數位系統設計

課　程　簡　介

第一章　數字系統與數碼

第二章　布林代數與基本邏輯閘

第三章　布林代數的化簡

第四章　組合邏輯電路之分析與設計

第五章　組合邏輯電路設計─算術運算電路

第六章　組合邏輯電路設計─資料處理電路

第七章　基本正反器之認識

第八章　序向邏輯電路之設計

第九章　序向邏輯電路之設計—計數器

第十章　序向邏輯電路之設計—移位暫存器與隨機存取記憶器

第十一章　可規劃邏輯裝置

第十二章　數位系統積體化之實現方法

Chapter 1

數字系統與數碼

概　　述

- 邏輯(Logic)是一套人類分析與解決問題之思考方法，而針對這些思維發展出一套數學(Algebra)與數字系統(Number System)來表示，然後找出以一套較具規則性之步驟，以設計出硬體電路來實現人類之邏輯思考方法，最後形成一套完整之系統來處理人類的各種事務，這些硬體電路被稱為數位系統(Digital System)。
- 由於物理之限制，目前之數位系統僅能用來處理二元性(Binary)訊號(Signal)，為處理這些二元性訊號所發展之數學稱為交換代數(Switching Algebra)。
- 因二元性訊號為有限狀態之數字系統，故數位系統是設計來處理有限數字的資料系統，此系統具有相當可控制性(Controllable)，因此可得到相當精確之結果，故已成為目前實現硬體電路之主流。
- 數位電路僅能處理二元性數位碼(二進位數碼：Binary Numerical Code)，而人類卻習慣使用十進位碼(Decimal Numerical Code)，為有效且精確的處理這些數字系統之問題，必須能精確把這些數值表達出來，因此本章首先介紹二進碼與十進位碼之表示法後，再討論兩者之互相關係與各種算數運算方法。
- 因二進位數字系統，亦可用來處理非數值性之文數字碼(Alphanumeric Code)，本章亦會討論一些目前較常使用之文數字碼。

數字系統之表示法

◆ 在一基底(Base)為 b 之數字系統中，則該系統中之任一正數 N 皆可用下面之數列來定義：

$$N=\sum_{i=-n}^{m} a_i \cdot b^i$$

其中 a_i 為一常數。當基底 b 為 2 時，則為二進位數字系統，而基底 b 為 10 時，則為十進位數字系統。

◆ 在數字系統中，當基底 b 為 10 時，則該系統中之任一正數 N 定義為

$$N=\sum_{i=-n}^{m} a_i \cdot 10^i = a_m \times 10^m + \cdots + a_0 \times 10^0 + a_{-1} \times 10^{-1} + \cdots + a_{-n} \times 10^{-n}$$

上式之每個數字 a_i (亦可稱為位元；Bit)所在之位置，皆指出不同之倍率，而這些倍率即是所謂加權值(Weighting Value)，其中 a_m (最左邊之數字)稱為最高有效位元(Most Significant Bit；MSB)，a_{-n} (最右邊之數字)稱為最低有效位元(Least Significant Bit；LSB)，其中 a_i 之值可為 0 至 9 中之十個數字之任一個。

◆ 在數字系統中，當基底 b 為 2 時，則該系統中之任一正數 N 定義為

$$N = \sum_{i=-n}^{m} a_i \cdot 2^i = a_m \times 2^m + \cdots + a_0 \times 2^0 + a_{-1} \times 2^{-1} + \cdots + a_{-n} \times 2^{-n}$$

◆ 上式之每個數字 a_i (亦可稱為位元；Bit)所在之位置，皆指出不同之倍率，而這些倍率即是所謂加權值(Weighting Value)，其中 a_m (最左邊之數字)亦稱為最高有效位元(Most Significant Bit；MSB)，a_{-n} (最右邊之數字)稱為最低有效位元(Least Significant Bit；LSB)，其中 a_i 之值僅可為 0 與 1 兩個數字其中之一。

數字系統之轉換(十進位至二進位之轉換)

- 將十進位轉換成為二進位之數字系統，須分成整數(Round Number)與分數(Fraction)兩個部分來進行。
- 整數部分之轉換規則如下：

1. 將已知的十進位數之整數部份連除 2，並取出其餘數，直到所得之商數至 0 為止。
2. 接著由下至上依序取得所有餘數，即以所得之最後餘數為最高有效位元(MSB)，而最先取得之餘數為最低有效位元(LSB)，即為所求整數部分的二進位數字。

例題 1-3

試將 $N=67$ 之十進位數字，轉換成為二進位數字。

解

$67 \div 2 = 33 \longrightarrow$ **1** LSB

$33 \div 2 = 16 \longrightarrow$ **1**

$16 \div 2 = 8 \longrightarrow$ **0**

$8 \div 2 = 4 \longrightarrow$ **0**

$4 \div 2 = 2 \longrightarrow$ **0**

$2 \div 2 = 1 \longrightarrow$ **0**

$1 \div 2 = 0 \longrightarrow$ **1** MSB

由左邊之計算可得 $N=(67)_{10}=(1000011)_2$

數字系統之轉換(十進位至二進位之轉換)

