前言

好想複習數學——不論是什麼理由，這大概就是本書讀者的共同想法吧！當然，想認真學習數學的話，參考書會更適合，但是各位應該不是為了考試而讀書的吧？

我想各位若不是程式設計師，就是正想要成為程式設計師。應該有人會在寫程式時覺得「如果當時有好好念數學就好了……」、「想要學習現在熱門的機器學習和 AI 而買書來讀，但完全看不懂到底在寫啥！」，其中應該也有像這樣挫折的人，如果能幫到有這些想法的讀者……懷著這樣的想法，我寫了這本書。

本書所選擇的範圍，只是從小學到高中數學主題中的一小部分而已。不同於參考書重視問題的解法，我們試著回答學生時代常有的疑問——「這可以用在哪裡？」、「到底有什麼用？」：「在電腦世界裡，可以像這樣使用喔」、「用了這個就可以做到這樣的事情」。本書也選了現在日本高中數學沒學的「矩陣」，因為在電腦圖學（Computer Graphics, CG）與遊戲的世界中，「矩陣」是相當重要的課題。

還有，為了不讓大家就這樣看過去只留下模糊的理解，我們也努力讓各位能用 Python 寫簡單的腳本程式並實際運行，然後發現「原來如此！」。接著改變腳本中的變數值或表達式，並觀察結果會有何改變。我想這樣更能加深對數學的理解，同時也能學到以程式實作數學公式等要領。

再說一次，本書並不是幫忙解決數學問題的書，本書的目標是讓各位能了解，在我們身邊——特別是電腦世界中——是如何使用數學的，觀看具體的例子，體驗並真正擁有這些數學知識。數學原本是一門因沒有模棱兩可而美麗的學問，但本書的優先目標是讓數學成為我們身邊熟悉的東西，有些地方對於想學習正統數學的人來說可能會覺得「嗯？」，還請各位睜一隻眼閉一隻眼。

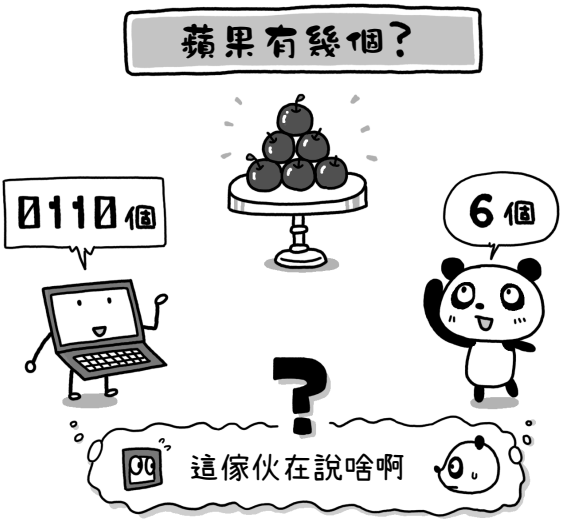
最後，本書寫作時，得到許多電氣通信大學情報理工學系專攻資訊及網路的關口雄太先生的寶貴意見，在此表達由衷的感謝。

谷尻 かおり

第 1 章

電腦與「數字」

人類雖然不像電腦一樣擅長計算，但是有能力分析歸納各種現象。而電腦雖然可以瞬間進行複雜計算得到答案，但就算性能再好也無法自行思考。為了讓人類與電腦能協同作業，人類必須更了解不會思考的電腦。首先就讓我們從「數字」開始，了解電腦是如何進行處理的。



1 進位計數法

雖然不常聽到「進位計數法」這個詞，但其實就是用來表示「數字」的方式。我們平常所使用的是「10 進位計數法」，電腦使用的則是「2 進位計數法」。兩者的差異在於數數時所使用的數字種類，就讓我們從這裡開始了解吧！

1.1 10 進位記數法

我們平常都是遵從 **10 進位計數法**這樣的規則來表達數字。**10 進制**這個詞有聽過嗎？也就是依照

・使用 0、1、2、3、4、5、6、7、8、9 等 10 種數字

・排列在一起的數字，從右側開始按順序是個位、十位、百位……

這樣的規則來表示。數數時，按照 1、2、3……的順序，數到 9 的下一個數時會進位，所以接著是 10、11、12……。因為使用了 10 種數字，所以稱為 **10 進位計數法**（或是 **10 進制**），依照此規則表示的數字稱為 **10 進位數**。

