Assembly Language Final Project

物理四　　4107054018　　陳昱銓

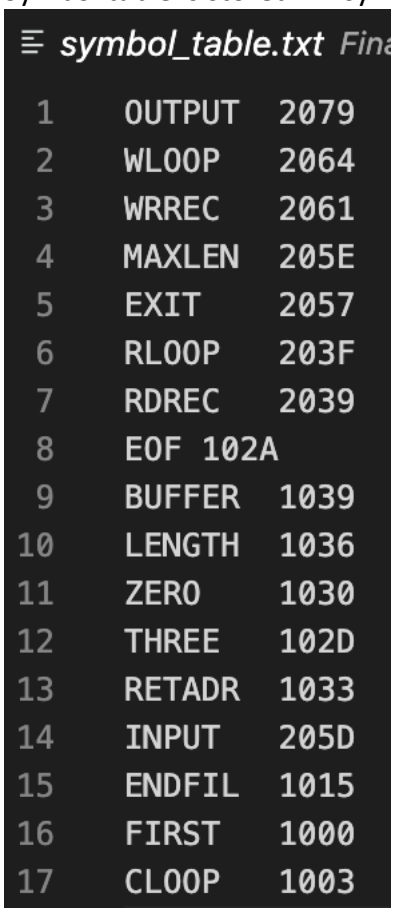
Pass 1

1. Location counter and source statement is stored in “intermediate_file.txt”.

intermediate_file.txt — Final Project • intermediate_file.txt

```
1   1000    COPY    START   1000
2   1000    FIRST   STL RETADR
3   1003    CLOOP   JSUB    RDREC
4   1006        LDA LENGTH
5   1009        COMP    ZERO
6   100C        JEQ ENDFIL
7   100F        JSUB    WRREC
8   1012        J   CLOOP
9   1015    ENDFIL  LDA EOF
10  1018        STA BUFFER
11  101B        LDA THREE
12  101E        STA LENGTH
13  1021        JSUB    WRREC
14  1024        LDL RETADR
15  1027        RSUB
16  102A    EOF BYTE    C'EOF'
17  102D    THREE   WORD    3
18  1030    ZERO    WORD    0
19  1033    RETADR  RESW    1
20  1036    LENGTH  RESW    1
21  1039    BUFFER  RESB    4096
22  2039    RDREC   LDX ZERO
23  203C        LDA ZERO
24  203F    RLOOP   TD  INPUT
25  2042        JEQ RLOOP
26  2045        RD  INPUT
27  2048        COMP    ZERO
28  204B        JEQ EXIT
29  204E        STCH    BUFFER,X
30  2051        TIX MAXLEN
31  2054        JLT RLOOP
32  2057    EXIT    STX LENGTH
33  205A        RSUB
34  205D    INPUT   BYTE    X'F1'
35  205E    MAXLEN  WORD    4096
36  2061    WRREC   LDX ZERO
37  2064    WLOOP   TD  OUTPUT
38  2067        JEQ WLOOP
39  206A        LDCH    BUFFER,X
40  206D        WD  OUTPUT
41  2070        TIX LENGTH
42  2073        JLT WLOOP
43  2076        RSUB
44  2079    OUTPUT  BYTE    X'05'
45      END FIRST
```

2. Symbol table is stored in “symbol_table.txt”.



Pass 2

1. Source program with object code is stored in “source_program_with_object_code.txt”.

source_program_with_object_code.txt Final Project

```
1   1000    COPY    START   1000
2   1000    FIRST   STL RETADR  141033
3   1003    CLOOP   JSUB    RDREC   482039
4   1006        LDA LENGTH  001036
5   1009        COMP    ZERO    281030
6   100C        JEQ ENDFIL  301015
7   100F        JSUB    WRREC   482061
8   1012        J   CLOOP   3C1003
9   1015    ENDFIL  LDA EOF 00102A
10  1018        STA BUFFER  0C1039
11  101B        LDA THREE   00102D
12  101E        STA LENGTH  0C1036
13  1021        JSUB    WRREC   482061
14  1024        LDL RETADR  081033
15  1027        RSUB        4C0000
16  102A    EOF BYTE    C'EOF'  454F46
17  102D    THREE   WORD    3   000003
18  1030    ZERO    WORD    0   000000
19  1033    RETADR  RESW    1
20  1036    LENGTH  RESW    1
21  1039    BUFFER  RESB    4096
22  2039    RDREC   LDX ZERO    041030
23  203C        LDA ZERO    001030
24  203F    RLOOP   TD  INPUT   E0205D
25  2042        JEQ RLOOP   30203F
26  2045        RD  INPUT   D8205D
27  2048        COMP    ZERO    281030
28  204B        JEQ EXIT    302057
29  204E        STCH    BUFFER,X    549039
30  2051        TIX MAXLEN  2C205E
31  2054        JLT RLOOP   38203F
32  2057    EXIT    STX LENGTH  101036
33  205A        RSUB        4C0000
34  205D    INPUT   BYTE    X'F1'   F1
35  205E    MAXLEN  WORD    4096    001000
36  2061    WRREC   LDX ZERO    041030
37  2064    WLOOP   TD  OUTPUT  E02079
38  2067        JEQ WLOOP   302064
39  206A        LDCH    BUFFER,X    509039
40  206D        WD  OUTPUT  DC2079
41  2070        TIX LENGTH  2C1036
42  2073        JLT WLOOP   382064
43  2076        RSUB        4C0000
44  2079    OUTPUT  BYTE    X'05'   05
45          END FIRST
```

