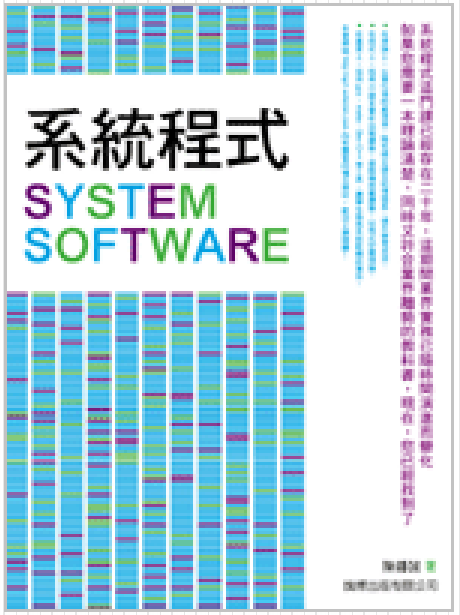
第 4 章、組譯器

作者：陳鍾誠

旗標出版社



第 4 章、組譯器

- 4.1　組譯器簡介
- 4.2　組譯器的演算法
- 4.3　完整的組譯範例
- 4.4　實務案例：處理器 IA32 上的 GNU 組譯器

4.1 組譯器簡介

- 組譯器乃是將組合語言轉換為機器碼的工具。
  - 組合語言 → (組譯器) → 目的檔 (或執行檔)
- 組譯器是組合語言程式師所使用的主要工具。

組譯器的過程示意圖

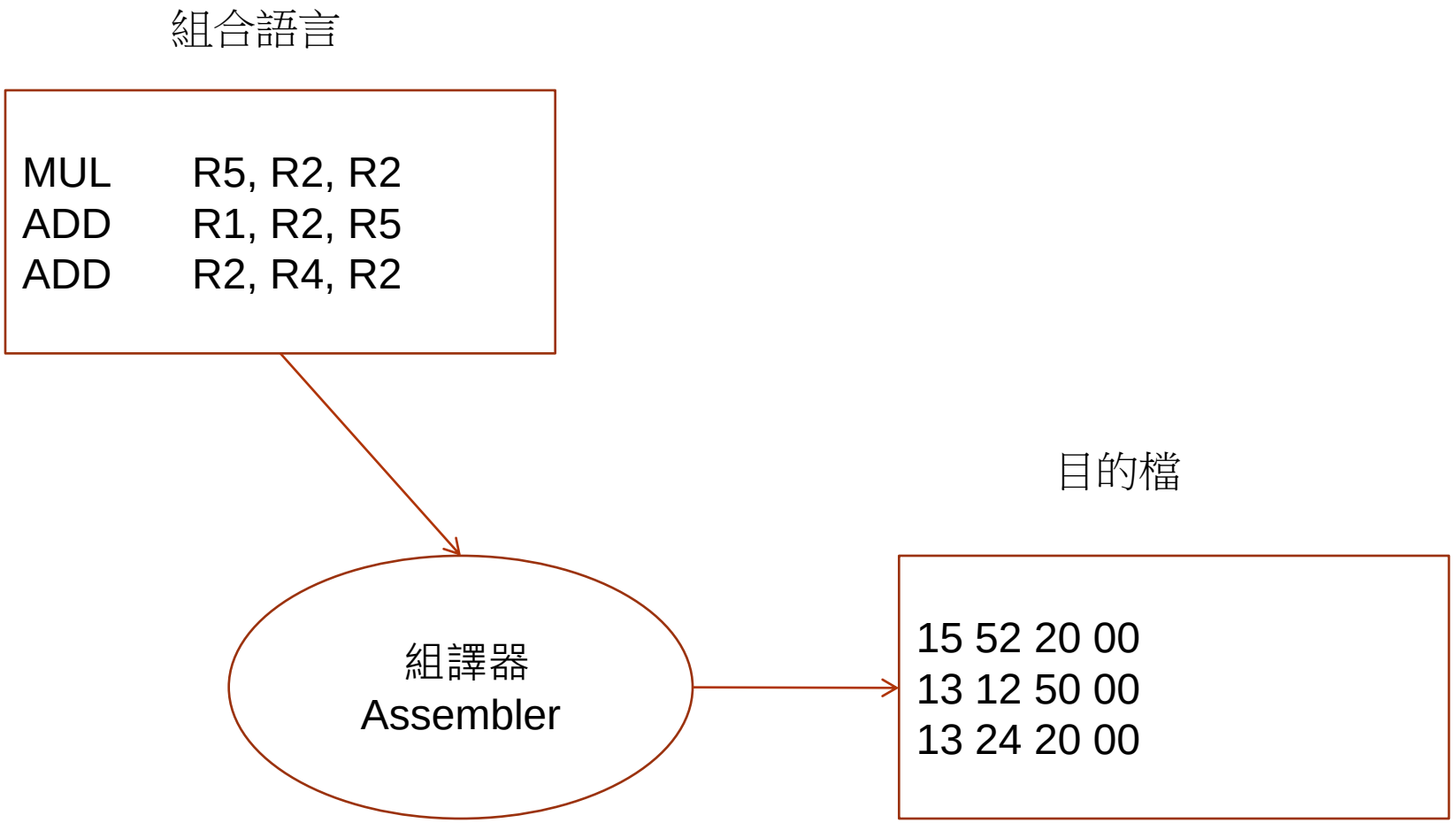


圖 4.1 組譯器的過程示意圖

簡單的組合語言程式

▶範例 4.1 簡單的組合語言程式

組合語言程式碼			說明
	LD	R1, B	載入記憶體變數 B 到暫存器 R1 當中
	ST	R1, A	將暫存器 R1 存回記憶體變數 A 當中
	RET		返回
A:	RESW	1	保留一個字組 (Word) 給變數 A
B:	WORD	29	宣告變數 B 為字組, 並設初值為 29

組譯：將組合語言轉譯成目的碼

▶範例 4.2 為組合語言程式加上目的碼

位址	程式碼			目的碼 （絕對定址版）	目的碼 （相對定址版）
0000		LD	R1, B	00100010	001F000C
