2022/11/04

System Programming

prepared by:

410921365 陳祁鎔 410921313 陳京生

Table of Contents

The problem description

Assignment Description:

Write an SIC assembler that reads an SIC assembly program, translates SIC statements into their machine code equivalents, and generates an object file.

Goals:

1. Get familiar with C++ programming language.
2. Learn to use I/O facilities and library functions provided by C++.
3. Get experience with system-level programming.
4. Get familiar with separate compilations, make utility, and C++ debugger.

HOW I WRITE THE PROGRAM

I declared two map STL for storing symbol_table and opcode_table that will be used in the second algorithm PASS2, besides, I declared an int variable for storing the program's length.

Then, I just called the first algorithm PASS1 to create the intermediate file which has their address before each instruction.

If the intermediate file is generated successfully, I would call the second algorithm PASS2 to generate the object file.

PASS1

1. Get the first instruction, separate it by space, and then calculate the start address.
2. Get each instruction row by row, and split it by space.
3. Classify each instruction by how many elements that contained in the instruction, so they will have different outputs.
4. Count their address by their mnemonic, which has different counting rules.
5. return to 2 until the end of the program.

PASS2

1. Get the first instruction with their address from the intermediate file, and output the first label name, start address, and length of the program to the object file.
2. Get each instruction row by row, and split it by space.
3. move each element to the corresponding position, such as label, mnemonic, or operand by their different type.
4. Classify each instruction by its mnemonic, and calculate its output, then count=count+its length (count is the length for T).
5. If the instruction is the last one, then cover the original ^00 I wrote by the final count because step 6 doesn't include the last situation. return to step 8
6. If count>=the limited size || the rule I set, then cover the original ^00 I write by the final count.
7. return to step 2
8. output "E^" and the start address as the end of the object fifle.

```
map<string, int> sym;
map<string, string> op_tab = {
    {"ADD", "18"},
    {"AND", "40"},
    {"COMP", "28"},
    {"DIV", "24"},
    {"J", "3C"},
    {"JEQ", "30"},
    {"JGT", "34"},
    {"JLT", "38"},
    {"JSUB", "48"},
    {"LDA", "00"},
    {"LDCH", "50"},
    {"LDL", "08"},
    {"LDX", "04"},
    {"MUL", "20"},
    {"OR", "44"},
    {"RD", "D8"},
    {"RSUB", "4C"},
    {"STA", "0C"},
    {"STCH", "54"},
    {"STL", "14"},
    {"STX", "10"},
    {"SUB", "1C"},
    {"TD", "E0"},
    {"TIX", "2C"},
    {"WD", "DC"}};
int program_length;
```

```
void PASS1()
{
    ifstream program;
    ofstream intermediate;
    string instruction, space_delimiter = " ", instr[3];
    vector<string> sym_exist;
    sym_exist.clear();
    int pos = 0, parameter = 0, locctr = 0X0, start = 0X0, count = 0X0;
    program.open("SIC_lab.txt", ios::in);
    intermediate.open("intermediate.txt", ios::out);
    if (!program.is_open())
    {
        cout << "File is missing." << endl;
        return;
    }
    getline(program, instruction);
    while ((pos = instruction.find(space_delimiter)) != string::npos)
    {
        instr[parameter] = instruction.substr(0, pos);
        instruction.erase(0, pos + space_delimiter.length());
        parameter++;
    }
    instr[parameter] = instruction.c_str();
    parameter = 0;
    if (instr[1] == "START")
    {
        locctr = stoi(instr[2]);
        while (locctr > 0)
        {
            start = start + (locctr % 10) * pow(16, count);
            locctr = locctr / 10;
            count++;
        }
        locctr = start;
        intermediate << hex << uppercase << start << " " << instr[0] << " " << instr[1] << " " << instr[2];
    }
    else
        locctr = 0X0;
```

```
while (getline(program, instruction))
{
    if (instruction[0] == ';' || instruction[0] == '.')
        continue;
    while ((pos = instruction.find(space_delimiter)) != string::npos)
    {
        instr[parameter] = instruction.substr(0, pos);
        instruction.erase(0, pos + space_delimiter.length());
        parameter++;
    }
    instr[parameter] = instruction.c_str();
    if (instr[0] == "END")
        break;
    if (instr[0][0] != ';' && instr[0][0] != '.') // not comment
    {
        if (parameter == 0)
