Lab 7 LC-3 Assembler

郑悟强 PB22051082

2023.12.30

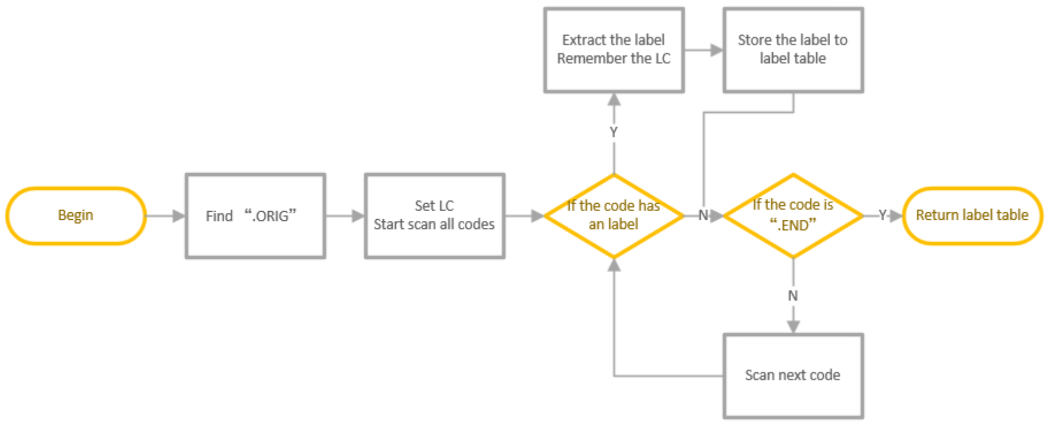
1 Purpose

This lab involves implementing a basic assembler for the LC-3 assembly language. Your task is to create a toy assembler, which will be tested using specific .asm files (not disclosed to you).
The purpose of this lab is to deepen your understanding of the assembly process. Therefore, you can disregard certain complexities typically associated with .asm files.

