國立嘉義大學資訊工程學系

系統程式期末專題報告

Department of Computer Science and Information Engineering National Chiayi University System Program Final Project Report

SIC/XE Assembler & Loader

指導教授: 王智弘 教授

組員: 1102913 王柏勝

1102920 陳柏凱

1102928 林聖博

1102956 陳為盛

1102937 余貫瑒

中華民國 一百一十三 年 六 月

一、專案架構與組員分工

我們的組別所被分配到的專案為 Two Pass 的 SIC/XE Assembler 和 Loader。而 Assembler 和 Loader 皆是使用JavaScript撰寫。

在 Two Pass 的組譯器中，我們將其分成分成三個部分，並分配給四個組員，每個組員將負責的部分撰寫成 JavaScript 中的 function，最後將每個 function 併入 export function，促成一個 module，再由主程式透過 import module 的方式執行各個module 中的 export function，來完成組譯器的工作。

如圖1，這是我們的專案目錄結構圖，在根錄下中的 index.html 是我們用來執行組譯器和loader 的地方。目錄中有一個名為 module 的子目錄，是組員分工一起寫的 module，其中 config.json 為大家共用的 module，而紅色的 Error_Scan.js、藍色的 Pass1.js 以及綠色的 Pass2.js 和 loader.js 為組員們寫 module，其中綠色的 Pass2.js 和 loader.js 由兩人共同完成，而黃色的 sicasm.js 用來啟動所有 module 的程式。

圖1、專案目錄結構圖

組員分工以及組譯器流程圖如下方的圖2與圖3，首先由 1102937 余貫瑒負責蒐集測試資料與撰寫答案。接著由 1102913 王柏勝負責寫組譯器中的 Error Scan，這個部分會偵測 source（FileContent）中是否含有 Error（除了Expression illegal檢查與PC or Base relative

檢查），若有 Error，程式會暫停執行，並將錯誤結果輸出在 Console Log 中，反之，則會進入到由 1102956 陳為盛負責的 Pass 1。

經過 Pass 1 會生成 source 的 SYMTAB 和 InterFile，此時會進入由 1102920 陳柏凱和 1102928 林聖博負責的 Pass 2。

在 Pass 2 中會對 source 中進行 PC or Base relative 的檢查，若發現有不合法的狀況，會停止程式執行，並在 Console Log 中輸出錯誤訊息，反之，則繼續執行檢查所有的 expression 是否合法，若發現不合法的狀況，程式則會終止，並輸出錯誤資訊，若皆為合法，則會在 Html 中顯示 SYMTAB 以及 Object code 並可提供下載。