        {
            instr[1] = instr[0];
            intermediate << endl
                         << hex << uppercase << setw(4) << setfill('0') << locctr << " " << instr[1];
        }
        if (parameter == 1)
        {
            instr[2] = instr[1];
            instr[1] = instr[0];
            intermediate << endl
                         << hex << uppercase << setw(4) << setfill('0') << locctr << " " << instr[1] << " " << instr[2];
        }
        if (parameter == 2)
        {
            vector<string>::iterator it = find(sym_exist.begin(), sym_exist.end(), instr[0]);
            if (it != sym_exist.end())
            {
                cout << "Duplicate label found: " << instr[0] << endl;
                return;
            }
            else
            {
                sym_exist.push_back(instr[0]);
                sym.insert(pair<string, int>(instr[0], locctr));
                intermediate << endl
                             << hex << uppercase << locctr << " " << instr[0] << " " << instr[1] << " " << instr[2];
            }
        }
```

```
            map<string, string>::iterator it = op_tab.find(instr[1]);
            if (!(it == op_tab.end()))
                locctr += 3;
            else if (instr[1] == "WORD" || instr[1] == "word")
                locctr += 3;
            else if (instr[1] == "RESW" || instr[1] == "resw")
                locctr += 3 * stoi(instr[2]);
            else if (instr[1] == "RESB" || instr[1] == "resb")
                locctr += stoi(instr[2]);
            else if (instr[1] == "BYTE" || instr[1] == "byte")
            {
                int len = instr[2].length();
                if (instr[2][0] != 'C' && instr[2][0] != 'X')
                    locctr += 1;
                else if (instr[1] == "BYTE" && instr[2][0] == 'C')
                    locctr += (len - 3);
                else if (instr[1] == "BYTE" && instr[2][0] == 'X')
                {
                    if ((len - 3) % 2 != 0)
                        locctr += (len - 3) / 2 + 1;
                    else
                        locctr += (len - 3) / 2;
                }
            }
            else
            {
                cout << "Cannot find " << instr[1] << " in OPTAB" << endl
                     << "Exiting ..." << endl;
                return;
            }
        }
        parameter = 0;
    }
    cout << "Intermediate file are generated" << endl;
    program_length = locctr - start;
    program.close();
    intermediate.close();
    return;
}
```

```
void PASS2()
{
    ifstream intermediate;
    ofstream result;
    stringstream tmp;
    int locctr = 0X0, start = 0X0, pos = 0, parameter = 0, count = 0X0, recorded_length = 0X0, addr = 0X0, label_addr = 0X0, label_target = 0X0;
    long int count_begin, count_end;
    string instr[4], instruction, OP_code, space_delimiter = " ", last_instr;
    intermediate.open("intermediate.txt", ios::in);
    result.open("result.txt", ios::out | ios::trunc);
    if (!intermediate.is_open())
    {
        cout << "Intermediate file is missing." << endl;
        return;
    }
    intermediate >> instr[0] >> instr[1] >> instr[2] >> instr[3]; // loc, label, mn, op
    tmp << hex << uppercase << instr[0];
    tmp >> locctr;
    tmp.clear();
    if (instr[2] == "START")
    {
        tmp << hex << uppercase << instr[3];
        tmp >> start;
        result << "H^" << left << setw(6) << setfill(' ') << instr[1]
               << "^" << right << hex << uppercase << setw(6) << setfill('0') << start
               << "^" << right << hex << uppercase << setw(6) << setfill('0') << program_length;
        result << endl
               << "T^" << hex << uppercase << setw(6) << setfill('0') << start << "^00";
        count_begin = result.tellp();
    }
    getline(intermediate, instruction);
while (getline(intermediate, instruction))
{
    parameter = 0;
    while ((pos = instruction.find(space_delimiter)) != string::npos)
    {
        instr[parameter] = instruction.substr(0, pos);
        instruction.erase(0, pos + space_delimiter.length());
        parameter++;
    }
    instr[parameter] = instruction.c_str();
    tmp << instr[0];
    tmp >> locctr;
    tmp.clear();
    if (parameter == 1)
        instr[2] = instr[1];
    else if (parameter == 2)
    {
        instr[3] = instr[2];
        instr[2] = instr[1];
    }
    if (count >= 0X37 || ((last_instr == "RESB" || last_instr == "RESW") && (instr[2] != "RESW" && instr[2] != "RESB")) || instr[2] == "END")
