表 2-3　十六進位數與其相對應的二、八、十進位數對照表

十六進位數	二進位數	八進位數	十進位數
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
A	1010	12	10
B	1011	13	11
C	1100	14	12
D	1101	15	13
E	1110	16	14
F	1111	17	15
10	10000	20	16
⋮	⋮	⋮	⋮

數字系統	識　別　符　號
十進位	在數字右下角加上 10，或 D（Decimal），爲預設進位。
二進位	在數字右下角加上 2，或 B（Binary）
八進位	在數字右下角加上 8，或 O（Octal）
十六進位	在數字右下角加上 16，或 H（Hexidecimal）

英文字母置於數目右方

$2^8 = 256$

$2^{10} = 1024 = 1K$ (Kilo) $\fallingdotseq 10^3$

$2^{16} = 65536 = 64 \times 1024 = 64K$

$2^{20} = 1K \times 1K = 1M$ (Mega) $\fallingdotseq 10^6$

$2^{30} = 1K \times 1M = 1G$ (Giga) $\fallingdotseq 10^9$

$2^{40} = 1M \times 1M = 1T$ (Tera) $\fallingdotseq 10^{12}$

數字的最左邊爲最高位數（MSD，Most significant digit），最右邊爲最低位數（LSD，least significant digit）。

(1)一個二進位數字，稱爲一個「位元」（Bit）。其最高位元稱爲 MSB（most significant bit），最低位元稱爲 LSB（least significant bit）

(2)八個位元稱爲一個位元組（byte），四個位元稱爲半位元組（nibble）。

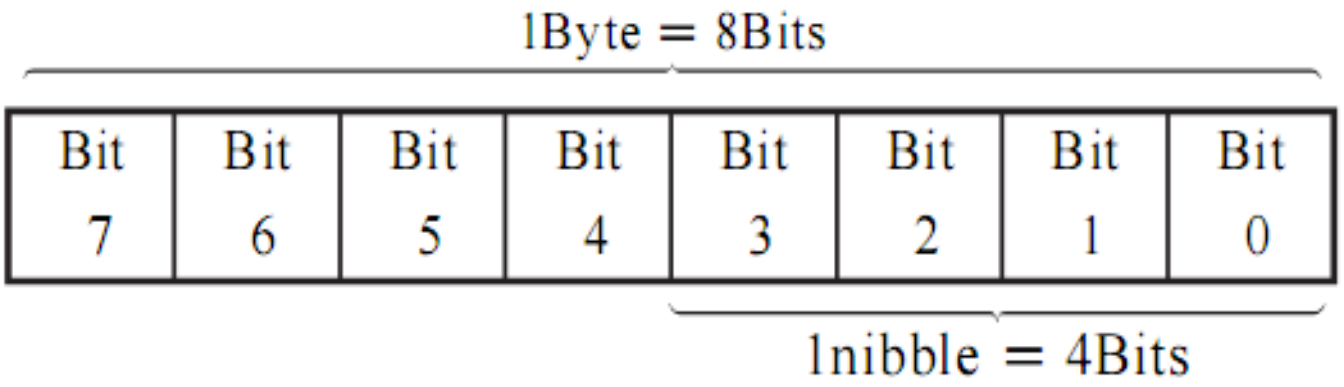


圖 2-1　一個位元組（Byte）的組成

例題 1 二進數轉換成十進數

$$1011.101_{(2)} = 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$$
$$= 8 + 2 + 1 + 0.5 + 0.125$$
$$= 11.625_{(10)}$$

例題 2 八進數轉換成十進數

$$34.5_{(8)} = 3\times8^1 + 4\times8^0 + 5\times8^{-1}$$
$$= 24 + 4 + 0.625$$
$$= 28.625_{(10)}$$

其他進數轉換成
十進數的方法亦同

例題 3 十六進數轉換成十進數

$$1D8.4H = 1\times16^2 + 13\times16^1 + 8\times16^0 + 4\times16^{-1}$$
$$= 256 + 208 + 8 + 0.25$$
$$= 472.25_{(10)}$$

整數的十進位數轉換成二進位數

1. 權值比例和法

此方法是直接找出總和等於被轉換的十進位數其適當權值比例的組合，較適用於數值不大(通常小於 512 較佳)，而且使用者須熟記 2 的各項乘方值，

例題 4

試將 $108_{(10)}$ 轉換成二進位數

解

$$
\begin{array}{r}
108 \\
-\ 64 \\
\hline
44 \\
-\ 32 \\
\hline
12 \\
-\ 8 \\
\hline
4 \\
-\ 4 \\
\hline
0
\end{array}
$$

- 108 → 最接近 108，且小於或等於 108 的 2 的乘方值爲 64 (2^6)
- 44 → 最接近 44，且小於或等於 44 的 2 的乘方值爲 32 (2^5)
- 12 → 最接近 12，且小於或等於 12 的 2 的乘方值爲 8 (2^3)
- 4 → 最接近 4，且小於或等於 4 的 2 的乘方值爲 4 (2^2)

所以 $108_{(10)} = 1\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0_{(2)}$

$2^6\ \ 2^5\ \ 2^4\ \ 2^3\ \ 2^2\ \ 2^1\ \ 2^0$

$64\ \ 32\ \ 16\ \ 8\ \ 4\ \ 2\ \ 1$

(各位元的權值)

例題 5

試將 $243_{(10)}$ 轉換成二進位數

解

$$
\begin{array}{rl}
243 & \\
-\ 128 & \dashrightarrow \text{最接近 243，且小於或等於 243 的 2 的乘方值爲 128 } (2^7) \\
\hline
115 & \\
-\ 64 & \dashrightarrow \text{最接近 115，且小於或等於 115 的 2 的乘方值爲 64 } (2^6) \\
\hline
51 & \\
-\ 32 & \dashrightarrow \text{最接近 51，且小於或等於 51 的 2 的乘方值爲 32 } (2^5) \\
\hline
19 & \\
-\ 16 & \dashrightarrow \text{最接近 19，且小於或等於 19 的 2 的乘方值爲 16 } (2^4) \\
\hline
3 & \\
-\ 2 & \dashrightarrow \text{最接近 3，且小於或等於 3 的 2 的乘方值爲 2 } (2^1) \\
\hline
1 & \\
-\ 1 & \dashrightarrow \text{最接近 1，且小於或等於 1 的 2 的乘方值爲 1 } (2^0) \\
\hline
0 &
\end{array}
$$