而如圖2所示，在組譯程式的 module 中，皆會有一些共用的 module 來撰寫，

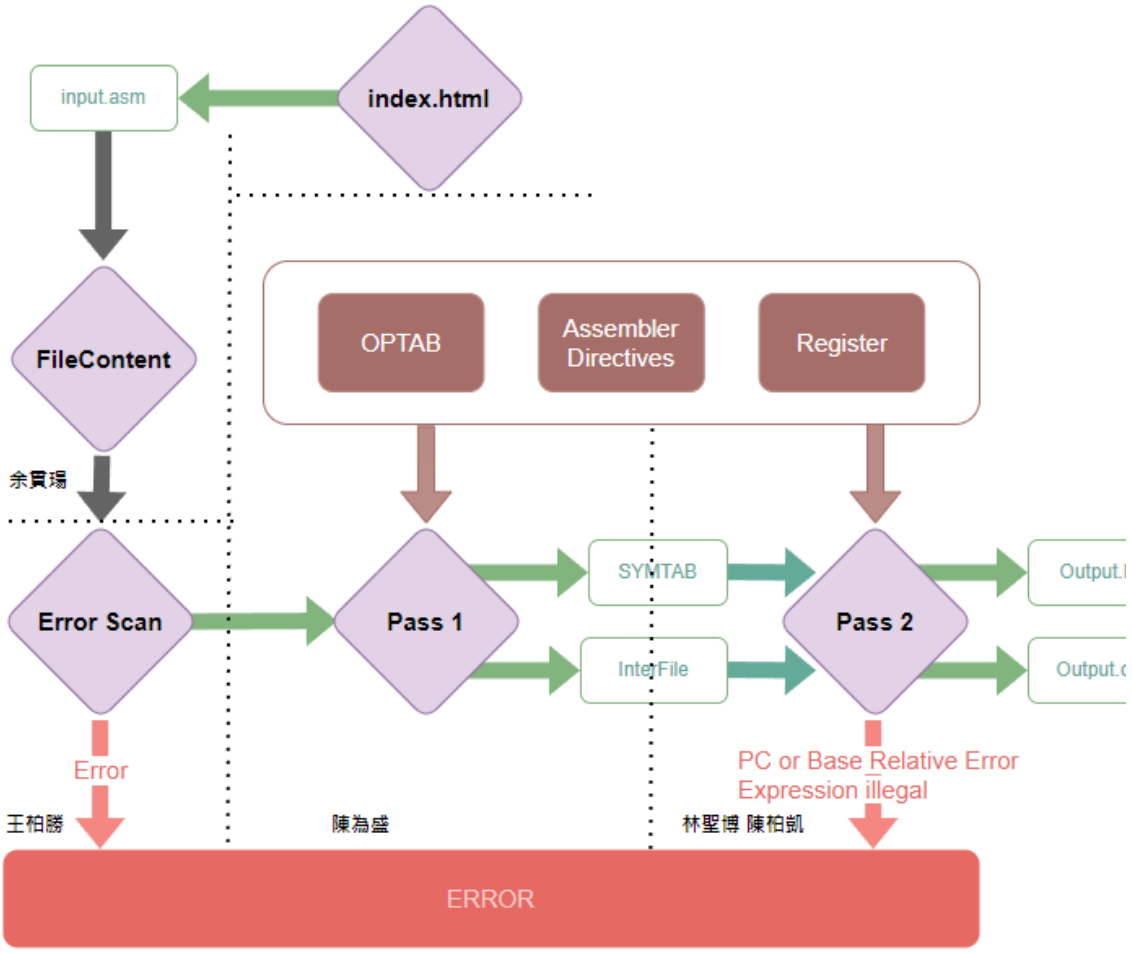


圖2、組譯器流程與分工圖

在 Loader 的部分，我們會將先前生成的 Object file 輸入進由 1102928 林聖博所撰寫的 Loader。

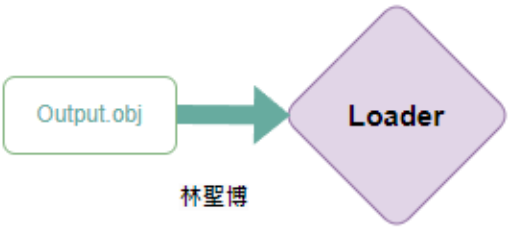


圖3、Loader流程圖與分工圖

如圖4，這些為專案中共用的 module，OPTAB、Assembler Directives和Register，全部都包含在 config.json中。

```
"Assembler_directives": [
  "START",
  "END",
  "BYTE",
  "WORD",
  "RESB",
  "RESW",
  "BASE",
  "EQU"
],
"Register": {
  "A": "0",
  "X": "1",
  "L": "2",
  "B": "3",
  "S": "4",
  "T": "5",
  "F": "6",
  "PC": "8",
  "SW": "9"
},
"OPTAB": {
  "ADD": "18",
  "CLEAR": "B4",
  "COMP": "28",
  "COMPR": "A0",
  "DIV": "24",
  "J": "3C",
  "JEQ": "30",
```

圖4、config.json

二、測資蒐集與答案撰寫

1. 製作輸入的程式檔

Mnemonic	Format	Opcode	Mnemonic	Format	Opcode	Mnemonic	Format	Opcode
ADD	3/4	18	LDA	3/4	00	STB	3/4	78
CLEAR	2	B4	LDB	3/4	68	STCH	3/4	54
COMP	3/4	28	LDCH	3/4	50	STL	3/4	14
COMPR	2	A0	LDL	3/4	08	STT	3/4	84
DIV	3/4	24	LDT	3/4	74	STX	3/4	10
J	3/4	3C	LDX	3/4	04	SUB	3/4	1C
JEQ	3/4	30	MUL	3/4	20	TD	3/4	E0
JGT	3/4	34	RD	3/4	D8	TIX	3/4	2C
JLT	3/4	38	RSUB	3/4	4C	TIXR	2	B8
JSUB	3/4	48	STA	3/4	0C	WD	3/4	DC

表1、Format and Opcode of Instruction

Directives
START
END
BYTE
WORD
RESB
RESW
BASE
EQI

表2、List of Directives

Input Field	Column
Label	1~9
opcode	13~21
operand	25~44

表3、Input field

上述的表1、表2以及表3列出了題目所給的Directives、Mnemonic、Input field，根據題目的要求，蒐集了符合格式要求的輸入測資，其中包括了教學平台提供的測資、學長姐提供的部分測資。

2. 製作輸出的檔案

```
=SYMTAB=
COPY      0000
FIRST     0000
CLOOP     0006
...
...

=OBJECT CODES=
1   LOC=0000   null
2   LOC=0000   pc-relative, nixbpe=110010, obj=17202D
3   LOC=0003   pc-relative, nixbpe=010010, obj=69202D
4   LOC=0003   null
```

圖5、輸出範例

根據圖5，產生符合格式的輸出檔，該檔案包含了 SYMTAB 和 OBJECT CODES，在 SYMTAB 中需要包含 Label 和 LOC。在 OBJECT CODES 則需列出行數、LOC、所使用的format 和 nixbpe、object code。若該行為 '.'開頭的註解，則需在該行列出 comment，如果是 format 3，則必須判斷為 PC 或 BASE Relative。

三、ERROR SCAN

1. 簡介

這個程序使得檔案在進入Pass1處理前，能先經過基本的錯誤掃描，檢查輸入檔案是否存在格式、語法的錯誤，來決定要交付給Pass1繼續進行處理還是直接輸出錯誤訊息以及修改建議。

2. 架構介紹(由左至右運行)

Error Scan的架構圖如圖6所示，較細地劃分成四個步驟(ppt版粗略分成三 個步驟，將步驟二、三合併):

i. 步驟一:讀取檔案

ii. 步驟二:檢查每行格式

iii. 步驟三:檢查每行指令和Operand

iv. 步驟四:輸出錯誤、警告和修改建議訊息

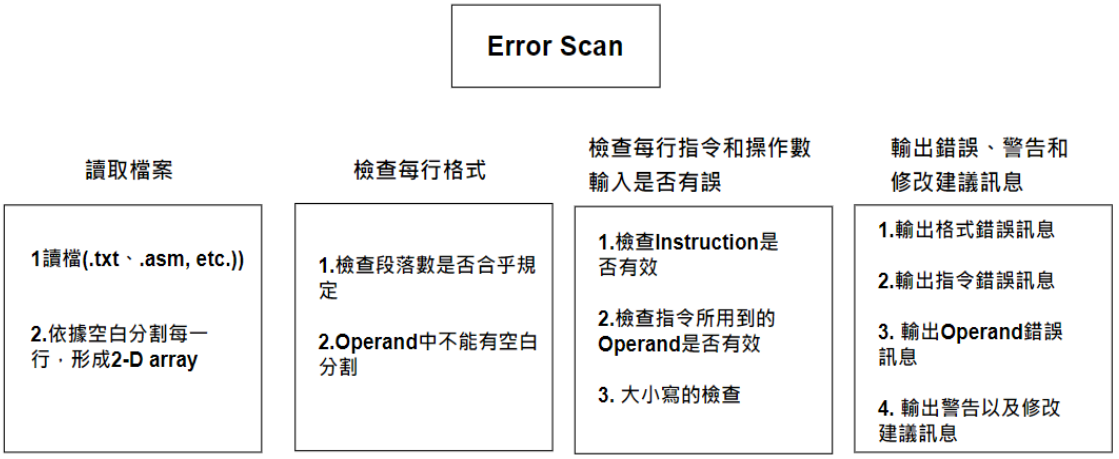


圖6、Error Scan架構圖

3. 詳細流程講解

a. 步驟一(讀檔)