例如「2365」這個數，其中的「2」「3」「6」「5」這些數字並不是無關的，而是

1000 有 2 個

100 有 3 個

10 有 6 個

1 有 5 個

並且將這些全部加起來所得到的值。用數學式表達的話就是

$$2365=(1000\times2)+(100\times3)+(10\times6)+(1\times5)$$

1000、100、10、1 等數值稱為**權重**，是表達每一位數字意義的重要數值。為了解權重的意義，我們將上面的數學式變形一下。

$$\begin{aligned}2365&=(10\times10\times10\times2)+(10\times10\times3)+(10\times6)+(1\times5)\\&=(10^3\times2)+(10^2\times3)+(10^1\times6)+(10^0\times5)\end{aligned}$$

相信你已經發現每一位的權重都是「10 的〇次方」了吧。而且 10 的右上角有個比較小的數字（＊1），從右邊開始 0, 1, 2, 3……逐漸增加。這表示「10 進制裡，位數每往上 1 位，權重會變為 10 倍」。

再來，作為權重基本的「10」這個數字，也就是 10 進制的「10」，我們稱為**基數**或是**底**。像是之後會說明的 2 進制（2 進位計數法）與 16 進制（16 進位計數法），基數就各是「2」與「16」。

＊1　稱為「指數」。

1.2 〇〇的 0 次方

Python 裡面計算「m 的 n 次方」用的是「**」指數運算子。例如 10 的 3 次方

```
>>> 10**3
1000
```

按照這樣輸入就能計算。同樣來計算看看「10 的 0 次方」與「2 的 0 次方」，結果答案只會出現「1」。是否覺得奇怪呢？

```
>>> 10**0
1
>>> 2**0
1
```

10^n（10 的 n 次方）表示「10 連乘 n 次」的意思。按照這個規則，10^1（10 的 1 次方）答案是「10」也很合理。那麼，10^0（10 的 0 次方）又如何呢？10 連乘 0 次，所以 10×0 答案是 0 ——並不是這樣的。「10×0」表示用 0 乘上 10，與 10 連乘 0 次的意思完全不同。

不過「10 連乘 0 次」其實有點難想像，所以請看圖 1-1。10 的右上角的數字每減少 1，計算出來的值就變成 $\frac{1}{10}$。按照這樣下去，10 的 0 次方就是「1」。

圖 1-1　指數的值越來越小的話……（10 進位數的情況）

$$
\begin{aligned}
10^4 &= 10000 \\
10^3 &= 1000 \quad \times\frac{1}{10} \\
10^2 &= 100 \quad \times\frac{1}{10} \\
10^1 &= 10 \quad \times\frac{1}{10} \\
10^0 &= 1 \quad \times\frac{1}{10}
\end{aligned}
$$

同樣來看看基數是「2」的情形（圖 1-2）。2的右上角的數字每減少 1，計算出來的值就變成 $\frac{1}{2}$ 倍，所以 2^0（2 的 0 次方）就是「1」。也就是說，不論基數是什麼值，「○○的 0 次方」一定會是「1」。

圖 1-2　指數的值越來越小的話……（2 進位數的情況）

$$
\begin{aligned}
2^4 &= 16 \\
2^3 &= 8 \quad \times\frac{1}{2} \\
2^2 &= 4 \quad \times\frac{1}{2} \\
2^1 &= 2 \quad \times\frac{1}{2} \\
2^0 &= 1 \quad \times\frac{1}{2}
\end{aligned}
$$

1.3 2 進位計數法

我們使用 10 進制來計數，在電腦世界則是採用 **2 進位計數法（2 進制）**。也就是遵循

- 使用 0 與 1 這兩個數字
- 排列在一起的數字，從右側開始按順序表示 2^0、2^1、2^2、2^3……

這樣的規則來表示數字，而依照此規則表示的數值則稱為 **2 進位數**。

為什麼電腦要使用 2 進制呢？因為電腦是使用電力來操作的機器。燈泡有電流通過就會亮燈，沒有電流通過的時候就不會亮，對吧？當然電腦裡面沒有燈泡，而是稱為 IC 的電子零件，但用流經電子零件的電流訊號來代表開與關的原理相同。電腦處理的是電流訊號的開與關，用數字表示就只有「1」與「0」兩種。

接下來我們就來用電腦的方式計數吧！從 0 開始，首先就是 1，這樣就把能用的 2 種數字都用完了，所以進位到下一位，因此接著是 10、11，然後再次進位，100、101、110、111……一直這樣下去。對了，2 進制的數值就按照所看到的「一」和「零」發音。譬如「10」念成「一、零」，「100」念成「一、零、零」。

Try Python 從 10 進位數轉換為 2 進位數

要在 Python 中將 10 進位數轉為 2 進位數，可以使用 bin() 函式。

```
>>> bin(10)     ←將 10 進位數的「10」作為引數傳入，轉為 2 進位數
'0b1010'        ←轉換結果（顯示為 2 進位的「1010」）
```

