PMAT 402 - Systems Programming
Assignment -1
Two-Pass Assembler

Gandholi Sarat - 23008

April 11, 2025

Contents

1 Objective

The objective of this project is to design and implement a Two-Pass Assembler for the Simplified Instructional Computer (SIC) architecture. The assembler translates assembly language programs into object code, manages a symbol table, and handles all standard SIC instructions and directives.

2 Idea of Implementation

The assembler works in two passes:

- **Pass 1:**
    - Initializes location counter.
    - Parses each line to extract label, opcode, operand.
    - Builds the symbol table (SymTab) with label-address mappings.
    - Generates an intermediate file with line addresses.
- **Pass 2:**
    - Reads the intermediate file and symbol table.
    - Uses the operation table (OpTab) to fetch opcode for instructions.
    - Replaces symbols with actual addresses.
    - Generates object code records (Header, Text, End).

3 Functions Description

Below are the key functions used in the assembler, along with their purpose and code implementation.

3.1 Hexadecimal to Decimal Conversion

This function converts a hexadecimal string (e.g., `"1A3"`) into its equivalent decimal integer. It processes each digit from right to left, multiplying it by powers of 16 based on its position.

```
int hextodec(string hex) {
    int len = hex.length();
    int base = 1;
    int dec_num = 0;
    for (int i = len - 1; i >= 0; i--) {
        dec_num += dec(hex[i]) * base;
        base *= 16;
    }
    return dec_num;
}
```

Listing 1: hextodec() - Hex to Decimal

3.2 Decimal to Hexadecimal Conversion

This function takes a decimal integer and converts it into a hexadecimal string. It uses a stack to reverse the remainders when dividing the number by 16, constructing the hexadecimal result from most to least significant digit.

```
string dectohex(int dec_num) {
    string hex_str = "";
    stack<int> stack;
    int div = dec_num;
    int rem = div % 16;
    stack.push(rem);
    div /= 16;
    while (div > 15) {
        rem = div % 16;
        stack.push(rem);
        div /= 16;
    }
    stack.push(div);
    while (!stack.empty()) {
        hex_str += hex(stack.top());
        stack.pop();
    }
    return hex_str;
}
```

Listing 2: dectohex() - Decimal to Hex

3.3 String Padding

These functions are used to format output fields by adjusting string lengths.

- `padWithZeros` ensures a string has a specified length by prepending zeros to the front.
- `padWithSpaces` appends spaces to the end to meet the desired length.

```
std::string padWithZeros(const std::string& input, int desiredLength) {
    return (input.length() >= desiredLength) ? input :
        std::string(desiredLength - input.length(), '0') + input;
}

std::string padWithSpaces(const std::string& input, int desiredLength) {
    return (input.length() >= desiredLength) ? input :
        input + std::string(desiredLength - input.length(), ' ');
}
```

Listing 3: padWithZeros() and padWithSpaces()

3.4 Token Splitter

This function splits a string into tokens based on a specified delimiter. It's especially useful for breaking input lines (like assembly instructions) into label, opcode, and operand fields.

```
vector<string> split(string str, char del) {
    vector<string> v;
    string temp = "";
    for (int i = 0; i < str.size(); i++) {
        if (str[i] != del) temp += str[i];
        else {
            v.push_back(temp);
            temp = "";
        }
    }
    v.push_back(temp);
    return v;
}
```

Listing 4: split() - Tokenizing by delimiter

3.5 Character to Decimal Conversion

Function: `dec()`
This function converts a hexadecimal character (0-9, A-F) into its corresponding decimal value. It is used inside the `hextodec()` function.

```
int dec(char c) {
    switch (c) {
        case ’A’: return 10;
        case ’B’: return 11;
        case ’C’: return 12;
        case ’D’: return 13;
        case ’E’: return 14;
        case ’F’: return 15;
        default: return stoi(to_string(c)) - 48; // Converts ASCII to int
    }
}
```

Listing 5: dec() - Hex Character to Decimal

—

3.6 Decimal to Hex Character Conversion

Function: `hex()`
This function is a helper for `dectohex()` and is used to convert a number between 0-15 into its corresponding hexadecimal character.

```
string hex(int d) {
    switch (d) {
        case 10: return "A";
        case 11: return "B";
        case 12: return "C";
        case 13: return "D";
        case 14: return "E";
        case 15: return "F";
        default: return to_string(d);
    }
}
```