i. 讀取檔案，這個檔案的副檔名可以是.asm、.txt等。

ii. 利用JS的map方法將每一行遍歷，再將每行頭尾的空白去除，再根據正則表達式，將字符之間的空白作為分割陣列的依據，即每行存成一個含有多個元素的陣列，最終多行讀入，所形成的陣列為一個2-D的array。

iii. 初始化一個Boolean變數 check，用來追蹤file的錯誤，當　檔案內容有發生錯誤，Set check flag為false

b. 步驟二(格式錯誤):

i. 檢查每一行所輸入的element是否超過三個

ii. 若指令為'+'開頭，代表為format4，先將這個符號去除，再繼續判定後面其他的operand是否有多餘的空格

c. (若步驟二沒出現問題，才進行處理輸入形式錯誤):

i. 檢查指令是否合法(指令以非大寫形式呈現)

ii. 檢查指令是否合法(輸入的指令不在指令集內)

iii. operand是否為數字、label、register，以及是不是大寫形式，若出現未定義、非大寫的變數則報錯

iv. 碰到例如:C'EOF'、X'17'等這類的資料，要檢查C和X是否為大寫形式

v. 碰到例如: #4014這類的資料，要檢查#後的資料是否為數字

vi. 碰到例如: LDA @BUFFER這類的資料，要檢查@後是否為memory address的一個label

vii. EQU不能出現Forward Reference

viii. EQU後面的運算元不能出現未定義的Label、Register

ix. EQU的運算元必須得是大寫形式

x. START、RESB、RESW後面接的需為一個數字，以表示相應大寫以及空間

xi. 檢查檔案開頭是否以START開始

xii. 檢查檔案結尾是否以END為終

d. 步驟四(輸出): 如圖7所示

i. 依據檔案出現的錯誤產生相關錯誤、警告和建議修改訊息，此訊息產生在Chrome瀏覽器的Dev Tools中(F12)，倘若檔案沒問題，則沒產生相關訊息

ii. 當檔案內容全部檢查完，return check flag

圖7、Console Log查看錯誤訊息

四、PASS 1

1. 負責工作

Pass 1 的功能，會先載入已經定義好的 OPTAB 以及 Assembler Directives，並且依據使用者輸入的來源檔，產生對應的 symbol table 以及 interFile。錯誤檢查以及表示式(Expression)合法性的部分則是拆給其他同學負責。比較特別的是——這個階段會另外輸出程式的檔名，這是後面其他人負責的程式需要的資訊。

輸入	輸出
• **OPTAB** • **Assembler Directives** • **Source (file content)**	• **SYMTAB** • **interFile** • **ProgramName**

表4、Pass 1 輸入與輸出

➔ 根據老師提供的測試資料進行測試，結果如下圖

```
0: "0000 COPY       START       0"
1: "0000 FIRST      STL         RETADR"
2: "0003 LDB        #LENGTH"
3: "0006 BASE       LENGTH"
4: "0006 CLOOP      +JSUB       RDREC"
5: "000a LDA        LENGTH"
6: "000d COMP       #0"
7: "0010 JEQ        ENDFIL"
8: "0013 +JSUB      WRREC"
9: "0017 J          CLOOP"
10: "001a ENDFIL     LDA         EOF"
11: "001d STA        BUFFER"
12: "0020 LDA        #3"
13: "0023 STA        LENGTH"
14: "0026 +JSUB      WRREC"
15: "002a J          @RETADR"
16: "002d EOF        BYTE        C'EOF'"
17: "0030 RETADR     RESW        1"
18: "0033 LENGTH     RESW        1"
19: "0036 BUFFER     RESB        4096"
20: "1036 BUFEND     EQU         *"
21: "1036 MAXLEN     EQU         BUFEND-BUFFER"
22: "1036 ."
23: "1036 .   SUBROUTINE RDREC"
```

圖8、InterFile

```
BUFEND: "1036"
BUFFER: "0036"
CLOOP: "0006"
COPY: "0000"
ENDFIL: "001a"
EOF: "002d"
EXIT: "1056"
FIRST: "0000"
INPUT: "105c"
LENGTH: "0033"
MAXLEN: "1000"
OUTPUT: "1079"
RDREC: "1036"
REF: "1073"
RETADR: "0030"
RLOOP: "1040"
WLOOP: "1062"
WRREC: "105d"
```

圖9、SYMTAB

2. 特色

- 處理多運算元表示式

○ Pass 1 可處理含有多個運算元的表示式

```
0: "0000 SUM START 0"
1: "0000 P LDS #3"
2: "0003 T EQU *"
3: "0003 LDT #570"
4: "0006 LDS #2"
5: "0009 CONS EQU *"
6: "0009 LDA #4"
7: "000C ITEM1 EQU *"
8: "000C LDX #0"
9: "000F LDT #230"
10: "0012 LDA #10"
11: "0015 ITEM2 EQU *"
12: "0015 STA GAMMA,X"
13: "0018 ITEM3 EQU *"
14: "0079 ITEM4 EQU ITEM2+100"
15: "0073 FIN EQU ITEM4-ITEM1+CONS-T"
16: "0018 ALPHA RESW 190"
17: "0252 BETA RESW 190"
18: "048C GAMMA RESW 190"
19: "06C6 EX WORD ITEM4-ITEM1+CONS"
20: "06C9 NE WORD -5"
21: "06CC END P"
```

圖10、Interfile of multi-operand

```
=SYMTAB=
SUM       0000
P         0000
T         0003
CONS      0009
ITEM1     000C
ITEM2     0015
ITEM3     0018
ITEM4     0079
FIN       0073
ALPHA     0018
BETA      0252
GAMMA     048C
EX        06C6
NE        06C9
```

