Semester	T.E. Semester VI– SPCC
Subject	Software Engineering
Subject Professor In-charge	Prof. Pankaj Vanvari
Assisting Teachers	Prof. Pankaj Vanvari
Laboratory	M310B

Student Name	Deep Salunkhe
Roll Number	21102A0014
TE Division	A

Title:

2 Pass Assembler

Approach:

1. **Input Processing:**
    - Beginning with reading the assembly language code from a file, I wrote the **readAssembly** function. This function meticulously takes each word in the code, storing them in a vector of strings. This approach not only helps in organizing the code but also enables tracking the current position within the source file during both passes of assembly.
2. **Database Management:**
    - To effectively manage the various components of the assembly process, I went for map data structures in C++. These hash tables facilitated the creation of essential databases:
        - **POT (Program Operand Table)** and **MOT (Mnemonic Operand Table)**: These tables, implemented using maps, store vital information related to mnemonics and equates.
        - **Label Definition Table**: Another map-based structure designed to record memory addresses where labels are defined.
        - **Label Reference Table**: Similar to the label definition table, this map records memory addresses where labels are referenced.
3. **Output Generation (Simulates Pass 1):**

- The core functionality of generating intermediate representation falls under this phase, simulated by the **createOutputFile** function. Here, memory addresses for labels are computed, EQU directives are handled, and the sizes of DC and DS directives are determined. This preparatory phase lays the groundwork for producing the final machine code output.

4. **Final Object Code Generation (Simulates Pass 2):**

   - The **Final_OBJ_File** function creates the final obj file . Using the previously generated data structures, this phase resolves symbolic references and generates accurate machine code sequences. The resulting machine code is then saved to a file which has all the code in the form of opcode, registers and address location

Challenges Encountered:

- Throughout the process, I faced many challenges, notably in converting the code from the .asm file into a format conducive to further processing. Additionally, handling references before definitions posed a significant challenging to implement.

Implementation:

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

using namespace std;

int readAssembly(string &fileName, vector<string> &input)
{
    char ch;
    fstream fp;
    fp.open(fileName.c_str(), std::fstream::in);

    if (!fp)
    {
        cerr << "Error opening the file: " << fileName << endl;
        return 1; // Return an error code
    }

    string word;
    while (fp >> noskipws >> ch)
    {
        if (ch == '\n')
        {
```

```
            input.push_back(word);
            input.push_back(";");
            word = "";
        }
        else if (ch == ' ')
        {
            input.push_back(word);
            word = "";
        }
        else
        {
            word += ch;
        }
    }

    input.push_back(word);

    fp.close();
    return 0; // Return success code
}

void createOutputFile(vector<string> &input, map<string, string> &mneomon-
ics_oppcode, map<string, int> &mneomonics_size, map<string, int> &label_def,
map<string, int> &label_ref, map<string, string> &equ_values, map<string, int>
&dc_ds_length, map<string, int> &dc_ds_address, string &fileName)
{
    ofstream outputFile;
    outputFile.open(fileName.c_str(), std::ios::out);

    if (!outputFile.is_open())
    {
        cerr << "Error creating the output file: " << fileName << endl;
        return;
    }

    int startAddress = 0;
    // counter to now which part of the line we are in
    int counter = 0;

    for (int i = 0; i < input.size(); i++)
    {
        string temp = input[i];

        if (temp == ";")
```

```
        {
            counter = 0;
            continue;
        }

        if (mneomonics_oppcode.find(temp) == mneomonics_oppcode.end())
        {

            if (input[i + 1] == "EQU")
            {
                // Convert character to integer without using stoi

                equ_values[temp] = input[i + 2];
                i = i + 2;
                continue;
            }

            if (input[i + 1] == "DC")
            {

                string sv = input[i + 2];
                if (sv[0] == 'F')
                    dc_ds_length[temp] = 4;

                dc_ds_address[temp] = startAddress;
                cout << "some randon" << endl;
                cout << startAddress << endl;
                startAddress = startAddress + dc_ds_length[temp];
                i = i + 2;
                continue;
            }

            if (input[i + 1] == "DS")
            {

                string sv = input[i + 2];
                if (sv[1] == 'F')
                    dc_ds_length[temp] = 4 * (sv[0] - '0');

                dc_ds_address[temp] = startAddress;
                startAddress = startAddress + dc_ds_length[temp];
                i = i + 2;
                continue;
            }
```