Listing 6: hex() - Decimal (0-15) to Hex Character

—

3.7 Opcode Table Initialization

Function: `const_optab()`
This function creates and returns a map (dictionary) containing mnemonic instructions as keys and their respective opcode values in hexadecimal as values. It is a core part of the assembler's translation logic.

```
map<string, string> const_optab() {
    map<string, string> optab;
    optab["ADD"] = "18";
    optab["ADDF"] = "58";
    optab["ADDR"] = "90";
    optab["AND"] = "40";
    optab["CLEAR"] = "B4";
    optab["COMP"] = "28";
    optab["COMPF"] = "88";
    optab["COMPR"] = "A0";
    optab["DIV"] = "24";
    optab["DIVF"] = "64";
    optab["DIVR"] = "9C";
    optab["FIX"] = "C4";
    optab["FLOAT"] = "C0";
    optab["HIO"] = "F4";
    optab["J"] = "3C";
    optab["JEQ"] = "30";
    optab["JGT"] = "34";
    optab["JLT"] = "38";
    optab["JSUB"] = "48";
    optab["LDA"] = "00";
    optab["LDB"] = "68";
    optab["LDCH"] = "50";
    optab["LDF"] = "70";
    optab["LDL"] = "08";
    optab["LDS"] = "6C";
    optab["LDT"] = "74";
    optab["LDX"] = "04";
    optab["LPS"] = "D0";
    optab["MUL"] = "20";
    optab["MULF"] = "60";
    optab["MULR"] = "98";
    optab["NORM"] = "C8";
    optab["OR"] = "44";
    optab["RD"] = "D8";
    optab["RMO"] = "AC";
    optab["RSUB"] = "4C";
    optab["SHIFTL"] = "A4";
    optab["SHIFTR"] = "A8";
    optab["SIO"] = "F0";
    optab["SSK"] = "EC";
    optab["STA"] = "0C";
    optab["STB"] = "78";
    optab["STCH"] = "54";
    optab["STF"] = "80";
    optab["STI"] = "D4";
    optab["STL"] = "14";
    optab["STS"] = "7C";
    optab["STSW"] = "E8";
    optab["STT"] = "84";
    optab["STX"] = "10";
    optab["SUB"] = "1C";
    optab["SUBF"] = "5C";
    optab["SUBR"] = "94";
    optab["SVC"] = "B0";
    optab["TD"] = "E0";
    optab["TIO"] = "F8";
    optab["TIX"] = "2C";
    optab["TIXR"] = "B8";
    optab["WD"] = "DC";
    return optab;
}
```

Listing 7: const_optab() - Opcode Table Initialization

4 Full Code

4.1 Operation Table

```
#include <iostream>
#include <string>
#include <stack>
#include <map>
#include <string>

using namespace std;

map <string, string> const_optab()
{
    map <string, string> optab;

        optab["ADD"] = "18";
        optab["ADDF"] = "58";
        optab["ADDR"] = "90";
        optab["AND"] = "40";
        optab["CLEAR"] = "B4";
        optab["COMP"] = "28";
        optab["COMPF"] = "88";
        optab["COMPR"] = "A0";
        optab["DIV"] = "24";
        optab["DIVF"] = "64";
        optab["DIVR"] = "9C";
        optab["FIX"] = "C4";
        optab["FLOAT"] = "C0";
        optab["HIO"] = "F4";
        optab["J"] = "3C";
        optab["JEQ"] = "30";
        optab["JGT"] = "34";
        optab["JLT"] = "38";
        optab["JSUB"] = "48";
        optab["LDA"] = "00";
        optab["LDB"] = "68";
        optab["LDCH"] = "50";
        optab["LDF"] = "70";
        optab["LDL"] = "08";
        optab["LDS"] = "6C";
        optab["LDT"] = "74";
        optab["LDX"] = "04";
        optab["LPS"] = "D0";
        optab["MUL"] = "20";
        optab["MULF"] = "60";
        optab["MULR"] = "98";
        optab["NORM"] = "C8";
        optab["OR"] = "44";
        optab["RD"] = "D8";
        optab["RMO"] = "AC";
        optab["RSUB"] = "4C";
        optab["SHIFTL"] = "A4";
        optab["SHIFTR"] = "A8";
        optab["SIO"] = "F0";
        optab["SSK"] = "EC";
        optab["STA"] = "0C";
        optab["STB"] = "78";
        optab["STCH"] = "54";
        optab["STF"] = "80";
        optab["STI"] = "D4";
        optab["STL"] = "14";
        optab["STS"] = "7C";
        optab["STSW"] = "E8";
        optab["STT"] = "84";
        optab["STX"] = "10";
        optab["SUB"] = "1C";
        optab["SUBF"] = "5C";
        optab["SUBR"] = "94";
        optab["SVC"] = "B0";
        optab["TD"] = "E0";
        optab["TIO"] = "F8";
        optab["TIX"] = "2C";
        optab["TIXR"] = "B8";
        optab["WD"] = "DC";

        return optab;
}
```