圖11、SYMTAB of multi-operand

3. 運作流程

如圖12，在逐行讀入 Source file 後，會先進行判斷是否為註解，如果是，將該行編入 InterFile；如果不是，則將內容依照空格切割，並依照長度進行判斷，如果長度為1，則只有可能是單一指令，例如：RSUB、JSUB，如果；如果長度為2，判斷位置2是否為 BASE 或 END，如果是如果長度為3，會先進行判斷位置2是否是 START，原流程圖如下，經過實作後與下圖的過程不一樣，但得出了個人認為不錯的解法，也就沒有進行更改了。

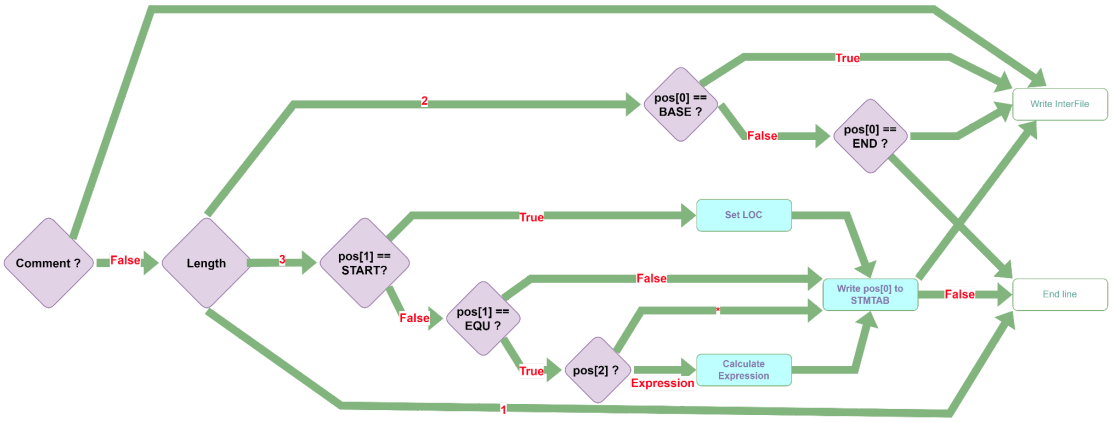


圖12、Pass 1 流程圖

五、PASS 2

1. 簡介

這個部分主要是在將Pass1算完的LOC以及Symbol table進行後續處哩，完善整個應該生成的輸出，包含LST和OBJ，對於Expression和disp計算的錯誤也會在此處查出並跳出錯誤訊息。

在程式內部會進行計算disp(判斷為PC或Base)、處理instruction以及operand的處理，生成出相對應的opcode。

2. 使用介紹

首先會先Upload將欲處理的asm檔放入程式中，上傳後，按鈕會顯示檔案的檔名，在下方還有一個輸入框用來填入輸出檔案的檔名，並在使用者按下Compile之後在右邊的輸出框輸出程式結果LST和OBJ，點擊右側的按鈕就能分別下載LST與OBJ，其檔名即前方的輸入框。

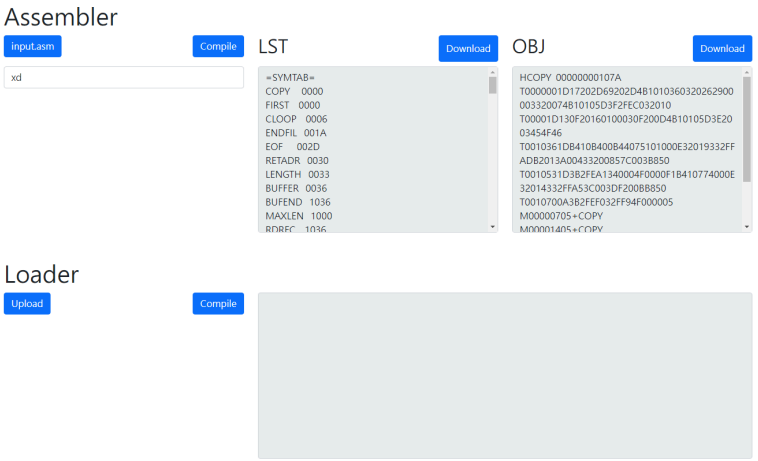


圖13、使用說明1

3. 程式內容

```
554  export function pass2(interFile, SYMTAB, programName, file) {
555      output = []
556      objectCode = ""
557      base = ""
558      symbolType = {}
559      modificationLine = []
560      fileName = file
561      let obj :{interFile2: ..., output: ...}  = pass2_1(interFile, SYMTAB)
562      pass2_2(programName, obj, SYMTAB)
563  }
```

圖14、Pass 2 export function

- 將 Pass1 取得的 interfile 與 SYMTAB 放入 pass2_1 中(計算)
- 將 pass2_1 取得的 obj(interfile2, output) 放入並生成 LST 和 OBJ

4. 函式介紹

```
265  function pass2_1(interFile, SYMTAB) {
266      let interFile2 :any[]  = []
267      output.push('=SYMTAB=')
268  ›   for (let symbol in SYMTAB) {...}
271      output.push('')
272      output.push('=OBJECT CODES=')
273      let index :number  = 1
274      base = ""
275  ›   for (let lineStr of interFile) {...}
329      return {output, interFile2}
330  }
```

