Magawin 整數+小數減法器

電機甲 2A　1070557　古心妤

簽署本報告所有內容為本人親自撰寫，並無抄襲有違學術倫理之行為，所有文責自負。

姓名：　古心妤　（請打字輸入，以示負責）

一、 專題目的（200 字以上）

此次專題的目的為將這一學期所學，學以致用並應用 Magawin 上，驗收這一學期所學，我們學過如何取得按鍵碼、如何做 LED 輸出顯示以及 Delay 副程式的撰寫……等等，藉由學號分配每個人的鍵盤按鍵應為何，因按鍵並非像之前上課練習時一樣按照順序，因此在撰寫上會需要格外小心注意。

在減法器的製作上，最需要注意的地方就是借位的問題，因為借位再加上 code 的撰寫大部分為 16 進制，需要在另外換算成十進制，也就類似我們日常生活中計算機的功能，而數值大小的判斷也是一個重點，當輸入兩數值相減為負數時最右的小 LED 燈會亮起，在 LED 上會顯示最低位的 4 位數。

二、 詳細設計原理、記憶體定義、與完整流程圖（相關內容必須與下一節之完整程式相契合）

詳細設計原理：

減法器最重要的第一步就是要能夠判斷輸入兩數值的大小，才能以大減小進行下一步運算，在判斷大小的地方我用的是數小數點前有幾位來判斷，而儲存數字時我是對齊左邊，若小數點前擁有的位數一樣多時，則會從最高位開始相減，若相減出來 high byte 出現 F，則表第二個輸入數值較大。

若假設 20.1-30.5，我的記憶體會是這樣儲存(紅字為記憶體初始化的部分)：

First_num	4Eh	4Fh	50h	51h
	2	**0**	**1**	0
Sec_num	52h	53h	54h	55h
	3	**0**	**5**	0

↳ 因小數點前位數相同，因此需要用相減判斷大小，4Eh-52h=FFh，因此 First_num 較大

在得知哪個數值較大後，Which_Big 會記住是 1 還是 2，之後進入 Adjust 副程式進行數值對齊動作，我分別定義了各占 8 byte 的 Caculate1 與 Caculate2 去記對齊後的數值，會利用 8 byte 的原因是因為我規劃他們最左邊開始為千位、百位、十位、個位、10^(-1) 、10^(-2) 、10^(-3) 、10^(-4) ，因此才會定義 8 byte，藉由先前的比大小，我知道兩數值分別在小數點前有幾位，因此能知道該從哪裡開始存，例如：小數點前有 3 位數，代表從百位、十位、個位、10^(-1)依序往低位存，有因為我們最少小數點前一定有一位數(個位數 Ex：0.123) ，因此我們小數點前只會出現 1~4 位數，下頁表格接續說明。

舉 First_Num 與 Caculate1 所對齊之地址為例：

小數點前位數	First_num	4Eh	4Fh	50h	51h
4 位	Caculate1	58h	59h	5Ah	5Bh
3 位		59h	5Ah	5Bh	5Ch
2 位		5Ah	5Bh	5Ch	5Dh
1 位		5Bh	5Ch	5Dh	5Eh

若假設 999-0.1 對齊後(紅字為記憶體初始化的部分)：

	千位	百位	十位	個位	10^(-1)	10^(-2)	10^(-3)	10^(-4)
Caculate1	58h	59h	5Ah	5Bh	5Ch	5Dh	5Eh	5Fh
	0	**9**	**9**	**9**	0	0	0	0
Caculate2	60h	61h	62h	63h	64h	65h	66h	67h
	0	0	0	**0**	**1**	0	0	0

↳ 此表為數值對齊後

經過副程式 Adjust 對齊後，即可呼叫副程式 Subtraction 做相減運算去取得差值，而運算後而得的差 值我定義為 Sum，它也和前面 Caculate1、Caculate2 同樣擁有 8 byte 空間存值，課本上有教說 SUBB 是不單單只是"被減數－減數"，它會連同借位一起相減，也就是"結果＝被減數－減數－借位" ，因此我的程式會從最低位(也就是 10^(-4)處)開始接續相減至千位數為止。

若假設 999-0.1 相減：

	千位	百位	十位	個位	10^(-1)	10^(-2)	10^(-3)	10^(-4)
Caculate1	58h	59h	5Ah	5Bh	5Ch	5Dh	5Eh	5Fh
	0	**9**	**9**	**9**	0	0	0	0
Caculate2	60h	61h	62h	63h	64h	65h	66h	67h
	0	0	0	**0**	**1**	0	0	0
Sum	6Ah	6Bh	6Ch	6Dh	6Eh	6Fh	71h	72h
	0	**9**	**9**	**8**	**FF**	0	0	0

↳ 此表為數值相減後

在相減後因為借位會發生可能出現 F7~FF 之間的數，因此我撰寫副程式 ConvertBCD 去處理數值轉換，我們會發現 F7~FF 剛好代表 1~9，也可以用 ANL A, #0Fh、SUBB A, #06h 去將數值換算出來。

Sum	6Ah	6Bh	6Ch	6Dh	6Eh	6Fh	71h	72h
	0	**9**	**9**	**8**	**1**	0	0	0

↳ 此表為數值轉換後

在數值轉換後，因為 LED 只有 4 個，我們只能顯示四位數，此時執行副程式 Print_who 去尋找從最低位哪裡開始做顯示，PrintFrom 會記住開始顯示的 4 位數中最低位處的地址，會從最低位處開始找是因為後面的 0 無意義，因此找小數點後不為 0 的數之地址給 PrintFrom，倘若小數點後皆為 0，則代表由個位數開始做顯示。

接續上面範例，6Eh 處不為 0，MOV PrintFrom, #6Eh。

因為這個 999-0.1 這個舉例剛好會顯示 998.1，但倘若是 10-9，我們則要讓它顯示空空空 1 而非 0001，此時就需要副程式 Delete_zero 去將千位、百位、十位的 0 改值為 0FFh，也就是無字型，而此時就要從最高位(也就是千位)開始確認是否為 0，若最高位至個位皆為 0，則全部都要清除，若不為 0 即可出副程式。