0004		ST	R1, A	0110000C	011F0004
0008		RET		2C000000	2C000000
000C	A:	RESW	1	00000000	00000000
0010	B:	WORD	29	0000001D	0000001D

組譯的原理 (絕對定址)

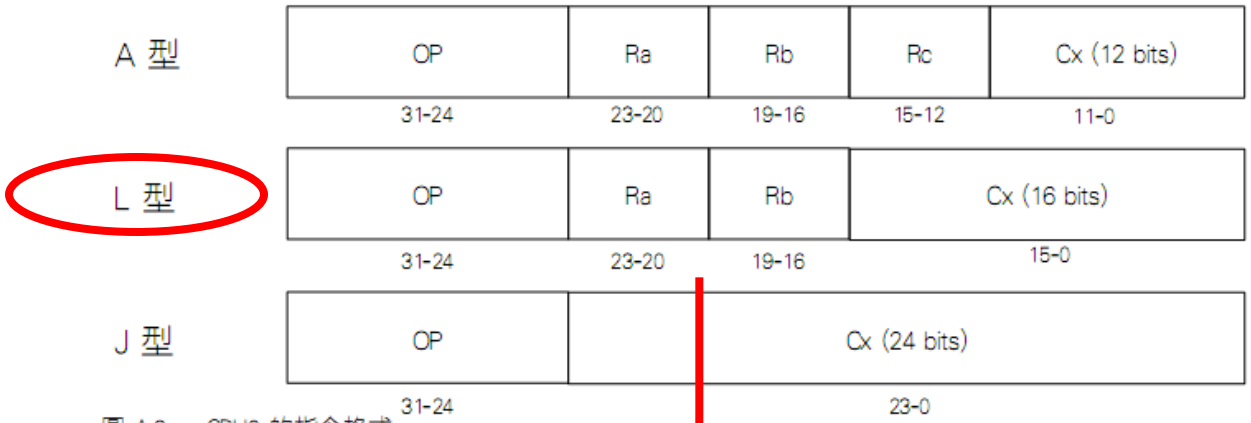


圖 A.2　CPU0 的指令格式

表格 A.1 CPU0 的指令編碼表

類型	格式	指令	OP	說明
載入儲存	L	LD[1]	00	載入 word
	L	ST	01	儲存 word
	L	LDB	02	載入 byte
	L	STB	03	儲存 byte
	A	LDR	04	LD 的暫存器版
	A	STR	05	ST 的暫存器版
	A	LBR	06	LDB 的暫存器版
	A	SBR	07	STB 的暫存器版
	L	LDI	08	立即載入

圖 4.2 將 LD R1, B 指令以絕對定址法編為目的碼

```
LD      Ra,     [Rb     +       Cx]
LD      R1,                     B
00      1       0               0010
```

組譯的原理 (相對於 PC 定址)