Listing 8: Operation Table (optab)

4.2 Utility Functions

```
#include <iostream>
#include <string>
#include <stack>
#include <iomanip>
#include <sstream>
using namespace std;

int dec(char c)
{
    switch (c)
    {
        case 'A':
        return 10;
        case 'B':
        return 11;
        case 'C':
        return 12;
        case 'D':
        return 13;
        case 'E':
        return 14;
        case 'F':
        return 15;
```

```
            default:
            return stoi(to_string(c))-48;   //ascii value to magnitude
        }
    }

    int hextodec(string hex)
    {
        int len = hex.length();
        int base = 1;
        int dec_num = 0;
        for (int i = len-1; i>=0; i--)
        {
            dec_num += dec(hex[i])*base;
            base = base*16;
        }
        return dec_num;
    }

    string hex(int d)
    {
        switch (d)
        {
            case 10:
            return "A";
            case 11:
            return "B";
            case 12:
            return "C";
            case 13:
            return "D";
            case 14:
            return "E";
            case 15:
            return "F";
            default:
            return to_string(d);
        }
    }

    string dectohex(int dec_num)
    {
        string hex_str = "";
        stack<int> stack;

        int div, rem;
        div = dec_num;
        rem = div % 16;
        //cout << "rem:" << rem << endl;
        stack.push(rem);
        div = (div)/16;
        //cout << "div:" << div << endl;
        while(div > 15)
        {
            rem = div % 16;
```

```
         //cout << "rem:" << rem << endl;
         stack.push(rem);
         div = (div)/16;
         //cout << "div:" << div << endl;
     }
     stack.push(div);
     while(!stack.empty())
     {
         //cout << stack.top();

         hex_str = hex_str + hex(stack.top());
         stack.pop();
     }
     return hex_str;

 }

 std::string padWithZeros(const std::string& input, int desiredLength)
{
        if (input.length() >= desiredLength) {
         return input;
     } else {
         std::stringstream ss;
         ss << std::string(desiredLength - input.length(), '0') <<
input;
         return ss.str();
     }
 }

 std::string padWithSpaces(const std::string& input, int desiredLength)
 {
        if (input.length() >= desiredLength) {
         return input;
     } else {
         std::stringstream ss;
         ss << input << std::string(desiredLength - input.length(), ' '
);
         return ss.str();
     }
 }

 vector<string> split(string str, char del)
 {
       vector<string> v;
       string temp = "";

       for(int i=0; i<str.size(); i++)
         {
             if(str[i] != del)
             {
                 temp += str[i];
             }
             else
             {
                 v.push_back(temp);
                 temp = "";
             }
         }

       v.push_back(temp);
       return v;
 }
```

Listing 9: Helper Functions

4.3 Pass 1

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

#include "Op_tab.h"
#include "functions.h"


using namespace std;

int main()
{
    int locctr;
    int start_add;
    int prog_len;
    int len;
    string line;
    vector<string> inst_fields;
    map <string, string> OpTab;
    map<string, string> SymTab;

    //map <string, string> SymTab;

