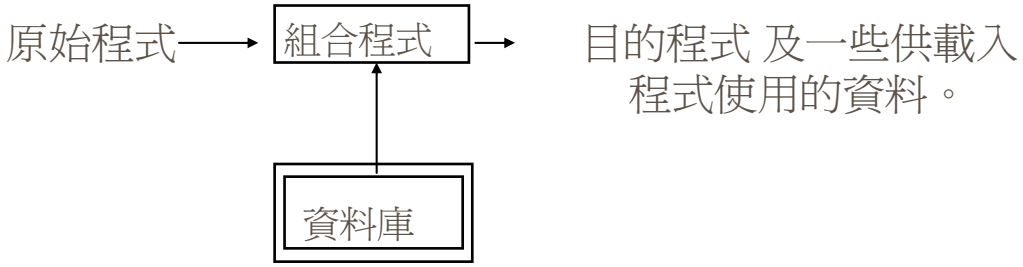
組合程式



- (1) 將助記憶式指令轉換成相對應的機器碼。
- (2) 將符號式運算元轉換成相對應的機器位址。
- (3) 依電腦硬體格式產生相對應的機器指令。
- (4) 將原始程式內定義的資料常數轉換成機器內部表示的型態。
- (5) 輸出目的程式並列出組合語言

組合程式之類型

組合程式依處理方式可以分成三大類：

- (1) 一次處理組合程式

  a. 載入並且執行組合程式

  b.一次處理模組組合程式
- (2) 兩次處理組合程式
- (3) 多次處理組合程式

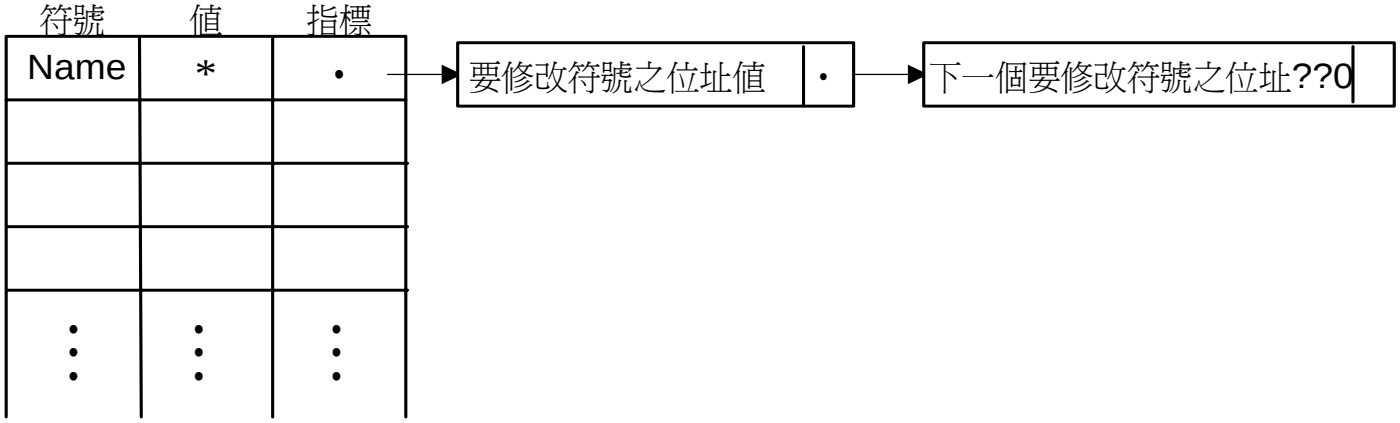
- 一次處理組合程式 (One-Pass Assembler) -
- 若程式中所有使用到的符號皆事先定義過，且沒有向前參考 (Forward Reference) 的情況，皆可用一次處理組合程式來處理。此類的組合程式有兩種：

  (1) 載入並且執行組合程式 (Load-And-Go Assembler)

  (2) 一次處理模組組合程式 (One-Pass-Module Assembler)

- a. 載入並且執行組合程式
- 此種組合程式並不產生目的程式 ，而是直接產生絕對位址式的機器碼，並且立即載入實際的記憶體中執行。

- 載入並且執行組合程式處理符號的方式則採用鏈結串列，當未定義的符號首次出現時，則將它加於符號表內，並註明上未定義；當此符號再次出現，則利用指標將它連接能來，並記錄此符號的出現位址值。待此符號的定義指令出現時，即刻將定義的值入符號表，再經由指標找到其後尚未定義符號的位址，並填入定義的值。這種做法稱爲回溯法 (Backtracking)。



