

圖解 6502 指令集

第一章 基本概念

當你和外國人交談時，必須使用對方懂得的語言，或者經由第三者翻譯，才能彼此溝通。人與電腦 "交談" 也是一樣，必須使用電腦所能接受的語言。電腦所能接受的語言，稱之為 "機器語言" (machine Language)。

什麼是機器語言？

培基 (BASIC) 語言是不是機器語言？不是！培基語言只是為了使於人類理解而設計的一種電腦語言，電腦之所以能接受這種語言，是因其內部有一個 "培基語言直譯器" (BASIC Interpreter)，您輸入的每一行敘述都先經由直譯器加以檢查，並 "翻譯" 成電腦能夠瞭解的機器語言，電腦就會依照指示來執行，然後將執行結果再轉換為培基型式輸出。

電腦只是一部機器，它所能接受的基本訊號只有兩種狀態 "有" 或 "沒有"；好比電燈一樣只能是 "開" 或 "關" 兩種情形；這兩種狀態若以數字來表示即為 "1" 或 "0"。換句話說，電腦所能接受的資料必須由 1 或 0 來組成，這種最基本的資料表示單位稱為 "位元" (bit)。

由若干個位元可以組合成許多不同的數字碼，由於這些數字碼只由 0 與 1 來表示，所以又稱為 "二進位制數碼"。例如，101、1100、01101101 等等。

一般的電腦均使用 8 個位元來組合成各種不同的數字碼，而將這些數字碼加以規劃，各賦予不同的明確意義。這些數字碼就是電腦所能辨認的 "字"，它們組成的語言就稱為 "機器語言"。例如：

```
1 0 1 0 1 1 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1
1 0 0 0 1 1 0 1
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 1
```

這組二進制碼就是一小段程式，每個數碼均賦予不同的意義，電腦可以依其指示執行所需的動作。

由 8 個位元組成的一個數碼，稱為位元組 (Byte)。

數字系統

既然機器語言是由不同的 "二進制數碼" 所組成，因此，若想要直接與電腦交談，必須先對二進制數字系統有個概念。

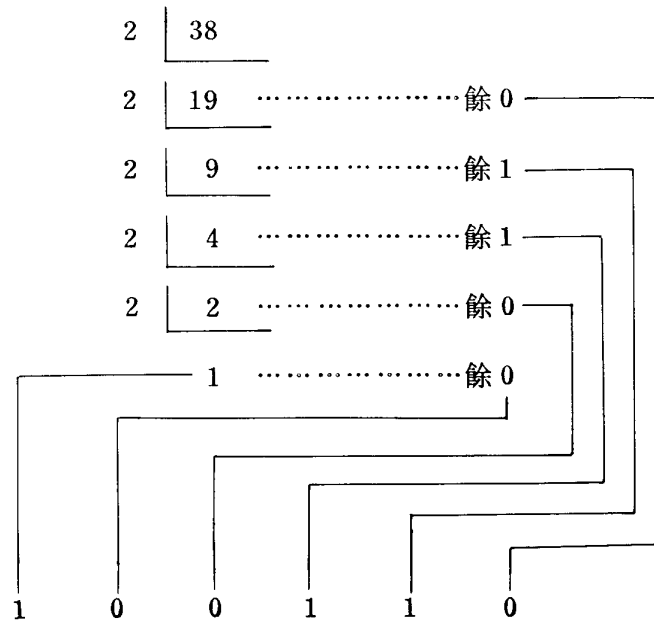
二進制數字系統

我們習慣上使用的十進制有 0~9 十個數字，逢 "十" 進位；而二進制只有 0 與 1 兩個數字，逢 "二" 進位。那麼，二進制與十進制之間如何轉換呢？

1. 十進制轉換為二進制：

(1) 以逐次除以 2 的方法，求出每次除後之餘數。

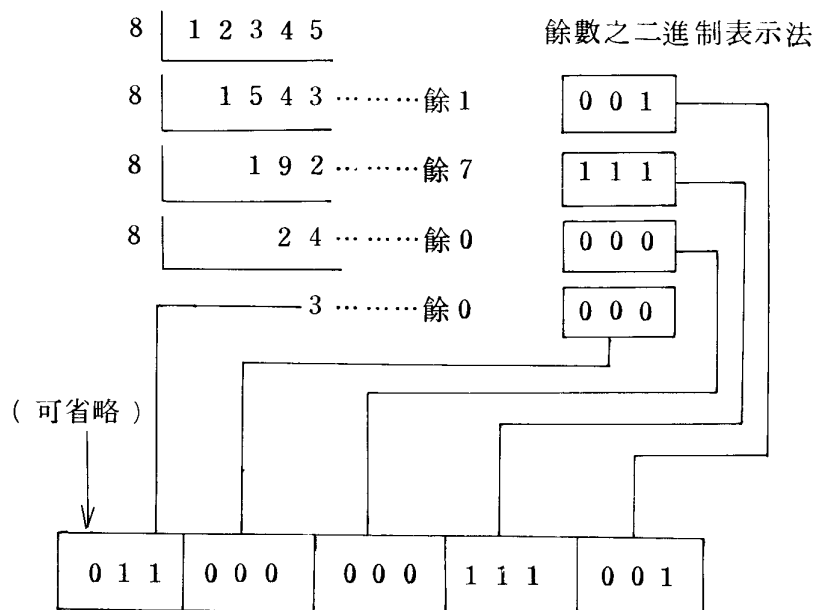
〔例 1〕 38 之二進制表示法：



所以， $38 = (100110)_2$

(2) 當數目很大時，以上述方法計算將會很繁，而且容易出錯。如果你對 0~7 之二進制表示法已很熟悉，可以改用逐次除以 8 的方法比較簡潔。

〔例 2〕 12345 之二進制表示法



每一次的餘數均以三個位元來表示，最後將這些位元依序組合即可。

所以， $12345 = 11000000111001$ 。

0 至 7 之二進制表示法可由表 1-1 查出。

8 位元之二進制表示法列於表 1-2

十進制	0	1	2	3	4	5	6	7
二進制	000	001	010	011	100	101	110	111

表 1-1 0 至 7 的二進制表示法

十 進	二 進	十 進	二 進
0	00000000	32	00100000
1	00000001	33	00100001
2	00000010	:	
3	00000011	:	
4	00000100	:	
5	00000101	:	
6	00000110	63	00111111
7	00000111	64	01000000
8	00001000	65	01000001
9	00001001	:	
10	00001010	:	
11	00001011	:	
12	00001100	:	
13	00001101	127	01111111
14	00001110	128	10000000
15	00001111	129	10000001
16	00010000	:	
17	00010001	:	
:		:	
:		254	11111110
31	00011111	255	11111111

表 1-2 十進制與 8 位元二進制之轉換

2. 二進制轉換為十進制：

(1) 恰與前述方法相反，可以將二進制的每個位元逐次乘 2 相加或逐次乘 8 相加來轉換成十進制。

〔例 3〕將 1011001 轉換為十進制：

$$\begin{array}{ccccccc}
 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
 \swarrow & \swarrow & \swarrow & \swarrow & \swarrow & \swarrow & \swarrow \\
 (((((1 \times 2 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 0) \times 2 + 1 = 89
 \end{array}$$

所以，1011001=89

〔例 4〕將 10110101001 轉換為十進制

1 0	1 1 0	1 0 1	0 0 1
-----	-------	-------	-------

$$((2 \times 8 + 6) \times 8 + 5) \times 8 + 1 = 1449$$

將二進位數從右至左每 3 位分成一組先轉換為十進制，再逐次乘 8 相加即可。
所以，10110101001=1449

(2) 利用每個位元的基量 (Weight) 來轉換。

習慣上為了識別方便，通常將二進制的每個位元從右至左加以編號（從 0 開始編起）。每個位元依其所在位置均代表不同的基量，如表 1-3 所示。

位元編號	7	6	5	4	3	2	1	0	十進值
基量	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1	
例	0	0	0	0	1	0	1	1	11
	0	0	0	1	0	1	0	1	21
	0	0	1	0	1	0	1	0	42
	0	1	0	1	1	0	1	0	90
子	0	1	1	0	0	0	0	1	97
	1	0	1	0	1	0	0	0	168
註：第 n 個位元之基量為 2^n									

表 1-3、利用基量來換算二進制與十進制

利用表 1-3 的基量，可以很容易的將二進制轉換為十進制。

〔例 5〕將 100110101 轉換為十進制。

將二進位數中為 "1" 的位元對照表 1-3，求出其對應的基量，相加之後即為十進位數。

$$256 + 32 + 16 + 4 + 1 = 309$$

所以，100110101=309

十六進制數字系統

使用二進制機器語言來編寫程式時，會有許多困難，諸如：

- (1) 編寫程式所花的時間很長，而且容易出錯。
- (2) 程式冗長，輸入電腦的速度很慢。
- (3) 難以瞭解程式的內容。
- (4) 以後除錯時，很難發覺錯誤。

圖解 6502 指令集

5

爲了解決上述困難，大多數的電腦都具有接受十六進制碼的功能。因爲電腦通常以一個位元組（8 個位元）來代表一個 "字"（Word），如果將每 4 個位元轉換成一個十六進制碼，那麼一個位元組只要用兩個十六進制碼來表示即可。

十六進制數字系統使用十六個符號來表示（0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F）。表 1-4 列出十進制、二進制與十六進制值之間的關係。

十進制	二進制	十六進制
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

表 1-4 十進制、二進制、十六進制對照表

一個位元組共有 8 個位元，所以只能表示 0 至 255 的十進位數，或者 \$0 至 \$FF 的十六進位數。（在本文件中述及十六進位數時均在前面加上符號 \$ 來表示，例如 \$65，\$FF）

〔例 6〕將 235 轉換爲十六進制

$$\begin{array}{r} 16 \overline{) 235} \\ 14 \cdots \cdots \cdots \text{餘 } 11 \\ \hline \end{array}$$

所以，235 = \$E B

〔例 7〕將 (10011100)₂ 轉換爲十六進制

1 0 0 1	1 1 0 0
---------	---------

↓ ↓

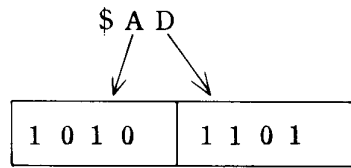
\$ 9 C

〔例 8〕將 \$AD 轉換爲十進制及二進制

$$\begin{array}{c} \$ A D \\ \swarrow \quad \downarrow \\ 10 \times 16 + 13 = 173 \end{array}$$

圖解 6502 指令集

6

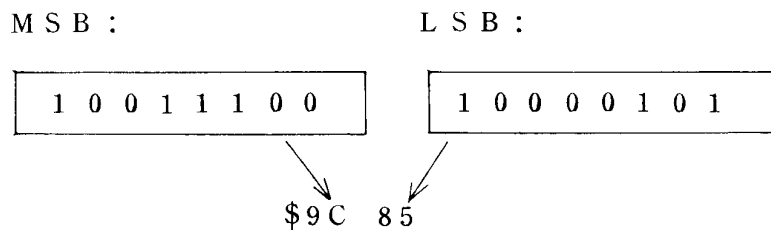


現在我們可以用十六進制來改寫前面所述的那一小段程式了。

二進制程式	十六進制程式
10101101	AD
00000000	00
00000011	03
10001101	8D
00010000	10
00000011	03

這段程式可將記憶位址 \$300 所存的資料移入位址 \$310。

雖然一個位元組只能表示 0 至 255 之間的數值，然而我們可將兩個位元組連在一起使用，以處理更大的數值。兩個位元組合併使用時，我們將它區分為 "高位元組" (Most-Significant Byte，簡稱 MSB) 與 "低位元組" (Least-Significant Byte，簡稱 LSB)。例如，



以十進制表示時，MSB 可以看成 LSB 之延伸，所以其值應為 $(32768 + 4096 + 2048 + 1024 + 128 + 4 + 1) = 40069$ 。

二進制算術

由於電腦是以一個位元組（8 個位元）來代表一個 "字"，因此，以下討論的算術均以 8 個位元為主。當然，我們亦可擴展為兩個位元組、三個位元組.....的算術。

1. 二進制加法與進位旗號：

二進制的加法很簡單，只要依下列規則將位元逐一相加即可。

0+0=0	無進位
0+1=1	無進位
1+0=1	無進位
1+1=0	有進位

要注意的是，在相加過程中，必須同時加上上一位的進位位元。

〔例 1〕求 53 與 113 之和

十進制	二進制	十六進制
	$ \begin{array}{r} 11 \quad 1 \\ \swarrow \quad \searrow \quad \swarrow \\ 00110101 \quad (進位) \\ + 01110001 \\ \hline 10100110 \end{array} $	
53		\$ 3 5
+ 113	+ 0 1 1 1 0 0 0 1	+ \$ 7 1
<hr/> 166	<hr/> 1 0 1 0 0 1 1 0	<hr/> \$ A 6

〔例 2〕求 128 與 129 之和

十進制	二進制	十六進制
	$ \begin{array}{r} 1 \\ \swarrow \\ 10000000 \\ + 10000001 \\ \hline 00000001 \end{array} $	
128		\$ 8 0
+ 129	+ 1 0 0 0 0 0 0 1	+ \$ 8 1
<hr/> 257	<hr/> 0 0 0 0 0 0 0 1	<hr/> \$ 0 1

在最高位元 (Bit7) 產生進位，怎麼辦？

在電腦中有一狀態暫存器 (Processor status register)，含有一個進位旗號 (carry flag) 用來儲存最高位元的進位。如果只執行 8 個位元的加法而進位旗號為 1 時，表示運算結果已超出範圍；如果執行的是多位元組的加法時，電腦會在下一個位元組的運算中，將進位旗號加到最低位元上，如下例。

〔例 3〕求 384 與 129 之和

十進制	二進制	十六進制
	$ \begin{array}{r} 11 \\ \swarrow \quad \searrow \\ 00000001:10000000 \\ + 00000000:10000001 \\ \hline 00000010:00000001 \end{array} $	
384		\$ 0 1 8 0
+ 129	+ 0 0 0 0 0 0 0 0 : 1 0 0 0 0 0 0 1	+ \$ 0 0 8 1
<hr/> 513	<hr/> 0 0 0 0 0 0 1 0 : 0 0 0 0 0 0 0 1	<hr/> \$ 0 2 0 1

2. 二進制減法與借位旗號：

電腦的硬體設計上只有加法器而沒有減法器，所以電腦不能直接做減法運算。那麼電腦如何執行二進制的減法呢？

我們先來看看基本的減法算術。在代數中，可以利用負數的觀念，把減去一個數看成加上這個數的負數，例如， $m - n$ 可以看成 $m + (-n)$ 。

但是，電腦使用二進制系統，無法以負號 "-" 來表示一個負數，所以必須先找出一種負數的表示法。從負數的特性知， $m + (-m) = 0$ ；我們可以利用這個特性在二進制數系中定義一個數的負數。

假設 M 是一個 8 位元的二進制數，若將 M 中為 0 的位元變為 1，為 1 的位元變為 0，可得到另一個數，以 \overline{M} 表示之。 \overline{M} 稱之為 M 的 "1 的補數" (1's complement)。

很明顯的， $M + \overline{M} = 11111111 = \FF ，在 8 位元運算中，\$FF 加 1 之後結果為 \$00

所以 $M + (\overline{M} + 1) = \00

由上式關係知，我們可以把 $(\overline{M} + 1)$ 看成 M 之負數。在二進制中， $(\overline{M} + 1)$ 稱為 M 的 "2 的補數" (2's complement)。

因此，電腦在執行二進制減法時，先將減數轉換成 "2 的補數"，再與被減數相加。

〔例 4〕計算 $\$35 - \24 之結果

$$\$24 = 00100100$$

$$\$24 \text{ 之 "1 的補數"} = 11011011$$

$$+ 1$$

$$\$24 \text{ 之 "2 的補數"} = 11011100 = \$DC$$

所以， $\$35 - \$24 = \$35 + \DC

$$\begin{array}{r} \$ 35 = 00110101 \\ + \$DC = 11011100 \\ \hline \end{array}$$

$$00010001 = \$11$$

所得結果為 $\$11$ ，但有進位產生，進位旗號 $C=1$ 。

〔例 5〕計算 $\$24 - \35 之結果

$$\$35 = 00110101$$

$$\overline{\$35} = 11001010$$

$$\overline{\$35} + 1 = 11001011 = \$CB$$

所以， $\$24 - \$35 = \$24 + \CB

$$\begin{array}{r} \$ 24 = 00100100 \\ + \$CB = 11001011 \\ \hline \end{array}$$

$$11101111 = \$EF$$

所得結果為 $\$EF$ ，無進位產生，進位旗號 $C=0$ ，本例的結果 $\$EF$ 實際上是 $\$11$ 之 "2 的補數"。

由以上兩例，可以得到下面的結論

(1) 進位旗號 $C=1$ 時，所得結果為一正數。

(2) 進位旗號 $C=0$ 時，所得結果為一負數，但以 "2 的補數" 來表示。

通常，在二進制的減法中，把進位旗號 C 的補數 \overline{C} 看做 "借位旗號"，亦即當 $\overline{C}=0$ ($C=1$) 時，表示無借位；當 $\overline{C}=1$ ($C=0$) 時，表示有借位。

如果是執行多位元組的減法時，電腦會將進位旗號 C 加到下一個位元組的最低位元上。如下例。

〔例 6〕計算 $\$0124 - \0035 之結果

$$\$ 0035 = 00000000 \ 00110101$$

$$\overline{\$ 0035} = 11111111 \ 11001010$$

$$\overline{\$ 0035} + 1 = 11111111 \ 11001011 = \$FFCB$$

所以， $\$0124 - \$0035 = \$0124 + \$FFCB$

圖解 6502 指令集

9

C = 0

$$\begin{array}{r}
 \$ 0124 = 00000001\ 00100100 \\
 + \$ FF CB = 11111111\ 11001011 \\
 \hline
 C = 1 \leftarrow 00000000\ 11101111 = \$ 00EF
 \end{array}$$

所得結果為 \$00EF，但有進位產生，C=1。因為 $C=1(\bar{C}=0)$ ，表示沒有借位，所以結果 \$00EF 為一正數。（請比較例 5 與例 6 之差別）

3. 有符號之二進制算術：

由以上討論已知，我們可以用 M 之 "2 的補數" 來表示 M 的負數。但是，對於任意一個數，我們無法判別它是代表正數或者是另一個數的 "2 的補數"。為了解決這個問題，電腦習慣上採用 "有符號的二進制數"。