    //Construction of optab
    OpTab = const_optab();

    ifstream fin("input.txt");
    ofstream fout("intermediate_file.txt");
    ofstream f2out("SymTab.txt");

    getline(fin, line);
    inst_fields = split(line,’ ’);
    string label = inst_fields[0];
    string opcode = inst_fields[1];
    string operand = inst_fields[2];

    if(opcode == "START")
    {
        start_add =  stoi(operand);
        locctr = hextodec(to_string(start_add)); //here locctr is already in hex
        fout << dectohex(locctr) << " " << line << endl;
        getline(fin, line);
        inst_fields = split(line,' ');
        if(inst_fields[0]!="")
        label = inst_fields[0];
        if(inst_fields[1]!="")
        opcode = inst_fields[1];
        if(inst_fields[2]!="")
        operand = inst_fields[2];
    }
    else
    {locctr = 0;}


    while(opcode != "END")
    {
        if (label != "")
        {
            if(SymTab[label]!="")
            {cout << locctr <<  "Error : Duplicate Symbol " << label << endl;}
            else
            {SymTab[label] = dectohex(locctr);}
        }

        fout << dectohex(locctr) << " " << line << endl;

        if(OpTab[opcode]!="")
            locctr = locctr + 3;
        else if (opcode == "WORD")
            locctr = locctr + 3;
        else if (opcode == "RESW")
            locctr = locctr + (3*stoi(operand));
        else if (opcode == "RESB")
            locctr = locctr + stoi(operand);
        else if (opcode == "BYTE")
        {
            if(operand[0] == 'C')
                len = (operand.length() - 3); // removing character {c,' , '}
            else
                len = (operand.length() - 3)/2;  //removing the characters {X ' '}

            locctr = locctr + len;
        }
        else
            cout << "Error : Invalid operation code" << endl;

        //fout << locctr << " " << line << endl;
        label = "";
        getline(fin, line);
        inst_fields = split(line,' ');
        int length = inst_fields.size();
        if(length-- &&  inst_fields[0]!="")
        label = inst_fields[0];
        if(length-- &&  inst_fields[1]!="")
        opcode = inst_fields[1];
        if(length-- &&  inst_fields[2]!="")
        operand = inst_fields[2];
    }
    fout  << " " << line << endl;

    prog_len = locctr - start_add + 1;

    if (!f2out.is_open()) {
        cerr << "Error: Unable to open SymTab.txt for writing." << endl;
        return 1;
    }

    for (const auto& pair : SymTab) {
        f2out << pair.first << " " << pair.second << endl;
    }

    fin.close();
    fout.close();
    f2out.close();