2. Object program stored in “object_program.txt”.

object_program.txt *Final Project • object_program.txt*

```
1    HCOPY   00100000107A
2    T0010001E141033482039001036281030301015482061 3C100300102A0C103900102D
3    T00101E1E0C10364820610810334C0000454F46000003000000041030001030E0205D
4    T0020421C30203FD8205D28103030205754903 92C205E38203F1010364C0000F1
5    T00205E1C001000041030E02079302064509039DC20792C10363820644C000005
6    E001000
```

Complementary Tools

Class OperationCodeTable:

Description:

The class OperationCodeTable is used to get its opcode and check if the mnemonic exists.

```
13  class OperationCodeTable {
14      private:
15          unordered_map<string, string> opcodeTable;
16
17      public:
18          void addOpCode(string mnemonic, string code) {
19              opcodeTable.insert(pair<string, string>(mnemonic, code));
20          }
21
22          string getOpCode(string mnemonic) {
23              return opcodeTable[mnemonic];
24          }
25
26          int contains(string mnemonic) {
27              return opcodeTable.count(mnemonic);
28          }
29  };
```

Class SymbolTable:

Description:

The class SymbolTable is used to get the address where the label is defined and to check if the symbol exists.

```
32  class SymbolTable {
33      public:
34          unordered_map<string, int> symbolTable;
35
36          void addSymAddr(string symbol, int addr) {
37              symbolTable.insert(pair<string, int>(symbol, addr));
38          }
39
40          int getSymAddr(string symbol) {
41              return symbolTable[symbol];
42          }
43
44          int contains(string symbol) {
45              return symbolTable.count(symbol);
46          }
47  };
```

Function readSourceLine():

Description:

Read one line of the source program file and parse the string to get label, mnemonic and operand.

Parameters:

file: fstream object

label: string

mnemonic: string

operand: string

```
50  void readSourceLine(fstream &file, string &label, string &mnemonic, string &operand) {
51
52      label = "";
53      mnemonic = "";
54      operand = "";
55      string text;
56      char *word;
57
58      getline(file, text);
59      char charArr[text.length() + 1];
60      strcpy(charArr, text.c_str());
61
62
63      // Check if there is a label on this line. Initialize the label as empty string, "".
64      // If there is a new label which is not in symbol table, add the label into symbol table.
65      if (text[0] != '\t') {   // If the first character is not Tab character, it means there is a label.
66          word = strtok(charArr, " \t\r");
67          label = word;
68
69          // Get the mnemonic.
70          word = strtok(NULL, " \t\r");
71          mnemonic = word;
72
73          // Check if there is an operand on this line.
74          // If there is an operand, store the operand value in integer.
75          word = strtok(NULL, " \t\r");
76          if (word != NULL) {
77              operand = word;
78          }
79      }
80      else {
81          // Get the mnemonic.
82          word = strtok(charArr, " \t\r");
83          mnemonic = word;
84
85          // Check if there is an operand on this line.
86          // If there is an operand, store the operand value in integer.
87          word = strtok(NULL, " \t\r");
88          if (word != NULL) {
89              operand = word;
90          }
91      }
92  }
```

Function readIntermediateLine():

Description:

Read one line of the intermediate file and parse the string to get location, label, mnemonic and operand.

Parameters:

file: fstream object

location: string

label: string

mnemonic: string

operand: string

```
95  void readIntermediateLine(fstream &file, string &location, string &label, string &mnemonic, string &operand) {
96
97      location = "";
98      label = "";
99      mnemonic = "";
100     operand = "";
101     string text;
102     char *word;
103
104     getline(file, text);
105     char charArr[text.length() + 1];
106     strcpy(charArr, text.c_str());
107
108
109     // Check if there is a location on this line. Initialize the label as empty string, "".
110     if (text[0] != '\t') {   // If the first character is not Tab character, it means there is a location.
111         word = strtok(charArr, " \t\r");
112         location = word;
113
114         // Get the label if it exists.
115         if (text[location.length() + 1] != '\t') {
116             word = strtok(NULL, " \t\r");
117             label = word;
118         }
119
120         // Get the mnemonic.
121         word = strtok(NULL, " \t\r");
122         mnemonic = word;
123
124         // Check if there is an operand on this line.
125         // If there is an operand, store the operand value in integer.
126         word = strtok(NULL, " \t\r");
127         if (word != NULL) {
128             operand = word;
129         }
130     }
131     else {
132         // Get the mnemonic.
133         word = strtok(charArr, " \t\r");
134         mnemonic = word;
135
136         // Check if there is an operand on this line.
137         // If there is an operand, store the operand value in integer.
138         word = strtok(NULL, " \t\r");
139         if (word != NULL) {
140             operand = word;
141         }
142     }
143 }
```