圖15、Pass 2_1 function - 1

- pass2_1中主要處理所有計算方面，並含有數個副函式(calNegativeHex, calExpression, calOpcodeniHex, calDispHex, handleDirective, handleOP, handleComment)，objectcode會在此處生成，並且會在生成後交由pass2_2進行檔案的輸出
- calNegativeHex會將十進位負數轉為十六進位補數供後續使用
- calExpression會將operand中label運算的部分計算出結果
- calOpcodeniHex會判斷當前的opcode其為何種模式(indirect, immediate, simple)，並回傳ni
- calDispHex會判斷出此處該使用PC或Base並計算出disp

```
function handleDirective(index, line, directive, operand, SYMTAB) {
    if (directive === 'START') {
        output.push(`${`${index}`.padEnd(2, " ")} LOC=${line[0]}  none`)
    } else if (directive === 'BASE') {
        base = SYMTAB[operand]
        output.push(`${`${index}`.padEnd(2, " ")} LOC=${line[0]}  none`)
    } else if (directive === 'BYTE') {
        objectCode = ''
        if (operand[0] === 'C') {
            // 要儲存字串
            let string = operand.split("'")[1] //取欲儲存的字串
            for (let char :any  of string)
                objectCode += char.charCodeAt(0).toString( radix: 16).toUpperCase() // 將字串轉換為十六進位的ASCII
        } else if (operand[0] === 'X') {
            // 要儲存十六進位
            objectCode = operand.split("'")[1].toUpperCase() //取欲儲存的十六進位(2個十六進位組成一個BYTE)
        }
        output.push(`${`${index}`.padEnd(2, " ")} LOC=${line[0]}  ${objectCode}, BYTE`)
    } else if (directive === 'WORD') {
        if (operand.includes( key: '+') || operand.includes( key: '-')) {
            let expression = operand
            objectCode = calExpression(expression, SYMTAB).padStart(6, '0')
        } else {
            objectCode = parseInt(operand).toString( radix: 16).toUpperCase().padStart(6, '0')
        }
        output.push(`${`${index}`.padEnd(2, " ")} LOC=${line[0]}  ${objectCode}, WORD`)
    } else {
        // 處理 RESB RESW EQU END
        output.push(`${`${index}`.padEnd(2, " ")} LOC=${line[0]}  none`)
    }
}
```

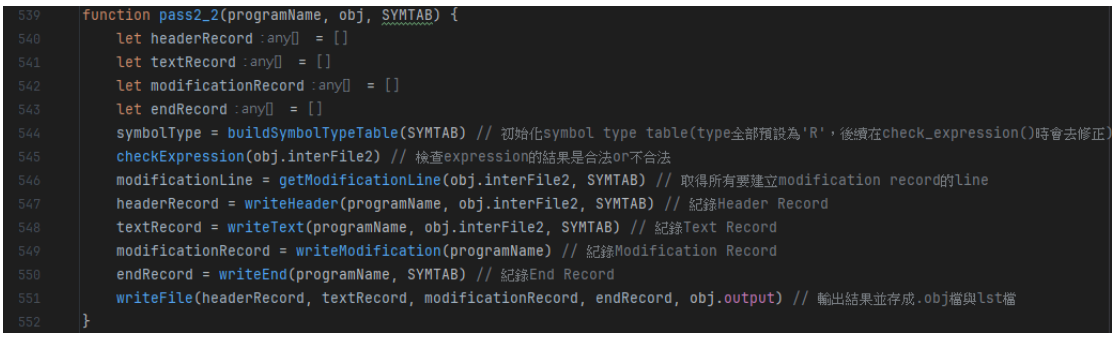
圖16、Pass 2_1 function - 2

- handleDirective會將指令(START, BASE, BYTE, WORD)進行相對應的處理

```
function handleOP(index, line, pc, base, mnemonic, operand, SYMTAB) {
    let isFormat2 :number  = 0
    let isFormat4 :number  = 0
    let isPC :number  = 0
    let isBase :number  = 0
    let isError :number  = 0
    let isImmediate :number  = 0
    let isIndirect :number  = 0
    let isIndexed :number  = 0

    let loc = line[0]
    let xbpeHex :string  = 'A'

    // format 2
    if (config.Format2_mnemonic.includes(mnemonic)) {...}
    // format 4
    else if (mnemonic[0] === '+') {...}

    // immediate 、 indirect 與 indexed 為互斥的
    if (operand[0] === '#') {...} else if (operand[0] === '@') {...} else if (operand.includes( key: ',X') && !isFormat2) {...}
    if (isFormat2) {...} else if (isFormat4) {...} else {...}
    let nixbpe
    if (isFormat2)
        nixbpe = ''
    else if (!isIndirect && !isImmediate) {
        // simple addressing(n=1,i=1)
        nixbpe = `, nixbpe=11${parseInt(xbpeHex,  radix: 16).toString( radix: 2).padStart(4, '0')}`  // 11xbpe
    } else {
        nixbpe = `, nixbpe=${isIndirect}${isImmediate}${parseInt(xbpeHex,  radix: 16).toString( radix: 2).padStart(4, '0')}`
    }  // ??xbpe
    let mode :string  = isFormat2 ? 'format2' : (isFormat4 ? 'format4' : (isPC ? 'pc-relative' : (isBase ? 'base-relative' : 'none')))
    output.push(`${`${index}`.padEnd(2, " ")} LOC=${loc}  ${objectCode}, ${mode}${nixbpe}`)
}
```