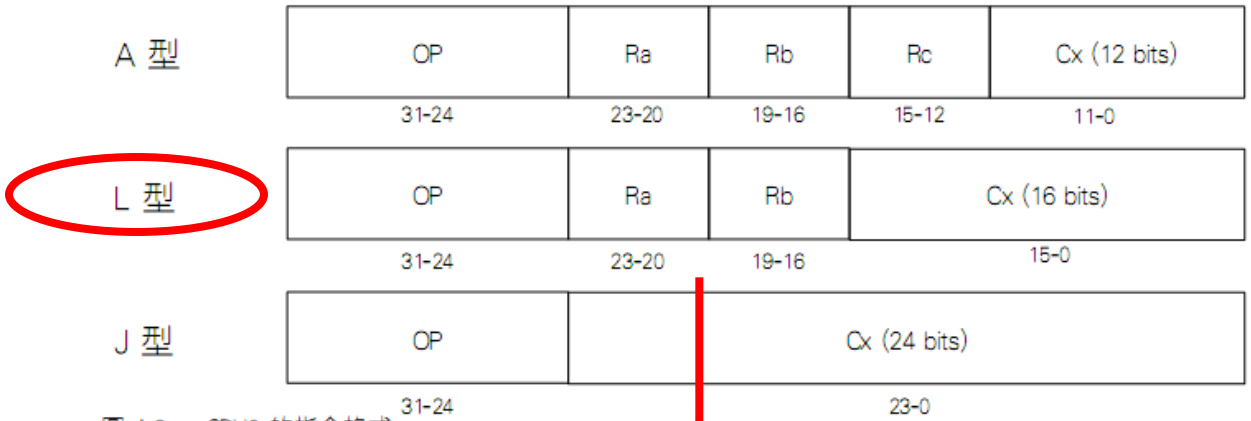


圖 A.2　CPU0 的指令格式

表格 A.1 CPU0 的指令編碼表

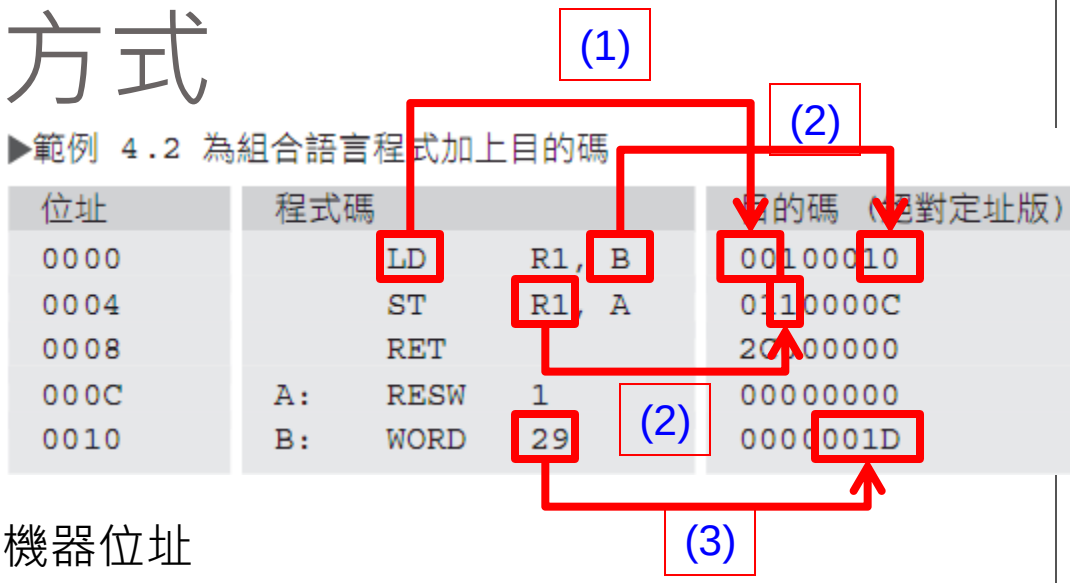
類型	格式	指令	OP	說明
載入儲存	L	LD[1]	00	載入 word
	L	ST	01	儲存 word
	L	LDB	02	載入 byte
	L	STB	03	儲存 byte
	A	LDR	04	LD 的暫存器版
	A	STR	05	ST 的暫存器版
	A	LBR	06	LDB 的暫存器版
	A	SBR	07	STB 的暫存器版
	L	LDI	08	立即載入

▶圖 4.3 將 LD R1, B 指令以相對於 PC 的方法編為目的碼

指令編碼					計算 (Cx=B-PC)	
LD	Rd,	[Ra	+	Cx]	0x0010	(B)
LD	R1,	R15	+	(B-PC)	- 0x0004	(PC)
00	1	F		000C	= 0x000C	(Cx)