使用 8 個位元來表示一個數且時，若將第 7 位元（Bit7）看成符號位元，即可由此 8 個位元來表示十進制整數 -128 至 +127。當第 7 位元為 1 時，表示負數；第 7 位元為 0 時，表示正數。因此，\$00 至 \$7F 表示從 0 至 +127 之十進制非負整數，\$80 至 \$FF 表示從 -128 至 -1 之十進制負整數。如表 1-5 所示。（表中所示之負數均為正數之 "2 的補數"，請自行計算即知）

十進制	有符號之二進制	十六進制
+127	0111 1111	\$7F
+126	0111 1110	\$7E
+125	0111 1101	\$7D
:	:	:
:	:	:
:	:	:
+3	0000 0011	\$03
+2	0000 0010	\$02
+1	0000 0001	\$01
0	0000 0000	\$00
-1	1111 1111	\$FF
-2	1111 1110	\$FE
-3	1111 1101	\$FD
:	:	:
:	:	:
:	:	:
-126	1000 0010	\$82
-127	1000 0001	\$81
-128	1000 0000	\$80

表 1-5 有符號之二進制表示法

若要處理 +127 至 -128 範圍以外的數目時，就需要兩個以上的位元組，但仍將最高位元組之第 7 位元當做符號元。因此，16 位元的有符號數可表示從 -32768 至 +32767 之十進制整數。

有符號之二進制算術舉例說明於下：

圖解 6502 指令集

10

〔例 7〕將 -12 與 +7 相加
直接由表 1-5 查出，-12=\$F4，+7=\$07

$$\begin{array}{r}
 \$F4 = 11110100 \\
 + \$07 = 00000111 \\
 \hline
 11111011 = \$FB
 \end{array}$$

所得結果為 \$FB，但因第 7 位元 (Bit7) 為 1，表示負數。

其餘位元之 "2 的補數" 為 5，所以 \$FB 代表 -5。

由於 8 個位元所能表示的十進制範圍只有 -128 至 +127，如果兩數相加的結果超出這個範圍時，會產生什麼結果？

〔例 8〕

$$\begin{array}{r}
 \overset{1}{\curvearrowright} 01000000 \quad (64) \\
 + 01000001 \quad (65) \\
 \hline
 = 10000001 = (-127)
 \end{array}$$

64 與 65 相加結果應為 129，已超出 8 位元的 "有符號二進制數" 的表示範圍，所以運算的結果 10000001 以有符號數表示時，變成 -127，顯然是錯誤的。這是由於第 6 位元相加時產生進位而使第 7 位元改變 "符號" 所致。

電腦在處理有符號之二進制算術時，當兩正數之和超過 01111111 (+127)，則在狀態暫存器中將溢位旗號 V (Overflow flag) 置定為 "1"，表示運算結果超出範圍 (溢位)。

〔例 9〕

$$\begin{array}{r}
 \overset{1}{\curvearrowright} 11000000 \quad (-64) \\
 + 10111111 \quad (-65) \\
 \hline
 = 01111111 = (+127)
 \end{array}$$

本例中，第 7 位元相加的結果產生進位，以致第 7 位元改變了 "符號"，所以產生一個錯誤的結果。也就是說，兩負數相加之和超過 -128 (10000000)，亦會發生 "溢位"，電腦會將溢位旗號 V 置定為 "1"。

〔例 10〕

$$\begin{array}{r}
 \overset{1}{\curvearrowright} \overset{1}{\curvearrowright} 11111110 \quad (-2) \\
 + 11111101 \quad (-3) \\
 \hline
 11111011 = (-5)
 \end{array}$$

本例中，第 6 位元相加的結果產生進位，而第 7 位元相加亦產生進位。但是，最後所得的結果卻仍然正確。

由上面三個例子，可以得到下面幾點結論：

(1) 溢位的情形只會發生在：

a 內個正數之和 (或一個正數減去一個負數) 超過 +127 時。

- b 兩個負數之和（或一個負數減去一個正數）超過 -128 時。
- (2) 電腦判定溢位發生（置定溢位旗號 V 為 1）是在：
- a 第 6 位元相加產生進位，而第 7 位元相加沒有進位時。
- b 第 6 位元相加沒有進位，而第 7 位元相加產生進位時。

十進制二進位碼（BCD）

電腦雖然使用二進制來處理資料，但是人類卻對十進制比較習慣，希望在輸入資料或讀取資料時以十進制來表示比較方便。為了使用方便，電腦大多設計了以十進制來處理二進位碼的功能。這種數字系統稱為十進制二進位碼（Binary-code Decimal，簡稱 BCD 碼）。

BCD 碼的構成與傳統的二進制碼類似，將十進制的每一個數字（0~9）均以 4 個位元來表示，如表 1-6 所示。

十進制	BCD 碼
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
註：1010，1011 ，1100，1101 ，1110，1111 均不使用	

表 1-6 BCD 碼之表示法

因此，一個位元組可以表示兩位數的十進制數，例如，0011 1000 表示十進制的 38，而不是 56。BCD 碼的算術與二進制算術有很大的差異，例如，想計算 38 與 24 之和時，以 BCD 碼表示為

$$\begin{array}{r}
 0011 \quad 1000 \\
 + 0010 \quad 0100 \\
 \hline
 0101 \quad 1100
 \end{array}$$

若模仿二進制加法，則所得結果為 0101 1100。因為 1100 在 BCD 碼中並未使用（參考表 1-6），所以我們無法表示此運算結果。

那麼，電腦如何處理 BCD 碼的運算呢？

1. BCD 碼的加法

處理 BCD 碼的加法時，只要先算出二進制加法的結果，然後依照下述規則加以修正即可。

圖解 6502 指令集

12

(1) 兩數字之和大於 9 而小於 16 時，總和必須加 6 才能得到正確的結果。例如，

$$\begin{array}{r}
 0101 \quad (5) \\
 + 1000 \quad (8) \\
 \hline
 1101 \dots\dots\dots \text{不是 BCD 碼} \\
 + 0110 \dots\dots\dots \text{加 6} \\
 \hline
 0001 \quad 0011 \dots\dots\dots \text{正確的 BCD 碼 (13)}
 \end{array}$$

(2) 兩數字之和大於或等於 16 時，總和仍須加 6 才是正確的結果。例如，

$$\begin{array}{r}
 1000 \quad (8) \\
 + 1001 \quad (9) \\
 \hline
 0001 \quad 0001 \dots\dots\dots \text{雖為 BCD 碼，但因進位而產生錯誤} \\
 \hspace{10em} \text{結果} \\
 + 0110 \dots\dots\dots \text{加 6} \\
 \hline
 0001 \quad 0111 \dots\dots\dots \text{正確的 BCD 碼 (17)}
 \end{array}$$

2. BCD 碼的減法

處理 BCD 碼的減法時，不能以二進制的方法求得，因為 BCD 碼是十進位的，所以必須把減數先轉換成 "10 的補數" (10's complement)，而以加法來運算。

在十進制中，兩數相加的結果，若每位數字均為 "9" 時，則這兩數互稱為 "9 的補數"。例如，2 的 "9 的補數" 為 7；73 的 "9 的補數" 為 26，以此類推。

將某數的 "9 的補數" 加 1 之後，即為該數的 "10 的補數"。例如，73 的 "10 的補數" 為 27。現舉例說明 BCD 碼的減法。

〔例 11〕 求 25 減去 16 之結果

先求出 16 的 "9 的補數" 為 83，再加 1 即為 16 的 "10 的補數"——84。

$$\begin{array}{r}
 0010 \quad 0101 \quad (25) \\
 + 1000 \quad 0100 \quad (84) \\
 \hline
 \text{不是 BCD 碼} \dots\dots 1010 \quad 1001 \\
 + 0110 \quad \dots\dots\dots \text{加 6} \\
 \hline
 0000 \quad 1001 \quad (9)
 \end{array}$$

進位，C=1

本例中，因為十位數相加結果為 1010，不是 BCD 碼，所以必須加 6 來修正。修正之後，有進位產生 (C=1 時：表示運算結果為正數)，所以最後結果 "9" 是正確答案。

〔例 12〕 求 28 減去 73 之結果

73 之 "10 的補數" 為 27。

$$\begin{array}{r}
 0010 \quad 1000 \quad (28) \\
 + 0010 \quad 0111 \quad (27) \\
 \hline
 0100 \quad 1111 \quad \text{.....不是 B C D 碼} \\
 + \quad \quad \quad 0110 \quad \text{.....加 6} \\
 \hline
 0101 \quad 0101
 \end{array}$$

無進位，C=0

本例中，個位數相加結果為 1111，不是 BCD 碼，所以必須加 6 來修正。最後的結果為 55，且無進位產生（C=0，表示運算結果為負數），所以 55 所表示的值，實際上是 -45（55 之 "10 的補數"）。

第二章 6502 暫存器／定址法說明

6502 暫存器說明：

- (1) 累積器 (Accumulator)：這是一個多用途的暫存器，記憶體內的資料轉移或存取都需要利用它；很多算術或邏輯運算亦直接利用它來存放運算結果。
- (2) 索引暫存器 X, Y (Index register X, Y)：用於索引定址或資料轉移的暫存器。
- (3) 堆疊暫存器 (stack point register)：在記憶體內有一個固定位置的堆疊區 (stack area)，此堆疊區由一個長度為 8 個位元的堆疊指向器 (stack pointer) 所指引，所以堆疊區所佔的記憶空間為 256 個位元組。堆疊指向器只有 8 個位元，如何指向記憶體中的堆疊位址 (stack address) 呢？原來 CPU 把堆疊指向器的 8 個位元當做低位元組 (LSB)，而固定在高位元組補上 \$01，所以堆疊區在記憶體中所佔的位址從 \$0100 至 \$01FF。堆疊區以 "後進先出" (Last-in first-out) 的形式來堆放資料，也就是說，最後存入的資料最早取出，我們無法從堆疊區的中間取出資料。其實際功用將在以後詳述。
- (4) 狀態暫存器 (status register)：包含了 6 個狀態旗號 (status flag) 及一個插斷控制旗號，如下圖所示：

7	6	5	4	3	2	1	0
N	V		B	D	I	Z	C

進位旗號 C (carry flag)：用來存放算術運算後最高位元之進位。

零值旗號 Z (Zero flag)：算術或邏輯運算後，若得到的結果為 0，則此旗號被置定為 "1"，否則置定為 "0"。

插斷控制旗號 I (Interrupt control flag)：用來控制 "允許或禁止" 插斷。當 I=1 時，禁止插斷；I=0 時，允許插斷。

十進位模式旗號 D (Decimal mode flag)：當 D=1 時，使得 ADC 及 SBC 兩個指令可以執行十進制二進位碼 (BCD 碼，Binary Coded Decimal) 的運算；D=0 時，執行一般的二進位運算。

中斷旗號 B (Break flag)：執行中斷指令 BRK 時 CPU 自動將 B 置定為 "1"。

溢位旗號 V (Overflow flag)：在執行有符號的二進位算術運算時，若運算後的值太大而無法表示時，CPU 自動將溢位旗號置定為 "1"。(詳見第一章)

負值位元 N (Negative bit，又稱 Sign bit)：運算或存取的數值資料為負數時，N=1；否則 N=0。

bit 5 沒使用，通常為 "1"。

定址模式 (Addressing Model)

6502 可以執行 56 種不同的指令，但是若以不同的定址模式加以組合，則可執行 151 個不同的功能。所以，若想有效、靈活的運用每個指令，必須先對定址模式有徹底的瞭解。

假設我們想執行 LDA 這個指令，它代表 "將記憶位址內的資料存入累積器"；可是，微處理機如何知道從那一個記憶位址得到資料呢？這就需要利用到定址模式了。

6502 的定址模式可分為九大類：

- (1) 立即 (immediate) 定址法。
- (2) 直接定址法：分為絕對 (absolute) 定址法與零頁 (Zero page) 定址法。
- (3) 間接定址法。
- (4) 索引 (indexed) 定址法：分為絕對索引 (absolute indexed) 與零頁索引 (Zero page indexed) 定址法。
- (5) 先索引間接 (Pre-indexed indirect) 定址法。
- (6) 後索引間接 (Post-indexed indirect) 定址法。
- (7) 相對 (relative) 定址法。
- (8) 隱含 (implied) 或固有 (inherent) 定址法。
- (9) 累積器 (accumulator) 定址法。

不同的定址法各有不同的運算碼對應，微處理機由運算碼即可知道使用那一種定址法來執行指令。

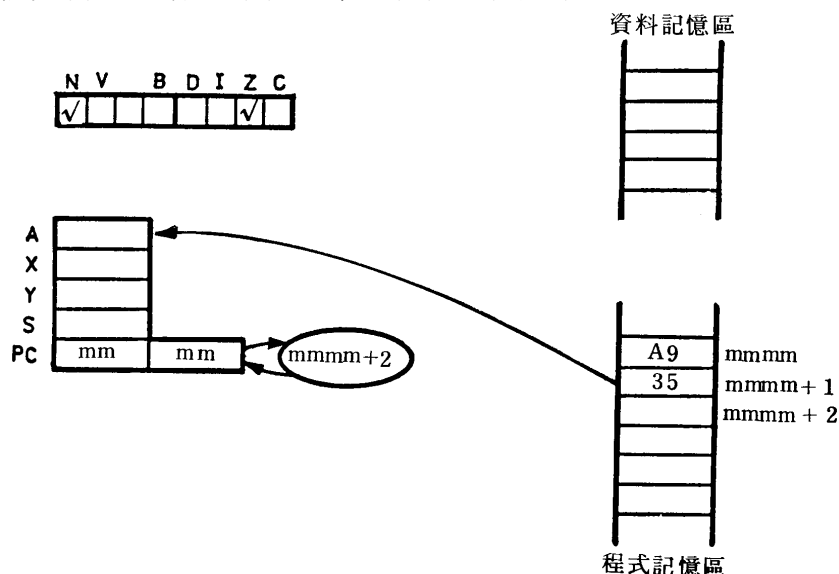
1. 立即定址法

這種形式的定址是在運算碼之後，緊跟著運算元，此運算元為立即運算所需的資料，通常我們在這種運算元之前加上符號 "#" 以表示其為立即運算元。

例如：

LDA #\$35

表示立即將數值資料 \$35 存入累積器中。執行過程如下：



註：上圖中，把記憶體分為 "資料記憶區" 與 "程式記憶區" 只是為了說明方便。實際上，記憶體並沒有做這類區分。

註：狀態暫存器中的旗號 (Flag)，打 "√" 的旗號表示會受此指令的影響，其值須由執行結果來決定。

2. 直接定址法

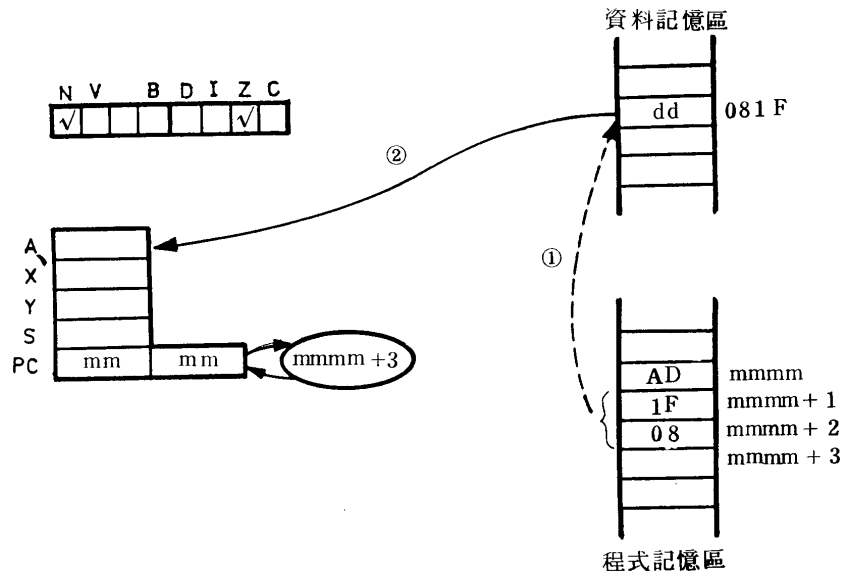
分為絕對定址與零頁定址兩種：

絕對定址法：運算碼之後接了兩個位元組，這兩個位元組指定了運算元所在的記憶位址。

例如：

LDA \$081F

表示將記憶位址 \$081F 所存的資料存入累積器中。



註：圖中的符號①、②.....表示指令執行的順序，但並不是 CPU 真正的執行步驟，只是為了幫助您瞭解指令及定址法的功能而已。另外，虛線部份表示 "找出對應記憶位址的內容"。

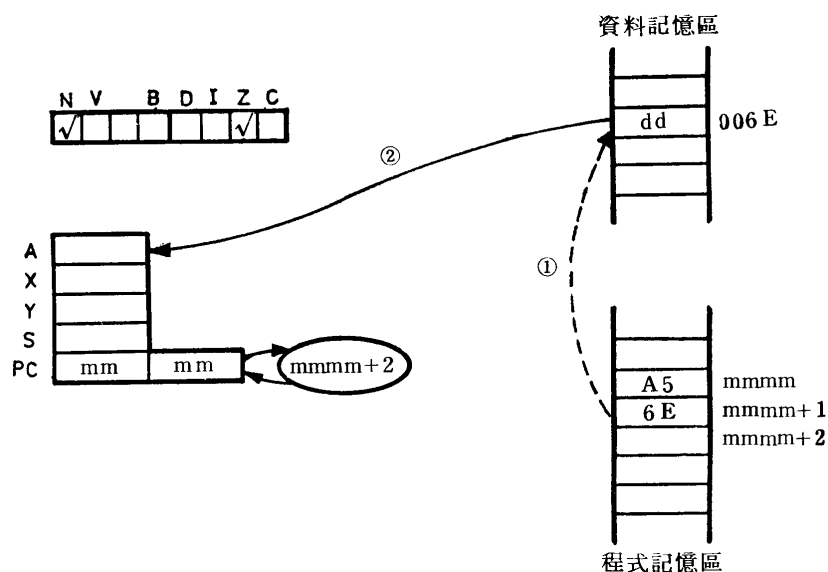
必須特別注意的是，16 位元的位址之儲存方式為：先存 "低位元組" (LSB)，再存 "高位元組" (MSB)，如上圖之程式記憶區所示。