Sum	6Ah	6Bh	6Ch	6Dh	6Eh	6Fh	71h	72h
	FF	**9**	**9**	**8**	**1**	0	0	0

↬ 此表為經過 Delete_zero 副程式後

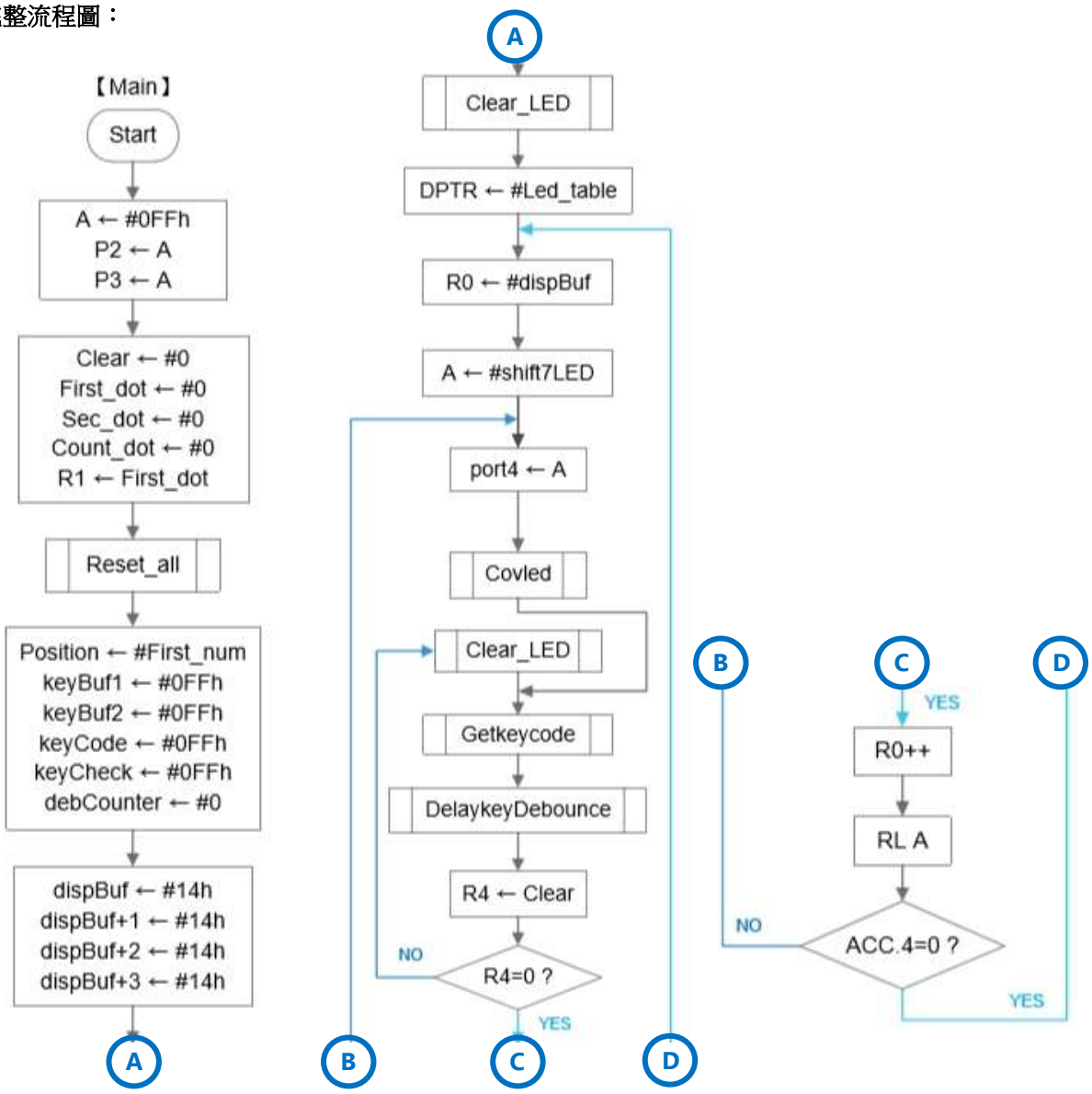
之後呼叫 Oh_Ya_Print 做輸出顯示即完成減法器之運算。

記憶體定義：

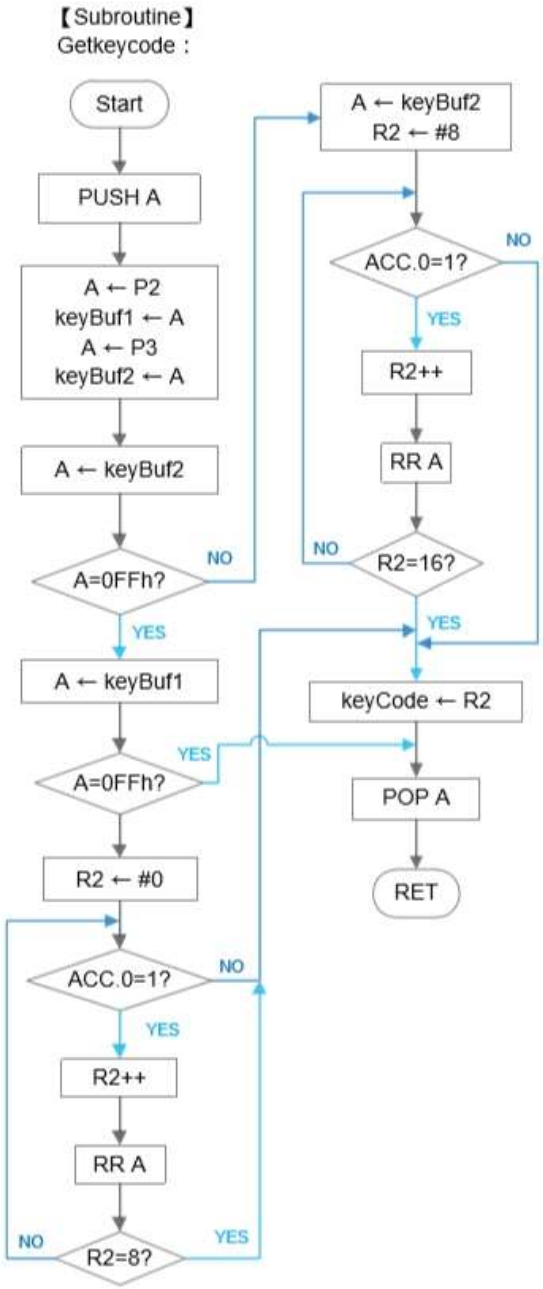
```
;-------------------------------------------------------------------
;【Define】 pseudo name
//用來自定義新的名稱或位置去存取,紀錄其他東西
;-------------------------------------------------------------------
port4           EQU     0E8h    //定義 port4 為位置 0E8h
shift7LED       EQU     0FEh    //目的使七字節每次只有一個點亮
dispBuf         EQU     40h     //最左 LED 為 40h,往右為 41h、42h、43h
keyBuf1         EQU     44h     //存取按鍵 0～7 的輸入，若有輸入顯示 0
keyBuf2         EQU     45h     //存取按鍵 8～F 的輸入，若有輸入顯示 0
keyCode         EQU     46h     //用來存取按鍵碼
keyCheck        EQU     47h     //檢查是否和 keyCode 一樣，有無取到跳躍信號
debCounter      EQU     48h     //用來檢查是否過了跳躍信號
debTimes        EQU     60      //用來延遲時間
Clear           EQU     49h     //判斷是否要執行 clear
Save_position   EQU     4Ah     //於副程式 Clear_LED 中存 R0 原本所擁有的內容
First_dot       EQU     4Bh     //用來記錄輸入的第一個數值小數點前有幾位
Sec_dot         EQU     4Ch     //用來記錄輸入的第二個數值小數點前有幾位
Count_dot       EQU     4Dh     //用來數小數點所在位置
First_num       EQU     4Eh     //4Eh~51h 用來存輸入的第一個數值
Sec_num         EQU     52h     //52h~55h 用來存輸入的第二個數值
Position        EQU     57h     //用來記住現在輸入哪個數值哪個位置
```

Caculate1	EQU	58h	//用來對齊記住輸入的第一個數值的整數+小數
Caculate2	EQU	60h	//用來對齊記住輸入的第二個數值的整數+小數
Count4	EQU	68h	//在副程式 Adjust 中輔助 ,數數字 4 個對齊完沒
Which_Big	EQU	69h	//用來記住輸入數值 1 大還是輸入數值 2 大
Sum_tail	EQU	71h	//6Ah~71h 存 Sum
Borrow	BIT	0	//用來在副程式 Subtraction 中記住是否有 CY
PrintFrom	EQU	72h	//用來記住最低位要從哪裡開始輸出數字
Blank	BIT	1	//用來記住相減後答案是否為 0
WhereSave	EQU	73h	//副程式 Subtraction 中用來記住差值儲存位置
What_num	EQU	74h	//在 Adjust 中因為不能 MOV @R1,@R0
			//所以用 What_num 在中間過度傳數字過去

