System Software

Nehal Jhajharia (U20CS093)

Assembler

```
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <vector>


using namespace std;

std::string to_string(int i)
{
   std::stringstream ss;
   ss << i;
   return ss.str();
}

string convert(int decimal) //Number to Binary Convertion
{
   string binary = "";
   for(int i = decimal;i > 0;i/=2)
   {
       ostringstream str1;
       str1 << ( i % 2 );
       binary = str1.str();
   }
   if(binary.length() < 8)
       binary = string(8-binary.length(),'0').append(binary);
   return binary;
}

//Data Type Declarations
struct mnemonics{
   string name;
```

```
   string binary;
   int size;
}mot[13];
//struct
struct symbol{ //Symbol Table format
   string name;
   string type;
   int location;
   int size;
   int section_id;
   string is_global;
};

struct section{ //Section Table format
   int id;
   string name;
   int size;
};

vector<symbol> symlab; //Symbol Table
vector<section> sec; //Section Table
int lc = 0; //Controls Location Counter
int sec_id = 0; //Manage section Id
int var_lc; //Store location of variable in Pass2
ifstream infile; //Input File Stream
ofstream outfile; //Output File Stream
string word; //Read Word by Word from file
string temp; //Temporary Variable
int control; //Control Variable for search
int size = 0;  //Control Variable size for search

void init()
{
   //Initializing Machine Opcode Table
   mot[0] = {"ADD","00000001",1};
   mot[1] = {"ADDI","00000010",5};
   mot[2] = {"CMP","00000011",5};
   mot[3] = {"INC","00000100",1};
   mot[4] = {"JE","00000101",5};
   mot[5] = {"JMP","00000110",5};
   mot[6] = {"LOAD","00000111",5};
```

```
   mot[7] = {"LOADI","00001000",1};
   mot[8] = {"MVI","00001001",5};
   mot[9] = {"MOV","00001010",1};
   mot[10] = {"STOP","00001011",1};
   mot[11] = {"STORE","00001100",5};
   mot[12] = {"STORI","00001101",1};
}

int search_mot(string opcode) //Search Machine Opcode Table
{
   int index = -1;
   for(int i = 0;i < 13;i++)
   {
       if(mot[i].name == opcode)
       {
           index = i;
           break;
       }
   }
   return index;
}

int search_symbol(string variable) //Find Location of the Given Symbol
{
   int location = -1;
   for(vector<symbol>::const_iterator i = symlab.begin();i != symlab.end();++i)
   {
       if(i->name == variable)
       {
           location = i->location;
           break;
       }
   }
   return location;
}

int size_evaluation(string data) //Evaluate size of Variable defined
{
   int size = 0;
   for(int i = 0;i < data.length();i++)
   {
```

```
        if(data[i] == ',')
            size += 4;
    }
    size += 4;
    return size;
}

string data_break(string data) //Convert String of Input Number into Binary String
{
    string final;
    string temporary = "";
    for(int i = 0;i < data.length();i++)
    {
        if(data[i] == ',')
        {
            final += convert(atoi(temporary.c_str()))+",";
            temporary = "";
        }
        else
            temporary += data[i];
    }
    final.erase(final.length()-1,1);
    return final;
}

void store_symlab() //Storing Symbol Table in File
{
    outfile.open("symbol.csv");
    outfile << "Name,Type,Location,Size,SectionID,IsGlobal\n";
    for(vector<symbol>::const_iterator i = symlab.begin();i != symlab.end();++i)
    {
        outfile << i->name<<",";
        outfile << i->type<<",";
        outfile << i->location<<",";
        outfile << i->size<<",";
        outfile << i->section_id<<",";
        outfile << i->is_global<<"\n";
    }
    outfile.close();
}
```

```
void store_sec() //Storing Section Table in File
{
   outfile.open("section.csv");
   outfile << "ID,Name,Size\n";
   for(vector<section>::const_iterator i = sec.begin();i != sec.end();++i)
   {
       outfile << i->id<<",";
       outfile << i->name<<",";
       outfile << i->size<<"\n";
   }
   outfile.close();
}

void pass1()
{
   infile.open("input.txt");
   while(infile >> word)
   {
       control = search_mot(word);
       if(control == -1)
       {
           temp = word;
           if(word.find(":") != -1)//Label is Found
           {

symlab.push_back({temp.erase(word.length()-1,1),"label",lc,-1,sec_id,"false"});
//Inserting into Symbol Table
           }