<缺點>

- a. 不具連結的能力：

  個別組譯的程式無法結合在一起，成爲一個大程式。

- b. 不具重定位的能力：

  程式必須在固定的記憶體位址組譯執行

- c. 需具備足夠的記憶體空間：

  因爲組合程式本身與機器碼都同時存於記憶體中。

- 一次處理模組組譯程式 (One-Pass Module Assembler)-
- 此種組合程式不若載入並且執行組合程式，它並不產生機器碼，而是產生目的碼，其中亦包含供連結程式及載入程式使用的資訊。一次處理模組組合程式僅完成部份的組譯工作，其餘則交由載入程式處理。

- 此組合程式處理未定義的符號 (指外在符號) 是以一個跳躍至某一位址的指令替代；即遇到未定義的符號，則將它的位址值加入跳前表中，且記錄它的符號及其出現的位址值，當再有相同的符號出現時，則用鏈結串列的方式將它的位址值串連，此表格將供載入程式於載入時調整相對應的位址值。這種做法稱之爲轉移向量 (Transfer Vector)。

- 兩次處理組合程式 (Two-Pass Assembler)
  —
- 在做組譯工作時，會對原始程式做兩次掃瞄，所以允許程式中存在向前參考的未定義符號。簡單來說，典型的兩次處理組合程式其功能就是在第一次處理 (Pass 1) 時產生一個中間檔案，而於第二次處 (Pass 2) 時讀取該中間檔案，並處理第一次處理未完成的工作。