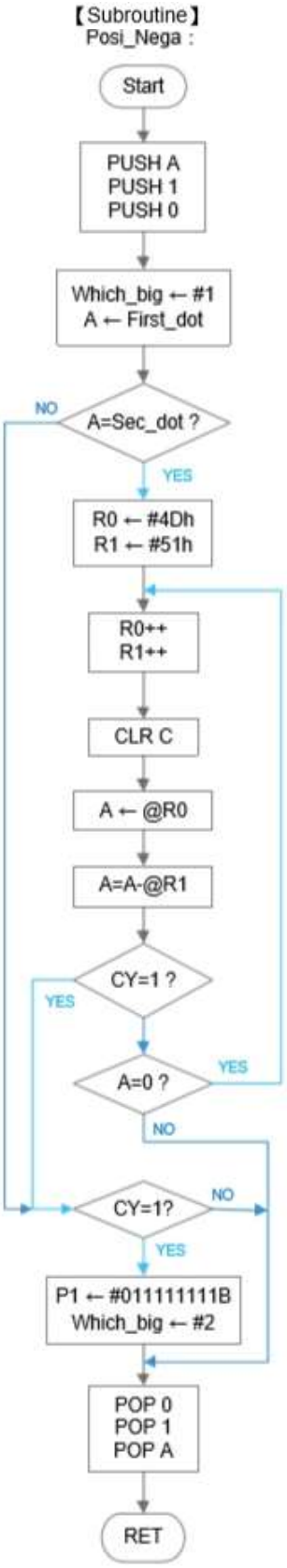
完整流程圖：



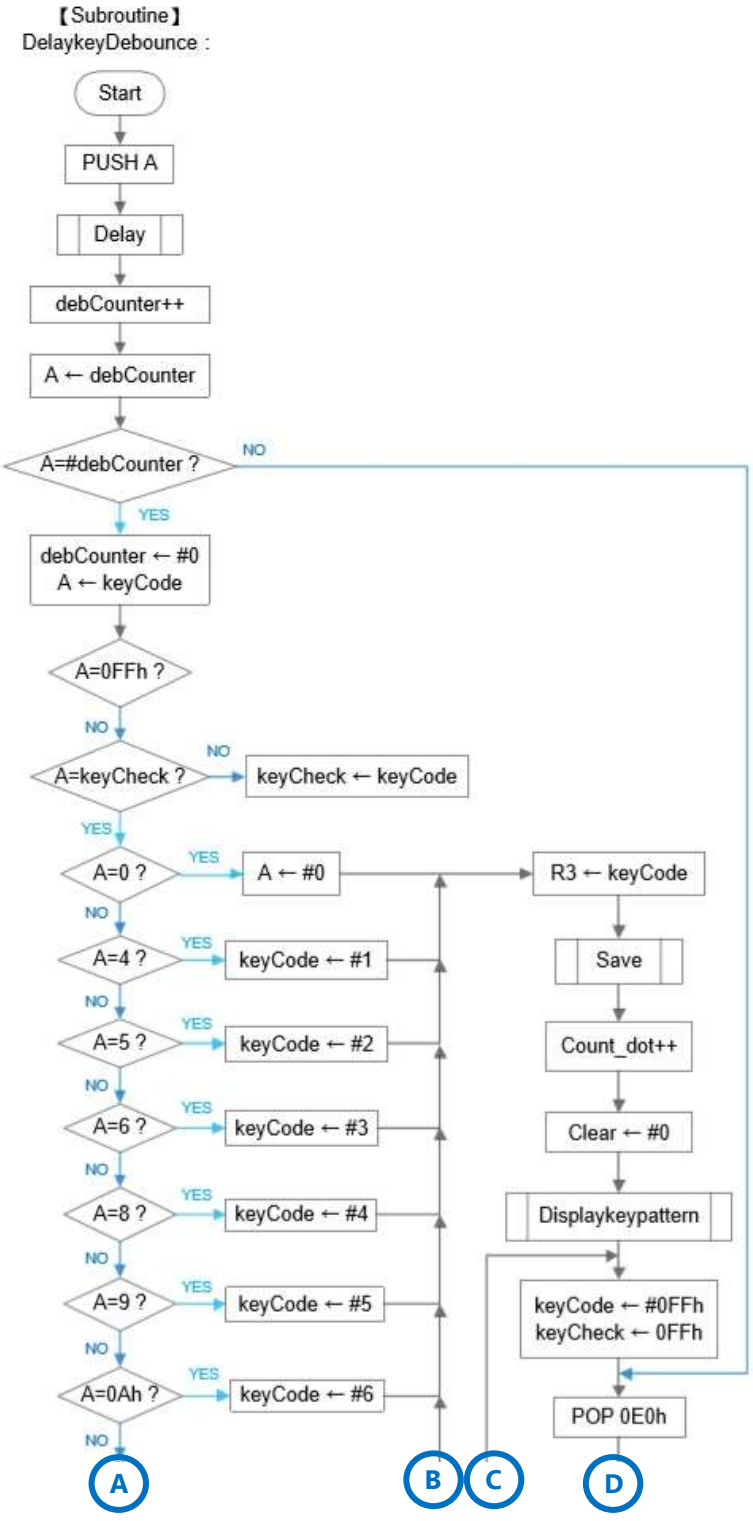
▲Main 流程圖



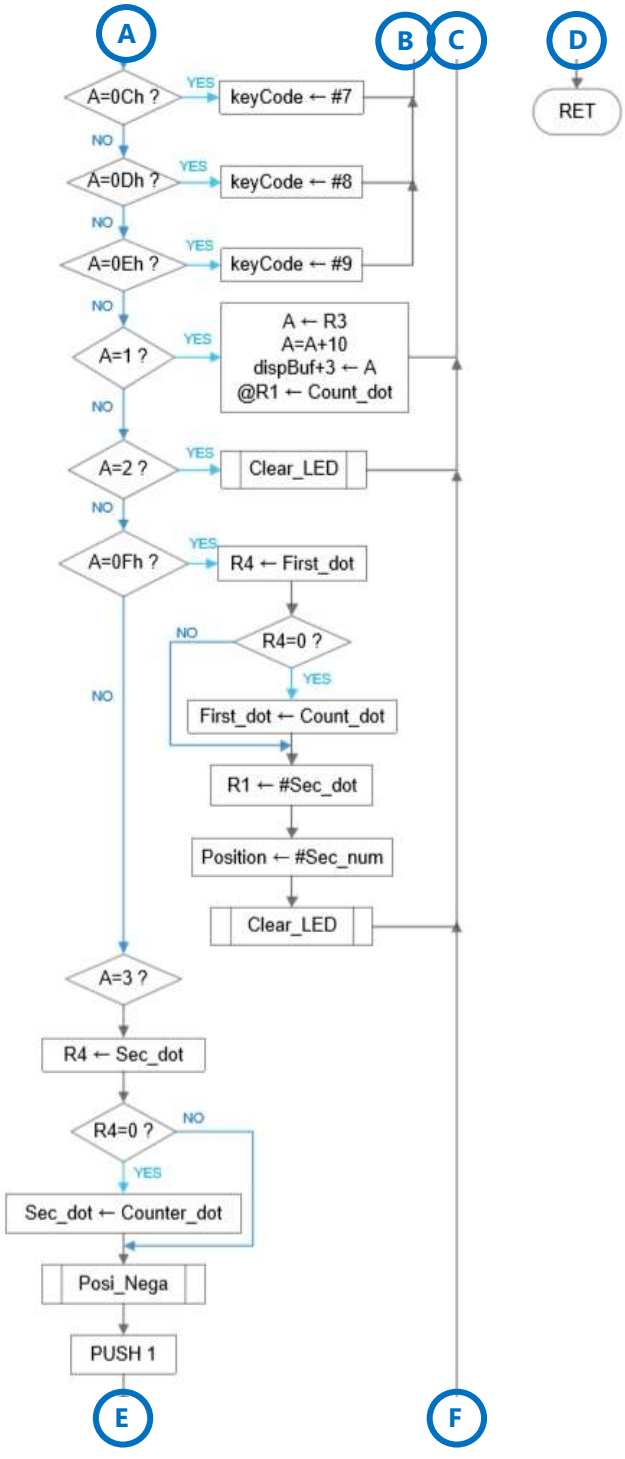
▲Getkeycode 流程圖



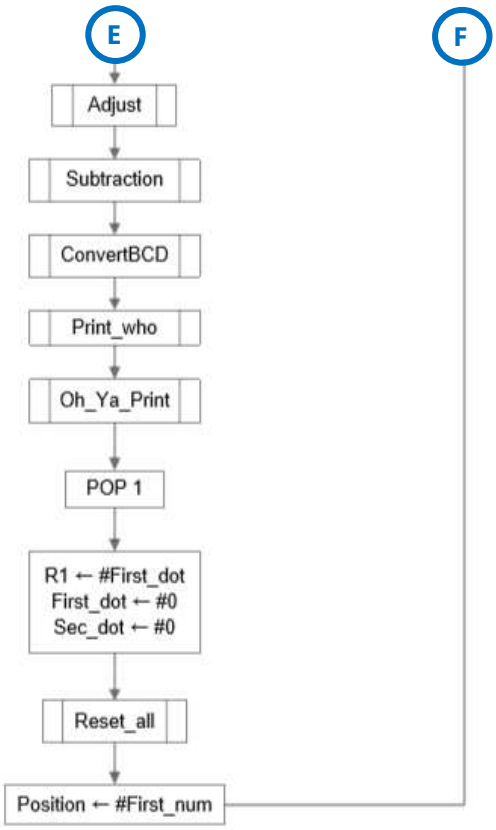
▲Posi_Nega 流程圖



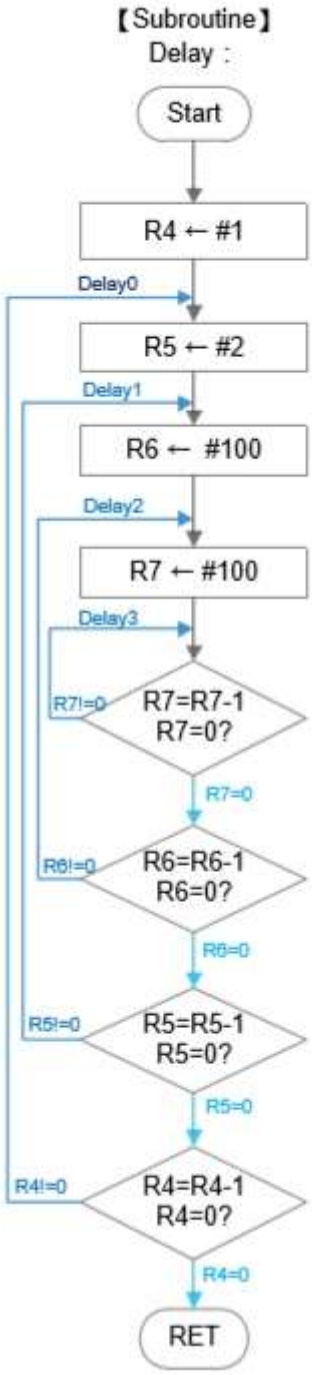
▲DelaykeyDebounce-1 流程圖



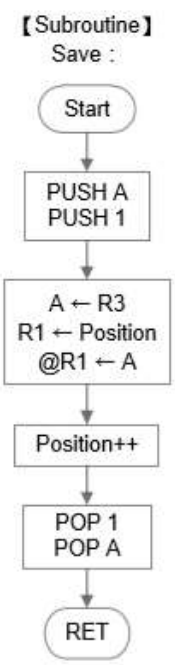
▲DelaykeyDebounce-2 流程圖



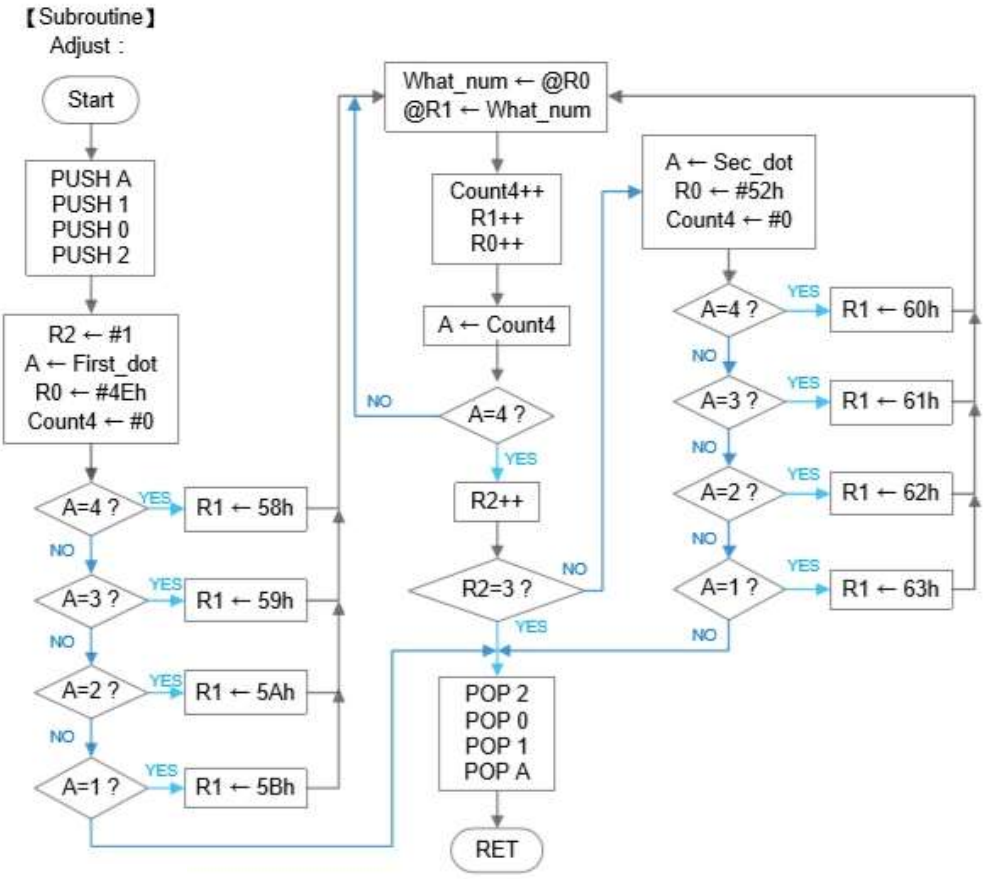
▲DelaykeyDebounce-3 流程圖



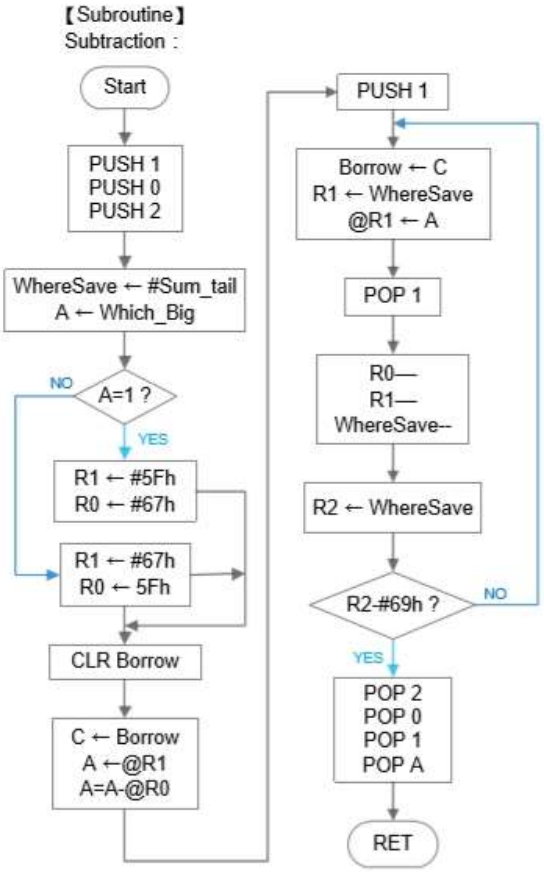
▲Delay 流程圖



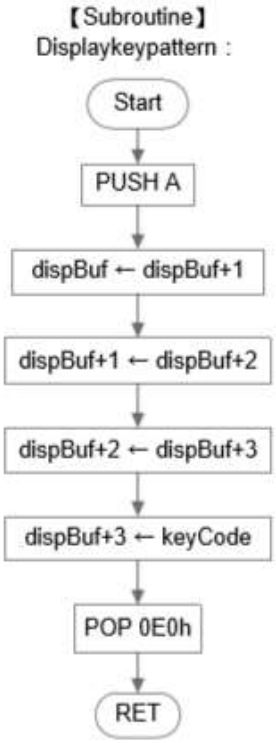
▲Save 流程圖



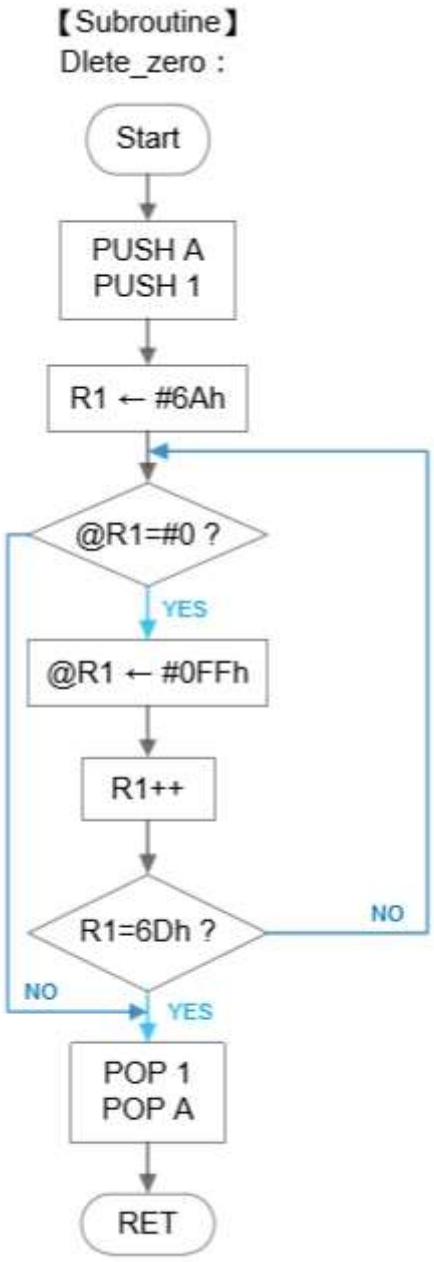
▲Adjust 流程圖



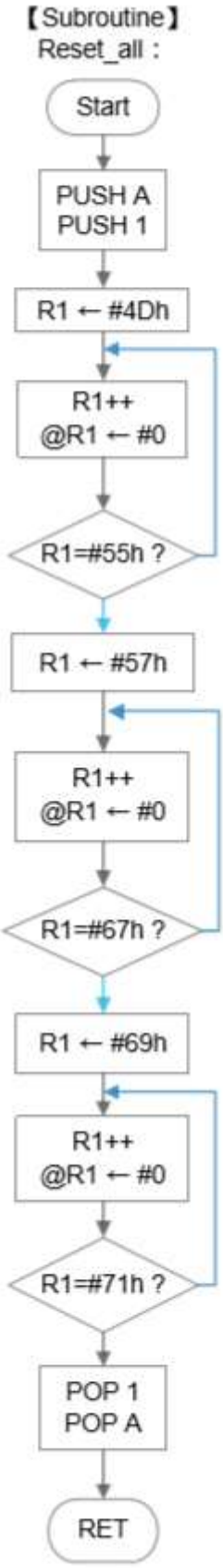
▲Subtraction 流程圖



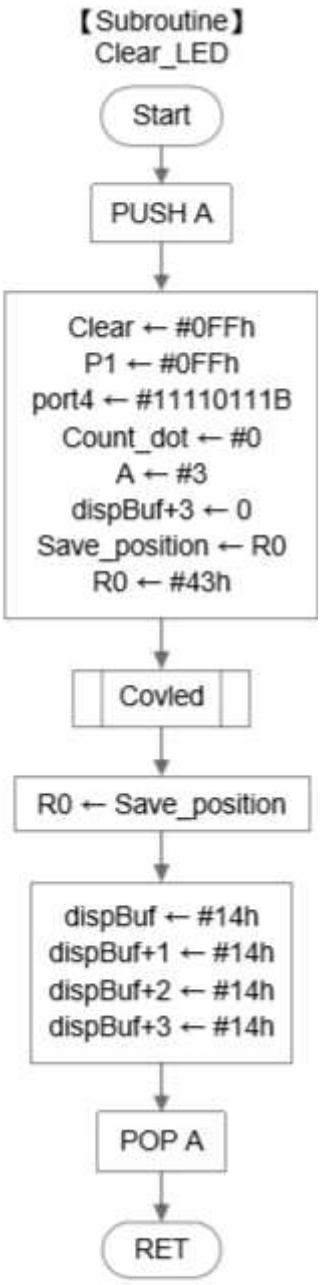
▲Displaykeypattern 流程圖



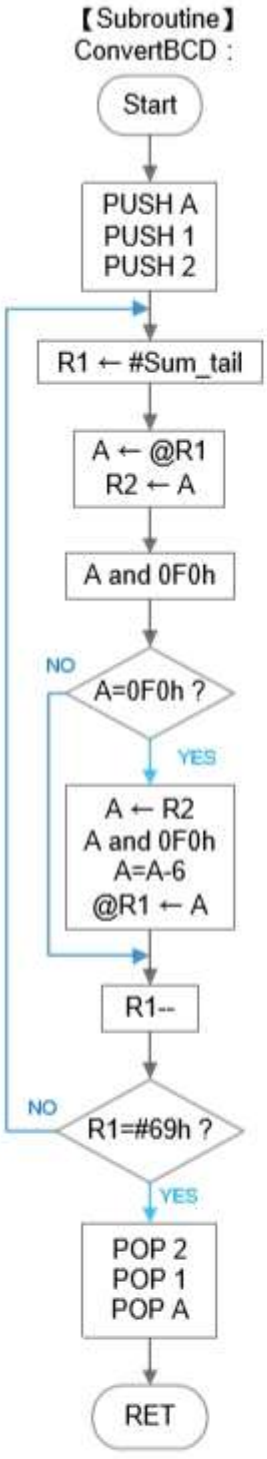
