

# PMAT 402 - Systems Programming

## Assignment -1

### Two-Pass Assembler

Gandholi Sarat - 23008

April 11, 2025

## Contents

<b>1</b>	<b>Objective</b>	<b>2</b>
<b>2</b>	<b>Idea of Implementation</b>	<b>2</b>
<b>3</b>	<b>Functions Description</b>	<b>2</b>
3.1	Hexadecimal to Decimal Conversion . . . . .	2
3.2	Decimal to Hexadecimal Conversion . . . . .	3
3.3	String Padding . . . . .	3
3.4	Token Splitter . . . . .	3
3.5	Character to Decimal Conversion . . . . .	4
3.6	Decimal to Hex Character Conversion . . . . .	4
3.7	Opcode Table Initialization . . . . .	5
<b>4</b>	<b>Full Code</b>	<b>7</b>
4.1	Operation Table . . . . .	7
4.2	Utility Functions . . . . .	8
4.3	Pass 1 . . . . .	11
4.4	Pass 2 . . . . .	13
<b>5</b>	<b>Input and Output Examples</b>	<b>19</b>
5.1	Input File (input.txt) . . . . .	19
5.2	Intermediate File (intermediate_file.txt) . . . . .	20
5.3	Symbol Table (SymTab.txt) . . . . .	21
5.4	Object Code (obj_prog.txt) . . . . .	21

# 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.

```
1 int hextodec(string hex) {  
2     int len = hex.length();  
3     int base = 1;  
4     int dec_num = 0;  
5     for (int i = len - 1; i >= 0; i--) {  
6         dec_num += dec(hex[i]) * base;  
7         base *= 16;  
8     }  
9     return dec_num;  
10 }
```

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.

```
1 string dectohex(int dec_num) {
2     string hex_str = "";
3     stack<int> stack;
4     int div = dec_num;
5     int rem = div % 16;
6     stack.push(rem);
7     div /= 16;
8     while (div > 15) {
9         rem = div % 16;
10        stack.push(rem);
11        div /= 16;
12    }
13    stack.push(div);
14    while (!stack.empty()) {
15        hex_str += hex(stack.top());
16        stack.pop();
17    }
18    return hex_str;
19 }
```

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.

```
1 std::string padWithZeros(const std::string& input, int desiredLength) {
2     return (input.length() >= desiredLength) ? input :
3         std::string(desiredLength - input.length(), '0') + input;
4 }
5
6 std::string padWithSpaces(const std::string& input, int desiredLength) {
7     return (input.length() >= desiredLength) ? input :
8         input + std::string(desiredLength - input.length(), ' ');
9 }
```

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.

```
1 vector<string> split(string str, char del) {
2     vector<string> v;
3     string temp = "";
4     for (int i = 0; i < str.size(); i++) {
5         if (str[i] != del) temp += str[i];
6         else {
7             v.push_back(temp);
8             temp = "";
9         }
10    }
11    v.push_back(temp);
12    return v;
13 }
```

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 `hexdec()` function.

```
1 int dec(char c) {
2     switch (c) {
3         case 'A': return 10;
4         case 'B': return 11;
5         case 'C': return 12;
6         case 'D': return 13;
7         case 'E': return 14;
8         case 'F': return 15;
9         default: return stoi(to_string(c)) - 48; // Converts ASCII to int
10    }
11 }
```

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.

```
1 string hex(int d) {
2     switch (d) {
3         case 10: return "A";
4         case 11: return "B";
5         case 12: return "C";
6         case 13: return "D";
7         case 14: return "E";
8         case 15: return "F";
9         default: return to_string(d);
10    }
```

```

10     }
11 }

```

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.

```

1 map<string, string> const_optab() {
2     map<string, string> optab;
3     optab["ADD"] = "18";
4     optab["ADDF"] = "58";
5     optab["ADDR"] = "90";
6     optab["AND"] = "40";
7     optab["CLEAR"] = "B4";
8     optab["COMP"] = "28";
9     optab["COMPF"] = "88";
10    optab["COMPR"] = "A0";
11    optab["DIV"] = "24";
12    optab["DIVF"] = "64";
13    optab["DIVR"] = "9C";
14    optab["FIX"] = "C4";
15    optab["FLOAT"] = "C0";
16    optab["HIO"] = "F4";
17    optab["J"] = "3C";
18    optab["JEQ"] = "30";
19    optab["JGT"] = "34";
20    optab["JLT"] = "38";
21    optab["JSUB"] = "48";
22    optab["LDA"] = "00";
23    optab["LDB"] = "68";
24    optab["LDCH"] = "50";
25    optab["LDF"] = "70";
26    optab["LDL"] = "08";
27    optab["LDS"] = "6C";
28    optab["LDT"] = "74";
29    optab["LDX"] = "04";
30    optab["LPS"] = "D0";
31    optab["MUL"] = "20";
32    optab["MULF"] = "60";
33    optab["MULR"] = "98";
34    optab["NORM"] = "C8";
35    optab["OR"] = "44";
36    optab["RD"] = "D8";
37    optab["RMO"] = "AC";
38    optab["RSUB"] = "4C";
39    optab["SHIFTL"] = "A4";
40    optab["SHIFTR"] = "A8";