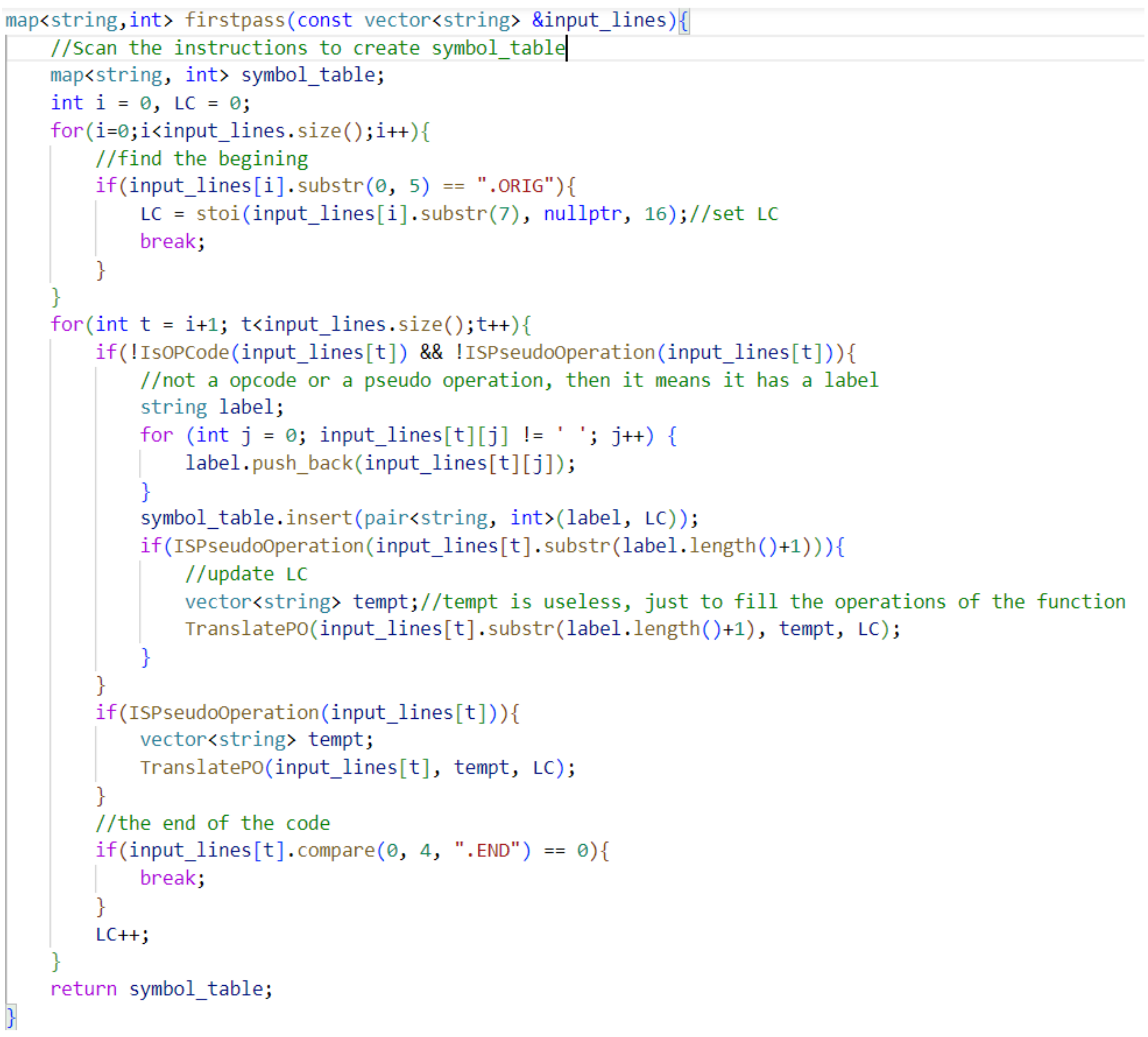
2 Principles

2.1 1st Pass

- **Purpose of the first scan:** Create the label table.
- **Flow chart:**



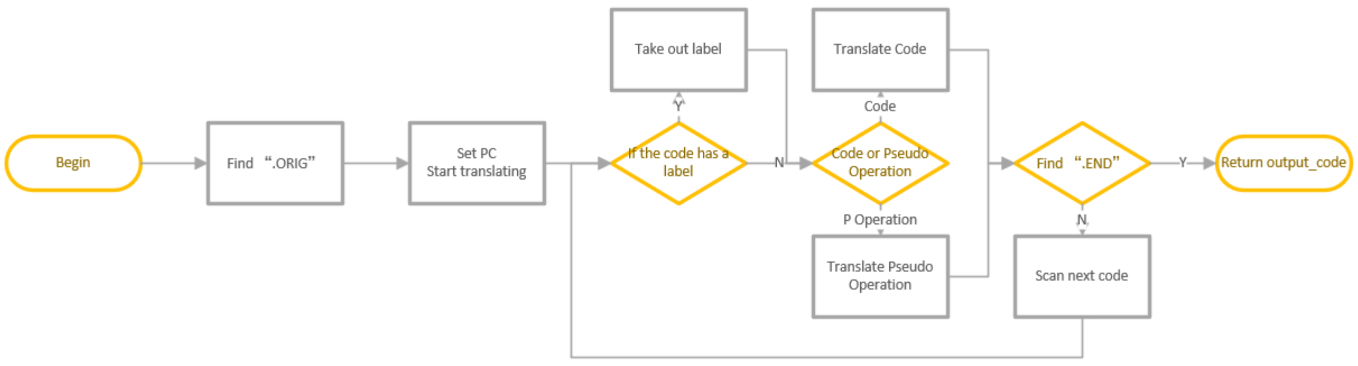
- **Code:**



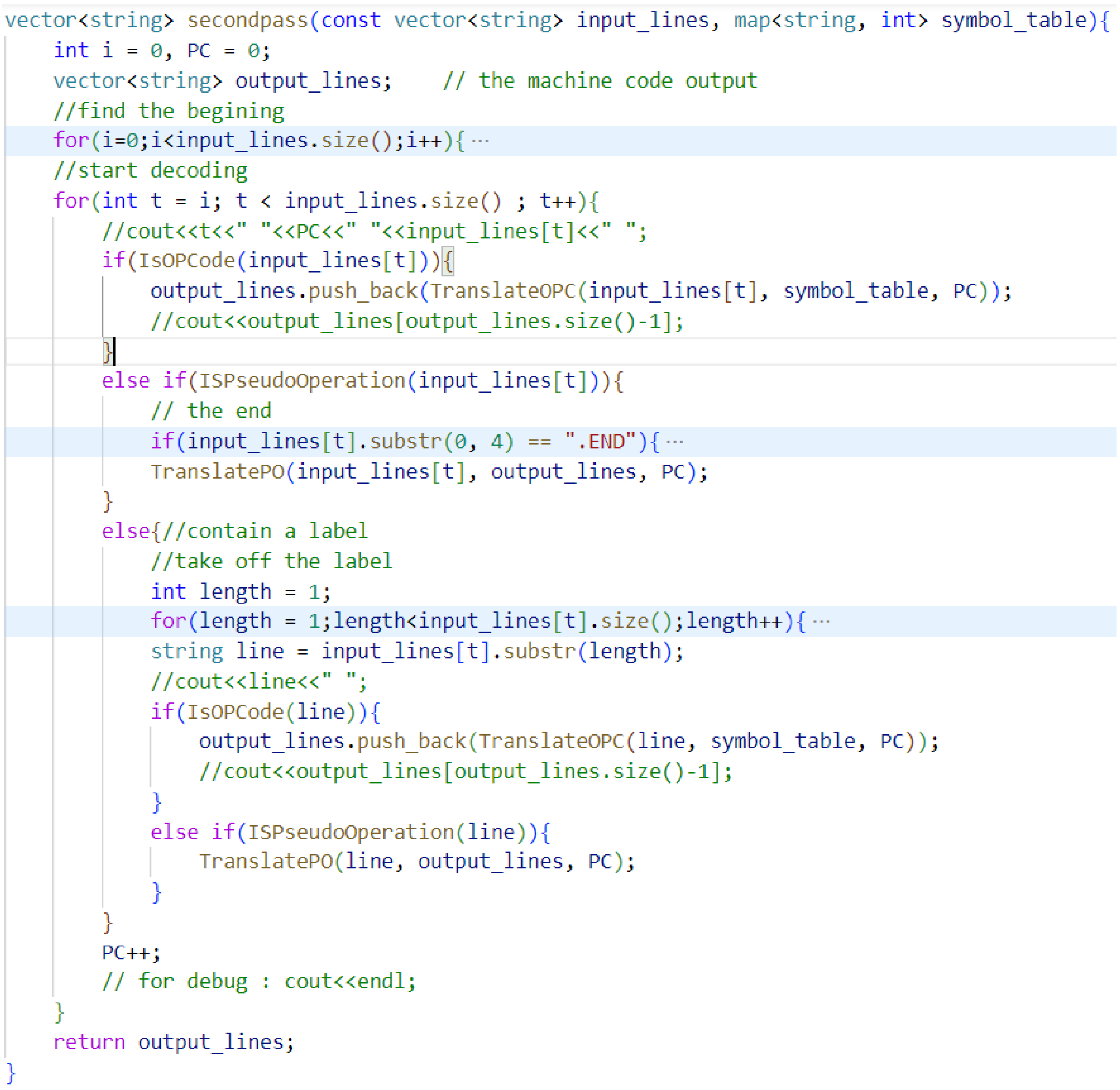
```
map<string,int> firstpass(const vector<string> &input_lines){
    //Scan the instructions to create symbol_table
    map<string, int> symbol_table;
    int i = 0, LC = 0;
    for(i=0;i<input_lines.size();i++){
        //find the begining
        if(input_lines[i].substr(0, 5) == ".ORIG"){
            LC = stoi(input_lines[i].substr(7), nullptr, 16);//set LC
            break;
        }
    }
    for(int t = i+1; t<input_lines.size();t++){
        if(!IsOPCode(input_lines[t]) && !ISPseudoOperation(input_lines[t])){
            //not a opcode or a pseudo operation, then it means it has a label
            string label;
            for (int j = 0; input_lines[t][j] != ' '; j++) {
                label.push_back(input_lines[t][j]);
            }
            symbol_table.insert(pair<string, int>(label, LC));
            if(ISPseudoOperation(input_lines[t].substr(label.length()+1))){
                //update LC
                vector<string> tempt;//tempt is useless, just to fill the operations of the function
                TranslatePO(input_lines[t].substr(label.length()+1), tempt, LC);
            }
        }
        if(ISPseudoOperation(input_lines[t])){
            vector<string> tempt;
            TranslatePO(input_lines[t], tempt, LC);
        }
        //the end of the code
        if(input_lines[t].compare(0, 4, ".END") == 0){
            break;
        }
        LC++;
    }
    return symbol_table;
}
```