圖17、Pass 2_1 function - 3

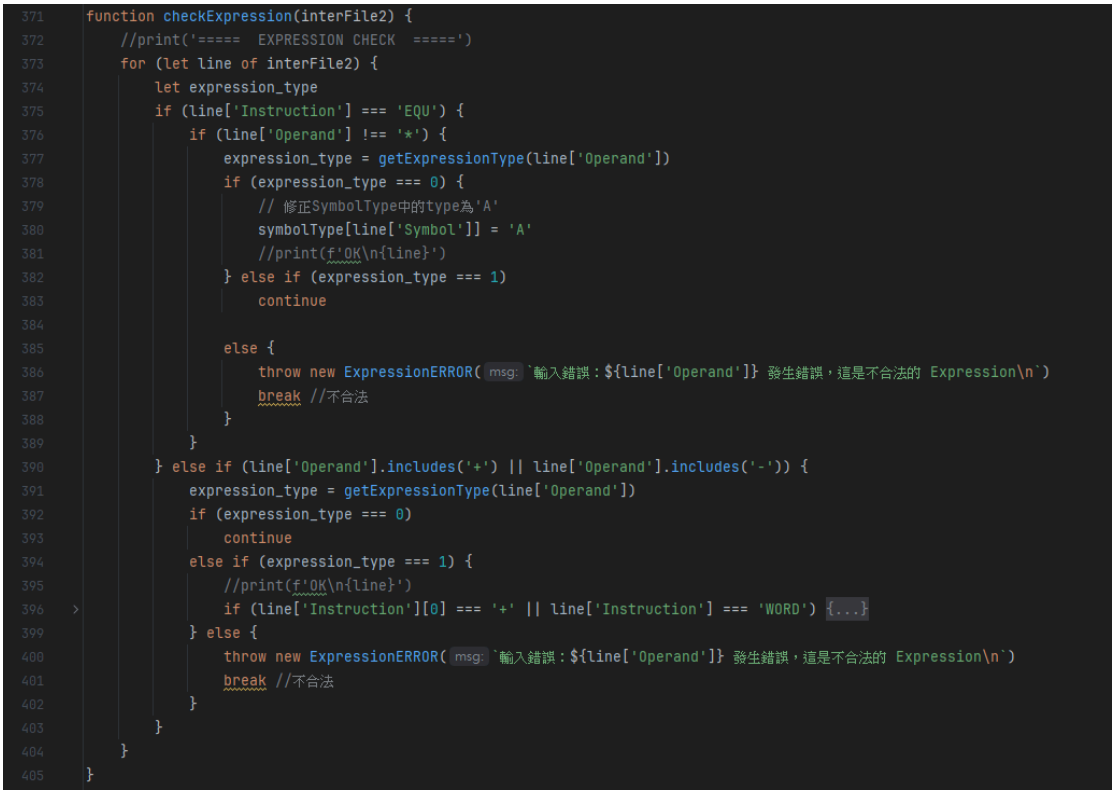
- handleOP會處理指令，並對operand進行操作，其中也會判斷其format並進行不同運算，並在此處將nixbpe、disp處理完成，生成出完整的opcode
- handleComment會處理註解



```
function pass2_2(programName, obj, SYMTAB) {
    let headerRecord :any[]  = []
    let textRecord :any[]  = []
    let modificationRecord :any[]  = []
    let endRecord :any[]  = []
    symbolType = buildSymbolTypeTable(SYMTAB) // 初始化symbol type table(type全部預設為'R'，後續在check_expression()時會去修正)
    checkExpression(obj.interFile2) // 檢查expression的結果是合法or不合法
    modificationLine = getModificationLine(obj.interFile2, SYMTAB) // 取得所有要建立modification record的line
    headerRecord = writeHeader(programName, obj.interFile2, SYMTAB) // 紀錄Header Record
    textRecord = writeText(programName, obj.interFile2, SYMTAB) // 紀錄Text Record
    modificationRecord = writeModification(programName) // 紀錄Modification Record
    endRecord = writeEnd(programName, SYMTAB) // 紀錄End Record
    writeFile(headerRecord, textRecord, modificationRecord, endRecord, obj.output) // 輸出結果並存成.obj檔與lst檔
}
```

圖18、Pass 2_1 function - 4

- pass2_2會將pass2_1生成出的檔案轉換成OBJ，並將LST和OBJ生成與輸出，其中含有數個副函式(buildSymbolTypeTable, getExpressionType, checkExpression, getModificationLine, writeHeader, writeText, writeModification, writeEnd, writeFile)
- buildSymbolTypeTable建立一存入symboltype的表供後續使用
- getExpressionType將symboltype的表填入正確的值



```
function checkExpression(interFile2) {
    //print('=====  EXPRESSION CHECK  =====')
    for (let line of interFile2) {
        let expression_type
        if (line['Instruction'] === 'EQU') {
            if (line['Operand'] !== '*') {
                expression_type = getExpressionType(line['Operand'])
                if (expression_type === 0) {
                    // 修正SymbolType中的type為'A'
                    symbolType[line['Symbol']] = 'A'
                    //print(f'OK\n{line}')
                } else if (expression_type === 1)
                    continue

                else {
                    throw new ExpressionERROR( msg: `輸入錯誤：${line['Operand']} 發生錯誤，這是不合法的 Expression\n`)
                    break //不合法
                }
            }
        } else if (line['Operand'].includes('+') || line['Operand'].includes('-')) {
            expression_type = getExpressionType(line['Operand'])
            if (expression_type === 0)
                continue
            else if (expression_type === 1) {
                //print(f'OK\n{line}')
                if (line['Instruction'][0] === '+' || line['Instruction'] === 'WORD') {...}
            } else {
                throw new ExpressionERROR( msg: `輸入錯誤：${line['Operand']} 發生錯誤，這是不合法的 Expression\n`)
                break //不合法
            }
        }
    }
}
```

圖19、Pass 2_1 function - 5

- checkExpression檢查expression是否正確，錯誤則會拋出ExpressionERROR
- getModificationLine會取得需要進行modify的行數
- writeHeader, writeText, writeModification, writeEnd會將pass2_1生成出的檔案一一轉為在OBJ檔中需要存在的格式並分別寫入檔案中供writeFile處理
- writeFile會將LST與OBJ檔的內容填入文本框中，使按鈕點選時能夠取得相對應的內容並下載

5. 錯誤處理

```
function PCBaseERROR(msg) {
    this.name = 'PCBaseERROR'
    this.msg = msg
}

Show usages
function ExpressionERROR(msg) {
    this.name = 'ExpressionERROR'
    this.msg = msg
}

PCBaseERROR.prototype = new Error()
PCBaseERROR.prototype.constructor = PCBaseERROR

ExpressionERROR.prototype = new Error()
ExpressionERROR.prototype.constructor = ExpressionERROR
```

```
            if (isError) {
                throw new PCBaseERROR( msg: `輸入錯誤：第${index}行發生錯誤，PC Relative 與 Base Relative 皆無法使用\n`)
            }
        }
```

```
                else {
                    throw new ExpressionERROR( msg: `輸入錯誤：${line['Operand']} 發生錯誤，這是不合法的 Expression\n`)
                    break //不合法
                }
```

```
    try {
        const { Interfile : [] , SYMTAB : {} , ProgramName } = pass1(asmContent);
        let fileName = fileNameContent.value;
        pass2(Interfile, SYMTAB, ProgramName, fileName);
    } catch (e) {
        console.log(e)
        if (e.name === "PCBaseERROR") {
            console.log(e.msg)
            console.log("程式因錯誤暫停執行!!")
        } else if (e.name === "ExpressionERROR") {
            console.log(e.msg)
            console.log("程式因錯誤暫停執行!!")
        } else {
            console.log("程式因不明錯誤暫停執行!!")
        }
    }
```