```

```
41     optab["SIO"] = "F0";
42     optab["SSK"] = "EC";
43     optab["STA"] = "0C";
44     optab["STB"] = "78";
45     optab["STCH"] = "54";
46     optab["STF"] = "80";
47     optab["STI"] = "D4";
48     optab["STL"] = "14";
49     optab["STS"] = "7C";
50     optab["STSW"] = "E8";
51     optab["STT"] = "84";
52     optab["STX"] = "10";
53     optab["SUB"] = "1C";
54     optab["SUBF"] = "5C";
55     optab["SUBR"] = "94";
56     optab["SVC"] = "B0";
57     optab["TD"] = "E0";
58     optab["TIO"] = "F8";
59     optab["TIX"] = "2C";
60     optab["TIXR"] = "B8";
61     optab["WD"] = "DC";
62     return optab;
63 }
```

Listing 7: const\_optab() - Opcode Table Initialization

## 4 Full Code

### 4.1 Operation Table

```
1  #include <iostream>
2  #include <string>
3  #include <stack>
4  #include <map>
5  #include <string>
6
7  using namespace std;
8
9  map <string, string> const_optab()
10 {
11     map <string, string> optab;
12
13     optab["ADD"] = "18";
14     optab["ADDF"] = "58";
15     optab["ADDR"] = "90";
16     optab["AND"] = "40";
17     optab["CLEAR"] = "B4";
18     optab["COMP"] = "28";
19     optab["COMPF"] = "88";
20     optab["COMPR"] = "A0";
21     optab["DIV"] = "24";
22     optab["DIVF"] = "64";
23     optab["DIVR"] = "9C";
24     optab["FIX"] = "C4";
25     optab["FLOAT"] = "C0";
26     optab["HIO"] = "F4";
27     optab["J"] = "3C";
28     optab["JEQ"] = "30";
29     optab["JGT"] = "34";
30     optab["JLT"] = "38";
31     optab["JSUB"] = "48";
32     optab["LDA"] = "00";
33     optab["LDB"] = "68";
34     optab["LDCH"] = "50";
35     optab["LDF"] = "70";
36     optab["LDL"] = "08";
37     optab["LDS"] = "6C";
38     optab["LDT"] = "74";
39     optab["LDX"] = "04";
40     optab["LPS"] = "D0";
41     optab["MUL"] = "20";
42     optab["MULF"] = "60";
43     optab["MULR"] = "98";
44     optab["NORM"] = "C8";
45     optab["OR"] = "44";
46     optab["RD"] = "D8";
47     optab["RMO"] = "AC";
48     optab["RSUB"] = "4C";
49     optab["SHIFTL"] = "A4";
```

```
50     optab["SHIFTR"] = "A8";
51     optab["SIO"] = "F0";
52     optab["SSK"] = "EC";
53     optab["STA"] = "0C";
54     optab["STB"] = "78";
55     optab["STCH"] = "54";
56     optab["STF"] = "80";
57     optab["STI"] = "D4";
58     optab["STL"] = "14";
59     optab["STS"] = "7C";
60     optab["STSW"] = "E8";
61     optab["STT"] = "84";
62     optab["STX"] = "10";
63     optab["SUB"] = "1C";
64     optab["SUBF"] = "5C";
65     optab["SUBR"] = "94";
66     optab["SVC"] = "B0";
67     optab["TD"] = "E0";
68     optab["TIO"] = "F8";
69     optab["TIX"] = "2C";
70     optab["TIXR"] = "B8";
71     optab["WD"] = "DC";
72
73     return optab;
74 }
```

Listing 8: Operation Table (optab)