▲Delete_zero 流程圖



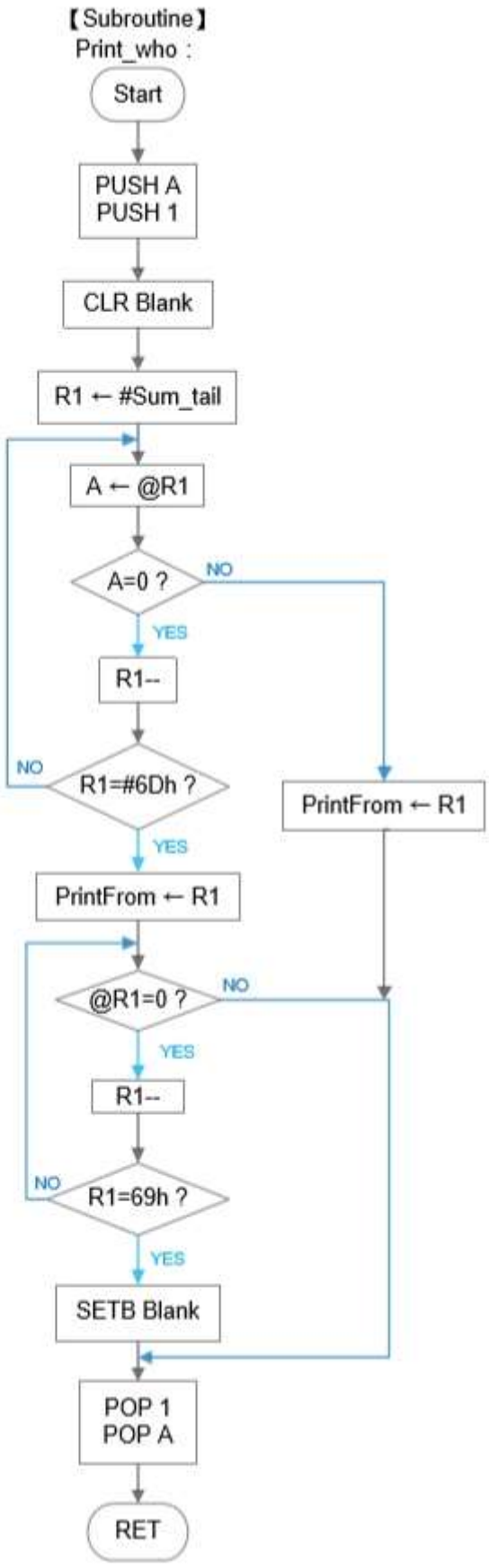
▲Reset_all 流程圖



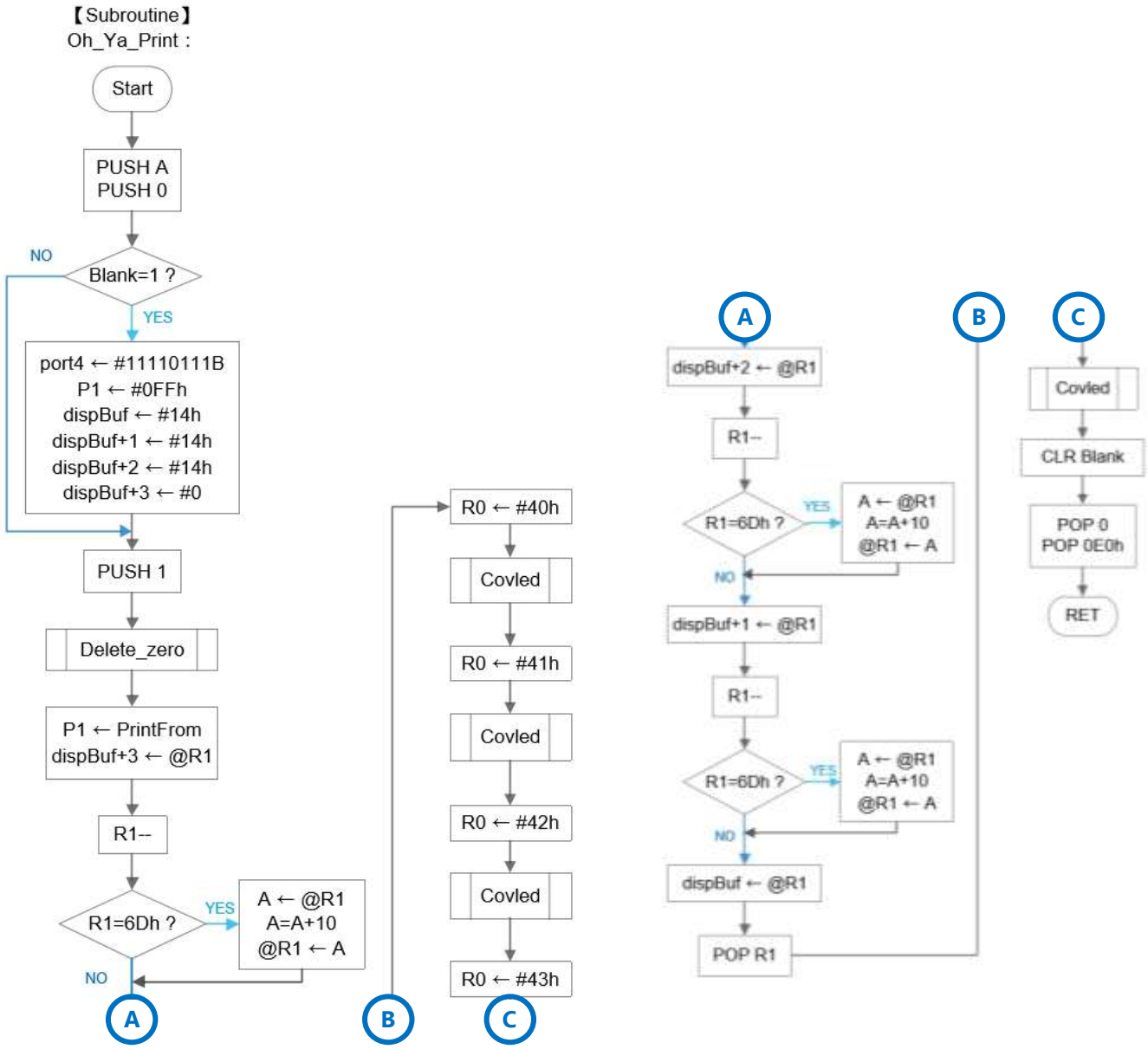
▲Clear_LED 流程圖



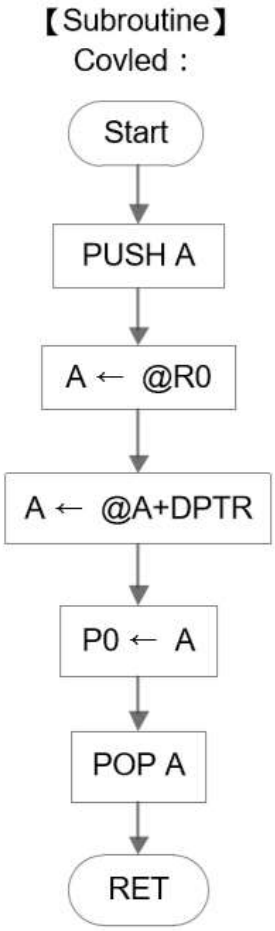
▲ConvertBCD 流程圖



▲Print_who 流程圖



▲Oh_Ya_Print 流程圖



▲Covled 流程圖

```
LED_table: DB 0C0h   ;0
           DB 0F9h   ;1
           DB 0A4h   ;2
           DB 0B0h   ;3
           DB 99h    ;4
           DB 92h    ;5
           DB 82h    ;6
           DB 0D8h   ;7
           DB 80h    ;8
           DB 90h    ;9
           DB 40h    ;0.
           DB 79h    ;1.
           DB 24h    ;2.
           DB 30h    ;3.
           DB 19h    ;4.
           DB 12h    ;5.
           DB 2h     ;6.
           DB 58h    ;7.
           DB 0h     ;8.
           DB 10h    ;9.
          DB 0FFh  ;無字型
```