◆ 分數部分之轉換規則如下：

1. 將已知的十進位數之分數部份乘 2，並取出所得之整數，此數值代表所求二進位之小數點右邊第一位數。

2. 若將整數去掉後之餘數不為零，則再重覆步驟(1)之計算，直到去掉整數後之餘數是零為止。

註：分數部分之轉換可能會造成相當長的計算程序，讀者可依實際所需求之精確度，在適當的時候終止計算過程即可。

例題 1-4

試將 $N = 0.6875$ 之十進位數字，轉換成為二進位數字。

解

$0.6875 \times 2 = 1.375 - 1 = 0.375 \longrightarrow$ **1**

$0.375 \times 2 = 0.75 \longrightarrow$ **0**

$0.75 \times 2 = 1.5 - 1 = 0.5 \longrightarrow$ **1**

$0.5 \times 2 = 1 - 1 = 0 \longrightarrow$ **1**

由左邊之計算可得 $N = (0.6875)_{10} = (0.1011)_2$

數字系統之轉換(二進位至十進位之轉換)

◆ 欲將二進位轉換成為十進位之數字系統，只要將所有係數 a_i 乘上所對應之位元所佔之權值(Weight)之和，即可用 $N=\sum_{i=-n}^{m} a_i \cdot 2^i = a_m \times 2^m + \cdots + a_0 \times 2^0 + a_{-1} \times 2^{-1} + \cdots + a_{-n} \times 2^{-n}$ 來表示。

例題 1-5

試將 $N=1101.101$ 之二進位數字，轉換成為十進位數字。

解

$$N = 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^3$$
$$= 8+4+1+0.5+0.125 = 13.625$$

由以上之算可得 $N=(1101.101)_2=(13.625)_{10}$

二進位數字系統之數值表示法

- 二進位之數值表示法，可分為未帶符號數(Unsigned Number)與帶符號數(Signed Number)等兩種。未帶符號數之數值表示法無正、負數之分別，一般皆將其數值視為正數；而帶符號數之數值表示法，則有正、負數的分別。
- 數位系統在處理算數運算時，必須有某種方法來表示正、負數，若採用硬體電路來區分正、負數，則可能會增加電路之複雜度，不合乎經濟效益。而數位系統在區分正、負數之方法，通常會保留二進位數之最高有效位元(MSB)，來當作符號位元(Sign Bit)，以「0」代表正數，「1」代表負數，而此位元僅用來儲存二進位數之正、負數。
- 二進位數字系統之正、負數的數值表示法，可分為真值式(True Magnitude)、1 的補數(1's Complement：基底減 1 的補數)與 2 的補數(2's Complement：基底的補數)等三種方法。

真值式表示法

◆ 二進位數字系統之真值式 (True Magnitude)表示法，亦可稱為正數表示法，此種數值表示法是以數值大小之絕對值的等效二進位數來表示，且保留二進位數之 MSB 當作符號位元。

例題 1-6

試將十進位數字之(+3)至(-3)轉換成為二進位之真值式表示法。

解

若以 4 個位元之二進位數來表示(含符號位元)，則可得結果如下表所示。

十進位數字	符號位元	二進位之數值
+3	0	011
+2	0	010
+1	0	001
+0	0	000
−0	1	000
−1	1	001
−2	1	010
−3	1	011

1 的補數表示法

- 二進位數字系統之 1 的補數(1's Complement)表示法，即符號位元保持不變，當所求之數值為「正數」時，則二進位數值與真值式表示法相同；而當所求之數值為「負數」時，則須將二進位數取 1 的補數來表示。
- 二進位數之 1 的補數可用 1 減去各個位元(Bit)即可獲得，因二進位數字僅有 0 與 1 兩個數字，故可簡化為僅須將每個位元之 0 變成 1 與 1 變成 0，即為所求。

例題 1-7

試將十進位數字之(+3)至(-3)轉換成為二進位之 1 的補數表示法。

解

若以 4 個位元之二進位數來表示(含符號位元)，則可得結果如下表所示。

十進位數字	符號位元	二進位之數值
＋3	0	011
＋2	0	010
＋1	0	001
+0	0	000
－0	1	111
－1	1	110
－2	1	101
－3	1	100

2 的補數表示法

- 二進位數字系統之 2 的補數(2's Complement)表示法，即符號位元保持不變，當所求之數值為「正數」時，則二進位數值與真值式表示法相同；而當所求之數值為「負數」時，則須將二進位數取 2 的補數來表示。
- 二進位數之 2 的補數等於 1 的補數再加上 2^{-m} (其中 m 為二進位數值之小數點位數)。

例題 1-8

試將十進位數字之(+3)至(-3)轉換成為二進位之 2 的補數表示法。

解

若以 4 個位元之二進位數來表示(含符號位元)，則可得結果如下表所示。

十進位數字	符號位元	二進位之數值
+3	0	011
+2	0	010
+1	0	001
+0	0	000
−0	1	000
−1	1	111
−2	1	110
−3	1	101

二進位之加法運算

- 二進位與十進位之加法運算規則完全相同，即是從最低有效位元(LSB)開始相加，並須考慮進位傳輸之問題，一直加到最高有效位元(MSB)為止。
- 兩個一位元之二進位數相加之四個基本規則如下：

$$
\begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array}
\qquad
\begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array}
\qquad
\begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array}
\qquad
\begin{array}{r} 1 \\ +\ 1 \\ \hline 10 \end{array}
$$