所以 $243_{(10)} = 1\ 1\ 1\ 1\ 0\ 0\ 1\ 1_{(2)}$

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$128\ 64\ 32\ 16\ 8\ 4\ 2\ 1$

(各位元的權值)

例題 4

試將 $108_{(10)}$ 轉換成二進位數

STEP1： 64 32 16 8 4 2 1 (總和為64×2−1=127)

STEP2： 64 32 16 8 4 2 1 (劃記數字之和為108)

STEP3： 1 1 0 1 1 0 0 (答案)

例題 5

試將 $243_{(10)}$ 轉換成二進位數

STEP1： 128 64 32 16 8 4 2 1 (總和為128×2−1=255)

STEP2： 128 64 32 16 8 4 2 1 (劃記數字之和為243)

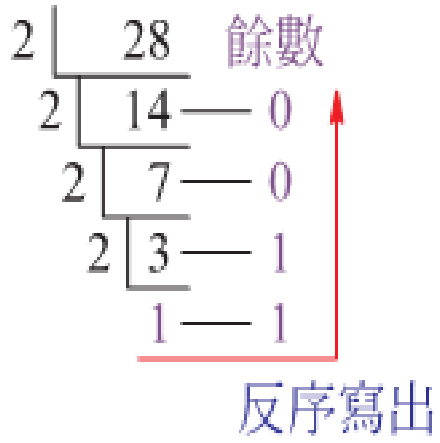
STEP3： 1 1 1 1 0 0 1 1 (答案)

2. 2的連除法

例題 6

試將 $28_{(10)}$ 轉換成二進位數

解

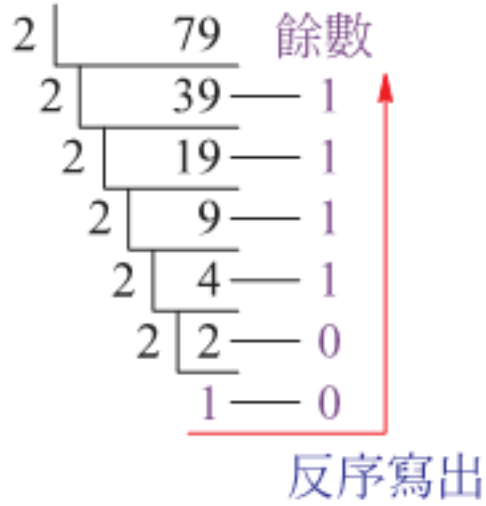


所以 $28_{(10)} = 11100_{(2)}$

例題 7

試將 $79_{(10)}$ 轉換成二進位數

解



所以 $79_{(10)} = 1001111_{(2)}$

有小數的十進位數轉換成二進位數

例題 9

試將 $19.3125_{(10)}$ 轉換成二進位數

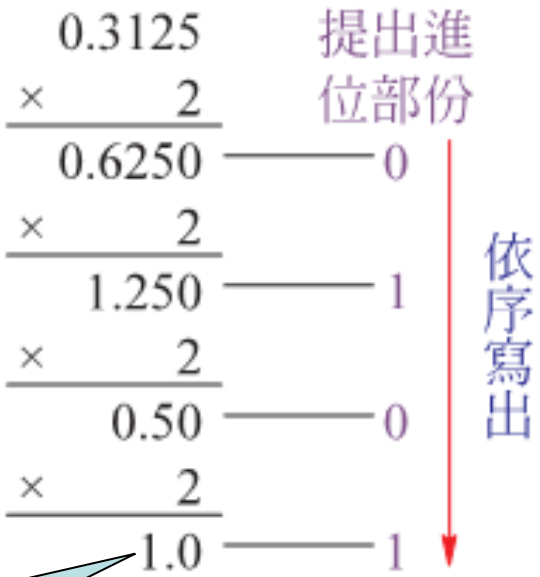
十進數轉換成其他進數的小數部分可能有誤差

解

整數部份

所以 $19_{(10)} = 10011_{(2)}$

小數部份



所以 $0.3125_{(10)} = 0.0101_{(2)}$

故 $19.3125_{(10)} = 10011.0101_{(2)}$

若永遠不為1.0則取適當的位元數作為結果，位元數越多則還原後越精確

2-5-2　十進位數轉換成八進位數

例題 11

試將 $142.6875_{(10)}$ 轉換成八進位數

解

整數部份

$8 \mid 142$

$8 \mid 17$ — 6

2 — 1

所以 $142_{(10)} = 216_{(8)}$

小數部份

0.6875×8

5.5000 — 5

$\times 8$

4.0 — 4

所以 $0.6875_{(10)} = 0.54_{(8)}$

故 $142.6875_{(10)} = 216.54_{(8)}$

2-5-3　十進位數轉換成十六進位數

例題 13

試將 $158.8125_{(10)}$ 轉換成十六進位數

解

整數部份

16 | 158　　餘數

9 —— 14(E)

所以 $158_{(10)} = 9EH$

小數部份

$$\begin{array}{r} 0.8125 \\ \times \quad 16 \\ \hline 4\,8750 \\ +8\,125 \\ \hline 13.0000 \end{array}$$ —— 13 (D)

