SIC/XE 流程說明
(由於程式碼有一千多行,只挑重點部分說明)

(程式執行前,需有 opcode.txt 和 SICXEsource.txt)

首先我定義了一個結構,每一行由一個 line 所組成
會將整行拆成 label mnemonic operand,之後會比較好處理

```
struct line //store the info of each line
{
    string wholeString;

    string label;
    string mnemonic;
    string operand;

    int location;
    string myOPCODE;

    string nixbpe;

    int byteSize;

};
```

Reg 儲存了暫存器的名子與編號

```
struct reg
{
    string name;
    int num;
};
```

symbolNode 會儲存 symbol 與對應的 location
(程式中的 location 我都會以 int 儲存,之後輸出才會轉成 16 進位)

```
struct symbolNode //store the info of the symbol and its address
{
    string symbol;
    int address;
};
```

各種 function:

將 16 進位的字串轉成對應的 int

```
int hexToDec(string s)//string to int
```

將 10 進位的字串轉成對應的 int

```
int dec_atoi(string s)//string to int
```

將 10 進位數字轉成 16 進位的字串

```
string decToHex(int num)//
```

將 10 進位數字轉成 16 進位的字串(不夠 4 位會補 0)

```
string decToHex_four(int num)//tu
```

將 10 進位數字轉成 16 進位的字串(不夠 3 位會補 0,這個可處理負數的情況,用在填入 opcode 的 disp)

```
string decToHex_three(int num)//
```

將 10 進位數字轉成 16 進位的字串(不夠 5 位會補 0，用在 format_4)

```
string decToHex_five(int num)/
```

這邊會用 hashMap 來儲存 mnemonic 所對應的 opcode,以及 symbol 所對應的 address

```
unordered_map<string,string> opTable;
unordered_map<string,int> symTable;

int symbolCount = 0;
symbolNode symbolTable[1000];//set the maximum symbol to 1000
int lineCount = 0;
line lineTable[1000];//set the maximum line to 1000

reg registerTable[9];//register table

registerTable[0].name = "A";
registerTable[0].num = 0;

registerTable[1].name = "X";
registerTable[1].num = 1;
```

這邊會讀取 SICXEsource.txt

```
fstream inputFile;
inputFile.open("SICXEsource.txt",ios::in); //read the SICXEsource.txt

fstream opcodeFile;
opcodeFile.open("opcode.txt",ios::in); //read the opcode.txt
```

建立 opTable

```
while(opcodeFile >> s)//put the opcode to opTable
{
    opcodeFile >> codeNum;
    opTable[s] = codeNum;
}
```

(以下程式碼太多,沒辦法完整截圖,只能截一小部分)
開始 pass_1
一開始會先讀取第一行,來取得我的 starting address
途中會進行字串處理把一行切成 label,mnemonic,operand

```
//==================================pass_1==================================
getline(inputFile,s); // get first line

line firstLine;
firstLine.wholeString = s;
unsigned int index = 0;//point the char from the first line
firstLine.myOPCODE = "";
firstLine.label = "";
while(true)//construct the label name till meet tab
{
    if(s[index] == ' ')//skip space
    {
        index++;
        continue;
    }
    if(s[index] == '\t')
```

之後把 pc 設成第一行的 location

```
int pc = firstLine.location; //set the program counter
```

之後再繼續取得其他行

```
while(getline(inputFile,s))//get other lines
{
    line temp;
    temp.wholeString = s;
    temp.location = pc;//set the location to pc
    unsigned int i = 0;
    if(s[0] != '\t')//have label(the first character is not tab)
    {
        temp.label = "";
        while(true)
        {
            if(s[i] == ' ')//skip space
            {
                i++;
                continue;
```

如果有 label 就把他加入 symbol table 內

```
symbolNode newSymbol; //create new symbol object
newSymbol.symbol = temp.label;
newSymbol.address = pc;

symbolTable[symbolCount] = newSymbol; //insert the
symTable[newSymbol.symbol] = newSymbol.address; //i
symbolCount++;
```