    return 0;
}
```

Listing 10: Pass 1 - Intermediate File Generator

4.4 Pass 2

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

#include "Op_tab.h"
#include "functions.h"

using namespace std;


int main()
{
    string line;
    vector<string> inst_fields;
    map <string, string> OpTab;
    map<string, string> SymTab;

    ifstream fin2("SymTab.txt");

    if (!fin2.is_open()) {
        cerr << "Error: Unable to open SymTab.txt for reading." << endl;
        return 1;
    }

    string line2;
    while (getline(fin2, line2)) {
        // Split each line into key and value
        size_t pos = line2.find(' ');
        if (pos != string::npos) {
            string key = line2.substr(0, pos);
            string value = line2.substr(pos + 1);
            SymTab[key] = value;
        } else {
            cerr << "Error: Invalid line format in SymTab.txt" << endl;
        }
    }

    ifstream fin("intermediate_file.txt");
    ofstream fout("obj_prog.txt");

    if (!fin.is_open() || !fout.is_open()) {
        cerr << "Error: Unable to open input or output file." << endl;
        return 1;
    }

    string locctr;
    string label;
    string opcode;
    string operand;

    // Store the position of the first line
    streampos firstLinePos = fin.tellg();

    getline(fin, line);
    inst_fields = split(line,' ');
    locctr = inst_fields[0];
    label = inst_fields[1];
    opcode = inst_fields[2];
    operand = inst_fields[3];

    int start_add = hextodec(locctr); //here locctr is already in hex

    string prevLine;
    string prevAddress;
    while (getline(fin, line)) {
        // Trim leading and trailing whitespace
        line.erase(0, line.find_first_not_of(" \t")); // trim leading whitespace
```

```
            line.erase(line.find_last_not_of(" \t") + 1); // trim trailing whitespace

            // Store the address from the previous line
            if (!prevLine.empty()) {
                size_t pos = prevLine.find_first_of(" \t");
                prevAddress = prevLine.substr(0, pos);
            }

            // Store the current line as the previous line
            prevLine = line;
        }

        // Go back to the first line
        fin.clear(); // Clear any error flags
        fin.seekg(firstLinePos);
        getline(fin, line); // reads first line which is not required

        int last_add = hextodec(prevAddress);

        int length =  last_add-start_add+1;
        string l = dectohex(length);
        l = padWithZeros(l, 6);

        if(opcode == "START")
        {
            fout << "H^" <<  padWithSpaces(label, 6) << "^" << padWithZeros(locctr, 6) << "^" <<  l;
        }


        OpTab = const_optab();
        getline(fin, line); // reads second line from which we need object code

            inst_fields = split(line, ' ');
            locctr = inst_fields[0];
            label = inst_fields[1];
            opcode = inst_fields[2];
            if (inst_fields.size() >= 4) {
            operand = inst_fields[3];
            } else {
            operand = ""; // Set operand to empty string if it's not present
            }

        int text_length;

        while(opcode != "END"){//text_length > 6){
            text_length = 60;
            fout << '\n' << "T^" << padWithZeros(locctr , 6);
            while (text_length > 0){//opcode != "END") {
                int opAddress;
                string objCode ="";
                if(OpTab.find(opcode)!=OpTab.end())
                {
                    if(operand[operand.length()-1] == 'X' && operand[operand.length()-2] == ',')
                    {
                    int l = hextodec(SymTab[operand.substr(0, operand.length()-2)]);
                    l = l+32768;
                    objCode = OpTab[opcode] + dectohex(l);
                    }
                    else
                    {
                        if(SymTab.find(operand) == SymTab.end())
                            objCode = OpTab[opcode] + "0000";
                        else
                            objCode = OpTab[opcode] + SymTab[operand];
                    }
                    if(text_length >= 6){
                        text_length -= 6;
                        fout << "^" << objCode;
                    }
                    else
                        break;
                }else if(opcode == "BYTE")
                {
                    if(operand[0] == 'C')
                    {
                        for(int i=2; i< operand.length()-1; i++)
                        {
                            char c = operand[i];
                            int asciiValue = c;
                            objCode += (dectohex(asciiValue));
                        }
                    }
                    else
                    {
                        for(int i=2; i< operand.length()-1; i++)
                        {
                            objCode += operand[i];
                        }
                    }
                    if(objCode.length() < text_length)
                    {
                        text_length -= objCode.length();
                        fout << "^" << objCode;
                    }
                    else{
                        break;
                    }
                }
                else if(opcode == "WORD")
                {
                    objCode = dectohex(stoi(operand));
                    objCode = padWithZeros(objCode, 6);
                    if(text_length >= 6){
                        text_length -= 6;
                        fout << "^" << objCode;
                    }
                }
                else if(opcode == "RESW" || opcode == "RESB")
                {
                    while(opcode == "RESW" || opcode == "RESB")
                    {
                        if (!getline(fin, line)) {
                        cerr << "Error: Unexpected end of file." << endl;
                        break;
                        }
                        inst_fields = split(line, ' ');
                        locctr = inst_fields[0];
                        label = inst_fields[1];
                        opcode = inst_fields[2];
                        if (inst_fields.size() >= 4) {
                        operand = inst_fields[3];
                        } else {
                        operand = ""; // Set operand to empty string if it's not present
                        }
                    }
                    break;
                }

                if (!getline(fin, line)) {
                    cerr << "Error: Unexpected end of file." << endl;
                    break;
                    }
                    inst_fields = split(line, ' ');
                    locctr = inst_fields[0];
                    label = inst_fields[1];
                    opcode = inst_fields[2];
                    if (inst_fields.size() >= 4) {
                    operand = inst_fields[3];
                    } else {
                    operand = ""; // Set operand to empty string if it's not present
                    }
            }
        }

        fout << '\n' << "E^" << padWithZeros(dectohex(start_add), 6) << endl;

        fin2.close();
        fin.close();
        fout.close();


     //for adding length of each record
        ifstream fin3("obj_prog.txt");
```

```
        ofstream fout2("temp.txt");
        getline(fin3, line); // reads the header record
        fout2 << line << endl;

        vector<string> v;
        int text_size;
        while(getline(fin3, line) && line[0] == 'T'){
            v = split(line, '^');
            text_size = 0;
            for(int i=2; i< v.size(); i++)
            {
                text_size += v[i].length();
            }
            line.insert(8, "^"+dectohex(text_size/2));
            fout2 << line << endl;
            v.erase(v.begin());
        }

        fout2 << line;

        fin3.close();
        fout2.close();

        remove("obj_prog.txt");
        rename("temp.txt", "obj_prog.txt");
        return 0;
    }
```