所以 $0.8125_{(10)} = 0.DH$

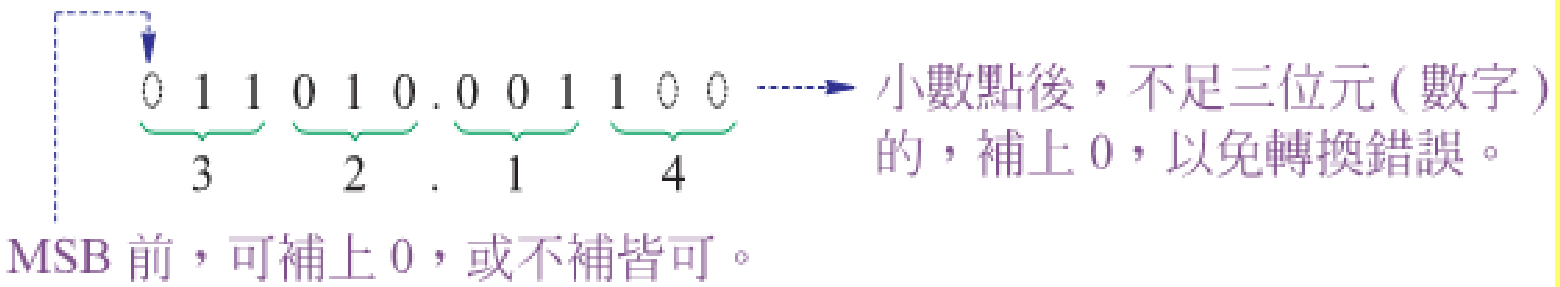
故 $158.8125_{(10)} = 9E.DH$

十進數轉換成其他進數的方法亦同

例題 14

試將 $11010.0011_{(2)}$ 轉換成八進位數

解

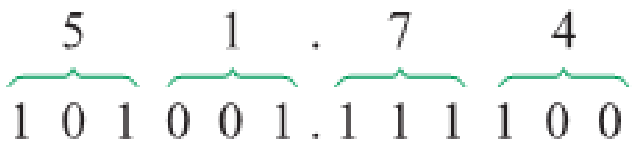


所以 $11010.0011_{(2)} = 32.14_{(8)}$

例題 15

試將 $51.74_{(8)}$ 轉換成二進位數

解



所以 $51.74_{(8)} = 101001.1111_{(2)}$

例題 16

試將 $1101101.10101_{(2)}$ 轉換成十六進位數

解

$$\underbrace{0\ 1\ 1\ 0}_{6}\ \underbrace{1\ 1\ 0\ 1}_{D}\ .\ \underbrace{1\ 0\ 1\ 0}_{A}\ \underbrace{1\ 0\ 0\ 0}_{8}$$

所以 $1101101.10101_{(2)} = 6D.A8H$

例題 17

試將 4FB.9H 轉換成二進位數

解

$$\underbrace{4}_{0\ 1\ 0\ 0}\ \underbrace{F}_{1\ 1\ 1\ 1}\ \underbrace{B}_{1\ 0\ 1\ 1}\ .\ \underbrace{9}_{1\ 0\ 0\ 1}$$

所以 $4FB.9H = 10011111011.1001_{(2)}$

例題 18

試將 $657.14_{(8)}$ 轉換成十六進位數（透過二進數）

解

$$\underbrace{6}_{110}\ \underbrace{5}_{101}\ \underbrace{7}_{111}\ .\ \underbrace{1}_{001}\ \underbrace{4}_{100}{}_{(8)} = \underbrace{1}_{1}\ \underbrace{1\,0\,1\,0}_{A}\ \underbrace{1\,1\,1\,1}_{F}\ .\ \underbrace{0\,0\,1\,1}_{3}\ 0\,0_{(2)}$$

$$=1AF.3H$$

例題 19

試將 6A.FH 轉換成八進位數 （透過二進數）

解

$$\underbrace{6}_{0110}\ \underbrace{A}_{1010}\ .\ \underbrace{F}_{1111}{}_{(H)} = \underbrace{0\,0\,1}_{1}\ \underbrace{1\,0\,1}_{5}\ \underbrace{0\,1\,0}_{2}\ .\ \underbrace{1\,1\,1}_{7}\ \underbrace{1\,0\,0}_{4}\ {}_{(2)}$$