轉換結果開頭的「0b」表示此數值為 2 進位數，這種表示方式是 Python 的規則。其他還有 16 進位數會加上「0x」、8 進位數會加上「0o」的規則，可以先記起來。

Column ○進制與○進位數

要表示數字，有「10 進位計數法」與「2 進位計數法」等各種方式。本書中寫到「10 進制」、「2 進制」時，就是指其各自的進位計數法。另外，「10 進位數」與「2 進位數」各代表以其進位計數法所表示的數值。本書也會出現「用 2 進制表記的 1101」和「2 進制的 1101」這樣的用法，其意義都是相同的。

1.4 16 進位計數法

對電腦來說，2 進制是方便的計數方式，但位數很多，因此對我們來說不太容易處理。解決的方式是使用 **16 進位計數法（16 進制）**，也就是遵循

- 使 用 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F 這 16 種 (＊2) 文字
- 排列在一起的值，從右側開始按順序代表 16^0 、 16^1 、 16^2 、 16^3 ……

這樣的規則來表示數字，而以此規則表示的數值稱為 **16 進位數**。

為了讓每一位能表達 16 種數值而動用到字母，理由就是

$$16=2\times2\times2\times2$$

也就是說，有了 16 種數值，可以表達的數值就相當於 2 進位數的 4 位數。例如「11111111」乍看之下很難看出有多少個 1，但如果換成「FF」呢？懂 16 進制的人，就能馬上在腦中轉成「1111 1111」吧！電腦是以 2 進制處理資料，而我們則是習慣 10 進制，在這兩者之間作為中介的 16 進制，其實是一種很便利的表示法。

表 1-1 是以各種進位計數法來表示 0 ～ 31 的值。

表 1-1　0 ～ 31 的值

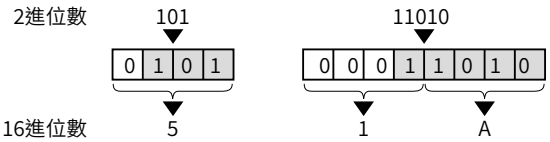
10 進制	2 進制	16 進制	10 進制	2 進制	16 進制
0	0	0	16	10000	10
1	1	1	17	10001	11
2	10	2	18	10010	12
3	11	3	19	10011	13
4	100	4	20	10100	14
5	101	5	21	10101	15
6	110	6	22	10110	16
7	111	7	23	10111	17
8	1000	8	24	11000	18
9	1001	9	25	11001	19
10	1010	A	26	11010	1A
11	1011	B	27	11011	1B
12	1100	C	28	11100	1C
13	1101	D	29	11101	1D
14	1110	E	30	11110	1E
15	1111	F	31	11111	1F

我們先來看看將 2 進位數轉換為 16 進位數的方法（＊3），重點在於這兩點：

- 進位計數法中，其意義是從右邊的位數開始
- 2 進位的 4 位相當於 16 進位的 1 位

根據這兩點，首先我們將 2 進位數從右邊開始，每 4 位分一段，再將分好段的 4 位數逐段轉成 16 進位數。而「101」這樣未滿 4 位的，就在左側補 0 寫成「0101」。譬如像「11010」就寫成「0001 1010」，再將前面 4 位與後面 4 位各自換成 16 進位數（圖 1-3）。

圖 1-3　從 2 進位數轉成 16 進位數



＊2　用小寫的 a、b、c、d、e、f 也可以。
＊3　從 10 進位數轉為 2 進位數或 16 進位數的方法會在「2 基數轉換」說明。

Try Python 從 10 進位數、2 進位數轉換成 16 進位數

用 hex() 函式可以將 10 進位數與 2 進位數轉換成 16 進位數。另外，如前所述，在 Python 中，16 進位數前面會加上「0x」，2 進位數則會加上「0b」。

```
>>> hex(28)          ← 10 進位數的「28」轉換為 16 進位數
'0x1c'               ←顯示結果（16 進位的「1C」）
>>> hex(0b11010)     ← 2 進位數的「11010」轉換為 16 進位數
'0x1a'               ←顯示結果（16 進位的「1A」）
```

