組合語言

- 早期的系統程式皆是用組合語言所撰寫的，然而，現今的系統程式大多可由高階語言來撰寫。
- 機器語言與組合語及組合語言與高階語言比較

使用組合語言替代機器語言的優缺點 －

<優點>：

- a. 指令爲助記憶式符號，記憶及撰寫時較方便。
- b. 位址是以符號表示，較具有彈性與變化性。
- c. 較機器語言容易瞭解及閱讀。
- d. 程式中引用資料較容易。

<缺點>：

- a. 必須由組合程式將原始程式轉成目的程式 。

使用組合語言替代高階語言的優缺點 –

<優點>：

- a. 執行速度快。
- b. 程式佔用空間較小。

<缺點>：

- a. 撰寫程式較困難。
- b. 與機器相關
- c. 不具有可攜性

組合語言的指令可分成兩大類：

(1) 機器指令：

- 爲可執行的指令，有相對應的機器碼。

(2) 虛擬指令：

- 爲不可執行的指令，並沒有產生對應的機器碼，只能命令組合程式做某些動作。

IBM 360/370 組合語言

- 用一個 IBM 360/370 之組合語言程式來介紹其指令之功能

組合語言程式			相對位置		
PROG	START				
HEAD	BALR	15,0	0	BALR	15,0
	USING	HEAD+2,15			
	SR	4,4	2	SR	4,4
	L	3, = F'10'	4	L	3,30(0,15)
REPEAT	L	2,DATA(4)	8	L	2,42(4,15)
	A	2, =F'49'	12	A	2,34(0,15)
	ST	2, =DATA(4)	16	ST	2,42(4,5)
	A	4, =F'4'	20	A	4,38(0,15)
	BCT	3, * - 16	24	BCT	3,6(0,15)
	BR	14	28	BCR	15,14
	LTORG				
			32	10	
			36	49	
			40	4	
DATA	DC	F'1,3,3,3,3,4,5,9,0'	44	1	
			48	3	
			52	3	
				⋮	
	END				

(1) USING：

- 此爲一虛擬指令，用以指示組合程式使用那一個暫存器爲基底暫存器，並指定該暫存器內所含之值爲何。如：USING HEAD + 2,15 表示以編號第 15 號之暫存器爲基底暫存器，而其內所含之值爲 HEAD + 2。

(2) START：

- 此爲一虛擬指令，用以指示組合程式此處爲程式之開端且其名稱爲何。如：PROG START 表示程式之名稱爲 PROG。

(3) END：

- 此爲一虛擬指令，用以指示組合程式此處爲程式的結束。

(4) BALR：

- 此爲一機器指令是跳躍並鏈結暫存器之縮寫。此指令是將下一個要執行指令的位址存入指定的暫存器中，然後再跳到第二個欄位暫存器所指示的位址去執行。若第二個欄位的運算元是第 0 號暫存器時，就直接去執行下一條指令。如：BALR 15,0 表示將下一條指令的位址儲存於編號第 15 號的暫存內，但是由於第二個欄位的運算元爲第 0 號暫存器，故直接執行下一條指令。

(5) BR：

- 此爲一機器指令，表示無條件跳至其後暫存器內所指示的位址處執行。如：BR 14。

(6) LTORG：

- 此爲一虛擬指令，用以指示組合程式將文字表中的常數從此處開始存放。此指令通常用於較長程式中。一般而言，若是沒有此一指令，則組合程式會將文字表放置於 END 指令後。

(7) DC：

- 此爲一虛擬指令，是定義常數的縮寫。指示組合程式定義某一符號爲一常數值。如：Beta DC F'10'，表示 Bata 的值爲 10。

(8) DS：

- 此爲一虛擬指令，是定義儲存空間之縮寫。指示組合程式預留記憶體空間。

(9) EQU：

- 此爲一虛擬指令，亦是用以定義常數，但用 EQU 所定義的常數在程式中爲絕對的，不能再變更其值。而用 DC 所定義的常數則可以重新定義新的常數值。

- 組合語言之指令格式 –

操作碼欄 (Operation)	運算元欄 (Operand Field)