二階段的組譯方式

- (1). 運算元轉換：
  - 將指令名稱轉換為機器語言
  - 例如 LD 轉為00, ST 轉為01等



- (2). 參數轉換：
  - 將暫存器轉為代號，符號轉換成機器位址
  - 例如 R1 轉為 1， A 轉為 000C，B 轉為 0010 等。

- (3). 資料轉換：
  - 將原始程式當中的資料常數轉換為內部的機器碼
  - 例如 29 轉換為 001D。

- (4). 目的碼產生：
  - 根據指令格式, 轉換成目的碼, 輸出到目的檔中。

映像檔

- 映像檔：最簡單的目的碼
- 組合語言 → (組譯器) → 映像檔

▶範例 4.2 為組合語言程式加上目的碼

位址	程式碼			目的碼 (絕對定址版)
0000		LD	R1, B	00100010
0004		ST	R1, A	0110000C
0008		RET		2C000000
000C	A:	RESW	1	00000000
0010	B:	WORD	29	0000001D

範例 4.3 範例 4.2 的映像檔 (相對定址版)

```
001F000C 011F0004 2C000000 00000000
0000001D
```

as0 組譯器

▶範例 4.4 使用 as0 組譯器組譯 Ex4_1.asm0

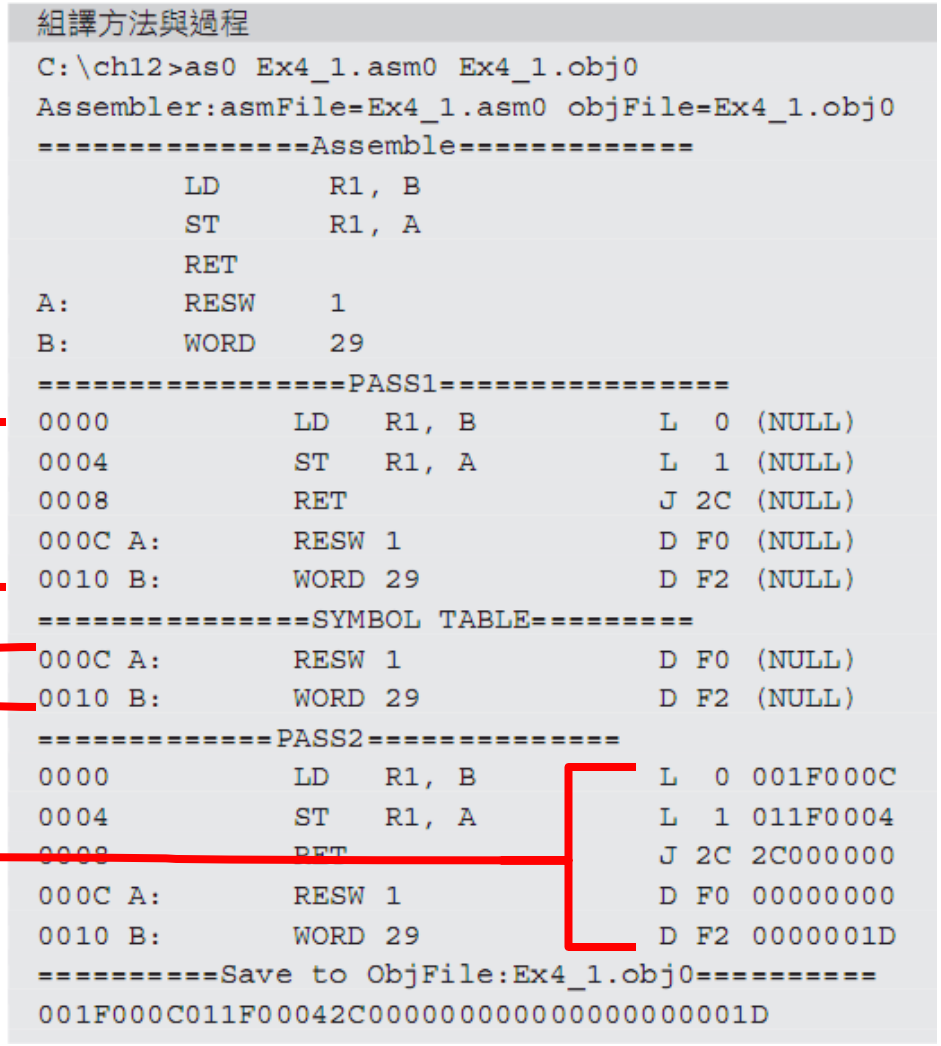
組譯方法與過程	說明
C:\ch12>as0 Ex4_1.asm0 Ex4_1.obj0	組譯 Ex4_1.asm0 輸出到 Ex4_1.obj0
Assembler:asmFile=Ex4_1.asm0 objFile=Ex4_1.obj0	
==========Assemble==========	
LD R1, B	讀入程式並輸出檢視
ST R1, A	
RET	
A: RESW 1	
B: WORD 29	
==========PASS1==========	組譯的第一階段 計算位址 並建立符號表
0000 LD R1, B L 0 (NULL)	
0004 ST R1, A L 1 (NULL)	
0008 RET J 2C (NULL)	
000C A: RESW 1 D F0 (NULL)	
0010 B: WORD 29 D F2 (NULL)	
==========SYMBOL TABLE=========	顯示符號表
000C A: RESW 1 D F0 (NULL)	
0010 B: WORD 29 D F2 (NULL)	
==========PASS2==========	組譯的第二階段 編定目的碼
0000 LD R1, B L 0 001F000C	
0004 ST R1, A L 1 011F0004	
0008 RET J 2C 2C000000	
000C A: RESW 1 D F0 00000000	
0010 B: WORD 29 D F2 0000001D	
==========Save to ObjFile:Ex4_1.obj0==========	將目的碼存入檔案 Ex4_1.obj0
001F000C011F00042C0000000000000000000001D	