2 基數轉換

10 進位數轉成 2 進位數，2 進位數轉成 10 進位數——將某個數字表示用別的進位計數法來表示，稱為**基數轉換**。

2.1 10 進位轉 2 進位

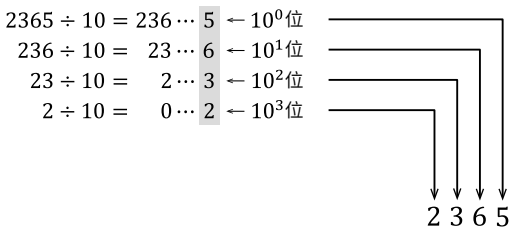
10 進位數的「2365」就是

1000 的位是 2
100 的位是 3
10 的位是 6

1 的位是 5

那麼這些位數為何是「2」「3」「6」「5」呢？請用計算的方式來找答案。該怎麼計算呢？「就是 2365 啊，用看的不就知道了嗎！」這樣的回答不行喔，請看著圖 1-4 思考看看。

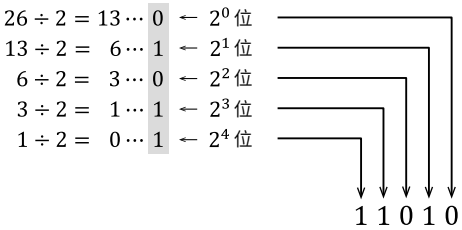
圖 1-4　10 進位數每一位的值



答案是「將原本的值不斷除以基數所得到的餘數」。因為是 10 進制，請用 10 作為除數，此時的餘數就是個位的值。接著把商再除以 10，第二次除法的餘數就是十位數的值。接著再把商除以 10……，如此重複直到商為 0 為止，就能取得每一位的值。算完以後，將餘數按順序由右邊開始排列。然後……就能得到原來的值！

10 進位數轉 2 進位數時，請除以目標進制的基數「2」，並重複進行除法直到商為 0，再將所得餘數從右邊開始排列，這樣就能將原來的 10 進位數以 2 進位數來表示（圖 1-5）。

圖 1-5　10 進位數轉成 2 進位數



Try Python 將 10 進位數轉換為 2 進位數的程式

我們將圖 1-5 所進行的操作改用電腦來進行。程式 1-1 的 dec2bin() 函式可以將 10 進位數的值轉成 2 進位數。Python 的 bin() 函式內部也是進行這樣的處理。

執行 dec2bin() 函式時，引數 target 請給 10 進制的數值。譬如要將「26」轉成 2 進位數時，就像下面這樣執行指令。雖然跟 bin() 函式的表示方式不同，但數字排列是一樣的。

```
>>> dec2bin(26)      ←以 dec2bin() 函式將 10 進制的 26 轉成 2 進制
[1, 1, 0, 1, 0]      ←顯示結果
>>> bin(26)          ←以 Python 的 bin() 函式將 10 進制的 26 轉成 2 進制
'0b11010'            ←顯示結果
```

程式 1-1　將 10 進位數轉成 2 進位數的「dec2bin」

```
 1. def dec2bin(target):
 2.     amari = []   # 存放餘數的串列
 3.
 4.     # 直到除法的商為 0
 5.     while target != 0:
 6.         amari.append(target % 2)    # 餘數    ←①
 7.         target = target // 2        # 商
 8.
 9.     # 反轉串列內元素順序
10.     amari.reverse()                            ←②
11.     return amari
```

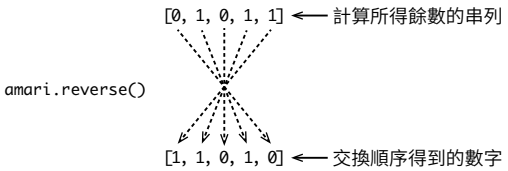
我們簡單看一下程式的內容。amari 是要拿來存放除法的餘數的空串列（list）（譯註：amari 來自日文的「余り」，餘數的意思）。①的 while 迴圈用來控制迴圈，會一直進行到 target 的值變成 0 為止。請執行這段程式：

```
amari.append(target % 2)   ←將 target 除以 2 所得
                            餘數加到 amari 串列
```

```
target = target // 2        ← target 除以 2 後
                              把商覆寫回 target
```

結束 while 迴圈後的②，是將 amari 的元素反向排列的指令，這樣餘數也會是從右排到左了（圖 1-6）。

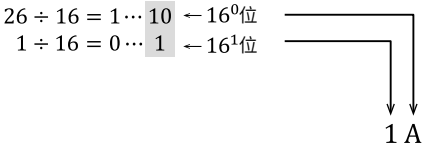
圖 1-6　將串列的元素反向排列



2.2 10 進位轉 16 進位

10 進位數轉成 2 進位數時，是用 2 來進行除法。10 進位數轉成 16 進位數時也是一樣，一直除以 16 直到商為 0，再將餘數從右邊開始排列。只是，除以 16 得到的餘數是 0～15，其中的 10～15 必須換成 A～F（圖 1-7）。