```
            if (counter == 0)
            {
                label_ref[temp] = startAddress;
            }

            if (counter == 1)
            {
                // as I am adding the size of the mneomonics in the start address
previously
                // so I am subtracting it from the start address to get the ad-
dress of the label
                label_def[temp] = startAddress - mneomonics_size[input[i - 1]];
            }

            //  cout<<"some randon"<<endl;
            // cout<<startAddress<<endl;
        }

        if (mneomonics_oppcode.find(temp) != mneomonics_oppcode.end())
            outputFile << temp << " " << mneomonics_oppcode[temp] << " " << star-
tAddress << endl;
        startAddress += 4;
        counter++;
    }

    outputFile.close();
}

// frinting the label ref and def
void ref_def(std::map<std::string, int> &label_def, std::map<std::string, int>
&label_ref)
{
    std::cout << "Label Def" << std::endl;
    for (std::map<std::string, int>::iterator it = label_def.begin(); it != la-
bel_def.end(); ++it)
    {
        std::cout << it->first << " " << it->second << std::endl;
    }
    std::cout << "Label Ref" << std::endl;
    for (std::map<std::string, int>::iterator it = label_ref.begin(); it != la-
bel_ref.end(); ++it)
    {
        std::cout << it->first << " " << it->second << std::endl;
```

```
    }
}

void EQU_Print(map<string, string> &equ_values)
{
    std::cout << "EQU Values" << std::endl;
    for (std::map<std::string, string>::iterator it = equ_values.begin(); it != 
equ_values.end(); ++it)
    {
        std::cout << it->first << " " << it->second << std::endl;
    }
}

void DC_DS_Print(map<string, int> dc_ds_length, map<string, int> dc_ds_address)
{
    std::cout << "DS&DC address" << std::endl;
    for (std::map<std::string, int>::iterator it = dc_ds_address.begin(); it != 
dc_ds_address.end(); ++it)
    {
        std::cout << it->first << " " << it->second << std::endl;
    }

    std::cout << "DS&DC length" << std::endl;
    for (std::map<std::string, int>::iterator it = dc_ds_length.begin(); it != 
dc_ds_length.end(); ++it)
    {
        std::cout << it->first << " " << it->second << std::endl;
    }
}

void print_vector(vector<string> &input)
{
    for (int i = 0; i < input.size(); i++)
    {
        cout << input[i] << " ";
    }
    cout << endl;
}

void Final_OBJ_File(vector<string> &input, map<string, string> &mneomon-
ics_oppcode, map<string, int> &mneomonics_size, map<string, int> &label_def, 
map<string, int> &label_ref, map<string, string> &equ_values, map<string, int> 
&dc_ds_length, map<string, int> &dc_ds_address, string &fileName)
{
```

```
    // what about  this
    ofstream outputFile;
    outputFile.open(fileName.c_str(), std::ios::out);

    if (!outputFile.is_open())
    {
        cerr << "Error creating the Final output file: " << fileName << endl;
        return;
    }

    int base_address = 15;
    int index_pointer = 0;
    int startAddress = 0;

    for (int i = 0; i < input.size(); i++)
    {
        string temp = input[i];

        if (temp == ";")
        {
            continue;
        }
        if (input[i + 1] == "EQU")
        {
            i = i + 2;
            continue;
        }

        if (mneomonics_oppcode.find(temp) != mneomonics_oppcode.end())
        {

            outputFile << startAddress << " ";
            outputFile << mneomonics_oppcode[temp] << " ";
            startAddress += mneomonics_size[temp];

            // here we will check if the instruction if of RR type or RX type
            // I have not included all the cases just according to the given in-
put

            if ((input[i + 2] == "1" ||
                 input[i + 2] == "2" ||
                 input[i + 2] == "3" ||
                 input[i + 2] == "4" ||
```

```
                input[i + 2] == "5" ||
                input[i + 2] == "6" ||
                input[i + 2] == "7") &&
               equ_values.find(input[i + 1]) != equ_values.end())
           {
               outputFile << equ_values[input[i + 1]] << " " << input[i + 2];
               startAddress += 2;
           }