    {
        count_end = result.tellp();
        result.seekp(-(count_end - count_begin) - 3L, ios::cur);
        recorded_length = count / 0X2;
        result << "^" << hex << uppercase << setw(2) << setfill('0') << recorded_length;
        result.seekp(0L, ios::end);
        if (instr[2] == "END")
            break;
        addr = locctr;
        if (instr[2] != "RESW")
            result << endl
                   << "T^" << hex << uppercase << setw(6) << setfill('0') << addr << "^00";
        count_begin = result.tellp();
        count = 0X0;
    }
```

```
map<string, string>::iterator it = op_tab.find(instr[2]);
string opcode, value_of_X = "";
if (it != op_tab.end())
{
    opcode = op_tab[instr[2]];
    int k = instr[3].length();
    if (instr[3][k - 1] == 'X' && instr[3][k - 2] == ',')
    {
        instr[3] = instr[3].substr(0, k - 2);
        map<string, int>::iterator itt = sym.find(instr[3]);
        if (itt != sym.end())
        {
            label_addr = sym[instr[3]];
            label_target = label_addr + 0X8000;
        }
        result << "^" << setw(2) << opcode << hex << uppercase << setw(4) << setfill('0') << label_target;
        count = count + 0X6;
        if (intermediate.peek() == EOF)
        {
            count_end = result.tellp();
            result.seekp(-(count_end - count_begin) - 3L, ios::cur);
            recorded_length = count / 0X2;
            result << "^" << hex << uppercase << setw(2) << setfill('0') << recorded_length;
            result.seekp(0L, ios::end);
        }
        continue;
    }
       else if (instr[2] != "RSUB")
       {
           map<string, int>::iterator itt = sym.find(instr[3]);
           if (itt != sym.end())
           {
               label_addr = sym[instr[3]];
               label_target = label_addr;
           }
           cout << "opcode: " << opcode << " target address: " << hex << uppercase << label_target << endl;
           result << "^" << setw(2) << opcode << hex << uppercase << setw(4) << setfill('0') << label_target;
           count = count + 0X6;
           last_instr = instr[2];
           if (intermediate.peek() == EOF)
           {
               count_end = result.tellp();
               result.seekp(-(count_end - count_begin) - 3L, ios::cur);
               recorded_length = count / 0X2;
               result << "^" << hex << uppercase << setw(2) << setfill('0') << recorded_length;
               result.seekp(0L, ios::end);
           }
           continue;
       }
       else if (instr[2] == "RSUB")
       {
           result << "^" << opcode << "0000";
           count = count + 0X6;
           last_instr = instr[2];
           if (intermediate.peek() == EOF)
           {
               count_end = result.tellp();
               result.seekp(-(count_end - count_begin) - 3L, ios::cur);
               recorded_length = count / 0X2;
               result << "^" << hex << uppercase << setw(2) << setfill('0') << recorded_length;
               result.seekp(0L, ios::end);
           }
           continue;
       }
   }
```

Snapshots

```
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
.
. SUBROUTINE TO READ RECORD INTO BUFFER
.
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
.
. SUBROUTINE TO WRITE RECORD FROM BUFFER
.
WRREC LDX ZERO
WLOOP TD OUTPUT
JEQ WLOOP
LDCH BUFFER,X
WD OUTPUT
TIX LENGTH
JLT WLOOP
RSUB
OUTPUT BYTE X'05'
END FIRST
```

Snapshots

```
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'
```

```
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
```

Discussion

It is really interesting that we can learn how to implement an assembler ourselves. I thought it was not that complicated when I was doing the practice during the class. However, I was wrong; we needed to set up several rules for programs to classify each instruction with different calculating rules. It is quite challenging for me to consider all the situations.

For the first algorithm PASS1, it is not that troublesome for me because the main thing we need to do is calculate the address for each instruction.

The second algorithm PASS2, it's really complicated because we need to move the pointer in the object file in order to renew the length of the T area, and there are a lot of rules we need to obey, but it has several exceptions needed to be handled. Fortunately, I finally fixed all the problems, and I learned a lot from this assignment.

Thank you for your reading!!

Resource

- 不認識這些圖標，還敢稱自己是碼農？ - 每日頭條 (kknews.cc)