圖 1-7　10 進位數轉為 16 進位數



Try Python 將 10 進位數轉換為 16 進位數的程式

程式 1-2 可以將 10 進位數的值轉成 16 進位數。除了將除法餘數的 10～15 換成 A～F 之外（①的 for 迴圈），其他內容與程式 1-1 幾乎一樣。引數 target 請給 10 進制的數值。

```
>>> dec2hex(26)    ←以 dec2hex() 函式將 10 進位的 26 轉成 16 進位數
[1, 'A']           ←顯示結果
>>> hex(26)        ←以 Python 的 hex() 函式將 10 進制 26 轉成 16 進制
'0x1a'             ←顯示結果
```

程式 1-2　將 10 進位數轉成 16 進位數的「dec2hex」

```
 1. def dec2hex(target):
 2.     amari = []  # 存放餘數的串列
 3.
 4.     # 直到除法的商為 0
 5.     while target != 0:
 6.         amari.append(target % 16)     # 餘數
 7.         target = target // 16         # 商
 8.
 9.     # 將餘數 10 ～ 15 換成 A ～ F
10.     for i in range(len(amari)):
11.         if amari[i] == 10:          amari[i] = 'A'
12.         elif amari[i] == 11:        amari[i] = 'B'
13.         elif amari[i] == 12:        amari[i] = 'C'   ←①
14.         elif amari[i] == 13:        amari[i] = 'D'
15.         elif amari[i] == 14:        amari[i] = 'E'
16.         elif amari[i] == 15:        amari[i] = 'F'
17.
18.     # 反轉串列內元素順序
19.     amari.reverse()
20.     return amari
```

2.3 2 進位與 16 進位轉 10 進位

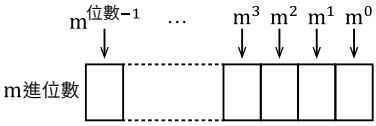
接著我們來將 2 進位數及 16 進位數轉成 10 進位數。其實，其他的進位計數法要轉成 10 進位時，也都可以用同樣的方式，這個法則就是

・m 進制的「m」就是基數

・各個位數的權重以「m 的 n 次方」表示，n 的值由右開始為 0、1、2、3、……，也就是「位數 -1」

這就是進位計數法的特徵（圖 1-8）。

圖 1-8　m 進位數的位數權重



譬如要從「2」「3」「6」「5」的這樣 4 個數字變成 10 進位數的「2365」，只要

$(10^3 \times 2) + (10^2 \times 3) + (10^1 \times 6) + (10^0 \times 5)$

$= 2000 + 300 + 60 + 5$

$= 2365$

像這樣計算即可。

從 2 進位數和 16 進位數轉成 10 進位數時，也是用一樣的計算方式。2 進制的基數是「2」，所以「11010」的話，就是

$(2^4 \times 1) + (2^3 \times 1) + (2^2 \times 0) + (2^1 \times 1) + (2^0 \times 0)$

$= 16 + 8 + 0 + 2 + 0$

$= 26$

這樣就能得到 10 進制的「26」。再來，像 16 進制的「1A」只要這樣計算，就可以得到 10 進制的「26」。

$(16^1 \times 1) + (16^0 \times 10)$

$= 16 + 10$

$= 26$

Try Python 從其他進位計數法轉換為 10 進位數的程式

使用 Python 的 int() 函式，可以將其他進制的數字轉成 10 進位數。第 1 個引數是要轉換的值的字串，第 2 個引數則是指定基數。

```
>>> int('0b11010', 2)   ← 2 進制的「11010」轉成 10 進制
26                      ←顯示結果
>>> int('0x1A', 16)     ← 16 進制的「1A」轉成 10 進制
26                      ←顯示結果
```

雖然只用了 int() 函式，但既然都知道計算方式了，我們就來寫個程式將其他進制的數字轉成 10 進位數吧！請看程式 1-3，引數 target 是轉換前的值的字串，第 2 個引數 m 請指定基數。還有，這個程式所能轉成 10 進位數的，是 2 進位數到 16 進位數。

```
>>> any2dec('11010', 2) ← 2 進制的「11010」轉成 10 進制
26                      ←顯示結果
>>> any2dec('1A', 16)   ← 16 進制的「1A」轉成 10 進制
26                      ←顯示結果
```