           // for the instruction of RX type

           if ((input[i + 1] == "1" ||
                input[i + 1] == "2" ||
                input[i + 1] == "3" ||
                input[i + 1] == "4" ||
                input[i + 1] == "5" ||
                input[i + 1] == "6" ||
                input[i + 1] == "7") &&
               dc_ds_length.find(input[i + 2]) != dc_ds_length.end())
           {
               outputFile << input[i + 1] << " " << index_pointer << " " <<
base_address << " " << dc_ds_address[input[i + 2]];
               startAddress += 4;
           }

           // for the instruction of RX type
           if (equ_values.find(input[i + 1]) != equ_values.end() &&
dc_ds_length.find(input[i + 2]) != dc_ds_length.end())
           {
               outputFile << equ_values[input[i + 1]] << " " << index_pointer <<
" " << base_address << " " << dc_ds_address[input[i + 2]];
               startAddress += 4;
           }

       }else if (dc_ds_length.find(input[i]) != dc_ds_length.end())
       {
           string temp = input[i];

           // allign the address
           if (startAddress % 4 != 0)
           {
               startAddress += 4 - startAddress % 4;
           }
```

```
            if (temp[0] == 'F')
            {
                outputFile << startAddress << " " << temp[1];
                startAddress += 4;
            }
            else
            {
                outputFile << startAddress << " "
                           << "XXXXX";
                startAddress += dc_ds_length[input[i]];
            }
            startAddress += dc_ds_length[input[i]];
        }

        outputFile << endl;
    }

    outputFile.close();
}

int main()
{

    // the data set of the  mneomonics_oppcode and mneomonics
    map<string, string> mneomonics_oppcode;
    mneomonics_oppcode["L"] = "000AH";
    mneomonics_oppcode["AR"] = "000CH";
    mneomonics_oppcode["ST"] = "0003H";
    mneomonics_oppcode["JUMP"] = "0002H";

    map<string, int> mneomonics_size;
    mneomonics_size["l"] = 1;
    mneomonics_size["AR"] = 1;
    mneomonics_size["ST"] = 1;
    mneomonics_size["JUMP"] = 1;
    map<string, int> label_def;
    map<string, int> label_ref;

    // mapping EQU and the values
    map<string, string> equ_values;

    // DC&DS Address and length
    map<string, int> dc_ds_address;
    map<string, int> dc_ds_length;
```

```
    // the dataset end here
    vector<string> input;
    string inputfile;
    string outputfile;
    string final_outputfile;

    cout << "Enter filename to read" << endl;
    cin >> inputfile;

    // Call the readAssembly function to populate the input vector
    if (readAssembly(inputfile, input) == 0)
    {
        print_vector(input);
        cout << "Save file as" << endl;
        cin >> outputfile;
        final_outputfile = "f" + outputfile;

        createOutputFile(input, mneomonics_oppcode, mneomonics_size, label_def,
label_ref, equ_values, dc_ds_length, dc_ds_address, outputfile);
        Final_OBJ_File(input, mneomonics_oppcode, mneomonics_size, label_def, la-
bel_ref, equ_values, dc_ds_length, dc_ds_address, final_outputfile);
        cout << "File Saved" << endl;
        ref_def(label_def, label_ref);
        EQU_Print(equ_values);
        DC_DS_Print(dc_ds_length, dc_ds_address);
    }

    return 0;
}
```

End Result:

```
PS E:\GIt> cd "e:\GIt\SEM-6\SPCC\" ; if ($?) { g++ temp.cpp -o temp } ; if ($?) { .\temp }
Enter filename to read
asm2.txt
ACC EQU 4 ; L ACC FOUR ; L 3 FIVE ; AR ACC 3 ; ST ACC RESULT ; FOUR DC F4 ; FIVE DC F5 ; RESULT DS 1F
Save file as
objfinal.txt
some randon
48
some randon
52
File Saved
Label Def
3 16
ACC 39
Label Ref
EQU Values
ACC 4
DS&DC address
FIVE 52
FOUR 48
RESULT 56
DS&DC length
FIVE 4
FOUR 4
RESULT 4
PS E:\GIt\SEM-6\SPCC>
```

objfinal.txt M

SEM-6 > SPCC > objfinal.txt

```
1  L 000AH 0
2  L 000AH 12
3  AR 000CH 24
4  ST 0003H 36
5
```

SEM-6 > SPCC > ≡ fobjfinal.txt

```
0 000AH 4 0 15 48

4 O
12 000AH 3 0 15 52

16 I
24 000CH 4 3


27 0003H 4 0 15 56

32 XXXXX
40 O


48 I


56 XXXXX

```