Listing 11: Pass 2 - Object Code Generator

5 Input and Output Examples

5.1 Input File (input.txt)

```
COPY START 1000
FIRST STL RETADR
CLOOP JSUB RDREC
 LDA LENGTH
 COMP ZERO
 JEQ ENDFIL
 JSUB WRREC
 J CLOOP
ENDFIL LDA EOF
 STA BUFFER
 LDA THREE
 STA LENGTH
 JSUB WRREC
 LDL RETADR
 RSUB
EOF BYTE C’EOF’
THREE WORD 3
ZERO WORD 0
RETADR RESW 1
LENGTH RESW 1
BUFFER RESB 4096
RDREC LDX ZERO
 LDA ZERO
RLOOP TD INPUT
 JEQ RLOOP
 RD INPUT
 COMP ZERO
 JEQ EXIT
 STCH BUFFER,X
 TIX MAXLEN
 JLT RLOOP
EXIT STX LENGTH
 RSUB
INPUT BYTE X’F1’
MAXLEN WORD 4096
WRREC LDX ZERO
WLOOP TD OUTPUT
 JEQ WLOOP
 LDCH BUFFER,X
 WD OUTPUT
 TIX LENGTH
 JLT WLOOP
 RSUB
OUTPUT BYTE X’05’
 END FIRST
```

5.2 Intermediate File (intermediate_file.txt)

```
1000 COPY START 1000
1000 FIRST STL RETADR
1003 CLOOP JSUB RDREC
1006  LDA LENGTH
1009  COMP ZERO
100C  JEQ ENDFIL
100F  JSUB WRREC
1012  J CLOOP
1015 ENDFIL LDA EOF
1018  STA BUFFER
101B  LDA THREE
101E  STA LENGTH
1021  JSUB WRREC
1024  LDL RETADR
1027  RSUB
102A EOF BYTE C’EOF’
102D THREE WORD 3
1030 ZERO WORD 0
1033 RETADR RESW 1
1036 LENGTH RESW 1
1039 BUFFER RESB 4096
2039 RDREC LDX ZERO
203C  LDA ZERO
203F RLOOP TD INPUT
2042  JEQ RLOOP
2045  RD INPUT
2048  COMP ZERO
204B  JEQ EXIT
204E  STCH BUFFER,X
2051  TIX MAXLEN
2054  JLT RLOOP
2057 EXIT STX LENGTH
205A  RSUB
205D INPUT BYTE X’F1’
205E MAXLEN WORD 4096
2061 WRREC LDX ZERO
2064 WLOOP TD OUTPUT
2067  JEQ WLOOP
```

```
206A  LDCH BUFFER,X
206D  WD OUTPUT
2070  TIX LENGTH
2073  JLT WLOOP
2076  RSUB
2079 OUTPUT BYTE X'05'
  END FIRST
```

5.3 Symbol Table (SymTab.txt)

```
BUFFER 1039
CLOOP 1003
ENDFIL 1015
EOF 102A
EXIT 2057
FIRST 1000
INPUT 205D
LENGTH 1036
MAXLEN 205E
OUTPUT 2079
RDREC 2039
RETADR 1033
RLOOP 203F
THREE 102D
WLOOP 2064
WRREC 2061
ZERO 1030
```

5.4 Object Code (obj_prog.txt)

```
H^COPY  ^001000^00107A
T^001000^1E^141033^482039^001036^281030^301015^482061^3C1003^00102A^0C1039^00102D
T^00101E^15^0C1036^482061^081033^4C0000^454F46^000003^000000
T^002039^1E^041030^001030^E0205D^30203F^D8205D^281030^302057^549039^2C205E^38203F
T^002057^1C^101036^4C0000^F1^001000^041030^E02079^302064^509039^DC2079^2C1036
T^002073^07^382064^4C0000^05
E^001000
```