程式 1-3　將 m 進位數轉成 10 進位數的「any2dec」

```
1.  def any2dec(target, m):
2.      n = len(target)-1    # 指數的最大值                    ←①
3.      sum = 0              # 轉為 10 進位數後的值
4.
5.      # 重複次數為字元數
6.      for i in range(len(target)):
7.          if target[i] == 'A':     num = 10
8.          elif target[i] == 'B':   num = 11
9.          elif target[i] == 'C':   num = 12
10.         elif target[i] == 'D':   num = 13
11.         elif target[i] == 'E':   num = 14                   ←②
12.         elif target[i] == 'F':   num = 15
13.         else:                    num = int(target[i])
14.
```

```
15.         sum += (m ** n) * num     #「m 的 n 次方 × 每一位的值」加總
16.         n -= 1                    # 權重減少 1 階
17.     return sum
```

①的變數 n 是計算每一位權重時用的值（指數）。不論基數是多少，最高位的指數都是「位數 -1」（＊4）。

②的 for 迴圈則是逐一確認 target 內容的字元，遇到 'A' ～ 'F' 時就轉成 10 ～ 15，在這之外的 '0' ～ '9' 就轉成其數值，然後執行

```
sum += (m ** n) * num     ←「m 的 n 次方 × 每一位的值」加總
n -= 1                   ←權重減少 1 階
```

for 迴圈結束後，此時 sum 的值，就是轉換為 10 進制的值。

＊4　len() 函式可以用來取得引數傳入字串的字元數。

3 關於電腦世界的數字

本書為了增進對於數學的了解，使用擅於計算的電腦來進行各種計算。此時為了不會遺失重要的資料，我們先來看看在電腦的世界中，是如何處理資料的。關鍵字是**位元**與**位元組**。

1 位元（bit）是電腦的最小處理單位，表示 2 進位數的 1 位。8 個位元合在一起就是 1 位元組（Byte）。也就是說，1 個位元組可以處理 2 進位的 8 位數，2 個位元組則是 16 位數（=2×8）。

3.1 資料的處理方式

電腦為了更有效率地進行計算，會將資料放入固定大小的容器中進行讀寫。用來表示容器大小的單位就是**位元組**。譬如，

```
>>> a = 6
```

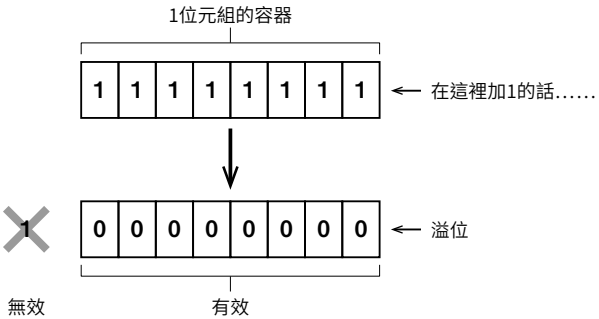
執行之後，電腦就會準備一個稱作 a 的容器（＊5），然後在其中放入「00000110」（圖 1-9）。雖然 10 進位數的「6」用 2 進制表示是「110」，但電腦基本上以位元組為單位，如果用 1 位元組的容器的話，左側就會填入 0，以使用到全部的 8 個位元。

圖 1-9　1 位元組是 8 位元

現在，假設 1 位元組的容器裡面放的是「11111111」。如果再加「1」，就會進位變成「100000000」，但容器只有 1 位元組（8 位數）的大小，此時就會捨棄放不下的值，只有右側的 8 位有效（圖 1-10）。

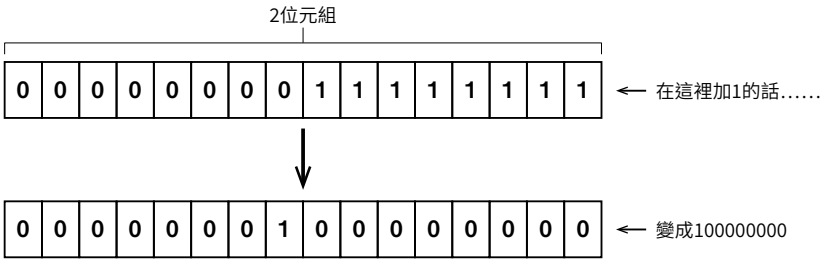
這種狀態稱為**溢位**（overflow）。不管電腦計算有多正確，發生溢位時就無法得到正確答案，請多加注意。