圖20、錯誤處理 function

- 如果程式發生錯誤，其會throw回主程式，並輸出錯誤訊息

6. JS部分

```
lstDownload.addEventListener( type: 'click', listener: () => {
    let blob :Blob  = new Blob( blobParts: [showLst.textContent], {type: "application/octet-stream"...})
    let url :string  = window.URL.createObjectURL(blob);
    let a :HTMLAnchorElement  = document.createElement( tagName: 'a');
    a.href = url;
    a.download = `${fileName}.lst`;
    document.body.appendChild(a);
    a.click();
    window.URL.revokeObjectURL(url);
    document.body.removeChild(a);
})

objDownload.addEventListener( type: 'click', listener: () => {
    let blob :Blob  = new Blob( blobParts: [showObj.textContent], {type: "application/octet-stream"...})
    let url :string  = window.URL.createObjectURL(blob);
    let a :HTMLAnchorElement  = document.createElement( tagName: 'a');
    a.href = url;
    a.download = `${fileName}.obj`;
    document.body.appendChild(a);
    a.click();
    window.URL.revokeObjectURL(url);
    document.body.removeChild(a);
})
```

圖21、JS

- 其主要處理按鈕點擊時的操作，點擊後會取得文本框中的內容並開始下載

六、組譯器主程式說明

```javascript
import {error_scan} from "./Error_Scan.js";
import {pass1} from "./Pass1.js";
import {pass2} from "./Pass2.js";

let asmContent = "";
const asmUploader = document.querySelector('#asmUploader');
const asmName = document.querySelector('#asmName');
const fileNameContent = document.querySelector('#fileNameContent');
asmName.addEventListener('click',()=>{
    asmUploader.click();
});

asmUploader.addEventListener('change', (e) => {
    const file = e.target.files[0];
    asmName.textContent = e.target.files[0].name;
    if (file && file.type === 'text/plain') {
        const reader = new FileReader();
        reader.readAsText(file);
        reader.onload = function (e) {
            asmContent = e.target.result;
        };
    } else {
        alert('請上傳文字檔');
    }
});
```

```javascript
function main() {
    if (error_scan(asmContent)) {
        try {
            const { Interfile, SYMTAB, ProgramName } = pass1(asmContent);
            let fileName = fileNameContent.value;
            pass2(Interfile, SYMTAB, ProgramName, fileName);
        } catch (e) {
            console.log(e);
            if (e.name === "PCBaseERROR") {
                console.log(e.msg);
                console.log("程式因錯誤暫停執行!!");
            } else if (e.name === "ExpressionERROR") {
                console.log(e.msg);
                console.log("程式因錯誤暫停執行!!");
            } else {
                console.log("程式因不明錯誤暫停執行!!");
            }
        }
    } else {
        console.log("程式因錯誤暫停執行!!");
    }
}

window.main = main;
```

圖22、組譯器主程式

圖22為組譯器的主程式，第1到第3行為呼叫 Error Scan、Pass 1、Pass 2的export funtion；第5至第25行為監聽事件，用來上傳 Source File，第27行開始，則是組譯器的 main function，將 Source File 的內容，存入變數 asmContent中，

首先進行 Error Scan 的錯誤偵測，當沒有發現任何錯誤時，執行 Pass 1 的 function，回傳的參數為 Interfile、SYMTAB、ProgramName，再將這些參數傳入Pass 2，得到結果。在執行 Pass 1 和 Pass 2 時，若發現錯誤，將錯誤結果輸出至控制台日誌中。

七、LOADER

因為是使用JavaScript來進行製作的關係，無法直接使用記憶體，因此我們的Loader會模擬對記憶體進行操作在每次Compile的時候會先隨機生成一段整數作為記憶體位置，並根據這個數值進行後續操作。

此程式主要是將obj檔中Modify的部分放進Text，並將處理好的Text轉換為Binary Form供記憶體讀取。

1. 使用介紹

首先會先Upload將欲處理的obj檔放入程式中，上傳後，按鈕會顯示檔案的檔名，並在使用者按下Compile之後在右邊的輸出框輸出程式結果。

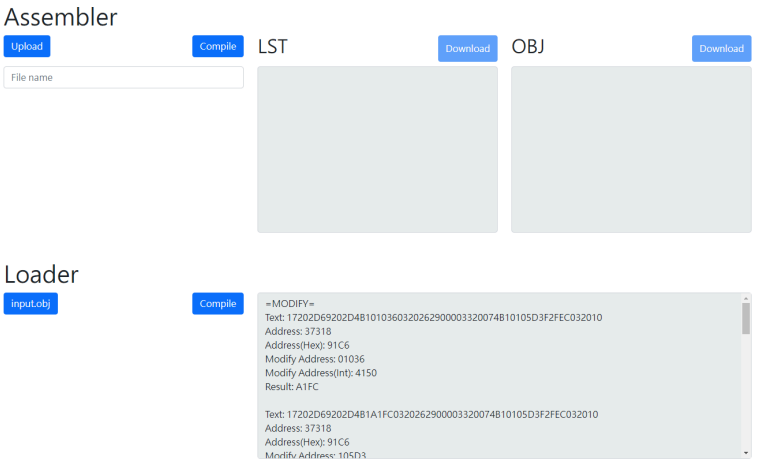


圖23、使用說明2

2. 使用介紹

```
function loader() {
    showLoader.textContent = ""
    generateAddress()
    let obj :string[]  = objContent.split( splitter: /\r?\n/)
    textContent = []
    output.push("=MODIFY=")
    for (let line :string  of obj) {
        if (line[0] === 'H')
            continue
        else if (line[0] === 'T') {
            handleText(line)
        } else if (line[0] === 'M') {
            handleModified(parseInt(line.substring(1, 7),  radix: 16), parseInt(line.substring(7, 9),  radix: 16))
        } else if (line[0] === 'E')
            break
    }

    output.push("=MODIFY TEXT=")
    for (let item of textContent)
        output.push(item.content)

    output.push('')
    output.push("=BINARY FORM=")
    for (let item of textContent) {
        let bin :string  = ""
        for (let hex :any|string  of item.content) {
            bin += parseInt(hex,  radix: 16).toString( radix: 2).padStart(4, '0')
        }
        output.push(bin)
        output.push('')
    }

    for (let item of output) {
        showLoader.textContent += item + '\n'
    }
}
```