4.2 組譯器的演算法

- 第一階段
  - 計算符號位址

資料結構：符號表

- 第二階段
  - 組譯指令與資料



組譯方法與過程

```
C:\ch12>as0 Ex4_1.asm0 Ex4_1.obj0
Assembler:asmFile=Ex4_1.asm0 objFile=Ex4_1.obj0
===============Assemble=============
        LD      R1, B
        ST      R1, A
        RET
A:      RESW    1
B:      WORD    29
================PASS1===============
0000            LD   R1, B          L  0 (NULL)
0004            ST   R1, A          L  1 (NULL)
0008            RET                 J 2C (NULL)
000C A:         RESW 1              D F0 (NULL)
0010 B:         WORD 29             D F2 (NULL)
===============SYMBOL TABLE=========
000C A:         RESW 1              D F0 (NULL)
0010 B:         WORD 29             D F2 (NULL)
=============PASS2==============
0000            LD   R1, B          L  0 001F000C
0004            ST   R1, A          L  1 011F0004
0008            RET                 J 2C 2C000000
000C A:         RESW 1              D F0 00000000
0010 B:         WORD 29             D F2 0000001D
==========Save to ObjFile:Ex4_1.obj0==========
001F000C011F00042C00000000000000000001D
```

第一階段 (計算符號位址)

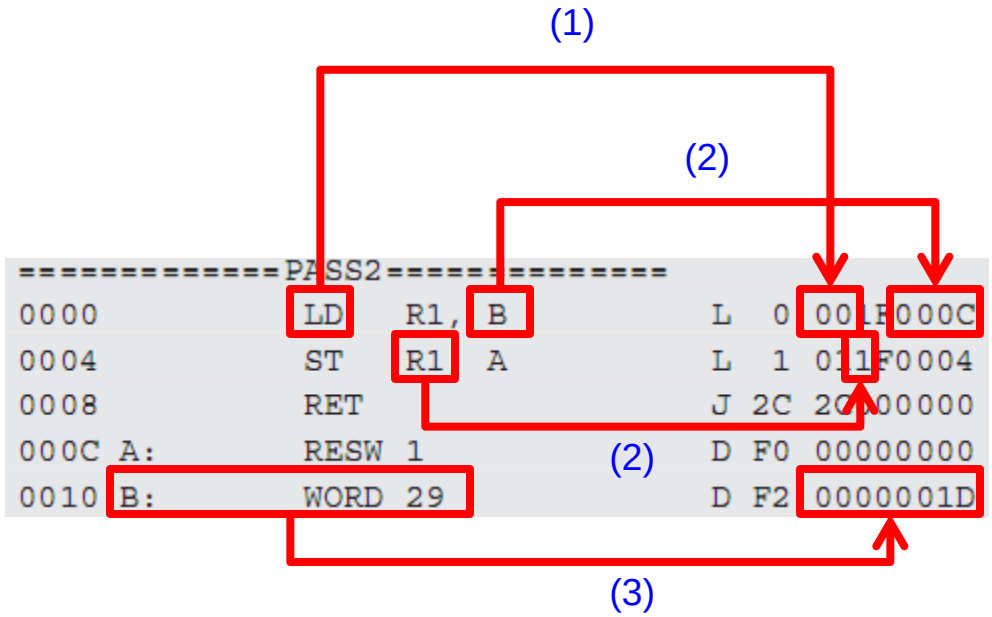
- 1. 決定每一個指令所佔記憶空間的大小
  - 例如
    - 決定 WORD、RESW 等指令所定義的資料長度
    - 決定 LD、ST 等指令所佔空間的大小
- 2. 計算出程式當中每一行的位址。
- 3. 儲存每一個標籤與變數的位址
  - 例如
    - 變數 A 為 0x000C
    - 變數 B 為 0x0010