零頁定址法：如果運算元所在的記憶位址介於\$00 與\$FF 之間（零頁記憶空間），則在運算碼之後只需一個位元組來表示即可。

例如：

LDA \$6E

表示將記憶位址\$006E 內的資料存入累積器中。



注意，雖然使用同一個指令 LDA，但由於定址法不同，所以其運算碼亦各不相同，不致混淆

不清。

3．間接定址法

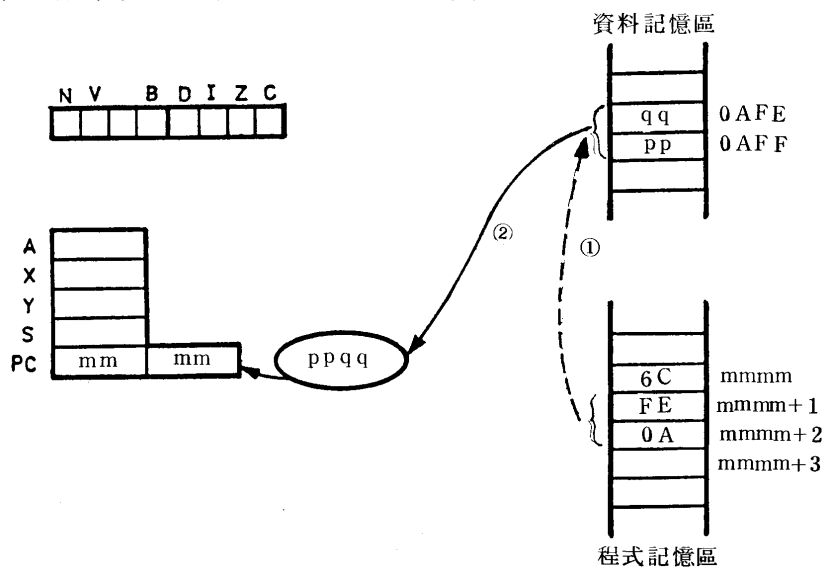
間接定址法僅能用於 **JMP** 指令（跳到新位址執行）。運算碼之後緊跟著兩個位元組，到這兩個位元組所代表之記憶位址中提取連續兩個位元組的資料，將它存程式計數器內，此兩位元組的資料即為所要跳越的新位址。

例如：

JMP (\$0AFE)

將 \$0AFE 與 \$0AFF 內所含的資料 \$qq 與 \$pp 存入程式計數器中，那麼程式會跳到新位址 \$ppqq 繼續執行。

在間接定址法中，指令後面的位址均加上括號來表示。



4．索引定址法

分為絕對索引與零頁索引兩種：

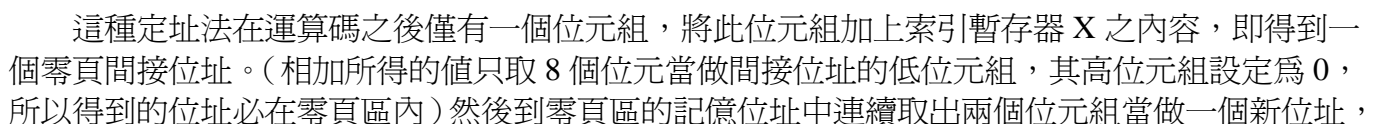
絕對索引定址法：運算碼之後有兩個位元組用來指定基底位址（Base address），然後將索引暫存器 X 或 Y 之內容加上基底位址以得到有效的絕對位址。

例如：

LDA \$0AFE, Y

假設索引暫存器 Y 的內容為 yy，則執行此指令時，會將記憶位址 \$(0AFE + yy) 之內容存入累積器中。

18



再根據這個新位址到記憶體中取出資料當做運算元。

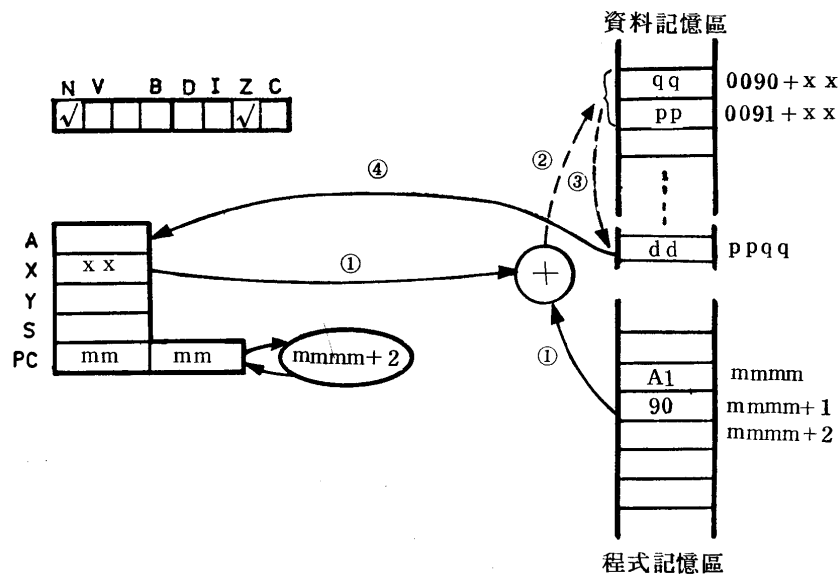
例如：

LDA (\$90, X)

假設 X 暫存器之內容為 \$xx，則執行此指令時，會先到記憶位址 \$(xx+90) 及 \$(xx+91) 找出其內容 \$qq 及 \$pp，然後取出位址 \$ppqq 之內容，將它存入累積器中。

這種定址方法可以看成索引定址與間接定址之組合，由其名稱可知，先做索引定址，再做間接定址，所以指令後面的位址表示法為 (\$90, X)。

注意：這種定址法只能使用 X 暫存器。

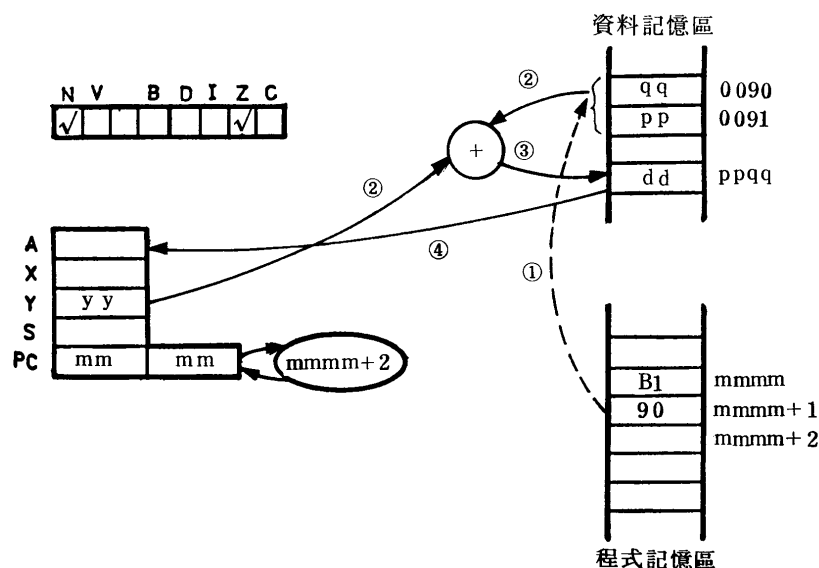


6. 後索引間接定址法

運算碼之後僅有一個位元組，根據這個位址到零頁區中連續取出兩個位元組當做一個新位址，然後將此新位址加上 Y 暫存器之內容而得到一個有效位址，再從這個有效位址的記憶體內取出資料來運算。

例如：

LDA (\$90)



先至零頁區的位址 \$90, \$91 取出其內容 \$qq, \$pp，然後至位址 \$(ppqq+yy) 取出資料存

入累積器中。(假設 Y 暫存器之內容為 \$yy)。

這種定址法是先做間接定址，再做索引定址（恰如其名），所以指令後面的位址表示法為 (\$90), Y

這種定址法只能使用 Y 暫存器。

7. 相對定址法

所有 6502 的條件分支指令均利用相對定址法。這種定址法在運算碼之後只有一個位元組，這個位元組是個有符號的二進制數，它代表分支時所要跳越的相對位址。

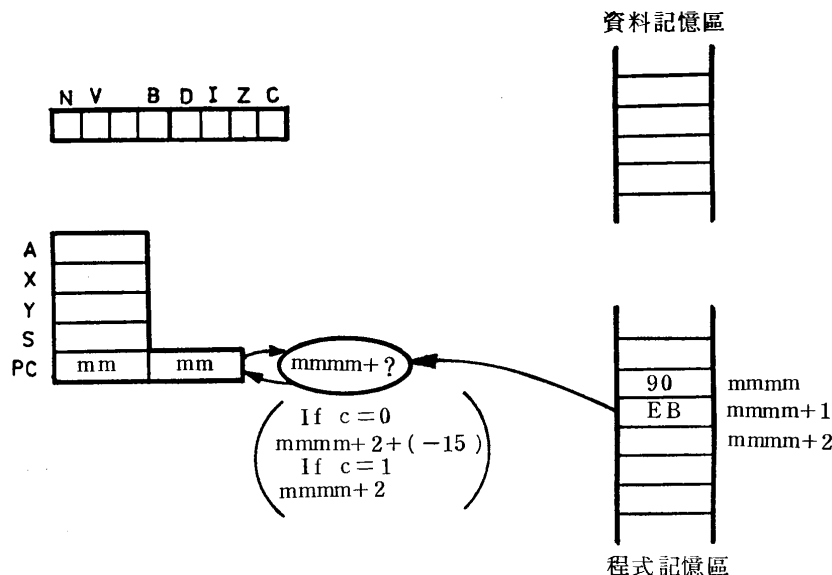
所謂有符號的二進位數是將位元組的最高位元 (bit 7) 當做符號，當此位元為 0 時，表示正數，前面 7 個位元代表它的數值；若最高位元為 1，表示負數，前面 7 個位元之 "2 的補數" 即為其數值。(請參考第一章)

相對位址是由下一個指令的位址開始算起。因為條件分支指令的格式佔了兩個位元組 (1 個運算碼，1 個運算元)，所以下一個指令位址為程式計數器內容加 2。相對位址的表示範圍從 -128 至 +127，所以分支跳越的範圍為條件分支指令算起 -126 至 +129。

現以 BCC 指令為例，說明這種定址法的功能。組合語言的編寫格式如下：

記憶位址	指令碼	標記	指令名稱	運算元	註 解
1000	90	?	BCC	\$0FED	
1002			下一個指令		

指令 BCC 的意義是 "若進位位元 C (狀態暫存器之 BIT0) 為 0 時，則分支跳越至指定的位址，否則後續下一個指令"。假設我們希望，當 $C=0$ 時跳至位址 \$0FED 則以 \$0FED 減去下一個指令的位址 (\$1002)，將所得的值 (不考慮借位) 取其低位元組，即為所要的相對位址。本例中， $\$0FED - \$1002 = \$FFEB$ ，所以相對位址為 \$EB，將它存入主程式的記憶位址 \$1001 中。



8 · 隱含或固有定址法

這種形式的定址法並未牽涉到任何位址即可執行指令，所以只需一個運算碼即可。
例如：

記憶位址	指令碼	標記	指令名稱	運算元	註	解
1000	AA		TAX			

可將累積器之內容存入 X 暫存器。

9 · 累積器定址法

這種形式的指令僅運算累積器的資料。6502 中這種形式的指令僅有：

ASL	(算術左移：Arithmetic shift Left)
LSR	(邏輯右移：Logical Shift Right)
ROL	(經進位位元左旋轉：Rotate Left through carry)
ROR	(經進位位元右旋轉：Rotate Right through carry)

例如：

記憶位址	指令碼	標記	指令名稱	運算元	註	解
1000	0A		ASL			

可將累積器內之 8 個位元向左移一個位置，同時將最高位元移入狀態暫存器之進位位元 C，最低位元存入 0。

第三章 6502 指令群

6502 指令群包括 56 個指令，每個指令可能有數種不同的定址法，所以 6502 共有 151 運算碼 (OP Code)，可執行不同的功能。

本章將 6502 指令依英文字母順序逐一詳細介紹，為了讓讀者便於瞭解和記憶，每個指令的介紹都分為七個部份：

1. 指令名稱及英文全名：指令名稱只是一種助憶符號，有時無法直接看出它的意義，瞭解其英文全名可以幫助我們記住它的功能。例如，TAX 之英文全名為 Transfer Accumulator into X register，可知它的功能為 "將累橫器 A 的內容存到 X 暫存器"。
2. 運算功能：以簡單的符號來表達指令的功能，使讀者能一目了然。例如，TAX 以 " $X \leftarrow [A]$ " 來表達。
3. 指令碼格式：顯示運算碼的 8 個位元以及其他位元組。
4. 說明：補充說明指令的功能與注意事項。
5. 定址模式：以表格列出指令在不同定址法下的運算碼、位元組個數、執行週期個數及特定位元。(註：CPU 的時序速度為 1MHz，所以一個執行週期為 1.0us (微秒)，表中標註星號 * 者表示當跳到另一頁區存取資料時，執行週期增加 1 次。)
6. 狀態旗號：將狀態暫存器內的旗號變化情形列出：
 - 空白：表示旗號不受指令的影響。
 - "✓"：表示執行指令時，旗號會受影響，置定為 "0" 或 "1" 視執行情形而定。
 - "0"：表示執行之後旗號被清除為 0。
 - "1"：表示執行之後旗號被置定為 1。
 - "M₆"：在 BIT 指令執行之後，溢位旗號 V 為被測位元組之第 6 位元。
 - "M₇"：在 BIT 指令執行之後，負值旗號 N 為被測位元組之第 7 位元。(請參考 BIT 指令)
7. 實例：以 "執行過程圖" 舉例說明指令的定址法及執行過程。圖中的符號①、②.....等，表示執行的步驟；實線箭頭表示資料的移動或運算；虛線箭頭表示 "找出對應記憶位址的內容"。另外，為了說明方便，把記憶體分為 "資料記憶區" 與 "程式記憶區"。

本書所用的符號：

- M 記憶體，以 [M] 表示記憶體之內容。
- A 累橫器，其內容以 aa 或 [A] 表示。
- X 索引暫存器，其內容以 xx 或 [X] 表示。
- Y 索引暫存器，其內容以 yy 或 [Y] 表示。
- S 堆疊暫存器，其內容以 ss 或 [S] 表示。
- PC 程式計數器，以 [PC] 表示 16 位元的內容。
- PCH 程式計數器的高位元組，以 [PCH] 或 mm 表示其內容。
- PCL 程式計數器的低位元組，以 [PCL] 或 nn 表示其內容。
- P 狀態暫存器，各狀態旗號如下所述：
 - N：負值旗號。N=0 表示正值；N=1 表示負值。
 - V：溢位旗號。V=0 表示無溢位；V=1 表示溢位。
 - B：中斷旗號。B=0 表示未執行 BRK 指令；B=1 表示執行了 BRK 指令。
 - D：十進位模式旗號。D=0 表示二進位；D=1 表示十進位。

圖解 6502 指令集

23

I：插斷控制旗號。I=0 表示允許插斷；I=1 表示禁止插斷。

Z：零值旗號。Z=0 表示運算結果不為 0；Z=1 表示結果為 0。

C：進位旗號。C=0 表示無進位；C=1 表示有進位。

STACK 堆疊區，以〔STACK〕表示堆疊區最上層之資料。

ADDRESS 16 位元的記憶位址，有時以 \$ppqq 表示之。

addr 8 位元之位址，可能為某記憶位址的低位元組或高位元組。

data 8 位元之資料，有時以 dd 表示其值。

rr 表示某一零頁位址。

qq 表示記憶位址的低位元組。

pp 表示記憶位址的高位元組。

〔 〕 括號內之記憶位址或暫存器的內容。例如，〔A〕表示累積器之內容，〔FFFE〕表示位址 \$FFFE 之內容。

∧ 邏輯 "及" 運算 (AND)。

∨ 邏輯 "或" 運算 (OR)。

⊕ 邏輯 "互斥或" 運算 (Exclusive OR)。

← 資料轉移。

ADC Add with carry (包含進位旗號的加法)

運算功能： $A \leftarrow [A] + [M] + C$

指令碼格式：

011bbb01	addr/data	addr
----------	-----------	------

說明：將記憶位址的內容或資料加上累積器的內容及進位旗號，其結果存於累積器中。

注意：

(1) ADC 可用於十進位或二進位加法，使用之前必須先將十進位模式旗號 **D** 設定為正確的。值。(D=1 時為十進位加法)

(2) 若想執行不帶進位旗號的加法時，可先用 CLC 指令（清除進位旗號）將進位旗號置定為 0。

定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 (<i>IND,X</i>)	後索引 間接 (<i>IND</i>),Y	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE	69	6D	65		7D	79	75		61	71			
BYTES	2	3	2		3	3	2		2	2			
CYCLES	2	4	3		4*	4*	4		6	5			
bbb	010	011	001		111	110	101		000	100			

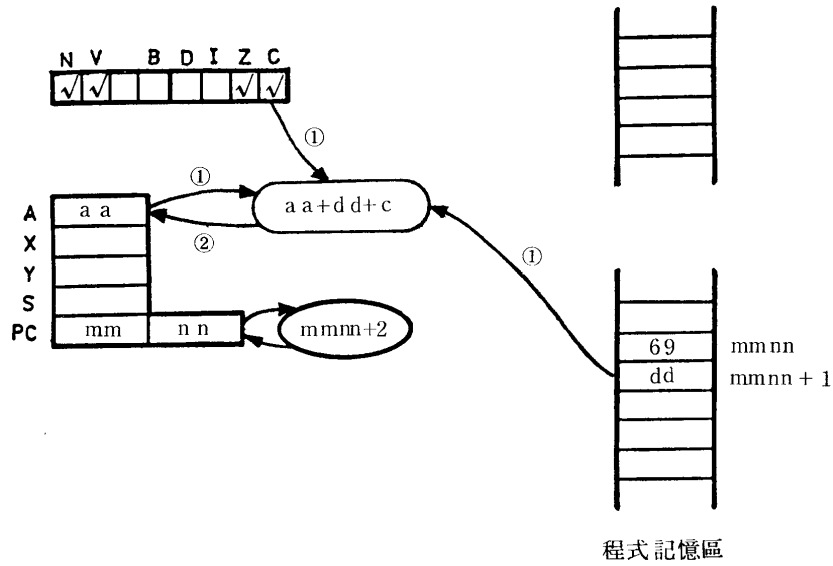
狀態旗號：

N	V		B	D	I	Z	C
✓	✓					✓	✓

圖解 6502 指令集

25

實例： ADC 指令的立即定址法： ADC # $\$dd$



假設：
 $aa = \$3A$
 $dd = \$7C$
 $C = 1$

執行 ADC # $\$7C$
 則累積器之內容變為 $\$B7$ 。運算過程如下

$\$3A = 00111010$	
$\$7C = 01111100$	
旗號 C =	1
<hr/>	
10110111 ($\$7B$)	