## 4.2 Utility Functions

```
1  #include <iostream>
2  #include <string>
3  #include <stack>
4  #include <iomanip>
5  #include <sstream>
6  using namespace std;
7
8  int dec(char c)
9  {
10     switch (c)
11     {
12         case 'A':
13             return 10;
14         case 'B':
15             return 11;
16         case 'C':
17             return 12;
18         case 'D':
19             return 13;
20         case 'E':
21             return 14;
22         case 'F':
23             return 15;
```



```
24         default:
25             return stoi(to_string(c))-48;    //ascii value to magnitude
26     }
27 }
28
29 int hextodec(string hex)
30 {
31     int len = hex.length();
32     int base = 1;
33     int dec_num = 0;
34     for (int i = len-1; i>=0; i--)
35     {
36         dec_num += dec(hex[i])*base;
37         base = base*16;
38     }
39     return dec_num;
40 }
41
42 string hex(int d)
43 {
44     switch (d)
45     {
46         case 10:
47             return "A";
48         case 11:
49             return "B";
50         case 12:
51             return "C";
52         case 13:
53             return "D";
54         case 14:
55             return "E";
56         case 15:
57             return "F";
58         default:
59             return to_string(d);
60     }
61 }
62
63 string dectohex(int dec_num)
64 {
65     string hex_str = "";
66     stack<int> stack;
67
68     int div, rem;
69     div = dec_num;
70     rem = div % 16;
71     //cout << "rem:" << rem << endl;
72     stack.push(rem);
73     div = (div)/16;
74     //cout << "div:" << div << endl;
75     while(div > 15)
76     {
77         rem = div % 16;
```

```

78         //cout << "rem:" << rem << endl;
79         stack.push(rem);
80         div = (div)/16;
81         //cout << "div:" << div << endl;
82     }
83     stack.push(div);
84     while(!stack.empty())
85     {
86         //cout << stack.top();
87
88         hex_str = hex_str + hex(stack.top());
89         stack.pop();
90     }
91     return hex_str;
92 }
93
94
95 std::string padWithZeros(const std::string& input, int desiredLength)
96 {
97     if (input.length() >= desiredLength) {
98         return input;
99     } else {
100         std::stringstream ss;
101         ss << std::string(desiredLength - input.length(), '0') <<
input;
102         return ss.str();
103     }
104
105 std::string padWithSpaces(const std::string& input, int desiredLength)
106 {
107     if (input.length() >= desiredLength) {
108         return input;
109     } else {
110         std::stringstream ss;
111         ss << input << std::string(desiredLength - input.length(), ' ');
112         return ss.str();
113     }
114
115 vector<string> split(string str, char del)
116 {
117     vector<string> v;
118     string temp = "";
119
120     for(int i=0; i<str.size(); i++)
121     {
122         if(str[i] != del)
123         {
124             temp += str[i];
125         }
126         else
127         {

```

```
128         v.push_back(temp);
129         temp = "";
130     }
131 }
132
133     v.push_back(temp);
134     return v;
135 }
```

Listing 9: Helper Functions

### 4.3 Pass 1

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <vector>
5  #include <map>
6
7  #include "Op_tab.h"
8  #include "functions.h"
9
10
11  using namespace std;
12
13  int main()
14  {
15      int locctr;
16      int start_add;
17      int prog_len;
18      int len;
19      string line;
20      vector<string> inst_fields;
21      map <string, string> OpTab;
22      map<string, string> SymTab;
23
24      //map <string, string> SymTab;
25
26      //Construction of optab
27      OpTab = const_optab();
28
29      ifstream fin("input.txt");
30      ofstream fout("intermediate_file.txt");
31      ofstream f2out("SymTab.txt");
32
33      getline(fin, line);
34      inst_fields = split(line, ' ');
35      string label = inst_fields[0];
36      string opcode = inst_fields[1];
37      string operand = inst_fields[2];
38
39      if(opcode == "START")
40      {
```