進位

- 考慮兩個 4 位元之二進位數，分別為被加數 $A = A_3A_2A_1A_0$ 與加數 $B = B_3B_2B_1B_0$，若欲對兩數進行加法運算 $(A + B)$，則可得計算規則如下：

$$
\begin{array}{cccccccl}
 & C_4 & C_3 & C_2 & C_1 & & \longleftarrow & \text{進位} \\
 & & A_3 & A_2 & A_1 & A_0 & \longleftarrow & \text{被加數} \\
+ & & B_3 & B_2 & B_1 & B_0 & \longleftarrow & \text{加數} \\
\hline
 & C_4 & S_3 & S_2 & S_1 & S_0 & \longleftarrow & \text{和}
\end{array}
\qquad
\begin{array}{cccccccl}
 & 1 & 0 & 0 & 1 & & \longleftarrow & \text{進位} \\
 & & 1 & 1 & 0 & 1 & \longleftarrow & \text{被加數} \\
+ & & 1 & 0 & 0 & 1 & \longleftarrow & \text{加數} \\
\hline
 & 1 & 0 & 1 & 1 & 0 & \longleftarrow & \text{和}
\end{array}
$$

$$A + B = 1101 + 1001 = 10110$$

二進制之減法運算

- 二進制與十進位之**減法運算規則**完全相同，即是從**最低有效位元 (LSB)** 開始相減，並須考慮**借位傳輸**之問題，一直減到最**高有效位元 (MSB)** 為止。
- 兩個一位元之二進位相減之四個**計算規則**如下：

$$
\begin{array}{r} 0 \\ -\ 0 \\ \hline 0 \end{array}
\qquad
\begin{array}{r} 0 \\ -\ 1 \\ \hline 11 \end{array}
\qquad
\begin{array}{r} 1 \\ -\ 0 \\ \hline 1 \end{array}
\qquad
\begin{array}{r} 1 \\ -\ 1 \\ \hline 0 \end{array}
$$

借位

- 考慮兩個 4 位元之二進位數分別為**被減數** $A = A_3A_2A_1A_0$ 與**減數** $C = C_3C_2C_1C_0$，若將兩數進行減法運算 $(A-C)$，則可計算規則如下：

$$
\begin{array}{ccccccl}
 & B_4 & B_3 & B_2 & B_1 & & \leftarrow \text{借位} \\
 & & A_3 & A_2 & A_1 & A_0 & \leftarrow \text{被減數} \\
- & & C_3 & C_2 & C_1 & C_0 & \leftarrow \text{減數} \\
\hline
 & B_4 & D_3 & D_2 & D_1 & D_0 & \leftarrow \text{差}
\end{array}
\qquad
\begin{array}{ccccccl}
 & 0 & 0 & 1 & 1 & & \leftarrow \text{借位} \\
 & & 1 & 1 & 0 & 0 & \leftarrow \text{被減數} \\
- & & 1 & 0 & 0 & 1 & \leftarrow \text{減數} \\
\hline
 & 0 & 0 & 0 & 1 & 1 & \leftarrow \text{差}
\end{array}
$$

$$\boldsymbol{A - C = 1100 - 1001 = 00011}$$

二進制之減法運算(使用 1 的補數)

◆ 為了降低硬體電路之複雜性，本節將引入補數之關念，利用加法來代替減法之算數運算。而引用補數之觀念，以加法來替代減法之算數運算，可分為「基底減 1 之補數」(1 的補數)與「基底的補數」(2 的補數)兩種方法。

◆ 1 的補數減法運算：

若欲對兩個二進位數，分別為被減數 A 與減數 B 進行減法運算，即進行 $A-B$ 之算數運算，若採用 1 的補數之方法，則其運算規則如下：

1. 將減數(B)取 1 的補數後，再與被減數作加法運算，即 $A-B=A+B_{1'S}$。
2. 當將兩數依二進位數之加法運算規則相加後，若有端迴進位 (End-Around Carry) 產生時，則表示所得之結果是正數。接著將兩數相加後之結果，再加上端迴進位「1」後，即為所求之答案。
3. 當將兩數依二進位數之加法運算規則相加後，若無端迴進位產生時，則表示所得之結果是負數。接著將兩數相加後之結果，再取 1 的補數後加上負號後，即為所求之答案。

例題 1-11

請將下列十進位數分別轉換成為二進位數後，再使用 1 的補數來執行 (a) 62 – 5與 (b) 30 – 60之二進位數減法運算。

解

(a)、62 – 5

$$
\begin{array}{l}
62 = 1\ 1\ 1\ 1\ 1\ 0 \\
5\ = 0\ 0\ 0\ 1\ 0\ 1 \xrightarrow{\text{1's}} 1\ 1\ 1\ 0\ 1\ 0
\end{array}
$$

$$
\begin{array}{rr}
 & 1\ 1\ 1\ 1\ 1\ 0 \\
+ & 1\ 1\ 1\ 0\ 1\ 0 \\
\hline
 & \cancel{1}\ 1\ 1\ 1\ 0\ 0\ 0 \\
+ & 1 \\
\hline
 & 1\ 1\ 1\ 0\ 0\ 1
\end{array}
$$