無進位， $C = 0$
 置定 $N = 1$
 無溢位， $V = 0$
 結果不為 0，所以零值旗號 Z 置定為 0。

AND logical AND (邏輯"及"運算)

運算功能： $A \leftarrow [A] \wedge [M]$

指令碼格式：

011bbb01	addr/data	addr
----------	-----------	------

說明：將記憶位址的內容或特定資料與累積器內容做邏輯 "及" (AND) 運算，其結果存入累積器。

"及" 運算的真值表為：

A \ M	0	1
0	0	0
1	0	1

定址模式

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引間接 (<i>IND,X</i>)	後索引間接 (<i>IND,Y</i>)	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE	29	2D	25		3D	39	35		21	31			
BYTES	2	3	2		3	3	2		2	2			
CYCLES	2	4	3		4*	4*	4		6	5*			
bbb	010	011	001		111	110	101		000	100			

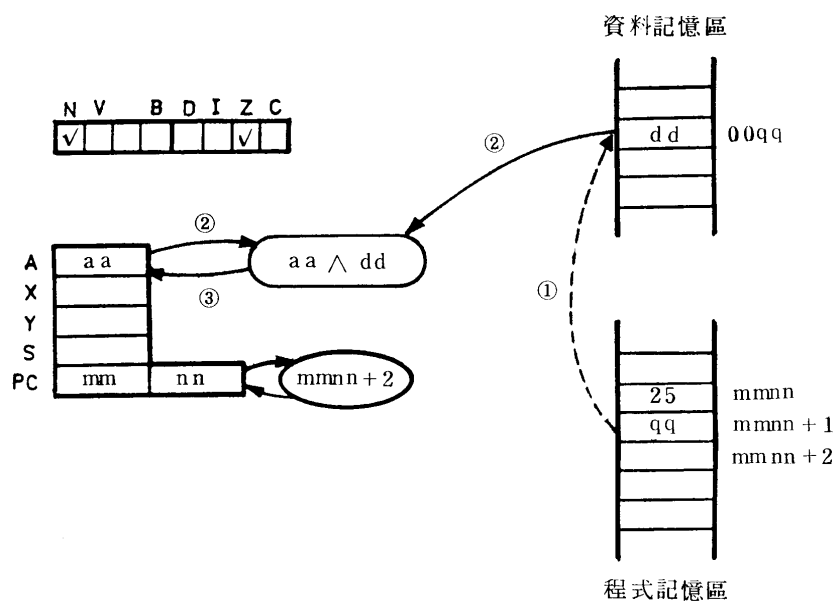
狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

圖解 6502 指令集

27

實例： AND 之零頁定址法 AND \$qq



假設： aa=\$FC
qq=\$40
dd=\$13 (位址 \$0040 之內容)

執行 AND \$40

結果累積之內容變為\$10，運算過程如下：

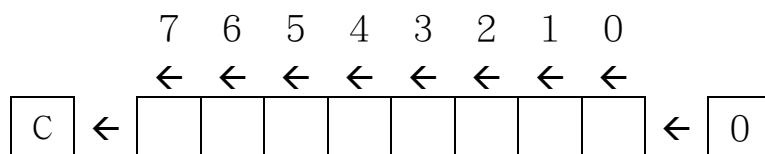
$$\begin{array}{r}
 FC = 11111100 \\
 13 = 00010011 \\
 \hline
 00010000 \text{ ($10)}
 \end{array}$$

置定 N=0

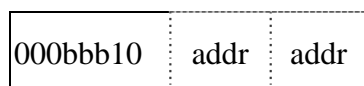
運算結果不為 0，所以 Z=0

ASL Arithmetic Shift Left (算術左移)

運算功能：



指令碼格式：



說明：將累積器或記憶位址的內容，依序向左移一位元，最高位元（Bit 7）移至進位旗號，最低位元（Bit 0）置定為 0。

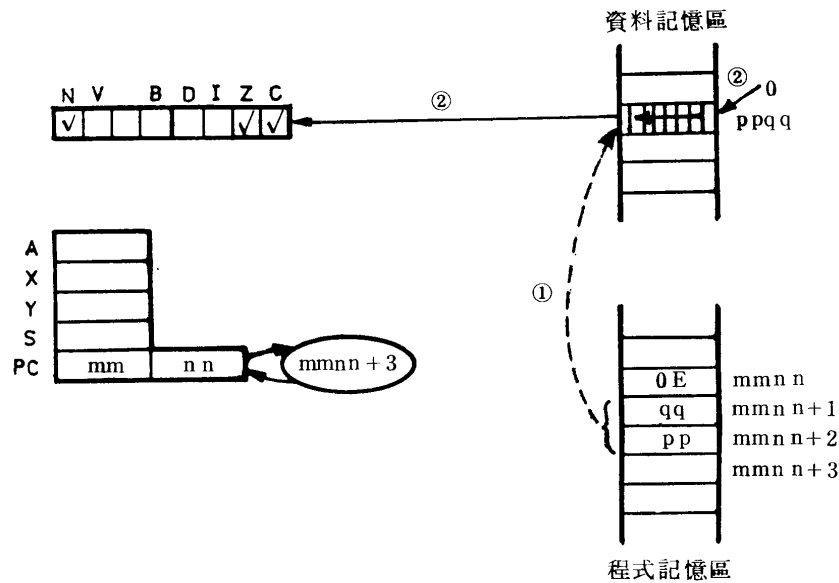
定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 <i>(IND,X)</i>	後索引 間接 <i>(IND),Y</i>	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE		0E	06		1E		16						0A
BYTES		3	2		3		2						1
CYCLES		6	5		7		6						2
bbb		011	001		111		101						010

狀態旗號：

N	V	B	D	I	Z	C
✓					✓	✓

實例： ASL 之絕對（直接）定址法： ASL \$ppqq



假設 $ppqq = \$3F86$

$[ppqq] = \$CB$ (11001011)

執行 ASL $\$3F86$

結果 $[3F86] = \$96$ (10010110)，且進位旗號 C 被設定為 1，且因移位之後結果不為 0，所以 $Z=0$ 。

BCC Branch on Carry Clear（進位旗號清除時，則分支跳越）

運算功能： 若進位旗號 $C=0$ ，則跳越至指定位址執行。

指令碼格式：

10010000	相對位址
----------	------

說明：檢查進位旗號 C，若 $C=0$ ，則跳越至指定位址（此位址為程式計數器之內容加上有符號的相對位址）。若 $C=1$ ，繼續執行下面的程式，不分支跳越。有符號的相對位址值必須在 -128 至 $+127$ （ $\$80$ 至 $\$7F$ ）之間，其計算方法請參考第二章之 "相對位址定址法"

定址模式： 只能使用相對定址法

OP Code = 90

BYTES = 2

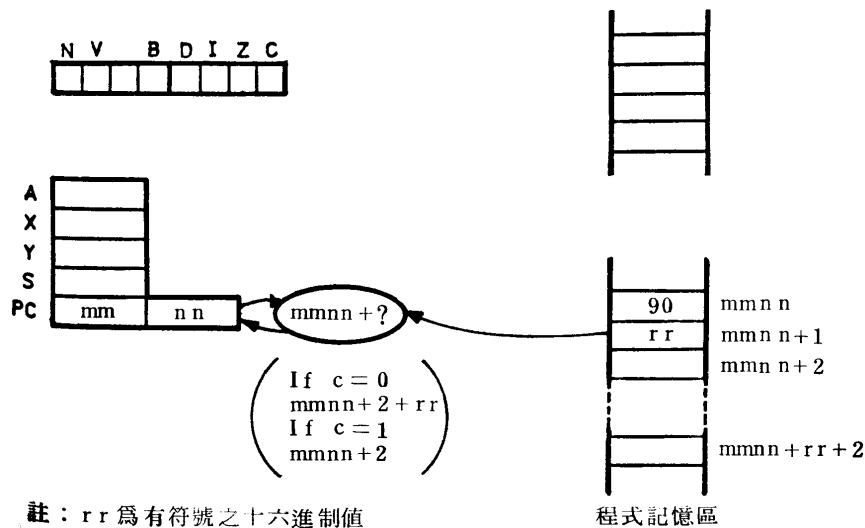
CYCLES = 2 + 1（若 $C=0$ 而分支跳越時）

+ 2（若跳到另一頁區時）

狀態旗號：不受 BCC 指令的影響。

圖解 6502 指令集

30



假設程式記憶區從位址 \$1000 開始（即 $mmnn = 1000$ ），我們希望當 $C = 0$ 時跳到位址 \$0FED 執行，必須先算出分支跳越的相對位址。將 \$0FED 減去下一個指令的位址（\$1002）而將所得的值（不考慮借位）取其低位元組，即為跳越的相對位址。本例中， $\$0FED - \$1002 = \$FFEB$ ，所以對位址為 EB。標準的指令寫法為：

記憶位址	指令碼	標記	指令名稱	運算元	註解
1000	90 EB		BCC	\$0FED	
1002			下一個指令		

圖中的各變數值即為：

$mmnn = 1000$ ， $rr = EB$

執行上述程式時，若 $C = 0$ ，則跳越到 \$0FED 繼續執行；若 $C = 1$ ，則程式從下一位址（\$1002）繼續執行。

註：若直接編寫組合程式時，只要在跳越指令後面加上要跳越的所位址，電腦在翻譯成機器語言時會自動換算成相對位址。

若是以人工組合的方式直接編寫機器語言，必須自己換算成相對位址。

BCS Branch on Carry Set（進位旗號被設定為 1 時，分支跳越）

運算功能：若進位旗號 $C = 1$ ，則跳越至指定位址執行。

指令碼格式：

10110000	相對位址
----------	------

說明：檢查進位旗號 C ，若 $C = 0$ ，程式從下一位址繼續執行；若 $C = 1$ ，則跳越至指定位址（恰與 BCC 指令作用相反，請參考 BCC 指令）

定址模式：只能使用相對定址法。

OP Code = B0

BYTES = 2

CYCLES = 2 + 1（若 $C = 1$ 而分支跳越時）

+2（若跳越至另一頁區時）

狀態旗號：不受 BCS 指令的影響。

BEQ Branch if Equal to zero（零值時，支分越跳）

運算功能：零值旗號 $Z=1$ 時，跳越至指定位址執行。

指令碼格式：

11110000	相對位址
----------	------

說明：檢查零值旗號 Z ，若 $Z=0$ ，程式繼續執行下一個指令；若 $Z=1$ （表示結果為零），則跳越至指定位址執行。此指定位址由程式計數器的內容加上相對位址而得。

定址模式：只能使用相對定址法。

OP Code = F0

BYTES = 2

CYCLES = 2 + 1（若 $Z=1$ 而分支跳越時）
+ 2（若跳越至另一頁區時）

狀態旗號：不受 BEQ 指令的影響。

BIT compare memory Bits with accumulator (比較記憶內容與累積器的各位元)

運算功能： $Z \leftarrow \overline{[A] \wedge [M]}$ ， $N \leftarrow [M_7]$ ， $V \leftarrow [M_6]$

指令碼格式：

0010b100	addr	addr
----------	------	------

說明：BIT 指令將記憶內容與累積器做邏輯 "及" 運算 (logical AND)，但是執行結果並不存入累積器中，而是利用執行結果來設定零值旗號 Z。當記憶內容與累積器相同時，邏輯 "及" 運算結果為零，所以零值旗號 Z 設定為 "1"；否則，Z=0。另外，將記憶內容的第 6、第 7 位元分別存入溢位旗號 V 及負值旗號 N 中。

定址模式： 只有絕對定址與零頁定址兩種。

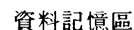
	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引間接 (<i>IND,X</i>)	後索引間接 (<i>IND</i>),Y	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE		2C	24										
BYTES		3	2										
CYCLES		4	3										
bbb		1	0										

狀態旗號：

N	V		B	D	I	Z	C
M ₇	M ₆					✓	

33

\$ppqq



aa=\$A6

ppqq=\$1641

dd=\$E0

BIT \$1641

但狀態旗號已更改如下

\$E0= 11100000

\$E0= 11100000

10100110

執行結果不為零，所以 Z=0；記憶內容（\$E0）之第 6、第 7 位元均為 1，所以 V=1，N=1。

BMI Branch on Minus (負值則分支跳越)

運算功能： 若 $N=1$ ，則跳到指定位址執行。

指令碼格式：

00110000	相對位址
----------	------

說明：檢查負值旗號 N ，若 $N=1$ ，則跳至指定位址執行，此位址由程式計數器內容加上相對位址而得；若 $N=0$ ，則繼續執行下一個指令，不跳越。

定址模式： 只能使用相對定址法。

OP Code = 30

BYTES = 2

CYCLES = 2 + 1 (若 $N=1$ 而分支跳越時)
+ 2 (若跳越至另一頁區時)

狀態旗號： 不受 BMI 指令的影響。

BNE Branch on Not Equal to zero (不等於 0 則分支跳越)

運算功能： 若 $Z=0$ ，則跳越至指定位址執行。

指令碼格式：

11010000	相對位址
----------	------

說明：BNE 指令恰與 BEQ 指令作用相反。檢查零值旗號 Z ，若 $Z=0$ 則跳越至指定位址執行； $Z=1$ 則不跳越，繼續執行下一指令。(請參考 BEQ 指令)

定址模式： 只能使用相對定址法。

OP Code = D0

BYTES = 2

CYCLES = 2 + 1 (若 $Z=0$ 而分支跳越時)
+ 2 (若跳越至另一頁區時)

狀態旗號： 不受 BNE 指令的影響。

BPL Branch on Plus (正值時，分支跳越)

運算功能： 當 $N=0$ 時，跳越至指定位址執行。

指令碼格式：

00010000	相對位址
----------	------

說明：BPL 指令恰與 BMI 指令作用相反。檢查負值旗號 N ，若 $N=0$ ，則跳越至指定位址執行；若 $N=1$ 則繼續執行下一個指令，不跳越。(請參考 BMI 指令)

定址模式： 只能使用相對定址法。

OP Code = 10

BYTES = 2

CYCLES = 2 + 1 (若 $N=0$ 而分支跳越時)
+ 2 (若跳越至另一頁區時)

狀態旗號： 不受 BPL 指令的影響。

BRK Break (中斷)

運算功能： $STACK \leftarrow [PC] + 2$ ； $B = 1$ ， $STACK \leftarrow [P]$ ； $PCL \leftarrow [FFFF]$ ， $PCH \leftarrow [FFFF]$

指令碼格式：

00000000

說明：BRK 指令可以使程式中斷，其作用類似 "插斷" (Interupt)。執行 BRK 指令之過程為：(請參考後面的執行過程圖)

- (1) 程式計數器 PC 之內容加 2 之後，存入堆疊區 (PCH 存入堆疊位址 $\$01ss$ ，PCL 存入 $\$01ss-1$ ；ss 為執行 BRK 指令之前的堆疊暫存器內容)。
- (2) 狀態暫存器 P 的中斷旗號 B 被設定為 1。
- (3) 將狀態暫存器之內容存於堆疊位址 $\$01ss-2$ 。
- (4) 堆疊暫存器在每次資料存入堆疊區之後，自動將其內容減 1。目前堆疊暫存器之內容已變成 $ss-3$ 。
- (5) 狀態暫存器之插斷控制旗號 I 被設定為 1，亦即表示 "禁止插斷"。由此可知，BRK 指令可禁止 (disable) 6502 的插斷服務；此時，從週邊設備 (peripheral device) 送至 6502 的插斷要求不起任何作用。
- (6) 把插斷指向器 (interrupt pointer，即記憶位址 $\$FFFE$ 、 $\$FFFF$) 之內容存到程式計數器中，以執行插斷檢查 (註 2)。

注意： 執行完 BRK 指令之後，螢幕上會顯示最後的程式計數器及各暫存器之內容。程式計數器的內容即為原來存入堆疊區的內容 $[PC] + 2$ ，這個值不一定是下一個指令的位址。

定址模式： 只能使用隱含定址法。

OP Code=00

BYTES=1

CYCLES=7

狀態旗號： 分成三個階段：

在上述第（5）步驟之前

N	V	B	D	I	Z	C
		1		0		

在第（5）步驟之後

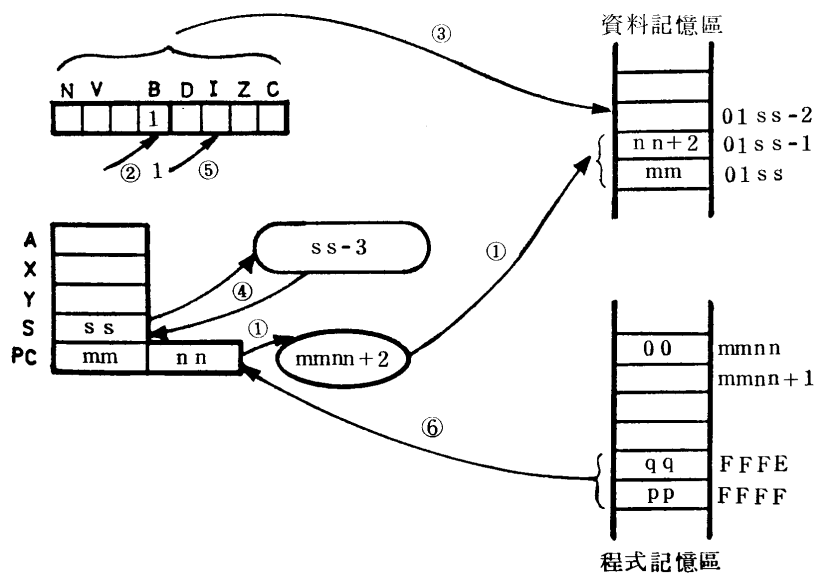
		1		1		
--	--	---	--	---	--	--

執行完 BRK 指令之後

		1		0		
--	--	---	--	---	--	--

執行過程圖：

執行過程圖：



註 1：BRK 指令使程式產生中斷點（break point），一般在程式除錯時（debug），均利用它來做程式的分段檢查。

註 2：BRK 指令產生的中斷與一般的插斷要求（IRQ，Interrupt Request）不同，其判別的方法是取出堆疊區的状态暫存器內容，檢查中斷旗號 B；若 B=1，表示 BRK 指令產生的中斷；若 B=0 則表示插斷要求。