$$=152.74_{(8)}$$

二進數←→四進數：2位元　　二進數←→16進數：4位元

二進數←→八進數：3位元　　二進數←→32進數：5位元

其餘非2^n進數之間的轉換不可依上列方式，必須透過十進數

（一）補數的表示法

補數（Complement）的應用，是用加法取代減法運算，來簡化電腦硬體結構。實際上減法運算是利用“被減數”＋“減數的補數”來完成。

以 r 爲基底的任何數字系統，補數的表示法有兩種：

⑴ r－1 的補數

⑵ r 的補數

例如：基底爲 10，則有 10 的補數與 9 的補數。

基底爲 2，則有 2 的補數與 1 的補數。

1. 補數的求法：

⑴ r－1 的補數求法

每位數都用（r－1）數去減。

範例 1 若基底爲 10，求 5678 取 9 的補數。

解 5678 →取 9'S 補數爲 4321

即
$$\begin{array}{r} 9999 \\ -\ 5678 \\ \hline 4321 \end{array}$$

範例 2 若基底爲 2，求 10101110 取 1 的補數。

解 10101110 →取 1'S 補數爲 01010001

即
$$\begin{array}{r} 11111111 \\ -\ 10101110 \\ \hline 01010001 \end{array}$$

速算法：當二進位數（即 2 爲基底時）取 1 的補數時，只需將 1 改爲 0，0 改爲 1。（只有以 2 爲基底時才可以）

範例 3 若基底爲 16，求 9AB 取 15 的補數。

解 9AB →取 15'S 補數為 $654_{(16)}$

即
$$\begin{array}{r} FFF \\ -\ 9AB \\ \hline 654 \end{array}$$

⑵ r 的補數求法

先求 r−1 的補數，再加 1。

例如：10101110 →1'S 補數 01010001

→2'S 補數 01010001 + 1 = 01010010

速算法：在 2 進位數取 2 的補數時，可將原數由右至左遇到第一個 1 保留，其餘 0 變 1，1 變 0。（只有以 2 爲基底時才可以）

範例 1 將 $00110100_{(2)}$ 取 2'S 補數

解

0 0 1 1 0 1 0 0

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

1 1 0 0 1 1 0 0

即 $00110100_{(2)}$ →2'S 補數為 11001100。

範例 2 若基底爲 10，求 5678 取 10 的補數

解

5678 →取 10'S 補數爲 4322

即 9999

− 5678

4321

＋ 1

4322

範例 3 若基底爲 2，求 10101110 取 2 的補數。

解

10101110 →取 1'S 補數爲 01010010

即 11111111

− 10101110

01010001

＋ 1

01010010

範例 4 若基底爲 16，求 9AB 取 16 的補數

解

FFF

− 9AB

654

＋ 1

655

十六進數的16補數
與其2的補數相同

二進位數表示法

(1)無號數（Unsigned number）表示法（只能表示正數及0）

n位元數值的表示範圍爲0～(2^n-1)。

例如：8位元數值的範圍爲$00000000_{(2)}$～$11111111_{(2)}$，

即$0_{(10)}$～$255_{(10)}$。

此表示法只能做正數及0的運算。

(2)有號數（Signed number）表示法

1的補數表示法（可表示正、負數及0，0只有一種表示方式）

適合做運算，其減法步驟如下：

(a)被減數不變，減數取1的補數，再相加。

(b)相加後若有進位，表結果爲正數，必須將進位加至最低位LSB，即爲結果。此進位稱爲端迴進位。

(c)相加後若無進位，表結果爲負數，且爲1的補數形式。

其n位元可表示數值範圍爲$-(2^{n-1}-1)$到$+(2^{n-1}-1)$。

結果之正負以最高位元(符號元)為準，0為正，1為負

範例 1 試以 8Bits 之 1 的補數法，求 (1) 14−10 = ? (2) 10−14 = ?

解

(1) $+14_{(10)} = 00001110$ $+10_{(10)} = 00001010$

$-14_{(10)} = 11110001$ $-10_{(10)} = 11110101$

$$\begin{array}{rcl} 14 & 14 & 00001110 \\ -\ 10 & \Rightarrow +(-10) \Rightarrow & +\ \ 11110101 \\ \hline 4 & +\ 4 & 1\ \ 00000011 \end{array}$$

$$\begin{array}{r} + \qquad\quad 1 \\ \hline 00000100 = +4 \end{array}$$

(2)

$$\begin{array}{rcl} 10 & 10 & 00001010 \\ -14 & \Rightarrow +(-14) \Rightarrow & +\ 11110001 \\ \hline & -14 & 11111011 = -4 \end{array}$$

（1 的補數形式）

2的補數表示法

適合做運算。其減法步驟如下：

(a)被減數不變，減數取2的補數，再相加。

(b)相加後若有進位，表結果爲正數，將進位捨棄，即爲結果。

(c)相加後若無進位，表結果爲負數，且爲2的補數形式。

目前電腦內部採用此表示法的原因有二，第一爲相加後若有進位則捨棄，所以電路設計簡單，執行速度快。第二爲其可表示數值範圍比1的補數表示法大。

其n位元可表示數值範圍爲-2^{n-1}到$+(2^{n-1}-1)$。

結果之正負以最高位元(符號元)為準，0為正，1為負

範例 1 試以 8Bits 之 2 的補數法，求 (1) 14−10 = ？ (2) 10−14 = ？

解

(1) $+14_{(10)} = 00001110$ $+10_{(10)} = 00001010$

$-14_{(10)} = 11110010$ $-10_{(10)} = 11110110$

14 14 00001110

$-10 \Rightarrow +(-10) \Rightarrow +1111010 = +4$

4 $+4$ $\boxed{1}00000100$

(2) 10 10 00001010

$-14 \Rightarrow +(-14) \Rightarrow +11110010 = -4$

-4 -4 11111100

以2的補數轉換為負值

-00000100

我們來比較三種有號數表示法。對於 n 位元的二進位有號數，其所能表示的數值範圍爲：

~~符號大小表示法：$-(2^{n-1}-1)$到$+(2^{n-1}-1)$~~

1 的補數表示法：$-(2^{n-1}-1)$到$+(2^{n-1}-1)$

2 的補數表示法：-2^{n-1}到$+(2^{n-1}-1)$

2的補數最大範圍

8位元：2^8＝256，正負數各128個，0歸正數，(－1…－128)～(0…+127)

10位元：2^{10}＝1024，正負數各512個，0歸正數，(－1…－512)～(0…+511)

16位元：2^{16}＝65536，正負數各32768個，(－1…－32768)～(0…+32767)

表 2-9　4 位元二進位有號數表示法

十進位	符號大小表示法	1 的補數表示法	2 的補數表示法
＋ 7	0111	0111	0111
＋ 6	0110	0110	0110
＋ 5	0101	0101	0101
＋ 4	0100	0100	0100
＋ 3	0011	0011	0011
＋ 2	0010	0010	0010
＋ 1	0001	0001	0001
＋ 0	0000	0000	0000
−0	1000	1111	0000
−1	1001	1110	1111
−2	1010	1101	1110
−3	1011	1100	1101
−4	1100	1011	1100
−5	1101	1010	1011
−6	1110	1001	1010
−7	1111	1000	1001
−8	（不可能）	（不可能）	1000

（請注意這兩列）

表 2-5　有符號位元 2 的補數表示法

十進位	二進位	十六進位
− 128	10000000	80
− 127	10000001	81
− 126	10000010	82
− 125	10000011	83
− 124	10000100	84
⋮	⋮	⋮
− 3	11111101	FD
− 2	11111110	FE
− 1	11111111	FF
0	00000000	00
+ 1	00000001	01
+ 2	00000010	02
+ 3	00000011	03
⋮	⋮	⋮
+ 125	01111101	7D
+ 126	01111110	7E
+ 127	01111111	7F

溢　位

當計算機執行算術運算，若所得的結果超出其所能表示的範圍時，此現象稱爲溢位(over-flow)，即表示運算結果發生錯誤，答案是不正確的；在大部分的中央處理單元(CPU)中均有一溢位旗號(OF, over-flow flag)，用以記錄 CPU 執行運算的過程中有否發生溢位，而溢位旗號(OF)的定義爲

$$OF = C_n \oplus C_{n-1}$$

其中，C_n表示最高有效位元(MSB)相加後的進位，C_{n-1}則表示次高有效位元(MSB的右邊位元)相加後的進位；⊕符號則表示C_n與C_{n-1}執行互斥或(XOR)運算，即當C_n與C_{n-1}不相同時，OF ＝ 1，若C_n與C_{n-1}相同(同時爲 1 或同時爲 0)時，則 OF ＝ 0。