圖 1-10　溢位（overflow）



發生溢位是因為容器對於所處理的資料來說太小了。若是圖 1-10 的情況，只要使用 2 位元組的容器，就能避免發生溢位（圖 1-11）。

圖 1-11　2 位元組是 16 位元

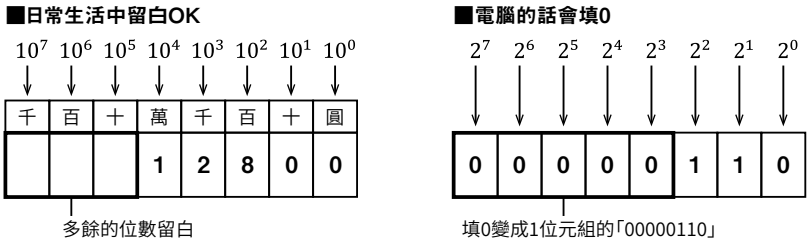


＊5　為了便於說明，在圖 1-9 中使用 1 位元組大小的容器，不過一般程式語言的整數使用的是 4 位元組或 8 位元組。

Column 填 0 的用意

進位計數法中，是從右側開始帶有意義的，所以將數值填入時基本都是靠右的（圖 1-12）。我們日常生活中填寫紙本時，將多餘的部分留白並不會有什麼問題，但電腦是以位元組為單位處理資料的，因此剩餘的位元會填入「0」，這是表示「這一位沒有任何值」的重要數值。

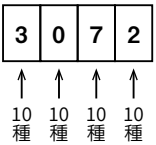
圖 1-12　0 的角色



3.2 處理的數值有其極限

10 進制的 4 位數能表現幾種值呢？——每個位數可以使用 0 ～ 9 的 10 種數字，所以是 10×10×10×10，答案是 10,000 種（圖 1-13）。當然，「10000」是 5 位數所以不算，4 位數能表示的值是 0 ～ 9999。

圖 1-13　10 進制的 4 位數能表示多少種值？



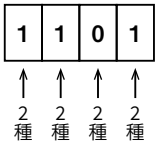
$$10\times10\times10\times10 = 10{,}000\text{種}$$

那麼 2 進制的 4 位數呢？每一位可以用的數字有 0 與 1 兩種，所以 4 位數的話就是

$$2\times2\times2\times2 = 16\text{種}$$

這樣計算（圖 1-14）。

圖 1-14　2 進制的 4 位數能表示的值



$2\times2\times2\times2 = 16$種

10 進制的話取「10」，2 進制的話取「2」，然後將這個數字連乘其位數的次數，所得到的數字就是這麼多位所能表達的數值總數。如果限定 0 以上（正數），那麼可以處理的值就是在

$0\sim m^n - 1$（m 是基數、n 是位數）

這個範圍內（表 1-2）。

表 1-2　可處理的數值範圍

位元組數	位元數	值的種類	數值範圍（正數）
1	8	$2^8 = 256$	0～255
2	16	$2^{16} = 65{,}536$	0～65,535
4	32	$2^{32} = 4{,}294{,}967{,}296$	0～4,294,967,295
8	64	$2^{64} = 18{,}446{,}744{,}073{,}709{,}551{,}616$	0～18,446,744,073,709,551,615

4 負數的處理方式

我們一般會在數字前面加上「－」來表示負數，就像「－5」、「－10」這樣，但在電腦世界中，所有資訊都以 0 和 1 來處理，所以不是使用「－」，而是以**符號位元**來表示負數。

4.1 計算 $x+1=0$

10 進位數的計算中，加了 1 會得到 0 的值是什麼呢？如果列出方程式，就是

$$x+1=0$$
$$x=-1$$

接著是 2 進位數的計算，考慮 8 位數的情形。加上「0000 0001」會得到「0000 0000」的值是什麼呢？請看圖 1-15 並思考看看。

圖 1-15 「1111 1111」加上「1」

	1	1	1	1	1	1	1	1
+	0	0	0	0	0	0	0	1
~~1~~	0	0	0	0	0	0	0	0

「1111 1111」加上「0000 0001」會進 1 位得到「1 0000 0000」，但前面有個條件是「考慮 8 位數的情形」，所以我們捨棄超過的位數，得到的答案是「0000 0000」。

加上 1 會變成 0 的值只有「－1」，也就是說，10 進位的「－1」如果用 2 進制表示就是「1111 1111」。

——這個說法，各位能欣然接受嗎？如本章「2.3 2 進位與 16 進位轉 10 進位」的說明，2 進制的「1111 1111」是

$$\begin{aligned}&2^7\times1+2^6\times1+2^5\times1+2^4\times1+2^3\times1+2^2\times1+2^1\times1+2^0\times1\\=&\ 128+64+32+16+8+4+2+1\\=&\ 255\end{aligned}$$