           else if(word == "section")//Section is Found
           {
               infile >> word;
               sec_id++;
               sec.push_back({sec_id,word,0}); //Inserting into Section Table
               if(sec_id != 1) // Updating previous section Size
               {
                   sec[sec_id-2].size = lc;
                   lc = 0;
               }
           }
           else if(word == "global") //Global Varaible is Found
           {
```

```
                infile >> word;
                symlab.push_back({word,"label",-1,-1,-1,"true"}); //Inserting into
Symbol Table
            }
            else if(word == "extern") //External Variable is found
            {
                infile >> word;
                symlab.push_back({word,"external",-1,-1,-1,"false"}); //Inserting into
Symbol Table
            }
            else//Variable is Found
            {
                infile >> word;
                infile >> word;
                size = size_evaluation(word);
                symlab.push_back({temp,"var",lc,size,sec_id,"false"}); //Inserting into
Symbol Table
                lc += size;
            }
        }
        else
        {
            if(!(control == 7 || control == 12)) //LOADI and STOREI do not have any
paramenter
                infile >> word;
            if(control==2 || control==8 || control == 9)
                infile >> word;
            lc += mot[control].size;
        }
    }

    sec[sec_id-1].size = lc; //Updating size of current Section

    store_symlab();
    store_sec();

    infile.close();
}

void pass2()
{
```

```
infile.open("input.txt");
outfile.open("output.txt");
while(infile >> word)
{
    control = search_mot(word);
    if(control == -1)
    {
        temp = word;
        if(word.find(":") != -1) //No Machine Code for Label
        {
            outfile << "";
        }
        else if(word == "global") //No change in Global content
        {
            infile >> word;
            outfile <<"global "<<word<<endl;
        }
        else if(word == "extern") //No change in External Content
        {
            infile >> word;
            outfile <<"extern "<<word<<endl;
        }
        else if(word == "section") //No change in Section content
        {
            infile >> word;
            outfile <<"section ."<<word<<endl;
            lc = 0;
        }
        else //Variables are converted to binary along with the values
        {
            infile >> word;
            infile >> word;
            outfile <<convert(lc)<<" "<<data_break(word)<<endl;
            size = size_evaluation(word);
            lc += size;
        }
    }
    else
    {
        outfile <<convert(lc)<<" "<<mot[control].binary;
        if(control==0||control==3) //ADD and INC have defined register following it
```

```
            {
                infile >> word;
                outfile <<" "<<word;
            }
            else if(control==1 || control==4 || control==5 || control==6 ||
control==11) //ADDI, JE, JMP, LOAD and STORE have one constant following it
            {
                infile >> word;
                var_lc = search_symbol(word);
                if(var_lc == -1)
                    outfile <<" "<<convert(atoi(word.c_str()));
                else
                    outfile <<" "<<convert(var_lc);
            }
            else if(control==2 || control==8) //CMP and MVI have one register and one
constant following it
            {
                infile >> word;
                outfile <<" "<<word;
                infile >> word;
                var_lc = search_symbol(word);
                if(var_lc == -1)
                    outfile <<" "<<convert(atoi(word.c_str()));
                else
                    outfile <<" "<<convert(var_lc);
            }
            else if(control == 9) //MOV have both registers following it
            {
                infile >> word;
                outfile <<" "<<word;
                infile >> word;
                outfile <<" "<<word;
            }
            lc += mot[control].size;
            outfile << "\n";
        }
    }
    outfile.close();
    infile.close();
}
```

```
int main()
{
   init();
   pass1();
   lc = 0;
   pass2();
   return 0;
}
```