以 4 位元爲例，其 2 的補數表示法範圍爲-2^{4-1}至$+(2^{4-1}-1)$，即－ 8 至＋ 7；當執行 $7_{(10)} - 4_{(10)} = +3_{(10)}$ 時，不會發生溢位，但執行 $-4_{(10)} - 6_{(10)}$ 時，則將發生溢位，所得的結果是錯誤的。

例題 30

試用 4 位元，以 2 的補數方式計算 $7_{(10)} - 4_{(10)}$

解

$+7_{(10)} = 0111_{(2)}$

$+4_{(10)} = 0100_{(2)}$

$-4_{(10)} = 1100_{(2)}$ ← 取 2 的補數

$C_{n-1} = 1$

$C_n = 1$

$$\begin{array}{r} 0111_{(2)} \\ +\ 1100_{(2)} \\ \hline 1)\ 0011_{(2)} \end{array}$$

進位捨去

MSB = 0，表示運算結果爲正數

由於 $OF = C_n \oplus C_{n-1} = 1 \oplus 1 = 0$，表示運算過程沒有發生溢位，結果是正確的，所以 $7_{(10)} - 4_{(10)} = 0011_{(2)} = +3_{(10)}$

例題 31

試用 4 位元，以 2 的補數方式計算 $-4_{(10)}-6_{(10)}$

解

$+4_{(10)}=0100_{(2)}$取 2 的補數得 $-4_{(10)}=1100_{(2)}$

$+6_{(10)}=0110_{(2)}$取 2 的補數得 $-6_{(10)}=1010_{(2)}$

由於 $OF = C_n \oplus C_{n-1} = 1 \oplus 0 = 1$，表示運算過程發生溢位，結果是錯誤的。

從前面的例子中，可以發現，當運算結果超出 n 位元所能表示的範圍時，就會發生溢位；如 $-4_{(10)}-6_{(10)}=-10_{(10)}$已經超出 4 位元所能表示的範圍 -8至 $+7$，所以運算的結果是錯誤的。

0.0110的1補數為0.1001 (不可為1.1001)

0.0110的2補數為0.1010 (不可為1.1010)

以2補數運算0.5－0.375：

0.375＝0.011B，其2補數為0.101

0.100B
+0.101B
1.001 B 整數進位不取，結果為+0.125

以2補數運算15.5－10.375：

10.375＝01010.011B，其2補數為10101.101 (注意整數)

01111.100B
+10101.101B
100101.001 B 整數最高進位不取，結果為+5.125

2-7-1 BCD(Binary Coded Decimal)碼

BCD碼(二進位十進碼)是以4個位元(bit)來表示一個十進位的數，如表2-6所示爲BCD碼與十進碼的對照表，由表中可發現BCD碼的前10碼(0～9)與二進位數碼完全一樣，而10碼以後就不同了。例如$17_{(10)}$，若以BCD碼表示，則爲$00010111_{(BCD)}$；另外，由於BCD碼仍爲一加權碼(如$0111_{(BCD)}=0\times2^3+1\times2^2+1\times2^1+1\times2^0=7_{(10)}$)，故又常以其權值稱爲8421碼。

表2-6 BCD碼與十進碼的對照表

BCD碼	十進碼
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
00010000	10
00010001	11
00010010	12
⋮	⋮
00100000	20
⋮	⋮

BCD碼的特色是將十進位數以4個位元的二進位數組合，既適合人類閱讀，又適合電腦運算，故廣受大家喜愛。

1. BCD 碼

BCD碼（Binary-Coded Decimal）是二進位編碼的十進位數。BCD 碼是將每一個十進位數字轉換成等值四位元的二進位數，即從 0000 至 1001，而 1010 至 1111 大於 9 不能存在 BCD 碼中，BCD 碼最左及最右位元的 0 不可刪去。

BCD碼是利用 4 個二進位碼來表示十進位的值，因此同一數值若以 BCD 碼表示，其位元數必比二進位碼的位元數多。

例如：$35_{(10)} = 00110101_{(BCD)} = 100011_{(2)}$

BCD 碼之加權值分別爲 2^3，2^2，2^1，2^0，故不特別說明即指 8421 碼。

邏輯電路的內部皆是以二進位運算，但邏輯電路外的世界大多是以十進位運算，所以經常要做十進位和二進位間的轉換。一個大數目在十進位數和二進位數之間轉換會很複雜且很久，因此使用 BCD 碼可以使十進位數和二進位數間轉換利用硬體電路完成。

範例 1 將 $258_{(10)}$ 轉成 BCD 碼。

解

$$\begin{array}{ccc} 2 & 5 & 8_{(10)} \\ \downarrow & \downarrow & \downarrow \\ 0010 & 0101 & 1000_{(BCD)} \end{array}$$

結果 $258_{(10)} = 001001011000_{(BCD)}$

範例 2 將 $732.56_{(10)}$ 轉成 BCD 碼。

解

$$\begin{array}{ccccccc} 7 & 3 & 2 & . & 5 & 6_{(10)} \\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow \\ 0111 & 0011 & 0010 & . & 0101 & 0110_{(BCD)} \end{array}$$

結果 $732.56_{(10)} = 011100110010.01010110_{(BCD)}$

範例 3 將 $001101010111.10000100_{(BCD)}$ 轉成等效的十進位數

解

$$\begin{array}{ccccccc} 0010 & 0101 & 0111 & . & 1000 & 0100_{(BCD)} \\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow \\ 3 & 5 & 7 & . & 8 & 4_{(10)} \end{array}$$

結果 $001101010111.10000100_{(BCD)} = 357.84_{(10)}$

2-7-2 超三碼(Excess-3 Code)

超三碼有時又稱爲加三碼，顧名思義，就是將每一組BCD碼(4bit)都加上$3_{(10)} = 0011_{(2)}$，如表 2-7 所示爲 BCD 碼、加三碼與十進碼的對照表，其特色爲每一組數碼至少都包含一個 1，因此具有檢誤能力，但無法以加權值的方式表示，故並非加權碼。