之後會依照 mnemonic 來增加 pc,並把 pc 填入該行的 location 中

```
//add the program counter
if(temp.mnemonic == "BASE")
{
    //pc won't add , just skip
}
else if(temp.mnemonic == "WORD")
{
    pc += 3;
}
else if(temp.mnemonic == "RESW")
{
    pc += 3*(dec_atoi(temp.operand));
}
else if(temp.mnemonic == "RESB")
{
    pc += dec_atoi(temp.operand);
}
```

```
else if(temp.mnemonic == "BYTE")
{
    if((temp.operand)[0] == 'C')
    {
        pc += strlen((temp.operand).c_str()) - 3
    }
    else//X' ' , one byte
    {
        pc += 1;
    }
}
else//normal instruction
{
    if((temp.mnemonic)[0] == '+')//format_4
    {
        pc += 4;
        temp.byteSize = 4;
    }
```

比較麻煩的是 format_2 與 format_3 的判斷

這邊我會判斷 operand 有沒有@,#,或是逗號

如果有@,#那就是 format_3,如果沒有的話要判斷 operand 中有無逗號

如果有逗號,要判斷逗號兩邊的 string 是不是都是 register name

如過是的話就是 format_2

否則就是 index addressing 的 format_3

```
        else
        {
            s1 += (temp.operand)[j];
        }
    }
    if(haveComma == true)//2 register or index addressing
    {
        for( ; j < operandLen; j++)//construct another string
        {
            s2 += (temp.operand)[j];
        }
        bool isReg_1 = false;
        for(int k = 0; k < 9; k++)
        {
            if(s1 == registerTable[k].name)
            {
                isReg_1 = true;
                break;
```

完成後輸出 location 與對應的 source statement

```
//write the locationAndSource.txt
fstream locationAndSource;
locationAndSource.open("pass1_locationAndSource.txt",ios::out);
locationAndSource << "Loc\t" << "Source statement" << endl;
locationAndSource << decToHex_four(firstLine.location) << "\t" << firstLine.wholeString << endl;
for(int i = 0; i < lineCount-1; i++)
{
    if(lineTable[i].mnemonic == "BASE")//base don't output location
    {
        locationAndSource << "\t" << lineTable[i].wholeString << endl;
    }
    else
    {
        locationAndSource << decToHex_four(lineTable[i].location) << "\t" << lineTable[i].wholeString << endl;
    }

}
```

並輸出 symbol tabel

```
locationAndSource << "\t" << lineTable[lineCount-1].wholeString << endl;
locationAndSource.close();
cout << "Pass 1 : " << endl << endl;
cout << "write the location of each source code to pass1_locationAndSource.txt!" << endl << endl;

//write the symbolTable.txt
fstream symbolTableFile;
symbolTableFile.open("pass1_symbolTable.txt",ios::out);
symbolTableFile << "Name\t" << "Address" << endl;
for(int i = 0; i < symbolCount; i++)
{
    symbolTableFile << symbolTable[i].symbol << "\t" << decToHex_four(symbolTable[i].address) << endl;
}
symbolTableFile.close();
cout << "write the symbol table to pass1_symbolTable.txt!" << endl << endl;
```

之後開始進入 pass_2

一開始會由 mnemonic 來分類,像 BASE,RESW,RESB 都不需填入 opcode

```
if(lineTable[i].mnemonic == "BASE")//set the value of base
{
    baseValue = symTable[lineTable[i].operand];
}
else if(lineTable[i].mnemonic == "RESW" || lineTable[i].mnemonic == "RESB")
{
    lineTable[i].myOPCODE = "";
    continue;
}
else
{
    if(lineTable[i].mnemonic == "WORD")
    {
        string opcode = "";
```

WORD 的話它的 operand 是一個數字,要把他轉成 16 進位(不夠 6 位要補 0)

```
if(lineTable[i].mnemonic == "WORD")
{
    string opcode = "";
    string s1 = decToHex(dec_atoi(lineTable[i].operand));
    for(unsigned int j = 0; j < (6-strlen(s1.c_str())); j++)//fill 0 to length 6
    {
        opcode += "0";
    }
    opcode += s1;
    lineTable[i].myOPCODE = opcode;
}
```