(1) 操作碼 ：

- 此欄位含有一已定義之機器碼。當 CPU 由記憶體中取出指令，經由指令解譯器 解讀後，此操作碼可用來告知 CPU 所要執行的動作。

(2) 運算元 ：

- 此欄位通常含有數個運算元。當指令經解讀後，此欄位用以告知 CPU 到何處存取資料以便運算。

- 組合語言指令之類型 –

(1) 記憶體參考式指令 ：

- 指令內含有數個運算元，而其值儲存於記憶體中，必須至記憶體中讀取該運算元的值。

(2) 暫存器參考式指令 ：

- 指令執行時僅對暫存器的內容做運算。如：將第 5 號暫存器的值清除爲零 CLR 5。

(3) 暫存器對暫存器運算指令 ：

- 針對兩個暫存器間做運算之指令。如：將第 5 號暫存器之值減掉第 2 號暫存器之值後，再將結果儲存於第 5 號暫存器內 SUB 5,2。

(4) 輸出入指令 ：

- 專司資料輸出入的工作或控制輸出入設備的指令。

(5) 巨集指令 ：

- 將一群指令事先以一指令替代，此指令謂巨集指令。其功能類似副程式。

定址模式

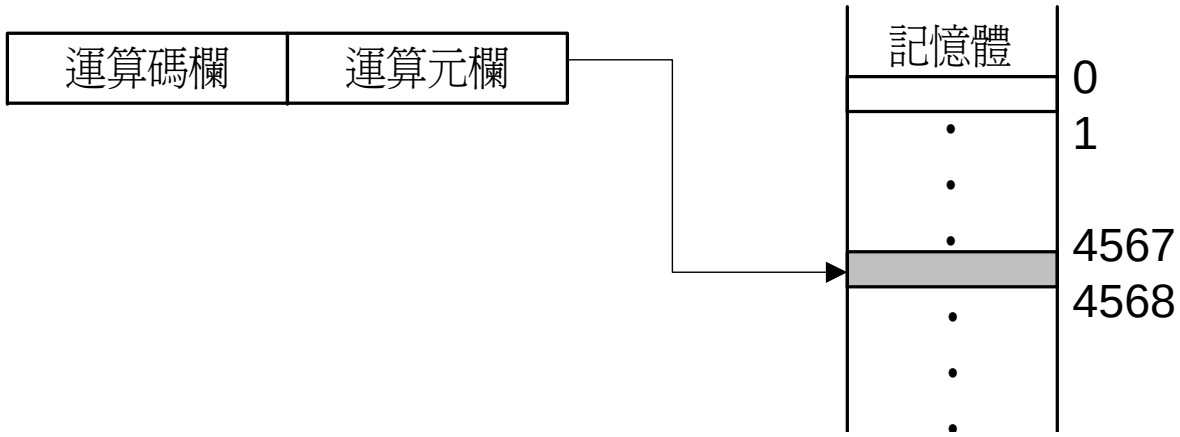
- 常見的定址模式有如下九種：

1. 立即定址模式 ：

- 指令的運算元欄內的值就是所要的資料，執行時不必再做額外的記憶體讀取動作，可立即使用該運算元欄內的值做運算。

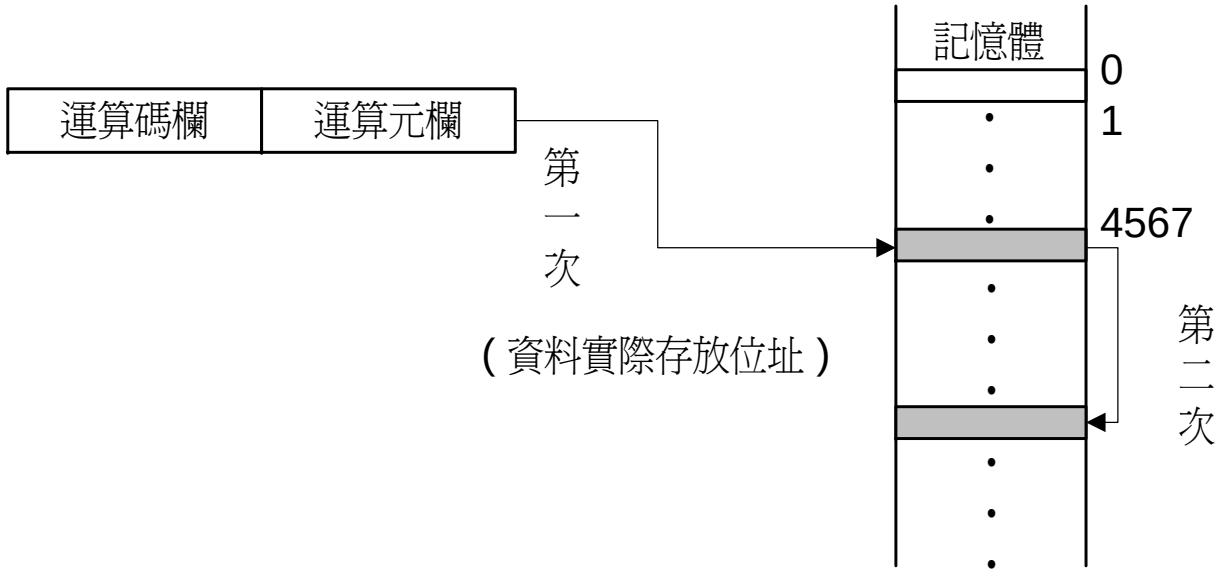
2. 直接定址模式 ：

- 指令的運算元欄內的值表示資料存放於記憶體的實際位址 (有效位址，Effective Address)，故需要做一次的記憶體讀取，以取得所需之資料。此模式亦稱爲絕對定地模式 。
- 例：MOV AX，[4567]



