系統程式 Final Project - SIC Assembler

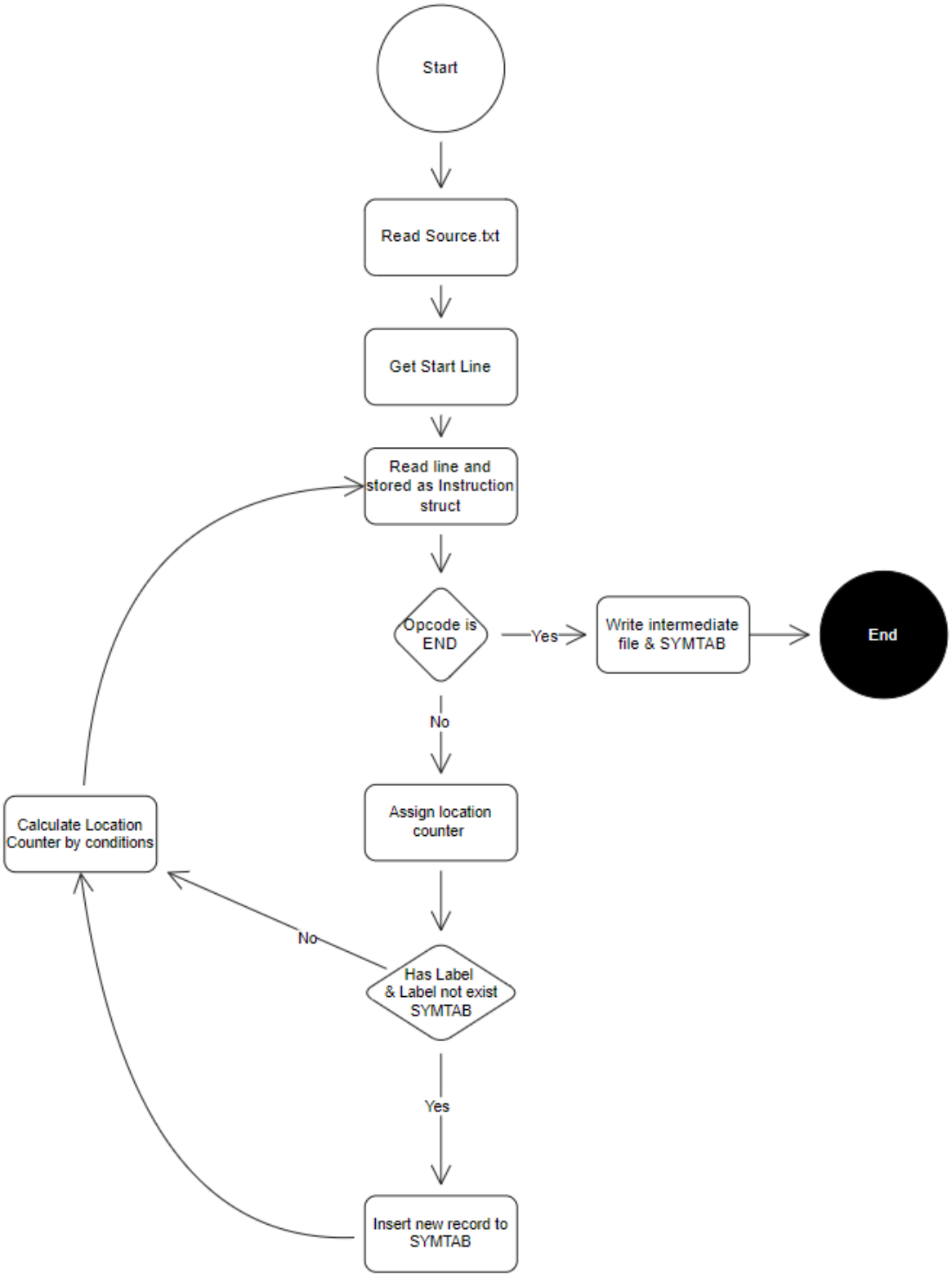
5110056005 劉祈廷

1. 執行環境

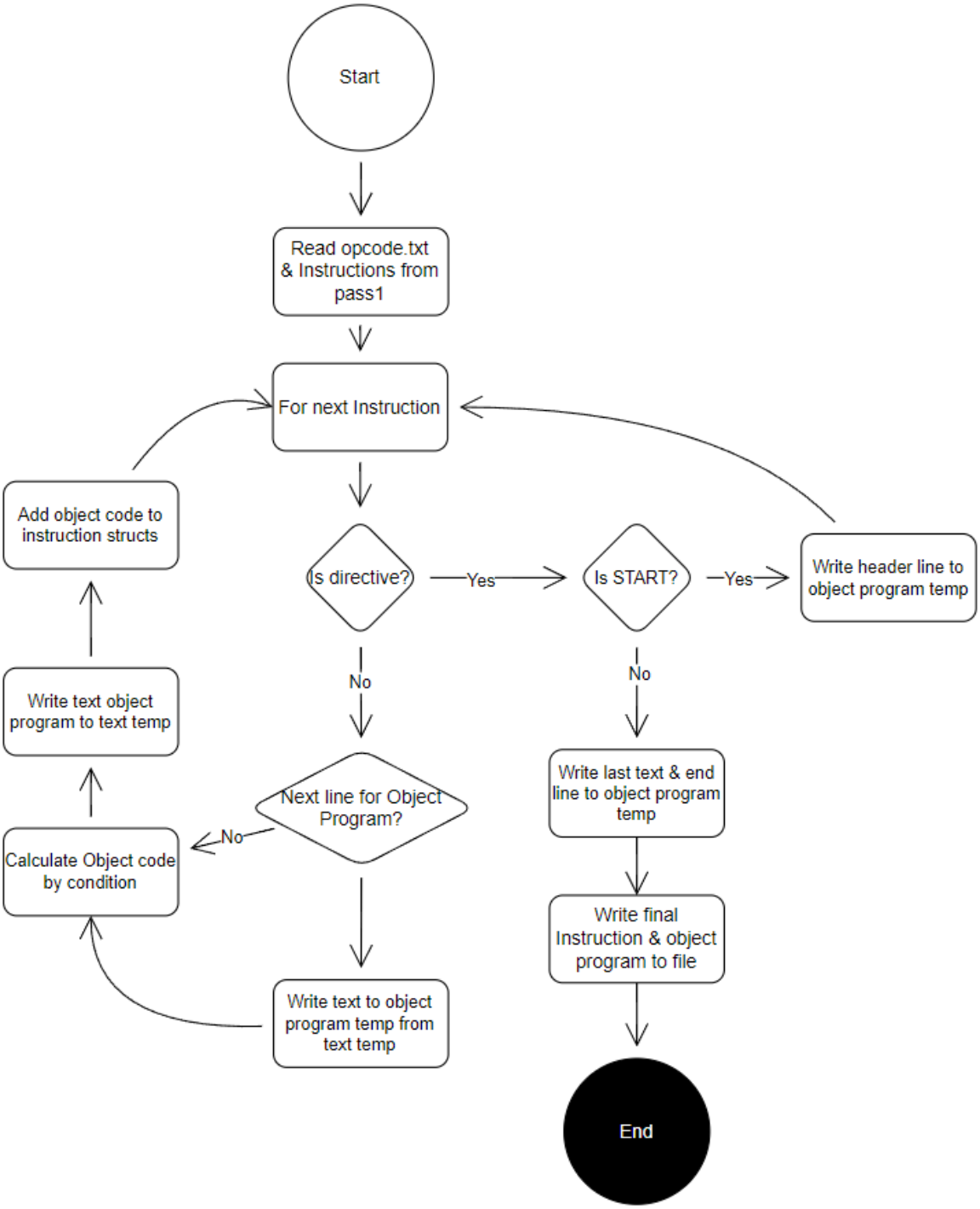
- Compiler: TDM-GCC 9.2.0
- Standard: C++11 以上
- 執行時請將 source.txt, opcode.txt 與 .cpp 放在同一階層
- Output:
    - pass1-instructions_list.txt
    - pass1-symbol_table.txt
    - pass2-instructions_list.txt
    - pass2-object_program.txt

2. 程式流程

Pass1


Start
Read Source.txt
Get Start Line
Read line and stored as Instruction struct
Opcode is END
Yes
Write intermediate file & SYMTAB
End
No
Assign location counter
Has Label & Label not exist SYMTAB
No
Calculate Location Counter by conditions
Yes
Insert new record to SYMTAB

Pass 2


Start
Read opcode.txt & Instructions from pass1
For next Instruction
Is directive?
Yes
Is START?
Yes
Write header line to object program temp
No
No
Add object code to instruction structs
Write text object program to text temp
Next line for Object Program?
No
Calculate Object code by condition
Write text to object program temp from text temp
Write last text & end line to object program temp
Write final Instruction & object program to file
End

3. 程式重點

Pass1

1. 讀取 source 並將 header line 保存至 struct 中，初始化 location counter

```
string row;
ifstream file_source(source_path, ifstream::in);

// 紀錄start
getline(file_source, row);
queue<string> row_values;
Split(row, '\t', row_values);

instructs[0x0].label = row_values.front();
row_values.pop();
instructs[0x0].opcode = row_values.front();
row_values.pop();
instructs[0x0].operand = row_values.front();
row_values.pop();
instructs[0x0].is_directive = 1;

// 取得起始address in hex
string s = "0x" + instructs[0x0].operand;
location_counter = std::stoul(s, nullptr, 16);
instructs[0x0].loc_count = location_counter;
```