端迴進位

由左邊之計算結果可得

$$(62)_{10} - (5)_{10} = (57)_{10} = (111001)_2$$

(b)、30 – 60

$$
\begin{array}{l}
30 = 0\ 1\ 1\ 1\ 1\ 0 \\
60 = 1\ 1\ 1\ 1\ 0\ 0 \xrightarrow{\text{1's}} 0\ 0\ 0\ 0\ 1\ 1
\end{array}
$$

$$
\begin{array}{rr}
 & 0\ 1\ 1\ 1\ 1\ 0 \\
+ & 0\ 0\ 0\ 0\ 1\ 1 \\
\hline
 & 1\ 0\ 0\ 0\ 0\ 1
\end{array}
\xrightarrow{\text{1's}} -0\ 1\ 1\ 1\ 1\ 0
$$

由左邊之計算結果可得

$$(30)_{10} - (60)_{10} = (-30)_{10} = (-011110)_2$$

二進制之減法運算(使用 2 的補數)

◆ 若欲對兩個二進位數，分別為被減數 A 與減數 B 進行減法運算，即進行 $A-B$ 之算數運算，若採用 2 的補數之方法，則其運算規則如下：

1. 將減數 (B) 取 2 的補數後，再與被減數作加法運算，即 $A-B=A+B_{2'S}$ 。
2. 當將兩數依二進位數之加法運算規則相加後，若有端迴進位 (End-Around Carry) 產生時，則表示兩數相加後之結果為正數。接著將所得之端迴進位捨去後，即為所求之答案。
3. 當將兩數依二進位數之加法運算規則相加後，若無端迴進位產生時，則表示所得之結果是負數。接著將兩數相加後之結果，再取 2 的補數後加上負號後，即為所求之答案。

例題 1-12

請將下列十進位數分別轉換成為二進位數後，再使用 2 的補數來執行 **(a)** 62 – 5 與 **(b)** 30 – 60 之二進位數減法運算。

解

(a)、62 – 5

$$62 = 1\ 1\ 1\ 1\ 1\ 0$$

$$5 = 0\ 0\ 0\ 1\ 0\ 1 \xrightarrow{1's} 1\ 1\ 1\ 0\ 1\ 0 \xrightarrow{2's} 1\ 1\ 1\ 0\ 1\ 1$$

$$\begin{array}{rr} & 1\ 1\ 1\ 1\ 1\ 0 \\ + & 1\ 1\ 1\ 0\ 1\ 1 \\ \hline & \cancel{1}\ 1\ 1\ 1\ 0\ 0\ 1 \end{array}$$

由左邊之計算結果可得

$$(62)_{10} - (5)_{10} = (57)_{10} = (111001)_2$$

(b)、30 – 60

$$30 = 0\ 1\ 1\ 1\ 1\ 0$$

$$60 = 1\ 1\ 1\ 1\ 0\ 0 \xrightarrow{1's} 0\ 0\ 0\ 0\ 1\ 1 \xrightarrow{2's} 0\ 0\ 0\ 1\ 0\ 0$$

$$\begin{array}{rr} & 0\ 1\ 1\ 1\ 1\ 0 \\ + & 0\ 0\ 0\ 1\ 0\ 0 \\ \hline & 1\ 0\ 0\ 0\ 1\ 0 \end{array} \xrightarrow{2's} -0\ 1\ 1\ 1\ 1\ 0$$

由左邊之計算結果可得

$$(30)_{10} - (60)_{10} = (-30)_{10} = (-011110)_2$$

使用 1 的補數與 2 的補數進行減法算數運算之比較

- 使用 1 的補數與 2 的補數來進行二進位之減法運算法之優、缺點列述於下：

1. 只須將減數之所有位元分別取反相，因此較容易採用數位電路來實現，此為 1 的補數減法運算之優點；然在實現減法電路時，若被減數與減數的 1 的補數相加後，若有端迴進位產生時，須進行二次加法運算，徒增硬體電路之複雜性，此為 1 的補數減法運算之缺點。

2. 雖然對減數取 2 的補數較為複雜，但實現減法電路時，不須進行二次加法運算，為使用 2 的補數執行減法運算電路之最大優點。

- 綜合上面之討論可知，1 的補數相當於邏輯之反運算時，實現此邏輯之硬體電路較為簡單，故應於非算數運算之電路較佳；而採用 2 的補數進行減法運算時，不須做二次加法運算，故常應用於算數運算方面。

二進制之乘法運算

◆ 二進制與十進位之乘法運算規則完全相同，因乘數僅有「0」與「1」兩個數值，故其計算過程較十進位簡單。而對兩個一位元之二進位相乘之四個基本規則如下：

$$\begin{array}{r}0\\ \times\ 0\\ \hline 0\end{array}\qquad\begin{array}{r}0\\ \times\ 1\\ \hline 0\end{array}\qquad\begin{array}{r}1\\ \times\ 0\\ \hline 0\end{array}\qquad\begin{array}{r}1\\ \times\ 1\\ \hline 1\end{array}$$

◆ 考慮被乘數 $A = A_2A_1A_0$ 與乘數 $B = B_1B_0$ 分別為 3 位元與 2 位元之二進位數，若將兩數進行乘法運算 $(A \times B)$，則計算方法可表示如下：

$$\begin{array}{rrrrr} & & A_2 & A_1 & A_0 \\ \times & & & B_1 & B_0 \\ \hline & & A_2B_0 & A_1B_0 & A_0B_0 \\ & A_2B_1 & A_1B_1 & A_0B_1 & \\ + C_2 & C_1 & C_0 & & \\ \hline F_4 & F_3 & F_2 & F_1 & F_0 \end{array}$$