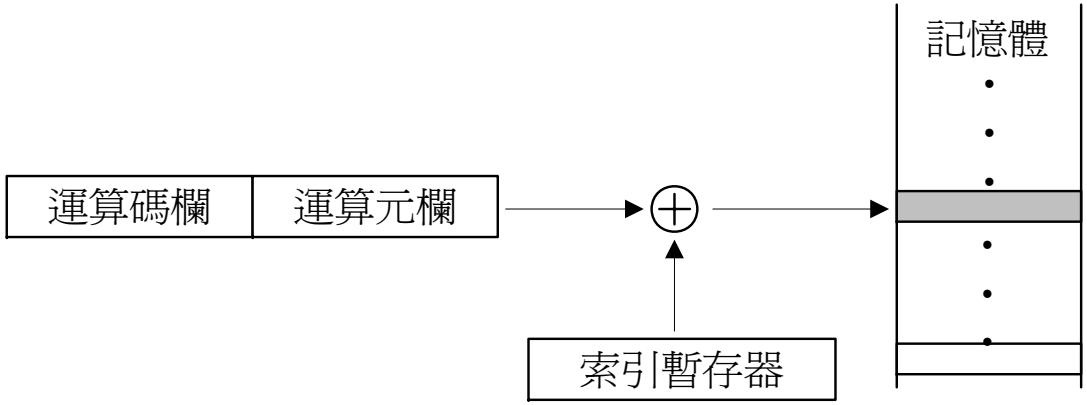
3. 間接定址模式 ：

- 指令的運算元欄內的值爲有效位址的位址值，故需做二次的記憶體讀取，以取得所需之資料。



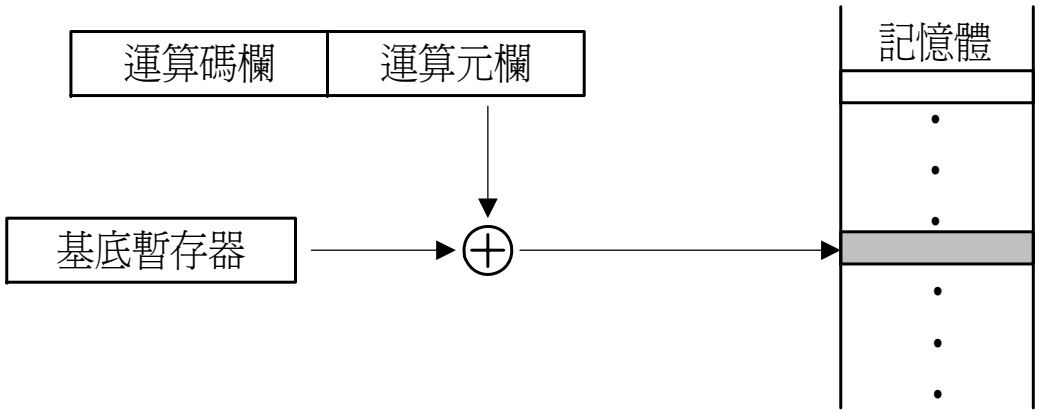
4. 索引定址模式 ：

- 將指令的運算元欄位內的值加上索引暫存器內的值，以求得存放資料的有效位址。在此模式中，使用者若想讀取記憶體中其它位址的資料，必須藉由改變索引暫存器內的值，而非調整運算元欄位內的值。



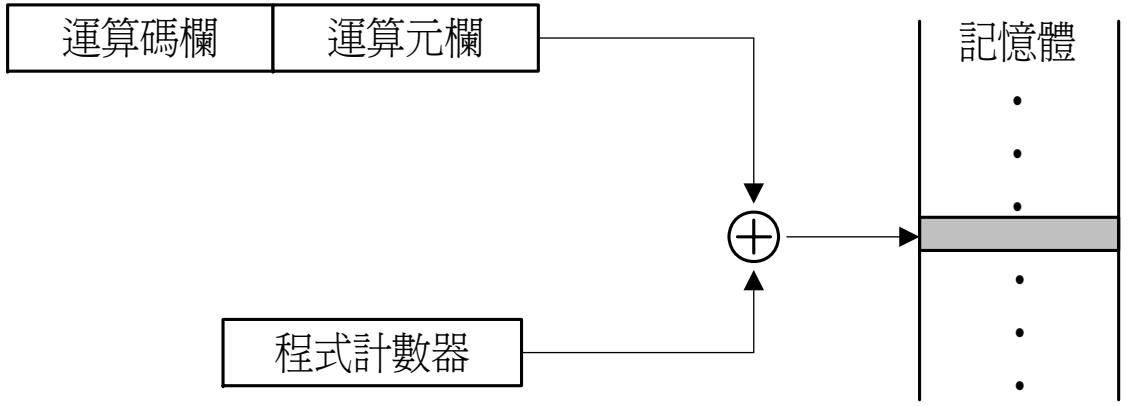
5. 基底定址模式 ：

- 將基底暫存器 內的值加上指令的運算元欄位內的值，以得到存放資料的有效位址。在此模式中，使用者若想讀取記憶體中其它位址的資料，必須藉由改變運算元欄位內的值，而非調整基底暫存器內的值。