▲LED_table 資料庫

三、 完整程式（指令必須有詳細清楚的註解，並能與前一節之流程圖相呼應）

```
;-------------------------------------------------------------------
;【Define】 pseudo name
//用來自定義新的名稱或位置去存取,紀錄其他東西
;-------------------------------------------------------------------
port4           EQU   0E8h   //定義 port4 為位置 0E8h
shift7LED       EQU   0FEh   //目的使七字節每次只有一個點亮
dispBuf         EQU   40h    //最左 LED 為 40h,往右為 41h、42h、43h
keyBuf1         EQU   44h    //存取按鍵 0～7 的輸入，若有輸入顯示 0
keyBuf2         EQU   45h    //存取按鍵 8～F 的輸入，若有輸入顯示 0
keyCode         EQU   46h    //用來存取按鍵碼
keyCheck        EQU   47h    //檢查是否和 keyCode 一樣，有無取到跳躍信號
debCounter      EQU   48h    //用來檢查是否過了跳躍信號
debTimes        EQU   60     //用來延遲時間
Clear           EQU   49h    //判斷是否要執行 clear
Save_position   EQU   4Ah    //於副程式 Clear_LED 中存 R0 原本所擁有的內容
First_dot       EQU   4Bh    //用來記錄輸入的第一個數值小數點前有幾位
Sec_dot         EQU   4Ch    //用來記錄輸入的第二個數值小數點前有幾位
Count_dot       EQU   4Dh    //用來數小數點所在位置
First_num       EQU   4Eh    //4Eh~51h 用來存輸入的第一個數值
Sec_num         EQU   52h    //52h~55h 用來存輸入的第二個數值
Position        EQU   57h    //用來記住現在輸入哪個數值哪個位置
Caculate1       EQU   58h    //用來對齊記住輸入的第一個數值的整數+小數
Caculate2       EQU   60h    //用來對齊記住輸入的第二個數值的整數+小數
Count4          EQU   68h    //在副程式 Adjust 中輔助 ,數數字 4 個對齊完沒
Which_Big       EQU   69h    //用來記住輸入數值 1 大還是輸入數值 2 大
Sum_tail        EQU   71h    //6Ah~71h 存 Sum
Borrow          BIT   0      //用來在副程式 Subtraction 中記住是否有 CY
PrintFrom       EQU   72h    //用來記住最低位要從哪裡開始輸出數字
Blank           BIT   1      //用來記住相減後答案是否為 0
WhereSave       EQU   73h    //副程式 Subtraction 中用來記住差值儲存位置
What_num        EQU   74h    //在 Adjust 中因為不能 MOV @R1,@R0
                             //所以用 What_num 在中間過度傳數字過去
```

▲**Table 1** 此為記憶體定義。

```
;-------------------------------------------------------------------
;【Main】 program
//主程式,包含許多數值的初始化以及各個副程式呼叫運行
;-------------------------------------------------------------------
        MOV A, #0FFh          //初始化 A 為 0FFh(表無輸入)
        MOV P2, A             //將 P2 給值為 0FFh(表 0～7 無輸入)
        MOV P3, A             //將 P3 給值為 0FFh(表 8～F 無輸入)

        MOV Clear, #0         //先設定 Clear 為 0,為 0 時不呼叫副程式 Clear

        MOV First_dot,#0      //初始化第一個數值小數點前有 0 位數
        MOV Sec_dot,#0        //初始化第二個數值小數點前有 0 位數
        MOV R1,#First_do      //讓 R1 記住 First_dot 的地址
        MOV Count_dot,#0      //初始化數小數點前有 0 位數

        ACALL Reset_all       //初始化存取數值 1.2 和對齊後數值 1.2 的地方

        MOV Position,#First_num//將 Position 給值為 First_num 的地址
        MOV keyBuf1, #0FFh      //用來存取 P2,設定為 0 輸入
        MOV keyBuf2, #0FFh      //用來存取 P3,設定為 0 輸入
        MOV keyCode,#0FFH       //初始化 keyCode 為無輸入
        MOV keyCheck,#0FFH      //初始化 keyCheck 為無輸入
        MOV debCounter,#0       //初始化"數跳躍信號是否結束"為 0

        MOV dispBuf, #14h       //初始化最左 LED 為無字型
        MOV dispBuf+1, #14h     //初始化左二 LED 為無字型
        MOV dispBuf+2, #14h     //初始化最右 LED 為無字型
        MOV dispBuf+3, #14h     //初始化右二 LED 為無字型
```

▲**Table 2** 此為主程式(上半)。

```
            ACALL Clear_LED          //呼叫副程式 Clear_LED
                                     //使一開機顯示空空空 0
            MOV DPTR, #Led_table     //將 DPTR 設為 LED_table
Reset:      MOV R0, #dispBuf         //初始化 R0 記住 dispBuf 的地址 (最左 LED)
            MOV A, #shift7LED        //初始化 A 為 ShiftLED
Loop:       MOV port4, A             //控制 port4 只亮一個 LED
            ACALL Covled             //呼叫 Covled 輸出字型
            SJMP Going               //跳至標籤 Going 繼續

Clearing:   ACALL Clear_LED          //若 Clear=1 會跳至這行
Going:      ACALL Getkeycode         //取得按鍵碼
            ACALL DelaykeyDebounce   //確認取的值是否為跳躍信號
                                     //若否,則該按鍵之功能為何?
            MOV R4, Clear            //將 Clear 值給 R4
            CJNE R4,#0, Clearing     //若 Clear=0，表非按 CLR 鍵，不用清除
            INC R0                   //換處理下一個 LED
            RL A                     //換亮下一個 LED
            JB ACC.4, loop           //因 LED 只有四個，若 ACC.4=1 繼續執行
            SJMP Reset               //若 ACC.4=0，要重置
```

▲**Table 3** 此為主程式(下半)。

```
;-------------------------------------------
;【Subroutine】 Save
//每輸入一個按鍵,就存一個數字 Ex:0.123,存 0123
//而 Position 記住現在該存數值 1 的位置還是數值 2 的位置
;-------------------------------------------
Save:       PUSH 0E0h            //進副程式先 PUSH A
            PUSH 1               //因為值會動到,所以 PUSH R1
            MOV A,R3             //將 R3(按鍵碼)給 A
            MOV R1,Position      //從 Position 取現在該存的地址給 R1
            MOV @R1,A            //將 A 存於該地址
            INC Position         //Position++(換下一個位置)
            POP 1                //POP R1(配合前面的 PUSH)
            POP 0E0h             //POP A(配合前面的 PUSH)
            RET                  //返回
```

▲**Table 4** 此為副程式－Save。

```
;-------------------------------------------
;【Subroutine】 Get key code
//藉由 P2(存取 0~7 按鍵輸入) .P3(存取 8~F 按鍵輸入)
//去利用 rotate 檢查按鍵碼為何,找到後存於 keyCode
;-------------------------------------------
Getkeycode:
          PUSH OE0h             //PUSH A 保留原本 A 值
          MOV A, P2             //將 P2 存取的按鍵輸入給 A
          MOV keyBuf1, A        //keyBuf1 存取 P2(0~7)按鍵輸入之值
          MOV A, P3             //將 P3 存取的按鍵輸入給 A
          MOV keyBuf2, A        //keyBuf2 存取 P2(8~F)按鍵輸入之值
;=============================
          MOV A, keyBuf2        //將 keyBuf2(存取 8~F 輸入)給 A 做以下處理
          CJNE A,#0FFh,nextcode2  //若 A 不是 FF(零輸入),跳至 nextcode2
          MOV A,keyBuf1         //將 keyBuf1(存取 0~7 輸入)給 A 做以下處理
          CJNE A,#0FFh,nextcode1  //若 A 不是 FF(零輸入),跳至 nextcode1
          SJMP exitgetkeycode //若走到這行代表 0~F 都無輸入,離開副程式
nextcode1:
          MOV R2,#0             //R2 用來數按鍵碼為何
nextcheck1:
          JNB OE0h,gotkeycode //若 ACC.0=0,跳至 gotkeycode(找到碼)
          INC R2                //若沒找到按鍵碼,則 R2+1
          RR A                  //將 A 的所有 bit 向右轉一個
          CJNE R2,#8,nextcheck1//若 R2 還沒+到 8,回去繼續檢查 0~7
          SJMP exitgetkeycode //當 R2=8 時,會走到這步,代表按鍵=7,離開
nextcode2:
          MOV A, keyBuf2        //將 keyBuf2 所存取的按鍵 8~F 輸入值給 A
          MOV R2, #8            //將 R2 給值為 8 開始找按鍵碼
nextcheck2:
          JNB OE0h,gotkeycode//若 ACC.0=0,跳至 gotkeycode(找到碼)
          INC R2                //若沒找到按鍵碼,則 R2+1
          RR A                  //將 A 的所有 bit 向右轉一個
          CJNE R2,#16,nextcheck2//若 R2 還沒+到 16,回去繼續檢查 8~16
          SJMP exitgetkeycode //當 R2=16 時,會走到這步,代表按鍵=16,離開
gotkeycode:
          MOV keyCode, R2       //若找到 keyCode,將 R2 存的按鍵給 keyCode

exitgetkeycode:
          POP OE0h              //因先前有 PUSH A,所以出副程式前必須 POP A
          RET                   //返回
```

▲**Table 5** 此為副程式－GetKeyCode。

```
;-------------------------------------------------------------
;【Subroutine】 Posi_Nega
//用來判定相減數值相減後為正數還是負數
//若為正數,記住輸入數值 1 較大
//若為負數,記住輸入數值 2 較大,且 P1.7 亮
;-------------------------------------------------------------
Posi_Nega:PUSH 0E0h                 //進副程式先 PUSH A
          PUSH 1                    //因為值會動到,所以 PUSH R1
          PUSH 0                    //因為值會動到,所以 PUSH R0
          MOV Which_big,#1          //先假設數值 1 較大
          MOV A,First_dot           //將 First_dot 內容給 A
          CJNE A,Sec_dot,Not_equal//First_dot 與 Sec_dot 若不同
                                    //跳至 Not_equal
          MOV R0,#4Dh               //若相等,執行以下動作,給 R0 值 4Dh
          MOV R1,#51h               //給 R1 值 51h
Againn:   INC R0                    //R0++
          INC R1                    //R1++
          CLR C                     //清除 CY
          MOV A,@R0                 //將 R0 所存取的地址(數值 1)給 A
          SUBB A,@R1                //將數值 1 與數值 2 做相減
          JC Not_equal              //若有 carry 表示為負數,跳至 Not_equal
          JNC Examine               //若無 carry 表示為正數,跳至 Examine
Examine:  CJNE A,#0,ExitPN          //若 A!=0,跳至 ExitPN(若正數,出副程式)
          SJMP Againn               //若 A=0,跳至 Againn 繼續
Not_equal:JNC ExitPN                //若無 carry 表示為正數,跳至 ExitPN
          MOV P1,#01111111B         //確認為負數,P1.7 亮
          MOV Which_big,#2          //若為負數,表數值 2 較大,紀錄 2
ExitPN:   POP 0                     //POP R0(配合前面的 PUSH)
          POP 1                     //POP R1(配合前面的 PUSH)
          POP 0E0h                  //POP A(配合前面的 PUSH)
          RET                       //返回
```