2.2 2nd Pass

- **Purpose of the second scan:** Use the symbol table to translate the assembly code into machine code.
- **Flow chart:**



- **Code:** (*Fold part of the repeated operation*)



```
vector<string> secondpass(const vector<string> input_lines, map<string, int> symbol_table){
    int i = 0, PC = 0;
    vector<string> output_lines;    // the machine code output
    //find the begining
    for(i=0;i<input_lines.size();i++){⋯
    //start decoding
    for(int t = i; t < input_lines.size() ; t++){
        //cout<<t<<" "<<PC<<" "<<input_lines[t]<<" ";
        if(IsOPCode(input_lines[t])){
            output_lines.push_back(TranslateOPC(input_lines[t], symbol_table, PC));
            //cout<<output_lines[output_lines.size()-1];
        }
        else if(ISPseudoOperation(input_lines[t])){
            // the end
            if(input_lines[t].substr(0, 4) == ".END"){⋯
            TranslatePO(input_lines[t], output_lines, PC);
        }
        else{//contain a label
            //take off the label
            int length = 1;
            for(length = 1;length<input_lines[t].size();length++){⋯
            string line = input_lines[t].substr(length);
            //cout<<line<<" ";
            if(IsOPCode(line)){
                output_lines.push_back(TranslateOPC(line, symbol_table, PC));
                //cout<<output_lines[output_lines.size()-1];
            }
            else if(ISPseudoOperation(line)){
                TranslatePO(line, output_lines, PC);
            }
        }
        PC++;
        // for debug : cout<<endl;
    }
    return output_lines;
}
```

2.3 Some Other Important Functions

2.3.1 TranslateCode

- **Procedure:**

1. Get the opcode   2. Get the code   3. Fill in some parameters (such as the PCOffset and the number of registers)

Take LDI and LDR as examples

```
    //LDI and LEA and STI
    else if(output == "1010" || output == "1110" || output == "1011"){
        output.append(bitset<3>(line[5] - '0').to_string());
        output.append(GetPCOffset(line.substr(8), symbol_table, 9, PC));
    }
    //LDR and STR
    else if(output == "0110" || output == "0111"){
        output.append(bitset<3>(line[5] - '0').to_string());
        output.append(bitset<3>(line[9] - '0').to_string());
        if(line[12]=='#'){
            output.append(bitset<6>(stoi(line.substr(13))).to_string());
        }
        else if(line[12] == 'x' || line[12] == 'X'){
            output.append(bitset<6>(stoi(line.substr(13), nullptr, 16)).to_string());
        }
        else{
            output.append(bitset<6>(stoi(line.substr(12), nullptr, 2)).to_string());
        }
    }
```