BYTE 的話有分成 C 開頭與 X 開頭
C 開頭代表後面有一些字元,會連續串在 opcode 上面

```
else if(lineTable[i].mnemonic == "BYTE")
{
    if((lineTable[i].operand)[0] == 'C')//C''
    {
        string opcode = "";
        for(unsigned int j = 2; j < (strlen((lineTable[i].operand).c_str())-1); j++)//
        {
            string t = decToHex((int)((lineTable[i].operand)[j]));
            if(strlen(t.c_str()) == 1)
            {
                opcode += "0";
            }
            opcode += t;
        }
        lineTable[i].myOPCODE = opcode;
    }
```

X 開頭的話為一個 16 進位(1 Byte)

```
else//X''
{
    int len = strlen((lineTable[i].operand).c_str());
    string opcode = "";
    for(int j = 2; j < len-1; j++)
    {
        opcode += (lineTable[i].operand)[j];
    }
    lineTable[i].myOPCODE = opcode;
}
```

剩下的就是 format_2 , format_3 以及 format_4 的 instruction 了
如果 mnemonic 是+開頭就是 format_4

```
if((lineTable[i].mnemonic)[0] == '+')//format_4
{
    if(lineTable[i].operand == "")//no operand , fill 5 zero
    {
        string realMneminic = "";
        for(unsigned int j = 1; j < strlen((lineTable[i].mnemonic).c_str()); j++)
        {
            realMneminic += (lineTable[i].mnemonic)[j];
        }

        lineTable[i].nixbpe = "110001";
        string opcode = decToHex(hexToDec(opTable[realMneminic])+3);//use optable to get opcode
        if(strlen(opcode.c_str()) == 1)
        {
            opcode = "0" + opcode;
        }
        opcode += "1";//xbpe
```

會分別針對@,#以及 index addressing 來進行處理

```
if((lineTable[i].operand)[0] == '@')//xbp = 0
{
    string realMneminic = "";
    for(unsigned int j = 1; j < strlen((lineTable[i].mnemonic).c_str()); j++)//get mnemonic with
    {
        realMneminic += (lineTable[i].mnemonic)[j];
    }

    string opcode;
    lineTable[i].nixbpe = "100001";
    opcode = decToHex(hexToDec(opTable[realMneminic])+2);
    if(strlen(opcode.c_str()) == 1)
    {
        opcode = "0" + opcode;
    }
```

之後會依據情況來設定 nixbpe(但實際上沒用到,只是標記一下方便計算)
n 與 i 我會直接換算成 10 進位的 0~3，再直接加上 optable 的數字
之後 opcode 再加上 xbpe 的數字(串一個字元),最後再填上 disp 就完成了

```
else if((lineTable[i].operand)[0] == '#')//xbp = 0
{
    string realMneminic = "";
    for(unsigned int j = 1; j < strlen((lineTable[i].mnemonic).c_str()); j++)//get mnemonic with
    {
        realMneminic += (lineTable[i].mnemonic)[j];
    }

    string opcode;
    lineTable[i].nixbpe = "010001";
    opcode = decToHex(hexToDec(opTable[realMneminic])+1);
    if(strlen(opcode.c_str()) == 1)
    {
        opcode = "0" + opcode;
    }
    opcode += "1";//xbpe
```

如果 operand 有#
要判斷是不是純數字,來做額外處理

```
if(allNumber == true)//operand is number
{
    string myNum = decToHex(dec_atoi(realOperand));
    for(unsigned int k = 0; k < (5-strlen(myNum.c_str())); k++)//fill 0 to length 5
    {
        opcode += "0";
    }
    opcode += myNum;
}
else//operand is symbol
{
    opcode += decToHex_five(symTable[realOperand]);
}
```