▲**Table 6** 此為副程式－Posi_Nega。

```
;-------------------------------------------
;【Subroutine】 DelaykeyDebounce:
//檢查得到的 keyCode 是否為穩定訊號,而非跳躍訊號的 keyCode
//檢查完後,判斷按鍵碼是否為功能鍵 C 或功能鍵 F
//若都不是,則+246 去取按鍵碼
// 若超過 255(按鍵碼可能為 A.B.D.E)會有 carry,直接重置數值出副程式
;-------------------------------------------
DelaykeyDebounce:
          PUSH 0E0H                 //進副程式前先 PUSH A
          ACALL Delay               //呼叫副程式 Delay 拖延時間
          INC debCounter            //decCounter++
          MOV A,debCounter          //將 debCounter 給 A 以利做下一步判斷
          CJNE A,#debCounter,exit   //若 A 還未達到#debCounter,離開副程式
          MOV debCounter,#0         //若達到,先重置 debCounter=0
          MOV A,keycode             //將得到之 keyCode 給 A 做其他判斷
          CJNE A,#0FFH,Here         //若 A!=#0FFh 跳至 Here
exit:     LJMP exitkey              //若 A=#0FFh 表無輸入,跳至 exitkey 離開
Here:     CJNE A,keycheck,There     //若 A!=keycheck,表示取到跳躍信號
                                    //跳至 There

          CJNE A, #0, Not_zero      //按鍵碼 A 若!=0 跳至 Not_zero
          MOV A, #0                 //轉換鍵盤,keyCode=0 時,為數字 0
          LJMP Get_num_word         //跳至 Get_num_word
There:    LJMP Again                //跳至 Again
Not_zero: CJNE A, #4, Not_one       //按鍵碼 A 若!=4 跳至 Not_one
          MOV keyCode, #1           //若 A=4,keyCode 給值為 1(4 號位為按鍵 1)
          LJMP Get_num_word         //跳至 Get_num_word 去取得字型
Not_one:  CJNE A, #5, Not_two       //按鍵碼 A 若!=5 跳至 Not_two
          MOV keyCode, #2           //若 A=5,keyCode 給值為 2(5 號位為按鍵 2)
          LJMP Get_num_word         //跳至 Get_num_word 去取得字型
Not_two:  CJNE A, #6, Not_three     //按鍵碼 A 若!=6 跳至 Not_three
          MOV keyCode, #3           //若 A=6,keyCode 給值為 3(6 號位為按鍵 3)
          LJMP Get_num_word         //跳至 Get_num_word 去取得字型
Not_three: CJNE A ,#8, Not_four     //按鍵碼 A 若!=8 跳至 Not_four
          MOV keyCode, #4           //若 A=8,keyCode 給值為 4(8 號位為按鍵 4)
          LJMP Get_num_word         //跳至 Get_num_word 去取得字型
Not_four: CJNE A, #9, Not_five      //按鍵碼 A 若!=9 跳至 Not_five
          MOV keyCode, #5           //若 A=9,keyCode 給值為 5(9 號位為按鍵 5)
          LJMP Get_num_word         //跳至 Get_num_word 去取得字型
Not_five: CJNE A, #0Ah, Not_six     //按鍵碼 A 若!=0Ah 跳至 Not_six
          MOV keyCode, #6           //若 A=0Ah,keyCode 給值為 6
                                    //(A 號位為按鍵 6)
```

▲**Table 7** 此為副程式－DelaykeyDebounce-1。

```
            LJMP Get_num_word        //跳至 Get_num_word 去取得字型
Not_six:    CJNE A, #0Ch, Not_seven  //按鍵碼 A 若!=0Ch 跳至 Not_seven
            MOV keyCode, #7          //若 A=0Ch,keyCode 給值為 7
                                     //(C 號位為按鍵 7)
            SJMP Get_num_word        //跳至 Get_num_word 去取得字型
Not_seven:  CJNE A,#0Dh,Not_eight    //按鍵碼 A 若!=0Dh 跳至 Not_eight
            MOV keyCode, #8          //若 A=0Dh,keyCode 給值為 8
                                     //(D 號位為按鍵 8)
            SJMP Get_num_word        //跳至 Get_num_word 去取得字型
Not_eight:  CJNE A,#0Eh,Not_nine     //按鍵碼 A 若!=0Eh 跳至 Not_nine
            MOV keyCode, #9          //若 A=0Eh,keyCode 給值為 9
                                     //(E 號位為按鍵 9)
            SJMP Get_num_word        //跳至 Get_num_word 去取得字型
Not_nine:   CJNE A, #1, Not_dot      //按鍵碼 A 若!=1 跳至 Not_dot
            MOV A, R3                //若 A=1,將 R3(最後一次數字值)給 A
            ADD A,#10                //A=A+10(目的為取小數 Table)
            MOV dispBuf+3,A          //將 A 值給 dispBuf+3 做輸出
            MOV @R1,Count_dot        //@R1(First 或 Sec_dot)存 Count_dot
            SJMP Res                 //跳至 Res 重設按鍵初值
Not_dot:    CJNE  A ,#2, Not_clear   //按鍵碼 A 若!=2 跳至 Not_cear
            ACALL Clear_LED          //呼叫副程式 Clear_LED 進行 Clear
            SJMP Res                 //跳至 Res 重設按鍵初值
Not_clear:  CJNE A,#0Fh,Not_op       //按鍵碼 A 若!=0Fh 跳至 Not_op
            MOV R4,First_dot         //將 First_dot 值給 R4
            CJNE R4,#0,Have_dot      //若 R4!=0,跳至 Have_dot
            MOV First_dot,Count_dot  //若 R4=0,將 Count_dot 值給 First_dot
            MOV R1,#Sec_dot          //R1 記住 Sec_dot 的地址
Have_dot:   MOV R1,#Sec_dot          //R1 記住 Sec_dot 的地址
            MOV Position,#Sec_num    //Position 記住 Sec_num 的地址

            ACALL Clear_LED          //呼叫 Clear 副程式
            SJMP Res                 //跳至 Res 重設按鍵初值

Not_op:     CJNE A,#3,Res            //按鍵碼 A!=3,跳至 Res(最後 1 個按鍵檢查)
            MOV R4,Sec_dot           //將 Sec_dot 值給 R4
            CJNE R4,#0,Have_secdot   //若 R4!=0,跳至 Have_secdot
            MOV Sec_dot,Count_dot    //若 R4=0,將 Count_dot 值給 Sec_dot
Have_secdot:ACALL Posi_Nega          //呼叫副程式 Posi_Nega 將兩數值比大小
            PUSH 1                   //因為會變更 R1 值,先 PUSH R1
            ACALL Adjust             //呼叫副程式 Adjust 整理並存取對齊後數值
            ACALL Subtraction        //呼叫副程式 Subtraction 去做減法運算
            ACALL ConvertBCD         //因會有借位問題,呼叫 ConvertBCD 做轉換
```

▲**Table 8** 此為副程式－DelaykeyDebounce-2。

```
            ACALL Print_who          //呼叫副程式 Print_who 去找輸出哪 4 位數
            ACALL Oh_Ya_Print        //呼記副程式 Oh_Ya_Print 做輸出
            POP 1                    //因前面有 PUSH R1,因此這裡要 POP R1
            MOV R1,#First_dot        //重設 R1 存取 First_dot 的地址
            MOV First_dot,#0         //重設 First_dot 為 0
            MOV Sec_dot,#0           //重設 Sec_dot 為 0
            ACALL Reset_all          //呼叫副程式 Reset_all
                                     //重設存取數值的各個地址內容為 0
            MOV Position,#First_num  //重設 Position 存 First_num 的地址
            SJMP Res                 //跳至 Res 重設按鍵初值

Get_num_word:MOV R3, keyCode         //若按鍵為數字,將 keyCode 給 R3 存
            ACALL Save               //呼叫副程式 Save 將數字存起來
            INC Count_dot            //Count_dot++
            MOV Clear,#0             //重設 Clear 值為 0(不做 Clear)
            ACALL Displaykeypattern  //呼叫 Displaykeypattern 做輸出顯示
Res:        MOV keycode,#0FFH        //重設 keyCode 值為 0FFh
            MOV keycheck,#0FFH       //重設 keyCheck 值為 0FFh
            SJMP exitkey             //跳至 exitkey
Again:      MOV keycheck,keycode     //若按鍵為跳躍信號,keyCode 給 keyCheck
exitkey:    POP 0E0H                 //因先前有 PUSH A,出副程式前必須 POP A
            RET                      //返回
```

▲**Table 9** 此為副程式－DelaykeyDebounce-3。

```
;-------------------------------------------
;【Subroutine】 Adjust
;用來將兩數值存到對齊後存到新的 8byte 空間
;以置中的方式對齊個位數
;-------------------------------------------
Adjust:   PUSH 0E0h            //進入副程式前先 PUSH A
          PUSH 1               //因為會用到 R1,因此先 PUSH R1
          PUSH 0               //因為會用到 R0,因此先 PUSH R0
          PUSH 2               //因為會用到 R2,因此先 PUSH R2

          MOV R2,#1            //R2 用來記住現在處理數值 1 還是數值 2

          MOV A,First_dot      //將 First_dot(記住數值 1 小數點前有幾位)給 A
          MOV R0,#4Eh          //將 R0 給值為 4Eh(數值 1 儲存位置 4Eh~51h)
          MOV Count4,#0        //給 Count4 值 0,輸入最多 4 個
                               //(數數字 4 個對齊完沒)
          CJNE A,#4,Not4       //小數點前有 4 位數?
          MOV R1,#58h          //若 4 位數,從 58h 存(58h~5Bh 存對齊後數值 1)
          SJMP Enter_num       //跳至 Enter_num 存對齊的數值,
                               //(存於 58.59.5A.5B)
Not4:     CJNE A,#3,Not3       //小數點前有 3 位數?
          MOV R1,#59h          //若 3 位數,從 59h 存(59h~5Ch 存對齊後數值 1)
          SJMP Enter_num       //跳至 Enter_num 存對齊的數值,
                               //(存於 59.5A.5B.5C)
Not3:     CJNE A,#2,Not2       //小數點前有 2 位數?
          MOV R1,#5Ah          //若 2 位數,從 5Ah 存(5Ah~5Dh 存對齊後數值 1)
          SJMP Enter_num       //跳至 Enter_num 存對齊的數值,
                               //(存於 5A.5B.5C.5D)
Not2:     CJNE A,#1,Exit_adjust//小數點前有 1 位數?
          MOV R1,#5Bh          //若 1 位數,從 5Bh 存(5Bh~5Eh 存對齊後數值 1)
          SJMP Enter_num       //跳至 Enter_num 存對齊的數值,
                               //(存於 5B.5C.5D.5E)
                               //以下迴圈為處理數值 2
Num2:     MOV A,Sec_dot        //將 Sec_dot(記住數值 2 小數點前有幾位)給 A
          MOV R0,#52h          //將 R0 給值為 52h(數值 2 儲存位置 52h~55h)
          MOV Count4,#0        //給 Count4 值 0,輸入最多 4 個
                               //(數數字 4 個對齊完沒)
          CJNE A,#4,Not44      //小數點前有 4 位數?
          MOV R1,#60h          //若 4 位數,從 60h 存(60h~63h 存對齊後數值 2)
          SJMP Enter_num       //跳至 Enter_num 存對齊的數值,
                               //(存於 60.61.62.63)
Not44:    CJNE A,#3,Not33      //小數點前有 3 位數?
```