組譯方法與過程

```
C:\ch12>as0 Ex4_1.asm0 Ex4_1.obj0
Assembler:asmFile=Ex4_1.asm0 objFile=Ex4_1.obj0
==============Assemble============
        LD      R1, B
        ST      R1, A
        RET
A:      RESW    1
B:      WORD    29
================PASS1==============
0000            LD   R1, B          L  0 (NULL)
0004            ST   R1, A          L  1 (NULL)
0008            RET                 J 2C (NULL)
000C A:         RESW 1              D F0 (NULL)
0010 B:         WORD 29             D F2 (NULL)
==============SYMBOL TABLE========
000C A:         RESW 1              D F0 (NULL)
0010 B:         WORD 29             D F2 (NULL)
============PASS2=============
0000            LD   R1, B          L  0 001F000C
0004            ST   R1, A          L  1 011F0004
0008            RET                 J 2C 2C000000
000C A:         RESW 1              D F0 00000000
0010 B:         WORD 29             D F2 0000001D
==========Save to ObjFile:Ex4_1.obj0==========
001F000C011F00042C0000000000000000000001D
```

第二階段 (組譯指令與資料)

- (1). 轉換指令OP欄位為機器碼
  - 例如
    - LD 轉換為 00
    - ST 轉換為 01
- (2). 轉換指令參數為機器碼
  - 例如
    - R1轉換為 1
    - B 轉換為 000C
- (3). 轉換資料定義指令為位元值
  - 例如
    - WORD B 29 轉為 0000001D
- (4). 產生目的碼並輸出到目的檔當中




```
==========Save to ObjFile:Ex4_1.obj0==========
001F000C011F00042C0000000000000000000001D
```

將目的碼存入檔案 Ex4_1.obj0

組譯器的資料結構

- 運算碼表 (Op Table)
  - 儲存指令與代碼的配對表
  - 通常使用雜湊表格

範例 4.5 資料結構 1 - 指令表

```
LD      00
ST      01
LDB     02
```

- 符號表 (Symbol Table)
  - 儲存符號與位址的配對表
  - 通常使用雜湊表格

範例 4.6 資料結構 2 - 符號表

```
A0000000C
B00000010
```

第一階段(PASS1)

- 計算位址

組譯器的第一階段演算法	說明
Algorithm AssemblerPass1	組譯器的第一階段演算法
Input AssmeblyFile	輸入：組合語言檔
Ouptut SymbolTable	輸出：符號表
SymbolTable = new Table();	建立空的符號表
file = open(AssemblyFile)	開啟組合語言檔
while not file.end	當檔案還沒結束前，繼續讀檔
line = readLine(file)	讀下一行
if line is comment	如果該行為註解
continue	則忽略此行，繼續下一輪迴圈
label= label(line)	取得該行中的標記
if label is not null	如果該行有標記
symbolRecord =	於符號表中尋找該標記
symbolTable.search(label)	
if symbolRecord is not null	如果找到該標記
report error	則報告錯誤，標記重複定義
else	否則
symbolTable.add(label, address)	將(標記、位址)放入符號表中
end if	
op = operator(line)	取得該行中的指令部分(助憶符號)
opRecord = opTable.search(op);	於指令表中尋找該指令
if opRecord is not null	如果找到該指令
address += 4;	則將位址加 4
else if op is 'BYTE'	如果指令是 BYTE
address += 1*length(parameters)	則將位址加 1 * 參數個數
else if op is 'WORD'	如果指令是 WORD
address += 4 * length(parameters)	則將位址加 4 * 參數個數
else if op is 'RESB'	如果指令是 RESB
address += length(parameter 1)	則將位址加上參數 1
else if op is 'RESW'	如果指令是 RESW
address += 4 * length(parameter 1)	則將位址加上 4 * 參數 1
else	如果不屬於上述情形之一，
report error	則代表該指令拼寫有誤，顯示錯誤訊息
end if	
end while	
End Algorithm	