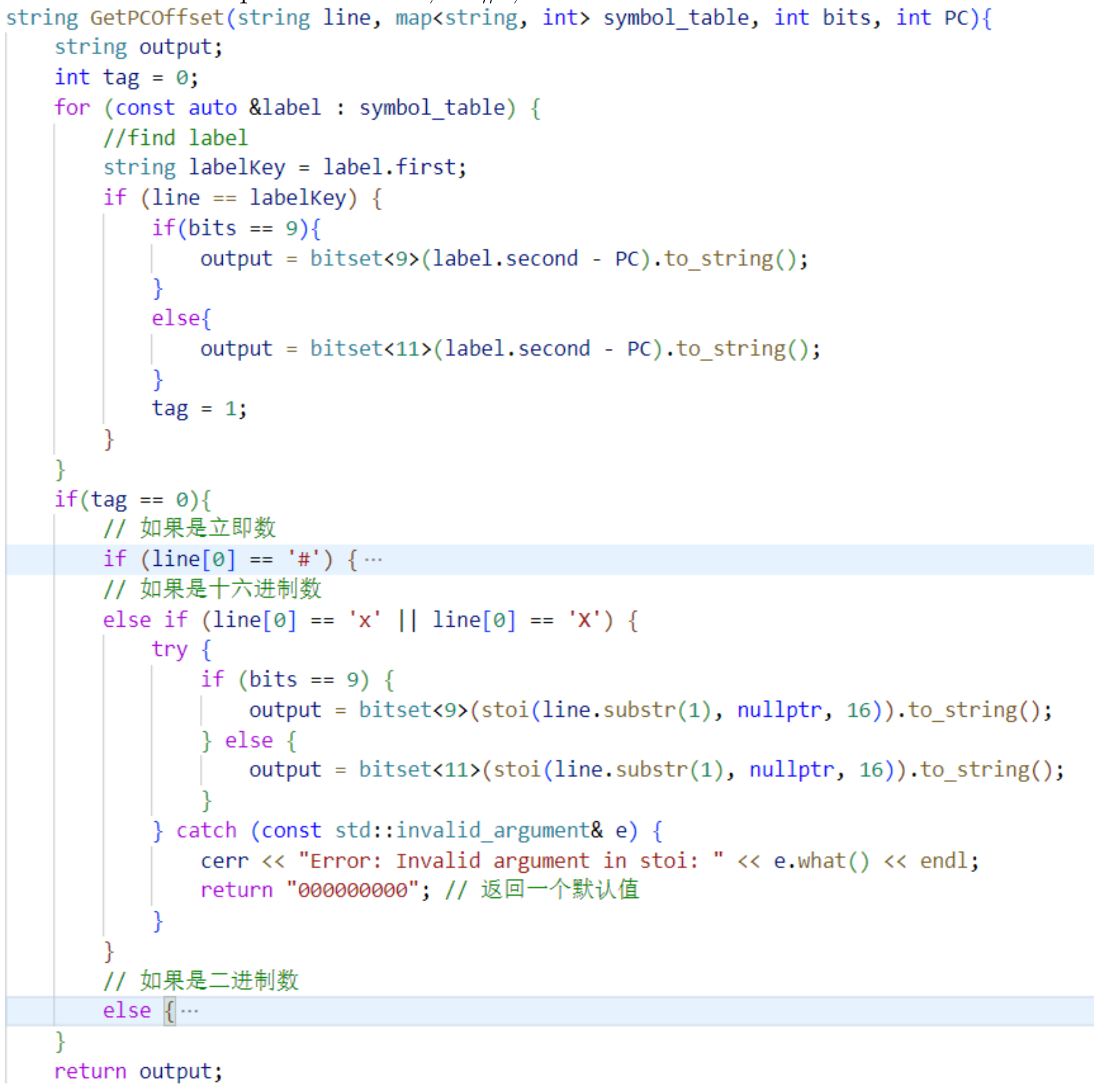
2.3.2 TranslatePseudoOperation

Take .STRINGZ as an example

```
//.STRINGZ
else if(line.substr(0, 8) == ".STRINGZ"){
    int start = 1, end = 1;
    for(int i = 8; i<line.size();i++){
        if(line[i] == '"'){
            if(start == 1){
                start = i;
            }
            else{
                end = i;
            }
        }
    }
    for(int j = start + 1; j<end;j++){
        output_lines.push_back(bitset<16>(int(line[j])).to_string());
        PC++;
    }
    output_lines.push_back("0000000000000000");
}
```

2.3.3 GetPCOffset

It contains three options: a label, a '#', a 'x'



```
string GetPCOffset(string line, map<string, int> symbol_table, int bits, int PC){
    string output;
    int tag = 0;
    for (const auto &label : symbol_table) {
        //find label
        string labelKey = label.first;
        if (line == labelKey) {
            if(bits == 9){
                output = bitset<9>(label.second - PC).to_string();
            }
            else{
                output = bitset<11>(label.second - PC).to_string();
            }
            tag = 1;
        }
    }
    if(tag == 0){
        // 如果是立即数
        if (line[0] == '#') {···
        // 如果是十六进制数
        else if (line[0] == 'x' || line[0] == 'X') {
            try {
                if (bits == 9) {
                    output = bitset<9>(stoi(line.substr(1), nullptr, 16)).to_string();
                } else {
                    output = bitset<11>(stoi(line.substr(1), nullptr, 16)).to_string();
                }
            } catch (const std::invalid_argument& e) {
                cerr << "Error: Invalid argument in stoi: " << e.what() << endl;
                return "000000000"; // 返回一个默认值
            }
        }
        // 如果是二进制数
        else {···
    }
    return output;
```

3 Procedure

3.1 Similar Opcodes or Similarity Between Opcode and Label

Examples: ST and STI   BR and BRANCH (a label)

Solution:

Judge STI first then ST (longer one first).   if(line(0, 3) == "STI") ...   else if(line(0, 2) == "ST")

Increase judgment   if(line(0, 2) == "BR" && (line[3] == ' ' || line.length() < 3))...