2. 輪循每一行指令，先將所有 instruction 轉為 struct 方便後續處理

```
// pass1 > 計算location counter & Symbol Table
while (getline(file_source, row))
{
    row_values = queue<string>(); // reset queue
    Split(row, '\t', row_values);

    instructs[location_counter].label = row_values.front();
    row_values.pop();
    instructs[location_counter].opcode = row_values.front();
    row_values.pop();

    // 避免沒有 operand 的指令造成錯誤, ex RSUB
    if (!row_values.empty())
    {
        instructs[location_counter].operand = row_values.front();
        row_values.pop();
    }
```

3. 遇到 END 就結束 pass1，否則將當前的 location counter 記錄到 struct 中

```
// 遇到END 就跳過，其他則紀錄location_counter
if (instructs[location_counter].opcode == "END")
{
    instructs[location_counter].is_directive = 1;
    break;
}
else
    instructs[location_counter].loc_count = location_counter;
```

4. 判斷有 Label 且 Label 不存在於 SYMTAB 時寫入 symbol table

```
// 如果有label且Symbol table 尚未有此record就寫到 Symbol table
if (!instructs[location_counter].label.empty() && !symtab.count(instructs[location_counter].label))
{
    symtab[instructs[location_counter].label] = location_counter;
}
```

5. 計算下一個指令的 location counter

```
// 計算下一指令的location counter
if (instructs[location_counter].opcode == "RESB")
{
    // RESB 將operand 轉成數字後加入location
    int reserve_bytes = stoi(instructs[location_counter].operand);
    location_counter += reserve_bytes;
}
else if (instructs[location_counter].opcode == "BYTE")
{
    // BYTE 有三種狀況
    if (instructs[location_counter].operand[0] == 'C')
    {
        // Start with C (C'EOF')
        location_counter += instructs[location_counter].operand.length() - 3;
    }
    else if (instructs[location_counter].operand[0] == 'X')
    {
        // Start with X (X'05')
        location_counter += (instructs[location_counter].operand.length() - 3) / 2;
    }
    else
        location_counter += 1;
```

6. 最後寫入檔案，結束 pass1，pass1 輸出結果如下

- pass1-instructions_list.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
```

- pass1-symbol_table.txt (我的 SYMTAB 是用 Label 字母順序排序)

```
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

```

Pass 2

1. 讀取 opcode.txt

```
// 讀取 opcode table，先將opcode table建立map
ifstream file_opcode(opcode_path, ifstream::in);
while (getline(file_opcode, row))
{
    row_values = queue<string>(); // reset queue
    Split(row, ' ', row_values);

    string opcode = row_values.front();
    row_values.pop();
    optab[opcode] = row_values.front();
    row_values.pop();
}
```

2. 輪循每一個指令，先判斷是不是 directive

```
    iter = instructs.begin();
    while (iter != instructs.end())
    {
        if (iter->second.is_directive != 1)
        {
```

3. 如不是 directive，先判斷 object program 是不是需要換行了，是的話將暫存的 text 加入 object program 的暫存中，待後續寫入檔案

```
if (inst_count == 10 || next_line_flag)
{
    // object program 一行滿了，寫入並換行
    ostringstream ss;
    ss << "T " << setfill('0') << setw(6) << uppercase << hex << line_start << " " << ToHexString(
    object_program[line++] = ss.str();

    // reset 所有flag
    inst_count = 0;
    next_line_flag = 0;
    line_start = iter->second.loc_count;
    text = "";
    add_line_length = 1;
}
```

4. 後面依據各 operator 的特性轉換成 object code，將 object code 存回 struct 中

```
if (iter->second.opcode == "BYTE")
{
    // BYTE系列另外處理
    if (iter->second.operand[0] == 'C')
    {
        // C: 將字串轉成ASCII 16進位
        string real_operand = iter->second.operand.substr(2, iter->second.operand.size() - 3);
        for (unsigned i = 0; i < real_operand.size(); i++)
        {
            iter->second.obj_code += ToHexString(int(real_operand[i]), 2);
        }
    }
    else if (iter->second.operand[0] == 'X')
    {
        // X: 直接輸出文字
        iter->second.obj_code = iter->second.operand.substr(2, iter->second.operand.size() - 3);
    }
}
else if (iter->second.opcode == "RESW" || iter->second.opcode == "RESB")
{
    // RES 系列不產生 object code，所以不計算line length
    add_line_length = 0;

    if (iter->second.opcode == "RESB")
        // 遇到 RESB 要換行
        next_line_flag = 1;
}
```