第二階段 (PASS2) – 主程式

- 指令轉為機器碼

組譯器的第二階段演算法	說明
Algorithm AssemblerPass2	組譯器的第二階段演算法
Input AssmeblyFile, SymbolTable	輸入：組合語言檔，符號表
Ouptut ObjFile	輸出：目的檔
file = open(AssemblyFile)	開啟組合語言檔作為輸入
while not file.end	當檔案還沒結束前，繼續讀檔
line = readLine(file)	讀下一行
(op, parameter) = parse(line)	取得指令碼與參數
opRecord = opTable.search(op)	於指令表中尋找該指令
if opRecord is not null	如果找到該指令
objCode = translateInstruction (parameter, address, opRecord)	將指令轉換為目的碼
address += length(objCode)	計算下一個指令位址
else if op is 'WORD' or 'BYTE'	如果指令是 WORD 或 BYTE
objCode = translateData(line)	將資料轉換為目的碼
address += length(objCode)	計算下一個指令位址
else if op is 'RESB'	如果指令是 RESB
address += parameter[1]	則將位址加上參數 1
else if op is 'RESW'	如果指令是 RESW
address += 4 * parameter[1]	則將位址加上 4 * 參數 1
end if	
output ObjCode to ObjFile	將目的碼寫入目的檔當中
end while	
End Algorithm	

第二階段 (PASS2) – 指令轉換函數

- 根據指令類型
  - L 型
  - A 型
  - J 型
- 轉換成該型編碼

```
Algorithm TranslateInstruction                  轉換指令為目的碼
Input parameter, pc, opRecord                   輸入：參數、程式計數器、指令記錄
Output objCode                                  輸出：目的碼
  if (opRecord.type is L)                       如果是 L 型指令
    Ra = parameter[1]                           設定 Ra 參數
    if (parameter[3] is Constant)               如果是常數
      Rb = 0
      Cx = toInteger(parameter[2]);               設定 Cx 為該常數
    if (parameter[3] is Variable)               如果是變數
      Cx = address(Variable) - pc                 設定 Cx 為位移 (標記-PC)
      Rb = parameter[2]                           設定 Rb 參數
    end if
    objCode = opRecord.opCode                   設定目的碼
      + id(Ra)+id(Rb)+hex(Cx);
  else if (op.type is A)                        如果是 A 型指令
    Ra = parameter[1]                           取得 Ra
    Rb = parameter[2]                           取得 Rb
    if (parameter[3] is Register)               如果參數 3 是暫存器
      Rc = parameter[3]                           取得 Rc
    else if (parameter[3] is Constant)          如果參數 3 是常數
      Cx = toInteger(parameter[3]);               設定 Cx 為該常數
    end if
    objCode = opRecord.opCode                   設定目的碼
      + id(Ra)+id(Rb)+id(Rc)+hex(Cx)
  else if (op.type is J)                        如果是 J 型指令
    Cx = parameter[1]                           取得 Cx
    objCode = opRecord.opCode+hex(Cx)           設定目的碼
  end if
  return objCode                                傳回目的碼
End Algorithm
```

4.3 完整的組譯範例

- 範例：陣列加總功能
  - 功能
    - 如右圖 C 語言所示
  - 原始程式
    - 如範例 4.7 所示
  - 組譯結果
    - 如範例 4.8 所示

C 語言

```
int a[] = {3, 7, 4};
int *aptr = &a;
int sum;
for (i=3; i>0; i--) {
 sum += *aptr;
 aptr += 4;
}
return sum;
```

原始程式

▶範例 4.7 組合語言程式及其 C 語言對照版 (加總功能)

行號	組合語言 (檔案 ArraySum.asm0)	C 語言
1	`LDI R1, 0 ; R1=0`	`int a[] = {3, 7, 4};`
2	`LD R2, aptr ; R2=aptr`	`int *aptr = &a;`
3	`LDI R3, 3 ; R3=3`	`int sum;`
4	`LDI R4, 4 ; R4=4`	`for (i=3; i>0; i--) {`
5	`LDI R9, 1 ; R9=1`	`  sum += *aptr;`
6	`FOR:`	`  aptr += 4;`
7	`LD R5, [R2] ; R5=*aptr`	`}`
8	`ADD R1, R1, R5 ; R1+=*aptr`	`return sum;`
9	`ADD R2, R2, R4 ; R2+=4`	
10	`SUB R3, R3, R9 ; R3--;`	
11	`CMP R3, R0 ; if (R3!=0)`	
12	`JNE FOR ;  goto FOR;`	
13	`ST R1, sum ; sum=R1`	
14	`LD R8, sum ; R8=sum`	
15	`RET`	
16	`a: WORD 3, 7, 4 ; int a[]={3, 7, 4}`	
17	`aptr: WORD a ; int *aptr = &a`	
18	`sum: WORD 0 ; int sum = 0`	