圖24、Loader function 1

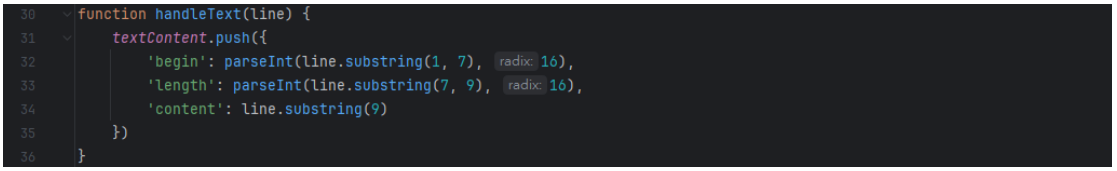
- 生成隨機記憶體位置
- 將objContent內容逐行分割存進obj陣列
- 在迴圈對陣列逐行進行處理，迴圈中會判斷其類型進行對應的處理
- Output是欲放入文本框的內容

3. 函式介紹

```
function generateAddress() {
    address = Math.floor( x: Math.random() * 999999) + 1
}
```

圖25、Loader function 2

- 生成一隨機數，並令其大小在1 ~ 1000000之間，作為記憶體值



```
function handleText(line) {
    textContent.push({
        'begin': parseInt(line.substring(1, 7), radix: 16),
        'length': parseInt(line.substring(7, 9), radix: 16),
        'content': line.substring(9)
    })
}
```

圖26、Loader function 3

- 對Text進行處理，將其分為三個字串
  - begin是該content中byte的起始位置
  - length是content總共有幾個byte
  - content中則為該行的所有operand

```
function handleModified(begin, length) {
    for (let item of textContent) {
        if (begin < item.begin + item.length) {
            output.push(`Text: ${item.content}`)
            output.push(`Address: ${address}`)
            output.push(`Address(Hex): ${address.toString( radix: 16).toUpperCase()}`)
            if (length % 2 === 1)
                begin = (begin - item.begin) * 2 + 1

            let modifyAddr :string  = item.content.substring(begin, begin + length)
            output.push(`Modify Address: ${modifyAddr}`)
            let intModifyAddr :number  = parseInt(modifyAddr, radix: 16)
            output.push(`Modify Address(Int): ${intModifyAddr}`)
            let result :string  = (address + intModifyAddr).toString( radix: 16).toUpperCase()
            output.push(`Result: ${result}`)
            item.content = item.content.substring(0, begin) + result + item.content.substring(begin + length)
            output.push('')
            break
        }
    }
}
```

圖27、Loader function 4

- 將Modify中所標記的位置放回Text中，其應將Text中原有的內容與記憶體位置相加算出其應放回的數值，並將其放入

4. 輸入輸出

- 輸入：
  - HCOPY 00000000107A
  - T0000001D17202D69202D4B10103603202629000033200 74B10105D3F2FEC032010
  - T00001D130F20160100030F200D4B10105D3E2003454F46
  - T0010361DB410B400B44075101000E32019332FFADB2013A00433200857C003B850

    - T0010531D3B2FEA1340004F0000F1B410774000E32014332FFA53C003DF200BB850
    - T0010700A3B2FEF032FF94F000005
    - M00000705+COPY
    - M00001405+COPY
    - M00002705+COPY
    - M00103D05+COPY
    - E000000
- 輸出:
    - =MODIFY=
    - Text: 17202D69202D4B1010360320262900003320074B10105D3F2FEC032010
    - Address: 37318
    - Address(Hex): 91C6
    - Modify Address: 01036
    - Modify Address(Int): 4150
    - Result: A1FC
    -
    - Text: 17202D69202D4B1A1FC03202629000033200 74B10105D3F2FEC032010
    - Address: 37318
    - Address(Hex): 91C6
    - Modify Address: 105D3
    - Modify Address(Int): 67027
    - Result: 19799
    -
    - Text: 0F20160100030F200D4B10105D3E2003454F46
    - Address: 37318
    - Address(Hex): 91C6
    - Modify Address: 0105D
    - Modify Address(Int): 4189
    - Result: A223
    -
    - Text: B410B400B44075101000E32019332FFADB2013A00433200857C003B850

- Address: 37318
- Address(Hex): 91C6
- Modify Address: 01000
- Modify Address(Int): 4096
- Result: A1C6
-
- =MODIFY TEXT=
- 17202D69202D4B1A1FC032026290000332007 4B1019799F2FEC032010
- 0F20160100030F200D4B1A2233E2003454F46
- B410B400B440751A1C6E32019332FFADB2013A00433200857C003B850
- 3B2FEA1340004F0000F1B410774000E32014332FFA53C003DF200BB850
- 3B2FEF032FF94F000005
-
- =BINARY FORM=
- 000101110010000000101101011010010010000000101101010010110001101000011111110000000011001000000010011000101001000000000000000000011001100100000000011101001011000100000001100101111001100111110010111111101100000000110010000000010000
-
- 000011110010000000010110000000010000000000000001100001111001000000000110101000101010011110100100110
-
- 101101000001000010110100000000001011010001000000011101010001101000011100011011100011001000000001100100110011001011111111101011011011001000000001001110100000000001000011001100100000000010000101011111000000000000011101110000001010000
-
- 001110110010111111101010000100110100000000000000010011110000000000000000001111000011011010000010000011101110100000000000000111000110010000000101000011001100101111111110100101001111000000000000111101111100100000000010111011110000101000 0
-

- 001110110010111111101111000000110010111111111001010011110000000000000000000000101

八、程式碼連結

- https://github.com/WilsonCWS/SIC-XE-Assembler-Loader

九、參考文獻

1. Tutorials Point. "Assembly Language - Tutorials Point." [Online]. Available: https://www.tutorialspoint.com/assembly_programming/index.htm. [Accessed: June 21, 2024].