BVC Branch on Overflow Clear (無溢位時，分支跳越)

運算功能： 當 $V=0$ 時，跳至指定位址執行。

指令碼格式：

01010000	相對位址
----------	------

說明：檢查溢位旗號 V ，若 $V=0$ ，則跳至指定位址執行，此位址由程式計數器內容加上相對位址而得；若 $V=1$ ，則繼續執行下一指令，不跳越。

定址模式： 只能使用相對定址法。

OP Code = 50

BYTES = 2

CYCLES = 2 + 1 (若 $V=0$ 而分支跳越時)
+ 2 (若跳越至另一頁區時)

狀態旗號： 不受 BVC 指令的影響。

BVS Branch on Overflow Set (溢位時，分支跳越)

運算功能： 若 $V=1$ ，則跳至指定位址執行。

指令碼格式：

01110000	相對位址
----------	------

說明：BVS 指令恰與 BVC 指令作用相反。若 $V=0$ ，則繼續執行下一個指令，不跳越；若 $V=1$ ，則跳至指定位址執行。(參考 BVC 指令)

定址模式： 只能使用相對定址法。

OP Code = 70

BYTES = 2

CYCLES = 2 + 1 (若 $V=1$ 而分支跳越時)
+ 2 (若跳越至另一頁區時)

狀態旗號： 不受 BVS 指令的影響。

CLC Clear Carry (清除進位旗號)

運算功能： $C \leftarrow 0$

指令碼格式：

00011000

說明：CLC 指令將進位旗號 C 清除為 0。CLC 指令經常用於 ADC 指令之前，使 ADC 指令執行加法時不受進位旗號的影響。(請參考 ADC 指令)

定址模式： 隱含定址法。

OP Code = 18

BYTES = 1

CYCLES = 2

狀態旗號：

N	V		B	D	I	Z	C
							0

CLD Clear Decimal flag (清除十進位旗號)

運算功能： $D \leftarrow 0$

指令碼格式：

11011000

說明：將十進位旗號 D 清除為 0 使 ADC 與 SBC 指令可以執行二進位的運算。

定址模式： 隱含定址法。

OP Code = D8

BYTES = 1

CYCLES = 2

狀態旗號：

N	V		B	D	I	Z	C
				0			

CLI Clear Interrupt mask (清除插斷遮罩)

運算功能： $I \leftarrow 0$ (允許插斷)

指令碼格式：

01011000

說明：將插斷遮罩位元 (mask bit) 清除為 0，此指令允許 6502 處理插斷服務。

定址模式： 隱含定址法。

OP Code = 58

BYTES = 1

CYCLES = 2

狀態旗號：

N	V	B	D	I	Z	C
				0		

CLV Clear Overflow flag (清除溢位旗號)

運算功能： $V \leftarrow 0$

指令碼格式：

10111000

說明：將溢位旗號 V 清除為 0。

定址模式： 隱含定址法。

OP Code = B8

BYTES = 1

CYCLES = 2

狀態旗號：

N	V	B	D	I	Z	C
	0					

CMP Compare to accumulator (與累積器比較)

運算功能： $N, Z, C \leftarrow [A] - [M]$

✓ 0 1	IF $[A] > [M]$
0 1 1	IF $[A] = [M]$
✓ 0 0	IF $[A] < [M]$

指令碼格式：

110bbb01	addr/data	addr
----------	-----------	------

說明：將累積器的內容減去記憶位址的內容或資料，但是執行之後，A 與 M 之內容均不改變，所得結果只是用來設定狀態旗號 N、Z、C 而已。

第一章曾經提過，在減法運算中，可以把進位旗號 C 的補數 \bar{C} 當做 "借位旗號" 使用。因此；在 $[A] < [M]$ 時，有借位產生 ($C=1$ ，亦即 $\bar{C}=0$)； $[A] \geq [M]$ 時， $C=0$ 。零值旗號 Z 只有在 $[A] = [M]$ 時才被置定為 1 (表示運算結果為 0)，而此時負值旗號 N 亦必為 0，其他兩種情形下的 N 值視運算結果而定。

一般在使用 CMP 指令時，後面都加上分支跳越的指令，用來檢查 N、Z 或 C 等旗號。

定址模式：

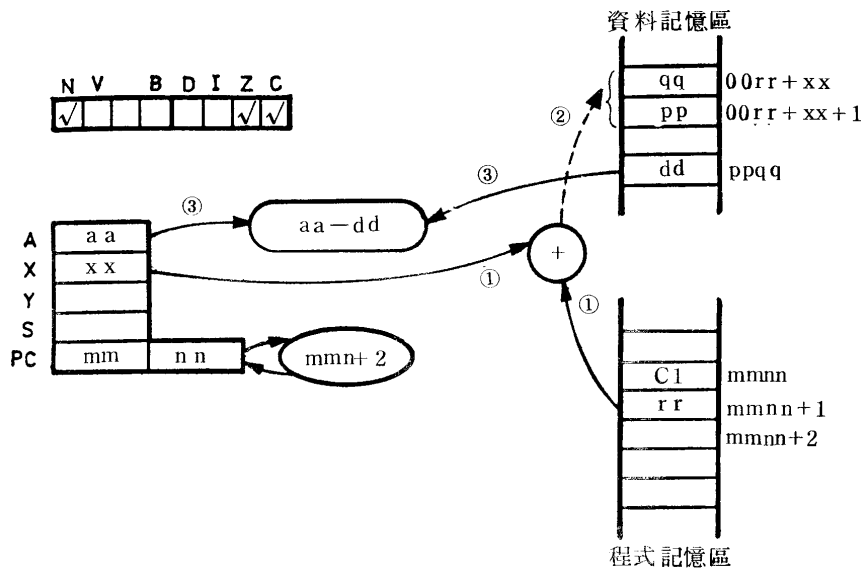
	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 (<i>IND,X</i>)	後索引 間接 (<i>IND</i>),Y	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE	C9	CD	C5		DD	D9	D5		C1	D1			
BYTES	2	3	2		3	3	2		2	2			
CYCLES	2	4	3		4*	4*	4		6	5*			
bbb	010	011	001		111	110	101		000	100			

狀態旗號：	N	V	B	D	I	Z	C
	✓					✓	✓

圖解 6502 指令集

41

實例： CMP 指令之先索引間接定址法： CMP (\$rr, X)



假設 rr=\$23, xx=\$20, aa=\$F6

qq = [\$0043] = \$6D

pp = [\$0044] = \$15

dd = [\$156D] = \$18

執行 CMP (\$23, X)

結果累積器之內容仍為\$F6，位址 \$156D 之內容仍為 \$18，但旗號 N、Z 及 C 分別被設定：

\$F6 = 11110110

\$18 之 "2 的補數" = 11101000

11101110

進位，C=1

設定 N=1

結果不為 0，所以 Z=0

CPX Compare to X register (與暫存器 X 比較)

運算功能： $N, Z, C \leftarrow [X] - [M]$

✓ 0 1	IF $[X] > [M]$
0 1 1	IF $[X] = [M]$
✓ 0 0	IF $[X] < [M]$

指令碼格式：

1110bb00	addr/data	addr
----------	-----------	------

說明：此指令是將記憶位址的內、或資料與 X 暫存器之內容相比較。除此之外，其功能與 CMP 指令均相同，請參考 CMP 指令之說明。

定址模式：與 CMP 指令不同，CPX 指令只有三種定址法：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 (<i>IND,X</i>)	後索引 間接 (<i>IND,Y</i>)	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE	E0	EC	E4										
BYTES	2	3	2										
CYCLES	2	4	3										
bbb	00	11	01										

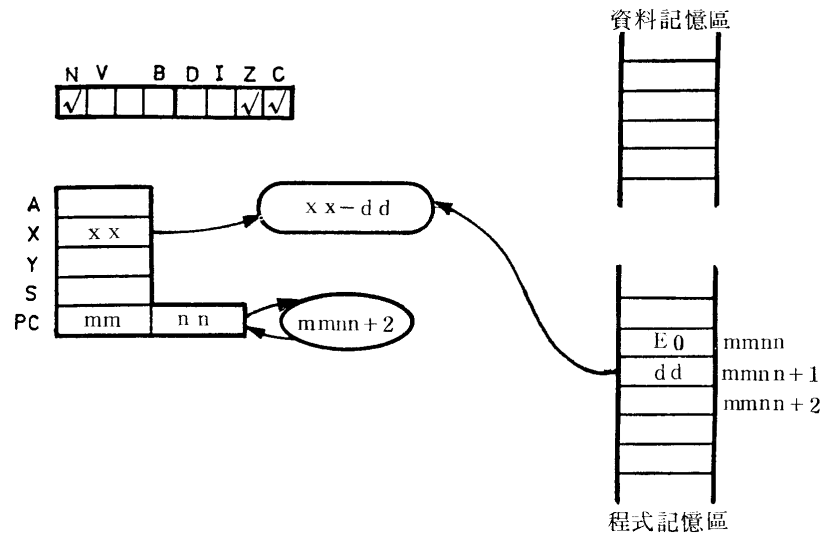
狀態旗號：

N	V		B	D	I	Z	C
✓						✓	✓

圖解 6502 指令集

43

實例： CPX 之立即定址法 CPX #\$dd



CPY Compare to Y register (與暫存器 Y 比較)

運算功能： $N, Z, C \leftarrow [Y] - [M]$

✓ 0 1	IF $[Y] > [M]$
0 1 1	IF $[Y] = [M]$
✓ 0 0	IF $[Y] < [M]$

指令碼格式：

1100bb00	addr/data	addr
----------	-----------	------

說明：將記憶位址之內容或資料與 Y 暫存器相比較，其結果用來設定旗號 N、Z、C。(請參考 CMP 或 CPX 指令)

定址模式： 只使用三種定址法。

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 (<i>IND, X</i>)	後索引 間接 (<i>IND</i>), Y	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE	C0	CC	C4										
BYTES	2	3	2										
CYCLES	2	4	3										
bbb	00	11	01										

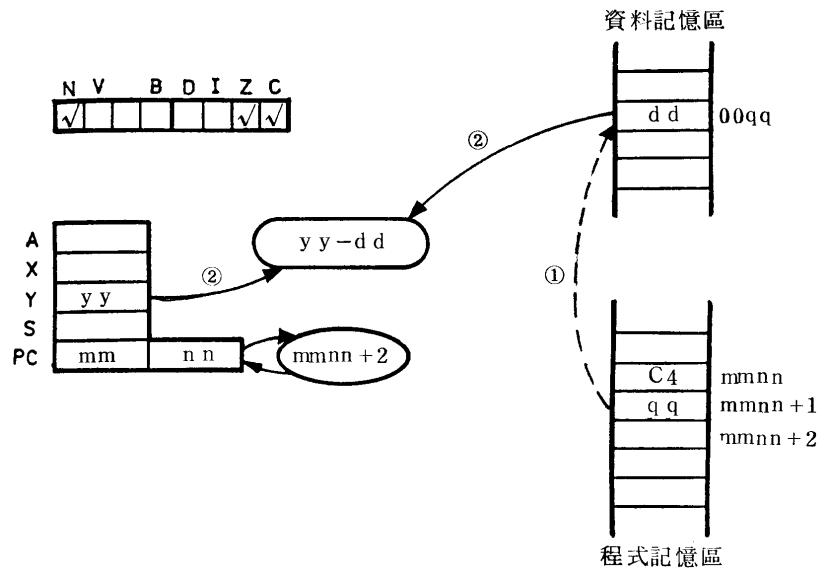
狀態旗號：

N	V		B	D	I	Z	C
✓						✓	✓

圖解 6502 指令集

45

實例： CPY 指令之零頁定址法： CPY \$qq



DEC Decrement memory (記憶內容減 1)

運算功能： $M \leftarrow [M] - 1$

指令碼格式：

110bb110	addr/data	addr
----------	-----------	------

說明：將記憶位址之內容減 1，其結果再存回該記憶位址。

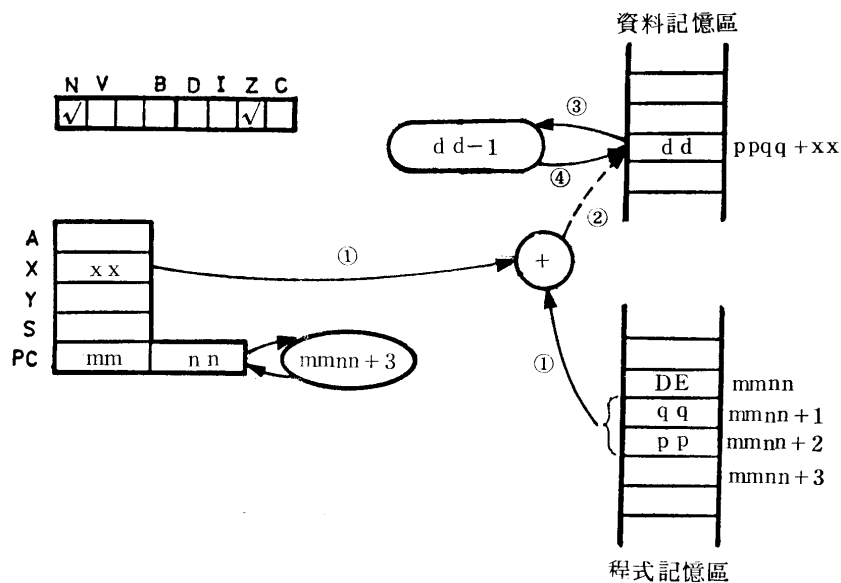
注意：電腦執行 DEC 指令時，實際上是將記憶位址的內容加上 \$FF (1 之 "2 的補數")。因此，除了記憶內容為 0 以外，其他的情形下均有進位產生。但是，電腦並不將進位存入進位旗號，所以進位旗號 C 不受此指令的影響。

定址模式：

	立即 IMMEDIATE	絕對定址法 ABSOLUTE	零頁定址法 0-PAGE	INDIRECT	絕對索引 X ABS-X	絕對索引 Y ABS-Y	零頁索引 X 0-PAGE X	零頁索引 Y 0-PAGE Y	先索引 間接 (IND,X)	後索引 間接 (IND),Y	相對 RELATIVE	隱含 IMPLIED	累加器 ACCUMULATOR
OP CODE		CE	C6		DE		D6						
BYTES		3	2		3		3						
CYCLES		6	5		7		6						
bbb		01	00		11		10						

狀態旗號：	N	V		B	D	I	Z	C
	✓						✓	

實例： DEC 之絕對索引定址法 DEC \$ppqq, X



DEX Decrement X (X 暫存器內容減 1)

運算功能： $X \leftarrow [X] - 1$

指令碼格式：

11001010

說明：將 X 暫存器之內容減 1。一般利用此指令來將 X 暫存器當做一個計數器 (Counter) 使用。進位旗號 C 不受此指令的影響 (請參考 DEC 指令)。

定址模式： 隱含定址法。

OP Code = CA

BYTES = 1

CYCLES = 2

狀態旗號：	N	V		B	D	I	Z	C
	✓						✓	

DEY Decrement Y (Y 暫存器內容減 1)

運算功能： $Y \leftarrow [Y] - 1$

指令碼格式：

10001000

說明：將 Y 暫存器之內容減 1。利用此指令可將 Y 暫作器當做計數器使用。(謂參考 DEX 指令)

定址模式： 隱含定址法。

OP Code = 88

BYTES = 1

CYCLES = 2

狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

EOR Exclusive OR with accumulator（與累積器之內容做 "互斥或" 邏輯運算）運算功能： $A \leftarrow [A] \vee [M]$

指令碼格式：

010bbb01	addr/data	addr
----------	-----------	------

說明：將記憶位址的內容或資料與累積器內容做 "互斥或" 邏輯運算。"互斥或" 邏輯運算之真值表如下：

A \ M	0	1
0	0	1
1	1	0

定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引間接 (<i>IND,X</i>)	後索引間接 (<i>IND,Y</i>)	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE	49	4D	45		5D	59	55		41	51			
BYTES	2	3	2		3	3	2		2	2			
CYCLES	2	4	3		4*	4*	4		6	5*			
bbb	010	011	001		111	110	101		000	100			

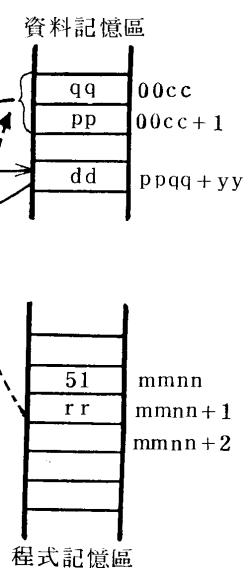
狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

49

實例： EOR 之後索引間接定址法

EOR (\$ rr) , Y



假設 $rr=\$40$, $yy=\$10$, $aa=\$E3$

$$qq = (\$0040) = \$1E$$
$$pp = (\$0041) = \$25$$

dd= (\$251E) = \$A0

執行 EOR (\$40), Y

結果累積器內容變為\$43，運算過程如下：

\$E3= \quad 11100011

\$A0= 10100000

01000011 (= \$43)

$$N=0$$

結果不爲 0，所以 $Z=0$

注意：如果執行 EOR # $\$FF$ ，會將累積器之內容變為 "1 的補數"（每一個為 "1" 的位元變為 "0"，而 "0" 的位元變為 "1"）

INC Increment memory (記憶內容加 1)

運算功能： $M \leftarrow [M] + 1$

指令碼格式：

111bb110	addr/data	addr
----------	-----------	------

說明：將記憶內容加 1，其結果再存入該記憶位址。此指令恰與 DEC 之作用相反。

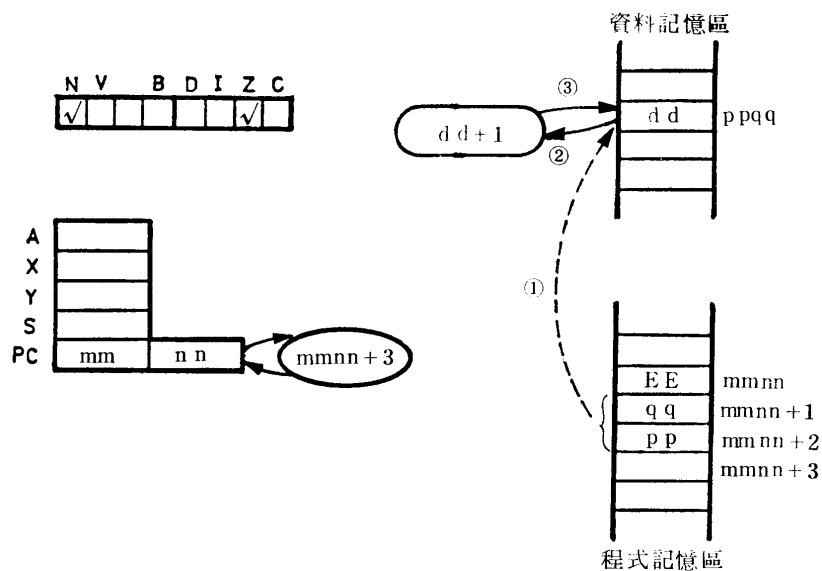
定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 <i>(IND, X)</i>	後索引 間接 <i>(IND), Y</i>	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE		EE	E6		FE		F6						
BYTES		3	2		3		2						
CYCLES		6	5		7		6						
bbb		01	00		11		10						

狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

實例：INC 之絕對定址法：INC \$ppqq



INX Increment X (X 暫存器之內容加 1)

運算功能： $X \leftarrow [X] + 1$

指令碼格式：

11101000

說明：將 X 暫存器之內容加 1。可利用此指令將 X 暫存器當做計數器使用。INX 與 DEX 之作用恰好相反。

定址模式： 隱含定址法。

OP Code = E8

BYTES = 1

CYCLES = 2

狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

INY Increment Y (Y 暫存器之內容加 1)

運算功能： $Y \leftarrow [Y] + 1$

指令碼格式：

11001000

說明：將 Y 暫存器之內容加 1。利用此指令可將 Y 暫存器當做計數器使用。INY 與 DEY 之作用恰好相反。

定址模式： 隱含定址法。

OP Code = C8

BYTES = 1

CYCLES = 2

狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

JMP **Jump to address**（跳越至指定位址）

運算功能： $PC \leftarrow ADDRESS$

指令碼格式：

01b01100	addr	addr
----------	------	------

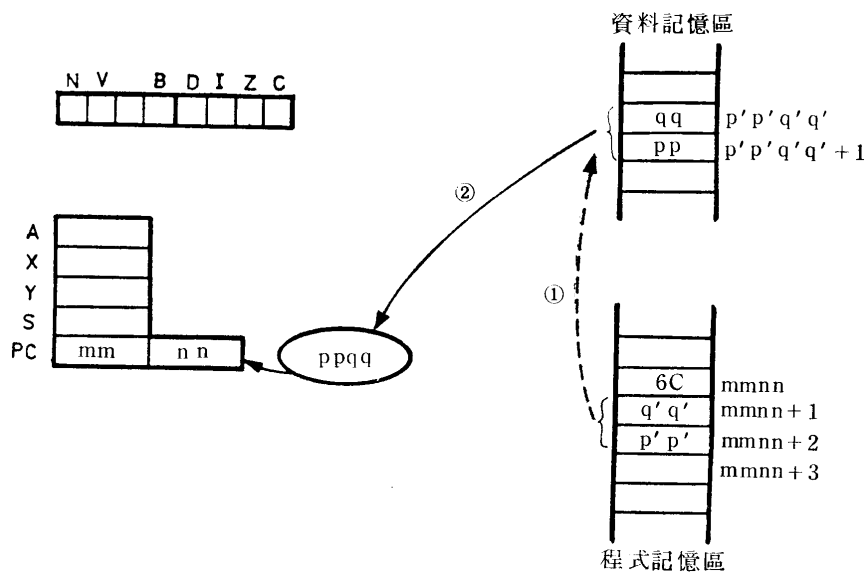
說明：將一個新的記憶位址存入程式計數器，而使程式跳到此新位址繼續執行。此新位址可由絕對定址法或間接定址法來決定。

定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 <i>(IND,X)</i>	後索引 間接 <i>(IND),Y</i>	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE		4C		6C									
BYTES		3		3									
CYCLES		3		5									
bbb		0		1									

狀態旗號： 不受 **JMP** 指令之影響。

實例： **JMP** 之間接定址法： **JMP** (\$ p'p'q'q')



JSR Jump to Subroutine (跳至副程式執行)

運算功能： $STACK \leftarrow [PC] + 2$; $PC \leftarrow ADDRESS$

指令碼格式：

00100000	addr	addr
----------	------	------

說明：當程式執行時，如果需要呼用某一段副程式，可以利用 JSR 指令；JSR 指令時，電腦動作的過程如下：（請參考後面的執行過程圖）

- (1) 將 PC 暫存器的內容加 2 之後存入堆疊區（高位元組存於堆疊位址 $01ss$ ，低位元組存於 $01ss-1$ ； ss 為執行之前的堆疊暫存器內容）。
- (2) 每一次將資料存入堆疊區之後，堆疊暫存器之內容自動減 1。目前堆疊暫存器之內容應為 $ss-2$ 。
- (3) 將副程式之起始位址存入程式計數器中。

定址模式：絕對定址法

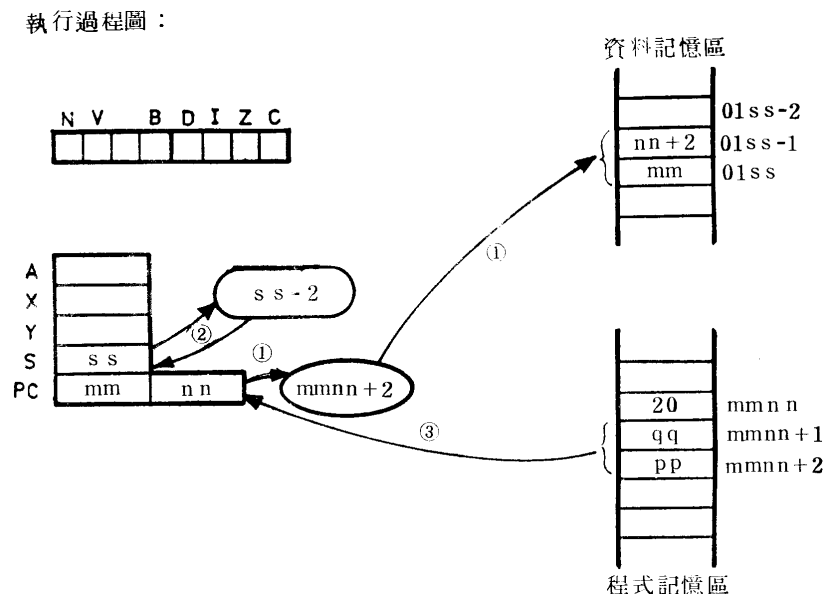
OP Code = 20

BYTES = 3

CYCLES = 6

狀態旗號：不受 JSR 指令之影響。

執行過程圖：



注意： JSR 指令佔有 3 個位元組，所以下一個指令的位址應該是 $mmnn+3$ ；為什麼我們存入堆疊區的值是 $mmnn+2$ ？因為在副程式中必有一個返回指令 RTS，電腦在執行 RTS 指令時會先取出堆疊區的內容，加 1 之後才放至程式計數器。所以，我們將 $mmnn+2$ 存入堆疊區，從副理式返回時，自然會在程式計數器存入 $mmnn+3$ 而指向下一個指令的位址了。

LDA Load Accumulator (載入累積器)

運算功能： $A \leftarrow [M]$

指令碼格式：

101bbb01	addr/data	addr
----------	-----------	------

說明：將記憶內容或資料載入累積器。執行之後，記憶位址的內容不變，狀態旗號 N、Z 由載入 A 之內容來設定。

定址模式：

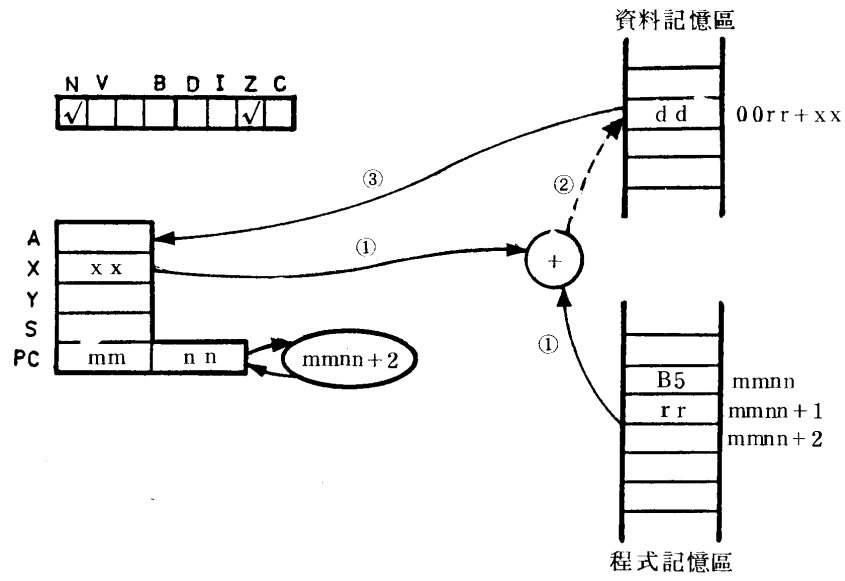
	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引間接 (<i>IND,X</i>)	後索引間接 (<i>IND</i>),Y	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE	A9	AD	A5		BD	B9	B5		A1	B1			
BYTES	2	3	2		3	3	2		2	2			
CYCLES	2	4	3		4*	4*	4		6	5*			
bbb	010	011	001		111	110	101		000	100			

狀態旗號：	N	V	B	D	I	Z	C
	✓					✓	

圖解 6502 指令集

55

實例： LDA 之 "X 暫存器零頁索引定址法"： LDA \$rr, X



LDX Load register X (載入 X 暫存器)

運算功能： $X \leftarrow [M]$

指令碼格式：

101bbb10	addr/data	addr
----------	-----------	------

說明：將記憶內容或資料載入 X 暫存器。

注意：LDX 指令不能使用任何包含 X 暫存器之定址法。

定址模式：

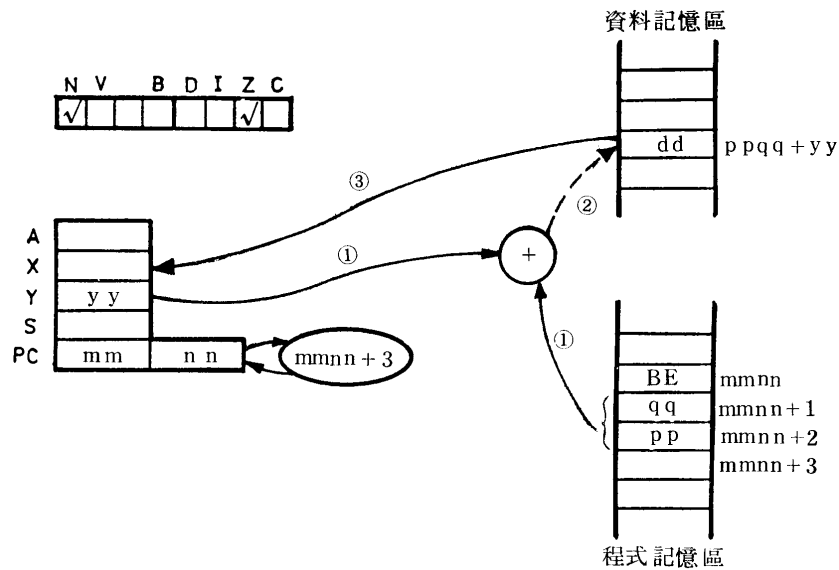
	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引間接 <i>(IND,X)</i>	後索引間接 <i>(IND),Y</i>	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE	A2	AE	A6			BE		B6					
BYTES	2	3	2			3		2					
CYCLES	2	4	3			4*		4					
bbb	000	011	001			111		110					

狀態旗號：	N	V	B	D	I	Z	C
	✓					✓	

圖解 6502 指令集

57

實例： LDX 之 "Y 暫存器絕對定址法"： LDX \$ppqq, Y



LDY Load register Y (載入 Y 暫存器)

運算功能： $Y \leftarrow [M]$

指令碼格式：

101bbb00	addr/data	addr
----------	-----------	------

說明：將記憶內容或資料載入 Y 暫存器。

注意：LDY 指令不能使用包含 Y 暫存器之任何定址法。

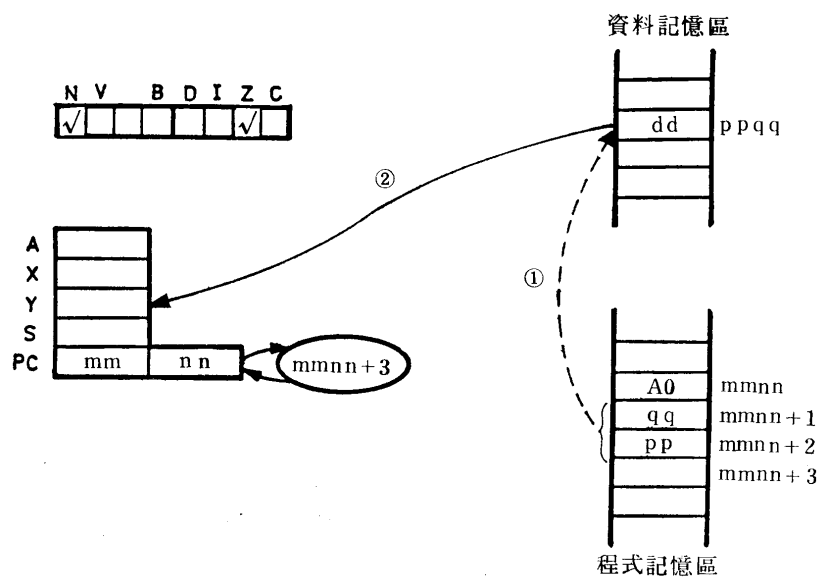
定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 (<i>IND,X</i>)	後索引 間接 (<i>IND</i>),Y	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE	A0	AC	A4		BC		B4						
BYTES	2	3	2		3		4						
CYCLES	2	4	3		4*		4						
bbb	000	011	001		111		101						

狀態旗號：

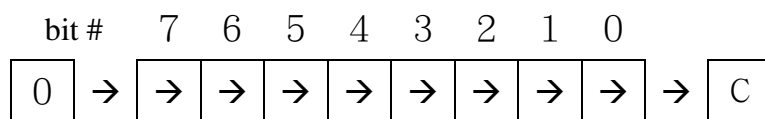
N	V		B	D	I	Z	C
✓						✓	

實例： LDY 之絕對定址法： LDY \$ppqq

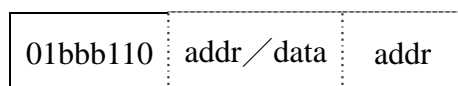


LSR Logical Shift Right (邏輯右移)

運算功能：



指令碼格式：



說明：將累積器或記憶位址的內容向右移一個位元；而把 "0" 移入最高位元 (Bit 7)，把最低位元 (Bit 0) 移至進位旗號，移完之後的值仍存於原記憶位址中。

狀態旗號 N 必然被設定為 0，因為最高位元已移入 "0"。

定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 <i>(IND,X)</i>	後索引 間接 <i>(IND),Y</i>	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE		4E	46		5E		56						4A
BYTES		3	2		3		2						1
CYCLES													2
bbb		011	001		111		101						010

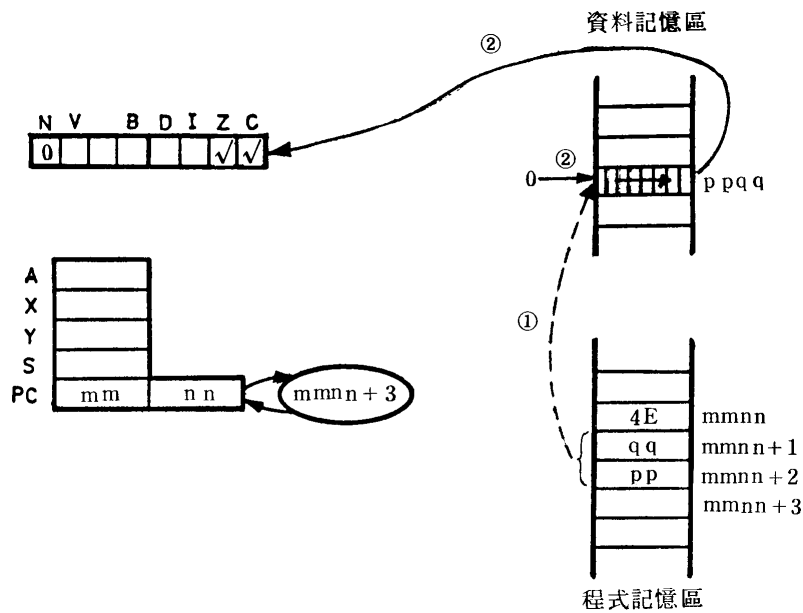
狀態旗號：

N	V		B	D	I	Z	C
0						✓	✓

圖解 6502 指令集

60

實例： LSR 之絕對定址法： LSR \$ppqq



NOP No Operation (無作用)

運算功能： 沒有

指令碼格式： 11101010

說明：NOP 指令不起任何作用，但是佔了兩個執行週期，一般利用它來延遲時間。在程式偵錯時，如果程式有更改，亦可利用它來取代不再使用的指令以維持各個指令的位址關係。

定址模式： 隱含定址法
 OP Code = EA
 BYTES = 1
 CYCLES = 2

狀態旗號： 不受 NOP 指令之影響。

圖解 6502 指令集

61

ORA inclusive OR with Accumulator (與累積器內容執行邏輯 "或" 運算)運算功能： $A \leftarrow [A] \vee [M]$

指令碼格式：

000bbb01	addr/data
----------	-----------

說明：將記憶內容或資料與累積器內容執行邏輯 "或" 運算，運算結果存入累積器 A。邏輯 "或" 運算的真值表如下：

A \ M	0	1
	0	1
0	0	1
1	1	1

ORA 指令經常用來開啓位元 C (Turn Bit "On")，亦即設定某一位元爲 1。例如 ORA #\$80，會無條件的將累積器的最高位元設定爲 "1"。

定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 <i>(IND,X)</i>	後索引 間接 <i>(IND),Y</i>	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE	09	0D	05		1D	19	15		01	11			
BYTES	2	3	2		3	3	2		2	2			
CYCLES	2	4	3		4*	4*	4		6	5*			
bbb	010	011	001		111	110	101		000	100			