▲**Table 10** 此為副程式－Adust(上半)。

```
            MOV R1,#61h          //若 3 位數,從 61h 存(61h~64h 存對齊後數值 2)
            SJMP Enter_num       //跳至 Enter_num 存對齊的數值,
                                 //(存於 61.62.63.64)
Not33:      CJNE A,#2,Not22      //小數點前有 2 位數?
            MOV R1,#62h          //若 2 位數,從 62h 存(62h~65h 存對齊後數值 2)
            SJMP Enter_num       //跳至 Enter_num 存對齊的數值,
                                 //(存於 62.63.64.65)
Not22:      CJNE A,#1,Exit_adjust//小數點前有 1 位數?
            MOV R1,#63h          //若 1 位數,從 63h 存(63h~66h 存對齊後數值 2)
            SJMP Enter_num       //跳至 Enter_num 存對齊的數值,
                                 //(存於 63.64.65.66)
                                 //因小數點前至少有 1 位,R1 一定存有對齊位置
                                 // R1 一定存有對齊位置的起始
Enter_num:MOV What_num,@R0       //給 What_num 值為@R0(原輸入數字)
            MOV @R1, What_num    //再將該數字存到@R1(R1 記對齊要從哪裡開始存)
            INC Count4           //Count4++(已存入幾個數值)
            INC R1               //R1++(換下一個新地址)
            INC R0               //R0++(換下一個舊地址)
            MOV A,Count4         //將 Count4 值給 A 去做判定
            CJNE A,#4,Enter_num  //A(Count4)是否等於 4
            INC R2               //若 A=4,表數值對齊轉換完畢,R2++
            CJNE R2,#3,Num2      //R2 是否等於 3 若等於 3 表兩數值對齊轉換完畢
Exit_adjust:  POP 2              //因先前有 PUSH R2,所以要 POP R2
            POP 0                //因先前有 PUSH R0,所以要 POP R0
            POP 1                //因先前有 PUSH R1,所以要 POP R1
            POP 0E0h             //因先前有 PUSH A,所以要 POP A
            RET                  //返回
```

▲**Table 11** 此為副程式－Adust(下半)。

```
;-----------------------------------------------
;【Subroutine】 Subtraction
//將對齊後的兩數值從最低位(最右邊)開始用 SUBB 連續相減
;-----------------------------------------------
Subtraction:  PUSH 0E0h            //進入副程式前先 PUSH A
              PUSH 1               //因為會用到 R1,因此先 PUSH R1
              PUSH 0               //因為會用到 R0,因此先 PUSH R0
              PUSH 2               //因為會用到 R2,因此先 PUSH R2
              MOV WhereSave,#Sum_tail //WhereSave 用來記住結果的最低位
              MOV A,Which_Big      //將 A 給值為 Which_big
              CJNE A,#1,Bigger2    //A!=1,表輸入數值 2 較大,跳至 Bigger2
              MOV R1,#5Fh          //R1 用來記住轉換後數值 1 的最低位的位置(大)
              MOV R0,#67h          //R0 用來記住轉換後數值 2 的最低位的位置(小)
              SJMP Howmany         //跳至 Howmany 進行減法運算
Bigger2:      MOV R1,#67h          //  R1 用來記住轉換後數值 2 的最低位的位置(大)
              MOV R0,#5Fh          //  R0 用來記住轉換後數值 1 的最低位的位置(大)
Howmany:      CLR Borrow           //先清楚 Borrow
Cacu:         MOV C,Borrow         //在將清楚後的 Borrow 值給 C
              MOV A,@R1            //將 R1 所存的地址去取內容給 A
              SUBB A,@R0           //A-@R0,相減之後的數值會存在 A

              PUSH 1               //因為會用到 R1,所以先 PUSH R1
              MOV Borrow,C         //怕 CY 值在下次減法前受影響,先用 Borrow 存
              MOV R1,WhereSave     //讓 R1 記住相減後結果要存的位置
              MOV @R1,A            //將相減後的值 A 給@R1 存
              POP 1                //POP R1 回來
              DEC R0               //R0--,換下一位數
              DEC R1               //R1--,換下一位數
              DEC WhereSave        //WhereSave,換下一個位置
              MOV R2,WhereSave     //將 WhereSave 值給 R2
              CJNE R2,#69h,Cacu    //R2!=69h,跳至 Cacu 繼續算(結果存 6Ah~71h)

Over:         POP 2                //因先前有 PUSH R2,所以要 POP R2
              POP 0                //因先前有 PUSH R0,所以要 POP R0
              POP 1                //因先前有 PUSH R1,所以要 POP R1
              POP 0E0h             //因先前有 PUSH A,所以要 POP A
              RET                  //返回
```

▲**Table 12** 此為副程式－Subtraction。

```
;-------------------------------------------
;【Subroutine】 ConvertBCD
//將相減後的數值，可能會出現 F7~FF 之間的數值
//需要將該數值 and 0Fh 後減 06h 或是直接做對應取值
//我是直接對應取值,Ex:FF=9
;-------------------------------------------
ConvertBCD:PUSH 0E0h              //進入副程式前先 PUSH A
           PUSH 1                 //因為會用到 R1,因此先 PUSH R1
           PUSH 2                 //因為會用到 R2,因此先 PUSH R2
           MOV R1,#Sum_tail       //R1 用來記住結果的最低位(Sum_tail)
NotFinish:MOV A,@R1               //將@R1(結果數值)給 A
           MOV R2,A               //再將結果數值由 A 給 R2
           ANL A,#0F0h            //A 去 and F0,留下 high byte
           CJNE A,#0F0h,NoF       //若 A!=0F0h,跳至 NoF,表數值不用轉換
           MOV A,R2               //若 A=0F0h,表數值要做轉換,將原數值 R2 給 A
           ANL A,#0Fh             //之後 A 再去 and 0Fh,留下 low byte
           SUBB A,#6              //將 A 的 low byte 減 6
           MOV @R1,A              //將轉換後數值給@R1(該結果所要存的位置)
NoF:       DEC R1                 //R1--,換下一個位置
           CJNE R1,#69h,NotFinish  //R1!=69h,表尚未處理完,跳至 NotFinish
           POP 2                  //因先前有 PUSH R2,所以要 POP R2
           POP 1                  //因先前有 PUSH R1,所以要 POP R1
           POP 0E0h               //因先前有 PUSH A,所以要 POP A
           RET                    //返回
```

▲**Table 13** 此為副程式－ConvertBCD。

```
;-------------------------------------------
;【Subroutine】 Print_who
//減法執行完且數值轉換完後,6Ah~71h 存有結果
//從最低位(71h)開始找不為 0 的數
//PrintFrom 記住該從哪一位開始印(結果的最低位)
;-------------------------------------------
Print_who:PUSH 0E0h             //進入副程式前先 PUSH A
          PUSH 1                //因為會用到 R1,因此先 PUSH R1
          CLR Blank             //先清空 Blank,運算後假設結果不為 0
          MOV R1,#Sum_tail      //R1 用來記住結果的最低位(Sum_tail)
Gogo:     MOV A,@R1             //將@R1(結果數值)給 A
          CJNE A,#0,Print       //若找到最低位結果數值不為 0,跳至 Print 處理
          DEC R1                //R1--,換下一個位數檢查
          CJNE R1,#6Dh,Gogo     //R1!=6Dh(個位數),跳至 Gogo 繼續找
          MOV PrintFrom,R1      //若R1=6Dh,從個位數開始印,PrintFrom記 6Dh
Check:    CJNE @R1,#0,Exit_print  //@R1!=0,表結果不為 0,跳至 Exit_print
          DEC R1                //若 R1=0,繼續檢查其他位數,R1--
          CJNE R1,#69h,Check    //若 R1!=69h,跳至 Check 繼續檢查
          SETB Blank            //若 R1=69h,表差值為 0,Blank 設為 1
          SJMP Exit_print       //跳至 Exit_print
Print:    MOV PrintFrom,R1      //若找到要從哪裡印,將 R1 位置給 PrintFrom 記
Exit_print:POP 1                //因先前有 PUSH R1,所以要 POP R1
          POP 0E0h              //因先前有 PUSH A,所以要 POP A
          RET                   //返回
```

▲**Table 14** 此為副程式－Print_who。

```
;-------------------------------------------
;【Define】 Delete_zero
//將高位沒用的 0 刪除
//Ex0012.3 刪除最左邊的兩個 0
;-------------------------------------------
Delete_zero:  PUSH 0E0h         //進入副程式前先 PUSH A
          PUSH 1                //因為會用到 R1,因此先 PUSH R1
          MOV R1,#6Ah           //將 R1 給值為 6Ah,結果的最高位
Againnn:  CJNE @R1,#0,Exit_delete //若 R1!=0,跳至 Exit_delete
          MOV @R1,#0FFh         //若 R1=0,將@R1 給值為 0FFh(無字型)
          INC R1                //R1++,換下一位
          CJNE R1,#6Dh,Againnn//若 R1!=6Dh 表尚未處理完,跳至 Againnn 繼續
Exit_delete:POP 1               //因先前有 PUSH R1,所以要 POP R1
          POP 0E0h              //因先前有 PUSH A,所以要 POP A
          RET                   //返回
```

▲**Table 15** 此為副程式－Delete_zero。

```
;-------------------------------------------
;【Subroutine】 Oh_Ya_Print
//此副程式在 Print_who 找到由誰開始印時
//因 PrinrFrom 記住要從哪裡開始印
//在這裡做輸出顯示
;-------------------------------------------
Oh_Ya_Print:
          PUSH 0E0h            //進入副程式前先 PUSH A
          PUSH 0               //因為會用到 R0,因此先 PUSH R0
          JNB Blank,Sum_not0   //先看看差值是否為 0,若非 0，跳至 Sum_not0
          MOV Port4,#11110111B//若差值為 0,只有 LED 最右會亮
          MOV P1,#0FFh         //字型輸出顯示給 0FFh(全關)
          Mov dispBuf, #14h    //給左一 LED#14h 無字型
          Mov dispBuf+1, #14h  //給左二 LED#14h 無字型
          Mov dispBuf+2, #14h  //給左三 LED#14h 無字型
          MOV dispBuf+3, #0    //給右一 LED#0 字型 0

Sum_not0: PUSH 1               //因為會用到 R1,所以先 PUSH R1
          ACALL Delete_zero    //呼叫副程式 Delete_zero 清除高位無用處之 0
          MOV R1, PrintFrom    //將 PrintFrom 值給 R1

          MOV dispBuf+3,@R1    //取 R1 內容為地址,取該地址內容給 dispBuf+3

          DEC R1               //R1--,換下一位
          CJNE R1,#6Dh,NoDot1  //R1!=6D,跳至 NoDot1
          MOV A,@R1            //若 R1=6D,將@R1 值給 A
          ADD A,#10            //A=A+10
          MOV @R1,A            //再將 A 值還給@R1
NoDot1:   MOV dispBuf+2,@R1    //取 R1 內容為地址,取該地址內容給 dispBuf+2

          DEC R1               //R1--,換下一位
          CJNE R1,#6Dh,NoDot2  //R1!=6D,跳至 NoDot2
          MOV A,@R1            //若 R1=6D,將@R1 值給 A
          ADD A,#10            // A=A+10
          MOV @R1,A            //再將 A 值還給@R1

NoDot2:   MOV dispBuf+1,@R1    //取 R1 內容為地址,取該地址內容給 dispBuf+1

          DEC R1               //R1--
          CJNE R1,#6Dh,NoDot3  //R1!=6D,跳至 NoDot3
          MOV A,@R1            //若 R1=6D,將@R1 值給 A
          ADD A,#10            // A=A+10
```