之後我還要看裡面有沒有存在逗號,有的話就是 index addressing

```
//check if it's index addressing
int x = -1;
for(unsigned int j = 0; j < strlen((lineTable[i].operand).c_str()); j++)//if :
{
    if((lineTable[i].operand)[j] == ',')//indexed addressing
    {
        x = j;
        break;
    }
}
```

剩下的情況就是 format_2 與 format_3

如果 operand 為空,為 format_3,先依據 nixbpe 建立 opcode,後面補 3 個 0

```
if(lineTable[i].operand == "")//no operand , fill with 3 zeros
{
    lineTable[i].nixbpe = "110000";
    string opcode = decToHex(hexToDec(opTable[lineTable[i].mnemonic])+3)
    if(strlen(opcode.c_str()) == 1)
    {
        opcode = "0" + opcode;
    }
    opcode += "0";
    opcode += "000";
    lineTable[i].myOPCODE = opcode;

}
```

如果 operand 有@,代表是 format_3

取得 TA 與 PC 看不能 pc-relative

不行的話就用 base

```
if((lineTable[i].operand)[0] == '@')//format3
{
    string opcode;
    string realOperand = "";
    for(unsigned int j = 1; j < strlen((lineTable[i].operand).c_str()); j++)//get t
    {
        realOperand += (lineTable[i].operand)[j];
    }
    int TA = symTable[realOperand];
    int PC = lineTable[i+1].location;

    if(TA-PC >= -2048 && TA-PC <= 2047)//pc relative
    {
        lineTable[i].nixbpe = "100010";
        opcode = decToHex(hexToDec(opTable[lineTable[i].mnemonic])+2);
        if(strlen(opcode.c_str()) == 1)
        {
            opcode = "0" + opcode;
```

```
{
    lineTable[i].nixbpe = "100010";
    opcode = decToHex(hexToDec(opTable[lineTable[i].mnemonic])+2);
    if(strlen(opcode.c_str()) == 1)
    {
        opcode = "0" + opcode;
    }
    opcode += "2";//xbpe
    opcode += decToHex_three(TA-PC);
    lineTable[i].myOPCODE = opcode;

}
else//base relative
{
    lineTable[i].nixbpe = "100100";
    opcode = decToHex(hexToDec(opTable[lineTable[i].mnemonic])+2);
    if(strlen(opcode.c_str()) == 1)
    {
```

如果 operand 有#,代表是 format_3

先看 operand 是不是全是數字

不是的話我們再填入 disp

取得 TA 與 PC 看不能 pc-relative

不行的話就用 base

```
else if((lineTable[i].operand)[0] == '#')//format3
{
    string opcode;
    string realOperand = "";
    bool allNumber = true;
    for(unsigned int j = 1; j < strlen((lineTable[i].operand).c_str()); j++)//get th
    {
        realOperand += (lineTable[i].operand)[j];
        if(!((lineTable[i].operand)[j] >= '0' && (lineTable[i].operand)[j] <= '9'))
        {
            allNumber = false;
        }
    }

    if(allNumber == true)//the operand is number
    {
        lineTable[i].nixbpe = "010000";
        opcode = decToHex(hexToDec(opTable[lineTable[i].mnemonic])+1);
```

```
else//the operand is symbol
{
    int TA = symTable[realOperand];
    int PC = lineTable[i+1].location;

    if(TA-PC >= -2048 && TA-PC <= 2047)//pc relative
    {
        lineTable[i].nixbpe = "010010";
        opcode = decToHex(hexToDec(opTable[lineTable[i].mnemonic])+1);
        if(strlen(opcode.c_str()) == 1)
        {
            opcode = "0" + opcode;
        }
        opcode += "2";//xbpe
        opcode += decToHex_three(TA-PC);
        lineTable[i].myOPCODE = opcode;

else//base relative
{
    lineTable[i].nixbpe = "010100";
    opcode = decToHex(hexToDec(opTable[lineTable[i].mnemonic])+1);
    if(strlen(opcode.c_str()) == 1)
    {
        opcode = "0" + opcode;
    }
    opcode += "4";//xbpe

    opcode += decToHex_three(TA-baseValue);
    lineTable[i].myOPCODE = opcode;

}
```