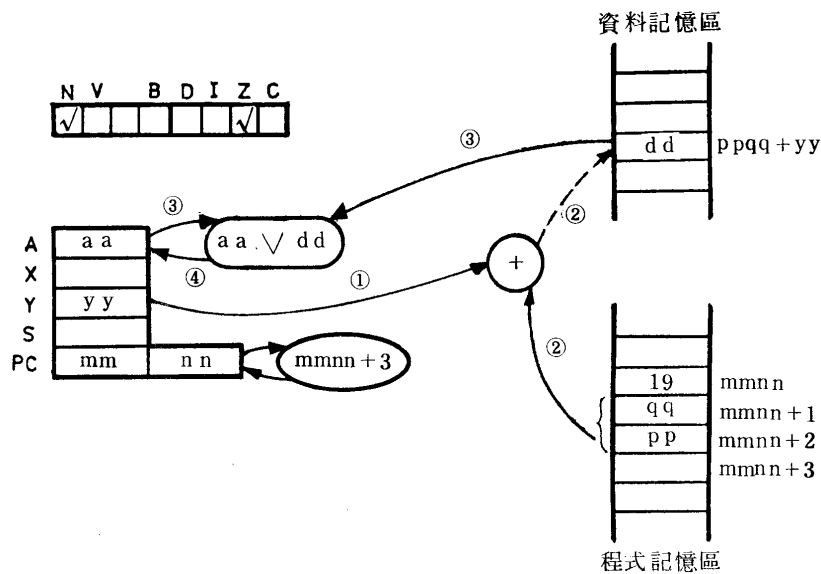
狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

圖解 6502 指令集

62

實例：ORA 之 "Y 暫存器絕對索引定址法"：ORA \$ppqq, Y



PHA Push A (將累積器內容 "推入" 堆疊區)

運算功能：STACK ← [A]; S ← [S] - 1

指令碼格式：01001000

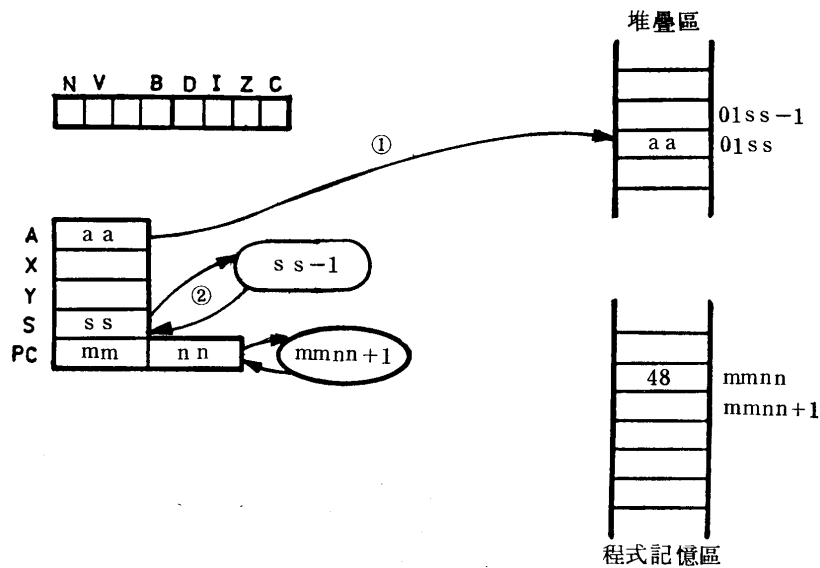
說明：將累積器 A 之內容 "推入" 堆疊區。A 之內容不改變，堆疊暫存器 S 之內容減 1。

堆疊暫存器 S 存放下一個堆疊位址的低位元組（其高位元組固定為 \$01），每一次將資料 "推入" 堆疊區之後，S 之內容自動減 1。因為堆疊區的資料是以 "堆積" 的方式儲存，先存入的資料在下（堆疊位址較大），後存入的資料在上（堆疊位址較小）。所以，要提取資料時，只能從堆疊區的最上面資料開始提取，其堆疊位址就是 "S 的內容加 1"（請參考 PLA 指令）。

定址模式：隱含定址法
OP Code = 48
BYTES = 1
CYCLES = 3

狀態旗號：不受 PHA 指令的影響。

執行過程圖：



PHP Push Processor status (將狀態暫存器 P 之內容 "推入" 堆疊區)

運算功能： $STACK \leftarrow [P]; S \leftarrow [S] - 1$

指令碼格式：

00001000

說明：將狀態暫存器 P 之內容存入堆疊區，然後將堆疊暫存器 S 之內容減 1。(請參考 PHA 指令之說明)

注意：PHP 指令通常用在呼叫副程式之前，將狀態暫存器的內容先儲存起來。但是，在插斷服務或執行 BRK 指令之前，並不需要執行 PHP 指令。因為，6502 在插斷服務或中斷 (BRK) 動作產生之前，會自動將狀態暫存器的內容推入堆疊區頂端。(請參考 BRK 指令)

定址模式：隱含定址法。

OP Code = 08

BYTES = 1

CYCLES = 3

狀態旗號：不受 PHP 指令的影響。

PLA Pull accumulator (從堆疊區提出資料存至累積器)

運算功能： $S \leftarrow [S] + 1; A \leftarrow [STACK]$

指令碼格式：

01101000

說明：先將堆疊暫存器 S 之內容加 1，使堆疊指示器指向堆疊區頂端的位址，然後將此堆疊位址內的資料提出，存至累積器 A 。

通常在呼叫副程式或執行插斷服務 (Servicing an interrupt) 之前，都用 **PHA** 指令先把累積器內容存至堆疊區。在完成副程式之後，再以 **PLA** 將原先儲存的累積器內容提出，存回累積器。

定址模式：隱含定址法。

OP Code = 68

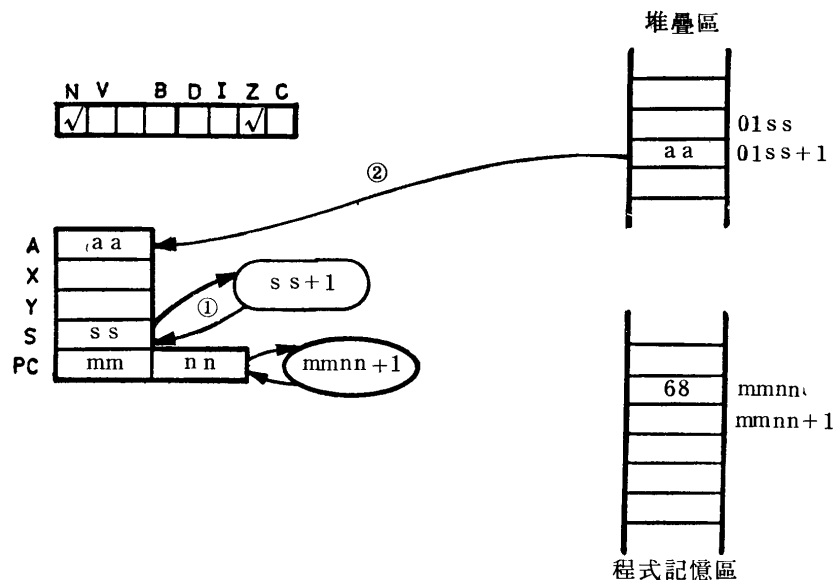
BYTES = 1

CYCLES = 4

狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

執行過程圖：



PLP Pull Processor status from stack（從堆疊區取出資料存至狀態暫存器）

運算功能： $S \leftarrow [S] + 1; P \leftarrow [STACK]$

指令碼格式：

00101000

說明：先將堆疊暫存器 **S** 之內容加 1，使堆疊指示器指向堆疊區頂端的位址，然後將此堆疊位址內的資料提出，存至狀態暫存器 **P**。

注意：**PLP** 與 **PHP** 指令是成對使用的；亦即，在呼叫副程式之前若曾執行 **PHP** 指令，則完成副程式之後就以 **PLP** 指令來恢復原先的狀態旗號。

定址模式：隱含定址法。

OP Code = 28

BYTES = 1

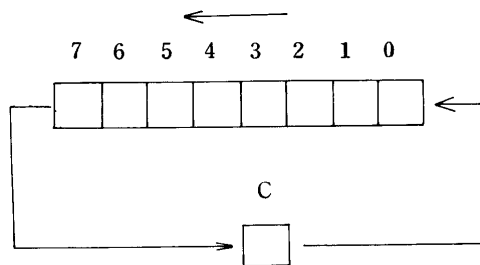
CYCLES = 4

狀態旗號：

N	V		B	D	I	Z	C
✓	✓	✓	✓	✓	✓	✓	✓

ROL Rotate Left one bit（向左旋轉一個位元）

運算功能：



指令碼格式：

001bbb10	addr/data	addr
----------	-----------	------

說明：將累積器或記憶內容向左旋轉一個位元，其最高位元（Bit 7）移入進位旗號，而原進位旗號移至最低位元（Bit 0）。

圖解 6502 指令集

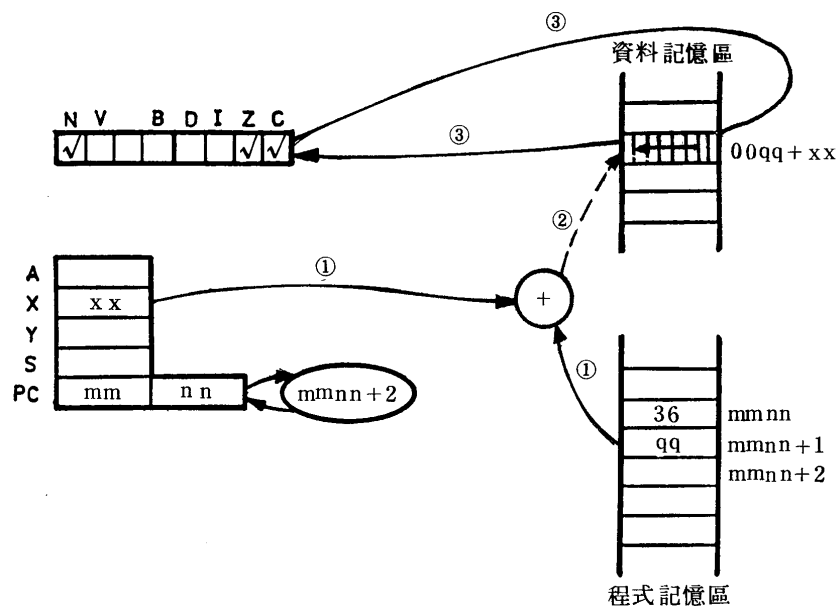
66

定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引間接 (<i>IND,X</i>)	後索引間接 (<i>IND</i>),Y	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE		2E	26		3E		36						2A
BYTES		3	2		3		2						1
CYCLES		6	5		7		6						2
bbb		011	001		111		101						010

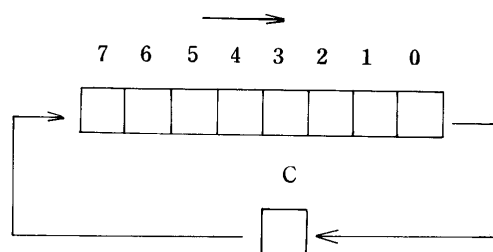
狀態旗號：	N	V		B	D	I	Z	C
	✓						✓	✓

實例：ROL 之 "X 暫存器零頁索引定址法"：ROL \$qq, X



ROR Rotate Right one bit（向右旋轉一個位元）

運算功能：



指令碼格式：

001bbb10	addr/data	addr
----------	-----------	------

說明：將累積器或記憶內容向右旋轉一個位元，其最低位元（Bit 0）移入進位旗號 C，原進位旗號 C 移至最高位元（Bit 7）。

定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 <i>(IND,X)</i>	後索引 間接 <i>(IND),Y</i>	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE		6E	66		7E		76						6A
BYTES		3	2		3		2						1
CYCLES		6	5		7		6						2
bbb		011	001		111		101						010

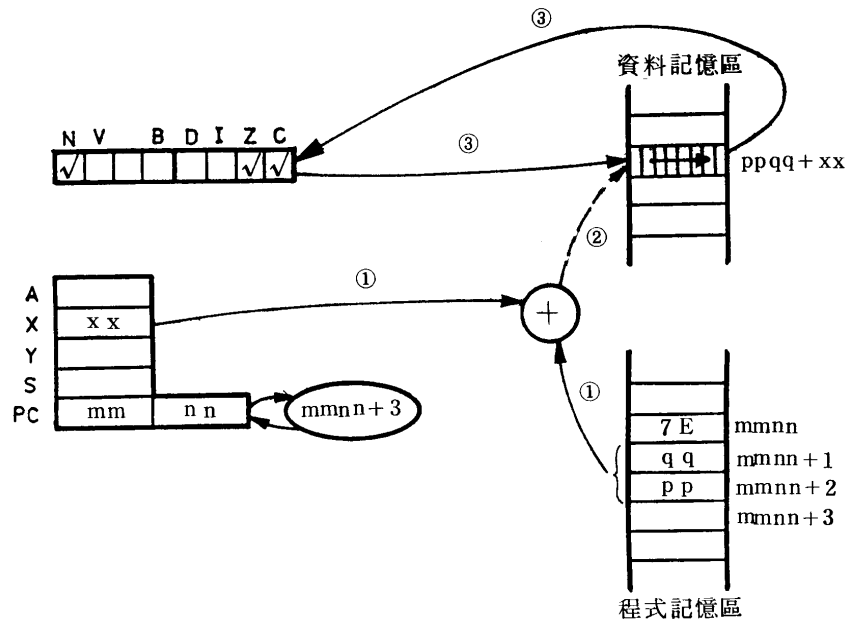
狀態旗號：

N	V		B	D	I	Z	C
✓						✓	✓

圖解 6502 指令集

68

實例： ROR 之 "X 暫存器絕對索引定址法"： ROR \$ppqq, X



RTI Return from Interrupt (從插斷返回)

運算功能： $S \leftarrow [S] + 1$; $P \leftarrow [STACK]$;
 $S \leftarrow [S] + 1$; $PCL \leftarrow [STACK]$;
 $S \leftarrow [S] + 1$; $PCH \leftarrow [STACK]$

指令碼格式：

01000000

說明：這個指令將堆疊區頂端的 3 個位元組提出，分別存入狀態暫存器及程式計數器內。假設執行 RTI 指令之前，堆疊暫存器之內為 *ss*，則

狀態暫存器 $P \leftarrow [00ss + 1]$

程式計數器之低位元組 $PCL \leftarrow [00ss + 2]$

程式計數器之高位元組 $PCH \leftarrow [00ss + 3]$

最後之堆疊暫存器內容為 $ss + 3$ 。

定址模式： 隱含定址法。

OP Code = 40

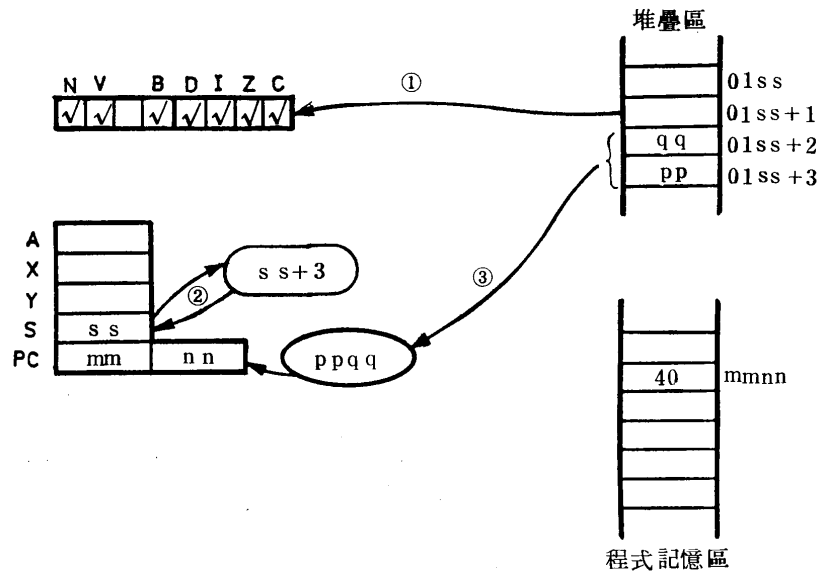
BYTES = 1

CYCLES = 6

狀態旗號：

N	V		B	D	I	Z	C
✓	✓		✓	✓	✓	✓	✓

執行過程圖：



RTS Return from Subroutine (從副程式返回)

運算功能：
 $S \leftarrow [S] + 1$; $PCL \leftarrow [STACK]$;
 $S \leftarrow [S] + 1$; $PCH \leftarrow [STACK]$;
 $PC \leftarrow [PC] + 1$

指令碼格式：

01100000

說明：RTS 指令僅寫於副程式中，當做副程式的結束指令，以便返回主程式繼續執行。

執行 RTS 指令時，電腦將堆疊區頂端的兩個位元組提出，加 1 之後再存入程式計數器中，此時程式計數器內容即為主程式中 JSR 之下一個指令的位址。所以，電腦會從 JSR 之下一個指令開始繼續執行主程式。(請對照 JSR 指令)

注意：(1) 每一個副程式中，至少須有一個 RTS 指令。

(2) 請記住，堆疊暫存器的內容為 "下一個可存資料的堆疊位址" 之低位元組。

定址模式：隱含定址法。

OP Code = 60

BYTES = 1

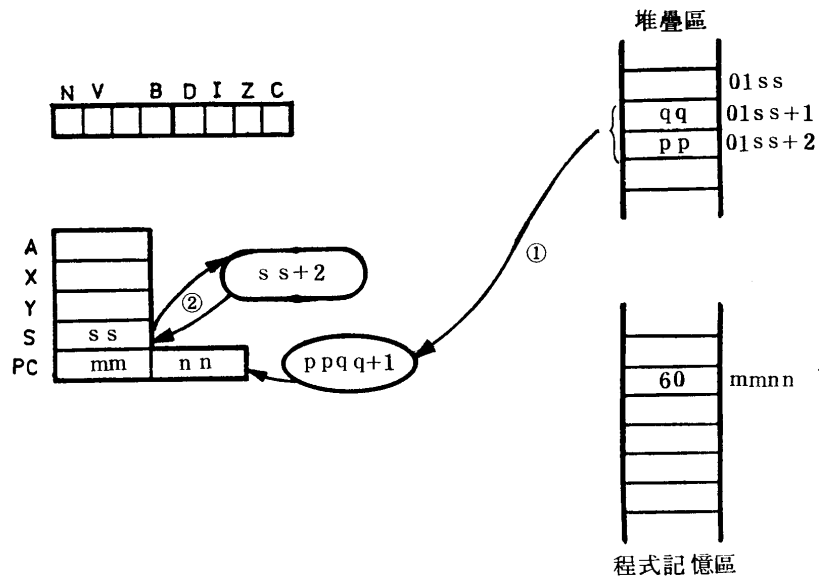
CYCLES = 6

狀態旗號：不受 RTS 指令的影響。

圖解 6502 指令集

70

執行過程圖：



SBC Subtract With Carry（將累積器內容減去記憶內容及進位旗號 \bar{C} 之補數）

運算功能： $A \leftarrow [A] - [M] - \bar{C}$

指令碼格式：

111bbb01	addr/data	addr
----------	-----------	------

說明：在減法運算中，進位旗號 C 的補數 (\bar{C}) 用來當做 "借位旗號"：若 $C=0$ ，表示有借位；若 $C=1$ ，表示無借位（請參考第一章）。所以，SBC 的作用就是將累積器內容減去記憶內容及 "借位旗號"，其結果存於累積器中。