其中

$$F_0 = A_0 \cdot B_0$$
$$F_1 = A_1 \cdot B_0 + A_0 \cdot B_1$$
$$F_2 = A_2 \cdot B_0 + A_1 \cdot B_1 + C_0$$
$$F_3 = A_2 \cdot B_1 + C_1$$
$$F_4 = C_2$$

$$\begin{array}{rrrrr} & & 1 & 1 & 0 \\ \times & & & 1 & 1 \\ \hline & & 1 & 1 & 0 \\ + & 1 & 1 & 0 & \\ \hline 1 & 0 & 0 & 1 & 0 \end{array}$$

$$A \times B = 110 \times 11 = 10010$$

註：上面運算式中之「+」為算數運算中之加法運算，而非邏輯運算之「OR」運算。

二進制之除法運算

◆ 二進制與十進位之除法運算規則完全相同，通常是以長除法來進行運算，因除數僅有「0」與「1」兩個數值，故其計算過程亦較十進位數簡單。

◆ 考慮兩個二進位數分別為被除數 $A = 1111$ 與除數 $B = 11$ 相除之計算方法如下：

$$
\begin{array}{rr}
 & 1\ 0\ 1 \\ \cline{2-2}
1\ 1 & \big)\,1\ 1\ 1\ 1 \\
 & 1\ 1 \\ \cline{2-2}
 & 1\ 1 \\
 & 1\ 1 \\ \cline{2-2}
 & 0
\end{array}
$$

$$A \div B = 1111 \div 11 = 101$$

二進位數十進位碼

- 為了方便二進位數與十進位數之轉換，通常以 4 個位元之二進位數，代表一個等值之十進位數，以簡化二進位數與十進位數之轉換程序，而此種數碼被稱為二進位數十進位碼 (Binary-Code-Decimal Code; BCD 碼)。
- BCD 碼僅使用 0000 至 1001 等 10 個二進位數碼，分別來表示十進位數之 0 至 9 等十個數目字，如下表所示。

BCD 數碼	十進位數字
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

例題 1-15

試將 (a) $N = 63$ 與 (b) $N = 968$ 之十進位數字轉換成為 BCD 碼。

解

	十進位數	BCD 碼
(a)	63	0110 0011
(b)	968	1001 0110 1000

BCD 碼之加法運算

◆ BCD 碼之加法運算規則與二進位數相似，因每 1 個位元之十進位數皆是由 4 個位元之二進位數碼所組成的，因此在進行 BCD 碼之加法運算時，必須以 4 個位元為一組來進行加法運算與處理進位之問題。接著列出進行 BCD 碼加法運算時，須遵守之規則如下：

1. 將兩個欲相加之 BCD 碼，以 4 個位元為 1 組分別進行加法運算。

2. 若兩組 4 位元之和小於或等於 1001(9)，且無端迴進位產生時，則所得之和為有效之 BCD 碼，故兩數相加後之結果即為所求。

3. 若兩組 4 位元之和大於 1001 或有端迴進位產生時，則所得之和為無效 BCD 碼，此時必須再將所得之和，再加上 0110，以跳過 6 個沒有用之二進位數碼，使其成為有效之 BCD 碼，並將進位加至上一組 BCD 碼之最低有效位數。

例題 1-16

試將兩個十進位數分別為被加數 $A = 63$ 與加數 $B = 37$，轉換成為 BCD 碼，並對兩個 BCD 碼進行加法運算。

解

$$A = 63 \xrightarrow{\text{BCD碼}} 0110\ 0011 \qquad B = 37 \xrightarrow{\text{BCD碼}} 0011\ 0111$$

$$\begin{array}{rr} & 63 \\ + & 37 \\ \hline & 100 \end{array}$$

$$\begin{array}{cccccccccc} & & & & & 1 & & & & \\ & & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ + & & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ \hline & & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ + & & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ \hline & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$A + B = (63)_{10} + (37)_{10} = \underline{0110}\ \underline{0011} + \underline{0011}\ \underline{0111} = \underline{1}\ \underline{0000}\ \underline{0000} = (100)_{10}$$

BCD 碼之減法運算

- BCD 碼之減法運算規則與二進位數相似，為了降低硬體電路之複雜度，亦可引用補數之觀念，以加法來替代減法運算。
- 引用補數之觀念，以加法來替代減法之算數運算方法，可分為「基底減 1 之補數(9 的補數)」與「基底的補數(10 的補數)」等兩種方法。
- 十進位數字系統中，在求 9 的補數時，只需分別用 9 減去所求之各個位數之十進位數，即為所求。

$(7)_{10} \xrightarrow{9's} (9-7)_{10} = (2)_{10} = \underline{0010}$ ， $(36)_{10} \xrightarrow{9's} (99-36)_{10} = (63)_{10} = \underline{0110}\ \underline{0011}$ ，

$(568)_{10} \xrightarrow{9's} (999-568)_{10} = (431)_{10} = \underline{0100}\ \underline{0011}\ \underline{0001}$

