2,"");

isIndex = true;

}

map<string,int>::iterator i = symbols.find(op);

if(i!=symbols.end()){

address = i->second;

}

else{

address = 0;

cout<<op<<": undefined Symbol"<<endl;

}

}

else{

address = 0;

}

objectcode = assemble(opcodes.find(mnumonic)->second, address, isIndex);

}

// byte or word

else if(!mnumonic.compare("BYTE")){

//if starts from C

if(operand.at(0) == 'C'){

for(int i=2; i<operand.length()-1; i++){

stringstream s;

s<<hex<<(int)operand.at(i);

objectcode+= s.str();

}

}

// If starts from X

if(operand.at(0) == 'X')

objectcode = operand.substr(2, operand.length()-3);

}

else if(!mnumonic.compare("WORD")){

stringstream s;

s<<setfill('0')<<setw(6)<<hex<<stoi(operand);

objectcode = s.str();

}

else if(!mnumonic.compare("RESW")){

t->fill();

}

else if(!mnumonic.compare("RESB")){

t->fill();

}

if(!t->canAdd(objectcode)){

output<<t->getRecord()<<endl;

stringstream s(location);

int locctr;

s>>hex>>locctr;

t = new text(locctr);

}

t->add(objectcode);

listing<<location<<"\t"<<label<<"\t"<<mnumonic<<"\t"<<operand<<"\t"<<"\t"<<objectcode<<endl;

if(!mnumonic.compare("END")){

break;

}

}

output<<t->getRecord()<<endl;

// End record

end1 e(startAddress);

output<<e.getRecord()<<endl;

// Close files

input.close();

optab.close();

symtab.close();

output.close();

}

**input.txt**

2000 COPY START 2000

2000 LDA FIVE

2003 STA ALPHA

2006 LDCH CHARZ

2009 STCH C1

200c ALPHA RESW 1

200f FIVE WORD 5

2012 CHARZ BYTE C'Z'

2013 C1 RESB 1

2014 END

output.txt

HCOPY 002000000014

T0020000c00200f0c200c502012542013

T00200f040000055a

E002000

**Develop absolute loader.**

#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

#include <iomanip>

using namespace std;

int main(){

int start, length;

ifstream input;

ofstream output;

input.open("input.txt");

output.open("output.txt");

string data;

char type;

getline(input, data);

type = data.at(0);

if(type=='H'){

string startString = data.substr(7, 6);

string lengthString = data.substr(13, 6);

stringstream s(startString);

stringstream ss(lengthString);

s>>hex>>start;

ss>>hex>>length;

}

else{

start = 0;

length = 0;

}

while(type != 'E'){

getline(input, data);

type = data.at(0);

if(type=='T'){

string opcodes = data.substr(9);

string locString = data.substr(1,6);

stringstream s(locString);

int loc;

s>>hex>>loc;

while(opcodes.length() > 0)

{

output<<hex<<loc<<" "<<opcodes.substr(0,2)<<endl;

opcodes.erase(0, 2);

loc++;

}

}

}

}

input.txt

HCOPY 001000000013

T0010000c142033483039102036220044

T0020000a298300230000f1302015

E001000

output.txt

1000 14

1001 20

1002 33

1003 48

1004 30

1005 39

1006 10

1007 20

1008 36

1009 22

100a 00

100b 44

2000 29

2001 83

2002 00

2003 23

2004 00

2005 00

2006 f1

2007 30

2008 20

2009 15

**Develop relocating loader using bitmask.**

#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

#include <iomanip>

#include <bitset>

#include <cstring>

using namespace std;

int main(){

int start, length;

ifstream input;

ofstream output;

input.open("input.txt");

output.open("output.txt");

string data;

char type;

cout<<"Enter starting address: "<<endl;

cin>>hex>>start;

getline(input, data);

type = data.at(0);

if(type=='H'){

}

while(type != 'E'){

getline(input, data);

type = data.at(0);

if(type=='T'){

string address, bin;

// strings in integers

int loc = 0, mod, add;

// Mask

int mask = 0x800;

string locString = data.substr(1, 6);

string modifier = data.substr(7, 3);

string opcodes = data.substr(10);

stringstream s1(locString);

s1>>hex>>loc;

loc += start;

stringstream s2(modifier);

s2>>hex>>mod;

while(opcodes.length() > 0)

{

address = opcodes.substr(0,6);

stringstream s3(address);

s3>>hex>>add;

int a = mod;

if(a & mask)

{

add += start;

}

mod = mod << 1;

a = mod;

output<<hex<<loc<<" "<<hex<<add<<endl;

opcodes.erase(0,6);

loc += 3;

}

}

}

}

input.txt

HCOPY 000000000

T000000C80141033481039901776921765571765

T002011E00232838432979892079892060682849

E000000

output.txt

4000 145033

4003 485039

4006 901776

4009 921765

400c 575765

6011 236838

6014 436979

6017 896079

601a 892060

601d 682849