組譯結果

- 組譯報表
  - 範例 4.8
- 符號表
  - 圖 4.6
- 目的檔
  - 範例 4.9

範例 4.8 組合語言程式及其目的碼 (加總功能)

```
記憶體位址     組合語言          指令類型  OP   目的碼
0000           LDI   R1, 0          L     8   08100000
0004           LD    R2, aptr       L     0   002f003c
0008           LDI   R3, 3          L     8   08300003
000c           LDI   R4, 4          L     8   08400004
0010           LDI   R9, 1          L     8   08900001
0014 FOR:                                ff
0014           LD    R5, [R2]       L     0   00520000
0018           ADD   R1, R1, R5     A    13   13115000
001c           ADD   R2, R2, R4     A    13   13224000
0020           SUB   R3, R3, R9     A    14   14339000
0024           CMP   R3, R0         A    10   10300000
0028           JNE   FOR            J    21   21ffffe8
002c           ST    R1, sum        L     1   011f0018
0030           LD    R8, sum        L     0   008f0014
0034           RET                  J    2c   2c000000
0038 a:        WORD 3, 7, 4         D    f2   000000030000000700000004
0044 aptr:     WORD a               D    f2   00000038
0048 sum:      WORD 0               D    f2   00000000
```

圖 4.6 組合語言程式範例 4.8 於組譯第一階段執行完後的符號表

符號名稱	位址
FOR	0x14
a	0x38
aptr	0x44
sum	0x48

範例 4.9 <範例 4.8 > 的目的檔

```
08100000 002F003C 08300003 08400004
08900001 00520000 13115000 13224000
14339000 10300000 21FFFFE8 011F0018
008F0014 2C000000 00000003 00000007
00000004 00000038 00000000
```

4.4　實務案例：處理器 IA32 上的 GNU 組譯器

- as : GNU 的組譯器
  - -a 參數
    - 讓 GNU 的 as 組譯器 產生組譯報表檔

GNU 的組譯報表

指令長度不固定

範例

addl %eax, sum
→ 010500000000

addl $1, %eax
→ 83C001

範例 4.10 使用 GNU as 產生組譯報表

```
C:\ch04>as -a gnu_sum.s
GAS LISTING gnu_sum.s                    page 1


   1                            .data
   2 0000 00000000      sum:    .long 0
   3 0004 00000000              .text
   3      00000000
   3      00000000
   4                    .globl _asmMain
   5                    .def _asmMain; .scl 2; .type 32; .endef
   6                    _asmMain:
   7 0000 B8010000              mov $1, %eax
   7      00
   8                    FOR1:
   9 0005 01050000              addl %eax, sum
   9      0000
  10 000b 83C001                addl $1, %eax
  11 000e 83F80A                cmpl $10, %eax
  12 0011 7EF2                  jle FOR1
  13      00
  14 0018 C3909090              ret
  14      90909090

GAS LISTING gnu_sum.s                    page 2


DEFINED SYMBOLS
                            *ABS*:00000000 fake
           gnu_sum.s:2      .data:00000000 sum
           gnu_sum.s:6      .text:00000000 _asmMain
           gnu_sum.s:8      .text:00000005 FOR1

NO UNDEFINED SYMBOLS
```

符號表 →

結語

- 對 IA32 的編碼方式有興趣的讀者，可以參考 Kip Irvine 的組合語言一書

習題

1. 請說明組譯器的輸入、輸出與功能為何？
2. 請說明組譯器第一階段 (PASS1) 的功能為何？
3. 請說明組譯器第二階段 (PASS2) 的功能為何？
4. 請說明組譯器當中的符號表之用途為何？
5. 請說明組譯器當中的指令表之用途為何？
6. 請仿照範例 4.4，使用本書第12章所實作的 as0 組譯器，組譯 Ex4_1.asm0 組合語言檔，並仔細觀察其輸出結果。
7. 請閱讀本書第12章所附的 Assembler.c 與 Assembler.h 等C語言程式，並且對照本章的演算法，以學習 CPU0 組譯器的實作方式。
8. 請按照4.4節的方法，操作 GNU 工具對組合語言進行組譯動作，並檢視組譯報表，找出各個符號的位址。
9. 請於 http://kipirvine.com/asm/ 網站下載Kip Irvine 書籍組合語言程式範例，並以 Visual Studio 進行組譯與執行。