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**](#_heading=h.eptm3u17qdb5) **4**

[**二、測資蒐集與答案撰寫**](#_heading=h.vdpm4q910ths) **8**

[**三、ERROR SCAN**](#_heading=h.1yo6998kfyyu) **10**

[**四、PASS 1**](#_heading=h.nw4ecf7fa8er) **13**

[**五、PASS 2**](#_heading=h.5l7cmpf8c7uv) **16**

[**六、組譯器主程式說明**](#_heading=h.ewpmi68sll00) **23**

[**七、LOADER**](#_heading=h.nik7b1jw6vg0) **24**

[**八、程式碼連結**](#_heading=h.gg298wecpiy1) **29**

[**九、參考文獻**](#_heading=h.zbwmi7q7krgw) **29**

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 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。

| 圖４、config.json |
| --- |

# 二、測資蒐集與答案撰寫

## 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。

## 製作輸出的檔案

| 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

## 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。

## 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):

* + 1. 步驟一:讀取檔案
    2. 步驟二:檢查每行格式
    3. 步驟三:檢查每行指令和Operand
    4. 步驟四:輸出錯誤、警告和修改建議訊息

| 一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片  自動產生的描述  圖６、Error Scan架構圖 |
| --- |

## 詳細流程講解

## 步驟一(讀檔)

* + 1. 讀取檔案，這個檔案的副檔名可以是.asm、.txt等。
    2. 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。
    3. 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
  1. 步驟二(格式錯誤):
     1. 檢查每一行所輸入的element是否超過三個
     2. 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
  2. (若步驟二沒出現問題，才進行處理輸入形式錯誤):
     1. 檢查指令是否合法(指令以非大寫形式呈現)
     2. 檢查指令是否合法(輸入的指令不在指令集內)
     3. 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
     4. 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
     5. 碰到例如: #4014這類的資料，要檢查#後的資料是否為數字
     6. 碰到例如: LDA @BUFFER這類的資料，要檢查@後是否為memory address的一個label
     7. EQU不能出現Forward Reference
     8. EQU後面的運算元不能出現未定義的Label、Register
     9. EQU的運算元必須得是大寫形式
     10. START、RESB、RESW後面接的需為一個數字，以表示相應大寫以及空間
     11. 檢查檔案開頭是否以START開始
     12. 檢查檔案結尾是否以END為終
  3. 步驟四(輸出): 如圖７所示
     1. 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
     2. 當檔案內容全部檢查完，return check flag

| 一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片  自動產生的描述  圖７、Console Log查看錯誤訊息 |
| --- |

# 四、PASS 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**

| 圖８、InterFile | 圖９、SYMTAB |
| --- | --- |

## 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

| 圖１０、Interfile of multi-operand | 圖１１、SYMTAB of multi-operand |
| --- | --- |

## 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

## 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。

## 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 |
| --- |

## 程式內容

| 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

## 函式介紹

| 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

| 圖１６、Pass 2\_1 function - ２ |
| --- |

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

| 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

| 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

| 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

## 錯誤處理

| 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

## JS部分

| 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

|  |
| --- |
| 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。

## 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 |
| --- |

## 使用介紹

| 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

## 函式介紹

| 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

| 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

| 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

## 輸入輸出

* 輸入：
  + HCOPY 00000000107A
  + T0000001D17202D69202D4B1010360320262900003320074B10105D3F2FEC032010
  + 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: 17202D69202D4B1A1FC0320262900003320074B10105D3F2FEC032010
  + 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=
  + 17202D69202D4B1A1FC0320262900003320074B1019799F2FEC032010
  + 0F20160100030F200D4B1A2233E2003454F46
  + B410B400B440751A1C6E32019332FFADB2013A00433200857C003B850
  + 3B2FEA1340004F0000F1B410774000E32014332FFA53C003DF200BB850
  + 3B2FEF032FF94F000005
  + =BINARY FORM=
  + 000101110010000000101101011010010010000000101101010010110001101000011111110000000011001000000010011000101001000000000000000000110011001000000000011101001011000100000001100101111001100111110010111111101100000000110010000000010000
  + 0000111100100000000101100000000100000000000000110000111100100000000011010100101100011010001000100011001111100010000000000011010001010100111101000110
  + 101101000001000010110100000000001011010001000000011101010001101000011100011011100011001000000001100100110011001011111111101011011011001000000001001110100000000001000011001100100000000010000101011111000000000000111011100001010000
  + 0011101100101111111010100001001101000000000000000100111100000000000000001111000110110100000100000111011101000000000000001110001100100000000101000011001100101111111110100101001111000000000000111101111100100000000010111011100001010000
  + 00111011001011111110111100000011001011111111100101001111000000000000000000000101

# 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.](https://www.tutorialspoint.com/assembly_programming/index.htm) [Accessed: June 21, 2024].