```
else if (iter->second.opcode == "WORD")
{
    // WORD 把後面數值直接轉16進位並補滿6位
    int number = stoi(iter->second.operand);
    iter->second.obj_code = ToHexString(number, 6);
}
else if (iter->second.opcode == "RSUB")
{
    // RSUB 輸出OP code 後面補滿0
    iter->second.obj_code = optab[iter->second.opcode] + "0000";
}
else if (iter->second.operand.find(",X") != std::string::npos)
{
    // Index addressing, add 0x8000讓 X bit == 1
    string real_operand = iter->second.operand.substr(0, iter->second.operand.size() - 2);
    iter->second.obj_code = optab[iter->second.opcode] + ToHexString(symtab[real_operand] + 0x8000, 4);
}
else
{
    iter->second.obj_code = optab[iter->second.opcode] + ToHexString(symtab[iter->second.operand], 4);
}
```

5. 如有產生 object code，將 object code 寫入 text 暫存，待換行時整行一同寫入 object program 的暫存

```
//是有產生 object code的指令，將其記錄至暫存
if (iter->second.obj_code.size() > 0)
{
    text += " " + iter->second.obj_code;
    inst_count++;
}
```

6. 如果指令是 directive，START 的情況直接 header line 寫入 object program 的暫存，END 的情況要連著最後一行 Text line 把 End line 寫入 object program 暫存

```
if (iter->second.opcode == "START")
{
    // 寫入 H line
    ostringstream ss;
    ss << "H " << setw(6) << std::left << iter->second.label << " " << ToHexString(iter->second.loc_count,
    object_program[line++] += ss.str();
    line_start = iter->second.loc_count;
}
else if (iter->second.opcode == "END")
{
    // 紀錄最後一組text
    ostringstream ss1;
    ss1 << "T " << setfill('0') << setw(6) << uppercase << hex << line_start << " " << ToHexString(locatio
    object_program[line++] = ss1.str();

    // 寫入 E line
    ostringstream ss2;
    ss2 << "E " << ToHexString(instructs[0x0].loc_count, 6);
    object_program[line] += ss2.str();
}
```

7. 最後將 instruction struct 和 object program temp 寫入檔案，結果如下

- pass2-instructions_list.txt

```
1000       COPY    START      1000
1000      FIRST      STL    RETADR    141033
1003      CLOOP     JSUB     RDREC    482039
1006                 LDA    LENGTH    001036
1009                COMP      ZERO    281030
100C                 JEQ    ENDFIL    301015
100F                JSUB     WRREC    482061
1012                   J     CLOOP    3C1003
1015     ENDFIL      LDA       EOF    00102A
1018                 STA    BUFFER    0C1039
101B                 LDA     THREE    00102D
101E                 STA    LENGTH    0C1036
1021                JSUB     WRREC    482061
1024                 LDL    RETADR    081033
1027                RSUB              4C0000
102A       EOF      BYTE    C'EOF'    454F46
102D     THREE      WORD         3    000003
1030      ZERO      WORD         0    000000
1033    RETADR      RESW         1
1036    LENGTH      RESW         1
1039    BUFFER      RESB      4096
2039     RDREC       LDX      ZERO    041030
203C                 LDA      ZERO    001030
203F     RLOOP        TD     INPUT    E0205D
2042                 JEQ     RLOOP    30203F
2045                  RD     INPUT    D8205D
2048                COMP      ZERO    281030
204B                 JEQ      EXIT    302057
204E                STCH  BUFFER,X    549039
2051                 TIX    MAXLEN    2C205E
2054                 JLT     RLOOP    38203F
2057      EXIT       STX    LENGTH    101036
205A                RSUB              4C0000
205D     INPUT      BYTE     X'F1'        F1
205E    MAXLEN      WORD      4096    001000
2061     WRREC       LDX      ZERO    041030
2064     WLOOP        TD    OUTPUT    E02079
2067                 JEQ     WLOOP    302064
206A                LDCH  BUFFER,X    509039
206D                  WD    OUTPUT    DC2079
2070                 TIX    LENGTH    2C1036
2073                 JLT     WLOOP    382064
2076                RSUB              4C0000
2079    OUTPUT      BYTE     X'05'        05
                     END     FIRST
```

- pass2-object_program.txt

```
H COPY   001000 107A
T 001000 1E 141033 482039 001036 281030 301015 482061 3C1003 00102A 0C1039 00102D
T 00101E 15 0C1036 482061 081033 4C0000 454F46 000003 000000
T 001036 FFFFFFFD
T 001039 FFFFFFFA
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

```

8. 結束