如果 mnemonic 沒有特殊贅詞,可分為 operand 有逗號與無逗號的狀況
如果有逗號,有可能是 format_2,也有可能是 format_3 的 index addressing
我們要看逗號兩側的 string 是否都為 register name
是的話就是 format_2
否則為 index addressing 的 format_3

```
else//format3(normal or index addressing) or format2(1 or 2 Register)
{

    unsigned int operandLen = strlen(lineTable[i].operand.c_str());
    bool haveComma = false;
    unsigned int j = 0;
    string s1 = "";
    string s2 = "";
    for( ; j < operandLen; j++)//check if there is a comma, if comma exist ,
    {
        if((lineTable[i].operand)[j] == ',')
        {
            haveComma = true;
            j++;
            break;
        }

 if(haveComma == true)//2 register or index addressing
 {
     for( ; j < operandLen; j++)//construct another string s2
     {
         s2 += (lineTable[i].operand)[j];
     }
     bool isReg_1 = false;
     for(int k = 0; k < 9; k++)
     {
         if(s1 == registerTable[k].name)
         {          reg main::registerTable
             isReg_1 = true;
             break;
         }
     }

if(isReg_1 == true && isReg_2 == true)//if two operand are all register, format_2
{
    string opcode;
    opcode = opTable[lineTable[i].mnemonic];
    for(int k = 0; k < 9; k++)
    {
        if(s1 == registerTable[k].name)
        {
            opcode += (char)('0' + registerTable[k].num);
            break;
        }
    }
    for(int k = 0; k < 9; k++)
    {
        if(s2 == registerTable[k].name)
        {
            opcode += (char)('0' + registerTable[k].num);
```

```
else//index addressing
{
    //real operand = s1

    string opcode;
    int TA = symTable[s1];
    int PC = lineTable[i+1].location;

    if(TA-PC >= -2048 && TA-PC <= 2047)//pc relative
    {
        lineTable[i].nixbpe = "111010";
        opcode = decToHex(hexToDec(opTable[lineTable[i].mnemonic])+3);
        if(strlen(opcode.c_str()) == 1)
        {
            opcode = "0" + opcode;
```

若為 format_2 填入 optable 對應的 opcode 與 2 個暫存器的代號即可
若為 format_3,nixbpe 的 x 會被設為 1,之後再看是用 pc-relative 或 base-relative

若沒有逗號,代表是一個正常的 operand,如果是 register name,為 format_2
填入 opcode 後再填入 register 代號,後面補一個 0
如果是 symbol,則使用 pc-relative 或 base-relative 來獲得 disp

```
else//one operand
{
    bool isReg = false;
    for(int k = 0; k < 9; k++)
    {
        if(lineTable[i].operand == registerTable[k].name)
        {
            isReg = true;
            break;
        }
    }

    if(isReg == true)//if the operand is register
    {
        string opcode;
        opcode = opTable[lineTable[i].mnemonic];
        for(int k = 0; k < 9; k++)
        {
```

最後把每行對應的 opcode 輸出

```
cout << "Pass_2 : " << endl << endl;
//write the source with location and objectcode to pass2_source_LocObj.txt
fstream source_LocObj;
source_LocObj.open("pass2_source_LocObj.txt",ios::out);
source_LocObj << decToHex_four(firstLine.location) << "\t" << firstLine.wholeString << endl;
for(int i = 0; i < lineCount-1; i++)
{
    if(lineTable[i].mnemonic == "BASE")
    {
        source_LocObj << "\t" << lineTable[i].wholeString << endl;
    }
    else if(strlen(lineTable[i].operand.c_str()) >= 8)
    {
        source_LocObj << decToHex_four(lineTable[i].location) << "\t" << lineTable[i].wholeString << "\t" << lineTab
    }
    else if(lineTable[i].operand == "")
    {
        source_LocObj << decToHex_four(lineTable[i].location) << "\t" << lineTable[i].wholeString << "\t\t\t" << lir
```

