**SIC組譯器 (Assembler)文件**

**一、簡介**

本組譯器的實作目標在於為 **SIC/SICXE** 架構（或類似的教學用架構）提供一個基本而完整的「兩遍式」(Two-Pass) 組譯器。透過此組譯器可以將組合語言程式轉換為目標碼 (Object Code)，並產生對應的**清單檔** (List File) 與**物件檔** (Object File)。

在此實作中，我們針對簡化的指令集做實作：包括 **Format 1**（如部分教學不常見）、**Format 2**（多用於寄存器指令）、**Format 3**（最常見之三位元組合尋址），並增加對常見組譯器 Directive（如 START, END, BYTE, WORD, RESB, RESW 等）的支援。

整個組譯器使用 C 語言開發，針對 Linux 或其他 POSIX 系統編譯即可順利執行。

**二、功能與流程概述**

**1. 兩遍式 (Two-Pass) 組譯流程**

1. **Pass1：建立符號表 (Symbol Table)**
   * 讀取每一行組合語言程式，若該行含標籤 (Label)，則將其符號名稱及當前位置 (Location Counter, LC) 填入符號表。
   * 判斷指令或 Directive 是否會佔據實際空間（如 BYTE, WORD, RESB, RESW, Format 2, Format 3...），並更新 LC 以符合下一條指令或資料定義的位址。
   * 遇到 START 則設定程式起始位址 (start\_addr)；若未出現 START，則預設為 0x0000。
   * 遇到 END 之後，基本上即可停止「主要程式段」的 LC 計算。但在教學環境下，通常會允許 END 出現後繼續讀入剩餘行（若程式碼仍有一些資料定義），並且更新 LC。
2. **Pass2：產生機器碼 (Object Code) 與輸出**
   * 根據 Pass1 所建立的符號表，重新讀取各行組合語言程式。
   * 遇到指令（如 LDA, ADD, COMP...）時，會檢查其操作碼、尋址模式 (立即, 間接, 簡單, 索引) 並計算目標位址。
   * Format 2 指令僅佔 2 Bytes：1 Byte Opcode + 1 Byte (兩個 4 bits 寄存器代號)
   * Format 3 指令佔 3 Bytes：1 Byte Opcode + 4 bits + 12 bits 位址 (其中 4 bits 可能是 n, i, x 三位元的組合)。
   * BYTE, WORD 若有立即常數或字串常數，就轉成對應的十六進位值存於 Object Code；RESB, RESW 則不產生機器碼但會在 LC 佔空間。
   * END 指令時，停止主要組譯步驟，不再產生任何機器碼。

完成上述兩遍後，即可產生：

1. **物件檔** (Object File)：含 H (Header), T (Text) 與 E (End) 等 Record，可供 Loader 載入執行。
2. **清單檔** (List File)：列出每一行的位址、標籤、助記符、操作元與對應的物件碼，方便除錯與教學使用。

**2. 功能特色**

1. **支援常見 Directive**
   * START, END：程序起始與結束
   * BYTE, WORD：定義常數
   * RESB, RESW：保留空間
   * 其他如 ORG, EQU, CSECT 等可擴充或僅部分支援
2. **格式 2 與格式 3 指令**
   * Format 2：有寄存器操作，如 CLEAR A, ADDR R1, R2...
   * Format 3：支援 # (立即)、@ (間接) 與 ,X (索引) 尋址模式，並能以位元 (n, i, x) 進行組合。
3. **符號表檢查**
   * Pass1 建立符號表時，若遇到重複定義，會主動報錯。
   * Pass2 若找不到符號，亦會報錯 Undefined symbol。
4. **簡易的錯誤處理**
   * 偵測到無效指令、格式不正確之立即值或十六進位字符串、重複符號、未定義符號時，會在終端印出錯誤訊息。
5. **T Record 自動分段**
   * 在產生物件檔時，每個 T Record 最多允許 **30 bytes** 的機器碼；若超過則自動拆分為多條 T Record，以符合 SIC/XE 物件檔載入規範。