- 在十進位數字系統中，欲求 10 的補數時，則須先求該數之 9 的補數後，再加上10^{-m} (其中 m 為所求十進位數之小數點位數)，即為所求。若所求之數字為整數時，則求取 10 的補數可簡化為$10's = 9's + 1$。

$(7)_{10} \xrightarrow{10's} (9-7+10^0)_{10} = (3)_{10} = \underline{0011}$ ， $(36)_{10} \xrightarrow{10's} (99-36+10^0)_{10} = (64)_{10} = \underline{0110}\ \underline{0100}$

$(568)_{10} \xrightarrow{10's} (999-568+10^0)_{10} = (432)_{10} = \underline{0100}\ \underline{0011}\ \underline{0010}$

9 的補數減法運算

◆ 引用 9 的補數來進行 BCD 碼之減法運算，對兩個 BCD 碼，分別為被減數 A 與減數 B 進行減法運算，即進行 $A-B$ 之算數運算。若採用 9 的補數之方法，則其運算規則如下：

1. 將減數 (B) 取 9 的補數後，再與被減數進行 BCD 碼之加法運算，即 $A-B=A+B_{9's}$。
2. 若將兩數依 BCD 碼之加法運算規則相加後，有端迴進位 (End-Around Carry) 產生時，表示所得之結果是正數，則加上端迴進位 (1) 後，即為所求。
3. 若將兩數依 BCD 碼之加法運算規則相加後，無端迴進位產生時，表示所得之結果是負數，則將所得之結果取 9 的補數，再加上負號後，即為所求。

例題 1-18

試使用 9 的補數來執行(a) 62 – 5 與(b) 30 – 60 之十進位數算數運算，並將 結果用 BCD 碼表示。

解

(a)

$$
\begin{array}{rcl}
62 & = & 0\ 1\ 1\ 0 \quad 0\ 0\ 1\ 0 \\
5 & = & 0\ 0\ 0\ 0 \quad 0\ 1\ 0\ 1 \xrightarrow{9's} 1\ 0\ 0\ 1 \quad 0\ 1\ 0\ 0
\end{array}
$$

$$
\begin{array}{rr}
 & 0\ 1\ 1\ 0 \quad 0\ 0\ 1\ 0 \\
+ & 1\ 0\ 0\ 1 \quad 0\ 1\ 0\ 0 \\
\hline
 & \cancel{1}\ \ 1\ 1\ 1\ 1 \quad 0\ 1\ 1\ 0 \\
+ & 0\ 1\ 1\ 0 \quad\ \ +\ 1 \\
\hline
 & 0\ 1\ 0\ 1 \quad 0\ 1\ 1\ 1
\end{array}
$$

端迴進位

$$(62)_{10}-(5)_{10}=(57)_{10}=\underline{0101}\,\underline{0111}$$

(b)

$$
\begin{array}{rcl}
30 & = & 0\ 0\ 1\ 1 \quad 0\ 0\ 0\ 0 \\
60 & = & 0\ 1\ 1\ 0 \quad 0\ 0\ 0\ 0 \xrightarrow{9's} 0\ 0\ 1\ 1 \quad 1\ 0\ 0\ 1
\end{array}
$$

$$
\begin{array}{rr}
 & 0\ 0\ 1\ 1 \quad 0\ 0\ 0\ 0 \\
+ & 0\ 0\ 1\ 1 \quad 1\ 0\ 0\ 1 \\
\hline
 & 0\ 1\ 1\ 0 \quad 1\ 0\ 0\ 1 \xrightarrow{9's} -0\ 0\ 1\ 1 \quad 0\ 0\ 0\ 0
\end{array}
$$

$$(30)_{10}-(60)_{10}=(-30)_{10}=-\underline{0011}\ \underline{0000}$$

10 的補數減法運算

◆ 引用 10 的補數來進行 BCD 碼之減法運算，對兩個 BCD 碼，分別為被減數 A 與減數 B 進行減法運算，即進行 $A-B$ 之算數運算。若採用 10 的補數之方法，則其運算規則如下：

1. 將減數 (B) 取 10 的補數後，再與被減數進行 BCD 碼之加法運算，即 $A-B=A+B_{10's}$。
2. 若將兩數依 BCD 碼之加法運算規則相加後，有端迴進位 (End-Around Carry) 產生時，表示所得之結果是正數，則去掉端迴進位，即為所求。
3. 若將兩數依 BCD 碼之加法運算規則相加後，無端迴進位產生時，表示所得之結果是負數，則將所得之結果取 10 的補數，再加上負號後，即為所求。