最後再輸出 text record

```
//write the text record to pass2_textRecord.txt
fstream pass2_textRecord;
pass2_textRecord.open("pass2_textRecord.txt",ios::out);
//out format of first line
pass2_textRecord << 'H';
pass2_textRecord << firstLine.label;
for(unsigned int i = 0; i < 6-strlen(firstLine.label.c_str()); i++)//fill space
{
    pass2_textRecord << ' ';
}
pass2_textRecord << "00" << decToHex_four(firstLine.location) << "00" << decToHex_four(lineTable[lineCount-1].locat

for(int i = 0; i < lineCount-1; )
{
    string opcodeBuffer = "";
    int recordLen = 0;
    bool first = true;
```

要注意的是 format_4 需要被 modified

所以我會再跑一次迴圈看每一行

如果是 format_4，先看是不是立即值

不是的話就需要被 modified

```
for(int i = 0; i < lineCount-1; i++)//to get format 4 and write modification record
{
    if((lineTable[i].mnemonic)[0] == '+')//format_4(format_4 can't use index addressing)
    {
        if(lineTable[i].operand != "")
        {
            if((lineTable[i].operand)[0] == '#')//to check if the operand is number
            {
                bool allNumber = true;
                for(unsigned int j = 1; j < strlen(lineTable[i].operand.c_str()); j++)
                {
                    if(!((lineTable[i].operand)[j] >= '0' && (lineTable[i].operand)[j] <= '9')
                    {
                        allNumber = false;
                    }
                }
```

結果:

pass1_locationAndSource

Loc	Source statement		
0000	COPY	START	0
0000	FIRST	STL	RETADR
0003		LDB	#LENGTH
		BASE	LENGTH
0006	CLOOP	+JSUB	RDREC
000A		LDA	LENGTH
000D		COMP	#0
0010		JEQ	ENDFIL
0013		+JSUB	WRREC
0017		J	CLOOP
001A	ENDFIL	LDA	EOF
001D		STA	BUFFER
0020		LDA	#3
0023		STA	LENGTH
0026		+JSUB	WRREC
002A		J	@RETADR
002D	EOF	BYTE	C'EOF'
0030	RETADR	RESW	1
0033	LENGTH	RESW	1
0036	BUFFER	RESB	4096
1036	RDREC	CLEAR	X
1038		CLEAR	A
103A		CLEAR	S
103C		+LDT	#4096
1040	RLOOP	TD	INPUT
1043		JEQ	RLOOP
1046		RD	INPUT
1049		COMPR	A,S

pass1_symbolTable

Name	Address
FIRST	0000
CLOOP	0006
ENDFIL	001A
EOF	002D
RETADR	0030
LENGTH	0033
BUFFER	0036
RDREC	1036
RLOOP	1040
EXIT	1056
INPUT	105C
WRREC	105D
WLOOP	1062
OUTPUT	1076

pass2_source_LocObj

pass2_source_LocObj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

0000	COPY	START	0	
0000	FIRST	STL	RETADR	17202D
0003		LDB	#LENGTH	69202D
		BASE	LENGTH	
0006	CLOOP	+JSUB	RDREC	4B101036
000A		LDA	LENGTH	032026
000D		COMP	#0	290000
0010		JEQ	ENDFIL	332007
0013		+JSUB	WRREC	4B10105D
0017		J	CLOOP	3F2FEC
001A	ENDFIL	LDA	EOF	032010
001D		STA	BUFFER	0F2016
0020		LDA	#3	010003
0023		STA	LENGTH	0F200D
0026		+JSUB	WRREC	4B10105D
002A		J	@RETADR	3E2003
002D	EOF	BYTE	C'EOF'	454F46
0030	RETADR	RESW	1	
0033	LENGTH	RESW	1	
0036	BUFFER	RESB	4096	
1036	RDREC	CLEAR	X	B410
1038		CLEAR	A	B400
103A		CLEAR	S	B440
103C		+LDT	#4096	75101000
1040	RLOOP	TD	INPUT	E32019
1043		JEQ	RLOOP	332FFA
1046		RD	INPUT	DB2013
1049		COMPR	A,S	A004
104B		JEQ	EXIT	332008