表 2-7 BCD 碼、加三碼與十進碼的對照表

BCD 碼	加三碼	十進碼
0000	0011	0
0001	0100	1
0010	0101	2
0011	0110	3
0100	0111	4
0101	1000	5
0110	1001	6
0111	1010	7
1000	1011	8
1001	1100	9

例題 32

試將 $52_{(10)}$ 轉換成加三碼

解

$$\begin{array}{rr} 5 & 2 \\ +\ 3 & +\ 3 \\ \hline 8 & 5 \\ \downarrow & \downarrow \\ 1000 & 0101 \end{array}$$

所以 $52_{(10)} = 10000101_{(\text{Excess}^{-3})}$

例題 33

試將加三碼 10010110 轉換成 BCD 碼

解

$$\begin{array}{rr} 1001 & 0110 \\ \downarrow & \downarrow \\ 9 & 6 \\ -\ 3 & -\ 3 \\ \hline 6 & 3 \\ \downarrow & \downarrow \\ 0110 & 0011 \end{array}$$

所以 $10010110_{(\text{Excess}^{-3})} = 63_{(10)}$

$= 01100011_{(\text{BCD})}$

範例 1

將 (1) $42_{(10)}$ (2) $12.345_{(10)}$ 轉成超三碼

解

(1)

$$\begin{array}{cc} 4 & 2_{(10)} \\ \downarrow & \downarrow \\ 0100 & 0010_{(BCD)} \\ \downarrow & \downarrow \\ 0111 & 0101_{(excess-3)} \end{array}$$

結果 $42_{(10)} = 01110101_{(excess-3)}$

(2)

$$\begin{array}{ccccccc} 1 & 2 & . & 3 & 4 & 5 & \\ \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \\ 0001 & 0010 & & 0011 & 0100 & 0101 & _{(BCD)} \\ \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow & \\ 0100 & 0101 & & 0110 & 0111 & 1000 & _{(excess-3)} \end{array}$$

結果 $12.345_{(10)} = 01000101.011001111000_{(excess-3)}$

範例 2 將 $01101001.1100_{(excess-3)}$ 轉成十進位數

解

$$\begin{array}{cccc} 0110 & 1001 & . & 1100 \quad (excess-3) \\ \downarrow & \downarrow & & \downarrow \\ 0011 & 0110 & . & 1001 \quad (BCD) \\ \downarrow & \downarrow & & \downarrow \\ 3 & 6 & & 9 \quad (10) \end{array}$$

結果 $01101001.1100_{(excess-3)} = 369_{(10)}$

超三碼的特色爲本身具有補數的性質。十位數中 0～9 的超三碼，取 1 的補數，所得之超三碼代表的十進位數字，與原數相加必爲 9（兩者互爲 9 的補數）。例如：十進位 5，對 5 而言，將 5 轉成超三碼爲 1000，將 1000 取 1 的補數爲 0111，而 0111 爲十進位數 4 之超三碼

$5 + 4 = 9$

加三碼的功能：解決BCD碼不易轉換成補數的問題 (例如6－2)

(BCD碼運算)

$6-2 \rightarrow 6+(-2) \rightarrow 6+7$

9的補數不易轉換

```
  0110
+ 0111
-------
1 0011
+     1
-------
  0100
```

(加三碼運算)

6的加三碼＝1001

2的加三碼＝0101，其1的補數為1010（自補作用，即某數1的補數等於其9的補數，如本例2的9補數為7，加三碼為1010)

易轉換

```
  1001
+ 1010
-------
1 0011
+     1
-------
  0100
```

2-7-3 格雷碼(Gray Code)

格雷碼是相鄰的兩碼中，變化最少的一種碼，尤其是由一數目變化到下一相鄰的數目時，僅僅只有一個位元的不同；由於此種特性，故十分適合應用於一般之輸入／輸出檢誤和類比／數位轉換器(A/D converter)上，用以減少資料傳送及轉換時發生錯誤(因爲類比的資料常常爲連續性的資料)。

如表 2-8 所示爲 0 到 15 的格雷碼與其他數碼的對照表；由表中可看出相鄰的格雷碼，只有一個位元不同，如 $7_{(10)}$ 與 $8_{(10)}$ 兩數，其格雷碼分別爲 0100 及 1100，僅只有G_3不同；且表的上、下部份有如鏡子般反射，故又稱爲反射碼(reflected code)。另外，由於格雷碼非加權碼，所以並不適合作爲算術運算。

表 2-8　格雷碼與其他數碼的對照表

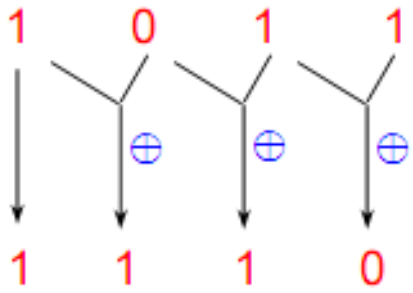
十進碼	二進碼				格雷碼			
	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