第一次處理 ：定義所有的符號 與文字 。

- a. 決定程式中所有敘述的位址。
- b. 儲存所有標籤的位址，供第二次處理使用。
- c. 處理程式中的虛擬指令 。

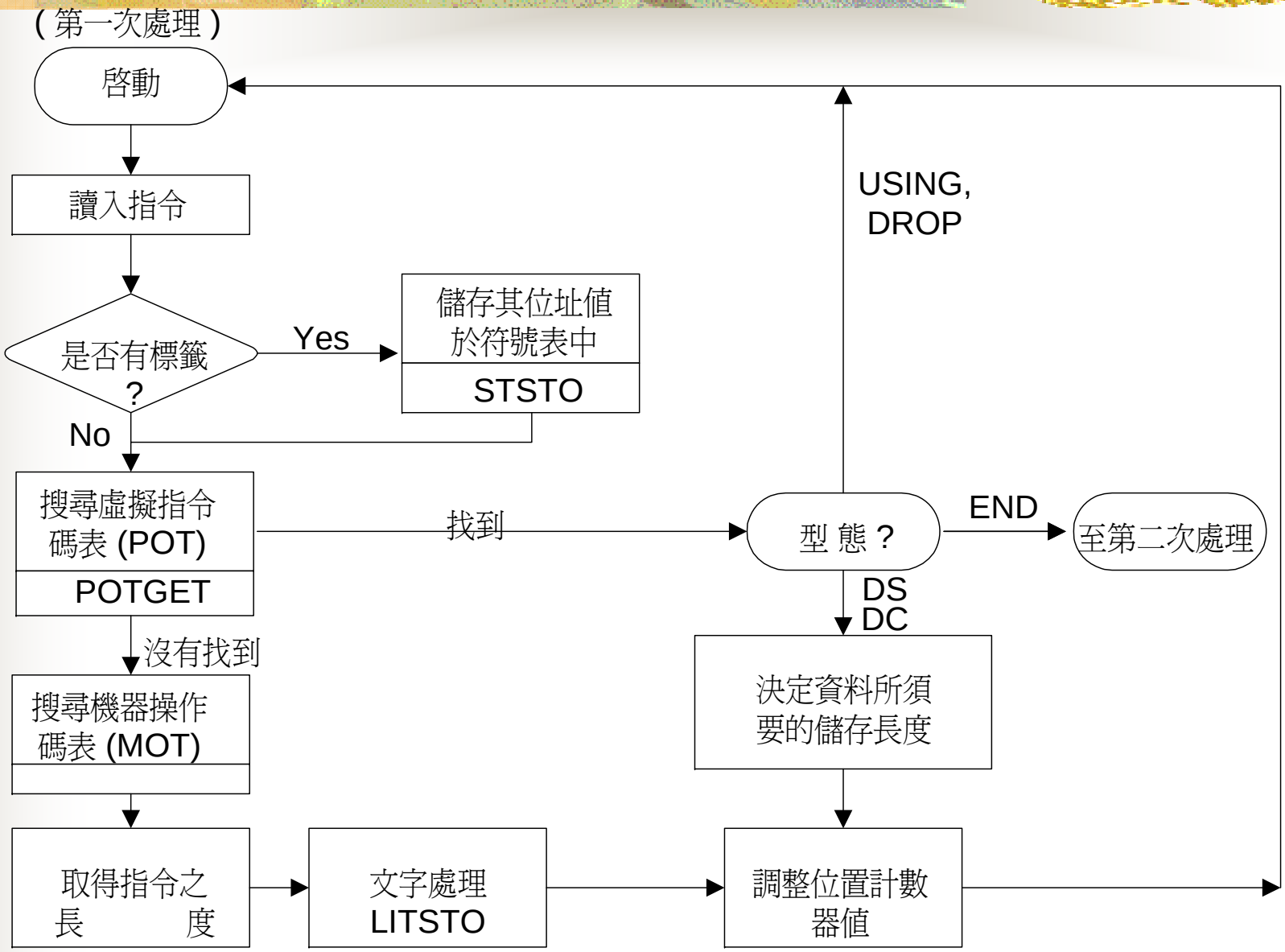
第一次處理所會用到的資料庫 (Data Bases)：

- (1) 輸入：原始程式。
- (2) 參考表格：

a. 虛擬操作碼表 (Pseudo-Operation Table，POT)： 用以查詢虛擬指令所對應的處理程式之位址。

b. 機器操作碼表 (Machine-Operation Table，MOT)： 用以將助記憶式指令對應成機器碼。

- (3) 暫存器 (Register)：
  程式計數器 (Program Counter，PC) 或稱為位址計數器 (Location Counter，LC)
- (4) 輸出 (Output)：
  a. 原始程式之副本 (Copy)。
  b. 符號表 (Symbol Table，ST)。
  c. 文字表 (Literal Table，LT)。


(第一次處理)
啓動
讀入指令
是否有標籤 ?
Yes
儲存其位址值 於符號表中
STSTO
No
搜尋虛擬指令 碼表 (POT)
POTGET
找到
型 態 ?
END
至第二次處理
USING, DROP
DS
DC
沒有找到
搜尋機器操作 碼表 (MOT)
決定資料所須 要的儲存長度
取得指令之 長 度
文字處理 LITSTO
調整位置計數 器值