**三、程式架構與程式碼說明**

**1. 檔案結構**

此組譯器整合在單一檔 assembler.c 中，使用下列函式區塊作為邏輯分工：

1. **initialize\_opcode\_map()**
   * 建立 **Opcode** 對應表，例如 LDA -> 00, ADD -> 18, CLEAR -> B4，含格式 2 與格式 3 常用指令。
2. **initialize\_register\_map()**
   * 建立寄存器對應表，例如 A -> 0, X -> 1, L -> 2...等。
3. **pass1(FILE \*fp)**
   * 負責掃描原始組合語言程式，根據指令長度或 Directive 分配位址 (LC)，並將標籤放進 **Symbol Table**。
   * 若偵測到 START，設定 start\_addr；若發現 END，則記錄該行位置。
4. **pass2()**
   * 依據符號表解析標籤位址，計算每條指令的 Object Code；
   * 處理 Format 2 (寄存器) 與 Format 3 (n, i, x) 位元組合；
   * BYTE, WORD 的常數轉換，並將結果存入 lines[i].object\_code。
5. **generate\_object\_file(const char \*obj\_filename)**
   * 輸出物件檔：
     + H Record (程式名稱、起始位址、程式長度)
     + T Record (每段最多 30 bytes)，將各行的 Object Code 串起來，並自動分段。
     + E Record (程式入口起始地址)
6. **generate\_list\_file(const char \*lst\_filename)**
   * 產生清單檔，列出「Address、Label、Mnemonic、Operand、Object Code」，方便除錯與學習。
7. **main(int argc, char \*argv[])**
   * 讀取命令列參數 <input\_file> <output\_obj> <output\_lst>
   * 呼叫 pass1() → pass2() → 分別產生物件檔與清單檔 → 程式結束。

**2. 部分程式碼重點**

以下簡單摘錄關鍵程式片段以供參考。詳細完整程式，請見附錄：

**(1) Pass1 計算長度**

一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

自動產生的描述

**(2) Pass2 產生機器碼**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述 **(3) 產生物件檔 (Object File)**

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

**四、使用方式**

**執行**

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

例如：

./assembler test.asm test.obj test.lst

1. **檢視結果**
   * test.obj：產生之 SIC/XE 物件檔，可以用相容的 Loader / 模擬器載入執行。
   * test.lst：清單檔，包含每一行的組譯結果與機器碼，利於除錯與教學參考。

**五、常見問題與後續擴充**

1. **為何物件檔的 T-Record 會自行分段？**  
   SIC/XE 規範每條 T Record 最多容納 30 Bytes。若程式碼較大，就會出現多條 T Record，各自標示起始位址與長度。
2. **END 指令之後還能有程式碼嗎？**  
   一般教學用法習慣 END 為最後，但有些情況下 (如 CSECT 或其他段) 仍可放一些資料定義。主要需看組譯器是否支援多段程式與多 CSECT。
3. **若想支援 Format 4 (+ 指令)，該怎麼做？**  
   在 Pass1 時須檢查 + 字元，以 4 bytes 計算指令長度；Pass2 時則在計算位址時支援 20 bits，也可考慮 BASE / PC-relative / extended 位址等進階特性。
4. **是否有支援 Re-locatable (重定位) 功能？**  
   目前簡化版本未實作 Modification Record；若要支援程式重定位，就須記錄位址中哪些 bits 需 Loader 做相對位移。

**六、結論**

本組譯器展示了一個教學版 **SIC/SICXE** 組譯器從零開始的完整流程，包含：

* **Pass1**：建立符號表
* **Pass2**：產生機器碼
* **輸出物件檔**：包含 H、T、E 記錄
* **產生清單檔**：顯示每一行對應的機器碼

藉由這個專案，讀者可以深入瞭解組合語言、目標碼與 Loader 之間的關係，也能快速體會到底層作業系統（或模擬器）是如何載入與執行程式。  
若需要更完整的功能，如 **Format 4**、**Base/PC 相對位址**、**Modification Record** 等，亦可在此基礎上進一步擴充。

此文件期盼能讓開發者與使用者瞭解程式內部運作與如何編譯、執行、除錯。如有任何疑問或改進建議，歡迎參考此程式設計架構，在程式碼中自行調整或優化。

**參考範疇**

* **Systems Software: An Introduction to Systems Programming** (Leland L. Beck)
* **SIC/XE Instruction Set**：教學版簡化指令架構，用於展示組合語言、組譯器原理
* **GNU Make / GCC**：編譯、建構工具