pass2_textRecord

pass2_textRecord - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
HCOPY    000000001077
T0000001D17202D69202D4B1010360320262900003320074B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T0010361DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T0010531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T001070073B2FEF4F000005
M00000705
M00001405
M00002705
E000000
```

我們把 SICXEsource 的 starting address 更改再看一下結果
將起始位置更改為 2000

```
COPY      START     2000
```

pass1_locationAndSource

pass1_locationAndSource - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
Loc       Source statement
2000      COPY      START     2000
2000      FIRST     STL       RETADR
2003                LDB       #LENGTH
                    BASE      LENGTH
2006      CLOOP     +JSUB     RDREC
200A                LDA       LENGTH
200D                COMP      #0
2010                JEQ       ENDFIL
2013                +JSUB     WRREC
2017                J         CLOOP
201A      ENDFIL    LDA       EOF
201D                STA       BUFFER
2020                LDA       #3
2023                STA       LENGTH
2026                +JSUB     WRREC
202A                J         @RETADR
202D      EOF       BYTE      C'EOF'
2030      RETADR    RESW      1
2033      LENGTH    RESW      1
2036      BUFFER    RESB      4096
3036      RDREC     CLEAR     X
3038                CLEAR     A
303A                CLEAR     S
303C                +LDT      #4096
3040      RLOOP     TD        INPUT
3043                JEQ       RLOOP
3046                RD        INPUT
3049                COMPR     A,S
```

pass1_symbolTable

pass1_symbolTable - 記事

檔案(F) 編輯(E) 格式(O) 檢視

Name	Address
FIRST	2000
CLOOP	2006
ENDFIL	201A
EOF	202D
RETADR	2030
LENGTH	2033
BUFFER	2036
RDREC	3036
RLOOP	3040
EXIT	3056
INPUT	305C
WRREC	305D
WLOOP	3062
OUTPUT	3076

pass2_source_LocObj

pass2_source_LocObj - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

2000	COPY	START	2000	
2000	FIRST	STL	RETADR	17202D
2003		LDB	#LENGTH	69202D
		BASE	LENGTH	
2006	CLOOP	+JSUB	RDREC	4B103036
200A		LDA	LENGTH	032026
200D		COMP	#0	290000
2010		JEQ	ENDFIL	332007
2013		+JSUB	WRREC	4B10305D
2017		J	CLOOP	3F2FEC
201A	ENDFIL	LDA	EOF	032010
201D		STA	BUFFER	0F2016
2020		LDA	#3	010003
2023		STA	LENGTH	0F200D
2026		+JSUB	WRREC	4B10305D
202A		J	@RETADR	3E2003
202D	EOF	BYTE	C'EOF'	454F46
2030	RETADR	RESW	1	
2033	LENGTH	RESW	1	
2036	BUFFER	RESB	4096	
3036	RDREC	CLEAR	X	B410
3038		CLEAR	A	B400
303A		CLEAR	S	B440
303C		+LDT	#4096	75101000
3040	RLOOP	TD	INPUT	E32019
3043		JEQ	RLOOP	332FFA
3046		RD	INPUT	DB2013
3049		COMPR	A,S	A004
304B		JEQ	EXIT	332008

這邊可發現除了 format_4 其他都不會產生變動,符合 relocation

pass2_textRecord

pass2_textRecord - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
HCOPY    002000001077
T0020001D17202D69202D4B10303603202629000003320074B10305D3F2FEC032010
T00201D130F20160100030F200D4B10305D3E2003454F46
T0030361DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T0030531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T003070073B2FEF4F000005
M00200705
M00201405
M00202705
E002000
```

Text record 也正常輸出