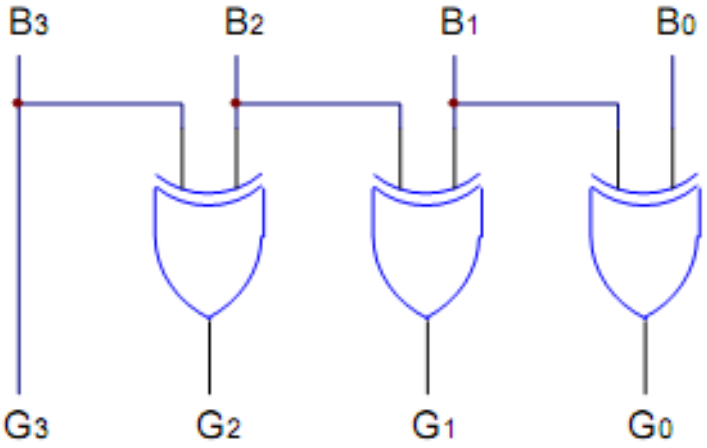
如鏡子般反射

四位元二進數&格雷碼

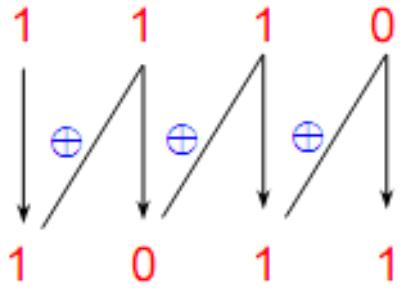
Binary	Gray Code	備　註
0 0 0 0	0 0 0 0	卡諾圖欄與列的標示即為格雷碼。 一個位元不同，利於偵錯。 在傳送連續值時，格雷碼的相鄰兩數僅
0 0 0 1	0 0 0 1	
0 0 1 0	0 0 1 1	
0 0 1 1	0 0 1 0	
0 1 0 0	0 1 1 0	
0 1 0 1	0 1 1 1	
0 1 1 0	0 1 0 1	
0 1 1 1	0 1 0 0	
1 0 0 0	1 1 0 0	
1 0 0 1	1 1 0 1	
1 0 1 0	1 1 1 1	
1 0 1 1	1 1 1 0	
1 1 0 0	1 0 1 0	
1 1 0 1	1 0 1 1	
1 1 1 0	1 0 0 1	
1 1 1 1	1 0 0 0	

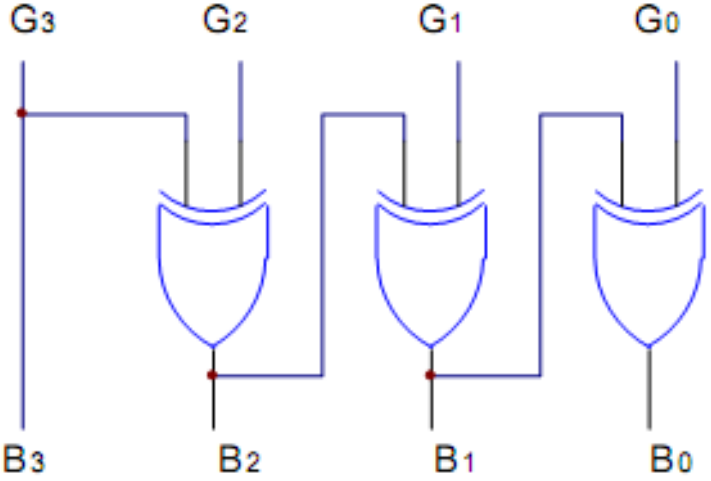
二進數轉換為格雷碼





格雷碼轉換為二進數





2-7-4 美國標準資訊交換碼(ASCII Code)

ASCII碼(American Standard Code for Information Interchange，美國標準資訊交換碼)，由7個位元的二進位碼所組成，共可代表128(2^7)種不同的符號，廣泛用於電腦及其輸入、輸出的週邊設備(如印表機、螢幕監視器、鍵盤等)上，作爲資料的傳遞與儲存。

$b_3b_2b_1b_0$	$b_6b_5b_4$							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	、	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LT	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	—	=	M	]	m	}
1110	SO	RS	·	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

第一部分由 00H 到 1FH 共 32 個，一般用來通訊或作為控制之用，有些字元可顯示於螢幕，有些則無法顯示在螢幕上，但能看到其效果(例如換行字元、歸位字元)。如下表：

ASCII 碼		字元	控制字元	意義	ASCII 碼		字元	控制字元	意義
十進位	十六進位				十進位	十六進位			
000	00		NULL	空字元	016	10	►	DLE	
001	01	☺	SOH		017	11	◄	DC1	
002	02	☻	STX		018	12	↕	DC2	
003	03	♥	ETX		019	13	‼	DC3	
004	04	♦	EOT		020	14	¶	DC4	
005	05	♣	ENQ		021	15	§	NAK	
006	06	♠	ACK		022	16	▬	SYN	
007	07	•	BELL	鈴聲	023	17	↨	ETB	
008	08	◘	BS	倒退鍵	024	18	↑	CAN	
009	09		HT	定位鍵	025	19	↓	EM	
010	0A		LF	line feed	026	1A	→	SUB	檔案結束
011	0B	♂	VT	home	027	1B	←	ESC	escape
012	0C	♀	FF	form feed	028	1C	L	FS	向右游標
013	0D		CR	carriage return	029	1D	↔	GS	向左游標
014	0E	♫	SO		030	1E	▲	RS	向上游標
015	0F	☼	SI		031	1F	▼	US	向下游標

第二部分是由 20H 到 7FH 共 96 個，這 95 個字元是用來表示阿拉伯數字、英文字母大小寫和底線、括號等符號，都可以顯示在螢幕上。如下表：

ASCII 碼		字元	ASCII 碼		字元	ASCII 碼		字元	ASCII 碼		字元
十進位	十六進位		十進位	十六進位		十進位	十六進位		十進位	十六進位	
032	20		056	38	8	080	50	P	104	68	h
033	21	!	057	39	9	081	51	Q	105	69	i
034	22	"	058	3A	:	082	52	R	106	6A	j
035	23	#	059	3B	;	083	53	S	107	6B	k
036	24	$	060	3C	<	084	54	T	108	6C	l
037	25	%	061	3D	=	085	55	U	109	6D	m
038	26	&	062	3E	>	086	56	V	110	6E	n
039	27	'	063	3F	?	087	57	W	111	6F	o
040	28	(	064	40	@	088	58	X	112	70	p
041	29	)	065	41	A	089	59	Y	113	71	q
042	2A	*	066	42	B	090	5A	Z	114	72	r
043	2B	+	067	43	C	091	5B	[	115	73	s
044	2C	,	068	44	D	092	5C	\	116	74	t
045	2D	-	069	45	E	093	5D	]	117	75	u
046	2E	.	070	46	F	094	5E	^	118	76	v
047	2F	/	071	47	G	095	5F	_	119	77	w
048	30	0	072	48	H	096	60	`	120	78	x
049	31	1	073	49	I	097	61	a	121	79	y
050	32	2	074	4A	J	098	62	b	122	7A	z
051	33	3	075	4B	K	099	63	c	123	7B	{
052	34	4	076	4C	L	100	64	d	124	7C	|
053	35	5	077	4D	M	101	65	e	125	7D	}
054	36	6	078	4E	N	102	66	f	126	7E	~
055	37	7	079	4F	O	103	67	g	127	7F	⌂