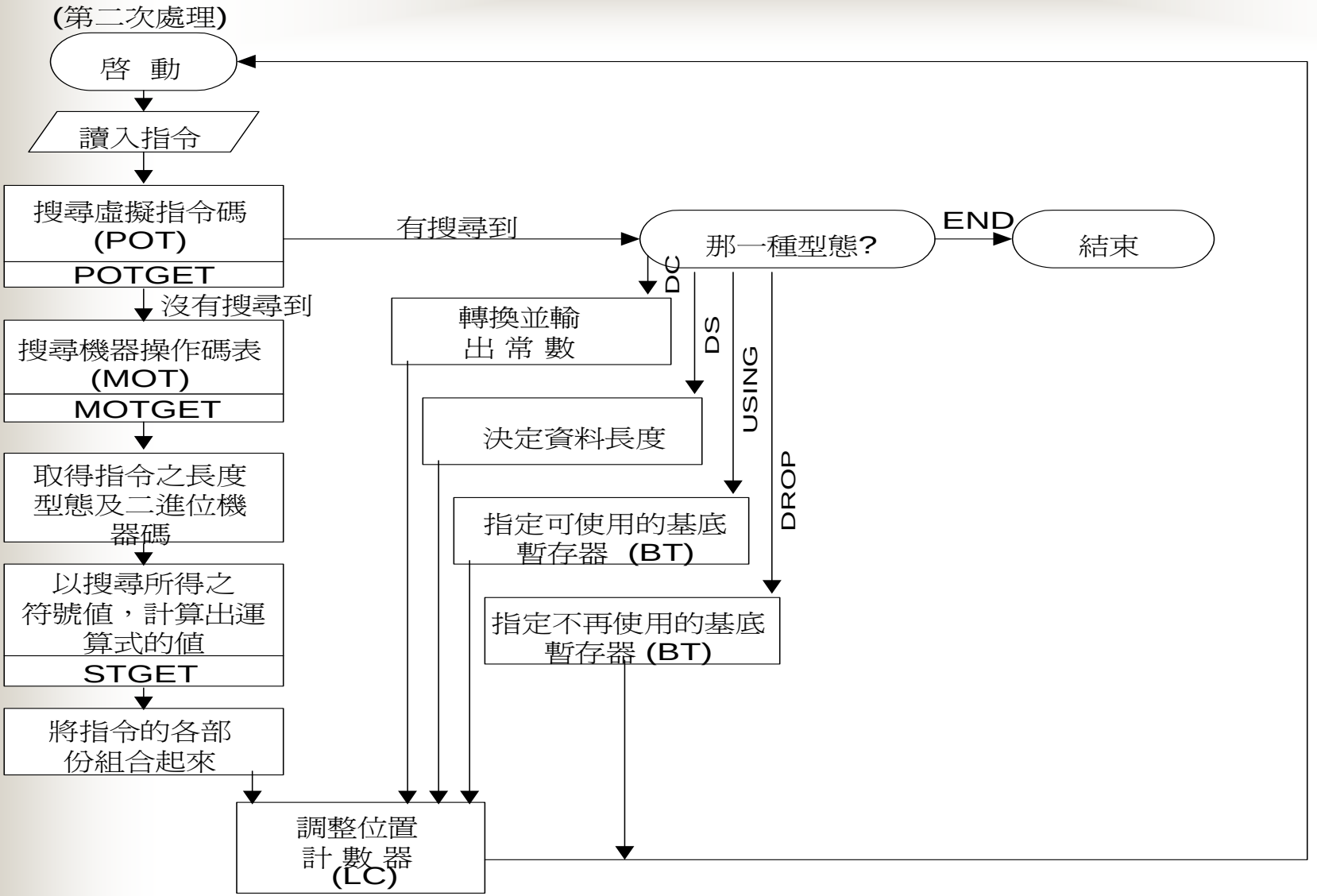
第二次處理：輸出目的程式 。

- a. 將指令組譯；即翻譯運算碼及符號位址的查詢。
- b. 產生 DS,DC 及文字所定義的值。
- c. 處理未處理的虛擬指令。
- d. 輸出目的程式並列出原始的組合語言程式。

第二次處理所會用到的資料庫 (Data Bases)：

- (1) 輸入 ：
  - a. 原始程式之副本。
  - b. 符號表。
  - c. 文字表。
- (2) 參考表格：
  - a. 機器操作碼表 。
  - b. 虛擬操作碼表 。

- (3) 暫存器 、表格及工作區 ：
  a. 暫存器： 位址計數器
  b. 表格： 用基底暫存器表來記錄那些暫存器被指定爲基底暫器及其內的值。
  c. 工作區：儲存需要做組譯的指令與要輸出的內容。
- (4) 輸出：
  a. 目的程式
  b. 列出原始的組合言程式


(第二次處理)
啓 動
讀入指令
搜尋虛擬指令碼
(POT)
POTGET
有搜尋到
沒有搜尋到
搜尋機器操作碼表
(MOT)
MOTGET
取得指令之長度
型態及二進位機
器碼
以搜尋所得之
符號值，計算出運
算式的值
STGET
將指令的各部
份組合起來
那一種型態?
END
結束
DC
DS
USING
DROP
轉換並輸
出 常 數
決定資料長度
指定可使用的基底
暫存器 (BT)
指定不再使用的基底
暫存器 (BT)
調整位置
計 數 器
(LC)

- 多次處理組合程式 (Multi-Pass Assembler) －
- 此種組合程式在組譯時會對原始程式做多次的掃瞄，所以允許程式中存在向前參考的未定義符號。於第一次處理時，先保留涉及向前參考之未定義符號，於第二次處理之後，再對這些未定義的符號做若干次 (Pass 3，Pass 4，…) 的額外處理。
- 如 "Variable EQU Expr"，Expr 可以是數值、名稱、運算式子。若 Expr 是一個名稱，則必須事定義過；否則，需多一次的處理，以便解決此向前參考的情況。

使用多次處理組合程式的好處，是其提供額外的四種功能：

- a. 可以使用巨集指令。
- b. 減少記憶體空間的使用量。
- c. 允許向前參考的符號存在，使程式易於撰寫，更具彈性。
- d. 易於程式碼最佳化的處理。

重定位之處理

組合程式在處理重定位址式程式之過程如下 －

- (1) 組合程式在組譯時，會產生一個命令於目的程式中，這命令會指示載入程式計算載入位址時須加上程式之起始位址。
- (2) 載入程式使用變更記錄 內的資訊，以便能將目的程式載入至適當的記憶體位址。

<註>：變更記錄內的資訊包含：

(1) 需要變更的位址欄相對於程式之起始位址。

(2) 需要變更的位址欄之長度。

絕對位址式程式與重定位址式程式之差異

- (1) 絕對位址式程式：

  程式之起始位址於組譯時便已經決定，且每次載入之位地都一樣。

- (2) 重定位址式程式：

  程式之起始位址於載入時才決定，且每次載入之位址不一定每次都一樣。

絕對位址式程式之缺點：

- (1) 記憶體空間不能有效的利用，因每次都佔用固定的位址。
- (2) 記憶體無法共用 。
- (3) 儲存於記憶體中的目的碼易造成混淆；即無法判定該值是位址值或是資料值 。

<註>：重定位址式程式，則可彌補上述之缺點。