3.2 Codes with Multiple Forms

BR(NZP): Divide all the eight forms (*BR, BRN, BRZ, BRP, BRNZ, BRNP, BRZP, BRNZP*) into different codes.

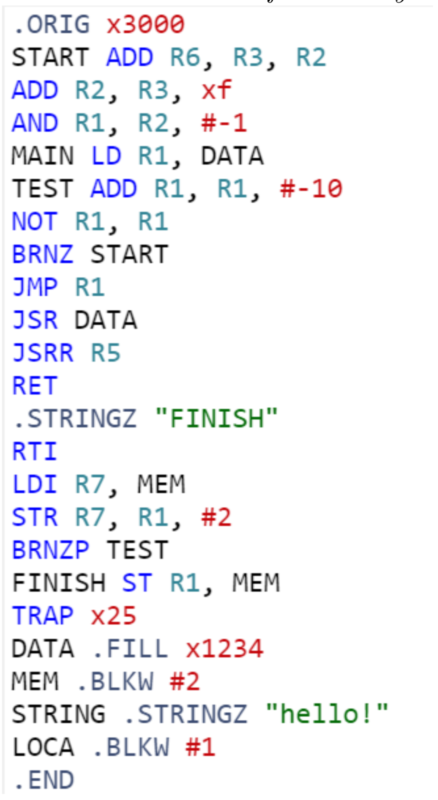
JSR(R): Divide them into different codes, and search JSR first (*the same as 3.1*)

AND, ADD: Detect if it has the third register, and then fill in different machine code if(line[12] == 'R')...   else...

4 Results

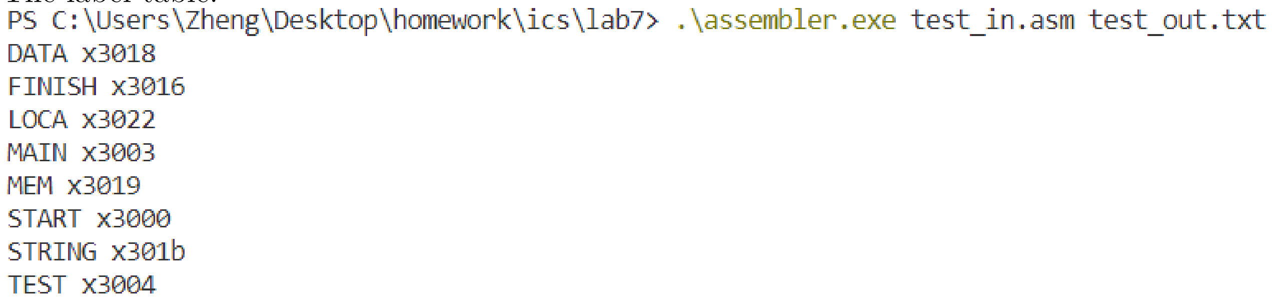
Let's take this assembly code as an example.

Explanation: This program contains no function; it is just a test for this assembler because this program contains all kinds of assembly codes



```
.ORIG x3000
START ADD R6, R3, R2
ADD R2, R3, xf
AND R1, R2, #-1
MAIN LD R1, DATA
TEST ADD R1, R1, #-10
NOT R1, R1
BRNZ START
JMP R1
JSR DATA
JSRR R5
RET
.STRINGZ "FINISH"
RTI
LDI R7, MEM
STR R7, R1, #2
BRNZP TEST
FINISH ST R1, MEM
TRAP x25
DATA .FILL x1234
MEM .BLKW #2
STRING .STRINGZ "hello!"
LOCA .BLKW #1
.END
```

The label table:



```
PS C:\Users\Zheng\Desktop\homework\ics\lab7> .\assembler.exe test_in.asm test_out.txt
DATA x3018
FINISH x3016
LOCA x3022
MAIN x3003
MEM x3019
START x3000
STRING x301b
TEST x3004
```

Then compare the output with the memory in LC3 tools; it is totally correct!