相當於 10 進位數的「255」，顯然跟「－1」是不同的數值，但在 2 進制卻成了同樣的值，思緒是否已經有點亂了呢？

4.2 2的「補數」是什麼？

我們將 2 進制簡化到 4 位數來思考這個問題。表 1-3 表示的是，當值 a 在 0 ～ 15 時，能使

$$a + x = 0$$

成立的值（圖 1-16）。仔細看 2 進制的 x 那欄，除了「0000」外，值 a 是從下面開始由小到大排列的。

圖 1-16　$a + x = 0$ 的計算

$$\begin{array}{r} a \\ +\ \ x \\ \hline 0 \end{array}$$

10進制

$$\begin{array}{r} 1 \\ +\ \ -1 \\ \hline 0 \end{array}$$

2進制

$$\begin{array}{r} 0001 \\ +\ \ 1111 \\ \hline \cancel{1}\,0000 \end{array}$$

表 1-3　使 $a + x = 0$ 成立的 x 值

10 進制		2 進制	
a	x	a	x
0	0	0000	0000
1	-1	0001	1111
2	-2	0010	1110
3	-3	0011	1101
4	-4	0100	1100
5	-5	0101	1011
6	-6	0110	1010
7	-7	0111	1001
8	-8	1000	1000
9	-9	1001	0111
10	-10	1010	0110
11	-11	1011	0101
12	-12	1100	0100
13	-13	1101	0011
14	-14	1110	0010
15	-15	1111	0001

2 進制的 x 欄所列的值稱為 **2 補數**，用下面的步驟就能得到：

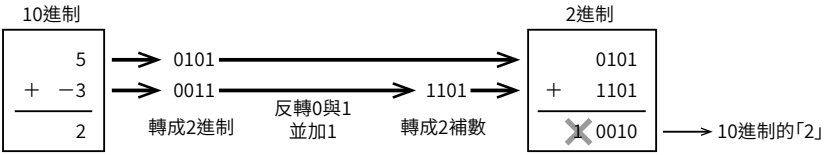
①將 2 進制每一位的 0 與 1 反轉

②反轉後得到的值加 1

譬如 2 進位數的「0001」，將每一位都反轉會得到「1110」，再加 1 就是「1111」，相當於 10 進制的「15」。看起來好像很不可思議，但 2 補數的確可以在計算中當作負數使用，接下來就來確認這件事。

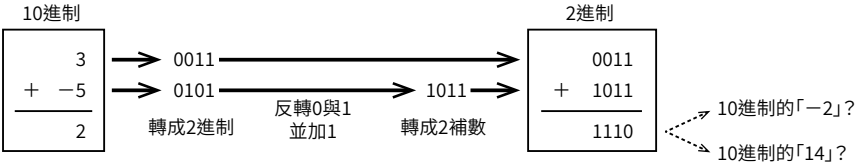
圖 1-17 是「5－3」的計算過程。首先將 2 個數字都用 2 進制表示，並且求出減數的 2 補數，這是為了將「5－3」變成用加法來計算的「5＋（－3）」。最後將 2 進位數的「0101」與「1101」相加得到「1 0010」，但因為只考慮 4 位數，所以捨棄超過的位數，得到答案「0010」，相當於 10 進位數的「2」。

圖 1-17 「5－3」的計算過程



再來一個，這次計算「3－5」（圖 1-18）。這可以視為「3＋（－5）」，並在計算中使用「5」的 2 補數。最後將「0011」與「1011」相加，答案是「1110」。看表 1-3 中 2 進位數的 x 欄，的確「1110」是「2」的 2 補數表示法，也就是「－2」。但是看 a 欄的話，「1110」也是 10 進位數的「14」。

圖 1-18 「3－5」的計算過程



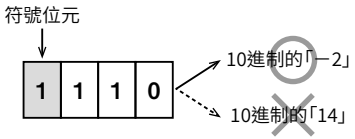
如同我們一路看過來的，對電腦來說，2 補數是很方便的數值，因為可以用加法線路來計算減法。但是同樣的數值有兩種詮釋方式也很令人困擾，因此接下來該輪到**符號位元**登場了。

4.3 以符號位元區分正負

我們人類定了規則「數字前面加上“－”表示負數」，但是電腦的世界裡只能處理 0 跟 1，因此我們將 2 進位數最左側的這 1 位命名為「符號位元」，並定下規則：

當符號位元為 0 時表示正數，為 1 時則表示負數（圖 1-19）。

圖 1-19　符號位元



使用這個規則的話，2 補數就只會有一種詮釋，2 進制的 4 位數所能表示的值則如表 1-4。

表 1-4　2 進制的 4 位數所能表示的值

2 進制	10 進制	2 進制	10 進制
0000	0	1000	-8
0001	1	1001	-7
0010	2	1010	-6
0011	3	1011	-5
0100	4	1100	-4
0101	5	1101	-3
0110	6	1110	-2
0111	7	1111	-1