例題 1-20

試使用 10 的補數來執行(a) 62 – 5與(b) 30 – 60之十進位數算數運算，並將 結果用 BCD 碼表示。

解

(a)

$$\begin{array}{rcl} 62 & = & 0\,1\,1\,0 \quad 0\,0\,1\,0 \\ 5 & = & 0\,0\,0\,0 \quad 0\,1\,0\,1 \xrightarrow{10's} 1\,0\,0\,1 \quad 0\,1\,0\,1 \end{array}$$

$$\begin{array}{rr} & 0\,1\,1\,0 \quad 0\,0\,1\,0 \\ + & 1\,0\,0\,1 \quad 0\,1\,0\,1 \\ \hline & 1\,1\,1\,1 \quad 0\,1\,1\,1 \\ + & 0\,1\,1\,0 \qquad\qquad \\ \hline \cancel{1} & 0\,1\,0\,1 \quad 0\,1\,1\,1 \end{array}$$

$$(62)_{10} - (5)_{10} = (57)_{10} = \underline{0101}\ \underline{0111}$$

(b)

$$\begin{array}{rcl} 30 & = & 0\,0\,1\,1 \quad 0\,0\,0\,0 \\ 60 & = & 0\,1\,1\,0 \quad 0\,0\,0\,0 \xrightarrow{10's} 0\,1\,0\,0 \quad 0\,0\,0\,0 \end{array}$$

$$\begin{array}{rr} & 0\,0\,1\,1 \quad 0\,0\,0\,0 \\ + & 0\,1\,0\,0 \quad 0\,0\,0\,0 \\ \hline & 0\,1\,1\,1 \quad 0\,0\,0\,0 \xrightarrow{10's} -0\,0\,1\,1 \quad 0\,0\,0\,0 \end{array}$$

$$(30)_{10} - (60)_{10} = (-30)_{10} = -\underline{0011}\ \underline{0000}$$

二進位數之文字碼(葛雷碼)

◆ 數位系統不僅可用來處理數值之算數運算外，亦可用來處理非數值之文數字碼 (Alphanumeric Code) 系統，即非加權性 (Unweighted) 數碼。

◆ 葛雷碼是一種非加權性之二進位數碼，故此種數碼不適用於二進位數之算數運算，但普遍使用於類比 (Analog) 至數位 (Digital) 之轉換方面。

◆ 葛雷碼是屬於一種改變最少之二進位數碼，即葛雷碼之最大特色為上一數碼與下一數碼間，僅允許有一個位元改變，而當依序增加到它的最大值時，葛雷碼會使進位歸零，故此種數碼之零值和最大值，亦僅允許有一個位元變化（ 即零與最大值亦是相鄰的 ），故此種數碼又可被稱為循環碼 (Cyclic Code)。

◆ 葛雷碼與二進位碼之轉換關係，即將二進位碼 B_i 轉換成為葛雷碼 G_i 之規則為

$$G_i = B_{i+1} \oplus B_i \qquad 0 \le i \le n-2 \qquad 、 \qquad G_{n-1} = B_{n-1}$$

◆ 將二進位數 $B = B_5B_4B_3B_2B_1B_0 = 101110$ 轉換成為葛雷碼

$$G_5 = B_5 = 1 \qquad G_4 = B_5 \oplus B_4 = 1 \oplus 0 = 1 \qquad G_3 = B_4 \oplus B_3 = 0 \oplus 1 = 1$$

$$G_2 = B_3 \oplus B_2 = 1 \oplus 1 = 0 \qquad G_1 = B_2 \oplus B_1 = 1 \oplus 1 = 0 \qquad G_0 = B_1 \oplus B_0 = 1 \oplus 0 = 1$$

二進位數 101110 之相對應葛雷碼為 $G = G_5G_4G_3G_2G_1G_0 = 111001$。

四位元之二進位數碼與十進位數碼、葛雷碼之對應表

十進位數碼	二進位數碼	葛雷碼	十進位數碼	二進位數碼	葛雷碼
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

- 葛雷碼亦是循環碼中最常被使用的一種，因這種數碼具有反射對稱之特性，即一位元之葛雷碼僅為 0 與 1。兩位元以上葛雷碼之建立，可在中間部分畫一條分隔線，且以此分隔線為一鏡面。
- 若將另外之數碼對稱的寫下來，即原來之數碼前加 0，而新的數碼加 1，便可輕易得到任何位元數之葛雷碼，如下表所示。

利用反射對稱之原理所建立之 1 至 4 位元之葛雷碼

順序	一位元	兩位元	三位元	四位元
0	0	00	000	0000
1	1	01	001	0001
2		11	011	0011
3		10	010	0010
4			110	0110
5			111	0111
6			101	0101
7			100	0100
8				1100
9				1101
10				1111
11				1110
12				1010
13				1011
14				1001
15				1000

反射面

二進位數之文字碼(ASCII 碼)

◆ 一種最通用之文數字碼稱為 ASCII（ 美國資料交換標準碼 ），此種數碼是用 7 個位元之二進位數來編碼，以組成 128 種不同之符號，分別代表 94 個圖形字元(包括 26 個英文大寫、26 個英文小寫字母、10 個十進位數字與 32 個常用之符號等 ）與 34 個可作為各種控制功能用之字元，如下表所示。

低位元組	高 位 元 組 $A_6A_5A_4$							
$A_3A_2A_1A_0$	000	001	010	011	100	101	110	111
0000	NUL	DEL	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	LF	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

二進位數之文字碼(EBCDIC 碼)

- 另一種常用於 IBM 機器之文數字碼稱為 EBCDIC (Extended BCD Interchange Code)，又可翻譯為擴充式二進位編碼十進位數交換碼，此種文數字碼是用 8 個位元之二進位數來編碼。此種數碼後 4 個位元之數字範圍與 BCD 碼相同，即 0～9 等十個文字之 EBCDIC 碼是在 BCD 碼前加 1111 而形成的，如下表所示。