```

41         start_add = stoi(operand);
42         locctr = hextohex(to_string(start_add)); //here locctr is
already in hex
43         fout << dectohex(locctr) << " " << line << endl;
44         getline(fin, line);
45         inst_fields = split(line, ' ');
46         if(inst_fields[0]!="")
47             label = inst_fields[0];
48         if(inst_fields[1]!="")
49             opcode = inst_fields[1];
50         if(inst_fields[2]!="")
51             operand = inst_fields[2];
52     }
53     else
54     {locctr = 0;}
55
56
57     while(opcode != "END")
58     {
59         if (label != "")
60         {
61             if(SymTab[label]!="")
62                 {cout << locctr << "Error : Duplicate Symbol " << label
<< endl;}
63             else
64                 {SymTab[label] = dectohex(locctr);}
65         }
66
67         fout << dectohex(locctr) << " " << line << endl;
68
69         if(OpTab[opcode]!="")
70             locctr = locctr + 3;
71         else if (opcode == "WORD")
72             locctr = locctr + 3;
73         else if (opcode == "RESW")
74             locctr = locctr + (3*stoi(operand));
75         else if (opcode == "RESB")
76             locctr = locctr + stoi(operand);
77         else if (opcode == "BYTE")
78         {
79             if(operand[0] == 'C')
80                 len = (operand.length() - 3); // removing character {c
, ' , '}
81             else
82                 len = (operand.length() - 3)/2; //removing the
characters {X , '}
83
84             locctr = locctr + len;
85         }
86         else
87             cout << "Error : Invalid operation code" << endl;
88
89         //fout << locctr << " " << line << endl;
90         label = "";

```

```

91         getline(fin, line);
92         inst_fields = split(line, ' ');
93         int length = inst_fields.size();
94         if(length-- && inst_fields[0]!="")
95             label = inst_fields[0];
96         if(length-- && inst_fields[1]!="")
97             opcode = inst_fields[1];
98         if(length-- && inst_fields[2]!="")
99             operand = inst_fields[2];
100     }
101     fout << " " << line << endl;
102
103     prog_len = locctr - start_add + 1;
104
105     if (!f2out.is_open()) {
106         cerr << "Error: Unable to open SymTab.txt for writing." <<
endl;
107         return 1;
108     }
109
110     for (const auto& pair : SymTab) {
111         f2out << pair.first << " " << pair.second << endl;
112     }
113
114     fin.close();
115     fout.close();
116     f2out.close();
117
118     return 0;
119 }

```

Listing 10: Pass 1 - Intermediate File Generator

## 4.4 Pass 2

```

1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <vector>
5  #include <unordered_map>
6
7  #include "Op_tab.h"
8  #include "functions.h"
9
10 using namespace std;
11
12
13 int main()
14 {
15     string line;
16     vector<string> inst_fields;
17     map <string, string> OpTab;
18     map<string, string> SymTab;

```

```

19     ifstream fin2("SymTab.txt");
20
21     if (!fin2.is_open()) {
22         cerr << "Error: Unable to open SymTab.txt for reading." <<
23 endl;
24         return 1;
25     }
26
27     string line2;
28     while (getline(fin2, line2)) {
29         // Split each line into key and value
30         size_t pos = line2.find(' ');
31         if (pos != string::npos) {
32             string key = line2.substr(0, pos);
33             string value = line2.substr(pos + 1);
34             SymTab[key] = value;
35         } else {
36             cerr << "Error: Invalid line format in SymTab.txt" << endl
37 ;
38         }
39     }
40
41     ifstream fin("intermediate_file.txt");
42     ofstream fout("obj_prog.txt");
43
44     if (!fin.is_open() || !fout.is_open()) {
45         cerr << "Error: Unable to open input or output file." << endl;
46         return 1;
47     }
48
49     string locctr;
50     string label;
51     string opcode;
52     string operand;
53
54     // Store the position of the first line
55     streampos firstLinePos = fin.tellg();
56
57     getline(fin, line);
58     inst_fields = split(line, ' ');
59     locctr = inst_fields[0];
60     label = inst_fields[1];
61     opcode = inst_fields[2];
62     operand = inst_fields[3];
63
64     int start_add = hextodec(locctr); //here locctr is already in hex
65
66     string prevLine;
67     string prevAddress;
68     while (getline(fin, line)) {
69         // Trim leading and trailing whitespace
70         line.erase(0, line.find_first_not_of(" \t")); // trim leading
71         whitespace

```

```

70         line.erase(line.find_last_not_of(" \t") + 1); // trim trailing
    whitespace
71
72         // Store the address from the previous line
73         if (!prevLine.empty()) {
74             size_t pos = prevLine.find_first_of(" \t");
75             prevAddress = prevLine.substr(0, pos);
76         }
77
78         // Store the current line as the previous line
79         prevLine = line;
80     }
81
82     // Go back to the first line
83     fin.clear(); // Clear any error flags
84     fin.seekg(firstLinePos);
85     getline(fin, line); // reads first line which is not required
86
87     int last_add = hextodec(prevAddress);
88
89     int length = last_add - start_add + 1;
90     string l = dectohex(length);
91     l = padWithZeros(l, 6);
92
93     if(opcode == "START")
94     {
95         fout << "H^" << padWithSpaces(label, 6) << "^" <<
padWithZeros(locctr, 6) << "^" << l;
96     }
97
98
99     OpTab = const_optab();
100    getline(fin, line); // reads second line from which we need object
code
101
102    inst_fields = split(line, ' ');
103    locctr = inst_fields[0];
104    label = inst_fields[1];
105    opcode = inst_fields[2];
106    if (inst_fields.size() >= 4) {
107        operand = inst_fields[3];
108    } else {
109        operand = ""; // Set operand to empty string if it's not
present
110    }
111
112    int text_length;
113
114    while(opcode != "END"){//text_length > 6){
115        text_length = 60;
116        fout << '\n' << "T^" << padWithZeros(locctr , 6);
117        while (text_length > 0){//opcode != "END") {
118            int opAddress;
119            string objCode = "";

```