注意：(1) SBC 指令可用於十進制或二進制模式，視十進制模式旗號 D 而定。

(2) 若不想考慮 "借位旗號" 時，可先利用 SEC 指令將進位旗號 C 設定為 1（亦即借位旗號 \bar{C} 為 0）。

定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引間接 (<i>IND,X</i>)	後索引間接 (<i>IND,Y</i>)	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE	E9	ED	E5		FD	F9	F5		E1	F1			
BYTES	2	3	2		3	3	2		2	2			
CYCLES	2	4	3		4*	4*	4		6	5*			
bbb	010	011	001		111	110	101		000	100			

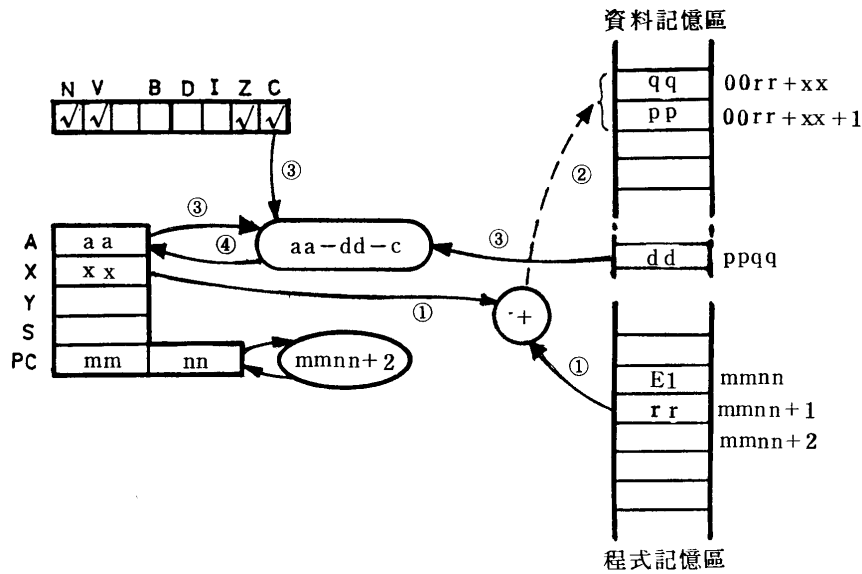
狀態旗號：

N	V	B	D	I	Z	C
✓	✓				✓	✓

圖解 6502 指令集

72

實例： SBC 之先索引間接定址法： SBC (\$rr, X)



假設執行 SBC 指令之前，狀態暫存器內容為

P=\$A0，也就是

N	V	B	D	I	Z	C
1	0	1	0	0	0	0

可知，

D=0 (執行二進制模式的運算)

C=0 (C=1，表示上次運算的結果有借位產生)

另外，假設：aa=\$14，xx=\$37，rr=\$15

qq = [\$0015 + 37] = \$E2

pp = [\$0015 + 38] = \$07

dd = [\$07E2] = \$34

則執行 SBC (\$15, X)

結果累積器之內容變為 \$DF，運算過程如下：

aa - dd - C = \$14 - \$34 - \$1 = \$14 - \$35

\$14 = 00010100

\$35 之 "2 的補數" = 11001011

11011111 (\$DF)

執行結果各狀態旗號如下：N=1，V=0，Z=0，C=0

注意：進位旗號 C=0，表示減法運算中產生借位 ($\bar{C}=1$)。

SEC Set Carry (將進位旗號 C 置定為 1)

運算功能： $C \leftarrow 1$

指令碼格式：

00111000

說明：將進位旗號 C 置定為 1，這個指令通常用於 SBC 指令之前，以便將減法運算中的 "借位旗號 \bar{C} " 清除為 0。(請參考 SBC 指令)

定址模式： 隱含定址法。

OP Code = 38

BYTES = 1

CYCLES = 2

狀態旗號：

N	V	B	D	I	Z	C
						1

SED Set Decimal mode (置定十進位模式旗號，D=1)

運算功能： $D \leftarrow 1$

指令碼格式：

11111000

說明：將十進位模式旗號 D 置定為 1，使 ADC 及 SBC 指令可以執行十進位運算。設計者應特別注意，D 值的設定與否，將影響運算結果，若設定錯誤會使 ADC 及 SBC 產生錯誤的運算結果。

定址模式： 隱含定址法

OP Code = F8

BYTES = 1

CYCLES = 2

狀態旗號：

N	V	B	D	I	Z	C
			1			

SEI Set Interrupt disable (置定插斷旗號為 1，禁止插斷)

運算功能： $I \leftarrow 1$

指令碼格式：

01111000

說明：將插斷遮罩旗號置定為 1， $I=1$ 表示禁止插斷。通常用於處理插斷時，禁止其他的插斷要求；或者在系統 "重置" (reset) 時之禁止插斷。

在 $I=1$ 時，6502 不接受週邊設備送來的插斷要求。

定址模式： 隱含定址法
 OP Code = 78
 BYTES = 1
 CYCLES = 2

狀態旗號：

N	V	B	D	I	Z	C
				1		

STA Store Accumulator in memory (將累積器內容存入記憶體)

運算功能： $M \leftarrow [A]$

指令碼格式：

100bbb01	addr	addr
----------	------	------

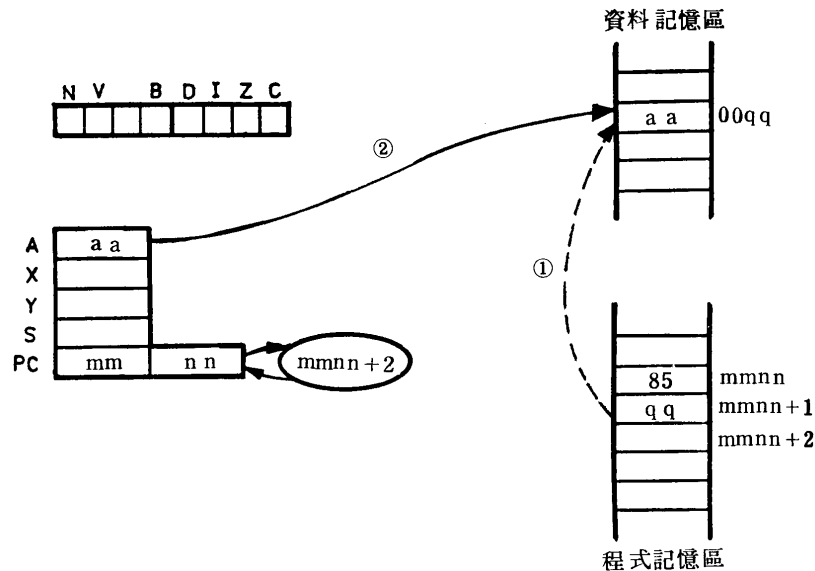
說明：將累積器 A 之內容存入指定的記憶位址，A 之內容保持不變。

定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 (<i>IND,X</i>)	後索引 間接 (<i>IND</i>), Y	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE		8D	85		9D	99	95		81	91			
BYTES		3	2		3	3	2		2	2			
CYCLES		4	3		5	5	4		6	6			
bbb		011	001		111	110	101		000	100			

狀態旗號： 不受 STA 指令之影響。

實例： STA 之零頁定址法： STA \$qq



STX Store X in memory (將 X 暫存器之內容存入記憶體)

運算功能： $M \leftarrow [X]$

指令碼格式：

100bb110	addr	addr
----------	------	------

說明：將 X 暫存器之內容存入指定的記憶位址，X 之內容保持不變。

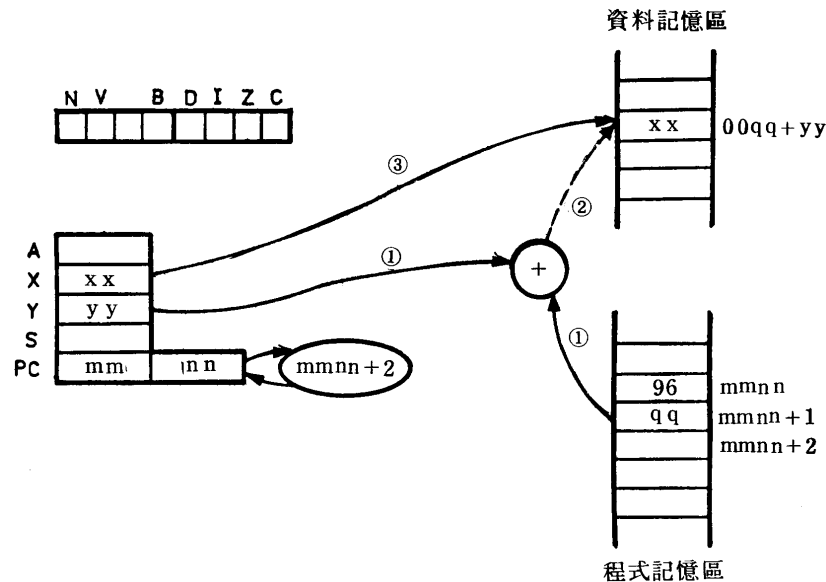
注意：STX 指令不能使用包含 X 暫存器之定址法。

定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 (<i>IND,X</i>)	後索引 間接 (<i>IND,Y</i>)	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE		8E	86					96					
BYTES		3	2					2					
CYCLES		4	3					4					
bbb		01	00					10					

狀態旗號： 不受 STX 指令的影響。

實例： STX 之 "Y 暫存器零頁索引定址法"： STX \$qq, Y



STY Store Y in memory (將 Y 暫存器內容存入記憶體)

運算功能： $M \leftarrow [Y]$

指令碼格式：

100bb100	addr	addr
----------	------	------

說明：將 Y 暫存器之內容存入指定的記憶位址中，Y 暫存器內容不變。

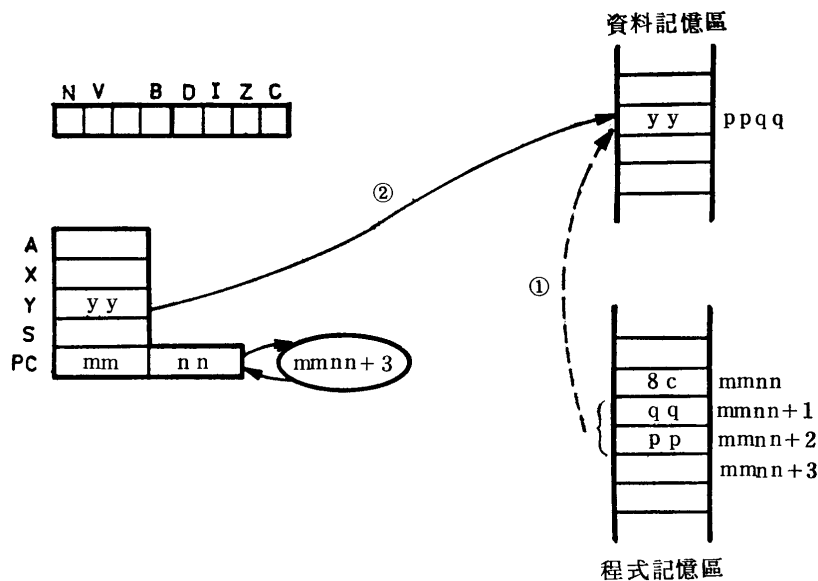
注意：STY 指令不能使用包含 Y 暫存器之定址法。

定址模式：

	立即 <i>IMMEDIATE</i>	絕對定址法 <i>ABSOLUTE</i>	零頁定址法 <i>0-PAGE</i>	<i>INDIRECT</i>	絕對索引 X <i>ABS-X</i>	絕對索引 Y <i>ABS-Y</i>	零頁索引 X <i>0-PAGE X</i>	零頁索引 Y <i>0-PAGE Y</i>	先索引 間接 (<i>IND,X</i>)	後索引 間接 (<i>IND,Y</i>)	相對 <i>RELATIVE</i>	隱含 <i>IMPLIED</i>	累加器 <i>ACCUMULATOR</i>
OP CODE		8C	84				94						
BYTES		3	2				4						
CYCLES		4	3				4						
bbb		01	00				10						

狀態旗號： 不受 STY 指令的影響。

實例： STY 之絕對定址法： STY \$ppqq



TAX Transfer Accumulator into X (將累積器內容移入 X 暫存器)

運算功能： $X \leftarrow [A]$

指令碼格式：

10101010

說明：將累積器 A 之內容存至 X 暫存器，A 之內容不變。狀態旗號 N 與 Z 由累積器內容來設定。

6502 無法直接將 X 或 Y 暫存器之內容存入堆疊區中，所以，跳到副程式之前，如果需要將 X 或 Y 暫存器的內容儲存起來時，可利用下面兩個指令：

TXA ；先將 X 之內容存至 A（請參考 TXA 指令）

PHA ；將 A 之內容存入堆疊區

執行完副程式之後，想從堆疊區取回 X 之內容時：

PLA ；從堆疊區取出資料存至 A

TAX ；將 A 之內容移入 X 暫存器

定址模式： 隱含定址法。

OP Code = AA

BYTES = 1

CYCLES = 2

狀態旗號：

N	V	B	D	I	Z	C
✓					✓	

TAY Transfer Accumulator into Y（將累積器內容移入 Y 暫存器）

運算功能： $Y \leftarrow [A]$

指令碼格式：

10101000

說明：將累積器 A 之內容移至 Y 暫存器，A 之內容不變。TAY 指令之用法與 TAX 類似，請參考 TAX 之說明。

定址模式： 隱含定址法。

OP Code = A8

BYTES = 1

CYCLES = 2

狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

TSX Transfer S into X（將堆疊暫存器之內容移至 X 暫存器）

運算功能： $X \leftarrow [S]$

指令碼格式：

10111010

說明：將堆疊暫存器 S 之內容存至 X 暫存器，S 之內容不變。TSX 指令是 6502 中，惟一對堆疊暫存器做提取（access）運算的指令。

如果需要將 S 的內容儲存在記憶體時，可利用下面兩個指令：

TSX ；將 S 之內容存到 X

STX \$ppqq ；將 X 之內容存到記憶位址 \$ppqq

定址模式： 隱含定址法

OP Code = BA

BYTES = 1

CYCLES = 2

狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

TXA Transfer X into Accumulator（將 X 暫存器內容移至累積器）

運算功能： $A \leftarrow [X]$

指令碼格式：

10001010

說明：將 X 暫存器內容存到累積器 A，X 之內容不變。（請參考 TAX 之說明）

定址模式： 隱含定址法。

OP Code = 8A

BYTES = 1

CYCLES = 2

狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

TXS Transfer X into S（將 X 暫存器內容移到堆疊暫存器）

運算功能： $S \leftarrow [X]$

指令碼格式：

10011010

說明：TXS 指令可將 X 暫存器內容移至堆疊暫存器 S，X 之內容保持不變。

前面提到，PLA 或 PLP 指令只能從堆疊區頂端提取資料；如果想從堆疊區中間提取資料時，怎麼辦？我們必須先設定堆疊暫存器的內容，使使它可以指向所要的堆疊位址，接著即可從該位址提取資料了。

程式如下：

STX	#\$dd	；先將所要的堆疊位址之低位元組 \$dd 存入 X
TXS		；將 X 之內容存至 S
PLA		；從 S 所指向的堆疊位址 \$01dd 提出資料存到累積器中。

定址模式： 隱含定址法

OP Code = 9A

BYTES = 1

CYCLES = 2

狀態旗號： 不受 TXS 指令的影響。

注意：TSX 指令對旗號 N 與 Z 有影響，而 TXS 對旗號均無影響。

圖解 6502 指令集

80

TYA Transfer Y into Accumulator（將 Y 暫存器內容移至累積器）

運算功能： $A \leftarrow [Y]$

指令碼格式：

10011000

說明：將 Y 暫存器內容移至累積器，Y 之內容不變。（請參考 TXA 指令）

定址模式： 隱含定址法。

OP Code = 98

BYTES = 1

CYCLES = 2

狀態旗號：

N	V		B	D	I	Z	C
✓						✓	

文件名稱：圖解 6502 指令集

文件分類	I
文件編號	00029
文件批號	00

製作群	原稿掃圖	文稿編輯
	原稿圖文分離	文稿整合
	原稿辨識	文稿校對
	文稿成品輸出	特別感謝名單

文件完成日期	初版	2007-01-09	其他加註
	再版		

文件出處	原圖書名	6502 組合語言
	原圖書作者	楊喜達
	原圖書出版者	小教授電腦股份有限公司出版部
	原圖書出版日期	民國 72 年 9 月 25 日

DDSC 文件 版權宣告

本文件版權屬原輸出公司、出版社、圖書公司或原作者人所有，作商業用途者請自行洽上述公司，本文件僅可在非商業上流傳或供私人收集資料用。另由於資料老舊，DDSC 不對原書內的內容負責，且除了更正原書內的錯字、漏字之外一切照原書內容所用的文字顯示。

Documents Digitize Service Center 製作
1998-2007

檔名格式說明：

DDSC — 文件分類 — 文件編號 — 文件批號 — 文件名.PDF

以 DDSC 為起頭，加上 1 個字母為分類代碼，再加上以 5 位數由 00001 起的編號，加上 2 位數由 01 起的編號，加上完整的文件名稱而成的。

其中分類代碼詳見下面列表。文件批號指該文件為非合訂版的，可能因書的內容過多而分批完成的，此項可有可無。

文件分類代碼說明	
代 碼	說 明
A	小說／文學類文章類
B	娛樂類
C	天文類
D	科學類
E	古文明事物類
F	自然界類
G	古怪事物類
H	動／植物類
I	電子類
J	電腦類