L \ H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL		DS		SP	&	−									0
1			SOS				/		a	j			A	J		1
2			FS						b	k	s		B	K	S	2
3		TM							c	l	t		C	L	T	3
4	PF	RES	BYP	PN					d	m	u		D	M	U	4
5	HT	NL	LF	RS					e	n	v		E	N	V	5
6	LC	BS	EOB	UC					f	o	w		F	O	W	6
7	DL		PRE	EOT					g	p	x		G	P	X	7
8									h	q	y		H	Q	Y	8
9									i	r	z		I	R	Z	9
A		CC	SM		Ç	!		:								
B						$	,	#								
C					<	*	%	@								
D					(	)	−	‘								
E					+	;	>	=								
F	CU1	CU2	CU3		|		?	“								

問題討論

1、將十進位數之 (+6) 至 (–6) 分別利用二進位數碼之真值數、1 的補數與 2 的補數表示法出來。

解：

利用 4 位元之二進位數來表示十進位數之 (+6) 至 (–6) 之**真值數**、**1 的補數**與 **2 的補數**，而使用**最左邊之位元(MSB)來表示符號位元**，當符號位元為「0」時，表示正數；而符號位元為「1」時，表示負數。

十進位數字	真值數	1 的補數	2 的補數
+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	0111	0000
–0	1000	1111	1000
–1	1001	1110	1111
–2	1010	1101	1110
–3	1011	1100	1101
–4	1100	1011	1100
–5	1101	1010	1011
–6	1110	1001	1010

2、討論使用 1 的補數與 2 的補數來進行二進位之減法運算之優、缺點與適用時機。

解：

1. 只須將減數之所有位元分別取反相（反運算），因此較容易採用數位電路來實現，此為 1 的補數減法運算之優點；然在實現減法電路時，若被減數與減數的 1 的補數相加後，若有端迴進位產生時，須進行二次加法運算，徒增硬體電路之複雜度，此為 1 的補數減法運算之缺點。

2. 雖然對減數取 2 的補數較為複雜，但實現減法電路時，不須進行二次加法運算，為此種減法運算電路之最大優點。

綜合上面之討論可得，1 的補數相當於邏輯之反運算，實現此邏輯運算之硬體電路較為簡單，故使用於非算數運算電路較佳；而採用 2 的補數進行減法運算時，不須做二次加法運算，故常應用於實現算數運算電路。

3、討論葛雷碼之主要特性與應用。

解：

葛雷碼是一種非加權性之二進位數碼，因此此種數碼不適用於二進位數之算數運算，但普遍使用於類比 (Analog) 至數位 (Digital) 之轉換方面。它是屬於一種改變最少之二進位數碼，即葛雷碼之最大特色為上一數碼與下一數碼間，僅允許有一個位元改變，而當依序增加到它的最大值時，葛雷碼會使進位歸零，即此種數碼之零值和最大值，亦僅允許有一個位元變化（即零與最大值亦是相鄰的），故此種數碼又可被稱為循環碼 (Cyclic Code)。

4、將(a) 11010 與(b) 101101 等二進位數碼轉換成為葛雷碼。

解：

(a) 二進位數 11010 之位元數為 5，可得相對應之葛雷碼為

$$G_4 = B_4 = 1$$

$$G_3 = B_4 \oplus B_3 = 1 \oplus 1 = 0$$

$$G_2 = B_3 \oplus B_2 = 1 \oplus 0 = 1$$

$$G_1 = B_2 \oplus B_1 = 0 \oplus 1 = 1$$

$$G_0 = B_1 \oplus B_0 = 1 \oplus 0 = 1$$

由以上之計算可得 11010 之相對應葛雷碼為 $G = G_4G_3G_2G_1G_0 = 10111$ 。

(b)二進位數 101101 之位元數為 6，可得相對應之葛雷碼為

$$G_5 = B_5 = 1$$

$$G_4 = B_5 \oplus B_4 = 1 \oplus 0 = 1$$

$$G_3 = B_4 \oplus B_3 = 0 \oplus 1 = 1$$

$$G_2 = B_3 \oplus B_2 = 1 \oplus 1 = 0$$

$$G_1 = B_2 \oplus B_1 = 1 \oplus 0 = 1$$

$$G_0 = B_1 \oplus B_0 = 0 \oplus 1 = 1$$

由以上之計算可得 101101 之相對應葛雷碼為 $G = G_5G_4G_3G_2G_1G_0 = 111011$。

5、討論通用非加權性之 ASCII 文字數碼的主要內容。

解：

最通用之文數字碼稱為 ASCII（美國資料交換標準碼），此種數碼用 7 個位元之二進位數來編碼，以組成 128 種不同之符號，分別代表 94 個圖形字元（包括 26 個英文大寫、26 個英文小寫字母、10 個十進位數字與 32 個常用之符號等）與 34 個可作為各種控制功能用之字元。

6、將(a) Electrical Engineering 與(b) Logic Design 等文字碼轉為 ASCII 碼。

解：

(a) Electrical Engineering

1000101 1101100 1100101 1100011 1110100 1110010 1101001 1100011 1100001
1101100 0100000 1000101 1101110 1100111 1101001 1101110 110 0101 110 0101
1110010 1101001 1101110 1100111

(b) Logic Design

1001100 1101111 1100111 1101001 1100011 0100001 1000100 1100101 1110011
1101001 1101110 1100111