```

120         if(OpTab.find(opcode)!=OpTab.end())
121         {
122             if(operand[operand.length()-1] == 'X' && operand[
operand.length()-2] == ',')
123             {
124                 int l = hexdec(SymTab[operand.substr(0, operand.
length()-2)]);
125                 l = l+32768;
126                 objCode = OpTab[opcode] + dectohex(l);
127             }
128             else
129             {
130                 if(SymTab.find(operand) == SymTab.end())
131                     objCode = OpTab[opcode] + "0000";
132                 else
133                     objCode = OpTab[opcode] + SymTab[operand];
134             }
135             if(text_length >= 6){
136                 text_length -= 6;
137                 fout << "^" << objCode;
138             }
139             else
140                 break;
141         }else if(opcode == "BYTE")
142         {
143             if(operand[0] == 'C')
144             {
145                 for(int i=2; i< operand.length()-1; i++)
146                 {
147                     char c = operand[i];
148                     int asciiValue = c;
149                     objCode += (dectohex(asciiValue));
150                 }
151             }
152             else
153             {
154                 for(int i=2; i< operand.length()-1; i++)
155                 {
156                     objCode += operand[i];
157                 }
158             }
159             if(objCode.length() < text_length)
160             {
161                 text_length -= objCode.length();
162                 fout << "^" << objCode;
163             }
164             else{
165                 break;
166             }
167         }
168         else if(opcode == "WORD")
169         {
170             objCode = dectohex(stoi(operand));
171             objCode = padWithZeros(objCode, 6);

```



```

172         if(text_length >= 6){
173             text_length -= 6;
174             fout << "^" << objCode;
175         }
176     }
177     else if(opcode == "RESW" || opcode == "RESB")
178     {
179         while(opcode == "RESW" || opcode == "RESB")
180         {
181             if (!getline(fin, line)) {
182                 cerr << "Error: Unexpected end of file." << endl;
183                 break;
184             }
185             inst_fields = split(line, ' ');
186             locctr = inst_fields[0];
187             label = inst_fields[1];
188             opcode = inst_fields[2];
189             if (inst_fields.size() >= 4) {
190                 operand = inst_fields[3];
191             } else {
192                 operand = ""; // Set operand to empty string if it
's not present
193             }
194         }
195         break;
196     }
197
198     if (!getline(fin, line)) {
199         cerr << "Error: Unexpected end of file." << endl;
200         break;
201     }
202     inst_fields = split(line, ' ');
203     locctr = inst_fields[0];
204     label = inst_fields[1];
205     opcode = inst_fields[2];
206     if (inst_fields.size() >= 4) {
207         operand = inst_fields[3];
208     } else {
209         operand = ""; // Set operand to empty string if it's
not present
210     }
211 }
212 }
213
214     fout << '\n' << "E^" << padWithZeros(dectohex(start_add), 6) <<
endl;
215
216     fin2.close();
217     fin.close();
218     fout.close();
219
220
221     //for adding length of each record
222     ifstream fin3("obj_prog.txt");

```

```
223     ofstream fout2("temp.txt");
224     getline(fin3, line); // reads the header record
225     fout2 << line << endl;
226
227     vector<string> v;
228     int text_size;
229     while(getline(fin3, line) && line[0] == 'T'){
230         v = split(line, '^');
231         text_size = 0;
232         for(int i=2; i< v.size(); i++)
233         {
234             text_size += v[i].length();
235         }
236         line.insert(8, "^"+dectohex(text_size/2));
237         fout2 << line << endl;
238         v.erase(v.begin());
239     }
240
241     fout2 << line;
242
243     fin3.close();
244     fout2.close();
245
246     remove("obj_prog.txt");
247     rename("temp.txt", "obj_prog.txt");
248     return 0;
249 }
```

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

```