▲**Table 16** 此為副程式－Oh_Ya_Print(上半)。

```
            MOV @R1,A           //再將 A 值還給@R1
NoDot3:     MOV dispBuf,@R1     //取 R1 內容為地址,取該地址內容給 dispBuf
            POP 1               // POP R1 回來

Show:       MOV R0,#40h         //R0 給值為 40h
            ACALL Covled        //呼叫副程式 Covled 做輸出顯示
            MOV R0,#41h         //R0 給值為 41h
            ACALL Covled        //呼叫副程式 Covled 做輸出顯示
            MOV R0,#42h         //R0 給值為 42h
            ACALL Covled        //呼叫副程式 Covled 做輸出顯示
            MOV R0,#43h         //R0 給值為 43h
            ACALL Covled        //呼叫副程式 Covled 做輸出顯示

Exit_oh:    CLR Blank           //重設 Blank 為 0
            POP 0               //因先前有 PUSH R0,所以要 POP R0
            POP 0E0h            //因先前有 PUSH A,所以要 POP A
            RET                 //返回
```

▲**Table 16** 此為副程式－Oh_Ya_Print(下半)。

```
;-------------------------------------------
;【Subroutine】 time delay
//用在時間延遲,透過迴圈去浪費時間造成 delay 效果
;-------------------------------------------
Delay:
            mov r4, #1          //R4 給值 1
delay0:     mov r5, #2          //R5 給值 2
delay1:     mov r6, #100        //R6 給值 100
delay2:     mov r7, #100        //R7 給值 100
delay3:     djnz r7, delay3     //若 R7!=0,跳至 delay3
            djnz r6, delay2     //若 R6!=0,跳至 delay2
            djnz r5, delay1     //若 R5!=0,跳至 delay1
            djnz r4, delay0     //若 R4!=0,跳至 delay0
            ret                 //返回
```

▲**Table 17** 此為副程式－Delay。

```
;-----------------------------------------
;【Subroutine】 7-seg LED pattern conversion
//主要是向 Led_teble 取字形並交給 P0 去做顯示
;-----------------------------------------
Covled:
          PUSH   0E0h          //進入副程式前先 PUSH A
          mov    a, @R0        //取 R0 內容當地址,到該地址取內容給 A
          movc   a, @a+dptr    //將 dptr+A(偏移量)到 Led_table 取字型
          mov    p0, a         //取到的字形傳至 P0 做輸出
          POP    0E0h          //因先前有 PUSH A,所以要 POP A
          Ret                  //返回
```

▲**Table 18** 此為副程式－Covled。

```
;-----------------------------------------
;【Subroutine】 Clear_LED
//Clear 鍵按下後,需顯示空空空 0
//其中有些數值也要隨之重設
;-----------------------------------------
Clear_LED:
          PUSH 0E0h            //進入副程式前先 PUSH A
          MOV Clear, #0FFh     //Clear 給值為 0FFh 代表啟動 Clear
          MOV P1,#0FFh         //將 P1 給值為 0FFh,表指示燈全關
          MOV Port4,#11110111B//Port4 給值為 1111011B,表只亮最右的 LED
          MOV Count_dot,#0     //將 Count_dot 給值為 0,表小數點前有 0 位
          MOV A,#3             //將 A 給值為 3
          MOV dispBuf+3, #0    //dispBuf++3 給值為 0
          MOV Save_position,R0//先將 R0 值給 Save_position 儲存原值
          MOV R0,#43h          //R0 給值為 43h
          ACALL Covled         //呼叫副程式 Covled 做顯示輸出
          MOV R0,Save_position//再將原本 R0 的值還給 R0

          Mov dispBuf, #14h    //給左一 LED#14h 無字型
          Mov dispBuf+1, #14h  //給左二 LED#14h 無字型
          Mov dispBuf+2, #14h  //給右二 LED#14h 無字型
          Mov dispBuf+3, #14h  //給右一 LED#14h 無字型

          POP    0E0h          //因先前有 PUSH A,所以要 POP A
          Ret                  //返回
```

▲**Table 18** 此為副程式－Clear_LED。

```
;-----------------------------------------------
;【Subroutine】 Reset_all
//將所有存數值的地方都重設為 0
//輸入數值 1 存於:4Eh~51h,輸入數值 2 存於:52h~55h
//對齊後數值 1 存於 58h~5Fh,對齊後數值 2 存於 60h~67h
//運算後結果(差值)存於 6Ah~71h,(以上空間都要重設為 0)
;-----------------------------------------------
Reset_all:PUSH 0E0h             //進入副程式前先 PUSH A
          PUSH 1                //PUSH R1
          MOV R1,#4Dh           //給 R1 為 4Dh(為了+1=4Eh,重設輸入數值 1.2)
Initial_1:INC R1                //R1++,換下一位
          MOV @R1,#0            //取 R1 內容當地址,該地址內容設為 0
          CJNE R1,#55h,Initial_1  //若 R1!=56h,跳至 Initial_1 繼續初始化

          MOV R1,#57h           //給 R1 為 57h(為了+1=58h,重設對齊數值 1.2)
Inintial_2:INC R1               //R1++,換下一位
          MOV @R1,#0            //取 R1 內容當地址,該地址內容設為 0
          CJNE R1,#67h,Inintial_2 //若 R1!=67h,跳至 Initial_2 繼續初始化

          MOV R1,#69h           //給 R1 為 69h(為了+1=58h,重設結果差值數值)
Inintial_3:INC R1               // R1++,換下一位
          MOV @R1,#0            //取 R1 內容當地址,該地址內容設為 0
          CJNE R1,#71h, Inintial_3//若 R1!=71h,跳至 Initial_3 繼續初始化

          POP 1                 //因先前有 PUSH R1,所以要 POP R1
          POP 0E0h              //因先前有 PUSH A,所以要 POP A
          RET                   //返回
```

▲**Table 19** 此為副程式－Reset_all。

```
;-----------------------------------------
;【Subroutine】Displaykeypattern
//將 LED 座向左傳遞的顯示
;-----------------------------------------
Displaykeypattern:
          PUSH 0E0H                 //進入副程式前先 PUSH A
          MOV  dispBuf ,dispBuf+1   //先將左二 LED 數值給左一 LED
          MOV dispBuf+1,dispBuf+2   //再將左三 LED 數值給左二 LED
          MOV dispBuf+2,dispBuf+3   //然後左一 LED 數值給右二 LED
          MOV dispBuf+3,KeyCode     //將取得隻按鍵碼給右一的 LED
          POP 0E0H                  //因先前有 PUSH A,所以要 POP A
          RET                       //返回
```

▲**Table 20** 此為副程式－Displatkeypattern。

```
;-------------------------------------------
;【Fixed data】 for table lookup
//用在事先存取好使 LED 亮 0~9,0.~9.時該給何值
//方便直接取值出去做輸出
;-------------------------------------------
LED_table:DB 0C0h              //當偏移值為 0 時,取 0c0h,字型 0,回去給 A
          DB 0F9h              //當偏移值為 1 時,取 0F9h,字型 1,回去給 A
          DB 0A4h              //當偏移值為 2 時,取 0A4h,字型 2,回去給 A
          DB 0B0h              //當偏移值為 3 時,取 0B0h,字型 3,回去給 A
          DB 99h               //當偏移值為 4 時,取 99h,字型 4,回去給 A
          DB 92h               //當偏移值為 5 時,取 92h,字型 5,回去給 A
          DB 82h               //當偏移值為 6 時,取 82h,字型 6,回去給 A
          DB 0D8h              //當偏移值為 7 時,取 0D8h,字型 7,回去給 A
          DB 80h               //當偏移值為 8 時,取 80h,字型 8,回去給 A
          DB 90h               //當偏移值為 9 時,取 90h,字型 9,回去給 A
          DB 40h               //當偏移值為 10 時,取 40h,字型 0.,回去給 A
          DB 79h               //當偏移值為 11 時,取 79h,字型 1.,回去給 A
          DB 24h               //當偏移值為 12 時,取 24h,字型 2.,回去給 A
          DB 30h               //當偏移值為 13 時,取 30h,字型 3.,回去給 A
          DB 19h               //當偏移值為 14 時,取 19h,字型 4.,回去給 A
          DB 12h               //當偏移值為 14 時,取 12h,字型 5.,回去給 A
          DB 2h                //當偏移值為 15 時,取 2h,字型 6.,回去給 A
          DB 58h               //當偏移值為 16 時,取 58h,字型 7.,回去給 A
          DB 0h                //當偏移值為 17 時,取 0h,字型 8.,回去給 A
          DB 10h               //當偏移值為 18 時,取 10h,字型 9.,回去給 A
          DB 0FFh              //當偏移值為 19 時,取 0FFh,字型無,回去給 A
          END                  //結束程式
```

▲**Table 21** 此為副程式－LED_table。

四、 實作過程、心得、與結論（200 字以上）

減法器

實作過程：

在實作過程中，因為我認為減法器最開始的基礎就是在判斷兩數值的大小，因此我在按鍵碼撰寫完後接著撰寫 Posi_Nega 副程式來判斷負號燈(P1.7)是否會亮起，加上真的很少人是撰寫減法器，就連我的室友也全是加法器，因此我沒有人可以討論，首先我最先想到的就是用小數點前有幾位數去判斷數值大小，而當小數點位數相同時，則從最高位開始比較，雖然想法出來的很快，但在這副程式的撰寫上卻也是我卡最久的地方，一開始其實算是很順利的，我能判斷出整數跟整數之間誰較大，但倘若扯到小數，燈號就會恆亮，我甚至為此 Debug 兩個晚上找到快崩潰，在無意間發現，自己犯了很大的一個錯，那就是沒將數值 2 的小數點前位數紀錄下，導致它恆為 0，負號燈才因此恆亮。

在撰寫這一大個程式中，發生的錯誤當然不只這個，像是去和小數點位置顯示也出過不少錯，而這程式的 Debug 其實並不容易，在撰寫上會慢慢發現，有時多定義一些東西或是將副程式切多一點，其實也會增加 Debug 的效率，而相對地當然 Debug 也需要很多的耐心，藉由 Magawin 的顯示問題去推敲錯誤在哪，這是我從實作最大的感觸。

心得&結論：

這次寫程式會發現，其實很多指令我們自己都沒注意到，像是不能 MOV @R0,@R1 抑或是 CJNE keyCode ,#0,Not0 以及 PUSH R1 等，其實自己錯一輪的印象會比書本告訴我們來的深刻，像我剛開始寫按鍵碼，就是因為不知道 CJNE 後面不能用自己定義的名稱，因此全部被畫紅線，而另外一個我覺得很實用的就是@R0 以及@R1 這個功能，它讓我可以去從別的地址取內容，在連續地址上(像是數字的儲存)其實方便許多，而 PUSH 、POP 其實在我從課本上剛學到時其實覺得它沒甚麼用，但等到自己撰寫程式會發現，有許多值是自己不願動到的，所以會大量用到 PUSH 和 POP，使它在我使用完後還能回復原值。

雖然這個專題讓我很多天睡不好，但是因為是自己從無到有生出，其實會有一種說不出的感動，第一次自己寫到 600 多行，也是第一次在完全沒有人的協助下自己想邏輯、自己 Debug，這次的專題我得到的不單單只是減法器的 code，也還有大大的成就感。

五、 參考文獻

無。