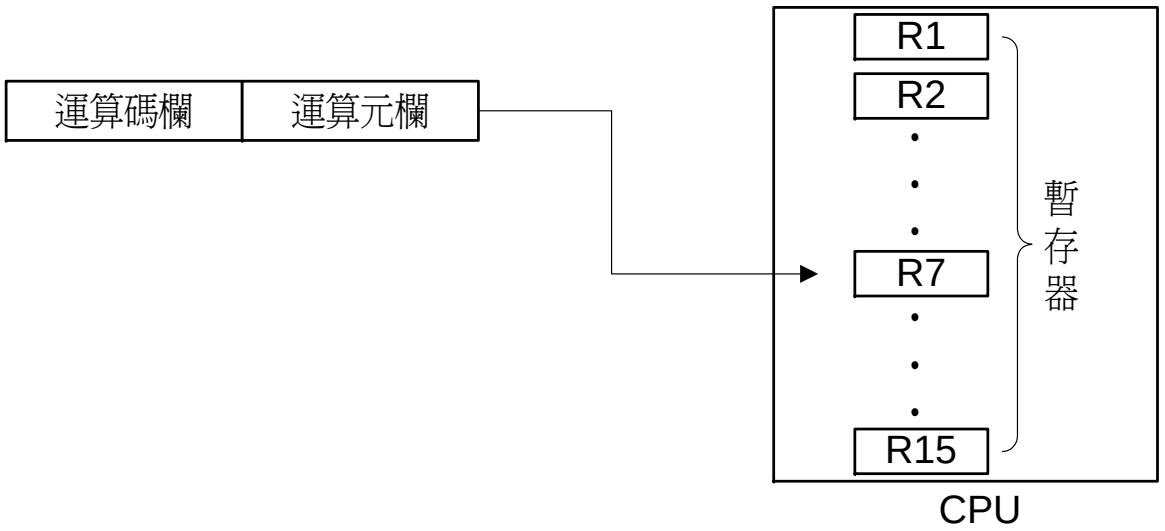
6. 相對定址模式：

- 將指令的運算元欄位內的值加上程式計數器內的值，以求得存放資料的有效位址。



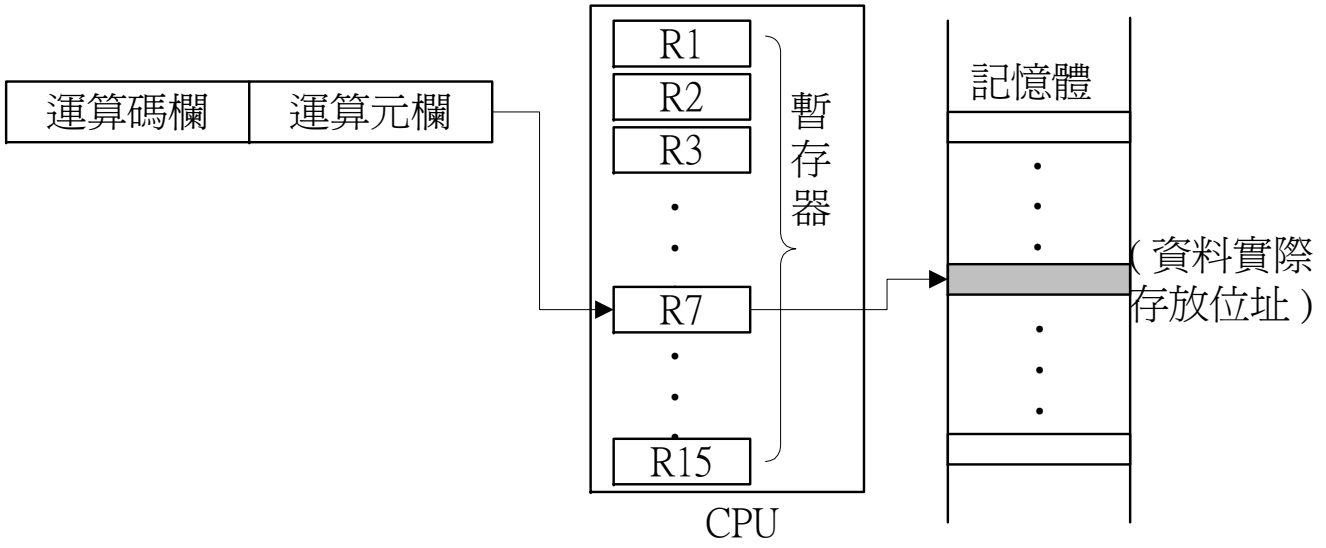
7. 暫存器定址模定：

- 存放資料的有效位址即是指令的運算元欄位內所指定的暫存器。
- 例：MOV AX，R7



8. 暫存器間接定址模式 ：

- 存取資料的有效位址即是指令內所指定暫存器內的值。
- 例：MOV AX，[R7]



9. 隱含定址模式 ：

- 此模式的運算元已經被隱含於指令內。如指令 CMA 即是對累加器內值做 1 的補數 。