第三部分由 80H 到 0FFH 共 128 個字元，一般稱為「擴充字元」，這 128 個擴充字元是由 IBM 制定的，並非標準的 ASCII 碼。這些字元是用來表示框線、音標和其他歐洲非英語系的字母。

ASCII 碼		字元	ASCII 碼		字元	ASCII 碼		字元	ASCII 碼		字元
十進位	十六進位		十進位	十六進位		十進位	十六進位		十進位	十六進位	
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	∈
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	▒	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▓	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	▉	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

在電腦間傳送資料爲便利做長距離傳輸，可將ASCII增加第8個位元，做同位位元以便於偵測錯誤。

IBM PC 採用 ASCII-8 碼，增加許多文字、符號、圖案，因此共有 256 個碼。

同位位元偵錯法

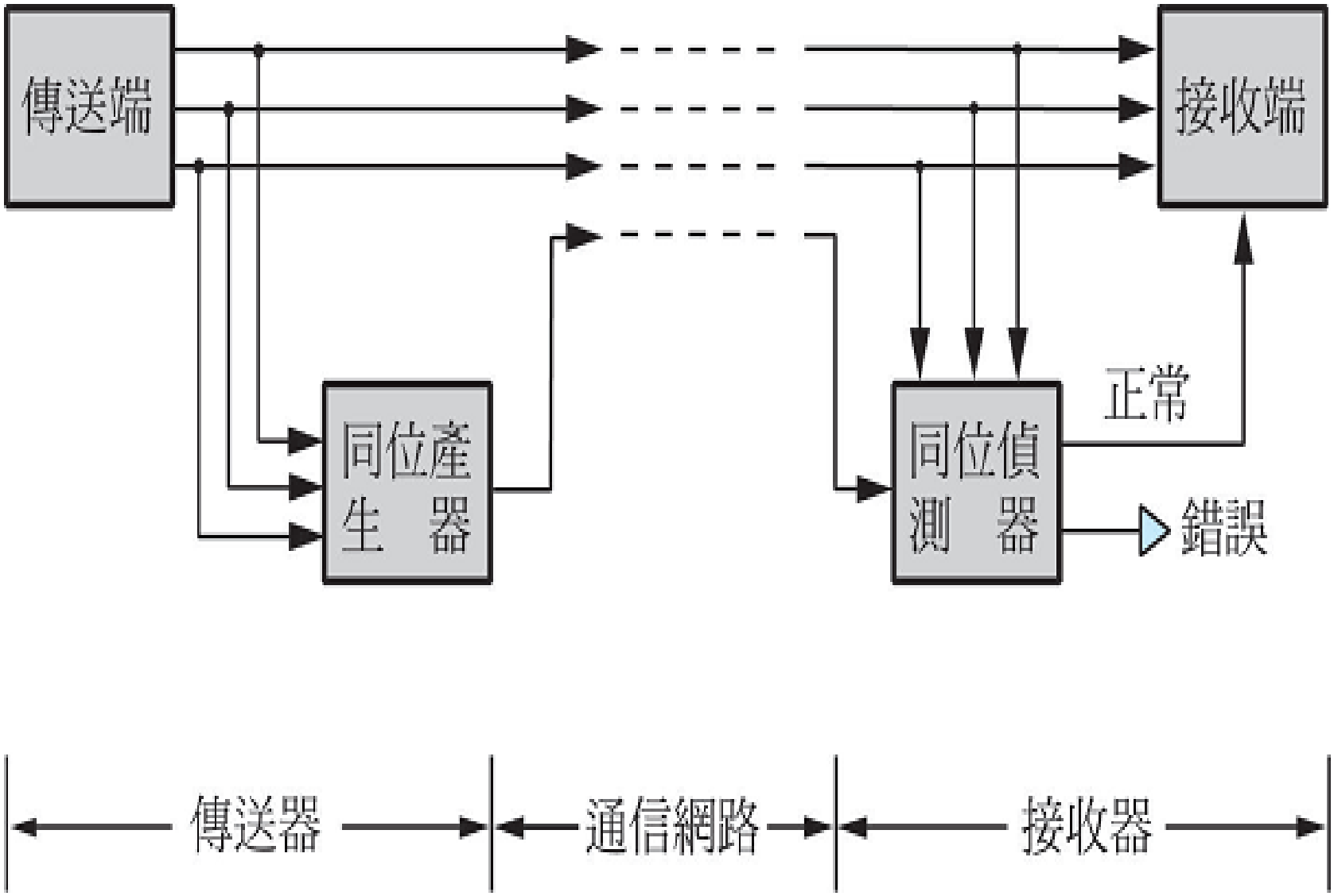
在數位系統資料傳送過程中，有許多雜訊因素會引發資料錯誤。一般偵錯誤方法採用同位位元（parity bit）方式。

同位位元有二種：

⑴偶同位（Even Parity）：增加 1 個位元（即同位位元），使其欲傳送資料中〝1〞的個數爲偶數。

⑵奇同位（Odd Parity）：增加 1 個位元，使其欲傳送資料中〝1〞的個數爲奇數。

同位位元傳送的系統方塊圖如圖2-2所示。



同位位元檢查法不能偵測偶數個位元的錯誤。

假使資料有偶數個位元錯誤，如下所示：

1 0 1 1 0 ←傳送端的資料

↓ ↓

1 1 1 0 0 ←接收端的資料

其中有兩個位元受到干擾而改變其值，但接收的資料仍含有奇數個1，因此不能偵測出錯誤。

同位位元偵錯方式，優點爲電路簡單，只需使用互斥或閘即可。

同位法在有兩個位元發生錯誤時是無法偵測的；換句話說，同位檢查方法只具有奇數個錯誤的檢測能力，而對偶數個錯誤則莫可奈何，不過在大多數的數位系統中，一次發生二個位元之錯誤機會是相當小的。